Acknowledgments

Firstly, I would like to thank my advisor, Dr. Rich Maclin, for all his help and guidance that he has given me over the past two years. I would like to express my gratitude to the members of my examination committee, Dr. Doug Dunham and Dr. Harlan Stech. I would also like to thank the CS faculty for providing me support during these two years of my graduate study.

Finally, I would like to thank all my fellow graduate students for making my stay in Duluth that much more pleasurable. The warmth that everyone has shown has made up for all the winter days in Duluth.

Reinforcement Learning in a Multi-agent Environment

Abstract

Reinforcement Learning (RL) is a form of Temporal Difference Learning for situations involving agents exploring domains. This technique provides rewards or punishments to the learning agents. Most of the work in this Machine Learning field has been done on single-agent environments. This work addresses some of the issues involved in applying RL to an environment with competing agents. The environment is a simulated air combat game. A specific type of RL called Q-learning is employed. A collection of agents are built and their behavior studied. Their ability to learn is evaluated by comparing their performances.

Contents

List of Figures

List of Tables

1 Introduction

Machine Learning (ML) is an active field of research. It is playing an increasingly central role in computer science and computer technology. ML is a multi-disciplinary field, drawing results from artificial intelligence, statistics, neurobiology and others. A computer program is said to *learn* from experience with respect to some class of tasks and a performance measure if its performance improves with experience.

One popular ML technique for creating intelligent autonomous agents is to let them explore the environment and provide them with reinforcement signals. While this approach has worked well for some tasks such as learning to play backgammon, it suffers several limitations. One limitation is that this type of learning, Reinforcement Learning (RL), is not applicable to multi-agent environments. This is an important issue since real world applications which involve robots that learn usually have multiple interacting agents.

Agents in a multi-agent system may be competitive or cooperative in nature or a mixture of both. For example, consider a situation where two robots are working together to move blocks in a room. Such a system would be considered cooperative, as both robots are working towards a common goal. On the other hand, consider two agents trying to play backgammon. In this case, each agent is trying to beat his opponent and is considered to be competitive. In more complex environments, the agents may be both competitive as well as cooperative. Consider trying to build two teams to play soccer. Both teams are learning, and each agent is learning to interact constructively with its team member, at the same time competing against the members of the other team.

In addition to the above, the agents might be either learning or pre-programmed agents. A pre-programmed[1] agent has a built-in strategy for choosing actions. It has no capability to

[1] In this thesis, the word programmed and pre-programmed are used interchangeably

learn or generalize on its experience. Typically, an agent which has a specific set of moves for each state is considered pre-programmed. The set of moves do not change and are not dependent on time.

In this work, I propose to address some of the issues that are concerned with applying RL to an environment with competing agents. Since RL is best suited for robot learning, I have performed various tests on game-playing agents. The game itself is based on *Ace of Aces*, a simple air combat game with two opposing players, whose aims are to shoot their respective opponents out of the sky. The game consists of a finite set of states (describing the players position and his opponents' position) with each player having a set of actions (turns, rolls, etc.) to choose from, depending on the state he finds himself in. The game has a set of rules that determine how the actions affect the state.

In this thesis, I have confined myself to a particular type of RL [Sutton,1988] called Q-learning [Watkins,1989]. In RL, the task of the agent is to learn a *policy* (usually a mapping from every state in the set of states of the world to some action in the set of actions available to the agent) that indicates the "best" action to take in each state. In Q-learning, this policy is implemented as an action-choosing module that employs a *utility function* that maps states and actions to a numeric value indicating the value (utility) of taking that action in that state.

The policy of the RL agent is a form of reactive plan. After learning the policy, the agent is able to take the best action using the policy function. This training makes the RL agent more responsive than an agent employing traditional planning methods. The agent is able to adapt to changing environments since the agent makes decisions about future actions and can adjust its actions if the environment changes.

In this work, I have built a collection of agents, pitting them against each other. In my pre-

liminary set of experiments, I train an RL agent against a random opponent, and another against a programmed opponent (one that picks an action from a specific set of moves) This was mainly to determine how well the learning task is handled by the reinforcement learner.

I have built two sets of such agents, each differing in their representation of the utility function. In one, I used a standard Q-table (which is a matrix of numbers, one for each state-action pair) and in the other, I used a connectionist [Anderson,1987, Barto et.al.,1983] representation of the utility function (which is a neural network, with the current state of the world as the input and the output of the network is the utility value for each action.) Regarding the above, I make the following claim:

Claim: *The learning task is tackled by both agents quite well, though their approaches and speed of learning are quite different*

In this thesis, I show that the above claim is justified, by training and testing both agents against specific opponents.

One problem with learning in game theory is that an agent learns against a specific opponent, and its performance against other opponents is not always satisfactory. (A more severe limitation, which is a common problem to AI, is that these agents are highly domain dependent, which means that agents trained for a specific problem cannot solve another task of a different nature). In this thesis, I shall explore this issue of generalization and study how an agent trained against one type of opponent performs against another type of opponent.

Claim: *While both learning agents do have capability to generalize, the connectionist agents are likely to outperform the standard Q-table implementation, for the above-mentioned simple cases.*

I show that the above claim is justified by training both learning agents against a programmed opponent and by testing them against random players. The claim is such that while both agents beat the random opponent, a comparative study shows that the connectionist model does better.

Multi-agent tasks are generally very complex. One limitation of RL is that each agent must explore the environment sufficiently in order to achieve good performance. This amount of exploration grows, usually exponentially, with the number of possible states of the world. In multi-agent tasks, the number of states of the world increases due to the presence of other agents. In this thesis, this limitation is effectively removed because of the finite nature of the game.

Another difficulty associated with multi-agent environments is that each of the agents might be learning. Thus for each agent, the behavior of the environment may be dependent on time. In single-agent domains, the world is assumed to be relatively constant over time. This evolving nature of the task raises a number of problems. In the case of cooperating agents, communication between agents may have to be carried out to achieve the final goal. The nature of such communication may be to update other agents regarding the change in the strategy of one particular agent. However, communication[2] is not an aspect of this work, since there are no cooperating agents.

Another problem raised by the evolving nature of agents, is that of *unlearning* what was already learned. This is because the opponent's behavior may change, invalidating previous experiences. In the second set of experiments that I carried out, I have addressed this key issue. In these experiments, I have one agent learning against another learning agent. As before, I have carried out variations on this idea of *co-evolution* by using different represen-

[2]The game has an advanced version which is based on a team of players competing against another, and communication between team members may have to be addressed in future work

tations of the utility function. Again, I have performed tests to study the generalization capability of such learners.

Claim: *The connectionist model responds to co-evolution much better than the Q-table implementation.*

For the above claim, a set of different strategies for choosing actions are defined and used to measure performance. This is to overcome a limitation that arises due to the finite nature of the game. Since these strategies aim at introducing more randomness, the claim can be justified by comparing the performances of both agents using the above strategies.

A major difficulty associated with RL in multi-agent domains is that of distributing credit for goals that are achieved by agents in a group. There are two simple approaches for this problem: in one, the agent performing the final action is given the credit, and in the other, credit is distributed to everyone in the group. In this thesis, since there are only two agents involved I have chosen to assign credit to both agents, depending on which agent is being shot at. There is an added reinforcement for achieving the final goal (i.e., to shoot the opponent completely out of the sky).

In the following section, I discuss reinforcement learning, artificial neural networks and issues involving planning in more detail. Section 3 presents the rules of the game used in performing the experiments, along with explanations and illustrations. Section 4 starts with the approaches and methodology employed in this thesis, and finishes with a collection of results and analysis of the experiments. In the last two sections, I describe research related to this thesis and the final conclusions and discussions.

2 Background

In this chapter I provide background material for my work. In the first section, I give an introduction to Reinforcement Learning. This machine learning technique, particularly the Q-learning algorithm, has been shown to work well for a class of robot-learning problems. In the second section I give a brief outline on neural networks. Finally, I present some issues on planning that relate to my work.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a form of Temporal Difference learning [Sutton,1988] for situations involving agents exploring domains. In the classical RL setting, the agent or learner perceives the state of the world, performs an action from a set of available actions, changing the state. During this process the agent receives rewards and punishments depending on the action chosen. RL techniques have been used in a number of early AI systems such as

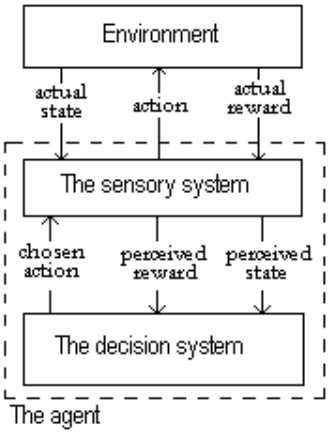

Figure 1: A formal model of a learning environment: the agent receives a description of the environment, selects an action that changes the environment and then receives reinforcements based on the action.

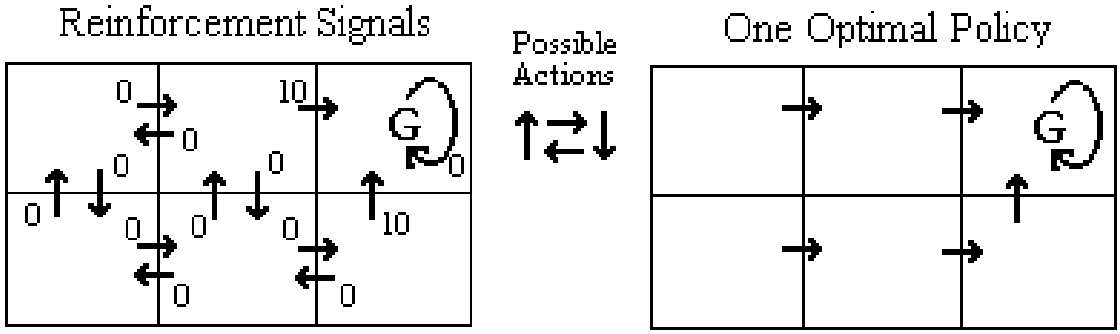

Figure 2: An example of a RL problem: On the left is the figure representing the possible states of the world, the actions the agent may take to move around in the environment and the reinforcement signals. On the right is the figure giving one optimal policy function for the problem. The optimal policy function should achieve the maximum discounted future reward.

Samuel's [1959] checker playing program and Holland's [1986] bucket-brigade algorithm.

Figure 1 gives a formal description of a reactive learning agent with a decision making capability in a world that is represented as a set of states. Figure 2 shows an example of an RL problem in which the world is a grid of squares, and the possible actions for each state are shown along with the reward for each action. The states of the agent correspond to the location of the agent in the grid, and actions are moves (as shown by the arrows) corresponding to moving to adjacent squares in the grid or staying in a particular state. The agent has to learn to reach the square marked **G** from any other square.

The agent's task is to learn a policy that gives the best action available to the agent in any given state. The agent explores the environment stochastically: the agent perceives the world, picks an action stochastically based on its predicted values, and performs the chosen action, changing the state of the world. The agent also receives rewards or punishments for these actions, and tries to learn the "best" action possible for any given state.

In this work I have confined myself to Q-learning [Watkins,1989]. Below, I present a mathematical formulation of Q-learning, which uses the idea of a discounted cumulative reward. The task of the agent is to learn a mapping for each $s \in S$ to some $a \in A$, where S is the set of distinct states in the environment and A is the set of actions available to the agent. Such a mapping, $\pi : S \rightarrow A$, is called a policy. A *discounted cumulative reward* function defined as:

$$V^{\pi}(s_t) = \sum_{i=1}^{\infty} \gamma^i r_{t+1}$$

gives a cumulative reward value for following the policy π from any state $s \in S$. The above function tries to predict the discounted cumulative reward resulting from any action. In the above definition, r_{t+1} is the reward obtained at the $t + 1^{th}$ time step, starting from state s at time t and following the policy π, where $0 \leq \gamma < 1$ is a constant discount factor. The agent's policy is to maximize the above function for all states s_t. Such a policy is referred to as an *optimal policy* and is denoted π^*.

The value of γ is chosen to lie between 0 and 1. This is done mainly because setting $\gamma = 0$ would correspond to a situation where the future rewards are ignored, and setting $\gamma = 1$ would mean that solutions differing in length but leading to the same goal would be considered equivalent. A value of $\gamma = 1$ also makes the learning rule simple. By adjusting the value of γ we can assign higher values to policies that achieve rewards sooner.

In Q-learning, the policy is implemented using a utility function, $Q : S \times A \rightarrow \Re$, that gives a numeric value in the real domain for each state-action pair. Therefore, the agent's policy at time t is given by,

$$\pi_t(s) = a \quad \ni Q_t(s, a) = \max(Q_t(s, x)) \;\; \forall x \in A$$

1. Randomize the Q-values for every state-action pair
2. Observe the current state (s_{curr})
3. Choose an action (a) stochastically
4. Execute the action
5. Observe the new state and the reward (s_{new} and r)
6. Update the Q-value for the state-action pair (s_{curr}, a) using the current state and utility function to arrive at an estimate of the expected utility for taking action a in s_{curr}, and update the Q-value for (s_{curr}, a) using the updating rule.
7. Goto step 2

Table 1: The steps followed in a Q-learning algorithm

When correct, $Q(s, a)$, quantifies all relevant information, not only about the state-action pair, but also all future consequences of choosing the action a.

The steps of the Q-learning algorithm are given in Table 1. To learn the utility function, the agents starts out with a randomized utility function and goes through steps 2-5 in the algorithm. During learning, the actual decision making (step 3) is done stochastically, using a probability distribution function that assigns higher weights for actions with higher Q-values. The update to the utility function is done by comparing a predicted reward value to an estimate of the true reward based on the actual reward obtained. The estimated utility is given by,

$$\hat{Q}(s_t, a_t) = r_t + \gamma \times \max\{Q(s_{t+1}, x), \forall x \in A\}$$

where $Q(s_t, a_t)$ is the current predicted utility value for a particular state s_t and action a_t. This is then used to update the utility function using:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times \{\hat{Q}(s_t, a_t) - Q(s_t, a_t)\}$$

where α is a learning rate parameter. A low value of α (e.g. 0.1) corresponds to slow learning. The above is actually a general class of learning rules, which assumes r_t is a one time-step reward. A more complex rule can also be derived using an *n-step* reward function, which would correspond to the reward being obtained after n time steps after the action was performed. For large n, the agent waits for the future to unfold before updating the Q-values. Watkins [1989] has shown, in the limit, this will converge to the optimal policy.

It is generally difficult to learn the target function accurately and most learning agents only approximate the function that is being learned. In Q-learning, the task is similar to function approximation problems, but differs in a few ways.

1. *Temporal credit assignment* [Sutton,1984] - the learning rule is based on reward values provided over time and does not use (input,output) pairs as most function approximation tasks do. The agent has to determine which of the actions led to the reward.

2. *Exploration* - in RL the agent controls the distribution of training examples by choosing a sequence of actions. This means that the agent faces a trade-off between exploration of unknown states (in an attempt to gather more information) and exploitation of known ones (in an attempt to maximize the cumulative reward with whatever experience it has gained)

3. *Partially observable states* - the agent faces many practical limitations one of which is that the actual state of the world may not be entirely observable. For example, an agent trying to play soccer [Salustowicz,1998] has a "limited vision" and would be able to see only the soccer field in front.

Figure 3 illustrates the idea of Q-learning using the example world described earlier. As before, the squares form a grid and an arrow represents a move from one square to the next. The figure on the left gives the state the agent is currently in (marked by $\mathcal{A}$), and some of the intial estimated Q values. The figure on the right shows the change in the Q value after the agent moved one step towards its right, obtaining a reward value of 0, and observing its present state.

Several approaches to the implementation of the policy function have been studied. In this thesis, I use both a standard Q-table and a connectionist neural network [Anderson,1987, Barto et.al.,1983]. In the connectionist model, the input to the network is the current state of the world, and the output of the network would be the utility value for each action. In connectionist Q-learning, the learner may be able to represent the entire table with a much smaller set of parameters. This would be helpful in situations where representing a complete Q-function would be space intensive. However, a connectionist model is not guaranteed to

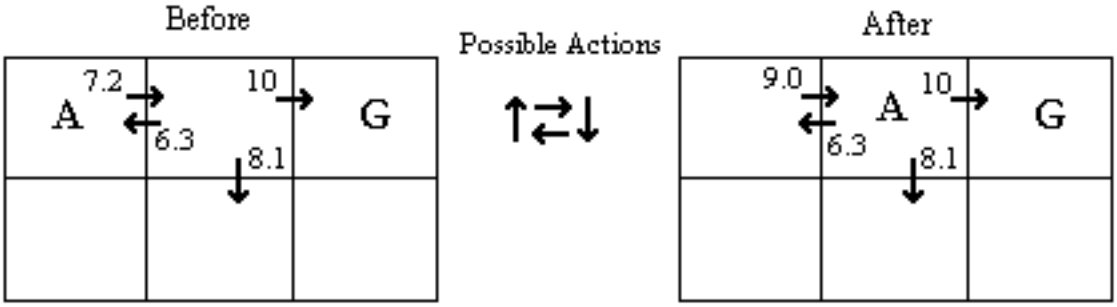

Figure 3: An example of an update to a Q-value of a state-action pair based on the initial state, action taken, reward observed and the resulting state. In the figure on the left, the state of the robot A is shown along-with some initial estimate of the Q-value. After "moving-right" (figure on the right) the agent updates the Q-value for that state-action pair, after receiving zero reward and observing the new state. γ is assumed to be 0.9

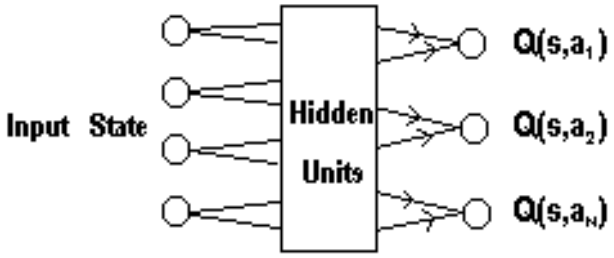

Figure 4: A Neural network implementation of a Q-function: The input to the network is a description of the state of the world and the output is the predicted utility for each of the action

find the optimal policy function, because of the gradient nature of neural networks.

2.2 Artificial Neural Networks

Neural networks have been shown to perform very well in problems involving function approximation, especially when the input data is highly complex with a lot of noise. The backpropagation algorithm has been effective in solving problems including character recognition [LeCun et. al.,1989], recognizing spoken words [Lang et.al.,1990] and in biological systems [Maclin,1993]. Pomerleau [1993] has provided an example of a neural control system, ALVINN, which uses an artificial neural network to steer an autonomous vehicle driving at normal speeds on public highways. In this section, I outline a feed-forward neural network and the backpropagation (BP) algorithm [Rumelhart et. al. 1986].

A neural network is a connectionist model inspired by the human nervous system. The network is made up of *neurons*, which are the fundamental units of the system, and a matrix depicting connections between the neurons. A real value, called the *activation value*, is associated with each neuron.

Let N be the number of neurons in the system, and $\psi_i^t \in \Re$, represent the activation value

of the i^{th} neuron at any instant t. Then,

$$\psi^t = (\psi_i^t, \psi_2^t, \ldots, \psi_N^t)$$

is a vector representing the state of individual neurons of the network at any instant t. Also, let w_{ij} represent the interconnection value or weight of the link from neuron i to j. Then the network is defined as:

$$\mathcal{N}^t = (\psi^t, \mathcal{J}^t)$$

where $\mathcal{J}^t = \{w_{ij} | i, j = 1 \ldots N\}$. A *layered* network is one in which the neurons in one layer are connected to neurons in another layer, but there are no interconnections within the layer. A *feed-forward* network is one whose topology allows no cycles.

Figure 5 depicts a *perceptron* [Rosenblatt,1962], a 2-layered feed-forward neural network with one output neuron. The activation values in the second layer are calculated using an activation function on the weighted sum of the activation values of the neurons in the

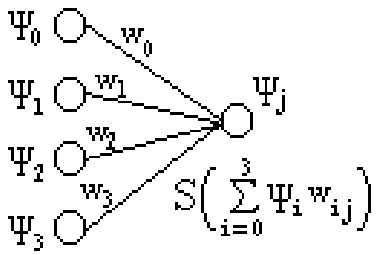

Figure 5: A perceptron: a simple two-layered neural network, with one output neuron. The activation of the output neuron is computed using $\psi_j = S(\Sigma_{i=0}^{k} w_{ij}\psi_i)$, where ψ_i is the activation value of neuron i and w_{ij} is the weight of the link from neuron i to the output neuron. In this diagram, the perceptron has 4 input neurons (k=3)

previous layer. In terms of a single neuron in the second layer we have:

$$\psi_j = S(\sum_{i=0}^{k} w_{ij} \times \psi_i)$$

where $k+1$ is the number of neurons in the first layer that are connected to neuron j. One popular activation function S is the *sigmoid* (or squashing) function. The squashing function given by:

$$S_\sigma(x) = \frac{1}{1+e^{-\sigma x}}$$

with $\sigma = 1$ being used in this thesis, which is also the most common value in the field of research. The above is a family of functions and in $\lim_{\sigma\to\infty} S_\sigma(x)$ converges to the function

$$f(x) = \left.\begin{array}{ll} 0 & x < 0 \\ \frac{1}{2} & x = 0 \\ 1 & x > 0 \end{array}\right\}$$

The *backpropagation* (BP) [Rumelhart et. al. 1986] algorithm has been widely used in neural network research, mainly because it has produced good results experimentally. It employs a gradient descent learning on an energy function defined on a feed-forward layered network system.

Consider the network of Figure 6. Such a network would typically be formulated to solve a problem of function approximation. The second or middle layer in the network is often called the *hidden* layer. It has been claimed that a network with one hidden layer, and sufficient number of units, has the power to solve highly complex non-linear dynamic systems and a network with a 2 hidden layer topology can solve most learning problems in the limit

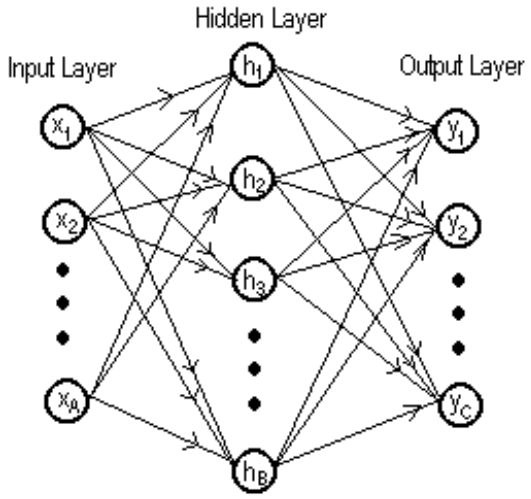

Figure 6: A 3-layered feed-forward neural network

because it can memorize the data, provided the network is formulated accordingly.

The BP algorithm is formalized in Table 2. The algorithm is performed on a set of *training* data (data sets for which the input-output vector pairs are known). After assuming an initial weight matrix, the error between the network output for any input vector, and the output vector (for that input vector in the training data) is measured and used to update the weight matrix. This process is iterated a number of times (chosen empirically).

In the connectionist implementation of RL, the input to the network is the state of the world (as perceived by the agent) and the output of the network is the utility value for each action. During learning, the agent activates the network for the current state and picks an action stochastically based on the utility values obtained in the output layer. If the agent is not learning, it picks the action with the highest utility value.

Neural networks allow the agent to generalize its predictions about the utility of actions to

- Let there be A,B,C number of input, hidden, output neurons in the network, respectively. Also let $i = 1 \ldots A, j = 1 \ldots B, k = 1 \ldots C$. Let w_{ij} represent the weight matrix from input to hidden layer and w_{jk} represent weight matrix from hidden to output layer. Let each input-output vector pair in the training data set (TRAIN) be written as $(\vec{x}, \vec{o})$.

  1. Randomize weight matrices
  2. For each $\vec{x} = (x_1, x_2, \ldots x_A) \in$ TRAIN
     (a) compute output layer units $y(\vec{x}) = (y_1, y_2, \ldots y_C)$
     (b) compute error in output layer (for each unit indexed by k)

$$\delta_k = y_k(1 - y_k)(o_k - y_k)$$

     (c) compute error in hidden layer (for each unit indexed by j)

$$\delta_j = h_j(1 - h_j) \sum_k w_{kj}\delta_k$$

     (d) update each network weight with

$$w_{ij} = w_{ij} + \alpha\delta_j x_{ij}$$

$$w_{jk} = w_{jk} + \alpha\delta_k h_j$$

Table 2: Backpropagation algorithm in a 3-layered neural network

other similar states of the world. This reduces the amount of exploration an agent need perform. However, as stated earlier, neural networks are likely to produce a sub-optimal policy for some states of the world. In the remainder of the chapter, I discuss issues concerning plans and reactive agents.

2.3 Planning in multi-agent environments

Intelligent systems must learn to operate and respond in a dynamically changing environment. Such systems are called *reactive*. Different life forms interact differently in such environments. The most complex of them have an inherent notion of anticipation of future

course of events and plan accordingly. In this section, I briefly introduce the various issues involved in planning.

Real world problems of the type involving human desires and the ability to choose from a set of choices, and the methods of achieving them, are very difficult to formulate. Such problems involve a knowledge of the world, and an ability to foresee the consequence of an action. There are a number of factors involved including the potentially infinite states of the world, the changing environment and the presence of other agents.

2.3.1 States and Actions

The primary issue concerned with the above classes of problems is that of representation of **actions** and **states**. Traditional AI work [McCarthy,1968] has sought to specify the entities involved in such problems: the *state* of the world, (which is potentially infinite) *actions* which the agent takes, and *events* that occur in the world without the direct influence of the agent. One assumption is that only actions and events can change the world state. Some work has also been done on trying to provide the world states with *properties*. These properties help provide values to the objects in the world state (including the agents)

Some other issues involved in such environments would be the non-deterministic property of the actions, which means that when an action (or event) occurs the resulting state may be one among a *set of states*. Another issue is *concurrency* of actions and events. Some of the models of such a system, where two or more actions (or events) occur at the same time are often simplified by assuming that these actions occur one after another without any other action occurring between them.

With the representation of states and actions, a formal method of operating on them is required. The study of *logic programming* provides a calculus for working with action sequences. *Modal temporal* logics [Prior,1967] provide a means to incorporate an element

of *time* into our system. Such a logic helps operate statements that determine *the state of a system n time steps from now*, which was not possible in some of the earlier logical constructs. However, some deficiencies in the calculus formulation gave rise to the STRIPS representation [Fikes,1971] which allows a state to be formed as a conjunction of logical formulas, and an action to be represented as an operator.

2.3.2 Plans

A second issue is plan formulation. Before discussing the different methods available to formulate plans, it is important to realize the notion of a plan. Though there is no formal established definition, a plan is understood to be *something, execution of which produces behavior in a machine.* Viewed in this manner, a program to a computer is a plan. The behavior depends entirely on the machine or the agent, and may be non-deterministic in nature.

Another issue is of concern is whether a plan is a *success* or a *failure.* A successful plan would be one such that every part of it is executed by the agent, and an unsuccessful plan would be one in which there is at least one part that is not executed by the agent. A plan can be considered to be composed of a few basic *non-decomposable plans*, where the methods of composing them would be a combination of *iterating, sequencing, recursing* or *choosing* from them.

Plan creation or formulation involves a more directed approach than just establishing states and actions in an executable format. The idea of a *goal* is inherently a part of a plan. There are different ways in which goals are looked at depending on the system. Classically, a goal was considered to be a set of desired states of a system to which an agent would go. However, real world problems are more complex and goals have been redefined and research has been done on assuming a goal to be maintaining or preventing some condition.

Theorem proving has been the center of logic studies for many years. It provides a deductive approach in formulating plans since our notion of plans revolves around a logical representation for the states and actions. Planning has also been extensively looked at in the form of search. There have been two different approaches to search: try to search the state space for a set of states which lead to the goal state; or try to search a plan space [Nilsson,1980, Tate,1984], which is to look for a plan where each element in the plan space is a partially completed plan. Iterative constructs have also been looked at, i.e. beginning with some plan on which a sequence of modifications are made, until the goal has been achieved.

2.3.3 Multi-agent domains

Since this thesis is based on a multi-agent environment, it is worthwhile to spend some time discussing some of the issues involved in such systems. The above single agent models cannot be used in multi-agent systems because of the incapability of the agents to understand other agents' actions.

When there are multiple agents in the system they may *cooperate* to achieve a common goal, or they might be *competitive* in nature, in which case they are opposing each other. They could also be *non-interfering*, which would mean they have no direct influence with other agents. In all these cases, the action representations have to be extended from the single agent case to be able to handle *concurrent* events (or actions). This becomes a much more complicated process, and can be modeled by approximating the event occurrences or by giving some of the events a property of *atomicity* which involves an idea like *during the occurrence of action a no other event or action occurs.* Such a property endows upon the system the notion of a state transformation very similar to single agent systems and hence the formulations of the classical system can be applied here. Pednault [1990] introduces a

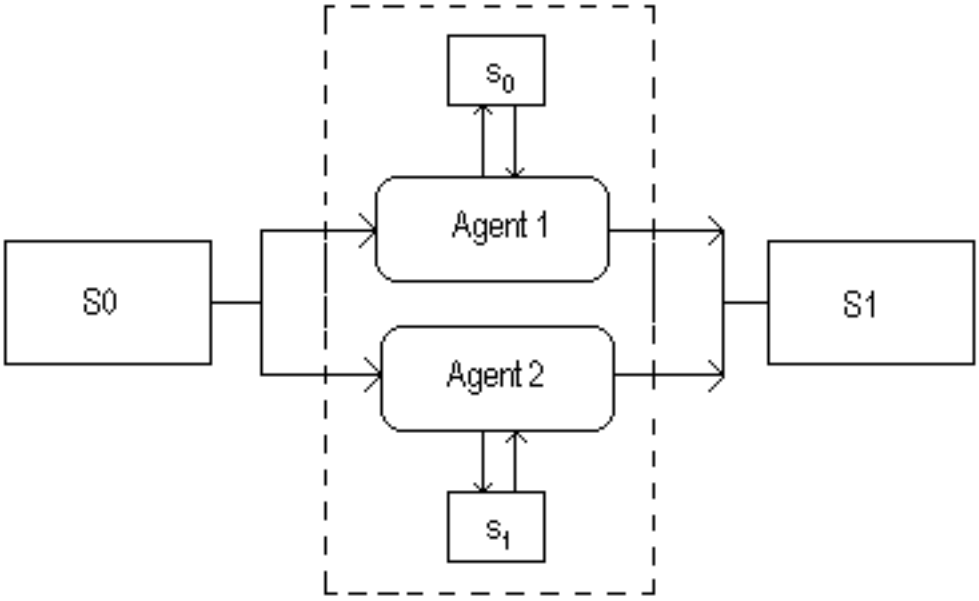

Figure 7: A formal model of the experimental environment

language called ADL, which is syntactically based on the STRIPS language, but has very different semantics, that allows multi-agent dynamic systems to be modeled as a single agent static system.

In this thesis, I have chosen to work with one level of greater complexity than the single agent system. It is a 2-agent environment, with finite state space and in which the agents are adversarial in nature, and each of them have exactly the same finite set of actions (atomic). There are no events (as defined in this thesis) involved, only actions which the agents perform. Also, the state transformation [3] of the system is a concurrent activity, which can be modelled as a *sequential pair of atomic actions*, the first by one agent and the

[3] All state transformations are deterministic

second by the opposing agent.

Each of these atomic actions, leads to a *internal intermediate* state transformation, which would not be the result of the state transformation of the actual concurrent activity. On a system level, the atomic actions are commutative, in the sense that if a occurs just before b occurs, then the resulting state of the system is the same if b occurs immediately before a (where a, b are atomic). This means that the intermediate states, even though affect the outcome of the resulting state of the system do not have a direct bearing on it. Figure 7 gives an incomplete pictorial representation of the system. The complete model is discussed with the experimental setup in later chapters.

3 Experimental Setup

Game theory is an ongoing area of research. Much of this work focuses on pitting man against machine. Some work has been done in applying RL techniques for playing games like back-gammon or simulated soccer. In this section I will describe the game I have used to study learning issues.

Playing a game generally involves *forming a strategy.* It may involve a *cooperative* effort to *compete* against an opponent(s). In such cases, the player usually tries to gain proficiency by playing the game a few times. Forming a general strategy that defeats any opponent would be extremely difficult. Usually, players develop a strategy to beat a particular opponent.

The ability to generalize what has been experienced before is called inductive learning. In my thesis, I will attempt to do *cross-testing.* I will primarily build a set of agents which are programmed with a particular strategy. Later, I will let learning agents play against any of these opponents and observe how well they perform against another of these opponents. I will then test these agents against other learning opponents.

I have chosen to work with a specific game and, as with most learning problems, generalization to other games may not be so easily achieved. Moreover, the game itself has only a finite set of states (unlike other more complex ones.) Nevertheless, this game serves as a useful starting point for doing research with multiple agents. Many issues of planning and learning strategies can be addressed. The game has advanced versions that serve to investigate advanced learning issues as well.

3.1 Description of the game

The game I have used to run my experiments on is called **Ace of Aces** - *WWI Air Combat Game* and is actually one among a series of games played using play-books. Each player

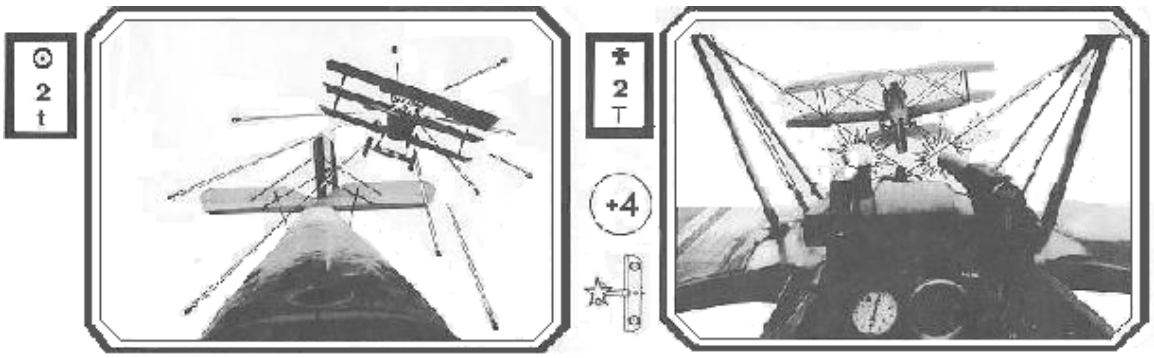

Figure 8: The right picture is a leaf out of the German pilot's book. The Allied aircraft is on the left.

1	The main picture	relative position of opponent
		range of the opponent
		direction the opponent is facing
2	Added information	Nationality
		Page Number
		Tailing information
		Points scored

Table 3: Information provided in each page

has a book, which has pages like the ones shown in Figure 8.

Each book has in all 223 pages, where each page is comprised of the information[4]provided in Table 3. The picture on each page gives the relative position of the opponent's aircraft. There are three different ranges possible for the distance between the planes: *close, medium* and *long*. The pages numbered 1-36 correspond to the close range, those numbered 37-108 are in medium range, and those from 109 - 222 are long range. Page 223 has special significance and is explained later. The picture also shows the direction in which the opponent

[4]There is some more information provided like altitude, critical place of hits, etc. I have not considered for my work.

is facing.

The German pilot has a cross + marked in the rectangular box to the left of the picture, the Allied pilot has a $\odot$ marked in his book. The points scored on each page appears just below the rectangular box (an encircled +4) in the picture on the right of fig. 8. The points scored depend on the distance between the planes. The player being shot at loses 4 points if the two airplanes are in the close range, 2 if they are in the medium range and 1 if they are far apart. Only pages in which the opponent is directly in front yield successful hits. The tailing information, given by the **T** or **t**, is used in the advanced versions of the game.

3.2 Objective

Generally, the objective of the game is to shoot the opponent out of the sky. A game ends either when one player is shot from the sky or when one player decides to withdraw (in the case of the special page 223)

Every game starts with each player having 12 points. During the course of the game, a player loses points if he is shot at. A player is shot out of the sky if his score goes to zero (or below.) In the case of the special page 223, the players have two options before them: to continue or escape. If both players choose to escape, the game is considered to be drawn; if only one of them chooses to escape then that player loses the game; however, if both choose to continue, the game starts from the initial settings with each player retaining his score.

In the advanced version of the game where there are multiple players, a dog fight can be simulated. In such a case, the objective is to earn victory points.

3.3 The Maneuvers

Every player has a set of moves that appear at the bottom of the picture in each page. These moves are based on real world aircraft maneuvers. They are classified into *slow*,

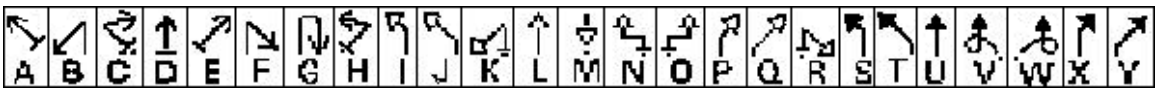

Figure 9: The set of maneuvers that can be performed by any player

medium and *fast*. Each move causes the aircraft to change its position in the sky. Figure 9 gives the list of possible moves that each player can perform.

For example, a move like ↖ means *stall left* (slow turn left). To perform this move, the pilot inclines the aircraft on an angle and the plane loses altitude until the wings become level again. A similar set of moves exist for both the slow and fast set of maneuvers, though they are performed differently.

A player chooses a move from this set of actions, and tells his opponent the number that appears below the action he chose. He does not reveal the actual move. For example, in Table 5, if the player B chooses to perform action marked C in page 2, then he tells his opponent the number 49. These numbers are actually different pages. The actions are fixed but the page numbers they correspond to depend on the current page. For example, A always refers to action ↖, but the number under this action depends on the player (Allied or German) as well as the page he is on.

All pages from 1 to 222 have a similar set of listing of actions and corresponding page numbers beneath them. Page 223 is a special page. This has no such set of actions listed. This page corresponds to players losing sight of each other. See section 3.5.

3.4 Levels of expertise

The game is played with different levels of expertise: *introductory, standard, advanced.* Table 4 gives the sequence of steps followed to the play the introductory game. A move is chosen by referring to the number below the actions marked A through Y (refer Table 5).

1. Start at page 170
2. Choose moves
    - Players make a log of the actions that they make (A - Z)
    - Call out the page numbers marked below the action
3. Locate Mid-turn page
    - This page for player G (B) is the number called out by player B (G)
4. Locate End-turn page
    - This is the page number below the action chosen in the mid-turn page
    - This page has to be same for both players.
    - Turn to this page
5. Record scores
    - make a note of the scores that are indicated next to the picture
    - e.g., if '+4' is marked, opponent loses 4 points
6. Goto step 2

Table 4: Sequence of play

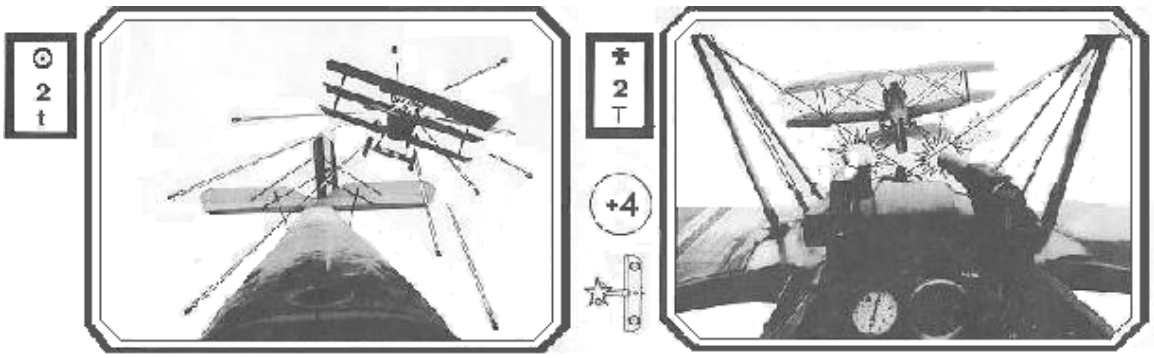

Figure 10: Starting pages for an example sequence of moves

Player B		A	B	C	D	E
	Page 2	36	84	49	2	8
	Page 36	26	19	1	36	2
Player G		A	B	C	D	E
	Page 2	9	29	36	2	31
	Page 36	3	23	26	36	25

Table 5: Table giving the moves for both players

Each player tells his opponent the number below the action that he chose. Both players then turn to the pages their respective opponent tells them to go to. This is the MID-TURN page. In this MID-TURN page both players now look under the action they chose earlier and go to the number below that action. This is the END-TURN page.

All the information in the MID-TURN page is ignored. Only the END-TURN page reflects the true state transition. The rules of the game are created such that, if a player is being tailed (indicated by a "t"), then his opponent who is tailing him (indicated by a "T") has to be given an extra information as to what the tailed player's next move would be. However, I have chosen to ignore this rule throughout my experiments.

As an example, consider Figure 10. Let B represent the Allied plane and G be the German

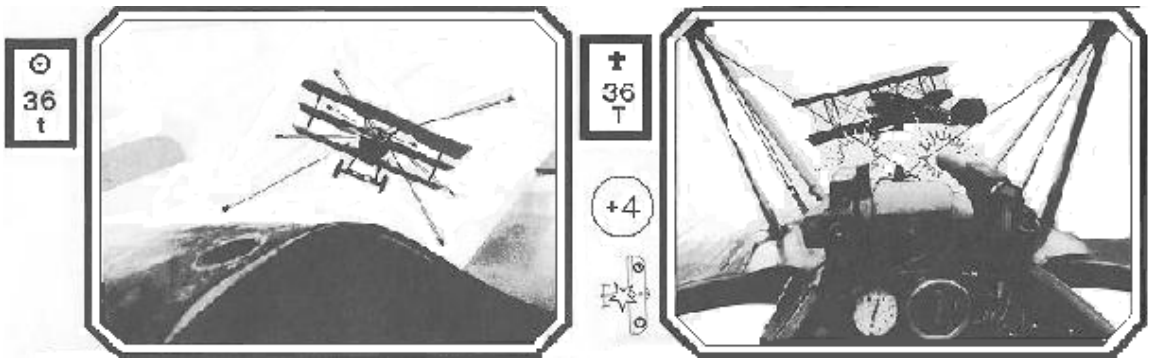

Figure 11: Ending pages for an example sequence of moves

plane. Let us assume both are on page 2 as given in the figure. Consider Table 5. Let us suppose B makes the move marked D, and G makes the move marked C. Then, B turns to page 36, and G remains in page 2. Page 2 is the MID-TURN page for the German pilot, while Page 36 is the MID-TURN page for the Allied aircraft.

In the MID-TURN page (Figure 11) for the Allied player we have, under action D (as he had chosen action D) the page number 36. Which means the END-TURN page is 36. This page can also be arrived at by referring to action C in the German's MID-TURN page 2 (Figure 10). At the end of this sequence of moves, the players always end up on the same page.

Figure 12 gives a state description of the players' moves from one page to another using the number given by the opponent. The numbers marked in bold correspond to the page numbers under that action for that player. For example, the bold faced 36 (Table 5) refers to number under action C in the German's book.

In the *standard* version of the game, the players take into account the different machines they are flying in. Each machine has some characteristics (which are detailed in the rules of the game) and may not be necessarily evenly matched. I have chosen to ignore this version

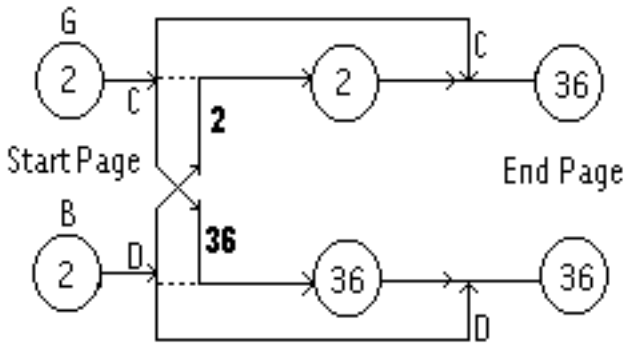

Figure 12: States representing sequence of moves

of the game.

In the *advanced* game, the starting page is not fixed at 170. Moreover, there is also the idea of different altitude of the players involved. I have however, chosen to ignore difference in heights in my experiments. The starting page is determined by casting a six-sided die twice. Table 6 gives the starting pages depending on the die rolls. The first roll refers to column, and the second roll refers to the row. For example, if there was a *4 rolled after 2* then the starting page would be 167 (marked in bold face in the table.)

There is one other version of the game, *the campaign*, which involves multiple players in a dog fight, which would be a good starting point for working on a team strategy.

3.5 Page 223

As mentioned before, Page 223 is a special situation. This page does not have the usual set of actions listed under it. It corresponds to the players losing sight of each other. If this page occurs as a MID-TURN page, then the player who ended up here should wait until his opponent figures out the end page. He should then go to the end page and resume the game

	1	2	3	4	5	6
1	112	133	114	175	176	177
2	169	170	171	148	149	150
3	181	182	183	151	152	153
4	166	**167**	168	173	174	172
5	178	179	180	184	185	186
6	169	170	171	172	173	174

Table 6: Starting page: The columns marked 1 through 6 are the numbers corresponding to the roll of the first die. The rows marked 1 through 6 correspond to the rollof the secind die.

However, if it occurs as the END-PAGE (which means both players have to land up here) then it means both players have lost sight of each other. The players have two options before them: they can choose to *search their opponent and resume the fight* or they can choose to *escape*. The players must write out what their choice of action, before they disclose it, after which they are not allowed to change it.

If both players chose to escape, the game ends in a draw in the introductory game irrespective of the point difference. If any one player chose to search, and the other escape, then the player who chose to search wins the game. He is considered to have *driven his opponent out of the sky*.

3.6 Implementation

All experiments are done as simulations. The game is played using a program written in C. The main program simply follows the steps of the game as outlined in Table 4.

The program uses a configuration file, which helps control the parameters involved while learning and testing. Different player modes are specified in this file. Each player mode has a file which does the decision making. In case of the learning agents, the rewards are also

programmed into the files. There is also a manual player mode that allows users to play the game. In this case, the different pages appear on the screen.

3.7 State Representation

Every state is referenced using the page number that corresponds to it. The learning agent using the Q-table has a matrix (222 possible states $\times$ 25 possible actions) of Q-values. For example, if action 5 in page 160 has the maximum value for that state, then Q[160][5] has the maximum value of all the Q[160][.] values.

The learning agent using the neural network, translates the number 160 into an input vector. The input vector consists of 8 units based on the analysis of the games's pages:

- 4 units for distance between planes: 1 for each of Close/Medium/Long/Strange
- 2 units for position of opponent: 1 each for the Sine and the Cosine of the angle to the opponent
- 2 units for position of agent (as viewed by opponent): 1 each for Sine and Cosine of the direction the opponent is facing.

The neural network has 25 hidden units, and 25 output units (1 for each action), and is fully connected from one layer to the next.

3.8 Deadlock Avoidance

Some sequences of moves lead to a cycle in the state transitions. In such a situation, the game goes on forever. This happens due to determinism in the action-choosing module for an agent. For example, let S be the current state. Suppose, both players can make only one action in this state. Let the resulting state be T. If both players have only one action in this state and by making them, they go back to state S, there is a cycle in the state transitions.

(This case is the simplest case.) Further, if both S and T do not involve hits by any player then the scores for each player would remain as they are, but the game would never end. (There is also the possibility of a cycle of length greater than two.)

Deadlocks can also occur when there is non-determinism, but the situation is similar to the above. The only difference in this case is each player has a set of actions instead of only one. The agent chooses among this set randomly, but whatever action it takes, leads to a cycle.

This issue of deadlock is not specific to this game. Similar situations are known to exist in other games and environments as well. For example, in the game of chess, if each player loses all the pieces except the "king", then neither player can achieve a decisive victory over the other. The problem can be resolved by declaring the game to be drawn. In this game, the main control program keeps track of the number of moves, and usually declares any game that goes beyond 150 moves as a draw.

3.9 Description of the pre-programmed agent

The strategy of the random agent is to pick any of the 25 possible actions in any state. The programmed opponent determines a set of reasonable actions for each state, and randomly picks from this set. The set is computed in the following manner

- Determine the current state.
- Assuming the opponent decides to continue flying in the current direction, determine which action would put the opponent in front

The above decision making is simple but has an "inherent" knowledge of which action leads to a particular state configuration. This does not mean that the agent is extremely powerful. A limitation to this agent is the assumption that opponent continues to fly straight (which

will not always be the case).

3.10 Choice of Action

The nature of the agent determines the actual choice of action. As said before, the random agent simply chooses any of the 25 actions. The programmed opponent chooses randomly among a set of reasonable actions. The learning agent picks actions stochastically (during the training phase.) according to [Lin,1992]:

$$Probability(a_i) = e^{\frac{merit_i}{Temp}} / \sum_{k \in Actions} e^{\frac{merit_k}{Temp}}$$

where $merit_i$ corresponds to Q-value of action i, $Temp$ is a temperature parameter, usually kept constant (in this thesis it is maintained at 2), and $Actions$ refers to the set of actions in any given state. In the above formula, the action with higher Q-value is assigned a greater weight, but actions with lower Q-values are not completely ignored. This attempts to achieve a balance between exploration and exploitation.

During testing, the learning agents choose actions deterministically. They pick the action that corresponds to the maximum value from Q-table or the output layer of the neural network.

4 Experiments

In this section I discuss experiments performed on the testbed. The preliminary set of tests consist of simple experiments. In these, the learning agent is trained against an opponent who is either random or programmed (with some randomness). These tests were essentially run to study the learner's behavior with respect to specific opponents.

In the preliminary tests, I attempt to answer the following questions relating to the agent behavior:

- Determine suitable learning rates (α)
  - How does the parameter α affect the learning? Is learning faster/better if α increases?
  - What are some suitable learning rates for both methods? (Are they the same?)
- Establish baselines for comparisons of agents
  - Does the learning rate affect the generalization capability?
  - Which model generalizes better (able to beat opponents not trained against) ?

Generally, each agent has a range of learning rates within which it works well. Determining suitable learning rates in this manner help in fixing the parameter for the more advanced experiments. In the more complex set of experiments, I try to investigate *co-evolution*, situations where both agents are learning. Some of the questions that occur are:

- Choices of Actions
  - Which method of choosing actions works better ?
  - Is it the same for both agents?

- Comparison between agents
  - Which agent works better?

4.1 Methodology

As mentioned before, the experiments are done with simulations. The results shown for each experiment are averaged over ten different initial conditions. For example, in the case of the Learning Agent (Q-table), a seed is used to generate the random initial values of the Q-table. Ten such seed values are arbitrarily chosen and the same experiment is run for each of the cases.

Data collected is the number of wins recorded by each player and the number of games that end in a draw. In obtaining the data presented in this section, I did the experiments

1. Training
   - Start with initial Q-values (say between -0.1 and +0.1)
     - for the connectionist model initialize the weights of the network
   - Train for some number of games (say N)
   - Record the set of Q-values (which have been updated using the learning rule)
     - for the connectionist model record the weights of the network
   - Repeat the above two steps until a sufficient number of Q-value sets are obtained
2. Testing
   - For each set of Q-values or weights recorded above
     (a) Play the game (without learning) for some number of games
     (b) Record results for each game

Table 7: The general methodology for the experiments performed. The experiments are carried out in 2 phases, *training* and *testing*. The steps for both cases are outlined above.

iteratively. First, I start with an initial set of parameters: *learning rate, number of games, level of game, etc.* I then train the agents and test them. The results of this testing are shown in a tabular and graphical formats.

Each experiment is carried out in two phases: the *training* phase and the *testing* phase. During the training phase, the learning agent chooses actions stochastically, while the opponent chooses actions in a deterministic manner. The Q-values (in the case of the Q-table representation) are recorded at regular intervals. In the testing phase, the learning agent, using the Q-values recorded at regular intervals during training, picks action deterministically (usually the maximum value from the Q-table). There is no updating of the Q-table during testing. The opponent would normally be the agent against whom the learner was trained, except during *cross testing.* Table 7 gives the steps for each phase.

The discount factor used in all these games is kept constant at 0.9. The results provided are usually for the advanced game level. All the data shown in the section are the results of the testing phase.

4.2 Preliminary Results

4.2.1 Learning Agent (Q-table) vs Random Agent

Methodology : The first set of experiments involved the Q agent learning against a random player using a full Q-table. The sequence of steps outlined in Table 7 was followed. Q-tables were stored every 3000 games for $\alpha = 0.01$ and every 1000 games for $\alpha = 0.05, 0.1$

Results : In Table 8, the learner is tested against a random opponent. The data shown are for learning rates 0.01, 0.05 and 0.1. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw.

α =0.01				α =0.05				α =0.1			
Num train games	Lrn Wins	Rnd Wins	Drw	Num train games	Lrn Wins	Rnd Wins	Drw	Num train games	Lrn Wins	Rnd WIns	Drw
3000	398	299	301	1000	305	347	347	1000	228	383	387
9000	319	334	346	3000	276	359	363	3000	407	296	296
15000	302	348	349	5000	372	314	313	5000	567	221	211
21000	378	306	314	7000	494	257	248	7000	685	159	155
27000	440	280	279	9000	595	203	200	9000	751	125	123
33000	530	237	232	11000	679	162	157	11000	799	102	97
39000	585	205	208	13000	748	123	128	13000	839	80	80
45000	657	168	174	15000	768	118	113	15000	844	78	77
51000	686	154	159	17000	834	85	80	17000	838	80	80
57000	717	142	139	19000	831	85	83	19000	853	71	75

Table 8: Learning Agent (Q-table) vs Random Agent: In the above table, the first column shows the number of games after which the Q-values were recorded during training, the second and third columns show the number of games won by the Learner and the Random agent respectively, and the last column shows the number of games that yielded a no-result (tested over 1000 games)

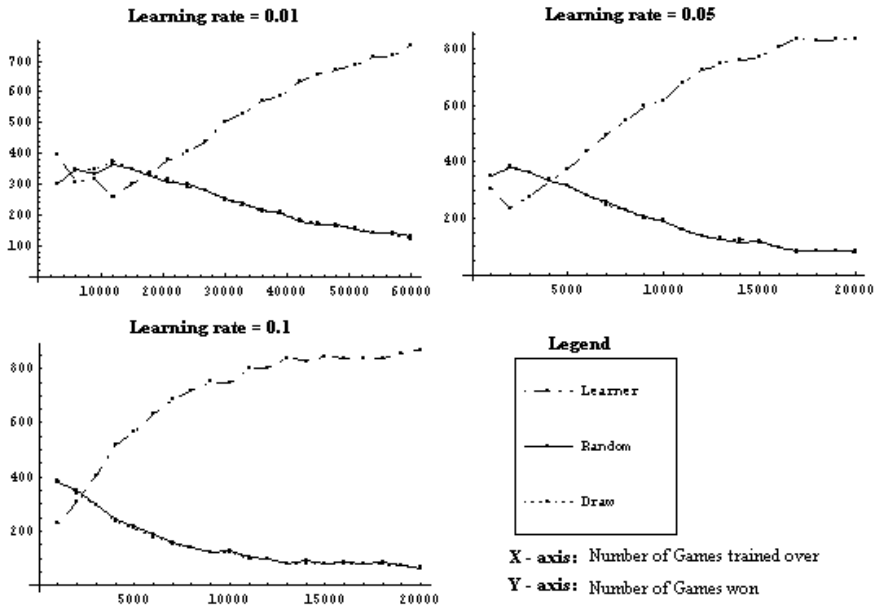

Figure 13: Graph Plots (Table 8) for Learning Agent (Q-table) vs Random Agent: The plot on the upper left corner corresponds to $\alpha = 0.01$, the one on the right $\alpha = 0.05$ and the one below corresponds to $\alpha = 0.1$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by a player (includes draws).

Figure 13 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points at which the Q-values were recorded.

Analysis: Clearly, from the graph we can see that the learning agent is able to beat the random opponent nearly 85% of the time. Also, from the 3 graphs plotted in Figure 13, we see that the learning rate indeed affects the number of games the agent has to be trained over, to achieve a win percentage of ≈ 85. The learning is faster in the case of $\alpha = 0.1$, reduces when $\alpha = 0.05$ and is slowest among the three when $\alpha = 0.01$.

α =0.01				α =0.05				α =0.1			
Num train games	Lrn Wins	Prg Wins	Drw	Num train games	Lrn Wins	Prg Wins	Drw	Num train games	Lrn Wins	Prg Wins	Drw
1000	262	246	491	1000	645	29	325	1000	923	3	73
3000	462	21	515	2000	927	3	69	2000	983	0	16
5000	654	4	340	3000	965	0	34	3000	988	0	11
7000	819	1	179	4000	986	0	13	4000	990	0	9
9000	907	1	91	5000	983	0	16	5000	988	0	11
11000	940	0	59	6000	984	0	15	6000	992	0	7
13000	968	0	31	7000	989	0	10	7000	995	0	4
15000	976	0	23	8000	991	0	8	8000	993	0	6
17000	981	0	18	9000	984	0	15	9000	993	0	7
19000	984	0	15	10000	989	0	10	10000	995	0	4

Table 9: Learning Agent (Q-table) vs Programmed Agent: In the above table, the first column shows the number of games after which the Q-values were recorded during training, the second and third columns show the number of games won by the Learner and its opponent respectively, and the last column shows the number of games that yielded a no-result (tested over 1000 games)

4.2.2 Learning Agent (Q-table) vs Programmed Agent

Methodology: The second set of experiments involved the Q agent learning against a programmed player. The sequence of steps outlined in Table 7 was followed. Q-tables were stored every 1000 games.

Results: In Table 9, the learner is tested against a programmed opponent (who has a specific move for every state.) The table shows data for learning rates 0.01, 0.05 and 0.1. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw.

Figure 14 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points

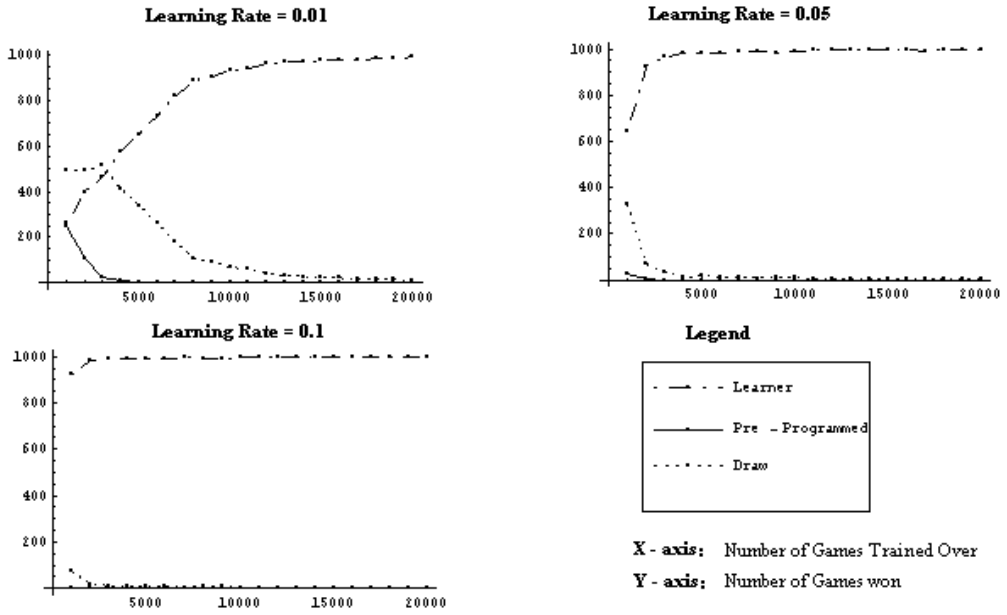

Figure 14: Graph Plots (Table 9) for Learning Agent (Q-table) vs Programmed Agent: The plot on the upper left corner corresponds to $\alpha = 0.01$, the one on the right $\alpha = 0.05$ and the one below corresponds to $\alpha = 0.1$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by a player (includes draws).

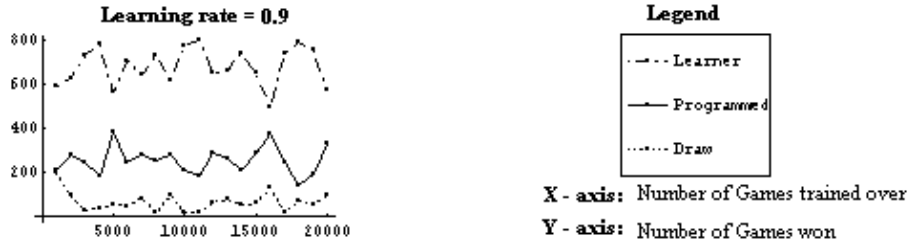

Figure 15: Graph Plot (Table 10) for Learning Agent (Q-table) vs Programmed Agent: The plot corresponds to learning rate 0.9. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by a player (includes draws).

Num of train games	Lrn Wins	Prg	Draw	Num of train games	Lrn Wins	Prg	Draw
1000	591	210	197	11000	799	181	19
2000	627	278	94	12000	648	284	66
3000	725	247	27	13000	661	261	77
4000	778	181	39	14000	741	207	51
5000	564	383	52	15000	648	290	60
6000	705	246	48	16000	494	375	130
7000	639	276	84	17000	740	241	17
8000	727	253	19	18000	788	139	72
9000	619	283	97	19000	752	190	57
10000	770	208	20	20000	575	328	96

Table 10: Learning Agent (Q-table) vs Programmed Agent: Learning Rate set at 0.9. In the above table, the first column shows the number of games after which the Q-values were recorded during training, the second and third columns show the number of games won by the learner and its opponent respectively, and the last column shows the number of games that yielded a no-result (tested over 1000 games)

at which the Q-values were recorded.

Analysis: Clearly, from the graph we can see that the learning agent is able to beat the programmed opponent more than 97% of the time. As before, we see that the learning rate affects the number of games the agent has to be trained over, to achieve a win percentage of $> 90\%$.

From Figures 13 and 14, it is clear that the agent being trained versus the programmed opponent learns faster and achieves a higher percentage of wins. The programmed opponent makes moves with little randomness. This means that during training, the learning agent has greater scope of obtaining rewards or punishments. In the case of the random opponent, the learning agent while exploring the states, does not make too many rewards (he gets zero

α =0.001				α =0.01			
Num of train games	Lrn Wins	Rnd Wins	Draw	Num of train games	Lrn Wins	Rnd Wins	Draw
10	993	5	1	10	992	6	1
30	994	4	1	30	991	6	1
50	992	6	1	50	992	5	1
70	992	6	1	70	994	4	1
90	992	6	1	90	993	5	1
110	992	5	1	110	995	3	1
130	991	7	1	130	992	6	1
150	993	5	1	150	994	4	1
170	994	4	1	170	992	6	1
190	994	4	1	190	992	6	1

Table 11: Learning Agent (NN) vs Random Agent: In the above table, the first column shows the number of games after which the Q-values were recorded during training, the second and third columns show the number of games won by the Learner and the Random agent respectively, and the last column shows the number of games that yielded a no-result (tested over 1000 games)

reward in most states) and it takes a while before he gains sufficient meaningful experience.

From Figure 15 and Table 10, we see that learning is not necessarily faster as the learning rate parameter increases. There seems to be a zig-zag in the graph, unlike the earlier cases. When the learning rate is high, each reward influences the updating of the Q-table significantly. This means that if an agent receives a high immediate reward for an action that ultimately did not help achieve the goal, it may continue taking the action (the exploration of the world is reduced) without actually learning the best action.

4.2.3 Learning Agent (NN) vs Random Agent

Methodology : This set of experiments involved the connectionist Q-agent learning against a random player. The sequence of steps outlined in Table 7 was followed. The weights of

the neural network were recorded every 10 games.

Results : In Table 11, the learner is tested against a random opponent. The data shown are for learning rates 0.001, and 0.01. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw.

From Table 11 we see that the learning agent has no trouble beating the random oponnent. It can also be seen from the data that "learning" is not visible, by which I mean that the learning process does not follow a smooth curve. Even a small number of games is sufficient to beat the random player significantly. This indicates the power of the neural network.

4.2.4 Learning Agent (NN) vs Programmed Agent

Methodology : This set of experiments involved the connectionist Q-agent learning against a programmed player. The sequence of steps outlined in Table 7 was followed. The weights of the neural network were recorded every 100 games.

Results : In Table 12, the learner is tested against a programmed opponent. The data shown are for learning rates 0.001, 0.01, and 0.1. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw.

Figure 16 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points at which the Q-values were recorded.

Analysis : Clearly, the programmed player loses a significant number of games. However, as in the case of the random opponent, the learning process does not follow a smooth curve. Instead, there seem to be small fluctuations. A learning rate of 0.1 seems to bring

α =0.1				α =0.01				α =0.001			
Num train games	Lrn Wins NN	Prg Wins	Drw	Num train games	Lrn Wins NN	Prg Wins	Drw	Num train games	Lrn Wins NN	Prg Wins	Drw
1000	879	96	24	1000	977	21	1	1000	843	154	1
3000	859	53	87	3000	962	36	1	3000	882	64	53
5000	854	139	5	5000	976	19	4	5000	860	135	4
7000	736	43	220	7000	958	8	32	7000	815	160	24
9000	847	18	134	9000	964	33	2	9000	879	118	1
11000	949	27	22	11000	897	68	34	11000	921	69	8
13000	761	29	208	13000	987	10	2	13000	889	106	3
15000	707	33	259	15000	980	17	2	15000	944	52	2
17000	727	29	242	17000	973	23	2	17000	899	99	1
19000	648	61	290	19000	968	24	7	19000	932	62	4

Table 12: Learning Agent (NN) vs Programmed Agent: Learning Rates 0.1,0.01,0.001. In the above table, the first column shows the number of games after which the Q-values were recorded during training, the second and third columns show the number of games won by the learner and its opponent respectively, and the last column shows the number of games that yielded a no-result (tested over 1000 games)

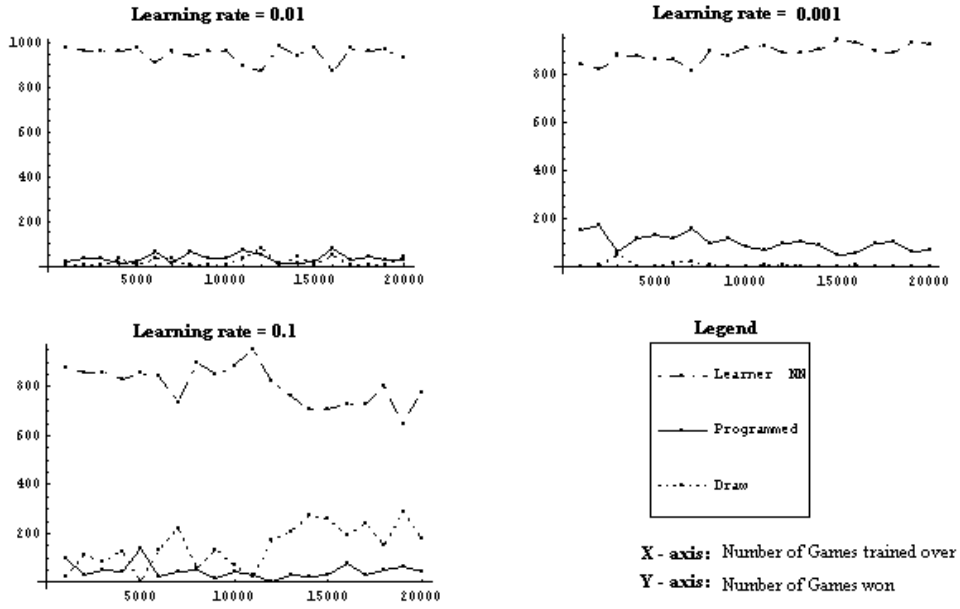

Figure 16: Graph Plots (Table 12) for Learning Agent (NN) vs Programmed Agent: The plot on the upper left corner corresponds to $\alpha = 0.01$, the one on the right $\alpha = 0.001$ and the one below corresponds to $\alpha = 0.1$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by a player (includes draws).

down the agent's performance, suggesting that a slower learning rate should be used for the connectionist model.

4.3 Preliminary Cross Testing

4.3.1 Learning Agent (Q-table) vs Random Opponent

Methodology : In this set of experiments, no training was done. Instead, the learning agents (using full Q-table and NN) trained against the programmed opponent were used. The agents were tested against random opponents and the results were recorded.

Results : In Table 13, the learned agent is tested against a random opponent. The data

shown are for learning rates 0.01, 0.05, and 0.1. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw.

Analysis : From Figure 17 and Table 13, we can see that no observable pattern exists. There is lot of variation as learning rate changes. The learned agent does not beat the random opponent significantly. In case of $\alpha = 0.1$ there is a noticeable decrease in win percentage. It might be worthwhile to observe the graphs in Figure 13 in comparison to the ones in Figure 17. This clearly shows that the generalization capability of the agent is limited.

α =0.01				α =0.05				α =0.1			
Num train games	Lrn Wins	Prg	Drw	Num train games	Lrn Wins	Prg	Drw	Num train games	Lrn Wins	Prg	Drw
1000	299	350	349	1000	305	347	347	1000	404	296	298
3000	300	348	351	3000	305	351	342	3000	208	400	390
5000	301	348	350	5000	307	348	343	5000	209	390	399
7000	402	296	301	7000	210	393	396	7000	212	389	397
9000	404	300	295	9000	212	393	393	9000	211	386	401
11000	404	296	299	11000	213	390	395	11000	213	389	397
13000	306	342	351	13000	214	390	394	13000	212	390	396
15000	306	338	354	15000	214	387	397	15000	214	392	393
17000	308	342	349	17000	214	384	401	17000	212	398	389
19000	309	341	349	19000	214	388	396	19000	213	391	395

Table 13: Learning Agent (Q-table) trained against Programmed Agent, tested against Random player: Learning rates 0.01, 0.05, 0.1. The agents trained in previous section were used. In the above table, the first column shows the number of games after which the Q-values were recorded during training, the second and third columns show the number of games won by the learner and its opponent respectively, and the last column shows the number of games that yielded a no-result (tested over 1000 games)

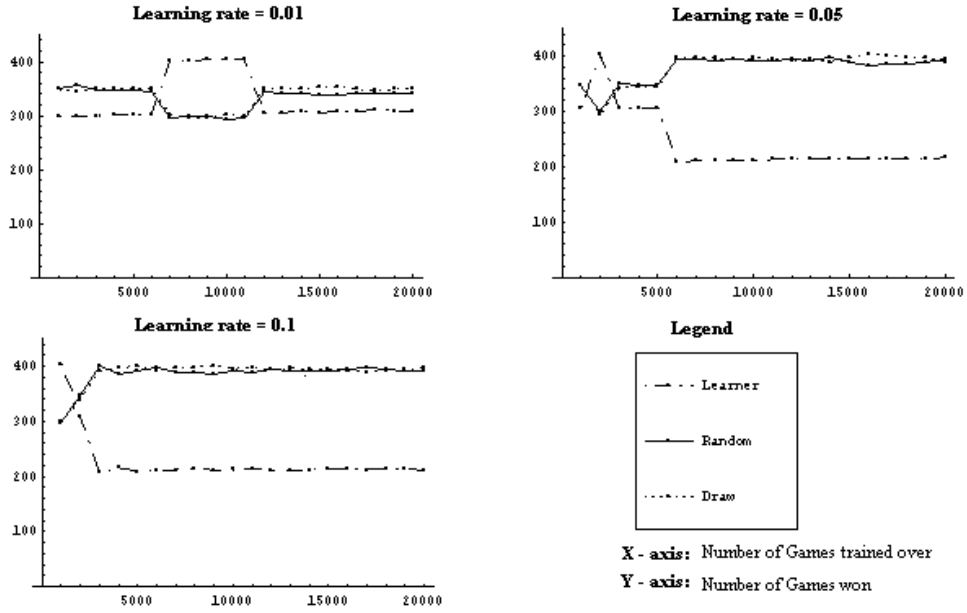

Figure 17: Graph Plots (Table 13) for Learning Agent (Q-table) trained against a programmed agent, tested against a Random Agent: The plot on the upper left corner corresponds to $\alpha = 0.01$, the one on the right $\alpha = 0.05$ and the one below corresponds to $\alpha = 0.1$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by a player (includes draws).

4.3.2 Learning Agent (NN) vs Random Opponent

Methodology : In this set of experiments, no training was done. Instead, the trained set of agents from section 4.2.4 were used. The agents were tested against random opponents and the results were recorded.

Results : In Table 14, the learned agent is tested against a random opponent. The data shown are for learning rates 0.001, and 0.1. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that

α =0.001				α =0.01			
Num of train games	Lrn Wins	Rnd Wins	Draw	Num of train games	Lrn Wins	Rnd Wins	Draw
1000	996	2	1	1000	994	4	1
3000	995	3	1	3000	993	4	1
5000	995	3	1	5000	993	5	1
7000	995	3	1	7000	990	8	1
9000	994	4	1	9000	991	6	1
11000	994	3	1	11000	991	7	1
13000	995	3	1	13000	994	4	1
15000	995	3	1	15000	992	5	1
17000	994	4	1	17000	993	4	1
19000	993	4	1	19000	993	4	1

Table 14: Learning Agent (NN) trained against Programmed opponent, tested against Random Agent: Learning rates 0.001, 0.01. Agents trained in previous sections were used. In the above table, the first column shows the number of games after which the Q-values were recorded during training, the second and third columns show the number of games won by the Learner and the Random agent respectively, and the last column shows the number of games that yielded a no-result (tested over 1000 games)

ended in a draw.

Analysis : From Table 14 it is clear that the random opponent loses to the connectionist agent. Even though the connectionist agent was trained against the programmed opponent, it is still able to beat the random opponent. From Figure 17 it is clear that the neural network model performs better than the Q-table model, in this case.

4.3.3 Summary

From the above collection of data, we can arrive at the following relating to agent behavior relating to learning rates:

- Increase in learning rate for small values of α, does increase the speed of learning. For higher values of α, learning ceases to be smooth.
- Learning is faster if opponent is less random, in the case of the Q-table.
- The neural network achieves a lower winning percent against the programmed opponent than does the Q-table.

We can also arrive at the following relating to generalization:

- The NN agent is able to beat the random opponent irrespective of who it was trained against.
- The learning rate affects the generalization capability of the Q-table agent, but does not affect the neural network agent.

4.4 Advanced Results

4.4.1 Methodology

For the following experiments the methodology as outlined in Table 15 is followed. Each agent learns for some number of games (in this thesis I refer to this number as the "swap"

- Training
  - Start with initial Q-values (say between -0.1 and +0.1) for both agents (A and B)
  - Train agent A for some number of games (say 100), agent B plays deterministically
  - Train agent B for the same number of games as above, agent A plays deterministically
  - Record the set of Q-values (which have been updated using the learning rule)
  - Repeat the above three steps until a sufficient number of Q-value sets are obtained

Table 15: The general methodology for the experiments performed. The steps for the training phase is outlined above. Testing is done using steps from Table 7

number) while the other agent picks actions deterministically. At any instant, only one agent is learning, but both agents eventually learn against each other. In this thesis, the "swap" is fixed at 100.

While this enables learning, it poses problems during the testing phase. In the testing phase, the learning agents pick only the action with the highest Q-value at any state. This means that, if both agents use this strategy, then the result of the games would depend solely on the starting page (and since this is fixed, would result in a finite set of results). To measure the ability of the agent, I chose to introduce an element of randomness in the testing stage. The following is the list of strategies used:

1. Agent picks (randomly) any action with a Q-value above a certain percentage of the highest Q-value for that state. (In this thesis, an arbitrary value of 97% was chosen).

2. Agent picks actions in a stochastic manner (similar to the learning process without updating the Q-table).

3. Agent picks (randomly) any action from all actions with the top N Q-values. (In this thesis an arbitrary N value of 3 was chosen).

4.4.2 Learning Agent (Q-table) vs Learning Agent (Q-table)

EXPERIMENT : Stochastic choice of action at test

Methodology : The first set of experiments involved the Q agent learning against another Q agent (both using full Q-tables). The sequence of steps outlined in Table 15 was followed. Q-values were recorded every 5000 games.

Results : In Table 16, the learner is tested against a trained opponent. The data shown are for learning rates 0.025 and 0.05. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins

α =0.05				α =0.025			
Num of train games	Lrn	Lrn (exp)	Draw	Num of train games	Lrn	Lrn (exp)	Draw
5000	408	315	276	5000	215	380	404
15000	314	340	344	15000	241	370	388
25000	366	321	312	25000	246	366	387
35000	373	317	308	35000	256	362	380
45000	428	298	273	45000	263	361	374
55000	157	404	437	55000	253	365	380
65000	176	398	425	65000	278	347	374
75000	196	406	397	75000	288	341	370
85000	226	373	400	85000	274	353	372
95000	280	352	366	95000	285	347	367

Table 16: Learning Agent (Q-table) vs Learning Agent (Q-table): The above table shows data for two learning agents. Each agent learns every alternate 100 games. During testing the agent in the second column plays deterministically (picks the maximum from Q-table) while the agent in third column chooses with an exponential distribution

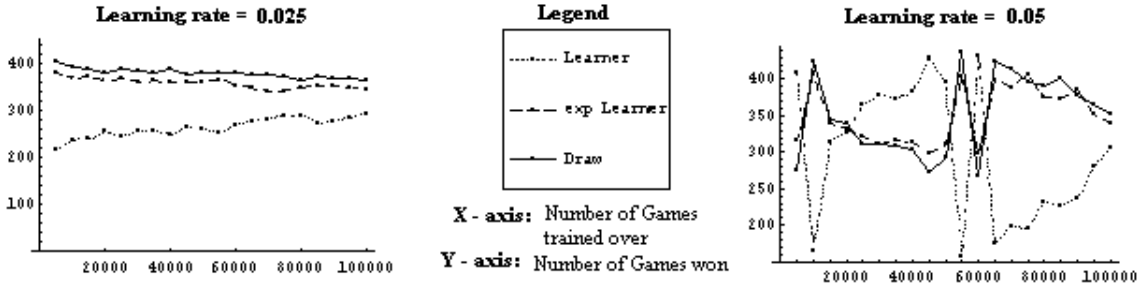

Figure 18: Graph Plots (Table 16) for Learning Agent (Q-table) trained against a Learning Agent (Q-table): The plot on the left corresponds to $\alpha = 0.025$, the one on the right corresponds to $\alpha = 0.05$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by each player. The agent marked "exp" picks actions stochastically during testing.

recorded by each player. The last column corresponds to the number of games that ended in a draw. The agent marked "exp" picks one action stochastically using the probabilities for each action (actions with higher Q-values are assigned higher weights.) The other agent picks the action with the maximum Q-value in each state.

Figure 18 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points at which the Q-values were recorded.

Analysis : The learner choosing actions stochastically outperforms the learner playing deterministically. For the case of $\alpha = 0.05$, there seem to be some fluctuations and for a short duration the deterministic agent plays better, but loses as the number of games is increased.

EXPERIMENT : Actions with Q-values $>= 97\%$ of the highest Q-value

Methodology : The second set of experiments involved the Q agent learning against another Q agent (both using full Q-tables). The sequence of steps outlined in Table 15 was followed. Q-values were recorded every 5000 games.

Results : In Table 17, the learner is tested against a trained opponent. The data shown are for learning rates 0.025 and 0.05. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw. The agent marked "pct" picks one action from the actions that have Q-values above 97% of the highest Q-value for that state. The other agent picks the action with the maximum Q-value in each state.

Figure 19 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points at which the Q-values were recorded.

α =0.05				α =0.025			
Num of train games	Lrn	Lrn (pct)	Draw	Num of train games	Lrn	Lrn (pct)	Draw
5000	31	628	340	5000	144	305	550
15000	156	277	566	15000	401	415	183
25000	68	446	485	25000	123	428	447
35000	143	508	348	35000	79	441	478
45000	250	168	581	45000	86	177	736
55000	410	338	251	55000	134	496	368
65000	88	533	378	65000	352	217	430
75000	348	325	326	75000	153	501	345
85000	330	326	342	85000	41	148	810
95000	169	355	475	95000	251	418	330

Table 17: Learning Agent (Q-table) vs Learning Agent (Q-table): The above table shows data for two learning agents. Each agent learns every alternate 100 games. During testing the agent in the second column plays deterministically (picks the maximum from Q-table) while the agent in third column chooses any action with a Q-value >= 97% of the highest Q-value.

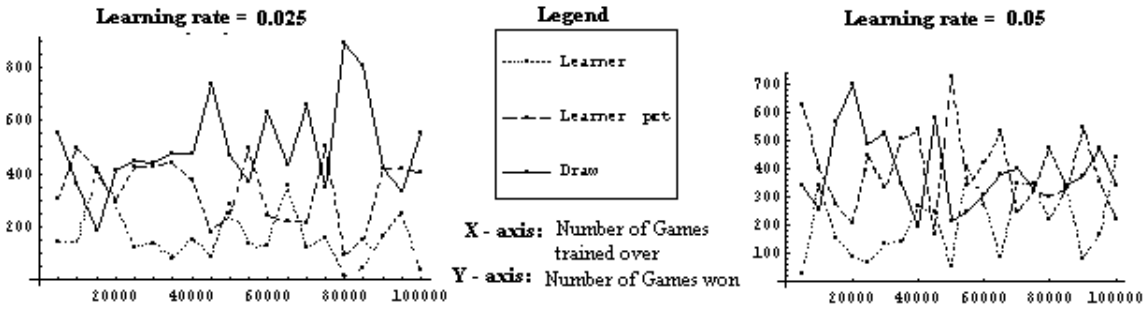

Figure 19: Graph Plots (Table 17) for Learning Agent (Q-table) trained against a Learning Agent (Q-table): The plot on the left corresponds to $\alpha = 0.025$, the one on the right corresponds to $\alpha = 0.05$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by each player. During testing, the agent marked "pct" picks any action with a Q-value that is 97.0% above the highest Q-value.

Analysis : Clearly, in both cases, the deterministic agent does not fare well against its opponent. While there is noticeable fluctuation, the opponent continues to win a significant number of games.

EXPERIMENT : Any among top 3 actions at test

Methodology : This set of experiments involved the Q agent learning against another Q agent (both using full Q-tables). The sequence of steps outlined in Table 15 was followed. Q-values were recorded every 5000 games.

Results : In Table 18, the learner is tested against a trained opponent. The data shown are for learning rates 0.025 and 0.05. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw. The agent marked (top 3) picks one action from the actions that correspond to the 3 highest Q-values. The other agent picks the action with the maximum Q-value in each state.

Figure 20 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points at which the Q-values were recorded.

Analysis : Again, the deterministic agent loses to its opponent. The win percentage seems to be much lower in this case than the other two cases, where the learner at least managed to beat its opponent for some specific Q-values.

α =0.05				α =0.025			
Num of train games	Lrn	Lrn (top 3)	Draw	Num of train games	Lrn	Lrn (top 3)	Draw
5000	165	788	45	15000	405	448	146
25000	124	599	276	35000	218	716	65
45000	328	555	116	55000	371	510	117
65000	256	580	162	75000	369	563	66
85000	314	546	139	95000	343	499	156
5000	177	210	612	15000	282	418	299
25000	220	601	178	35000	254	536	209
45000	290	468	241	55000	229	618	152
65000	360	467	172	75000	158	455	386
85000	197	689	113	95000	293	592	114

Table 18: Learning Agent (Q-table) vs Learning Agent (Q-table): The above table shows data for two learning agents. Each agent learns every alternate 100 games. During testing the agent in the second column plays deterministically (picks the maximum from Q-table) while the agent in third column chooses any of the actions with top 3 Q-values.

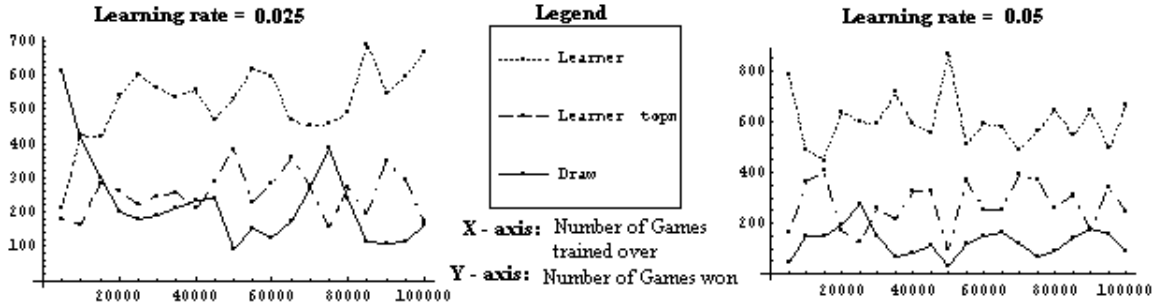

Figure 20: Graph Plots (Table 18) for Learning Agent (Q-table) trained against a Learning Agent (Q-table): The plot on the left corresponds to $\alpha = 0.025$, the one on the right corresponds to $\alpha = 0.05$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by each player. During testing, the agent marked "top 3" picks any action from actions with top 3 Q-values.

4.4.3 Learning Agent (NN) vs Learning Agent (NN)

EXPERIMENT : Stochastic choice of action at test

Methodology : The first set of experiments involved the Q agent learning against another Q agent (both using the connectionist implementation). The sequence of steps outlined in Table 15 was followed. The weights of the neural network were recorded every 5000 games.

Results : In Table 19, the learner is tested against a trained opponent. The data shown are for learning rates 0.005 and 0.01. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw. The agent marked "exp" picks one action stochastically using the probabilities for each action (actions with higher Q-values are assigned higher weights.) The other agent picks the action with the maximum Q-value in each state.

Figure 21 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points at which the Q-values were recorded.

Analysis : Clearly, the learner choosing the action corresponding to maximum value from the Q-values at any state, beats the opponent picking an action stochastically. This is in contrast to the Q-table implementation.

EXPERIMENT : Actions with Q-values $>= 97\%$ of the highest Q-value

Methodology : The second set of experiments involved the Q agent learning against another Q agent (both using the connectionist implementation). The sequence of steps outlined in Table 15 was followed. The weights of the neural network were recorded every 5000 games.

Results : In Table 20, the learner is tested against a trained opponent. The data shown

α =0.01				α =0.005			
Num of train games	Lrn	Lrn (exp)	Draw	Num of train games	Lrn	Lrn (exp)	Draw
5000	583	335	81	5000	799	167	33
15000	693	249	57	15000	726	214	58
25000	675	261	62	25000	693	251	55
35000	756	191	51	35000	715	234	50
45000	734	211	54	45000	734	215	50
55000	767	187	44	55000	668	245	86
65000	776	180	43	65000	633	294	72
75000	589	315	94	75000	777	182	40
85000	737	209	53	85000	723	216	60
95000	827	140	32	95000	712	235	52

Table 19: Learning Agent (NN) vs Learning Agent (NN): The above table shows data for two learning agents. Each agent learns every alternate 100 games. During testing the agent in the second column plays deterministically (picks the maximum from Q values) while the agent in third column chooses with an exponential distribution

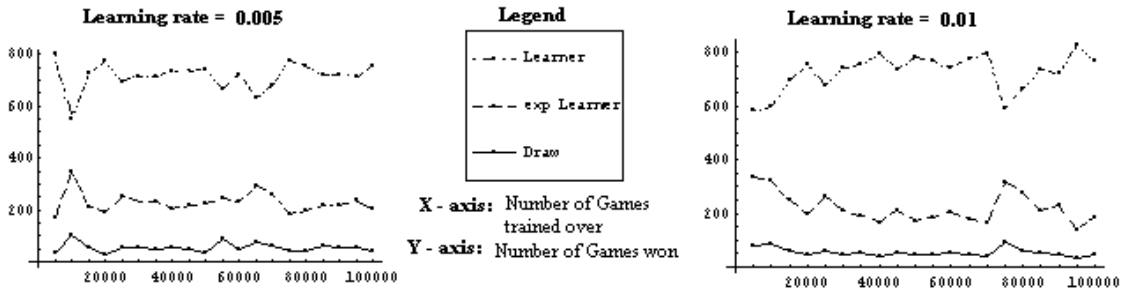

Figure 21: Graph Plots (Table 19) for Learning Agent (NN) trained against a Learning Agent (NN): The plot on the left corresponds to $\alpha = 0.005$, the one on the right corresponds to $\alpha = 0.01$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by each player. The agent marked "exp" picks actions stochastically during testing.

α =0.01				α =0.005			
Num of train games	Lrn	Lrn (pct)	Draw	Num of train games	Lrn	Lrn (pct)	Draw
5000	353	576	70	5000	540	367	92
15000	325	608	65	15000	380	442	177
25000	289	526	184	25000	352	573	74
35000	514	435	50	35000	300	411	287
45000	468	395	135	45000	469	478	51
55000	403	420	176	55000	372	509	118
65000	493	372	134	65000	298	589	111
75000	363	468	167	75000	585	372	42
85000	387	537	74	85000	409	538	52
95000	539	346	114	95000	373	579	46

Table 20: Learning Agent (NN) vs Learning Agent (NN): The above table shows data for two learning agents. Each agent learns every alternate 100 games. During testing the agent in the second column plays deterministically (picks the maximum from Q values) while the agent in third column chooses any action with a Q-value >= 97% of the highest Q-value.

are for learning rates 0.005 and 0.01. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw. The agent marked "pct" picks one action from the actions that have Q-values above 97% of the highest Q-value for that state. The other agent picks the action with the maximum Q-value in each state.

Figure 22 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points at which the Q-values were recorded.

Analysis : In this case, there is only a slight difference between the two players. This is again in contrast to the Q-table implementation, where the learner playing deterministically

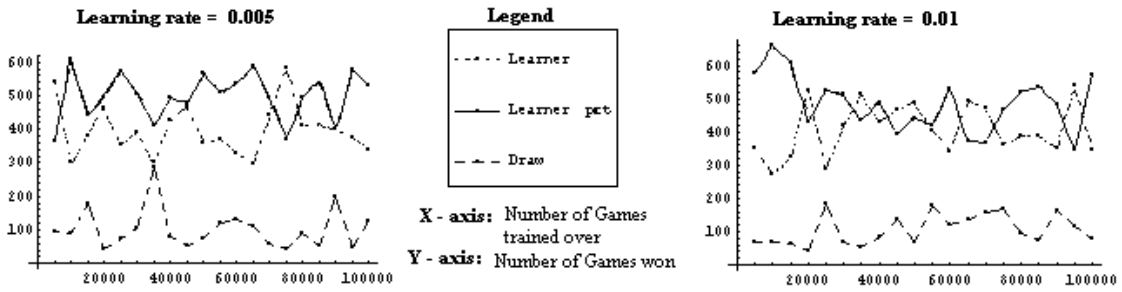

Figure 22: Graph Plots (Table 20) for Learning Agent (NN) trained against a Learning Agent (NN): The plot on the left corresponds to $\alpha = 0.005$, the one on the right corresponds to $\alpha = 0.01$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by each player. During testing, the agent marked "pct" picks any action with a Q-value that is 97.0% above the highest Q-value.

lost a significant number of games.

EXPERIMENT : Any among top 3 actions at test

Methodology : This set of experiments involved the Q agent learning against another Q agent (both using connectionist implementation). The sequence of steps outlined in Table 15 was followed. The weights of the neural network were recorded every 5000 games.

Results : In Table 21, the learner is tested against a trained opponent. The data shown are for learning rates 0.005 and 0.01. The column on the left shows the points at which the Q-values were recorded during training. Columns 2 and 3 give the number of wins recorded by each player. The last column corresponds to the number of games that ended in a draw. The agent marked (top 3) picks one action from the actions that correspond to the 3 highest Q-values. The other agent picks the action with the maximum Q-value in each state.

Figure 23 gives a plot of the wins recorded for each player. Each plot is plotted for 20 points at which the Q-values were recorded.

α =0.01				α =0.005			
Num of train games	Lrn	Lrn (top 3)	Draw	Num of train games	Lrn	Lrn (top 3)	Draw
5000	413	522	64	5000	593	356	49
15000	461	488	49	15000	522	412	64
25000	403	539	57	25000	395	552	52
35000	549	409	40	35000	459	495	46
45000	543	405	51	45000	482	481	36
55000	510	429	60	55000	491	424	83
65000	588	366	44	65000	410	554	35
75000	389	557	53	75000	599	354	46
85000	412	515	72	85000	462	497	39
95000	626	336	37	95000	458	496	45

Table 21: Learning Agent (NN) vs Learning Agent (NN): The above table shows data for two learning agents. Each agent learns every alternate 100 games. During testing the agent in the second column plays deterministically (picks the maximum from Q values) while the agent in third column chooses any of the actions with top 3 Q-values.

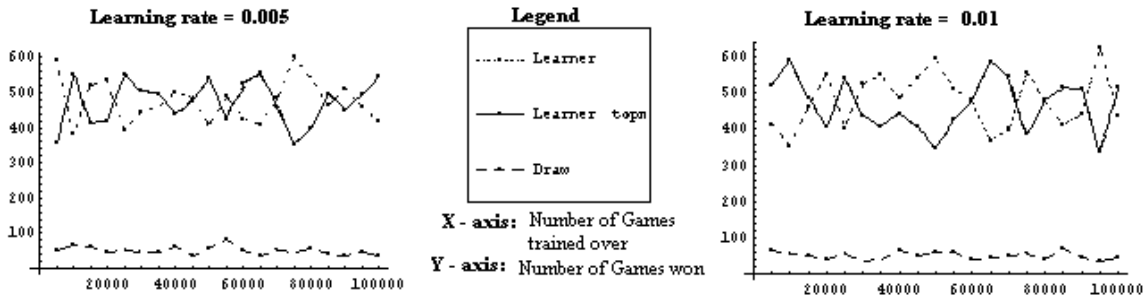

Figure 23: Graph Plots (Table 21) for Learning Agent (NN) trained against a Learning Agent (NN): The plot on the left corresponds to $\alpha = 0.005$, the one on the right corresponds to $\alpha = 0.01$. The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by each player. During testing, the agent marked "top 3" picks any action from actions with top 3 Q-values.

Analysis : Again, the learner choosing deterministically, plays its opponent with a greater ability than the Q-table implementation.

4.4.4 Cross Testing

EXPERIMENT : Learning Agent (Q-table) trained against Learning Agent (Q-table) tested against Programmed agent

Methodology : In this set of experiments, no training was done. Instead, the learning agents (using full Q-table) trained against learning agent (using full Q-table) were used. The agents were tested against the programmed opponent for different strategies of choosing actions.

Results : In Table 22, the The data shown is for learning rate 0.025. The column on the left shows the points at which the Q-values were recorded during training.

	det			exp			pct			top 3		
Num train games	Lrn det	Prg	Drw	Lrn exp	Prg	Drw	Lrn pct	Prg	Drw	Lrn top 3	Prg	Drw
8500	110	470	419	213	565	221	160	181	658	175	349	474
25500	165	152	682	216	565	218	142	191	665	161	294	544
42500	168	198	633	214	564	220	133	128	738	181	243	575
59500	199	125	675	221	564	214	143	216	640	114	305	579
76500	273	147	579	200	593	206	161	260	577	184	584	231
93500	83	109	807	221	563	214	203	178	618	181	325	493
127500	158	112	729	211	586	202	233	103	662	242	494	263
144000	171	76	752	204	585	210	224	228	546	231	441	327
161500	148	125	725	202	583	215	165	121	713	202	396	401
195500	150	135	714	210	572	216	105	126	768	182	339	477

Table 22: Learning Agent (Q-table) vs Learning Agent (Q-table) Tested against programmed opponent: The above table shows results for the learning rate set at 0.025. "Swap" is fixed at 100. The different strategies for choosing actions is shown as "det" corresponds to choosing actions deterministically, "exp" is choosing actions stochastically, "pct" corresponds to choosing an action from all the actions with Q-value greater than 97% of the highest Q-value, "top 3" is choosing an action from the set of actions with top 3 Q-values.

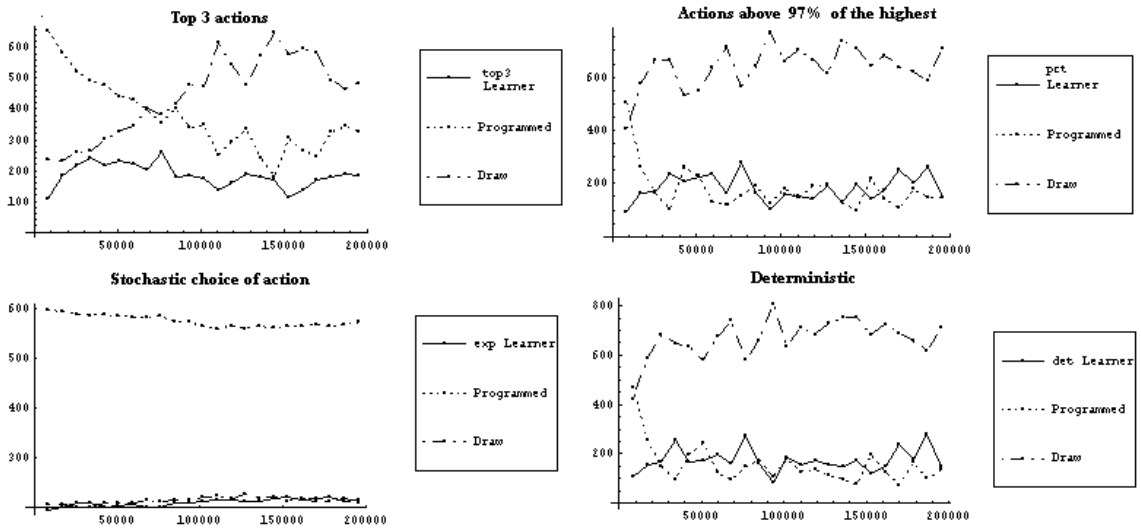

Figure 24: Graph Plots (Table 22) for Learning Agent (Q-table) trained against a Learning Agent (Q-table): Tested against programmed opponent. The plots correspond to $\alpha = 0.025$ The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by each player. During testing, the agent marked "top 3" picks any action from actions with top 3 Q-values.

Analysis : The learned agent does not beat its opponent in most of the cases. Among the 4 different means of choosing actions during testing, the deterministic choice of actions seems to work best. In this case alone, the learning agent outplays the opponent at some points (as seen from the bottom-right graph in Figure 24).

EXPERIMENT : Learning Agent (NN) trained against Learning Agent (NN) tested against Programmed agent

Methodology : In this set of experiments, no training was done. Instead, the learning agents (using connectionist implementation) trained against learning agent (using connectionist implementation) were used. The agents were tested against the programmed opponent for different strategies of choosing actions.

Results : In Table 23, the The data shown is for learning rate 0.005. The column on the left shows the points at which the Q-values were recorded during training.

Analysis : The learned agent comes close to beating its opponent in most of the cases. Among the 4 different means of choosing actions during testing, the deterministic choice of actions seems to work best. In this case, the learning agent outplays the opponent at some points (as seen from the bottom-right graph in Figure 25). In the other cases, the learning agent manages to stay close and is not completely outplayed

Clearly, from Figures 24 and 25, we can see that the learning agent using the connectionist model performs better against all programmed opponent when compared to the Q-table implementation. An interesting point to note would be the number of draws in both cases, which probably signifies that the learning agent actually learns to get rewards, whereas the Q-table agent learns to stay away from punishments.

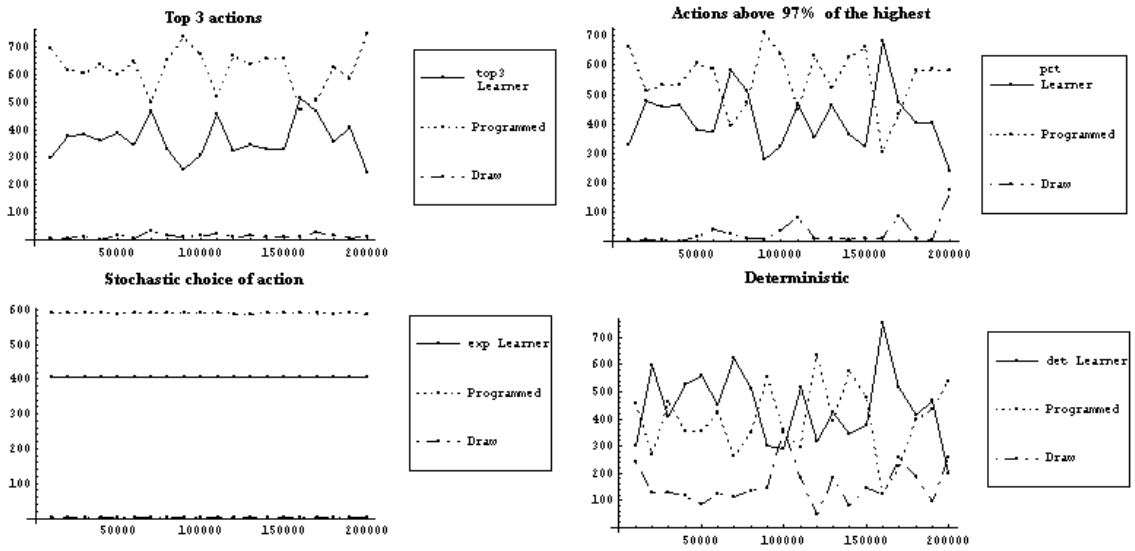

Figure 25: Graph Plots (Table 23) for Learning Agent (NN) trained against a Learning Agent (NN): Tested against programmed opponent. The plots correspond to $\alpha = 0.005$ The X-axis represents the regular intervals in which the Q-values were recorded during training, Y-axis is the number of wins by each player. During testing, the agent marked "top 3" picks any action from actions with top 3 Q-values.

	det			exp			pct			top 3		
Num of train games	Lrn det	Prg	Drw	Lrn exp	Prg	Drw	Lrn pct	Prg	Drw	Lrn top 3	Prg	Drw
10000	302	458	239	406	589	4	330	661	7	298	692	9
30000	406	463	130	406	589	4	459	532	7	384	604	11
50000	560	352	87	407	588	4	377	605	17	386	596	17
70000	625	261	113	407	589	4	582	392	25	467	498	34
90000	299	554	146	406	589	4	280	709	9	254	734	11
110000	517	298	183	406	589	4	467	449	82	457	520	21
130000	423	394	182	407	588	4	464	521	14	346	637	15
150000	377	477	145	406	590	4	326	661	12	331	657	11
170000	518	225	256	406	589	4	475	436	88	467	507	25
190000	467	435	96	406	589	4	406	585	7	407	585	7

Table 23: Learning Agent (NN) vs Learning Agent (NN) Tested against programmed opponent: The above table shows results for the learning rate set at 0.005. "Swap" is fixed at 100. The different strategies for choosing actions is shown as "det" corresponds to choosing actions deterministically, "exp" is choosing actions stochastically, "pct" corresponds to choosing an action from all the actions with Q-value greater than 97% of the highest Q-value, "top 3" is choosing an action from the set of actions with top 3 Q-values.

4.5 Summary

The above experiments were done using the results of the learning rate experiments done earlier. Based on the above analysis, the following can be summarized:

- Generally, picking the action corresponding to the maximum Q-value works better for the connectionist implementation. It is not clear about the rest of the strategies, though the stochastic choice of action does better than the others.
- The above statement does not hold good for the Q-table implementation. There is no clear winner in this case.

- In the cases of cross-testing, the neural network clearly does better (against the programmed opponent) than the Q-table implementation.

5 Limitations and Future Work

An obvious limitation of the thesis is the specific nature of the problem. Nevertheless, many issues concerning multi-agent learning were discussed even in such a simple domain. Some techniques followed in this thesis can be extended to more complex domains.

One key issue of multi-agent learning is that other agents are learning. The problem can be broadened by introducing more than two players and simulating a "dog fight". Such a domain would address issues concerning team strategies.

One limitation of multi-agent domains is that the amount of exploration of the agent grows exponentially with agents. This would play a role when there are more than two agents in the system.

There are some practical problems (as regards this particular game) that can be looked into such as the issue of deadlock. It would be interesting to see if the results change noticeably if the method of breaking the deadlock is changed. Currently, any game going beyond a certain number of moves is considered drawn.

Cross testing can be attempted in various different ways. For example, consider a learning agent being trained against another learning agent. During testing, we could test the same learners at different levels of their learning. i.e an agent trained for 1000 games can be tested against its opponent trained for 2000 games.

Co-evolution can be attempted by training agents of different types against each other. In this thesis, one learning agent learns against another of the same type. A connectionist model could be made to learn against the Q-table implementation. This might result in another group of agents, and give a more direct measure of comparing the performance of the learning agents.

6 Conclusions

In this thesis, I implement the Q-learning algorithm in two different ways, and evaluate them with respect to a learning task that involves two competing, game playing agents in a world with a finite number of states. Experiments on this task indicate that claims made earlier, are indeed valid, in the context of the game.

In the preliminary tests, I found that the agents learn to beat specific opponents with simple strategies. The results showed that each agent learns at a different rate justifying the claim made at the beginning: *The learning task is tackled by both agents quite well, though their approaches and speed of learning are quite different*

At learning rates close to 0, a small increase in the learning rate increases the speed of learning, but becomes highly uneven when the rate parameter is set close to 1. This was shown by setting the parameter to 0.9. From these sets of experiments, the learning rates were fixed for each of the opponents for the future set of experiments. For the Q-table implementation a rate of 0.025 or 0.05 was chosen, whereas for the neural network rates of 0.005 or 0.01 were preferred.

The second claim: *both agents have the capability to generalize, but the connectionist implementation outperforms the Q-table, for simple cases*, made in the thesis pertains to the ability of the agents to generalize and play opponents that they were not trained against. The learning agents were trained against programmed opponents and tested against random opponents. Both agents were found to be able to beat the random opponent, but differed significantly in their respective percentage of wins. It was clearly seen that the neural network generalizes better, in the above case.

The main contribution of this thesis is to investigate *co-evolution*, or the process where both agents are learning. The final claim was made in this regard: *The neural network*

implementation responds better to co-evolution than the full Q-table.

A set of strategies were developed to allow for a variety of competitive agents. The strategies were aimed at introducing some randomness into the learned agents. The connectionist model with a deterministic action-picking strategy outplays the other strategies. Further, the model does better against the programmed opponent than the Q-table implementation.

In this work, I have addressed multi-agent learning in a competing environment. I have shown that the claims made earlier can be justified by studying the experiments and the results.

References

[Anderson,1987] Anderson, C. (1987) Strategy learning with multilayer connectionist representations, *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA.

[Barto et.al.,1983] Barto, A., Sutton, R., & Anderson, C. (1983) Neuronlike adaptive elements that can solve difficult learning control problems *IEEE Transactions on Systems, Man, and Cybernetics*

[Fikes,1971] Fikes, R.E., Nilsson, N.J. (1971) STRIPS: a new approach to the application of theorem proving to problem solving, *Artificial Intelligence*

[Holland,1986] Holland, J. (1986) Escaping Brittleness: The possibilities of general-purpose earning algorithms applied to parallel rule-based systems, *Machine Learning: An AI Approach (vol 2)*, Morgan Kauffman, San Mateo, CA.

[Lang et.al.,1990] Lang, K.J., Waibel, A.H., & Hinton, G.E. (1990) A time-delay neural network architecture for isolated word recognition, *Neural Networks, vol 2.*

[LeCun et. al.,1989] LeCun, Y. Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W. & Jackel, L.D. (1989) Backpropagation applied to handwritten zip code recognition, *Neural Computation.*

[Lin,1992] Lin, L. (1992) Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching *Machine Learning*, Kluwer Academic Publ., Boston.

[Maclin,1993] Maclin, R. & Shavlik, J. (1993) Using knowledge-based neural networks to improve algorithms: Refining the Chou-Fasman algorithm for protein folding. *Machine Learning.*

[McCarthy,1968] McCarthy, J. (1968) Programs with common sense, *Semantic Information Processing*, Cambridge, Mass, MIT Press.

[Nilsson,1980] Nilsson, N.J. (1980) *Principles of Artificial Intelligence*, Palo Alto, CA, Tioga Publishers.

[Pednault,1990] Pednault, E. (1990) Formulating Multiagent, dynamic-world problems in the classical planning framework, Readings In Planning eds. Allen, J., Hendler J., Tate, A., Moragn Kauffman Publishers, San Mateo, CA.

[Pomerleau,1993] Pomerleau, D.A. (1993) Knowledge-based training of artificial neural networks for autonomous robot driving, *Robot Learning*, Kluwer

[Prior,1967] Prior, A.N. (1967) *Past, Present and Future*, Oxford, Clarendon Press.

[Rosenblatt,1962] Rosenblatt, F. (1962) *Principles of Neurodynamics*, Spartan, NY.

[Rumelhart et. al. 1986] Rumelhart, D., Hinton, G., & Williams, R. (1986) Learning internal representations by error propagation. In Rumelhart, D. & McClelland, J. editors, *Parallel Didtributed Processing: Explorations in the microstructure of cognition (vol 1)* MIT Press, Cambridge, MA. Academic Publishers.

[Salustowicz,1998] Salustowicz, R., Wiering, M.A., Schmidhuber, J. (1998) Learning Team Strategies: Soccer Case Studies *Machine Learning*, Kluwer Academic Publishers, Boston.

[Samuel,1959] Samuel, A. (1959) Some studies in machine learning using the game of checkers *IBM Journal on Research & Development*, Reprinted in E. Feigenbaum and J.Feldman, eds., 1963, *Computers and Thought*, McGraw Hill, NY.

[Sutton,1984] Sutton, R. (1984) *Temporal aredit assignment in reinforcement learning*, PhD Thesis, Dept. of Computer & Informations Science, University of Massachusetts.

[Sutton,1988] Sutton, R. (1988) Learning to predict by the method of temporal differences, *Machine Learning*, 3, Kluwer Academic Publishers, Boston, MA.

[Tate,1984] Tate, A. (1984) Planning in Expert Systems *Alvey IKBS Expert Systems Theme - First Workshop*, Oxford, March.

[Watkins,1989] Watkins, C.J.C.H. (1989) *Learning from delayed rewards*, PhD Thesis, University of Cambridge, England.