Learning to learn:
An Automated and Continuous Approach to Learning in Imperfect Environments

by

Hasan Mujtaba, MS (CS), MSc (CS)

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy to the FAST National University of Computer & Emerging Sciences

Department of Computer Science
FAST National University of Computer & Emerging Sciences,
Islamabad, Pakistan.
(June 2010)
Continuity of Learning in an Uncertain & Dynamic Environment of Imperfect Information
This page intentionally left blank
Dedicated to my parents, my siblings, my wife and my son...
Acknowledgements

I would like to bow my head before Allah Almighty, the Most Gracious, and the Most Merciful, whose benediction bestowed upon me talented teachers, provided me sufficient opportunity, and enabled me to undertake and carry out this research work.

This dissertation describes research undertaken at the department of Computer Science, National University of Computer & Emerging Sciences, Islamabad, between September 2006 and June 2010, under the supervision of Prof. Dr. Abdul Rauf Baig, to whom I am indebted for suggesting the subject and for his encouragement, guidance and support throughout the research program. His advice, discussion, and effective comments were always a source of motivation.

I am also thankful to Dr. Anwar Majeed Mirza, who not only managed and administrated my research activities, but also guided and encouraged my innovative ideas and gave me full moral support at every stage of my research work. I am also extremely thankful to Dr. Aftab Maroof, and Dr. Arshad Ali Shahid who not only allowed me to do my research work in FAST, but also appreciated my innovative ideas and gave me full moral support.

I would like to thank Dr. Graham Kendall for his guidance and cooperation. His comments and suggestions have improved the quality of this research tremendously. I would like to thank Ms, Deborah Pictforsk for all her help and assistance during my stay at Nottingham.

I would not have reached this stage but for the prayers, love and moral support of my mother and father. I would like to thank my brothers, Usman and Saad, my sister Kiran and my cousins Faisal, Fahad, Waqas, Kashif, Umer, Mahrulk, Aneeka, Nadia, Ayesha, Rabia, Amina, Mariyam, and all other well-wishers. I would also like to pay tribute to my friends A. Basit, A. Rauf, Ayyaz Hussain, Farrukh Zaib, Kashif Bashir, Hassan Shah, Liaqat Zamman, M. Ramzan, M. Amjad, M. Nazir, Naveed Kazim, Sohail Akhtar, Shehzad, Syed Adeel Naqvi, Waqas Cheema, Zahid Hussain.

I acknowledge the enabling role of the Higher Education Commission of Pakistan and appreciate their financial support through its Indigenous PhD Scheme.
## Contents

Chapter 1: Introduction .................................................................1
  1.1 Problem Statement ...............................................................1
  1.2 Background and Motivation ..................................................4
  1.3 Contribution ........................................................................8
  1.4 Thesis Organization .............................................................8

Chapter 2: AI & Evolutionary Learning ........................................12
  2.1 Introduction .........................................................................12
  2.2 Definitions of Artificial Intelligence .......................................13
  2.3 Disciplines of AI .................................................................14
    2.3.1 Computer Vision ..........................................................15
    2.3.2 Knowledge Representation .............................................15
    2.3.3 Natural Language Processing ..........................................16
    2.3.4 Automated Planning and Scheduling ...............................17
    2.3.5 Robotics ........................................................................17
  2.4 Machine Learning ...............................................................18
    2.4.1 Decision tree learning ...................................................19
    2.4.2 Instance based learning ..................................................20
    2.4.3 Case-based Reasoning (CBR) ..........................................21
    2.4.4 Bayesian learning ........................................................22
    2.4.5 Heuristic Learning .........................................................23
    2.4.6 Genetic learning ............................................................24
    2.4.7 Reinforcement Learning ...............................................26
    2.4.8 Cultural Algorithms ......................................................27
2.5 Artificial Neural Networks ................................................................. 27
   2.5.1 Learning Paradigms in ANNs ......................................................... 28
   2.5.2 Perceptrons .................................................................................. 29
   2.5.3 Backpropagation Learning ......................................................... 32
   2.5.4 Recurrent Networks .................................................................... 32
   2.5.5 Issues with Backpropagation ..................................................... 33
   2.5.6 Evolutionary ANNs ..................................................................... 34
2.6 Particle Swarm Optimization ............................................................. 37
2.7 Computer Game Playing ................................................................. 38
   2.7.1 Blondie24 ................................................................................... 41
   2.7.2 NERO ......................................................................................... 42
2.8 Limitations of Traditional Learning .................................................. 44
2.9 Summary: .......................................................................................... 46

Chapter 3: Imperfection in Evolutionary Systems .................................... 47
   3.1 Introduction ...................................................................................... 47
   3.2 Imperfect Evolutionary Systems ...................................................... 48
   3.3 Components of an Imperfect Evolutionary System ....................... 49
   3.4 Individuals relationship with its Environment ................................ 53
   3.5 Intelligence in an IES ..................................................................... 55
   3.6 Summary ......................................................................................... 58

Chapter 4: Continuous Learning Framework ........................................ 59
   4.1 Introduction ...................................................................................... 59
   4.2 Continuity of Learning .................................................................... 60
      4.2.1 Experience ............................................................................... 60
      4.2.2 Exploration ............................................................................. 61
4.2.3 Exploitation ..................................................................................................61

4.3 Individual Learning .............................................................................................62

4.4 Social Learning ...................................................................................................63

4.4.1 Community Based Learning ........................................................................64

4.5 Continuous Learning Framework ........................................................................64

4.6 Summary .............................................................................................................68

Chapter 5: Dynamic & Imperfect Gaming Environment ...............................................69

5.1 Introduction .........................................................................................................69

5.2 An Imperfect World ............................................................................................70

5.3 Agents and Artifacts in the Environment ............................................................71

5.3.1 Humanoids ...................................................................................................71

5.3.2 Terminals .....................................................................................................72

5.3.3 Creatures ......................................................................................................73

5.4 Phases in the Environment ..................................................................................73

5.4.1 Terminal Acquisition: ..................................................................................74

5.4.2 Avoiding Death ............................................................................................74

5.4.3 Capture Creatures.........................................................................................74

5.5 Agent Movements ...............................................................................................76

5.6 Summary: ............................................................................................................83

Chapter 6: Continuity of Learning in DIGE ..................................................................84

6.1 Introduction .........................................................................................................84

6.2 Specification of Learning Algorithm .....................................................................85

6.2.1 Particle Swarm Optimization .......................................................................85

6.2.2 Artificial Neural Network ............................................................................87

6.2.3 Experience pool policies ..............................................................................90
6.3 Experimental Setup ........................................................................................................ 91
6.3.1 Artificial Neural Network ............................................................................................ 92
6.3.2 Parameters of PSO ...................................................................................................... 93
6.3.3 Computational Cost and Implementation Details ...................................................... 95
6.4 Perfect Environment .................................................................................................... 96
6.4.1 Evaluation Criteria to determine learning ability .................................................... 97
6.5 Coping with Imperfection ............................................................................................. 98
6.6 Retaining historical lessons ........................................................................................ 99
6.7 Adding a new dimension to learning ........................................................................... 105
6.8 Summary ...................................................................................................................... 108

Chapter 7: Adaptive Game Intelligence ........................................................................... 110
7.1 Introduction ................................................................................................................ 110
7.2 Predictability in games ............................................................................................... 111
7.3 Automated Game Learning ......................................................................................... 114
7.4 Summary ...................................................................................................................... 116

Chapter 8: Conclusion ...................................................................................................... 117
References ..................................................................................................................... 120
List of Figures

Figure 2.1 Decision Tree for the concept of *Play Tennis* ................................................................. 19
Figure 2.2 A Bayesian network. ........................................................................................................ 22
Figure 2.3 Reinforcement learning based on agents interactions with environment ........ 26
Figure 2.4 Perceptron basic processing unit .................................................................................. 30
Figure 2.5 A two-layer perceptron .................................................................................................. 31
Figure 2.6 A three-layer perceptron ................................................................................................ 31
Figure 2.7 Simple Recurrent Network ............................................................................................. 33
Figure 2.8 EPNet for concurrent network architecture and connection evolution ................. 36
Figure 2.9 The main replacement cycle in rtNEAT ........................................................................ 43
Figure 3.1 Imperfection in Evolutionary Environment ..................................................................... 49
Figure 3.2 Individuals in an Imperfect Evolutionary Environment ............................................. 57
Figure 4.1 Continuity of Learning .................................................................................................... 60
Figure 5.1 Imperfect World with different races in it ....................................................................... 71
Figure 5.2 Different phases of the environment ............................................................................... 76
Figure 5.3 An agent “o” moving towards a terminal “x” .................................................................. 82
Figure 6.1 Architecture used for defining and connecting sub-swarms ...................................... 87
Figure 6.2 ANN Architecture used for storing Humanoids strategy ............................................. 92
Figure 6.2 Learning Ability of CLF in Perfect Environments ..................................................... 103
Figure 6.3 Comparison of CLF Humanoids against Untrained Strategies ................................... 103
Figure 6.4 Continuity of learning in an imperfect environment ................................................. 104
Figure 6.5 Comparison of learning with and without a social experience pool ..................... 105
Figure 6.6 Performance statistics of various strategies ................................................................. 106
Figure 6.7 Continuity of learning in multiple dimensions ............................................................. 108
Figure 7.1 Ignoring the changes occurring in the surroundings ................................................. 115
Figure 7.2 Reacting to changes occurring in the surroundings .................................................... 116
List of Tables

Table 2.1 An ID3 algorithm for decision tree learning .........................................................20
Table 2.2 A Genetic Algorithm ..........................................................................................24
Table 2.3 Some commonly used non-linear activation functions in MLPs .........................31
Table 2.4 A typical cycle for evolving connection weights in EANN’s. ...............................35
Table 2.5 A typical cycle for evolving network architectures in EANN’s ..............................35
Table 2.6 Operations performed every n nticks by rtNEAT ..................................................43
Table 4.1 Continuous Learning Framework ..........................................................................66
Table 6.1 Learning Algorithm based on Continuous Learning Framework .........................86
Table 6.2 Learning Algorithm Parameters ..........................................................................90
Table 6.3 Switching between environment states for socio-experience comparison ..........91
Table 7.1 Automated Game Learning Technique ................................................................113
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>CA</td>
<td>Cultural Algorithm</td>
</tr>
<tr>
<td>CBR</td>
<td>Case Based Reasoning</td>
</tr>
<tr>
<td>CLF</td>
<td>Continuous Learning Framework</td>
</tr>
<tr>
<td>CV</td>
<td>Computer Vision</td>
</tr>
<tr>
<td>ES</td>
<td>Expert Systems</td>
</tr>
<tr>
<td>EC</td>
<td>Evolutionary Computation</td>
</tr>
<tr>
<td>EL</td>
<td>Evolutionary Learning</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>GP</td>
<td>Genetic Programming</td>
</tr>
<tr>
<td>IES</td>
<td>Imperfect Evolutionary System</td>
</tr>
<tr>
<td>KR</td>
<td>Knowledge Representation</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi Layer Perceptrons</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
</tbody>
</table>
Abstract

Our quest to understand, model, and reproduce natural intelligence has opened new avenues of research. One such area is artificial intelligence (AI). AI is the branch of computer science aiming to create machines able to engage in activities that humans consider intelligent. The ability to create intelligence in a machine has intrigued humans ever since the advent of computers. With recent advancements in computer science we are coming closer every day to the realization of our dreams of smarter or intelligent machines. New algorithms and methods are constantly being designed by researchers. However these techniques must be evaluated and their performance compared before they can be accepted. For this purpose games have caught the attention of AI researchers and gaming environment have proven to be excellent test beds for such evaluation. Although games have redeemed AI research, one limitation most researchers have applied is of perfect information. Perfect information environments imply that the information available to the agents in the environment does not change. Essentially what this means is that agents can detect entities that they have been trained for but will ignore entities for which training has not taken place. This limitation results in agents that do not gain a single iota of learning while they are in the environment. Whatever learning has taken place during their training, they will not increase upon it. This would all be fine if we were living in a static world of perfect information, but we do not!

Learning in such an unpredictable and changing environment is a continuous process for the agents. For this reason we developed a “Continuous Learning Framework” (CLF). CLF enables each agent to detect the changes in the environment and take necessary action accordingly. Agents who fail to do so die out during the evolutionary process. CLF based learning is triggered by stimulus from the environment. We have intentionally kept CLF independent of this environment or of the underlying evolutionary approaches, allowing our CLF to be ported to other environments with dynamic nature. Learning new abilities and adapting successful strategies is crucial to the survival of species. Results of our experimentation show that CLF not only enables agents to learn new strategies suitable to their current environmental state but also
ensures proper dissemination of information within a species. Forgetfulness is an inherent feature of the co-evolutionary processes. Keeping this in view we have also explored the integration of historical information and the ability to retain and recall past learning experiences. We have tested a social learning based flavor of our CLF to see whether learning from past is profitable for agents. Each of the species was allowed to maintain a social pool of successful strategies. Results from these experiments show that strategy from the pool results in a significant boost to performance in cases where the environmental conditions are similar to when the strategy was established. This social pools acts like a general reservoir of knowledge which is similar in nature to the one we humans hold with ancient civilizations. This historical information also results in performance boosts by eliminating the “reinvention of wheel” phenomena common to evolutionary strategies.

This research not only presents a new way of learning along within a dynamic and uncertain medium but also aims to establish the importance of learning in such an imperfect environment. Much work still needs to be undertaken in this path. Possible future channels of this research include designing better performance evaluation criteria of agents residing in different locations of the environment, and establishing individual archive for learning based on personal experience.
Publications Produced

Following are the publications produced during the course of Ph.D. research.

**International Journal Publications:**


**International Journal Submissions:**


• Zahid Halim, A. Rauf Baig and Hasan Mujtaba, "Evolutionary Search for Entertainment in Games", submitted to Intelligent automation and Soft Computing, [ISSN: 10798587], (SCI, Impact Factor: 0.462).


International Conference Publications:

• Hasan Mujtaba and Rauf Baig, “Retaining the lessons from past for better performance in a dynamic multiple task environment”, Proceedings of the Eleventh conference on Congress on Evolutionary Computation 2009
Chapter 1: Introduction

A taste of things to come

I cannot teach anybody anything, I can only make them think
- Socrates

1.1 Problem Statement

Our quest to understand, model and reproduce natural intelligence has opened many avenues of research. One such area is artificial intelligence (AI). AI is the branch of computer science which aims at creating machines that are able to engage in activities that humans consider intelligent. AI encompasses many areas of studies including search, logic, planning, learning and complexity. An intelligent system has to act according to the peculiarities of its surroundings. Its behavior must be in coherence with its current situation. It must cater to changes in its environment and its own objectives. It must also make appropriate decisions given its perceptual and computational limitations [1]. This ability to create intelligence in machine has intrigued humans since the advent of computers and with recent advancements in computer science we are coming closer to the realization of our dream of intelligent machines.

One of the aims of research in AI is to improve our understanding of natural intelligence, and mimic intelligent behaviors in machines and other man made systems. In other words AI can be thought of as the quest for creating machines that are capable of solving problems that require human level intelligence. People have dreamt of human like machines (or humanoids) for ages (for example, Frankenstein). In current times AI has grown into a mature pillar of computer science and provides heavy lifting for many of the most difficult problems in different domains [2 - 4]. For some AI textbooks, see [5 – 7].
New algorithms and methods designed by AI researchers must be evaluated and their performance compared before they can be accepted. Games have had the attention of AI researchers for quite some time now and have proven to be excellent test beds for such evaluations. Gaming environments present us with a controllable and customizable universe where rules can be adjusted as per needs (just as we humans live in the real world, agents can live in a game universe). Agent interaction and behavior can be monitored and controlled within the gaming environment. These environments allow us to test theories and compare results of different algorithms. Using simulated environments, we can estimate performance of these theories in real-world.

Although games have enriched AI research, one limitation most researchers have applied is of perfect information. Perfect information environments imply that the information available to the agents in the environment does not change. Essentially what this means is that agents can deal with situations for which they have been trained for but fail if the environment changes. This limitation also implies that agents do not gain a single iota of learning after they have been trained and while they are in the environment. Nothing is added to the learning that has taken place during their training. This would all be fine if we were living in a static world of perfect information, but we do not.

In this thesis we present the idea of continuity of learning in an incomplete, imperfect and uncertain dynamic environment. Here incompleteness and imperfection of environment means the environment can change i.e. new information can be made available to its inhabitants and previously known information may become obsolete. While uncertainty means that this change in environment can occur at any time without any warning. Note that, in this thesis, perfect and complete, are used interchangeably, same as imperfect and incomplete. A perfect environment does not change and a perfect entity has no capability for continuous learning in changing environments.
In most cases the information available in the environment changes with the passage of time. Each new change (or challenge) opens up new opportunities for the agents i.e. they can formulate new strategies to deal with their environment using the new information made available to them. This dynamic and imperfect nature of the environment is similar to our natural world where things are in a constant state of change. In order to model the imperfectness of information we have created our own game-like environment. Availability of new information opens new avenues of learning. This environment supports multi-species (more than one species of agents) and multi-objective (agents have to perform more than one task to survive). Our environment also includes other entities, each performing a specific task defined by the environment.

Agents living in this environment must continually evolve and learn about their environment. Changes in environments are detected by the agents themselves. The whole evolution process is autonomous and is carried out without any human intervention. This allows us to observe interesting behaviors and strategies adapted by agents to deal with their surroundings. Each agent tries to make best of the information available to it.

Based on computational intelligence techniques (like Particle Swarm optimization and Artificial Neural Networks) we have developed a “Continuous Learning Framework” (CLF). CLF enables each agent to detect changes in the environment and take necessary action accordingly. Agents who fail to do so die during the evolutionary process. CLF based learning is triggered by stimulus from the environment. CLF works independent of its environment and the underlying evolutionary approaches. This enables the CLF to be ported to other dynamic environments. Results of our experimentation show that CLF not only enables agents to learn new strategies suitable to their current environmental state, but ensures dissemination of information among agents.

We have also explored the integration of historical information and the ability to retain and recall past learning experiences. We have tested a social learning based flavor of our CLF to test whether learning from past is profitable for agents. Each of the species is allowed to maintain a social pool of successful strategies. Results from these
experiments show that strategy from the pool results in a significant boost to performance in cases where the environmental conditions are similar to when the strategy was established. This social pools acts like a general reservoir of knowledge which is similar in its nature to the one we humans hold with ancient civilizations. This historical information also results in performance boosts by eliminating the “reinvention of wheel” phenomenon common to evolutionary strategies.

This research not only presents a new way of learning within a dynamic and uncertain medium but also aims to establish the importance of learning in such an imperfect environment.

1.2 Background and Motivation

The motivation behind our research in continuity of learning comes from the introduction of uncertainty and imperfectness into environments. This area of research is particularly important to AI research that deals with incompleteness of information like the field of computer game playing.

To gather new information natural intelligence relies on the surrounding environment. It is an important facet of our intelligence that we are able to learn new things on our own depending upon the information provided to us (usually by our environment). Over the years there have been many efforts to replicate such intelligence into machines.

Using gaming environments, researchers have tried to test the limits of automated (non-human) players. IBM developed Deep Blue [8], [9], a chess playing computer that beat the reigning chess world champion, Garry Kasporov, who is often considered the best player of all time. Chinook, developed by Jonathan Schaeffer and his team, won the world checkers title in 1994 [10], [11], the first automated player to do this and three years before the much more widely publicized success of Deep Blue. Both these systems relied on their brute force approach. Neither systems actually performed any learning during game play, instead they used mathematical models which computed many
possible outcomes from a given point and choose the best one. This led to researchers questioning their intelligence. This purely search based approach fails when a game cannot be modeled or when the possible outcomes are too large. Go, the ancient Chinese game, is an example of games resistant to types of game playing strategy. These brute force based machines relied on human guidance for their success. Blondie24, developed by Chellapilla and Fogel [12], [13], was a key advance from the human dependent approach of previous efforts. Blondie24 was an evolutionary program that learned to play checkers on its own, from scratch, without relying on human input. It applied a co-evolutionary approach to tune the weights of an artificial neural network (ANN) thereby improving its game playing ability. In many ways (detailed in chapter 2) Blondie24 was a major breakthrough in answering Samuel’s challenge of developing game playing programs that learn to play games without pre-injected knowledge [14]. In terms of automated intelligent game playing it was superior to traditional game playing approaches but due to its static neural network architecture it lacked adaptive intelligence. Therefore it didn’t fully answer the challenge of learning when the rules changed.

Stanley et al. [15] developed NERO, a computer game in which agents were able to improve and adapt during the game. They employed a real-time neuro-evolution methodology to constantly evolve complex ANNs which control the agent’s strategy. This was a significant achievement and we believe it to be a major accomplishment in dealing with the challenge of reproducing intelligent behavior in simulated agents. However, in NERO the training of agents is human controlled. Humans decide what inputs should be used to construct the ANNs. This limits the agent’s capacity to improve within the boundaries of what they are given. They lack the ability to perceive new options that may be available in the environment but were not used as inputs or were simply not available at the time of the training, but becomes available at a later stage. If one of the robot team develop a new kind of weapon that could see through solid obstacles, other teams that still use old set of inputs are likely to be defeated.
As mentioned by Kendall and Su in their work [16], this failure to react to new opportunities in the environment limits the intelligence of the system. In traditional approaches the relationship between an intelligent entity and its environment was ignored. All work mentioned above (and others) assume that the environment is perfect. By saying that an environment is perfect we mean that the information present at time $T_i$ is the same as $T_{i+1}$. Whereas we know that real-world environments are not perfect.

Real-world scenarios present an environment which changes over time. This change in the environment can be anything from new inputs becoming available, changes in the objective(s) of the agents or a completely new environment. A change in an agent’s objective is not merely the relocation of the optimum from one location of the search space to another but rather a complete change in the search space. Consider e.q. 1.1

$$O_i = \begin{cases} S^{T_1} & \text{search space for environment at } T_1 \\ S^{T_2} & \text{search space for environment at } T_2 \end{cases}$$

(1.1)

This equation represents the scenario where the optimal solution at one state of the environment is located in a different search space compared to the optimal solution at a different state of the environment. In terms of a changing-objective optimization problem a state change in the environment could mean the addition of another dimension in the search space. In terms of game playing agents we mean that an agent that has learned to play a certain game, is able to learn a new game without any human intervention.

Consider the example of an agent A, capable of playing a game, say checkers. We could represent this player as (1.2)

$$A^T_1 = \{ s_1^{\text{checkers}}, s_2^{\text{checkers}}, s_3^{\text{checkers}} \}$$

(1.2)
Here $A_1^T$ represents the strategy employed by the game playing agent A at the $T_1$ state of the environment. $s_1^{\text{checkers}}$ is a particular feature of the strategy employed by A. $s_1^{\text{checkers}}, s_2^{\text{checkers}}, s_3^{\text{checkers}}$ combine to represent the strategy A uses to play checkers. Other individuals can be represented in a similar manner.

At time $T_2$ the environment for this agent changes from playing checkers to playing chess. This is a substantial change within the environment. All strategies learned by A (to play checkers) will fail in this new environment because the rules of the two games are different from one another. Traditional learning approaches to this problem will force A to move towards its previous optimum oblivious to the fact that the optimum is now obsolete. This failure is the result of agent A’s ignorance of its environment. In this manner Blondie24, and Chinook, Deep Blue, are static end products with no adaptability. Hence their intelligence is questionable as they cannot learn from a real world environment. Human intelligence, however, is not a perfect end product. We lack a complete knowledge of our world and acknowledge that with time come changes. As mentioned earlier, an important aspect of human intelligence is our capability to learn new ‘things’ from our environment. Our skill and knowledge about our environment evolves with exposure to our surroundings. Human intelligence enables us to adapt to every change in our environment.

In this thesis we investigate an approach that overcomes the limitation of the learning process, taking another step towards the fulfillment of our quest for more human like behavior in machines. Our learning approach realizes the imperfection of its surrounding environment. Agents based on this approach acknowledge the fact that there environment can change to a new state at any time. Any new state of the environment may be slightly different from its current state or it may be completely unrelated to the previous states. The agent must then modify its current set of strategies to ensure survival in the altered environmental state.
1.3 Contribution

Main contributions of this dissertation are:

- We present a new perspective on continuity of learning, which we define as a set of three fundamental aspects of learning; Exploration, Evolution and Experience.
- A fully automated unsupervised learning framework has been developed, that enables simulated agents to adapt according to an unknown environment.
- Agents can learn to explore and exploit their surrounding environment on their own. They acquire new skills and update their knowledge space without relying on any human intervention.
- Our framework allows agents to learn and adapt as per the requirements of their environment (these requirements may change over time without any warning).
- Using Computational Intelligence approaches (a Particle Swarm Optimization (PSO) variant & Artificial Neural Network (ANN)) the framework detects change. Based on this change, new information can be added and obsolete information can be removed from the current knowledge space.
- Our approach focuses on building a dynamic relationship with its environment. This allows it to cater for changes and adjust self behavior when the unobservable becomes observable (or the unseen becomes seen meaning when new information is available).
- The presented framework trains agents depending solely on the information provided by the environment and makes no assumptions about the task or the surrounding environment.
- Agents learn multiple objectives (or skills) without any a priori information about those objectives (or skills).

1.4 Thesis Organization

The rest of the thesis is organized as follow.
Chapter 2 provides an overview of Evolutionary Learning (EL) techniques. It describes the problem that this thesis focuses on. It presents a comprehensive literature survey on AI research, machine learning techniques, and evolutionary approaches. This chapter also highlights computer game playing (which is a promising research domain for machine learning techniques) by giving an in-depth analysis and survey of evolutionary game playing systems. At the end of the chapter the limitations of current techniques are discussed.

Chapter 3 discusses the idea of imperfection in an evolutionary system or an Imperfect Evolutionary Systems (IES). IES were introduced by Kendall and Su in [16]. Based on their work, we will first define what an IES is and how it is different from traditional learning environments. This is followed by our definition of how imperfection can exist in a system and the relationship of the agents residing in an IES with their environment. We then discuss the mechanisms necessary to incorporate change in a system and how this information can be disseminated to the agents residing within it. A major focus of this chapter is to explore and elaborate different parameters of the environment and how imperfection affects them. This chapter will serve as a platform for further discussion about incompleteness in an evolutionary system.

Chapter 4 introduces the notion of continuity of learning in an imperfect and incomplete evolutionary environment. We discuss the underlying processes that form the basis of this continuous learning. We present a framework that employs these processes to enable automated skill learning of agents. The continuity of learning enables the agents to learn new abilities on their own without human intervention. This automated learning allows the agents to learn new abilities based on the information present in their environment and their abilities are only limited by the information available to them. This dynamic relationship enables agents to constantly monitor their surroundings for new information. Whenever new information is added to the environment, individuals then have the ability to decide whether to use or ignore this new information.

Chapter 5 presents an imperfect game like environment which we have developed to test the learning capabilities of the framework presented in the previous chapter. This
environment is primarily a computer game inspired test-bed and the design of the simulated agents used in this environment is deliberately kept abstract. This chapter details the different kinds of entities present in the environment and the challenges their surrounding world presents to them. This environment supports a multitude of species and for each species different objectives can be assigned. Due to the incomplete nature of the environment, anyone of its components can be changed at any time. Agents residing in the environment have the ability to detect different kinds of information present around them and based on this information formulate new strategies to deal with the demands of the environment.

Chapter 6 presents the experiments carried out in this environment to test the learning abilities of our framework. There is no information sharing between the environment and the framework other than the fitness evaluation of the strategies. All decisions are made by the framework to evolve a strategy that is able to handle any change in the environment. We first present the learning algorithm we used based on the continuous learning framework, then we present the environmental parameters used in testing and later we present the results of our experimentation. Different experiments are conducted on the framework to evaluate the performance of the framework and its behavior on the presentation of new dimensions to the learning agents during their evolutionary process.

Chapter 7 showcases an application of this research. Although this research can be applied to a wide array of problems and environments, in this chapter, as an example, we present how our research can be applied to the quandary of predictability in video games. We present an evolutionary approach based on our continuous framework (presented in chapter 4), that uses a modified particle swarm optimization (PSO) and artificial neural networks (ANN), to answer the dilemma of predictability. Video games usually require intelligent agents to adapt to new challenges and optimize their own utility with limited resources and our approach utilizes adaptive intelligence to improve agent’s game playing strategies. We will detail how our research is directly applicable to video games
research and evolutionary gaming. Our approach can be further extended to develop intelligent systems for exploitation of weaknesses in an evolutionary system.

Chapter 8 is the concluding chapter with a discussion on the significance of studying continuity of learning in imperfect environments. It provides a summary of our research and identifies potential future research directions.
Chapter 2: AI & Evolutionary Learning

A theoretical grounding for the thesis

If I have seen further than others, it is by standing upon the shoulders of giants
- Isaac Newton

2.1 Introduction

This chapter presents an overview of artificial intelligence (AI) research and evolutionary approaches to learning. We first look at aim of AI research and what we hope to achieve with it. This is followed by an overview of current learning methodologies. Sections 2.3 to 2.4 discuss different areas of AI with an emphasis on machine learning, artificial neural networks and computer game playing. Section 2.5 describes co-evolutionary approaches to game playing, providing brief introduction to some notable advancement in this area. We will briefly discuss Blondie24, a computer program which learns to play checkers without pre-injected human knowledge and NERO, a computer game designed with emphasis of training simulated robotic agents.

In the end we present a discussion upon the limitations of all the traditional learning techniques mentioned in the chapter. We highlight on the importance of this research area within AI research. In the discussion we show how current learning methodologies have yet to completely fulfill the definitions of AI mentioned at the start of the chapter and how a new learning paradigm is required to take current AI skills to newer level. Most of the term used in later sections will be defined in this chapter. We being by presenting different definitions of AI found in literature.
2.2 Definitions of Artificial Intelligence

The term Artificial Intelligence (AI) was first used by John MaCarthy in the Dartmouth summer conference in 1956. Depending upon their point of view various definitions of AI have been given by different researchers.

1. The goal of work in AI is to “build machines that perform tasks normally requiring human intelligence” [17].
2. Focus of AI research is to “try to get machines to exhibit behavior that we call intelligent behavior when we observe it in human beings” [18].
3. Major concern of AI is “studying the structure of information and the structure of problem solving processes independently of applications and independently of its realization in animals or humans” [19].
4. AI is the “use of computer programs and techniques to cast light on the principles of intelligence in general and human thought in particular” [20].
5. AI is the “automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning, etc” [21].
6. “The study of how to make computers to do things at which, at the moment, people are better” [22].
7. “The study of the computations that makes it possible to perceive reason and act” [23].
8. “Study of the design of intelligent agents” [1]. They further define an intelligent agent “is a system that acts intelligently” i.e. it reacts according to its surroundings and its needs.
9. Artificial intelligence research is “the scientific understanding of the mechanisms underlying thought and intelligent behavior and their embodiment in machines” [24].
10. “AI is both the intelligence of machines and the branch of computer science which aims to create it” [25].

Russell and Norvig [2] classify the definitions of AI into four categories:

i. Systems that think like humans
ii. Systems that act like humans
iii. Systems that think rationally
iv. Systems that act rationally

Although many AI techniques take their motivation from naturally occurring intelligent behavior, some vary from the way human intelligence works. However, the ultimate goal of AI research is the reproduction of human-like intelligence in machines and computer systems. As Luger comments in his book [26]:

“…we entered the field of AI ... to understand and explore the mechanisms of mind that enable intelligent thought and action ...”

In this thesis we agree more with the definition of AI given by [24]. We will be focusing on the replication and mimicking of human intelligence in machines. It should stand that these processes have not been clearly understood in humans and their reproduction in machines is at times abstract.

2.3 Disciplines of AI

Artificial intelligence is a diverse field where researchers address a wide range of problems that relate to intelligent behavior from machines. A logical question follows, *how do we decide if a machine is intelligent or not?*

This question was addressed in the early years of AI by Alan Turing in his well-known Turing Test [27]. The Turing test worked like this after posing some written questions to a human and a computer, a human interrogator tried to guess whether the reply was from the human or computer. In order to pass the Turing test, the machine has to first understand a question asked by an interrogator and then have the ability to provide an appropriate reply of that question. If the machine passes the Turing Test, it is considered to be possessing intelligence. We now move on to present a few disciplines commonly found in the field of AI.
2.3.1 Computer Vision

Extraction of useful information from a perceived image is dealt with in the field of computer vision (CV). This research area focuses on the machines visual sensory input and visual perception of its surroundings. Information extracted from the image can be used for recognition, manipulation and navigation [28]. An early attempt to understand the computational requirement for human visual system was presented by [29]. Forsyth and Ponce [28] provide extensive coverage on algorithms and techniques for image formation and processing. Computer vision can be thought of as the inverse of computer graphics. Computer graphics produces image data from 3D models while computer vision often produces 3D models from image data. Most Researchers further divide computer vision into the sub-domains of

- Scene Reconstruction
- Object Recognition
- Image Restoration
- Event Detection
- Video Tracking

2.3.2 Knowledge Representation

“A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.” [30]. KR is concerned with how to formally "think", that is, how to use a symbol system to represent "a domain of discourse". It follows that this symbol (or representation) is such that it may or may not be within the domain of discourse that allow inference (formalized reasoning) about the objects within the domain of discourse to occur. This symbol should hold its own meaning and can be described and talked about. An important application of KR techniques is Expert Systems [31, 32] and Deduction and Reasoning for problem solving.
**Expert Systems**

An Expert System (ES) is software that attempts to provide an answer to a problem, (and/or clarify uncertainties) where normally human experts would need to be consulted. Expert systems target a specific problem domain. Expert systems were introduced by researchers in the Stanford Heuristic Programming Project, Edward Feigenbaum, PI, with the Dendral and Mycin systems. The term expert system is reserved for programs whose knowledge base contains the knowledge used by human experts and often used as a synonym for knowledge-based systems (KBS). They are one of the most widespread type of AI application [33].

**Deduction, Reasoning and Problem Solving**

Deductive reasoning uses arguments to move from given statements known as premise, which are assumed to be true, to conclusions, which must be true if the premises are true [34 ,35]. Earlier algorithms imitated the step-by-step reasoning behavior commonly employed by human beings for logical deductions. This later developed into highly successful methods for dealing with uncertain or incomplete information, employing concepts from probability and economics [1], [6], [17], [25]. Reasoning problems usually start from a given initial state and try to reach some target goal state. The efficiency of such a system depends upon the number of transitions from one state to next till end. Minimizing the number of intermediate states improves the efficiency of the overall reasoning system. To cope with this, Brachman and Levesque [36], propose automated reasoning procedures.

**2.3.3 Natural Language Processing**

Natural Language Processing (NLP) deals with the ability of machines to process and understand a human language. NLP research is the study of building systems that are able to process and manipulate natural language used by human beings, such as speech recognition, natural language understanding, information retrieval and extraction.
etc. [37]. Allen [38] provides a detailed overview of language processing from an AI perspective. Brill and Mooney [39], give a summary of empirical NLP studies. The aim is to develop an NKP system able to acquire knowledge on its own. Such a system would be able to extract this knowledge from currently available resources like for example the internet. This information extraction process is commonly referred to as text mining. Machine translation and text mining are some of the applications of NLP [34].

2.3.4 Automated Planning and Scheduling

Intelligence requires that agents must be able to set and achieve goals [35]. Such systems must be having mechanisms to represent their world, using which they can make future predictions, plans and choices which maximize their utility [25]. Automated planning and scheduling deals with strategy formulation and action planning for execution by intelligent agents, autonomous robots and unmanned vehicles. These problems often require discovering complex solutions and optimization of strategies. For perfect (or completely known) environments planning and training can be performed offline (i.e. solutions can be found and evaluated prior to actual execution). For incomplete environments strategies require refinement and revision online. Usually this requires an iterative approach based on trial and error policies.

This thesis presents an automated framework that deals with such complex and multi-dimensional strategy discovery problems.

2.3.5 Robotics

Robotics is an amalgamation of electronics, mechanics and AI software. In respect to AI, it makes use of natural language processing, computer vision, knowledge representation and machine learning. Earlier approaches to robotics relied on representation of the surrounding environments based on the sensory inputs of the robot. Using these representations actions sequences were modeled via reasoning and planning. However complexity of the real world models cause computational bottlenecks while noisy sensors present another challenge. For this reason alternative approaches for formulating robot intelligence were sort after. Jones and Flynn presented such an
approach called Subsumption architecture [40]. It assumes that the resources for the modeling of an environment are limited or nonexistent. A robot is organized by task achieving behaviors layers, where intelligent behavior emerges from the interaction of simple rules of behavior in a complex environment. This approach is also known as the behavior-based robotics. A general coverage on robotics is presented by Murphy [41].

2.4 Machine Learning

Machine Learning aims to improve a machine’s performance (or learning) a task through experience [42]. A major focus of machine learning research is pattern recognition and making intelligent decisions based on data. Some machine learning systems attempt to eliminate the need for human intuition in data analysis, while others adopt a collaborative approach between human and machine. Machine learning algorithms are generally organized on the basis of the desired outcome of the algorithm.

1. **Supervised Learning** tries to infer a function from training data. It usually generates a function that maps inputs to desired outputs. The training data consist of pairs of input objects and desired outputs. If the output of the function is a continuous value it is called regression. If the output is a prediction of the class of input object it is called classification.

2. **Unsupervised Learning** seeks to determine the organization of data by clustering similar data.

3. **Semi-Supervised Learning** combines both supervised and unsupervised data examples to generate a suitable classifier.

4. **Reinforcement Learning** learns to react according to observations made from surroundings.

5. **Transduction** is learning to predict new outputs based on available training and testing data.

6. **Learning to Learn** uses previous experiences to learn its inductive bias.

Common machine learning techniques in AI include

1. Decision Tree Learning
2. Instance-Based Learning
3. Bayesian Learning
4. Heuristic Learning
5. Reinforcement Learning
6. Genetic Learning

2.4.1 Decision tree learning

This technique builds decision trees that are used in the decision making process. To build a decision tree, the algorithm selects an attribute to be tested at each level of the tree. The ID3 algorithm [42] is a classical learning algorithm. Based on given attributes, it performs a complete hypothesis of the search space. It then attempts to build a decision tree by means of a top-down, greedy search as shown in Table 2.1, taken from [43].

![Decision Tree for the concept of Play Tennis](image1)

Figure 2.1 Decision Tree for the concept of *Play Tennis*

Figure 2.1 illustrates an example of decision trees. Using this decision tree we can test different conditions if they are valid for playing tennis or not, e.g., if the outlook is rainy but the rain is weak we can play tennis.
Practical issues in decision tree learning include determining how deeply to grow a decision tree, handling continuous attributes, choosing an appropriate attribute selection measure, and handling training data with missing attribute values. Quinlan [42] developed a C4.5 algorithm, which extends from ID3 using post-pruning to find high accuracy hypotheses trees. Other works related to decision tree learning are [44 - 48].

### Table 2.1 An ID3 algorithm for decision tree learning

<table>
<thead>
<tr>
<th>ID3 (Examples, Target_attribute, Attributes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Examples are the training examples.}</td>
</tr>
<tr>
<td>Target_attribute is the attribute whose value is to be predicted by the tree.</td>
</tr>
<tr>
<td>Attributes is a list other attributes that may be tested by the learned decision tree.</td>
</tr>
<tr>
<td>The algorithm returns a decision tree that correctly classifies the given examples.</td>
</tr>
</tbody>
</table>

Create a Root node for the tree
If all Examples are positive, return the single-node tree Root with label = “+”
If all Examples are negative, return the single-node tree Root with label = “-”
If Attributes is empty, return the single-node tree Root, with label = “most common value of Target_attribute in Examples”.

Otherwise begin

A ← the attribute from Attributes that best classifies Examples, which has the highest information gain

The decision attribute for Root ← A

For each possible value, vᵢ, of A,

Add a new tree branch below Root

Let Examplesᵥᵢ be the subset of Examples that have value vᵢ for A

If Examplesᵥᵢ is empty

Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples

Else below this new branch add the subtree

ID3(Examplesᵥᵢ, Target_attributes, Attributes-{A})

End

Return Root

### 2.4.2 Instance based learning

Instance-based learning approaches, instead of performing explicit generalization, compare new problem instances with training samples stored in the memory. These approaches store the past-learned instances in memory. Upon encountering a new
instance the set of related instances are retrieved from memory and are used to classify the new query. An example of such approaches is the K-Nearest Neighbor algorithm for instance-based learning [49]. Instead of trying to build global function for evaluation of possible instances, these methods focus on related past instances. These similar instances are used to build approximated local functions for the new instances. A good discussion on instance based learning is provided by Aha et al. [50]. Daelemans and Van den Bosch describe variations of K-Nearest Neighbor algorithm for use in natural language processing (NLP). They claim that memory-based learning is both more psychologically realistic than other machine-learning schemes and practically effective [51].

2.4.3 Case-based Reasoning (CBR)

CBR is a type of instance-based learning method. In CBR, past examples are analyzed to classify new instances faced by the system. CBR has been formalized as a four-step process by [52]:

1. **Retrieve**: For any given problem, retrieve cases from memory which are relevant to solving it.
2. **Reuse**: Mapping of previously found solutions to the target problem. It may involve adapting the solution to fit the new situation.
3. **Revise**: Test the new solution (derived from Reuse process) in the real world (or a simulation) and revise as necessary.
4. **Retain**: Store successfully adapted solutions as a new case in memory.

A CBR system for course timetabling problems was presented by Burke et al. [53] which represented instances as attribute graphs. Aamodt and Plazas [52] provide a survey on case-based reasoning. One of the earliest successful CBR systems was Lockheed's CLAVIER [54], a system designed for laying out composite parts for baking in convection ovens. An extensive examination of CBR issues is presented by [55]. Roots of CBR systems can be traced to the work of Roger Schank and his students at Yale University in the early 1980s. Schank's model of dynamic memory [56] was the basis for
the earliest CBR systems: Janet Kolodner's CYRUS [57] and Michael Lebowitz's IPP [58]. Examples of CBR systems are [59 – 65].

2.4.4 Bayesian learning

Bayesian learning uses observations are to infer new information (or update) what is known about underlying parameters. Bayesian learning deals with Bayesian reasoning (probabilistic approach for inference reasoning under uncertainties) and Bayesian networks (probabilistic representations of the world based on dependences of a set of variables). Figure 2.2 gives an example of Bayesian network (taken from [43]).

![Bayesian network diagram]

Figure 2.2 A Bayesian network.

Bayesian learning deals with the issue of how to build a Bayesian network. The network structure may wither be known in advance or inferred from training data. Network variables may be completely or partially observable. Some recent books on the subject are [66 – 77].
2.4.5 Heuristic Learning

Reeves [78] describe heuristic methods as search techniques that find good (i.e. near-optimal) solutions at a reasonable computational cost. Heuristics are used to develop solutions that are hoped to be close to an ‘optimal’ (or best) solution. A precise description of heuristics is given by [79] as strategies using readily accessible, though loosely applicable, information to control problem solving in human beings and machines. Arguably the most common heuristic is the trial and error method. Other more complex heuristics are Tabu Search (a local search method which starts from random solutions, and continues to search in its neighbors for better solutions) [80, 81], Simulated Annealing (a local neighborhood search algorithm based on the metallurgical annealing process) [82, 83] etc.

Hyper-Heuristics

A hyper-heuristic seeks to automate, the process of selecting and applying heuristics to solve a computational search problem. Occasionally it requires combining several simpler heuristics (or their components). Hyper-heuristics aim to build systems which can handle classes of problems rather than solving just one problem [84 - 86]. As each heuristic has its own strength and weakness, multiple heuristics can be applied to a problem. The central idea is to automate algorithm development by combining strengths of different algorithms and compensating for their weaknesses [87]. Typically hyper-heuristic frameworks have a high-level methodology and a set of low-level heuristics. Typically a high-level method it used for selection of which low-level heuristics are to be applied at any given time.

On-line learning hyper-heuristics

In on-line learning hyper-heuristics, learning takes place while solving an instance of a problem. This enables the high-level strategy to determine the appropriate low-level heuristic to apply by using the local properties of the problem.
Off-line learning hyper-heuristics

In off-line learning hyper-heuristics, the idea is to gather knowledge in form of rules or programs, from a set of training instances. This knowledge can then be used to generalize the process of solving unseen instances.

Meta-heuristic

Meta-heuristic are heuristics that optimize a problem by iteratively trying to improve a candidate solution, considering a given measure of quality. These techniques make few or no assumptions about the problem being optimized and are designed to search very large spaces of candidate solutions. However, they do not guarantee an optimal solution will ever be found. Examples of meta-heuristics are [88, 89].

2.4.6 Genetic learning

GENETIC ALGORITHMS

Genetic Algorithms are global search heuristic and are used to find exact or approximate solutions to optimization and search problems. Genetic algorithms are population-based evolutionary approach and emulate the evolutionary processes occurring in nature. Using genetic operators such are crossover and mutation; they evolve a randomly generated initial population. The evolutionary process is based upon natural selection [90 – 93]. A generic approach to genetic algorithms is shown in Table 2.2 (abridged from [93]):

<table>
<thead>
<tr>
<th>Table 2.2 A Genetic Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>Fitness Function:</strong> The problem to be solved is defined in terms of an objective function known as fitness function which indicates the fitness of a potential solution.</td>
</tr>
<tr>
<td>2. <strong>Representation:</strong> A population of candidate solutions is initialized (usually this is random). A candidate solution may be represented using a binary string (called chromosome).</td>
</tr>
<tr>
<td>3. <strong>Evaluation:</strong> Each chromosome in the population is assigned a fitness score based on its evaluation.</td>
</tr>
<tr>
<td>4. <strong>Selection:</strong> A selection scheme is applied to the population, so that fitter solutions are selected as parents to generate new solutions.</td>
</tr>
</tbody>
</table>
5. **Reproduction:** Selected chromosomes combine to generate new off-springs via the use of specific genetic operators (e.g. crossover and mutation)

6. **Elimination:** From each generation poorly performing solutions are purged and a new population (based on best solutions) is produced for the next iteration in the evolution.

7. **Termination:** The process is terminated if the desired quality of solution is achieved or the computing time expires. Otherwise the process repeats itself.

Different values of the genetic operators have different effects on the performance of the overall algorithm. In [94], Yao presents an overview of the impact of these operators on the performance of GAs. Fogel [93] provides discussions on the theory of convergence for GAs. Application of GAs include function approximation [95] and training artificial neural networks [96, 97], high-dimensional optimization problems, [98], control system design [99], scheduling problems [100] etc. Other examples of GAs application can be found in [92,100-108]. Many variants of the standard GA have been developed by researchers e.g. Clustering-based Adaptive Genetic Algorithm (CAGA) [109], Adaptive genetic algorithm [110], etc.

**Genetic Programming**

Evolutionary or Genetic programming (GP) is a population-based probabilistic searching algorithm. GP is based on GA and focuses on the evolution of computer programs that perform a user defined task. In GP, a population of computer programs is optimized according to their fitness values. Fitness of a program is determined by its ability to meet with user specified criteria. Evolutionary Strategies (ES) [111 -113] were developed for numerical optimization problems. Two major selection schemes in evolutionary strategies [114] are

- **(λ + μ) - ES:** μ parents are used to create λ offspring. The entire population competes and the best solutions are selected as μ for next generation.
- **(λ, μ) - ES:** μ parents are used to create λ offspring, and only the λ offspring compete for survival.

Some works concerning evolutionary programming are [93, 114-117].
2.4.7 Reinforcement Learning

Reinforcement Learning (RL) deals with the ability of how an agent should behave in order to improve its survival chances (or long term fitness value). Sutton and Barto [118], define RL as a learning approach in which every agent tries to maximize its own reward. Reward is calculated on the basis of interactions between the agent and its environment. Figure 2.3 (adapted from [119]) illustrates this concept.

In the figure $a_i$ is an action while $s_i$ represents the state. Reward values are presented by $r_i$. Each time the agent performs an action in some state, it receives a reward. This produces a sequence of states $s_i$, action $a_i$ and immediate rewards $r_i$. The task of the agents is to learn a strategy that maximizes the expected sum of these rewards (denoted by $V$).

**Q Learning**

Q Learning is a reinforcement learning algorithm, that does not need a model of its environment making it very suited for repeated games against an unknown opponent. It works by estimating the values of state-action pairs. These values are defined to be the expected discounted sum of future payoffs obtained by taking an action from a particular state and following an optimal policy thereafter. After learning these Q values, the optimal action from any state is to choose the highest one. Q-functions are
represented in a tabular form with each input value followed by an output value. With an increase in state space, convergence time also increases rapidly for Q learning. To deal with this, function approximation was proposed by Russell and Norvig [2]. A detailed discussion on RL and along with its applications is presented in [118, 120].

### 2.4.8 Cultural Algorithms

In addition to population, Cultural Algorithms (CA) has a knowledge component called Belief Space. Based on the diversity of knowledge among the population, belief space of a cultural algorithm is divided into distinct categories. Each category represents different domains of knowledge that the population has of the search space. The belief space is updated every iteration by the best individuals of the population. Performance of each individual is measured using a fitness function. Using this performance, out of the whole population the best individual is selected. This is similar to other evolutionary models e.g. genetic algorithms. In this way cultural algorithms are computation models that imitate cultural evolutionary processes [121 – 123].

A variety of evolutionary paradigms can be used to model the population component such as genetic algorithms for optimization problems [124], genetic programming for evolving agent strategies [125], evolutionary programming for real valued function optimization [126, 127], Multi-agent based models [128, 129] and particle swarms [130]. Cultural algorithms have also been applied to Function Optimization problems in dynamic environments [131 – 133].

### 2.5 Artificial Neural Networks

Although a precise definition of artificial neural networks is lacking, most scientists are in agreement that they comprise of series of neurons (inter connected processing units). On the basis of their interconnections these neurons exhibit complex global behaviors. Artificial Neural Networks (ANNs) were inspired by the study of biological neural systems. ANNs consist of interconnected processing nodes (called
neurons) arranged in a network form. ANNs react to external stimulus provided by their environment. ANNs are mathematical (or computational) models based on biological neural networks. Each ANN is composed of interconnected artificial neurons arranged into layers (usually input, hidden and output). ANNs are adaptive in nature changing their structure or interconnection values based on the information they receive from their environment during their training (or learning) phase. In this section, we look at artificial neural networks with regards to network architectures, learning methods, and in particular, evolutionary artificial neural networks.

2.5.1 Learning Paradigms in ANNs

There are essentially three kinds of learning paradigms

**Supervised Learning**: In this kind of learning, the function is deduced from the training data. Training data is comprised of inputs and desired outputs. The objective of supervised learning is to predict the value of a function based on an input. This prediction is done on the basis of learning from training data i.e. paring of inputs with output values in the training data. To accomplish this, the learner generalizes information deduced from presented data, applying it to unseen situations. If \((x, y)\) are the learning pairs such that \(x \in X\) and \(y \in Y\), then the objective of supervised learning is to find \(f\) such that

\[
f : X \rightarrow Y
\]

**Unsupervised Learning**: This class of problems seeks to determine the organization and correlation present amongst data. To determine this organization data is clustered on the basis of common attributes. Contrary to supervised learning the learner is given only unlabeled examples.

**Reinforcement Learning**: In this kind of learning the data is not given but is generated by the interactions of learning agents with their environment. For more details see section 2.3.6.6
2.5.2 Perceptrons

In 1959 Rosenblatt [134], presented one of the earliest artificial neural network models which was developed to classify patterns through supervised learning. The basic processing unit of this model is called Perceptron (figure 2.4). To produce a required output based on the input, the network is trained to automatically tune the weights of its processing units (or perceptrons).

In perceptrons a processing unit receives a set of inputs $a_1, ..., a_n$ where $a_0$ is a special input known as bias and is always fixed at value of $+1$. Each connection between an input $i$ and the processing unit $j$ has an associated weight $f$ is called as activation function, which scales the output from the processing unit into a certain range. A weighted sum of all inputs is performed by each processing unit, as shown in equation 2.1.

$$S_j = \sum_{i=0}^{n} w_{ji} a_i$$  \hspace{1cm} (2.1)

A weighted sum of all inputs is performed by the processing units and the output is scaled within a certain range using the activation function (also called step function) $f$, using equation 2.2

$$X_j = f(S_j): \begin{cases} 0 & \text{if } S_j \leq 0 \\ 1 & \text{if } S_j > 1 \end{cases}$$  \hspace{1cm} (2.2)
The two-layered perceptron (Figure 2.5) can be trained for function approximation problems by using a simple learning rule which adjusts network weights on the basis of difference between desired output (from training data) and current network output (as shown in equation 2.3). Usually the training data set is repetitively presented to the network until the desired output is achieved.

$$w_{ji}^{\text{new}} = w_{ji}^{\text{old}} + c (t_{j} - x_{j})a_{i}$$  \hspace{1cm} (2.3)

The old weight is represented by $w_{ji}^{\text{old}}$, while $w_{ji}^{\text{new}}$ is the updated weight, the learning rate $c$ controls the scale of the updates, and thus the speed of learning. $t_{j}$ is the target output, $x_{j}$ is the output from the network and $a_{i}$ represents the input.

The two-layer perceptrons can be successfully trained for solving a number of function approximation and pattern classification problems. The convergence of this perception learning rule was proved by [135], however, due to its limited representation capabilities two layered perceptron cannot learn some simple functions (e.g. XOR [136]). Multiple layers and non-linear activation functions were added to simple two-layered perceptrons to overcome these limitations. The new Multi-layer Perceptron (MLP) was fully connected feedforward (a processing unit is connected to all neurons in the previous layer and signal flows through the network in a forward direction) architecture (Figure 2.6). Some non-linear activation functions commonly used in MLP are given in Table 2.3.
Table 2.3 Some commonly used non-linear activation functions in multi-layer perceptrons

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step Function</td>
<td>Step(x) = 1 if $x \geq 0$, else 0</td>
</tr>
<tr>
<td>Sign Function</td>
<td>Sign(x) = +1 if $x \geq 0$, else -1</td>
</tr>
<tr>
<td>Sigmoid Function</td>
<td>Sigmoid(x) = $1 / (1 + e^{-x})$</td>
</tr>
<tr>
<td>Hyperbolic Function</td>
<td>Tanh(x) = $(e^x - e^{-x}) / (e^x + e^{-x})$</td>
</tr>
</tbody>
</table>
2.5.3 Backpropagation Learning

Backpropagation is a training algorithm for multi layer perceptrons developed by Parker [137] and Rumelhart et al. [138]. Given a fixed network the algorithm tries to learn the best suited connection weights of a multi-layer perceptron for a given problem. It is essentially a gradient descent method that aims to minimize the error between actual output and desired output. Fausett [139] and Russell and Norvig [2] present a mathematical analysis of the backpropagation algorithm.

2.5.4 Recurrent Networks

Recurrent neural networks (RNNs) are neural networks that possess at least one feedback connection. This is contrary to feedforward networks where a signal only flows through the network in a forward (or linear) direction. RNNs have bi-directional data flow meaning RNNs also propagate data from later processing stages to earlier stages or in other words some signals in a RNN flow in a backward direction. This is in contrast to the feedforward architecture which only propagates data from input to output in a linear fashion. There are different kinds of recurrent networks, some of them are discussed below.

Simple Recurrent Network

This is a special case of the basic architecture and was employed by Jeff Elman [140] and Michael Jordan. In SPN, due to the addition of context units in input layer, a three-layer network is used. These context units can be connected with the hidden layer or output layer. At each time step, input is propagated in a linear feed-forward after which a backpropagation-like learning rule is applied. Due to the fixed back connections the context units always maintain a copy of the previous values of the hidden units. Figure 2.7 shows a Simple Recurrent Network (SPN) [140].
Hopfield networks

Hopfield networks [141] are recurrent neural networks in which all connections are symmetric, i.e., if there is a connection from unit x to y, then there must be a connection from unit y feedback to unit x.

Radial basis function (RBF) networks [142], probabilistic neural networks (PNN) [143], and Kohonen self-organizing maps (SOM) [144] are examples of some other popular neural networks. Further studies on neural networks can be found in [139,145-146].

2.5.5 Issues with Backpropogation

Despite of their successful application to practical problems like financial time series predictions [147], computer game playing [148], and industrial applications [149], backpropagation training on multi-layer perceptrons has a few drawbacks [150].

- As backpropagation algorithm is only applied to a fixed network it demands the network architecture to be known in advance. There are no globally accepted set of rules that specify the application of a particular architecture for a problem. ANNs are usually developed on the basis of experience using detailed experimentation.
• Being a gradient decent method, Backpropagation tends to become trapped in a local minimum. If the error function is multimodal and/or in-differentiable then this method may never find the global minimum.

Keeping these issues in view, evolutionary approaches for learning ANNs have been used in attempts to solve these problems. We now present some work on this topic.

2.5.6 Evolutionary ANNs

Moriarty and Mikkulainen [151] and Yao [114] present a comprehensive survey on evolutionary artificial neural networks. Evolutionary approach can be introduced into ANN at three levels [114].

Connections Weights

Evolving connection weights using an evolutionary algorithm, overcomes the problem of being trapped into local minima by providing a global search method for network weight training. The evolutionary approach to weight training in ANN’s requires two major decisions.

a. Representation of the connection weights.
b. Genetic operators to be used for the evolutionary process.

Table 2.4 describes the typical cycle of the evolution of connection weights, taken from [114]. Binary strings were used by Caudell and Dolan [152] and Garis [153] in their representation scheme. It was found that representation schemes that create solutions of very large length can lead to decreased performance [154]. Example of applications using evolutionary approaches for evolving connection weights can be found in [155-159].
Table 2.4 A typical cycle for evolving connection weights in EANN’s.

1. Decode each individual (a chromosome represents all connection weights) in the current generation into a set of connection weights and construct a corresponding ANN with the weights.
2. Evaluate each ANN by computing its total mean square error between actual and target outputs. Other error functions can also be used and problem-dependent. The higher the error, the lower the fitness. A regularization term may be included in the fitness function to penalize large weights.
3. Select parents with higher fitness for reproduction.
4. Apply search operators, such as crossover and/or mutation, to parents to generate offspring, which form the next generation of potential connection weights.

Network Architecture

Evolving network architectures without human intervention provides an automated approach to ANN design evolving both connection weights and structures. Evolving the architecture of an ANN (i.e. connectivity, and the activation function of neurons) is similar to searching through a space of all possible network architectures. The search ends when a suitable ANN is found. A typical cycle for network evolution is shown in Table 2.5 (adapted from [114]).

Table 2.5 A typical cycle for evolving network architectures in EANN’s.

1. Each hypothesis of network architecture in the current generation is encoded into chromosomes for genetic operations, by means of a direct encoding scheme or an indirect encoding scheme.
2. Evaluation of fitness. Decode each individual in the current generation into an architecture, and build the corresponding ANNs with different sets of random initial connection weights. Train the ANNs with a predefined learning rule, such as the Backpropagation algorithm. Compute the fitness of each individual (encoded architecture) according to the training results, for example, mean-square-error, and
other performance criteria such as the complexity of the architecture, e.g., less number of nodes and connections preferred.

3. Select parents from the population based on their fitness.
4. Apply search operators to the parents and generate offspring, which form the next generation.

A number of studies have been conducted on evolving architectures and connection weights simultaneously. EPNet an evolutionary system, described by Yao and Liu [160,161], uses evolutionary programming for the simultaneous evolution of ANN architectures and connection weights. The main structure of EPNet is shown in figure 2.8 (taken from [160]).

![Figure 2.8 EPNet for concurrent network architecture and connection evolution](image)

In Liu and Yao [162], EPNet randomly adds or deletes sigmoid and gaussian nodes in the current network architecture. EPNet has been successfully
applied to a number of practical problems [161,163]. Other applications of evolving ANN architecture and connection weights simultaneously can be found in [164 – 168].

**Learning Rules**

The evolution of learning rules is considered as the process of “learning how to learn”. In ANN’s evolution can be used for the adaptation of learning rules or weight training. Aside from the perceptron learning rule for SLP and backpropagation algorithm for MLPs other types of learning rules for different type of ANN’s also exist, such as the Hebbian learning rule [139]. In fact, we can assume any learning rules to be in a more general form as follows [92]:

\[ W_{ji}(t + 1) = W_{ji}(t) + \Delta W_{ji} \]  

(2.4)

Where

\[ \Delta W_{ji} = f(a_i, o_j, t_j, w_{ji}) \]  

(2.5)

Here \( a_i \) is the input to the unit \( i \), \( o_j \) is the output from the unit \( j \), \( t_j \) is the targeted output from the unit \( j \), \( w_{ji} \) is the current weight on the connection from \( i \) to \( j \). More examples of evolving learning rules can be found in [169, 170].

**2.6 Particle Swarm Optimization**

Particle Swarm Optimization (PSO) originates from the study on swarm intelligence in nature. Eberhart and Kennedy [171] developed PSO based on their observations of flock of birds. The fundamental concept behind PSO is that individuals in a swarm exchange their experiences while randomly moving in the search space. PSO algorithm maintains a population of potential solutions and in this regard is similar to other evolutionary approaches. Potential solutions are called particles and these are flown through the problem hyperspace. A velocity parameter controls the movement of these particles. This velocity is stochastically accelerated towards it’s the previous best position
found by the particle and the global best position found by the entire population. Equation 2.6 defines this movement.

\[ x_i(t + 1) = x_i(t) + v_i(t + 1) \quad (2.6) \]

where

\[ v_i(t + 1) = wv_i(t) + c_1r_1(t)(y_i(t) - x_i(t)) + c_2r_2(t)(\hat{y}_i(t) - x_i(t)) \quad (2.7) \]

In equation 2.7, \( x_i \) represents the current position of particle \( x \) in dimension \( i \), \( v_i \) represents current velocity, \( y_i \) is the personal best position of the particle and \( \hat{y}_i \) represents that sub-swarm’s global best (called neighborhood best) position to which the particle being updated belongs. The constants \( c_1, c_2, r_1, \) and \( r_2 \) are used to control the area in which search is to be conducted and \( w \) is inertia weight. PSO is more suitable for optimization problems, e.g., evolving connection weights for artificial neural networks [172]. Yoshida et al. [173] described a modified version of the continuous PSO algorithm, which was able to handle both discrete and continuous variables. Recently PSO have been applied to multitude of other areas e.g. surveillance [174], and biochemistry study [175].

### 2.7 Computer Game Playing

Computational intelligence techniques, due to their inherent mimicking of human and nature observed phenomenon, have been used for developing agents socializing with each other and acting intelligently to achieve goals [176-178]. These techniques are an ideal candidate for making games “intelligent”. Researchers have noted this fact and are experimenting and exploring this field [179]. Video games are ideal test-beds for AI research because game playing generally requires intelligent behavior like knowledge representation, automated reasoning and machine learning etc.

Experiments on the game of checkers were conducted using machine learning theory with promising results [180]. In his work Arthur Samuel developed computer program that learnt to play checkers. However, Samuel acknowledged his program’s reliance on human expertise and wrote in his paper:
“... some effort might well be expended in an attempt to get the program to generate its own parameters for the evaluation polynomial.”

This was a challenge left for other to design an AI based program that could learn the game from scratch without human dependence. Chinook [181,182], a checker playing program, won the man-machine world championship in 1994. This was the first time a machine won a world championship. Chinook relied on an opening and closing game database, all it has to do is to find a path from these database that results in a guaranteed win (or a draw). In a landmark match, Deep Blue [183], a chess program developed by IBM, defeated world chess champion Garry Kasparov in 1997. Campbell et al. [184], attribute the victory of Deep Blue to a number of reasons, none of them related to intelligence (pre-inject knowledge, parallel computing). Arguably Deep Blue is more of an accomplishment in computer design as opposed to advancement in artificial intelligence. Other game playing programs shared similar shortcomings. In general, a simple $A^*$ algorithm [185] can be used in one-person games to search for the best move. Whereas “minimax” search algorithm is commonly used for two player games [185-187].

The most severe criticism of traditional knowledge-based approaches for developing game-playing machine intelligence is centered on the large amount of pre-injected human expertise into the computer program and the lack of learning capabilities in these machines [13, 93]. Fogel [13], commented on this phenomenon in computer game-playing as follows:

“... To date, artificial intelligence has focused mainly on creating machines that emulate us. We capture what we already know and inscribed that knowledge in a computer program. We program computers to do things – and they do those things, such as play chess, but they only do what they are programmed to do. They are inherently “brittle”. ... We’ll need computer programs that can teach themselves how to solve problems, perhaps without our help. ...”
Chellapilla and Fogel presented one of the first successful attempts to provide a different approach to game-playing [13,188]. In their work, an artificially intelligent agent was developed to learn to play the game of checkers, the resulting effort referred to as Blondie24 used a co-evolutionary approach based on evolving a population of neural networks (NNs). In many ways (explained in section 2.2.14.1), Blondie 24 was a major step forward in an automated intelligent game playing program.

In 2004, Engelbrecht and Messerschmidt showed how Particle Swarm Optimization (PSO) based algorithm can be used to train ANNs to play the game of Tic-tac-toe [189]. Fryan, in 2005, used evolutionary approach for playing the commercial economy based game, Monopoly [190]. Fryan used evolutionary algorithms so that computer can evolve strategies for playing the games, smartly. Reynolds et al. [191], in 2005, used game theory principles to investigate the migration of Anasazi in the late A.D 1200s. Their goal was to determine the extent to which cooperation and competition need to be present among the agent households in order to produce a population structure and spatial distribution similar to what has been observed archaeologically. Other experiments with evolutionary approach and games include [192-195].

Stanley et al [196], developed Neuroevolving Robotic Operative (NERO) game based on real-time Neuroevolution of Augmenting Topologies (rtNEAT). They evolved complex NNs to train agents in game to perform certain tasks. Agents in game are trained by humans and later have to compete with agents trained by other humans. The abilities of NERO agents are restricted to the tasks for which they have been previously trained for and are unable to learn new tasks on their own.

Kendall et al [16] gave the idea of imperfect evolutionary systems. Imperfect environments act as dynamic environments which open up new opportunities for the agents over time. However the relationship between the agent and its environment, particularly in a gaming environment requires more definition. In the following sections we present three important works on evolutionary game playing and learning to play games.
2.7.1 Blondie24

Blondie24 is a checkers program that learns to play the game of checker to a level that is competitive with human experts [13, 93, 188]. Using evolution of artificial neural network for its training, Blondie24 was able to beat most human checkers players. Blondie 24 used a co-evolutionary approach, meaning it created a population of ANNs and each ANN played against 5 other ANNs selected randomly from this population. Using a minimax alpha-beta search, in each game, an associated game tree with 4 or 6-ply (a ply is a move in a game) was developed. A player scored -2, 0, or +1 points for a loss, draw, or win. A total of 150 games per generation were played (each network played an average of 10 games). After all games were complete, the top 15 networks (based on total points) were progressed as parents for the next generation.

We consider Blondie24 to be a significant advance in computer game playing for the following reasons

1. It learns to play checkers without relying on any pre-injected human knowledge about the game. Therefore it successfully answered Samuel’s challenge on designing programs that would invent their own features and learned to play the game of checkers on its own.

2. By learning to play a game using feedback based on playing against itself, Blondie24 fulfils Newell’s challenge [197].

3. It showed that evolution is an alternative way for the creation of artificial intelligence.

However the Achilles' heel in Blondie24, like deep blue, is the lack of adaptability. Blondie24 is more like an end product created for a perfect environment rather than an intelligent entity capable of adapting itself during its interactions with an incomplete environment. As commented by Eric Harley [198]:

“... An interesting point is that the end product which looks intelligent is Blondie, yet she is not in fact the intelligence. Like the individual wasp, Blondie is fixed in her responses. If she played a million games, she would not be iota smarter. In this sense, she is like
Deep Blue. ... Perhaps a better example of intelligence would be ... a human, who can adapt her behavior to any number of new challenges.”

Blondie24 does not learn during its interaction with its human opponents. Although evolution is the fundamental resource for its intelligence, however, it lacked the ability to update its knowledge incase anything in its environment changed. Other research involving game playing based on an evolutionary process are [27, 116, 199 – 206].

2.7.2 NERO

In contrast to traditional approaches, Stanley et al [196], presented their NeuroEvolution of Augmenting Topologies (NEAT) algorithm. The rationale behind this algorithm was to overcome the limitations of scripted agents and systems. Scripted game playing agents have a tendency of repeating their behavior. An observant human player, after some games, can learn to exploit the weakness in such a predictable system. To deal with this problem NEAT alters both the weighting parameters and structures of networks. This is done in attempts to find a balance between the fitness of evolved solutions and their diversity. NEAT is based on three techniques:

1. Tracking Genes through Historical Markings
2. Protecting Innovation through Speciation
3. Minimizing Dimensionality through Complexification

The real time version of NEAT or rtNEAT, can evolve increasingly complex ANNs in real time as a game is being played. The NeuroEvolving Robotic Operatives (NERO) game was built based on rtNEAT. In NERO, the player trains a team of virtual robots for combat against other players’ teams.
Table 2.6 Operations performed every n ticks by rtNEAT.

<table>
<thead>
<tr>
<th>The rtNEAT Loop:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Calculate the adjusted fitness of all current individuals in the population</td>
</tr>
<tr>
<td>2. Remove the agent with the worst adjusted fitness from the population provided</td>
</tr>
<tr>
<td>one has been alive sufficiently long so that it has been properly evaluated</td>
</tr>
<tr>
<td>3. Re-estimate the average fitness $F$ for all species</td>
</tr>
<tr>
<td>4. Choose a parent species to create the new offspring</td>
</tr>
<tr>
<td>5. Adjust dynamically and reassign all agents to species</td>
</tr>
<tr>
<td>6. Place the new agent in the world</td>
</tr>
</tbody>
</table>

Table 2.6, from [196], gives an insight on how the rtNEAT algorithm works. The basic idea of this algorithm is to use a lifetime timer, to put the population under constant evaluation. When a timer expires the current fitness of the network is measured. If the network ends at bottom of the population, it is discarded and a new network takes its
place. This new network is a child of two high-fitness parents. This process is repeated for the new network. This is called the replacement cycle and operates continually throughout the length of the game. This idea is shown in Figure 2.9, from [196], where agents are represented as small circles with an arrow indicating their direction. Agents represented are playing a game within the large box. The replacement cycle results in generating new behaviors that are largely invisible to the player. In the first phase of the NERO game, each player deploys robots in a ‘sandbox’ and trains them. Once training is completed, a second phase of play allows players robot to compete against robots trained by other players. This competition serves to test the success of training performed on robots.

Though NERO overcomes the issues of adaptability to some extent, however, it still does not answer the issue completely. A simple example of this limitation is the fact that if a player developed a way of cheating (for example, seeing through walls or flying etc.) the opposing agents would remain oblivious to these changes in the environment.

Another issue is the training of NERO agents is done by humans; this imposes a limit on the abilities of the agents to the perception of their trainer. To some extent these limitations will force NERO agents to behave like scripted agents ignoring the changes occurring in their environment. The designers of NERO decide what inputs should be in training that the virtual agents use to fight. It can be argued that agents in NERO can improve themselves based on what they are given. They lack the ability to decide what they should perceive from their environments and adapt according to any new stimulus provided by the environment.

### 2.8 Limitations of Traditional Learning

Learning is the nature of intelligence. We deem individuals as intelligent if they can learn from experience and adapt to changing environments. We have described a number of popular machine learning paradigms in this chapter. We believe that traditional approaches to artificial intelligence have ignored the importance of adaptation
of change. Evolutionary learning, such as genetic algorithms or swarm intelligence forces the population to move towards the best candidate solution found so far. These approaches ignore the fact that the best solution may become obsolete after a change in the environment. It is possible that a change in system may introduce new factors that need to be considered and have never appeared in the previous history. Fogel [93] pointed out that "every intelligent system in nature adopts a functionally equivalent process of reproduction, variation, competition, and selection".

Although from Deep Blue to IES, AI and AI based systems have come a long way in terms of learning. However, we believe that so far AI research has ignored the role of a changing environment in the evolutionary process. Consider, for example, the case of Blondie24. As mentioned earlier in this chapter, it combined ANNs and EP to learn to play checkers from scratch without any pre-injection of human knowledge. However, Blondie24 is still not an adaptive intelligence because it fails to respond to new challenges from the environment. Even though the rationale and process for creating Blondie24 is an adaptive in nature, the end product is not [16].

We believe that the problem is with the way people regard artificial intelligence as a perfect end product rather than as an evolutionary process. We humans living in the real world know the importance of change in our life. Our learning process is a continuous one and lasts a lifetime. We argue that an incomplete environment is the primary reason behind our continuous and adaptive learning process. We agree with Kendall and Su [16], who argue that intelligence should not be considered as a perfect end product, but should be considered as an imperfect evolutionary system that constantly adapts itself and responds to the new challenges from an incomplete environment.

In this thesis, we extend the notion of imperfectness and adaptive learning one step further by defining mechanisms of how learning should continue after a change has been introduced into the system. We present a framework that defines the learning processes into sub-processes, each of these is essential for successful adaptation of
change. Our work presents a new learning mechanism that deals with the imperfect and changing nature of the environment. In our approach the environment not only acts a medium for evolution but is itself a part of it. Intelligent agents in this environment must be able to respond to change(s) at any time and modify their behavior according to the nature of the change (a change can be availability of new input parameters, introduction of new entities or species in the environment, change in agent’s objectives etc.). This self motivated and self directed evolutionary process allows the agents to develop better strategies to meet the demands of their environment. Humans play no part in this process, using the algorithm it learns everything on its own. Hence it is possible to avail opportunities that might have been ignored by a human performing training of the agents.

2.9 Summary:

In this chapter we have presented an overview of some of the learning techniques currently being used artificial intelligence (AI) researchers. We have also presented a brief history of what we believe were major milestones or advancements in evolutionary game learning. Our emphasis in this chapter has been to present what has already been done. In the end we presented a discussion on what we believe has been lacking from AI research, imperfection and incompleteness of information.

In the next chapter we define what we mean by imperfection in an evolutionary environment. We will elaborate the dynamics and intricacies of such an incomplete environment. We introduce the fundamental component of an imperfect environment and how change can affect each of these components. We will explain the relationship between such an environment and the agents residing within this environment. A theoretical imperfect environment will also be presented.
Chapter 3: Imperfection in Evolutionary Systems

The things we know best are the things we haven't been taught
- Luc Vauvenargues

A background of Imperfection in Evolutionary Systems

3.1 Introduction

In this chapter we discuss the idea of imperfection in an evolutionary system or an Imperfect Evolutionary Systems (IES). IES has been introduced by Kendall and Su in [16]. Their paper on IES is the first to propose a shift from traditional perfect learning paradigm to an imperfect one for evolutionary systems. Based on their work, we first define what an IES is and how it is different from traditional learning environments. After that we move on to a discussion of how agents are motivated to improve their current utility by adapting to new challenges and opportunities presented by their environments. This is followed by our definition of how imperfection can exist in a system and the relationship of the agents residing in an IES with their environment. We then discuss the mechanisms necessary to incorporate change in a system and how this information can be disseminated to the agents residing within it. A major focus of this chapter is to explore and elaborate different parameters of the environment and how imperfection affects them. Ways to introduce a change in an imperfect and incomplete environment are also explored. This chapter serves as a platform for further discussion about incompleteness in an evolutionary system.
3.2 Imperfect Evolutionary Systems

Although current learning approaches have been successful in some domains, we believe their application is limited (for more details see chapter 2). As an example let’s consider a co-evolutionary approach to game playing. Players compete against one another to find the best strategy. Once a strategy is successful against an opponent the learning methodology will force the remaining strategies to be “more” similar to this successful strategy. Traditional approaches will ignore any new development in the opponent. If, for example, the opponent has devised a way to cheat, the learning methodology will remain oblivious to this fact forcing new learning to be influenced by obsolete techniques. We argue that this failure comes from the ignorance of change in the surrounding environment. We stress that an important feature of the learning process is the ability to explore and acquire new information. It is common practice among us (humans) to learn a new “thing” just by observing others. A person can learn to play a game of TicTacToe just by observing others playing. The same person can also learn to play checkers if he comes across it.

An evolutionary environment with change as its essential and integral part is termed as an Imperfect Evolutionary System. According to Kendall and Su [16], “... an imperfect evolutionary system is a system where intelligence individuals optimize their own utility, with available resources, while adapting themselves to the new challenges from an evolving and imperfect environment”. Figure 3.1 presents a diagrammatic view of such an IES at different time stages. \( E^T \) is the imperfect evolutionary system at time \( T \) (represented by the large outer circle).

The imperfect environment \( E^T \), is defined by a set of information \( I^T_n \). Entities residing in the environment have this information available to them. At each time step \( T \), there is a possibility of change in the environment. Each new change may open new opportunities for the inhabitants of the environment (Figure 3.1 b) allowing for further exploration or it may remove certain information from the environment rendering previously evolved strategies to be obsolete (Figure 3.1 c). This paradigm of
imperfectness introduces the notion of “imperfect individuals” [16], individuals that do not completely understand their surrounding environment. Each one of these individuals is able to see a limited area of the overall environment. Based on its perspective, each agent (term agent and individual are used interchangeably in this thesis) builds an understanding about the environment. After a change has been introduced to the environment, agents must react according to these changes. They must adjust their behavior to better utilize the new information available to them.

![Figure 3.1 Imperfection in Evolutionary Environment](image)

**Figure 3.1 Imperfection in Evolutionary Environment**

3.3 **Components of an Imperfect Evolutionary System**

Based on the original work of Kendall and Su [16], we now expand the concept of IES. To better understand the notion of change in an environment, we must first analyze what information is available to its inhabitants. In other words to know what has changed we must look at what can change. This will allow us to understand how each change will affect the behavior of individuals and how they can readjust their strategies to deal with it. For these reason we start by defining fundamental components of an imperfect system.
As change is at the heart of an incomplete (or imperfect) system, we then describe mechanism of change within these components of the systems.

The imperfect environment $E^T$ is defined by a set of information $I^T_n$ (3.1).

$$E^T = \{ I^T_1, I^T_2, I^T_3, I^T_4, ..., I^T_n \}$$

The first information block available for the agent of the IES at time $T$ is $I^T_1$. Each block $I_i$ is further comprised of smaller blocks of information about the environment. Equation 3.2 explains that $I^T$, is a set of union of set of $x^T$ which is the number of input parameters (or sensors) available to the individuals of the environment, $r^T$ is the set of all the rules currently effective in the environment, $o^T$ is the set of objectives the individuals must achieve (or optimize), $u^T$ is the set of environmental entities present in the environment, $a^T$ is the set of (evolving) agents (or species in case of more than one agent type). Any change in any one of these sets can lead to a drastic change for the inhabitants of the environment.

$$I^T = \left\{ \sum_{i}^{\infty} x^T + \sum_{i}^{\infty} r^T + \sum_{i}^{\infty} o^T + \sum_{i}^{\infty} u^T + \sum_{i}^{\infty} a^T \right\}$$

We briefly explain each of these components of information:

1. The set of rules currently applied to the environment (represented by $r^T$), are the rules that govern different aspect of the environment at any particular time $T$, e.g. if the environment contains agents that can move then a rule can govern their maximum speed, other rules can control how far an agent can see, etc. Over time these rules can change. Depending upon what rules have changed a species may have to abandon some or all of its previously learned strategies. It is possible for rules to be applied specific to a species, in which case other species will not be affected (and may not even notice) this change. However, it’s more likely that change in rule that affects one species directly will affect other species indirectly. For example, consider the example of a predator prey environment with three species, $A$, $B$ and $C$. Species $A$ hunts $B$ who in turn hunt $C$. Suppose an
environment rule limits the maximum step size of $A$ to be restricted to 1 step per turn. If this rule is changed to 2 steps per turn, population size of species $B$ will be affected (since $A$ can now move faster and they are more likely to catch $B$). The effect on $B$ (directly by the rule) will also affect $C$ (indirectly). The impact of this rule change on species $C$ will depend upon survival strategies employed by $B$. $B$ may choose to reproduce at faster rates which will pressurize $C$ for survival or if $B$ fail to deal with this change their number will dwindle resulting in a population boost for $C$.

2. The set of input parameters (represented by $x^T$), are the inputs available to the agents at time $T$. These inputs govern the different aspect of the environment visible (or sense-able) by the agent e.g. distance from closest neighbours, colour of the closest object, etc. In the mentioned example of species $A$, $B$ and $C$, suppose all three species have the ability to sense their surroundings using sight. If $A$ develop a sense of smell they can detect their prey otherwise hidden (e.g. behind walls). Again this will affect more than one species present in the environment. Given the uncertain and uneven (here meaning to be imbalanced) nature of environment, a sensor maybe be more suitable for some agents while other agents of the same species may not find them to be favourable for their strategy formulation.

3. The set of objectives (represented by $o^T$), contains the tasks agents are supposed to learn in order to survive within the environment. The performance in terms of a currently active objective, determines the fitness of an agent. Examples of objectives can be reaching a certain point on the terrain within a short time, finding a mate etc. Objectives can be divided into primary and secondary objectives. For example, the primary objective of all species may be to survive. This could require mate selection, food collection etc. For secondary objective agents can be made to favour one resource more than others. However, whenever secondary objectives come in conflict with primary objective, they must be abandoned (e.g. survival at all costs). New objective can be added to the
environment and primary objectives can switch at any time. Agents must determine if their objectives have changed (no explicit information is to be provided to them).

4. The set of environmental entities and parameters (represented by $u^T$), contains all the elements of the environment that affect its inhabitants. Environment parameters will directly influence the strategies adopted by the learning agents. Some examples of environmental parameters can be, current wind speed, sliding friction, strength of walls, availability of a particular resource. Entities residing in the environment can be different types of resources or objects that maybe present in the environment. Availability of new entities is likely to affect the objectives of agents.

5. The set of agents or species (represented by $a^T$), contains the different types of species of agents present in the environment. Each of these species will have attributes that distinguish it from other species. We would like to point out that it’s possible to have some non-intelligent (scripted) agent species. Each species struggles to survive and dominate the environment. This competition motivates the quest for superior and more competitive strategies.

Each of these sets can change at any time. A change in the environment can be motivated by an event (e.g. when a species learns a particular task to a certain level a new enemy is introduced to the environment), it can be introduced on the basis of time passed by the agents in the environment (e.g. after agents have spent a predetermined time in the environment a new objective is added for them to learn). A change in any of these components may render the current learning to be obsolete. In such a situation agents will have to discard (or modify) their current strategies to survive the new state of the environment.

We would also like to point out that a change in one of these sets may lead to change in other factors, e.g. if a new species of agents is introduced into the environment,
it will change the set of rules applied to the environment. It may also change the number
of sensors available. Depending upon the nature of environment and the relationship
between different species residing in it, a new species may also lead to change in
objectives of other species.

3.4 Individuals relationship with its Environment

As we have already mentioned in the beginning of this chapter, each agent builds
a relationship with its environment based on its own perspective. The perspective of the
agent is dependent upon its locality. Unlike some traditional approaches, the environment
is never completely visible to the agent. Each agent establishes a dynamic relationship
with its surrounding environment built upon its current knowledge space. This knowledge
space is constantly updated to accommodate for any changes that have occurred in the
environment.

Figure 3.2 (page 57) presents an example of the relationship present between an
agent and its surrounding environment. The agent detects its environment via a set of
available input sensors. The chosen sensors can be a subset of available input sensors. In
Figure 3.2 (a), we present the individuals picture of the environment. Each individual
forms this picture based on what it has detected from its surrounding. It is entirely
possible for agents belonging to the same specie to form two different pictures of the
same environment based on their individual experiences. The experience pool is also
visible to the agents at any time for them to consult past strategies that have been
successful. However, it must be noted that due to the imperfect and incomplete nature of
the system, there is no guarantee for the past experiences to be successful in the current
situation.

Each individual contains two features 1) attributes and 2) skills. Attributes are the
limitations imposed on the species collectively e.g. if the environment contains a species
of birds then they may have the ability to fly, other species in the environment may lack
this feature. Skills are those attributes that are learned by the agents over time. Each
species of agent can increase their current skill set by utilizing the information available in the environment. The individual then formulates a strategy to deal with the current situation presented in the environment. Each strategy formulated by the environment can be represented by an ANN or rule-set. Figure 3.2 (b) presents the example scenario where the agents use an ANN based representation for their strategy formulation. In such a scenario agents will form a strategy by combining different pieces of information visible to them. The input and output layers will contain the inputs and outputs chosen from the set defined by the environment. Agents will then have to perform evolve an ANN architecture that is suitable to the task they have to learn. Individuals residing in the IES are never explicitly informed about their learning task, only the fitness of their current strategy is available to them. Based on this evaluation they must decide whether to improve current strategy or abandon old strategies and formulate new ones. We believe this layer of abstraction allows the agents to learn within any environment. If agents are given explicit information about the task they have to learn, this will adversely affect their ability to learn in the presence of change without human intervention.

Being imperfect in nature, each component of this learning is dynamic and changeable. Successful individuals are places within the experience pool and serve to disseminate “good” strategies to the rest of the population. Individuals that are performing poorly have the option of choosing a strategy from the experience pool. It must be noted that the likelihood of an old strategy working within current situation is always probabilistic and never deterministic.

It is also possible that the attributes of a species are changed over time. This change can be stimulated either by time or by the current knowledge space (or skill set) of that specie. For example a species that currently does not have the ability to fly can learn to fly using the information and resources present in the environment. All in all, an imperfect evolutionary system is a vibrant world with dynamic interactions between inhabitants and their individuals. Each individual of the species is constantly monitoring its surrounding to improve the its own fitness which in turn effects fitness of the species.
3.5 Intelligence in an IES

As mentioned in previous subsection, to be deemed intelligent, the inhabitants of the IES must build a dynamic relationship with their surroundings. They must detect and adapt according to any change that may occur at any time. We now look at how individuals (or agents) utilize the information available to them.

The perspective of an individual is dependent upon its locality. Equation (3.3) presents the perspective of agent $I_A^T$ and the environmental variable $v_e^T$, currently present in its observable space.

$$I_A^T = \{v_j^T, \ldots, v_k^T \mid j,k \in n \}$$  \hspace{1cm} (3.3)

Just like we (humans), continuously add to our knowledge space by everyday experiences, individuals residing within an imperfect environment monitor their surrounding and constantly update their knowledge space to add any new information. Similar to humans, these agents continuously improve their abilities and adjust their behavior to better suit the demands of the surroundings. This continuous learning process enables them to utilize their surrounding environment for their benefit (greatly improving the chances of survival of their species).

In order to incorporate this intelligent behavior based on the continuous learning process we have developed a framework that allows simulated computer based agents to mimic the continuity of learning. Our learning algorithm is based on this framework that uses a particle swarm optimization (PSO) based approach to train artificial neural networks (ANN), these ANNs represent the abilities (or skills) of the agents. The framework independent of the underlying evolutionary approach; evolutionary algorithms (EA) can be used instead of PSO and rule-sets instead of ANNs.

In an IES, any change in the environment will bring about a change in the search space with the consequence of no point being optimal (or even near optimal) for all phases of the environment. We believe that in some ways this can be thought of as being
similar to function optimization in a dynamic environment where the optimum keeps changing. However in terms of an IES, we are not referring to the optimum changing its location in the search space, we mean the optimal point (or even a correct solution) jumps from one search space to another. Once the solution has moved from the current search space to another, no amount of search in the current search space will be fruitful. In our approach to learning, the evolution process takes into account any changes that have occurred within the environment or any changes in the observable space of the individual. Each individual then has to decide whether these changes affect them or not. In case the individual is affected, he must modify its behavior to either exploit or avoid them. This process of detection and self modification is an integral part of the overall evolutionary process. Adaptation to change is initiated by the introduction of change into the environment and occurs without any human intervention. We believe that intelligent agents must learn about their environment on their own without the guidance of human beings.
Figure 3.2 Individuals in an Imperfect Evolutionary Environment
3.6 Summary

In this chapter we have presented the dynamics of an imperfect evolutionary system. We have presented the details of how and why an evolutionary system is imperfect. We presented the relationship of this environment with the individuals residing within it. We have elaborated how change can effect different aspects of the environment and how these changes bring about a change in individual’s behavior. We have also presented an abstract imperfect environment that is dynamic in nature. We have shown how individuals view this environment and how they need to adapt and adjust their behavior to ensure their survival. These individuals must train themselves according to the specifications of their environment. The abilities of these agents are only limited by the set of restrictions imposed by their surrounding environment. Individuals should be able to improve their abilities and knowledge about their surroundings as they spend more time within it.

In the next chapter we discuss a continuous approach to learning. We present our continuous learning framework and we also present a learning algorithm based on this continuous framework. Agents applying this learning methodology have the capability to detect when their previous learning has become obsolete and they are able to learn new strategies for survival. Our learning algorithm is not designed for a particular environment rather it builds a relationship with its surrounding environment and continues to learn from it.
Chapter 4: Continuous Learning Framework

Dealing with imperfection of information

*I never teach my pupils; I only attempt to provide the conditions in which they can learn.*

- Albert Einstein

4.1 Introduction

The ability to learn new skills has been the hallmark of our (humans) success as an intelligent species. We can improve our current set of skills based on the information available to us, thus allowing us to expand our current knowledge space to incorporate new information (whenever it is required or available). In the previous chapter we present an imperfect evolutionary system (or an IES). This imperfect system is modeled after the real life scenarios we face. In our daily lives, we humans, face new information. This information is used to constantly update our knowledge pool. Similarly individuals residing in an IES are exposed to new information. This new information is utilized to update their current set of skills and abilities. In the previous chapter we presented how the individuals form a dynamic and evolving relationship with their incomplete environment.

In this chapter we introduce the notion of continuity of learning in an imperfect and incomplete evolutionary environment. We discuss the underlying processes that form the basis of this continuous learning. We present a framework that employs these processes to enable automated skill learning of agents. The continuity of learning enables the agents to learn new abilities on their own without human intervention. This automated learning allows the agents to learn new abilities based on the information present in their environment and their abilities are only limited by the information available to them. This dynamic relationship enables agents to constantly monitor their surroundings for new
Continuity of Learning in an Uncertain & Dynamic Environment of Imperfect Information

Whenever new information is added to the environment, individuals then have the ability to decide whether to use or ignore this new information.

4.2 Continuity of Learning

When faced with an alien environment, humans rely on their intelligence to learn about their surroundings. This natural intelligence determines how we learn and what we learn. We believe this learning process to be comprised of three sub-processes. We call these sub-processes the 3E’s of learning i.e. experience, exploration and exploitation. Figure 4.1 presents a diagrammatic integration of these three sub-processes. Each one plays a vital role in establishing a relationship with the environment. These processes help us in understanding and adapting to our surroundings, thereby ensuring survival of the species.

We now define these three processes as applied to continuity of learning and then present a framework that utilizes these concepts. In this sense our framework, is about learning how to learn.

Figure 4.1 Continuity of Learning

4.2.1 Experience

We use experience to mean to learn something new based on observation or via past events i.e. something that has happened to the learning agent or was observed by the
learning agent. Experience allows an intelligent entity to use previously gained knowledge and apply it to a new scenario. Using experience, learned abilities can be applied to explore new opportunities. Human beings constantly use experience in our daily lives. Whenever faced with an alien environment we utilize our past experiences to determine how we can handle the new challenge presented to us. For this reason, we believe, the first step in continuity of learning is to establish a link between past knowledge and the current scenario. Individuals residing within an imperfect system will also have to utilize all that they already know about their environment and their past exposure to it, in order to formulate a successful strategy for survival.

4.2.2 Exploration

Exploration is the searching of a (search) space for new information (or knowledge). Exploration allows intelligent beings to learn more about their environment. Generally the more we know about a situation, the better we understand it. Our understanding of problems enables us to formulate suitable strategies to deal with them and reach our goals. In terms of learning, exploration combined with experience allows humans to learn new abilities based on the set of abilities already possessed and new knowledge gained. Where experience provides the first step towards approaching a problem, exploration leads to a better understanding. Without exploration, any changes in the environment may remain undetectable. Exploration within IES occurs in the individual’s current locality. Individuals monitor their surrounding for new information. Once this new information is detected, agents may choose to use it in their strategy formulation. Depending upon the nature of the environment and the level of sharing between individuals it is possible for agents within the same species to have different levels of exploration.

4.2.3 Exploitation

The third process is exploitation i.e. the refinement of the current set of skills based on gained experience and exploration of the observable search space. Exploitation is responsible for adjusting an individual’s behavior according to the demands of the
environment. Exploitation is the part of continuous learning that enables agents to deal with new challenges based on all the information they have collected about the current state of the environment. This sub-process is crucial for learning to occur without human dependence.

### 4.3 Individual Learning

The complex nature of the environment paired with the additional layer of incompleteness and continuity of learning results in different levels of learning. We will discuss these one by one. We begin by discussing evolution occurring at the individual’s level (or individual learning).

Each agent residing in the environment is constantly trying to improve its own utility thereby increasing chances of its survival. At each iteration, best individuals out of the population are selected for reproduction in next generation. Every individual strives to be the best to ensure passing on its genes to its successors (hence survival of the fittest gene). The individual’s learning mechanism works in the following manner. Agents formulate strategies based on its own perspective of the environment, its past experiences and its understanding of the demands of the environment. Keeping these in view every individual tries to learn about its surrounding environment. As the individual spends more time in the environment its level of understanding about the environment increases.

The other task the individual has to perform is to adjust its strategies or behavior based on the changes that are occurring in the environment. Agents that reside within an imperfect environment are imperfect individuals [16]. These imperfect individuals try to optimize their fitness value (or that of their deployed strategy) by monitoring and adapting to change. Each individual of the species tries to form a strategy that is able to meet the requirements of the environment. Their non-static view of the world leads to formulation and deployment of interesting solutions (or strategies) for the given problem. These individuals make no assumptions about their environment but rather rely on the environment to provide them information about itself. This leads to individuals
developing strategies that can exploit any weakness of the evolutionary system. Therefore, individual learning drives the agents to improve their chances of survival enabling them to devise solutions that may be ignored (overlooked or not conceived by a human trainer).

4.4 Social Learning

When individuals of a species strive to excel in the evolutionary process, this individual effort leads to a collective improvement for the species. Social learning also implies that individuals of the same species learn from each other. The presence of an experience pool further motivates the social learning process. Inhabitants of the imperfect world adopt strategies that have been successfully applied by other agents. This also allows for dissemination of knowledge within the imperfect world.

Social learning stimulates the competitive behavior among other species as well where two or more species can compete for dominance over terrain or resources. However, it must be noted that social learning is driven by the individual’s quest for survival and optimizing its own utility. Agents within an incomplete environment have to constantly improve their own strategy for two reasons.

1. Each agent wants to be the best.
2. Each agent wants to survive.

Although these may seem to be the same objective, they have some differences and are at times in conflict with one another. Every agent wants to be the best within its species. By being the best it will have more chances of passing on its genes. However, agents must also ensure that they share their best strategy with other agents because an agent cannot survive on its own if its species dies out. Shared strategies are placed within the experience pool. These strategies are now available to other agents of the same species and they may chose to deploy them. So in the new scenario more than one agent is using the same best strategy (found so far). This motivates the agents to further improve this strategy or find a new even better strategy to be the best within their species.
Hence the desire to compete and survive at individual level results in improvement at social level.

### 4.4.1 Community Based Learning

Another instance of social learning is when agents of the same species residing within close locality form a community. This community of agents’ works on both the individual learning level and competes with agents of the species residing elsewhere. These agents can form other means of knowledge sharing improving their own fitness along with those of their community. Community can be formed on the basis of common elements other than locality, for example, agents of the same species can also form a community if they face a common enemy which other agents of that species are not currently exposed to. This community will design strategies to fight this enemy more rigorously than other individuals who are currently not threatened by it.

Our framework incorporates both the social and individual level of learning (and evolution). We now present our framework and describe in detail how its different components work.

### 4.5 Continuous Learning Framework

We believe the continuity of learning is the ability to invoke and utilize these processes for discovery and exploitation of information (knowledge) in the current environment and modifying behavior to improve the chances of survival (or improve fitness). In essence, our continuous learning framework provides simulated agents with the “ability to learn to learn”. The basic steps of the framework can be described as follows.

i. **Experience**: For every new problem faced, determine its relationship with already gained knowledge and abilities.

ii. **Exploration**: Formulate a strategy to deal with the situation based on the set of currently available information.
iii. *Exploitation:* Evaluate and refine (or modify) the current skill set based on the effectiveness of currently deployed strategies.

Learning agents must first determine if the problem faced is a new one or something they have experienced in the past. This determines whether they need to acquire a new ability or improve their current set of abilities. This step may provide a general direction of learning by reusing previously learned knowledge. However, there may be cases where a completely alien environment is faced and none of the previous abilities is applicable. This brings us to our next step of exploration. By exploring different possibilities provided by the environment, new opportunities can be exploited. Exploration leads to a better exploitation and understanding of the environment. Exploration also builds up experience by observation or occurrence. Evolution, then refines and improves the information gathered during exploration. Evolution acts like a filter for weeding out useless or obsolete information. As a simple example, exploration led to the invention of the wheel, evolution led to a better more usable form of the wheel and experience stops us from reinventing the wheel.

We present our framework in Table 4.1. This framework uses the processes defined for continuity of learning to initiate and acquire new learning. The framework is independent of the environment, problem or the evolutionary approach. The only information the simulated agents receive about the current environment state is the evaluation of their current strategy. No other information about the environment is explicitly shared with the agents. Step 3 of the framework allows compatibility with different evolutionary algorithms. Each agent represents a potential solution to the problem. For example if a genetic algorithm based approach is used, each agent will be represented by a chromosome. Agents can then decide upon the length and structure of the chromosome. In case the experience pool is already populated they may choose a candidate solution from that pool or initialize a randomly generated solution. Depending upon the number of input (sensors) available to the agents, they can perform different tasks. In other words what the agents can do depends upon what the environment provides (or allows). Availability of information from the environment determines the possible combinations of inputs and outputs. These different combinations result in
different abilities of the agents (e.g. if an agent is to walk the agent must first be able to see where it is going). Evolution then refines these abilities to higher levels, improving the agent’s fitness. Step 3 of the framework allows the agent to start its learning based on the socio-experience pool or from a random solution.

A socio-experience pool is similar to an archive that stores the best solutions found by the entire population. This pool can be further subdivided into smaller pool where each pool only stores the best strategies from a selected portion of the environment. Step 4e populates this experience pools. Each one of the policies mentioned in the framework can be either predetermined or evolved alongside the other evolutionary process. Step 4a-d controls the evolutionary part of the framework and step 4f deals with exploration process. It is interesting to note that in the absence of this socio-experience pool, agents forget all previous learning when faced with a new challenge. If there is a chance that the environment may revert back to one of its previous states, then this forgetfulness will result in agents relearning their forgotten abilities.

**Table 4.1 Continuous Learning Framework**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Initialize all parameters</td>
</tr>
<tr>
<td>2.</td>
<td>Create population of agents</td>
</tr>
</tbody>
</table>
| 3.   | Strategy Selection by agent (using either 3a or 3b)  
  a.  | Formulate strategy by allowing agent to  
  i. | Select a set of inputs (or sensors) out of available inputs  
  ii. | Select a set of outputs (or task)  
  iii. | Structuralize a strategy  
  b. | Select strategy already present in the experience pool |
| 4.   | Repeat for Tm iterations  
  a. | Evaluate the fitness of the agent based on the current state of the environment  
  b. | Determine the best solutions and their locality  
  c. | Adjust the exploration parameter  
  d. | Allow agents to move towards the best solution visible to them  
  e. | After Tm iterations populate the experience pool  
  i. | Determine candidates for the pool via selection policy  
  ii. | If candidate is not already present in the pool add to the pool  
  iii. | If there is no empty slot left in the pool invoke removal policy  
  f. | After Tx force exploration  
  i. | Determine the worst agents |
ii. If the worst agents have not improved in the previous $T_i$ iterations force them to abandon their current strategy and formulate new strategy using step 3.

iii. If worst agent is improving choose 2nd worst for exploration

iv. If both worst and 2nd worst are improving then do not check any other agents.

g. Invoke experience pool decay policy.

5. Go to step 4.

Due to the imperfect nature of the surrounding environment, inputs and outputs (or any other parameter) can change at any time without any warning. In case a certain input is no longer available in the environment, agents relying on that input will have to abandon their current strategy and formulate a new strategy based on the currently available information. In this manner agents build a dynamic bond with their surrounding environment. Agents make no assumptions about their surroundings. They constantly search for any changes that may have occurred in the surrounding and at each step their actions are evaluated. This evaluation guides the agents as to whether they have selected the correct path (or strategy) or not.

It must also be noted that availability of information in the environment can vary based on the current location of the agents. Agents residing in one “corner” might not be aware of inputs available at some other corner and vice versa. This allows for a more diverse population of agents, allowing for better exploitation and exploration of the environment. Due to this difference, agents are only made aware of the best strategies that are in their locality. Best strategies are determined on the basis of their effectiveness in achieving the currently set goal. As goals change these old best strategies will need to be abandoned. The framework determines when a deployed strategy has become obsolete and a new strategy needs to be formulated. The exploration parameter is adjusted to control the jump of the agent. Ideally an agent doing well should not jump far from its current location and should explore its current locality for a good solution to the problem. An agent performing badly should jump away from its current location to look elsewhere for a better solution.
4.6 Summary

In this chapter we have presented our learning framework. This framework only relies on the stimulus provided by its surrounding environment, this allows the agent to learn new abilities without being told to learn. Agents train themselves according to the task set by the environment. In this manner the environment not only acts as a medium for the evolution but itself becomes part of the evolutionary process. At any time the environment can be changed without informing the agents of this change. The agents are then able to learn about their new environment using the processes of experience, exploration and evolution.

In the next chapter we present an imperfect evolutionary environment that we developed in order to investigate the learning ability of our framework. We developed this environment to test the learning capabilities of our framework. This environment is a multi-agent, multi-objective, incomplete and dynamic environment. This environment supports multiple types of species that can compete against one another. This environment allows change to occur in any of the components mentioned in chapter 3. We will present details of different kinds of entities present in the environment and the challenges they have to face during the evolutionary process.
Chapter 5: Dynamic & Imperfect Gaming Environment

An uncertain gaming environment

The art of teaching is the art of assisting discovery
- Mark Van Doren

5.1 Introduction

In the previous chapter we presented a framework that allows the agents to continuously learn about their environment. Our framework employs the sub-processes of continuous learning (detailed in previous chapter) to enable automated skill learning of agents. Using our continuous framework, agents form a dynamic relationship that enables them to constantly monitor their surroundings for new information. Whenever new information is added to the environment, individuals then have the ability to decide whether to use or ignore this new information. This automated learning allows individuals to formulate new strategies that are suited to the needs and demands of their environment. In this manner the abilities of agents are only limited by the information available to them.

In this chapter we present an imperfect game like environment which we have developed to test the learning capabilities of the framework presented in the previous chapter. This environment is primarily a computer game inspired test-bed and the design of the simulated agents used in this environment is deliberately kept abstract. We will discuss the different component of the imperfect environment presented in chapter 3. This chapter details the different kinds of entities present in the environment and the challenges their surrounding world presents to them.
5.2 An Imperfect World

In order to test our learning framework we developed a predator-prey environment. The central idea is to model the imperfection of information. It is a multi-agent environment, supporting multi-objective evolution. The environment has two kinds of entities, agents and artifacts. Agents are evolvable and use the learning framework to enhance their abilities as allowed by the environment. Although the environment supports multiple species, for our current set of experiments we only use one species of agents for evolution. Artifacts are the non-evolutionary (scripted) components of the environment. The environment is 2D, torus like structure (agents reappear on the opposite side as they cross the boundaries of the environment). The current dimension used for our experimentation is 80 x 80. However, these dimensions can be set to any other appropriate size. The environment allows the agents to gather information about their environment using a set of inputs for the ANNs used in the evolutionary process. With a change in the environment state this input set can be modified. Each of the states is fully controllable and observable to the human.

Agents detect and explore the environment using inputs and only receive feedback of how good their learning is (i.e. fitness value). Sharing any other information, explicitly with the agents would greatly limit the learning ability of the agents. This allows these agents to exploit any weaknesses of the environment, allowing for interesting observations (some of these will be presented later). Agents explore the environment, in search of survival strategies (which are not pre-injected but are evolved based on the inputs provided by the environment). These agents can be relocated to a different environment (with a completely different set of task, objectives, inputs, global rules, or fitness function etc.) at any time without affecting their learning abilities. Figure 5.1 shows the very basic setup of this imperfect world. Different symbols represent different species, each with its own unique attributes.
5.3 Agents and Artifacts in the Environment

We now describe the entities present within the imperfect environment.

5.3.1 Humanoids

Humanoids are the basic evolvable agents within the environment. Although the environment supports multiple-species of agents, in our current set of experiments, only one species (i.e. humanoids) is used. Humanoids are deployed on this alien environment and are allowed to explore and understand it. At the time of their deployment into the environment they are given no information about the environment or what they have to do within this environment. No a priori information is made available to these agents. Everything they learn, they must learn from their environment. This allows the agents to be independent of their environment and at the same time allows the researcher to incorporate any kind of change in the environment without being concerned about the
abilities of the agents. However for our experiment we only focus on object-avoidance and target-achievement skills. Agents are not informed about which objects are to be avoided and which objects are to be targeted.

Each of the agents decides to use a set of ANN inputs \( p_{x_a}^T \) available from the set of inputs. \( p_{x_a}^T \) is defined as those inputs used by the agent which were available from the set of inputs \( x \) allowed for species \( a \) at time \( T \) (5.1).

\[
p_{x_a}^T = \{ x_1^T, x_2^T, ..., x_n^T \}
\] (5.1)

Every agent can choose different inputs from the universal set of inputs to derive its own input set. Restrictions on using inputs from the universal set can be applied to agents based upon their species, locality, time etc.

Similarly, each agent is allowed to learn any of the abilities currently allowed by the environment. Every component of the environment is generic and modifiable in nature. Abilities of the agents can be taken as ANN outputs (or a combination of series of ANN outputs or a ruleset etc). At the time of their deployment into the environment the agents have to learn everything scratch (i.e. they have to learn to walk and improve upon this ability to explore the environment).

For our current series of experimentation we restrict the ANN architecture of these agents similar to the one used by Yannakakis et. al [207]. Our objective is not to derive the best ANN architecture but rather allow the agents to learn new ability(s) on their own.

### 5.3.2 Terminals

These are non-evolving stationary points on the terrain. Once a humanoid reaches a terminal point it disappears and a new terminal point is created at a random location in
the environment. For our current experimentation we fix the total number of terminals in the environment. The more terminals reached by humanoids, the better their fitness. Interesting scenarios can be created where, for example, the number of terminal points in the environment is gradually decreased to create a drought (treating terminal points as food items) like scenario and the humanoids would be forced to learn faster.

5.3.3 Creatures

These are non-evolving agents in the environment. Their behavior is scripted. Creatures attack humanoids whenever they can find them while two or more humanoids can cooperate with one another to capture a creature. Once a creature detects that it is being surrounded by humanoids it tries to run and avoid capture. Due to their dual nature they play a role in the environment. Due to their scripted nature they always use the shortest distance between them and the closest agent. The number of pixels visible to a creature is always smaller than the number of pixels visible to the agents. This is done to give both parties a fair chance.

5.4 Phases in the Environment

The agents and artifacts are placed within the environment for a certain time period. During this time they face one or more possible phases (or states) of the environment. There can be many possible combinations of phases based on the set of parameters mentioned in the previous sections. Each one of these phases can be followed by any other phase change. Humanoids (the learning agents) are never explicitly told about the current state of the environment or when the environment has changed from one state to another. Providing humanoids with any direct information about the environment state or its change runs contrary to the concept of continuity of learning.

In the experimentation presented in this paper we focus on the following phases.
5.4.1 Terminal Acquisition:

In this phase the humanoids learn to capture as many terminal points as they can within a limited time period. Whenever a humanoid captures a terminal, a new terminal is created at a random point in the environment. The fitness function is based on how many points have been captured. We use the following fitness function.

\[
F_i^T = \min\left(1, \frac{t_i^T}{mT^T}\right)
\]

(5.2)

Where \(F_i^T\) is the fitness of the \(i^{th}\) humanoid and is calculated as the minimum of 1 and the total number of terminal points captured by the humanoid \(t_i^T\) divided by the maximum terminals available for capturing, \(mT^T\), at time \(T\).

5.4.2 Avoiding Death

In the second phase creatures are introduced into the environment (agents had never learned (or even faced) this enemy (or obstacle) previously). This change requires a drastic change in survival strategy and agents will have to switch their primary objective from terminal capturing to avoiding death by a creature. The fitness function is now

\[
F_i^T = \frac{\left(\min\left(1, \frac{t_i^T}{mT^T}\right) + \max\left(0, \frac{md^T - d_i^T}{md^T}\right)\right)}{2}
\]

(5.3)

where \(md^T\) is the maximum number of time humanoids are allowed to be killed by the creature and \(d_i^T\) is the number of times humanoid \(i\) was killed by the creatures.

5.4.3 Capture Creatures

In the final phase the humanoids are allowed to fight back by capturing the creatures. Capturing a creature requires teamwork. Two or more humanoids must trap a creature (i.e. two or more agents must get very close to the creature) to capture it. A
creature may accidentally move too close to a group of agents in which case it will be considered captured. We only use number of creatures captured as the fitness function.

\[
F_i^T = \left( \min\left(1, \frac{e_i^T}{me^T} \right) \right)
\]

(5.4)

here \(me^T\) is the maximum number of creature captures allowed and \(e_i^T\) is the number creatures captured by the humanoid. Whenever two or more agents cooperate to capture a creature this number is updated for all the humanoids who participated in the capture.

A major difference between phase 2 and 3 is that in phase 2 agents can only avoid creatures and they do not have any mechanism to fight back, while phase 3 allows agents to fight back by cooperating with other agents of their own species. More interestingly, more complicated tasks can be generated from the combination of these simple, yet different, activities. One example of such a complex learning activity could be (5.5).

\[
F_i^T = \frac{(fe_i^T + fc_i^T + ft_i^T)}{3}
\]

(5.5)

Here fitness is the summation of total creatures captured \(fe_i^T\), number of collisions made \(fc_i^T\) and number of terminals captured \(ft_i^T\) by the humanoid, divided by 3. Collisions are the number of times an agent bumped into other members of its specie other than during the search for terminals or avoiding creatures.
5.5 Agent Movements

![Figure 5.2](image)

Figure 5.2 Different phases of the environment

Figure 5.2 shows examples of different phases of the *imperfect* environment. “o” represents an agent and “+” is the agents target. Dotted line shows corresponding movement of an agent towards a terminal. The details of agent movements are as follow:

a. Terminal acquisition, agents must focus on capturing terminals. Figure 5.2 (a) shows two agents moving towards two terminals. In this phase focus solely on terminal capturing ignoring other hurdles (e.g. colliding with other agents, being eaten by creatures, etc.).

b. Terminal acquisition with collision avoidance, an example scenario where agents must avoid colliding with one another while moving towards their targets. The collision point is shown by a small grayish circle. This collision point is avoided by one agent either stopping for two steps or changing
direction. Agents are never informed about collision points they must detect if their next few steps can lead to collision or not.

c. Creature capturing, each agent must meet with a fellow agent at the target location. Here the agents are focusing on capturing a creature. Agents must make their own decisions about whether they will cooperate or not and who will cooperate with whom.

d. Death avoidance, agents must avoid the creatures spawned in the environment, the dotted circle shows the viewing range of a creature. A trained agent will avoid coming too close to any creatures lurking in the environment.

Figure 5.3 shows an actual scenario of humanoid capturing a terminal (highlighted in the red circle). In each of the images the humanoid takes one step towards the terminal. Once a terminal has been captured the humanoid moves on to other terminals. Other movements, like humanoids being captured by creatures or humanoids cornering a creature, work in similar fashion.
Continuity of Learning in an Uncertain & Dynamic Environment of Imperfect Information
Figure 5.3  An agent “o” moving towards a terminal “x”
5.6 Summary:

In this chapter we have presented a multi-objective, multi-agent, uncertain, dynamic imperfect evolutionary environment. This environment accommodates different types of agents (intelligent and others). These agents have to face different scenarios presented by the environment. Agent fitness (or survival rate) depends upon the adaption to changes presented by their world. Few features of the environment have been left abstract for latter scalability and modification. This environment serves as a test-bed for our framework.

The next chapter details the experiments we carried out to test the learning capabilities of our framework and the results of this experimentation. We present a learning algorithm based on the framework presented in chapter 4. This learning algorithm allows agents to continuously learn new abilities. We will test the ability of the framework to learn in the presence of imperfection. Different levels and kinds of imperfection are tested using specially designed experiments. The results of our experimentation show the ability of our framework to successfully learn in both complete and incomplete environments.
Chapter 6: Continuity of Learning in DIGE

Investigating continuity of learning

*Learning without thought is labor lost. Thought without learning is intellectual death*

- Confucius

6.1 Introduction

In the previous chapter we have presented an imperfect environment created by us. This environment supports a multitude of species and for each species different objectives can be assigned. The environment described in the previous section was designed as an imperfect evolutionary system. Due to the incomplete nature of the environment, anyone of its components can be changed at any time. Agents residing in the environment have the ability to detect different kinds of information present around them and based on this information formulate new strategies to deal with the demands of the environment.

In this chapter we present the experiments we carried out in this environment to test the learning abilities of our framework. There is no information sharing between the environment or the framework other than the fitness evaluation of the strategies. No information is passed to the framework about a change in the environment or when it has to find new strategies to deal with the environment. All decisions are made by the framework to evolve a strategy that is able to handle any change in the environment. We first present the learning algorithm we used based on the continuous learning framework, then we present the environmental parameters used in testing and later we present the results of our experimentation.

We will test the framework in different scenarios starting by its performance in a perfect environment and moving on to a changing and uncertain environment. We will
also test the capability of the framework to handle multiple objectives. We also evaluate the performance of the framework and how it behaves when new dimensions of learning are presented to the learning agents during their evolutionary process.

We begin by specifying our learning algorithm which is based on continuous learning framework.

### 6.2 Specification of Learning Algorithm

Our learning algorithm was based on the continuous learning framework presented in chapter 5. For evolution we used a particle swarm optimization (PSO) based approach.

#### 6.2.1 Particle Swarm Optimization

PSO is a heuristic search algorithm modeled on the behavior of flocking birds. A population of solutions called particles is spawned randomly and then flown in the search space to find areas of higher fitness. A record of the best particle found so far is kept. Each particle also remembers its own best position. These locations along with random components are then used to determine the position of the particle. In our experimentation we used a modified version of the lbest PSO approach. lbest PSO is a variant in which the population is divided into several sub-swarms. In lbest PSO all particles within a sub-swarm are completely connected with each other. Each particle forms a sub-swarm by completely connecting to its \( n \) neighboring particles. These neighbors are based upon the particle numbers. It has been proven that lbest PSO provides more diversity at the cost of slower convergence. Positions of the particles are updated using the equation:

\[
x_i(t + 1) = x_i(t) + v_i(t + 1) \tag{6.1}
\]

where

\[
v_i(t + 1) = w v_i(t) + c_1 r_1(t) (y_i(t) - x_i(t)) + c_2 r_2(t) (v_i(t) - x_i(t)) \tag{6.2}
\]
\( x_i \) represents the current position of particle \( x \) in dimension \( i \), \( v_i \) represents current velocity, \( y_i \) is the personal best position of the particle and \( \hat{y}_i \) represents that sub-swarm’s global best (called neighborhood best) position to which the particle being updated belongs. The constants \( c_1, c_2, r_1 \) and \( r_2 \) are used to control the area in which search is to be conducted and \( w \) is inertia weight. The velocity updates are clamped in a range \([-v_{\text{max}}, v_{\text{max}}]\).

We modified the regular \( lbest \) PSO, and instead of creating sub-swarms in a sliding window fashion (a particle is connected to its \( n \) neighbors and the next particle is connected to its \( n \) neighbors, and so on), we have several completely connected sub-swarms which are isolated from one another except through their corner particles. Each corner particle is completely connected with all the particles of two sub-swarms (Figure 6.1). These corner particles act as a source of information transfer. The learning algorithm used for our experimentation is presented in Table 6.1.

Table 6.1 Learning Algorithm based on Continuous Learning Framework

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Initialize parameters (e.g. ( N, R, C, c_1, c_2, r_1, r_2 ) and ( w )).</td>
</tr>
<tr>
<td>2.</td>
<td>Randomly initialize PSO particles ( P ) in the range ([-r, r]) or choose a strategy from the socio-experience pool.</td>
</tr>
<tr>
<td>3.</td>
<td>Repeat for ( T ) iterations</td>
</tr>
<tr>
<td>a.</td>
<td>Evaluate fitness of a particle</td>
</tr>
<tr>
<td>i.</td>
<td>For every particle create ( N ) clones</td>
</tr>
<tr>
<td>ii.</td>
<td>Allow each of these clones to spend ( E ) steps in the environment</td>
</tr>
<tr>
<td>iii.</td>
<td>Based on collective performance of these copies calculate fitness</td>
</tr>
<tr>
<td>b.</td>
<td>Calculate (and if necessary update) personal best position of particles and best particles of every sub-swarm</td>
</tr>
<tr>
<td>c.</td>
<td>Adjust ( v_{\text{max}} ) for each subswarm</td>
</tr>
<tr>
<td>d.</td>
<td>Every ( T_e ) iterations</td>
</tr>
<tr>
<td>i.</td>
<td>Choose a candidate for addition to socio-experience pool</td>
</tr>
<tr>
<td>ii.</td>
<td>Add to pool if not already present (duplication not allowed)</td>
</tr>
<tr>
<td>iii.</td>
<td>If empty slot available in archive then remove the oldest strategy from the pool</td>
</tr>
<tr>
<td>e.</td>
<td>Every ( T_e ) iterations</td>
</tr>
<tr>
<td>i.</td>
<td>Find the worst sub-swarm</td>
</tr>
<tr>
<td>ii.</td>
<td>If worst sub-swarm has not located its best particle within ( T_r ) iteration reinitialize it either randomly or from archive.</td>
</tr>
</tbody>
</table>
iii. If worst sub-swarm has improved itself in past $T_x$ iterations then check 2nd worst sub-swarm, if it has not improved then reinitialize it. If it too has improved then do not check any further sub-swarms.

f. Decay strategies in archive

4. Go to step 3

In the initial run there are no strategies in the archive so all particles must be initialized randomly.

![Figure 6.1 Architecture used for defining and connecting sub-swarms](image)

### 6.2.2 Artificial Neural Network

Each particle represents the weights of an artificial neural network. Each of these ANNs represents an agent trying to learn its own specific strategy for survival. Though the continuous framework allows agents to evolve their neural network architecture, our goal here is to present a generic framework able to adapt to its changes, so we do not deal with the optimization of underlying ANN architecture. We believe work of Stanley et al. [196] would provide a suitable method to deal with this.

The ANN architecture we use comprises 12 input neurons, 1 hidden layer with 5 neurons and 2 output neurons. Inputs comprise of distance and angle of the $z$ ($z = 2$) closest terminals, humanoids and creatures. A sigmoid activation function is used. The two output neurons give the step size and the direction (or angle) in which to take that step. The ANN may not be optimal but experimental results show that it works well.
In step 3.a.i of Table 6.1., for each of the particles we create $N$ clones. Each of these copies represents an agent within the environment using the strategy represented by its parent particle. In step 3.a.ii. (Table 6.1.), the movement and interactions of all clones is monitored. The fitness of the particle is then calculated based on these interactions using the fitness evaluation function associated with the current state of the environment. The values of all the clones are summed to represent the value of the parent particle e.g. the number of terminals reached by a particle in iteration $I$ is the summation of number of terminals reached by all its clones in $E$ steps of iteration $I$.

Personal best and best particles of each sub-swarm are then determined using these fitness values. To set the exploration parameter in step 4.c Table 4.1., we use a dynamic approach to determine the value of $v_{\text{max}}$ (3.c. Table 6.1.). This was used to control the jumps in the movement of a sub-swarm. This allowed for a better exploration of an area in the search space by the sub-swarms. By maintaining a low $v_{\text{max}}$ value for best sub-swarms, we allow the best sub-swarms to do an extensive search of their locality while higher $v_{\text{max}}$ values for poorly performing sub-swarms allow these sub-swarms to try other areas in the search space. The equation used for setting $v_{\text{max}}$ value is

$$v_{\text{max}} = \max \left( \left( \beta - \left( \frac{f_s^T}{f_p^T} \right) \right) , 1 \right)$$  \hspace{1cm} (6.3)

The threshold value $\beta$ (set to 1.05), $f_s^T$ is the fitness value of the best particle out of the sub-swarm $s$ at time $T$ and $f_p^T$ is the fitness of the best particles out of the entire population $P$ at time $T$. After every $T_c$ a candidate is chosen to be added to the archive. Candidates for addition to the archive can be chosen using different policies (explained later in this section).

In step 4e of Table 6.1., the algorithm after every $T_e$ iterations initiates the process of exploration. The worst performing sub-swarms are used as candidates for exploring new strategies (which may use new combination of inputs, outputs etc). Due to the noisy
and unpredictable nature of the environment, a sub-swarm with a generally good fitness value may perform poorly at $T_s$ iteration and end up as the worst sub-swarm. Reinitializing this sub-swarm could mean losing a potentially good solution. Hence we chose to use $F_s^T$ to determine the worst sub-swarm.

$$F_s^T = \sum f_p^T$$  \hspace{1cm} (6.4)

Accumulated fitness $F_s^T$ is calculated by the summation of fitness $f$ of all particles $p$ belonging to the sub-swarm $s$ over a period of $T$ iterations. This allows monitoring of a sub-swarm’s performance over a number of iterations, minimizing the effect of randomness. As the framework is never informed about a change in the state of environment, it must constantly monitor the evaluations of these particles to determine if something has changed or not. Therefore when we determine the worst sub-swarm we also check if it has found its best particle within the last $T_x$ iterations (step 3.e.ii in Table 6.1.). This check allows sub-swarms that are improving their fitness to not be reinitialized. The reason being that any sub-swarm that is currently improving itself should be allowed to improve even more. Once a sub-swarm has stopped improving and is currently the worst sub-swarm then it becomes an ideal candidate for exploration.

Since we use the accumulated fitness of the past $T$ iterations, only the sub-swarms that are improving (or have improved in the last $T$ iterations) are rated higher. A sub-swarm that was the best in a previous state of the environment and now relies on strategies that have become obsolete is forced to abandon its outdated methods and explore new opportunities. It should be noted that only the bottom two sub-swarms are checked and if both of these sub-swarms are improving then no candidate is selected for further exploration. This limit avoids the algorithm becoming a random search and to focus on improving current strategies via evolution. Exploration is one part of the process of continuity of learning and as explained in chapter 5, it must go hand in hand with other processes. Table 6.2 & 6.3 present the values of parameters involved in this continuous learning process.
6.2.3 Experience pool policies

We do not allow duplication in the pool because it was experimentally found to be less desirable. While adding to the socio-experience pool two different strategies were tested.

- After \( T_e \) iterations, add a solution to the pool only if it had attained fitness higher than a certain threshold. If more than one has attained fitness higher than this threshold then choose one with the highest fitness amongst them.
- Add the best solution to the pool after \( T_e \) iterations regardless of their fitness values.

For removal of strategies from the socio-experience pool we tested two policies

- Remove the oldest policy from the pool
- Remove a policy randomly

Step 3.f of Table 6.1. sets the decay value. This decay value is used to determine how old a strategy is (the oldest strategy will have the highest decay value).

<table>
<thead>
<tr>
<th>Table 6.2 Learning Algorithm Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P )</td>
</tr>
<tr>
<td>( c_1, c_2 )</td>
</tr>
<tr>
<td>( w )</td>
</tr>
<tr>
<td>( r_1, r_2 )</td>
</tr>
<tr>
<td>( v_{\text{max}} )</td>
</tr>
<tr>
<td>( T )</td>
</tr>
<tr>
<td>( T_e )</td>
</tr>
<tr>
<td>( T_x )</td>
</tr>
<tr>
<td>( r )</td>
</tr>
<tr>
<td>( E )</td>
</tr>
</tbody>
</table>
Table 6.3 Switching between environment states for socio-experience comparison

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Environment Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-500</td>
<td>Creature Avoidance</td>
</tr>
<tr>
<td>500-1000</td>
<td>Terminal Capture</td>
</tr>
<tr>
<td>1000-1500</td>
<td>Creature Avoidance</td>
</tr>
<tr>
<td>1500-2000</td>
<td>Terminal Capture</td>
</tr>
<tr>
<td>2000-2500</td>
<td>Creature Avoidance</td>
</tr>
</tbody>
</table>

We now present our experiments, results and discussions.

6.3 Experimental Setup

For our experiments we will first test the learning ability of the framework in a perfect environment and gradually move towards an imperfect one (i.e. increase the complexity of the problem). In the current set of experiments we test the ability of the framework to control the movement of the humanoids.

In each state of the environment, humanoids have to perform a certain task. They are evaluated on the basis of their performance of the assigned task. Humanoids are never informed about the task they have to learn or what is wrong with their current learning. The information provided to them is their current fitness evaluation. Using this fitness evaluation the framework must determine how to deploy the 3E's of learning (mentioned in Section III). It should also be noted that the performance of an agent depends on two factors $\delta_i^T$ and $\zeta_i^T$. $\delta_i^T$ represents the knowledge (or ability) of the particle $i$ at time $T$ and $\zeta_i^T$ is the impact of environmental factors on $I$ at time $T$. Both of these factors play a pivotal role in determining the fitness of a humanoid (or particle). Consider for example the terminal capturing environment state. Each humanoid has to capture as many terminals as possible within a certain time limit. As mentioned in the algorithm each humanoid (or particle) creates its clones in the environment and the number of terminals captured by a humanoid is the sum of the number of terminals captured by these clones.
The number of terminals captured by a clone depends primarily upon its ability to move towards its closest terminal. Using an ANN this ability is improved over a number of generations. However this is not the only factor that affects the fitness of the humanoid. Other factors, such as the average distance of the clones of the humanoid from their target, number of creatures in the environment etc. also play a role in the final outcome of the clones present in the environment. The effects of these external factors cannot be ignored. A humanoid with a mediocre survival strategy may win against a highly skilled strategy just because the former faced fewer creatures than the later. Due to the random placement of terminals, clones and creatures, some clones will have a better opportunity of capturing terminals than others. We now present the results of our experiments.

### 6.3.1 Artificial Neural Network

![Image of ANN Architecture](image)

For the current experiments we have only one type (or species) of agents which have the capability to sense the environment and move. However, it may be noted that the environment has the potential to support a multitude of agent types (or species), where each type can have some unique abilities or characteristics that differentiate it from others. Each agent is an ANN with fixed architecture shown in figure. 6.2.
Each of the agents receives 12 inputs: distance and angle from 2 closest terminals, distance and angle from 2 closest agents, and distance and angle from two closest creatures. The distance is Euclidean distance and the angle is relative to the current position of the agent. There are two outputs of the ANN and both of them give a number between 0 and 1. The first output is converted into number of steps to move (0 to 10) by multiplying the raw output by 10. The second output is converted into angle of movement, in degrees, by multiplying it by 360. Hidden layer consists of five neurons, and sigmoid activation function was used for all neurons. The architecture is not tested for optimality and has no other justification except that it seems reasonable and was found to be experimentally working.

### 6.3.2 Parameters of PSO

The PSO velocity update equation depends upon a cognitive learning rate, social learning rate (collectively known as acceleration coefficients), previous velocity, personal best, global best, and random components. The acceleration coefficients $c_1$ and $c_2$ are also referred to as trust parameters where $c_1$ is the trust a particle has on itself and $c_2$ is how much a particle trusts other members of the swarm. While in most applications $c_1$ is set equal to $c_2$, the exact value is problem dependent. The random component allows for probabilistic behavior and in some cases saves the swarm from stagnation (moving back and forth between two positions). Hence a particles’ trajectory is mainly controlled by three factors

1. Previous velocity $v_i(t)$ of the particle
2. Distance of previous location $x_i(t)$ to the particles personal best location $y_i(t)$
3. Distance of previous position $x_i(t)$ to the best position found by the population $\hat{y}_i(t)$.

The personal best and the global best positions are located by the swarm during execution and as such cannot be manipulated. The velocity of the particle can be tuned for optimum exploration and exploitation of the search space. The inertia weight $w$ is utilized to adjust the influence of previous velocity and to balance the exploration and exploitation abilities of the particles. For this reason, we believe the setting of $w$ and
 Researchers have found that a large inertia weight leads to better global exploration and smaller values lead to better local exploration ability. Global exploration helps in avoiding premature convergence. Local exploration confines the swarm to extensively search its current locality resulting in convergence. An optimal value of $w$ is problem dependent. Different algorithms have been proposed for adjusting the value of $w$. Linearly Decreasing Weight (LDW) is widely accepted and used. In LDW, the inertia weight is gradually decreased from a higher value. The rationale being, in the initial iterations, particles should be allowed to search a wider area of the search space while in the final iterations they should focus on better exploration of potentially good areas by using smaller values of $w$. For our problem, linear and time based adjustments are ineffective. This is because the assumption of a perfection of environment is no longer valid. Any adjustment to the inertia weight must consider that the learning of the swarm is independent of the current number of iterations or time.

Time varying parameters are used to exploit a certain assumption about the learning of a particle in an environment “the more time particles spend in the environment they more they learn about it”. For example, time varying inertia weight has been used by researchers to allow the swarm to concentrate on global exploration during early iterations, while focusing on local exploitation in the final iterations. This phenomenon works in environments that are generally stable and where the number of iterations has a positive correlation with the current learning performed by the particles. However, in case of imperfect environments, where the environment can change at any time, such a correlation does not exist. Since the environment can change at any time without any warning, we cannot assume that particles have learned something about their environment at any iteration. Here if the environment changes to something completely different at iteration 2000, all previously performed learning will become outdated and previously learned strategies will become obsolete.
For this reason we believe time varied and linearly changing parameters are not suitable for imperfect environments. These environments require a more adaptive approach. For this reason we set the inertia weight to high (continuously motivating global exploration). This allows the sub-swarms to search new areas in order to develop improved strategies. However, $v_{max}$ is changed adaptively (on the basis of current fitness of the sub-swarm). This allows the poorly performing sub-swarms to jump to other areas while better sub-swarms are motivated to exploit their current potentially good locations (due to restricted velocity).

Furthermore by using a fixed ring topology (figure 6.1) where particles are allocated to sub-swarms based on their index numbers (and not locality or position of the particle), we make sure that sub-swarms are not disturbed by stray particles.

Table 6.2 lists the values used for different parameters of the modified PSO used in our experimental setup.

6.3.3 Computational Cost and Implementation Details

The presented environment and the modified PSO algorithm were coded using C++ language. For debugging, compiling and other coding related issues Visual Studio .net was used. The total lines of code varied from 3000 to 5000 (depending upon the nature of experiments run). An OpenMP based multithreaded approach was used to run the experimentation. On average a single iteration required upto 10 minutes (without graphical simulation) and upto 30 minutes with graphical simulation. Graphical simulations usually resulted in days of processing.

Measuring the computational efficiency of the framework is not straight forward. This is because the framework is generic in nature and the actual computational cost will depend upon number of things, to name a few:

1. The nature of the implementation environment, whether single processor or multi processor machines are available.
2. The choice of underlying evolutionary algorithm
3. The representation of the solution
4. Domain (or range) of the search space

It must also be noted that whether the solution is designed for a real-time system or the game allows for time for the framework to train its solution (initially) while later on the training will be done in background in response to user’s actions. As for the computational efficiency if the system is a real time system, then of course, there is an efficiency issue. However, the framework is flexible and we could adopt a parallel processing approach, allowing the learning algorithm to run on many processors, perhaps, for days (even weeks). Since each sub-swarm is independent of others it is also possible to allocate a sub-swarm to a processor for its learning. It should be noted that the framework is flexible enough to enable these things to be considered by the end user. The framework and the modified PSO presented here serve as a basic platform for creating systems that exhibit automated learning and adapt to the demands of their learning environment without being problem or approach dependent.

### 6.4 Perfect Environment

The purpose of this experiment was to test the learning ability of the framework when its surrounding environment does not change. Humanoids were placed in the environment and were allowed to move around. The job of the framework was to learn the goal set by the current state of the environment. No a priori knowledge is assumed about the environment and everything must be learned by the framework on its own. We used the environment states along with their fitness function mentioned in chapter 5. Figure 6.2 shows the results of this learning activity.

Figure. 6.2 (a) shows the average and maximum number of terminals captured by the sub-swarms in 1000 iterations. Figure. 6.2 (b) shows the average and minimum number of times the humanoids died at the hand of creatures in 1000 iterations. Figure. 6.2 (c) presents the results of training humanoids to capture creatures in 1000 iterations. It
is evident from Figure. 6.2. that the complexity of these tasks is not the same (which is usually the case in real-world scenarios). Tasks may vary in their complexity and generally the more complex the task the more time required to learn it.

### 6.4.1 Evaluation Criteria to determine learning ability

It is apparent from Figure. 6.2 that we did not allow the sub-swarms to converge to a single point. This is in correlation with everyday life problems. We have to learn to do our best within a time frame. Agents in the imperfect environment must also learn to do their best with the available information within a certain time. The environment can change at any time without any warning; hence convergence to an optimal (or ideal) point is less likely to occur. This along with the random and unpredictable nature of the environment gives rise to a new problem.

*How do we determine if learning has occurred? Or in other words how do we determine if the humanoids have learned anything about their environment?*

To answer this, we opted for a competitive co-evolutionary approach. After 1000 learning iterations, the best found solutions from each environment had to compete against randomly generated solutions. If the average performance of a trained solution is better than random solutions than it can be deduced that some learning has taken place. If the trained solution loses to the random solutions then no learning has taken place and the best solution was just a lucky fluke.

Figure. 6.3 presents the competition of the best solutions from the training phase with a 100 random solutions. In all the cases the trained solutions performed better than the random solutions. This demonstrates that some learning (or improvement) has taken place. If candidates were given more time, it is likely they would have improved their abilities ever more.
6.5 Coping with Imperfection

We now move on to test the learning ability of the framework in a changing environment. For this test humanoids were allowed to evolve for 1000 iterations during which time all humanoids faced the same environmental conditions. After every 1000 iterations, the environment changed its state from its current state (Section IV B Phase 1) to the next state (Section IV B Phase 2) without any warning to the humanoids. The results of this experiment are shown in Figure 6.4. The results presented in Figure 6.4 are normalized to get a clear picture of the learning trend. (i.e. the number of terminals acquired shown at each point in the figure as the actual attained terminals divided by the maximum number of targets attained by the humanoids, likewise for captures and deaths).

Since in the first two phases of the environment the humanoids are not given any information about how to capture the creatures hence for those two stages the number of captures remains at zero. The first phase of the environment is the terminal capture phase and humanoids had to learn the ability to capture as many terminals as possible within the time limit (ignoring all other factors like being killed by creatures). Each time a creature kills a clone (of the humanoid) number of deaths is incremented for that humanoid and the clone spawns at some random location in the environment. It is clear from Figure 6.4 that when the humanoids were only focused on learning to capture terminals their clones sustained a high death rate. Since the fitness of the humanoids was not affected by their death ratio they simply ignored the presence of the creatures.

At the first environmental change fitness calculation policy changes. Now fitness is calculated on the basis of the number of times a humanoid dies (a higher number of deaths results in lower fitness). Once this new information is made available to the humanoids, their behavior changes. They adapt according to this new challenge and abandon their outdated policy of terminal capturing and start to avoid being killed by creatures. Since the fitness function employed in second phase does not take into account the number of terminals captured the humanoids focus their learning on creature
avoidance and abandon their outdated strategies. The number of terminals captured in this phase are merely chance captures and are not intentional (as terminals are no long important for humanoids).

None of this information about the change in the environment or the fitness function is transferred to the humanoids. They detect and adapt to changes on their own. In the third phase new information is made available to the agents about capturing the creatures (two or more agents can cooperate to capture a creature, however a single agent who wanders too close can still be killed by a creature). This new information leads the humanoids to change their strategy yet again and retaliate against creatures. Humanoids now know how to run away (if outnumbered) or capture (if they outnumber creatures) consequently the number of deaths of humanoids also goes down. By this time the entire focus of humanoids learning are creatures since the fitness function only takes into account the number of creatures captured by each humanoid. They ignore all terminal capturing activity (as evident from last 1000 iterations in Figure. 6.4).

### 6.6 Retaining historical lessons

In all the experiments so far none of the environment states repeated themselves. In Figure. 6.5 we present results of what happens when the environment reverts back to a previous state. For this experiment humanoids were given a task to learn in the environment, after every 500 iterations the task changed. Humanoids were never explicitly informed about when the task would change or what the new task would be. The environment started from the creature avoidance phase (i.e. humanoids had to avoid getting killed by the creatures) and then the environment changes to terminal capture phase. In this manner the environment continues to switch back and forth between the two phases every 500 iterations. (Table 6.3)

Figure. 6.5 (a) shows the results of this experiment without the presence of a socio-experience pool. It is evident from the figure that humanoids had to re-learn their older ability after each change. This essentially means that the humanoids abandoned (or
forgot) their outdated abilities once they were not required. Although it is desirable to abandon obsolete strategies it is, however, not advantageous to forget all previous learning (especially if there is a chance the environment can revert back to a previous state). It must be clarified that we do not want the framework to assume that a previous state may be repeat itself, rather our focus is what if the imperfect evolutionary environment changes to an older state. In such cases (e.g. re-emergence of an old enemy) a history of previously learned strategy could prove useful. Although the humanoids have the ability to re-learn these strategies it would be useful if these used their initial knowledge. In this manner a socio-experience pool serves two benefits.

1. Acts as a storage pool to maintain a historical record of strategies learned in the past, which becomes useful whenever these old scenarios (or similar ones) are faced in future.

2. Since the pool can be utilized at any time by the agents and it contains the best strategies learned so far by the entire population, it also helps in dissemination of knowledge between the humanoids, where the weak solutions can use this information to improve their current strategies.

Figure. 6.5(b) highlights both these points. Not only did the agents show a better learning pattern in their initial learning (0-500 iterations of Figure.6.5 (a) vs Figure. 6.5 (b)) but they also show improved learning at later stages when the environment reverts to an older state (iterations 1000-1500 and 2000-2500), demonstrating that the presence of a social-experience pool leads to an improvement in the learning ability of the framework.

While adding a strategy to the socio-experience pool based on a threshold value has the benefit of putting only the elites in the archive, so that when we reinitialize a strategy from the archive we know that it has been reinitialized with a mature strategy. If the re-initialization from archive hasn’t improved the fitness of the worst strategy it is not because the strategy used form the archive lacked the skill. Instead an unsuccessful re-initialization from the archive means that the current environment is different from environment in which that strategy was evolved and the strategy is not suited for this
environment. However, this does have the downside of not adding any strategies to the archive if the threshold is set too high and/or the environment changes before any of the strategies is mature (evolved) enough. In such a case the whole archive will be useless.

While adding the best strategy after every $T_e$ overcomes the downside of the previous approach i.e. we can be sure that the archive will not be empty. However, this will result in the other extreme i.e. too many strategies fighting for a spot in the archive resulting in constant overwriting and regular removal of strategies from the archive. Another downside of this approach is replacement from archive does not ensure that the archived strategy was actually mature enough to have positive effect on the evolution, e.g. suppose we filled the archive with the best strategy at time $T_1$ and the fitness of the best strategy was 0.3 (fitness values range from 0 – 1, 1 being best). If at time $T_2$ all the strategies are at fitness higher than 0.3 then replacement from archive is not likely to improve the fitness of the worst sub-swarm. It may even lead to poor performance.

Similarly there are issues with removal policy. There is a certain probability that any strategy from the archive could work in the current environment and there is always a chance that the current environment state has never been experienced before and none of these strategies will work. We believe that, given the unpredictable and dynamic nature of the environment, and the adaptive and continuous nature of learning required, a more adaptive policy should be deployed depending upon the complexity of the problem. We are currently working on allowing the framework to develop, monitor and evaluate its own strategies for addition and removal to/from the archive.
Continuity of Learning in an Uncertain & Dynamic Environment of Imperfect Information

(a) 

(b) 

(c)
Figure 6.2  Learning Ability of CLF in Perfect Environments

(a)  
(b)  
(c)  

Figure 6.3  Comparison of CLF Humanoids against Untrained Strategies
Figure 6.4 Continuity of learning in an imperfect environment
6.7 Adding a new dimension to learning

In all the tasks the humanoids had to learn, they only had to focus on one thing at a time. This fact is highlighted by the statistics shown in Figure 6.6. In Figure 6.6, St-Terminal stands for the best strategy learned by the humanoids to capture terminals. Likewise St-Deaths is the best strategy to avoid deaths and St-Captures is the best strategy for capturing creatures. The statistics represent the accumulated values of the best strategies in 3000 simulations. It is clear from Figure 6.6 that while each strategy performs well in its own area of specialty, its performance in other dimensions is usually poor. For example, a strategy good at capturing terminals takes too many hits and manages to capture only a few creatures (which is purely by luck). Likewise a strategy developed to avoid deaths is not good at capturing terminals or creatures.

One of the best death avoidance strategies evolved by the agents was to linger around the corners and jump to the other side when in danger of being eaten by a creature. Creatures, being scripted in nature, always move towards the closest clone and cannot jump to the other side (it was never thought that agents would learn such a clever trick). This strategy not only avoids death but also shows the humanoids understanding and exploitation of the environment that surrounds it. Best creature capturing was to hunt in packs trying to corner the creatures. Neither one of the death avoidance or creature capturing strategies pay any attention to terminal capturing (as shown from their stats in Figure 6.6 and Table 6.4).

However in most real-world scenarios human beings have to learn more than one thing at a time. Dimensions may be added (or subtracted) from the current learning ability at any time without any forewarning. Hence, we now test the ability of the framework to perform this dynamic, multi-dimensional learning using the following experiment.
For 500 iterations humanoids were allowed to learn how to capture targets. Their fitness was evaluated on the basis of the following fitness function.

\[ F_i^T = \left( \min \left( 1, \frac{t_i^T}{mt^T} \right) \right) \]  \hspace{1cm} (6.5)

This meant that the entire focus of their learning was to capture terminals ignoring all other factors (including the presence of creatures). After the initial 500 iterations, their fitness function was changed to accommodate two factors, capturing maximum number of targets while minimizing their deaths (at the hand of the creatures). The fitness function for this phase was

\[ F_i^T = \left( \min \left( 1, \frac{t_i^T}{mt^T} \right) + \max \left( 0, \frac{md^T - d_i^T}{md^T} \right) \right) \] \hspace{1cm} (6.6)
Here $F_i^T$ is the fitness of the $i^{th}$ humanoid which is the average of two entities.

The minimum of 1 and the total number of terminal points captured by the humanoid $t_i^T$ divided by the maximum terminals allowed for capturing $mt^T$ at time $T$. And the maximum of 0 and $md^T$ (the max number of time humanoids are allowed to be killed by the creature) minus $d_i^T$ is the number of times humanoid $i$ was killed by the creatures divided by $md^T$. Hence we add a new dimension to the fitness function at runtime without informing the humanoids of this change. Framework must detect this change in the fitness function, abandon obsolete strategies and formulate new strategies based on the information available in the environment. The results of this experiment are shown in Figure 6.7.

In the first part of the experiment (0-500 iterations) humanoids ignore their deaths. In the latter half of the experiments (500-1000) iterations they now have to learn to maximize the number of their terminal captures and also minimize their number of deaths. Since both parts carry equal weight the immediate response from the agents is to minimize the number of their terminal captures which also minimizes their deaths. Minimizing deaths will lead to a better fitness. But this technique can only get them so far and to improve their fitness even further they have to eventually capture more terminals. Hence in the latter half of the 2nd part of the experiments humanoids adopt a new strategy of focusing their efforts on capturing terminals that are close to the corners. It is evident by the gradually increasing difference between the two curves that the humanoids have learned to deal with this new and (relatively more) complex state of the environment.
6.8 Summary

In this chapter we presented the results of our experimentation. We created different scenarios to test the learning ability of our framework. The results of our experimentation show that our framework is capable of dealing with uncertain and unseen scenarios. Agents residing in an imperfect environment can use the framework to adapt according to the changes and challenges presented by their surrounding environment. We have tested the framework in different scenarios starting by its performance in a perfect environment and moving on to a changing and uncertain environment. Our learning algorithm based on our continuous learning framework is also able to successfully handle multiple objectives. Results show the ability of the algorithm to effectively couple with new dimensions of learning which may be presented to the learning agents during their evolutionary process.

In the next chapter we present how this research can be applied to commercial video games. We apply our learning algorithm to the problem of predictability in video games. With experimentation we show how adaptive intelligence in games is able to learn new abilities and games on its own. We believe this ability will lead to more improved replay values for video games.
Chapter 7: Adaptive Game Intelligence

Application of continuity of learning

You have to be fast on your feet and adaptive or else a strategy is useless.
- Charles de Gaulle

7.1 Introduction

In the previous chapter we presented results of detailed experimentation we conducted to test the learning ability of our continuous framework and the learning algorithm we designed based on the framework. Results show that the continuous learning framework is able to effectively handle imperfection and incompleteness of the environment.

Due to the inherent mimicking of real life scenarios the importance of imperfection in evolutionary systems is now well established. The significance of learning within such environments cannot be overstressed. Although this research can be applied to a wide array of problems and environments, in this chapter, as an example problem, we will present how our research can be applied to the quandary of predictability in video games.

Is it possible to develop a computer program that can learn to play different video games by itself depending on interactions with other players? Can video game characters learn new skills through interacting with human players? Can we make video games more interesting by allowing in game characters to behave according to human player’s strategy?

These are just some of the question that video game developers and AI researcher are working on. In this chapter we present an evolutionary approach based on our continuous framework (presented in chapter 4), that uses a modified particle swarm
optimization (PSO) and artificial neural networks (ANN), to answers these questions by allowing the agents to respond to the changes in their surroundings. Video games usually require intelligent agents to adapt to new challenges and optimize their own utility with limited resources and our approach utilizes adaptive intelligence to improve agent’s game playing strategies. In this chapter we present how our research is directly applicable to video games research and evolutionary gaming. Our approach can be further extended to develop intelligent systems for exploitation of weaknesses in an evolutionary system.

7.2 Predictability in games

Since the 1950s computer game playing has caught the eye of artificial intelligence (AI) researchers. Being a source of enjoyment, games have also challenged our ability to reason and learn. By providing a virtual and controlled environment, computer games allow us to monitor interactions of game playing agents and modify their actions. This opened new realms for AI research. These virtual environments can (and have) been used to mimic natural phenomenon and increase our understanding. Providing competitive and dynamic environments, games are ideal test beds for the evaluation of computational intelligence theories, architectures, and algorithms.

However traditional game playing programs have ignored adaptability in the learning ability. In this research we argued that an intelligent game playing agent is not designed to play a specific game, but rather learn from its environment. A number of AI game players have claimed themselves successful because they have play games to a level of human experts. This, however, does not mean they are intelligent. Given the number of humans, how many world chess champions, or checkers experts are there? Is the rest of the human population unintelligent? Another issue game playing agents have been facing is the predictable behavior of game players. Predictability decreases the replay value of games. We argue that intelligent and adaptive behavior of AI will lead to more interesting game playing. In this chapter we will focus on development of such adaptive AI based game player.
Freed et al. [208], argue that generating human-like players will make games more appealing and enjoyable. This idea is further supported by investigation of the correlation between believability of Non-playing characters (NPCs) and satisfaction of the player [209]. Iida et al. work on measures of entertainment in board games was the first attempt in this area. He introduced a general metric of entertainment for variants of chess games depending on average game length and possible moves [210]. Crispini in [211] discusses criteria’s to make simple online games appealing. The outcome of his work hypothesizes challenge, diversity and unpredictability as primary criteria for enjoyable opponent behaviors. Pedersen et al. [212] presents criteria that collectively define interest on any predator/ prey games are as follows:

1. When the game is neither too hard nor too easy.
2. When there is diversity in opponents’ behavior over the games.
3. When opponents’ behavior is aggressive rather than static.

Other works that deal with optimizing player satisfaction and modeling player experiences are [213 - 215]. Some others applications of game theory are presented in [216,217]. Our work is different from all previous work. We believe, to the best of our knowledge, this work has not been undertaken by anyone else. Our approach to game playing has the following main features.

1. Automated game learning from scratch without pre-injected knowledge.
2. Learning to play games based on exposure to new players.
3. The AI itself decides when it has to invoke new learning and discard previously learned (but now obsolete) information.
4. The ability to learn the game to the level of the human player, making the game more interesting to play (rather than to challenging or to easy).

The learning algorithm works in the following way (Table 7.1). We have used a Particle Swarm Optimization (PSO) variant as our evolutionary algorithm. Our PSO algorithm is based on the lbest PSO with a change in the sub-swarm connection
architecture. Evolution starts off by creating a random population of $P$ particles; these particles are adjusted into $S$ overlapping sub-swarms. Each of our sub-swarms contains 5 particles (total 8 sub-swarms). Each sub-swarm is connected to the next by the corner particles. In this manner these 8 sub-swarms create a ring like formation, where data is shared via the connecting particles. Each of the particle (or agent) is allowed to play a series of games against the scripted player. Number of wins and losses is monitored and fitness of the agent’s strategy is calculated on the basis of these statistics. Once all agents have played the game, we can find the $gbest$ (global best particle i.e. the particle with highest fitness value in the entire population) and $lbest$ (local best particle i.e. the particle with highest fitness in a sub-swarm). These values are then used in the PSO equations [26].

To limit the jump area of a particle we clamp the velocity of the particles. This is what we call the exploration parameter. Agents with better fitness values should be forced to extensively search their current locality while agents with poor fitness values should be allowed to jump greater distance in hopes of finding better search areas. After a number of iteration, the population must be forced to explore new opportunities that may be available in the environment. For this reason we determine the worst performing sub-swarm and force it to reinitialize its strategy. Care must be put in to avoid reinitializing a sub-swarm that is currently improving its fitness value.

### Table 7.1 Automated Game Learning Technique

1. Initialize parameters
2. Create randomly initialized $S$ sub-swarms
3. While iteration $< T$
   a. Allow agents to play the game
   b. Calculate fitness of the current strategy
   c. Locate $gbest$ and $lbest$ particles sub-swarms
   d. Adjust exploration parameter
   e. Update positions of the particles
   f. After every $T$ iterations explore new opportunities
      i. Detect worst performing sub-swarm
      ii. Force worst performer to create a new strategy
   g. Increment iterations
4. Go to Step 3
7.3 Automated Game Learning

For our experimentation we choose two games Connect4 and Pacman (we modified the standard version of the games for simplicity). In our current experimentation we do not allow neuro-evolution hence we choose an ANN that was applicable to both of these games. The entire board was input to the ANN (we use same sized board for both games). In both games empty squares were given an input of 0, opponent (or monster square) were set as -1 and own squares were set to 1 (for Pacman squares with dots were set as 1). The learning agents are never told which players or game they are playing. They are just informed about the fitness evaluation of their current strategy. Using this fitness evaluation they must take steps to improve their fitness value in order to survive. For Connect4 each agent plays against a scripted player. Scripted players were intentionally set to a mediocre level (playing not too stupidly or at expert level). Playing against a mediocre level player presents agents with a suitable challenge by forcing them to learn to play the games rather than apply random strategies. A few things must be noted

a. Our focus is on enabling the agents to learn to play new games without relying on human training.

b. We are not trying to create agents that are an expert at the games they play hence an ANN that works is more suitable than the most optimal ANN architecture.

c. We do not allow agents to find the optimal game playing strategy; instead they must learn to play the game as much as they can within a limited amount of time (or iterations).

d. In order to test if the agents have learned anything about a game, they play against random strategies. If their performance is better than these random strategies then we can deduce that (some) learning has occurred.

e. Fitness values are normalized to highlight the underlying learning pattern and make the learning for different games comparable with one another.
For our first experiment we allow the agents to learn to play the game of Connect 4 for first 1000 iterations and then Pacman for next 1000 iterations. During this time they may not come-up with optimal game playing strategies but they do learn to play the game at a beginner level. This learning occurs from scratch. The learning approach for our first experiment is the standard PSO algorithm. This algorithm is not adaptive in nature and it ignores the changes occurring in its environment. The PSO algorithm forces all agents to move towards the best solution in the population. A change in the surrounding, e.g. changing the rule of the game, adding new challenges to the game, etc., can render this old best solution as obsolete. Due to its ignorance of surrounding environment, the PSO algorithm will fail to realize this change and all particles will be forced to move towards the outdated solution. This will lead to a decrease in the population fitness. This fact is evident from figure 7.1.

![Figure 7.1 Ignoring the changes occurring in the surroundings](image)

After a change in the game rules (from Connect4 to Pacman, which requires entirely different game playing strategies), the fitness of the population decreases drastically and never recovers. The only way for the algorithm to require from this fault is to get lucky and find a strategy for the game played after the change by mere chance. For our second experiment, we use the same setup but this time instead of using the standard PSO for learning we use our own learning algorithm. As mentioned earlier our algorithm uses a modified version of the PSO algorithm. Our approach is more focused towards monitoring the environment for change. Once a change occurs, agents modify their
current strategies to better adapt to it. As shown in figure 7.2, after 1000 iterations agents are introduced to new game players who play a different game from the one they have already learned. These new players require the agents to update their strategies in order for the agents to win. Relying on old strategies leads to poor initial fitness values. These fitness values indicate to the algorithm that something about the game has changed. New opportunities must be explored to formulate strategies better suited for this new game.

![Figure 7.2 Reacting to changes occurring in the surroundings](image)

7.4 Summary

In this chapter we have presented an approach to adaptive intelligence which allows intelligent game playing agents to adapt to new challenges and optimize their own utility with limited resources. Our proposed approach overcomes the limitation of traditional approaches which fail to adapt to changes in the environment, our approach with its adaptive nature, allows the agents to adjust their behavior according to new challenges. This gives the agents the ability to learn in the presence of unseen scenarios (like learning to play new games) without relying on any pre-injected knowledge. The whole learning process is automated and stimulated by exposure to new players and situations. We believe this research will lead to better game AI and improving the replay value of video games. The learning algorithm assumes nothing about its surrounding environment, its objectives and this independent nature allows it to be applicable to other domains.
Chapter 8: Conclusion

*The distinction between past, present, and future is only a stubbornly persistent illusion*

- Albert Einstein

An imperfect environment constantly presents new challenges to its inhabitants. These inhabitants realize the imperfect nature of information available to them. They make no assumptions about their environment and rely solely on their learning ability to survive whichever state the environment evolves into. Each individual has a (possibly unique) perspective of the environment. Using this, it tries to optimize its own abilities, utilizing the information currently available to it. Whenever a change appears in the environment these individuals adapt according to this change, thereby improving their chances of survival. In this paper we have presented a framework that gives agents the ability to learn without depending upon any human intervention. These agents make no assumptions about their environment.

Each agent forms a dynamic relationship with its surrounding environment. Using this information it forms a better understanding about its environment and utilizes this understanding to increase its ability to deals with the environment. These agents form a dynamic society where each member contributes to the improvement of the society as a collective. Even the worst performing members of the society are utilized in a manner that they prove themselves to be useful for others. Individuals who have a better understanding of their environment (indirectly) guide others. This collection of seemingly unrelated and unconnected individuals forms a community of agents that works similar to a human community. Agents within this community explore and evolve strategies, adding to their own experience and the communal experience pool.

Using a continuous learning process agents are able to learn in any given environment and their abilities are only limited by the information provided by the
Their surrounding environment can change to anything at anytime without any warning. We developed such an environment which presents new information to its inhabitants, motivating the evolutionary process. The environment is dynamic in nature and new states can be added to it to further test the learning ability of the agents.

Our tests show that the self-motivated learning ability of the agents was successful. These agents not only learned to survive in their new environment but also exploited any weakness of the new environment. These agents learned new abilities on their own. We do not claim that the learning ability of these agents is optimal or that the strategies they evolved are optimal strategies. It was, however, demonstrated experimentally that the agents learned efficiently to deal with any new change in the environment within the given time.

We believe this to be a major step forward toward the realization of our goal of creating more natural-like intelligence. One that is guided by the information available to it and does not rely on a human to train it. These agents can learn to new skills and abilities on their own. We conclude that this system provides a better adaptive intelligence behavior, one that can be applied to other complex problems. As the continuous learning framework is independent of its surrounding environment, it can be applied to other environments.

Much work still needs to be undertaken in this path. Possible future channels of this research include designing better performance evaluation criteria of agents residing in different locations of the environment and establishing individual archive for learning based on personal experience. There are other questions that may still be explored. For example, can we apply the framework to problems in which even the structure or representation of the problem is unknown. In such a problem even the representation would be dependent upon the agents of the framework. How would they handle this? It would also be interesting to compete continuously learning species compete against one another for survival. We are currently working on the integration of better neuro-evolution strategy to the framework so that it can attempt to optimize its architecture as
per the requirements. Better algorithms for addition and removal from the experience pool are also being investigated. Application of this self-motivated, independent, multi-dimensional continuous learning framework upon other practical problems would be interesting.
References


[34] http://www.merriam-webster.com/dictionary/deduction


Continuity of Learning in an Uncertain & Dynamic Environment of Imperfect Information


Focus on the journey, not the destination. Joy is found not in finishing an activity but in doing it

- Greg Anderson