

# **Feasibility and Implementation Study of Adaptive Learning to Agent Based Models**

**Hussein Abbass and Michael Barlow**  
**abbass@cs.adfa.edu.au, spike@cs.adfa.edu.au**

School of Computing Science  
Australian Defence Force Academy  
University of New South Wales  
May 2002

# Contents

<b>Table of contents</b>	<b>i</b>
<b>List of figures</b>	<b>ii</b>
<b>List of tables</b>	<b>iii</b>
<b>Acknowledgement</b>	<b>iv</b>
<b>List of Abbreviations</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Purpose of This Document . . . . .	1
1.2 Introducing the Project . . . . .	1
1.2.1 Mana . . . . .	1
1.2.2 Research Objectives . . . . .	3
1.3 The Movement Algorithm . . . . .	4
<b>2 Machine Learning With Agent Based Distillations</b>	<b>5</b>
2.1 Concepts in machine learning . . . . .	5
2.2 Is learning an optimization problem? . . . . .	6
2.3 Potential use of this reports' outcome . . . . .	6
<b>3 Outline Of Approach</b>	<b>8</b>
<b>4 Preparing The Problem</b>	<b>11</b>
4.1 Re-visiting the Move Equation . . . . .	11
4.2 Criteria of Success . . . . .	11
4.3 Pre-processing . . . . .	12
4.3.1 Translation . . . . .	12
4.3.2 Determining the Center of Forces . . . . .	13
4.3.3 Reflection . . . . .	14
4.3.4 Inputs . . . . .	14

<b>5</b>	<b>Potentially Suitable Learning Algorithm</b>	<b>16</b>
5.1	Markov and Hidden Markov Models (HMMs) . . . . .	16
5.2	Artificial Neural Network . . . . .	18
5.3	Decision Trees . . . . .	18
5.4	Evolving Neural Networks Approach . . . . .	18
5.4.1	Definitions . . . . .	19
5.4.2	The Evolutionary Neural Network Algorithm . . . . .	19
5.4.3	Applying the method in MANA . . . . .	21
<b>6</b>	<b>Insight Extraction From The Learner</b>	<b>22</b>
6.1	Boolean Rule Extraction . . . . .	22
6.2	Validity Interval Analysis . . . . .	23
6.3	Rulex . . . . .	24
6.4	C-Net . . . . .	25
<b>7</b>	<b>Implementation - A Follow-on Study?</b>	<b>27</b>
7.1	Issues in Implementation . . . . .	28
<b>8</b>	<b>Conclusion And Future Work</b>	<b>30</b>
8.1	Research Questions . . . . .	31
	<b>References</b>	<b>33</b>

# List of Figures

3.1	The 2-phase process in using a machine-learner: training and application. . . . .	9
5.1	A 6-state, left-to-right HMM . . . . .	17
6.1	The C-Net algorithm . . . . .	25
6.2	The C-Net conceptual representation. . . . .	26

# List of Tables

1	List of abbreviations used throughout the report . . . . .	vi
---	--	----

## **Acknowledgment**

In acknowledging our debt to all those who helped to make this project a memorable intellectual experience, we wish firstly to record our deep gratitude to Dr. Andrew Gill for initiating this project and being a continuous source of support and guidance all over the course of this research.

Our thanks to all the staff members of the School of Computer Science, at the Australian Defence Force Academy, University of New South Wales. A special mention goes to Charles Newton for his continuous support.

We are indebted to our families who fed our souls during those hard times while undertaking this research with constant encouragement, patience, understanding, and love.

Our final acknowledgment of sincere gratitude goes to the Commonwealth and The Defence Science and Technology Organization (DSTO) for providing us with the necessary financial support to undertake this research.

## List of Abbreviations

---

ADFA	Australian Defence Force Academy
ANN	Artificial Neural Networks
CAS	Complex Adaptive Systems
DOTSE	Defence Operational Technology Support Establishment
DSTO	Defence Science and Technology Organization
EANN	Evolutionary Artificial Neural Networks
EC	Evolutionary Computations
HMM	Hidden Markov Models
LOD	Land Operation Division
MANA	Map Aware Non-uniform Automata
RL	Reinforcement Learning
SA	Simulated Annealing
SI	Swarm Intelligence
UNSW	University of New South Wales

---

Table 1: List of abbreviations used throughout the report

# Chapter 1

## Introduction

### 1.1 Purpose of This Document

Dr. Andrew Gill from the *Defence Science and Technology Organization* (DSTO) approached the *School of Computer Science*, the *University of New South Wales* (UNSW), at the *Australian Defence Force Academy* (ADFA) campus in Canberra, seeking technical advice regarding co-evolution of parameters in multi-agent environments. The School possesses a wide expertise in the area of *Complex Adaptive Systems* (CAS) with its internationally recognized research in *Evolutionary Computations* (EC) and *Swarm Intelligence* (SI).

The project was forwarded to two school members; Dr. Hussein A. Abbass who is an expert in the area of CAS and Dr. Michael Barlow who is an expert in the area of agent distillation. Both Dr. Abbass and Dr. Barlow have experiences with wargaming systems and understanding with the problem domain.

In the rest of this chapter, the project is introduced in Section 1.2 followed with the movement algorithm, proposed by Dr. Gill, in Section 1.3.

### 1.2 Introducing the Project

#### 1.2.1 Mana

*Map aware non-uniform automata* (MANA) [14] is an *agent-based distillation* (ABD) system developed by Roger Stephen and Michael Lauren of the *Defence Operational Technology Support Establishment* (DOTSE) New Zealand. It is one of the most commonly used distillation systems by people performing Operational Analysis - sharing that distinction with SOCRATES and EINSTEIN. MANA not only has an active “user community” but it is well supported by its authors - a number of additional features and new versions of MANA have appeared. That situation is likely

to continue for the foreseeable future. [12, 13]

Agent-based distillation systems are abstracted simulations of conflict between two or more forces. The features that, taken together, distinguish them from more traditional “constructive” simulations are:

1. **Agent-based:** Battlefield entities are represented by agents.
2. **Simplicity:** Each agent follows a simple set of rules.
3. **Emergence:** Battlefield behavior emerges from the interaction of the agents - it is not “pre-programmed” into the agents.
4. **Abstraction:** Battlefield entities have their capabilities represented in an abstract manner. There is not necessarily a 1-to-1 mapping between a real-world entity (e.g., tank) and one in the distillation. Similarly, measures of capability within the simulation tend to be unitless.
5. **Ease-of-Use:** Distillation scenarios are quick and easy to design and setup.

ABDs have received considerable interest from sectors of the Operational Analysis community in the last two to three years. This is attributable to a number of factors:

1. **Rapid Prototyping** - The ability to rapidly explore a large parameters space so as to obtain insights necessary to guide more detailed analysis. In this context ABDs are seen as the most abstracted element in the hierarchy of simulation that is sometimes known as operational synthesis.
2. **Nonlinearity** - The apparent ability of distillations to capture the essential and well-known non-linearity of modern battles.
3. **Intangibles** - The potential of ABDs to begin to quantify the effects of intangibles such as training, morale, and leadership upon battle outcomes.
4. **Coevolution** - The ability of ABDs that sees the actions of both sides alter in response to their perception of the enemy’s actions. That is agents do not make decisions in isolation but are influenced by their knowledge of teammates’ and enemies’ actions.

MANA is a “2nd generation” ABD expanding on the principles validated by ISAAC and EINSTEIN [9]. Its origins are still strongly rooted in the cellular automata approach but it incorporates a number of additional features. It is based on CAS rather than conventional combat models. The differences between the two are presented in [11]. MANA focuses on modelling squad-level scenarios, though it is capable of modelling more strategic conflicts also. One of the chief features employed to model these tactical-level conflicts is the incorporation of a group situational-awareness map. This

is a shared map that represents the knowledge agents on a side possess about their environment. As noteworthy elements in the world are discovered, these are stored in the situational awareness map. Once they have been added, they decay as time passes so that more recent ‘memories’ are given greater consideration than older ones, and ‘memories’ of a certain age are disregarded or forgotten.

### 1.2.2 Research Objectives

Adaptive learning is one of the three stated research thrusts of the US Marine Corps Project Albert research initiative. However, to date it appears to have received little attention but also offers significant potential to take the project to the next step. ADFA has previously researched this area, in terms of the architecture and frameworks, and the mathematical techniques appear to exist (though probably require some modifications) – however it remains to actually implement the concepts within a multi-agent simulation. The project seeks to engage appropriate researchers to develop mathematical or computational techniques for adaptive learning within multi-agent simulations.

Adaptive learning seeks to allow each agent to vary the parameters that it has direct control over, during a simulation instance, in order to learn from past experiences and to adopt improved agent behavior. The primary parameters of interest are those that describe the tactics adopted, but also of interest are the parameters that control the state transitions as well as those which describe the movement algorithm used.

The research objectives of the project are as follows:-

1. Initially focus on the mathematical technique of reinforcement learning and determine its suitability to enable adaptive learning within multi-agent simulations.
2. Produce adaptive learning algorithms, either from reinforcement learning or from other appropriate mathematical or computational fields, which can be most easily implemented within multi-agent simulations.
3. Detail the requirements for modifications, if any, to multi-agent simulations to enable the inclusion of the adaptive learning algorithms produced.
4. Provide a description of the potential uses and benefits of the adaptive learning algorithms developed for multi-agent simulations.
5. The project deliverable is a written report detailing the research results against each of the four objectives above.

### 1.3 The Movement Algorithm

In MANA, an agent decides on its next move - to a unoccupied cell within the movement range - using personality weights and behavior modifiers. A penalty function is used to estimate the potential of each move. Let us define the following:-

$Z_{new}$	the new measure for change in penalty by moving to location <i>new</i>
$Z_{old}$	the old measure for change in penalty by moving to location <i>new</i>
$D_{i,new}$	units for candidate location <i>new</i> to entity <i>i</i>
$D_{i,old}$	units for current location <i>old</i> to entity <i>i</i>
$D_{f,new}$	penalty of flag <i>f</i> at position <i>new</i>
$D_{f,old}$	penalty of flag <i>f</i> at position <i>old</i>
$Direction_i$	represents the attraction, (1) if $D_{i,new}$ is less than the threshold and (-1) otherwise.
$W_E$	weight assigned to enemy
$W_F$	weight assigned to friendly flag
$E$	number of enemy agents
$F$	number of friends agents
$\alpha$	parameter between 0 and 1
$r$	parameter between 0 and 1

The move with penalty less than some threshold is selected and ties are broken at random. The equation for calculating the penalty in MANA is

$$P_{new} = \frac{1}{E + F} \sum_i^{E+F} Direction_i \times \frac{D_{i,new} + (100.0 - D_{i,old})}{100.0} \quad (1.1)$$

The following equation is proposed by Dr. Gill and calculates the change in penalty if the agent decides to move from location *old* to location *new*.

$$Z_{new} = \frac{W_E}{E^\alpha} \sum_{i=1}^E \left( \frac{D_{i,new} - D_{i,old}}{D_{i,old}} \right)^r + W_F \left( \frac{D_{F,new} - D_{F,old}}{D_{F,old}} \right)^r \quad (1.2)$$

After calculating the change in penalty for each possible move in the neighborhood, the agent uses the following function to probabilistically decides on a suitable move.

$$P(Move = i) = \frac{\exp\left(\frac{-Z_i}{\tau}\right)}{\sum_{j=0}^I \exp\left(\frac{-Z_j}{\tau}\right)}, \quad i = 0, \dots, I \quad (1.3)$$

where  $\tau$  represents a temperature-like effect as in *Simulated Annealing* (SA) to balance exploitation and exploration of search.

# Chapter 2

## Machine Learning With Agent Based Distillations

### 2.1 Concepts in machine learning

In this section, we will introduce different concepts in machine learning that will be used in the rest of the report.

In machine learning, each input vector is called an *instance*, or *input pattern*, and the output is called an *output pattern*. The input-space is usually mapped to some features for the learning problem; these features construct the *feature space*. The set of all possible models that can map the feature space to the output space forms what is known as the *hypothesis space*. The process of updating the split points in a decision tree, the centroid in a cluster algorithm and the weights of a neural network until the network correctly maps the inputs to the outputs is called *learning*. The learning algorithm is called *learning machine*.

The data are usually partitioned into three sets, a set available for the learning machine for learning the function called the *training set*, a set for validating the model called the *validation set*, and a set for testing the generalization of the model called the *test set*. By *generalization* we mean the ability of the learning machine to give the correct output on unseen data (data which are not included in the training set). The learning machine is said to *over-fit* if it is biased towards the training set instead of learning the underlying function; that is, it memorizes the training set and does not generalize well over the feature space. A simple example to illustrate over-fitting is giving the machine the multiplication table and instead of learning how to multiply two numbers, it memorizes the input output pairs. More formally, over-fitting or over-learning is a situation where the learning machine learns idiosyncratic features of a training set that have no discriminative value in the operational environment.

## 2.2 Is learning an optimization problem?

There are major differences between optimizing a function and learning a function. In optimization, we are usually given the function to be optimized; therefore, there is no doubt in the underlying function. In other words, if we are given two points on the function, we can easily and undoubtedly conclude which point is better. For example, in the case of minimization, given two points  $x_1$  and  $x_2$  for the function  $f(x)$ , it is straightforward to conclude that  $x_1$  is better than  $x_2$  if  $f(x_1) < f(x_2)$ .

In learning, we search for a function which if used to map the inputs to the outputs, the error will be minimum. Unfortunately, finding the function which minimizes this error alone is nontrivial and is not a simple optimization problem because of the following:

1. There is no guarantee that this function is unique; there may exist hundreds of functions with very different characteristics and still do the mapping correctly.
2. Usually, we have a sample of the points and not the whole input domain of the function. This entails that there is always uncertainty whether the mapping we will end up with is the desired function or not.
3. Real life data is noisy. Noise can simply be defined as random variations in the data that we cannot associate its causes with any of the inputs. For example, data collected from sensors is usually noisy. Because of noise, it is not preferred in machine learning to choose a learning machine that maps the data with 100% accuracy. The reason for that is we know in advance that the data has noise and by fitting the data completely, we run into the risk that the machine will not perform equally on unseen data (generalization). In addition, usually we do not have a clue of how much this noise is.

In summary, a machine learning problem is very different from an optimization problem. A technique which can find the optimal solution of an optimization problem very accurately may not be suitable for machine learning; simply because it will over-fit the data.

## 2.3 Potential use of this reports' outcome

The outcome of this project is a learning machine that is able to map certain situations (states) to some actions (decisions). It is like a supervisor who watches how the work is going and tries to undertake corrective actions to get to a final state that is close enough to the desired state.

The learning machines which will result from this project will be used as control mechanisms during the game. They will act as a supervisor who is watching the wargame and decides on how to change the parameter values to reach the goal (winning).

A number of techniques exist to model this virtual supervisor. In *reinforcement learning*, the supervisor policy is some sort of a decision table (called optimal decision policy), when the environment is in state  $i$ , the next state is chosen from this table to be  $j$ . This sort of mapping is very relevant in a grid/cellular environment where each cell represents the agent's state and the transitions between cells represent the agent's actions. However, if the state space is continuous, reinforcement learning would typically require discretizing the state space. This discretization, although may be useful in certain occasions, may not be relevant in modelling the parameters of agent based modelling and wargaming. The reason being these models are chaotic (*ie* small changes may cause large effects). Discretization in this case may hide necessary information in the input space. Other problems in RL include being memoryless method because of its dependence on first degree Markov. Furthermore, the credit assignment problem where it is usually difficult to distribute the reward/penalty on different parts of the system.

The agents incorporating adaptive learning can be used to strengthen the weight of insights derived from OA experiments. Because the agents are "optimizing" their movement on the basis of the world state it might reasonably be argued that they are thus providing a better measure of what real units might do in that current situation, rather than the unchanging approach of current systems - quite unlike a real soldier or officer. There are caveats here though. The learning algorithm will learn to maximize the objective function on the basis of the training examples it sees. Thus those training examples should cover the range of parameter variation that will be experimented over. As an example, a classic OA task might be to consider the tradeoff between firepower and sensor range. This is often performed as a 2D landscape where each point represents a particular level of firepower and sensor range. Were adaptive learning agents to be used for the experiments they should either be trained for each point upon the surface (*i.e.*, a different ANN) or trained with a range of scenarios that incorporate different firepower and sensor range levels.

# Chapter 3

## Outline Of Approach

Incorporating self-adaptation into an ABD requires the integration of a new element into an agent - an algorithm, model, or mathematical function which seeks to adapt the behavior of the agent over time in response to its knowledge of the environment. In most ABDs, including MANA, agent behavior is defined by its movement choice. Hence self-adaptation in MANA becomes a matter of adapting the parameters of the movement function over time.

Formally stated then, self-adaptation in MANA is the process of mapping the movement parameters at time  $t$  to those at time  $t + 1$ . The input, or parameter, to that mapping function is the world state at time  $t$  - or at least the agent in question's knowledge of the world. This is shown in Equation 2.1 below:  $r$ ,  $\alpha$  and  $\tau$  are mapped from their values at time  $t$  to those at time  $t + 1$ , under the influence of the agent's world knowledge at time  $t - w_t$ .

$$f_{w_t}(r_t, \alpha_t, \tau_t) \rightarrow r_{t+1}, \alpha_{t+1}, \tau_{t+1} \quad (3.1)$$

Implementing the self-adaptation is then a matter of defining and implementing the mapping function. There are several choices possible for the nature of the mapping function. One initially appealing approach is to consider a set of simple rules - heuristics. The complexity of the world - number and position of each friendly agent, number and position of each enemy agent, position of agent itself, position of target flag, agent health and combat strength - yields a high dimensional space that the mapping function must learn (the heuristic rules must cover). Writing a simple set of rules which encompassed or abstracted all situations would be problematic at best, even for a very tightly constrained sub-problem.

The alternate approach is to have an algorithm "automatically" learn the mapping - machine learning. The great advantage of this approach is that a human does not need to write a set of rules which define the mapping. Rather, the human selects the machine-learning algorithm to use, the set of inputs to that algorithm, the function which the mapper (learning algorithm) is to optimize (not the same as the mapping function - this is the criteria to use to determine how effective the algorithm is), and a set of training examples. That is the approach pursued in this report.

Building and employing a machine-learner is a two-phase process. The first and critical phase involves constructing the learner. This process is known as training the algorithm - not only are the parameters of the algorithm selected (such as the number and form of inputs, the number of hidden nodes if an Artificial Neural Network is used, etc.) - the algorithm must then learn the mapping from inputs to desired outputs. This is achieved through the use of the training set and the optimization criteria - the parameters within the algorithm (such as the weights on a node in an ANN, or the transition probabilities in a Hidden Markov Model) are altered so as to maximize the objective function. Once the machine-learner has been trained (objective function has been optimized) its parameters are fixed and it is employed. Figure 3.1 shows this process - phase 1 being training via the objective function, and phase 2 seeing the application of that trained learner in the problem domain.

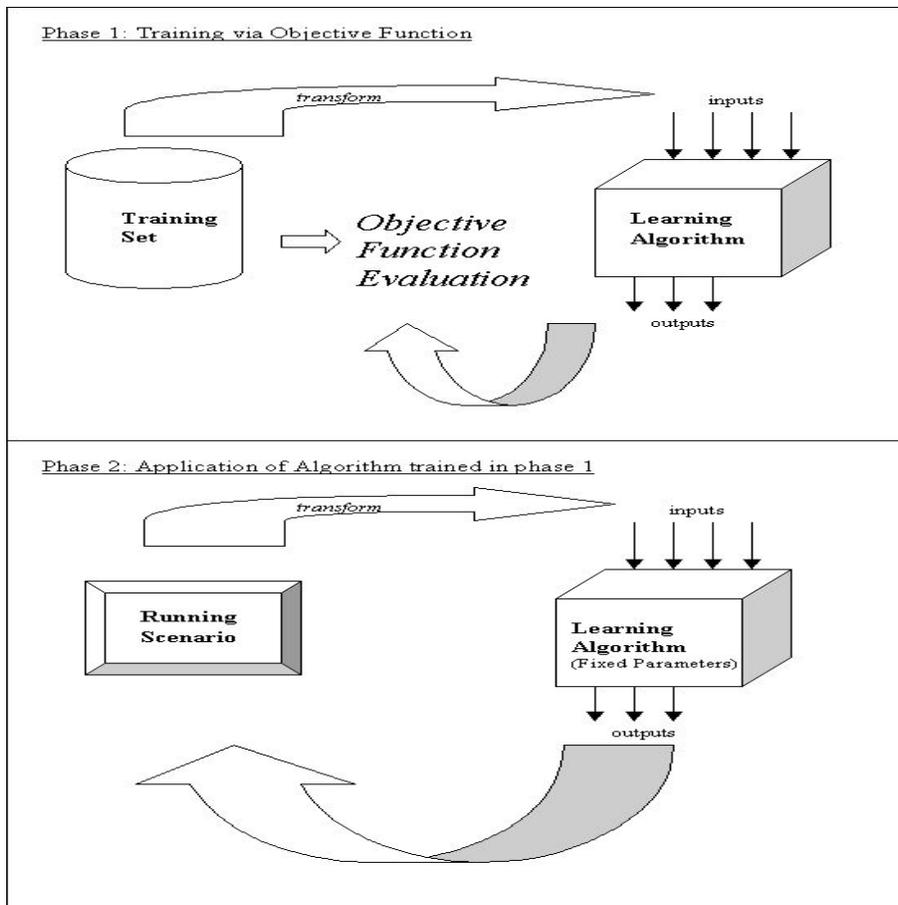


Figure 3.1: The 2-phase process in using a machine-learner: training and application.

In the machine-learning field it is well known that the training phase is crucial. Choices made at this stage about the learning algorithm, its parameters, the size plus scope of the training set, and

the exact form of the objective function; all have ramifications for the final performance that the algorithm will achieve when applied to the problem domain. As such a considerable portion of the report is devoted to this aspect of the problem.

A final point worth making is that it is often desirable to transform the original problem space (i.e., the state of the ABD's world) before input to the learning algorithm. This is simply for the purposes of the learning algorithm, it does not alter the agent's view of the world or their actions. Rather these transformations are undertaken either to reduce the parameter space, transform world values into a range suitable for the algorithm (normalization) or to simplify the problem (e.g., reflections and rotations). This approach of transforming inputs prior to the learner is taken in this report and discussed in a following section. The situation is shown in Figure 2.1 by the "transform" arrow - a process applied to the representation of the ABD world state prior to input to the learner.

# Chapter 4

## Preparing The Problem

Before we introduce the methodology, we will need to re-visit the move equation presented in Chapter 1. This will be undertaken in Section 4.1. Once we establish a valid move equation, we determine measures of success in Section 4.2 followed by necessary problem pre-processing in Section 4.3.

### 4.1 Re-visitng the Move Equation

A problem may arise with Equation 1.3. Because of the normalization condition in the denominator, when  $\tau \rightarrow \infty$ , the agent's move is taken at random according to a uniform distribution. When  $\tau \rightarrow 0$ , the agent move will also be taken at random according to a uniform distribution. In other words,  $\tau$  is not behaving as the temperature in SA. Therefore, we will need to change the equation to

$$P(\text{Move} = i) = \exp\left(\frac{-Z_i}{\tau}\right), \quad i = 0, \dots, I \quad (4.1)$$

### 4.2 Criteria of Success

Before investigating methods for self-adaptation of parameters, one need to clearly identify the overall criteria for a successful run. We will assume that there is a single flag and the overall success is achieved by reaching this flag. Adaptation is simply a process that guides the run to achieve a goal(s). However, in order to make our work as flexible as possible, we will propose a number of criteria for success.

**Criterion 1** Success is declared when at least one agent reaches the flag.

**Criterion 2** Success is declared when all agents reach the flag.

**Criterion 3** Loss exchange ratio

#### Criterion 4 Destruction of all enemy

Based on each criteria, a measure of fitness can be defined to guide the adaptation process locally till the overall objective is achieved. For criterion 1, a possible measure of fitness is the distance of the nearest agent to the flag. For criterion 2, a possible measure of fitness is the distance of the farthest agent from the flag. Another possible measure of fitness for criterion 2 is the average distance between all agents and the flag.

In a typical self-adaptive system, this fitness measure is used as measure for successful adaptation as well. But before we can proceed, we need to undertake some pre-processing for the environment.

### 4.3 Pre-processing

In an environment such as MANA, a number of problems would arise which may add unnecessary complexity to the learning problem. The first problem is symmetry, a simple rotation to the environment would result in a different problem for the learning machine but in fact the rotation does not change in principle the underlying learning problem. To overcome the complications of symmetry, we need to undertake a suitable translation and reflection to the space. Another problem is how to identify the center of concentration of forces. What we propose here is to apply a suitable mask to the board; once for the blue team and another for the red. In what follows, we will introduce each of these pre-processing.

#### 4.3.1 Translation

Let  $(x_f, y_f)$  be the coordinate of the flag  $f$  and  $(x_l, y_l)$  be the coordinate of the farthest agent  $l$ . We use the farthest agent to make sure that all other agents are in front of it; therefore after translation, all agents are in the positive side.

Let us assume that the environment has a dimension of  $x_w \times y_w$ . It is simple to see that, in order to always have the flag at the center of the environment at  $(x_w/2, y_w/2)$ , we need to calculate the vector  $d$  which represents the direction of the translation,

$$\begin{pmatrix} d_x \\ d_y \end{pmatrix} = \begin{pmatrix} x_f \\ y_f \end{pmatrix} - \begin{pmatrix} x_l \\ y_l \end{pmatrix} - \begin{pmatrix} x_w/2 \\ y_w/2 \end{pmatrix} \quad (4.2)$$

the following translation – using homogenous coordinates – need to be undertaken for each entity,  $(x_{old}, y_{old}, 1)$  in the environment.

$$\begin{pmatrix} x_{old} & y_{old} & 1 \end{pmatrix} = \begin{pmatrix} x_{new} & y_{new} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -d_x & -d_y & 1 \end{pmatrix} \quad (4.3)$$

It is worth mentioning that since in MANA the environment is a grid, all values of the previous coordinates are integer and they will remain integer after the translation; therefore we will still maintain a grid representation.

### 4.3.2 Determining the Center of Forces

To determine the center of concentration of forces for each team, let us assume the environment to be  $E = \{(x_i, y_j) | i = 1, \dots, x_w, j = 1, \dots, y_w\}$ . For each cell,  $(x_l, y_k)$ , we will need to define the neighborhood of this cell to be as follows:

$$\begin{pmatrix} (x_{l-1}, y_{l+1}) & (x_l, y_{l+1}) & (x_{l+1}, y_{l+1}) \\ (x_{l-1}, y_l) & (x_l, y_l) & (x_{l+1}, y_l) \\ (x_{l-1}, y_{l-1}) & (x_l, y_{l-1}) & (x_{l+1}, y_{l-1}) \end{pmatrix} \quad (4.4)$$

For cells at the boundary of the grid, the missing cells in their neighborhood will have zero values in the previous definition (*ie.* periodic boundaries are not assumed).

If one would like to calculate the concentration of forces for the blue team, we can imagine the grid as a binary matrix, where the cell contains 1 if there is an entity belonging to the blue team and 0 otherwise. In this case, we will define the blue environment as

$$E_{ij}^b = \{(x_i, y_j) = \begin{cases} 1 & \text{if the cell contains a blue entity} \\ 0 & \text{otherwise} \end{cases} \quad | i = 1, \dots, x_w, j = 1, \dots, y_w$$

The vice-versa can be defined for the red team,  $E_{ij}^r$ . We can then use the following mask synchronically

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad (4.5)$$

After applying the previous mask a number of times equal to  $\min(x_w/2, y_w/2)$ , the center of concentration of forces will be the cell with the largest number. The idea of this mask is that it propagates the number of entities in the neighborhood of a cell to identify the cell which is surrounded by the maximum number of entities.

### 4.3.3 Reflection

To minimize the load caused by symmetry on the learning machine, we need to fix the quadrant which contains the center of mass. We will assume that the center of mass will always be in the bottom left quadrant of the grid. To guarantee this assumption, all what we need to do is to rotate the grid around the flag clock-wise till the bottom left quadrant contains the center of mass.

### 4.3.4 Inputs

One can expect the following to be potential inputs to the learning machine

- Points of reference

It would be interesting to collect information regarding the grid in terms of points of reference. Points of reference identify a set of centroid, where each point represents a center for collecting local information about the neighborhood. We suggest four points of reference at coordinates reflecting the centroid of each quadrants; that is,

$$p_1 = (x_w/4, y_w/4), p_2 = (3x_w/4, y_w/4), p_3 = (x_w/4, 3y_w/4), p_4 = (3x_w/4, 3y_w/4)$$

After reflection, the center of mass will belong to the first quadrant  $p_1$ . For each centroid  $p_i$ , the following information will be available for blue and red forces in quadrant  $i$ :-

1.  $x_1, x_2$  Number of blue and red forces in the quadrant
  2.  $x_3, x_4$  Average range of fire for each force
  3.  $x_5, x_6$  Average probability of kill for each force
- $x_7$  The size of the blue forces alive

$$N^b = \sum_i \sum_j E_{ij}^b$$

- $x_8$  The size of the red forces alive

$$N^r = \sum_i \sum_j E_{ij}^r$$

- $x_9, x_{10}, x_{11}, x_{12}$  For each quadrant, the distance between projection of the closest enemy agent to the flag onto the line of sight between the closest friendly agent to the flag and the flag itself.

- Fitness of the blue for scenario 1

$$\min_{i,j} \left( \binom{x_w/2}{y_w/2} - \binom{i}{j} \right) \forall E_{ij}^b = 1$$

- Fitness of the blue for scenario 2

$$\max_{i,j} \left( \binom{x_w/2}{y_w/2} - \binom{i}{j} \right) \forall E_{ij}^b = 1$$

OR

$$\frac{1}{N^b} \sum_{i,j} \left( \binom{x_w/2}{y_w/2} - \binom{i}{j} \right) \forall E_{ij}^b = 1$$

- Fitness of the blue for scenario 3 Loss exchange ratio

$$\frac{N^b}{N^r}$$

- Fitness of the blue for scenario 4 Destruction of all enemy

$$N^r$$

# Chapter 5

## Potentially Suitable Learning Algorithm

The learning algorithm is the chief component of the adaptation system. However it is a component in the larger framework and hence can be treated as a replaceable element.

The machine-learning field has many different tools (algorithms) available for learning the mapping of inputs to outputs. No algorithm is pre-eminent under all situations - each has strengths and weaknesses. Hence any feasibility study of adaptation in ABDs should discuss a range of potential candidates together with their strengths and weaknesses for the task.

This section briefly discusses each of Hidden Markov Models (HMMs), Decision Trees, and Evolutionary Artificial Neural Networks (EANNs). EANNs are identified as the most suitable candidate for implementation due to their high (in objective function maximisation sense) performance and minimal human intervention in the process - the network architecture is automatically evolved rather than pre-assigned, and no set of "pre-calculated" outputs ( $r$ ,  $\alpha$  and  $\tau$ ) are needed for the training scenarios. Hence more detail is provided for EANNs than the other approaches.

### 5.1 Markov and Hidden Markov Models (HMMs)

Markov and Hidden Markov Models (HMMs) (also called Markov chains) [16] are a means of describing the underlying model that produces a sequence of observations. Those observations might be a DNA sequence, spectrum of speech, or the movement choices of an agent in an ABD.

The Markov model is a collection of states, each with their own output probabilities. Those probabilities describe the chance of the particular state producing the different observations. Similarly, states are connected and have transition probabilities associated with them. Figure 5.1 is an example of a 6-state left-to-right HMM - the model can only progress from left-to-right. Other architectures, such as fully-connected are also possible.

HMMs are powerful machine-learning tools that are heavily used in the areas of speech recognition,

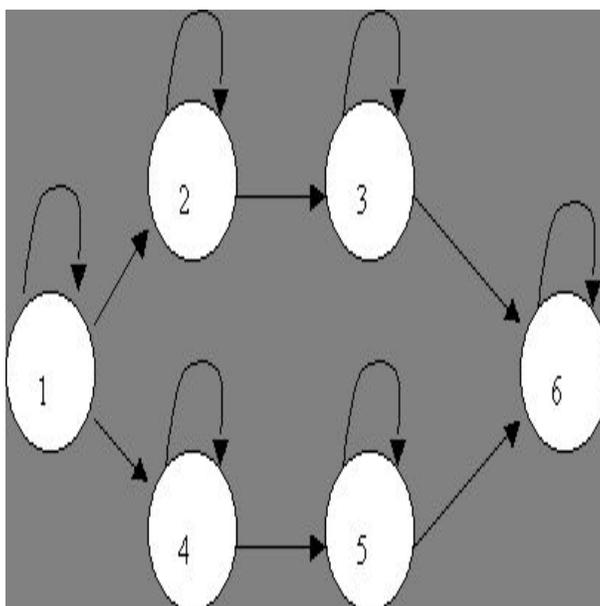


Figure 5.1: A 6-state, left-to-right HMM

machine vision, and protein sequencing, just to name a few. Automatic methods [10] can be used to both train a HMM and to score an observation sequence - determine the likelihood that a particular sequence was produced by the HMM in question.

A Markov process is one possible approach to implementing self-adaptation in an ABD. In the particular instance of the MANA movement algorithm the observation sequence would be the simulation world state at each time step, while the Markov states would reflect changes in the three parameters. For instance: decrease-significantly, decrease-slightly, don't-alter, increase-slightly, increase-significantly, for each of the three movement parameters. Thus a single 3D HMM lattice (one dimension for each of the 3 movement parameters) is one possible approach, while three separate HMMs (one for each parameter) is an alternate.

HMMs have the advantage that they are very good learners (often the best performer recognisers in different scientific areas are built around HMMs). Unsupervised training is also possible. In the case of the suggested application to the MANA movement algorithm though a set of labelled training data would be required - world states plus the best  $r$ ,  $a$  and  $t$  values at those times. While there are some semi-automatic techniques for deriving such training data this is a significant disadvantage when compared with EANN. The architecture of a HMM - number of states and what transitions are allowed - must be set prior to the start of training. Given that initial configuration the learning algorithm will then determine the optimal weights for the transition and state output probabilities based on the training data.

## 5.2 Artificial Neural Network

*Neural networks* - more specifically feedforward artificial neural networks - are universal function approximators; that is, they can approximate any function accurately. They are found to be suitable for both continuous and discrete domains. A neural network is usually seen as a mapping from the input space to the output space. If we imagine that we would like to approximate the decision policy in reinforcement learning, neural network can do the same job efficiently. The idea here is to use the network to decide on the next transition; therefore a good network calculates implicitly the potential of a transition and choose the one which will increase the overall reward. In addition, neural network can be used as a reinforcement learning tool by training the network to approximate the value function (the expected return from a transition).

## 5.3 Decision Trees

Another type of learning machines which may deem to be useful here is *decision trees* [15]. Decision trees can be used to provide the decision maker with some insight into the nature of the mapping. However, in machine learning we will find that there is a cost for each benefit! Decision trees are more expressive than a neural network; that is, a human can understand the language of a decision tree better than understanding the corresponding neural networks. On the other hand, neural networks are typically more accurate and more compact than a decision tree. They are more accurate because of their use of sigmoid functions which can approximate the target function smoothly. They are more compact because of their use of much lesser number of neurons than the number of hyperplanes used in decision trees.

## 5.4 Evolving Neural Networks Approach

Recalling that the adaptation process is a mapping from a set of parameters' values and current situation to a new set of parameters' values, *Artificial Neural Networks* (ANNs) [8] are quit efficient in approximating this type of mapping.

*Evolutionary Artificial Neural Networks* (EANNs) has proven successful in many application domains. Abbass [1, 2] used a Multi-objective Optimization (MOP) approach [3, 4] to evolve ANNs. The approach was successfully applied to a number of classification problems as well as more recently in robot control. In this section, we will describe this approach and propose it to undertake the adaptation process.

Necessary concepts are defined in Section 5.4.1 followed by the methodology in Section 5.4.2. The chapter is concluded with how to apply the methodology in MANA in Section 5.4.3.

### 5.4.1 Definitions

**Definition 1: Neighborhood of  $x$ ;** The set of all points within a fixed distance from  $x$ .

**Definition 2: Non-dominated solution;**  $x$  is a non-dominated solution if there is no other solution  $y$  which is better than  $x$  when compared on all possible criteria (objectives).

**Definition 3: Artificial Neural Networks (ANN);** An ANN is a graph:  $G(N, A, \psi)$ , where  $N$  is a set of neurons (also called nodes),  $A$  denotes the connections (also called arcs or synapses) between the neurons, and  $\psi$  represents the learning rules whereby neurons are able to adjust the strengths of their interconnections. A neuron receives its inputs (also called activation) from an external source or from other neurons in the network. It then undertakes some processing on this input and sends the result as an output. The underlying function of a neuron is called the activation function. The activation,  $a$ , is calculated as a weighted sum of the inputs to the node in addition to a constant value called the bias. The bias can be easily augmented to the input set and considered as another input.

**Definition 4: Multi-layered perceptron (MLP);** An MLP is in essence a non-parametric regression method which approximates underlying functionality in data by minimizing a loss function. The common loss functions used for training an ANN are the *quadratic* error function, where the task is to minimize the mean (expected) square error (MSE) between the actual observed or target values and the corresponding values predicted by the network.

**Definition 5: Evolutionary Algorithms;** Evolutionary algorithms is a kind of global optimization techniques that use selection and recombination as their primary operators to tackle optimization problems.

### 5.4.2 The Evolutionary Neural Network Algorithm

#### Representation

In deciding on an appropriate representation, we tried to choose a representation that can be used for other architectures without further modifications. Our chromosome is a class that contains one matrix  $\Omega$  and one vector  $\rho$ . The matrix  $\Omega$  is of dimension  $(I + O) \times (H + O)$ . Each element  $\omega_{ij} \in \Omega$ , is the weight connecting unit  $i$  with unit  $j$ , where  $i = 0, \dots, (I - 1)$  is the input unit  $i$ ,  $i = I, \dots, (I + O - 1)$  is the output unit  $i - I$ ,  $j = 0, \dots, (H - 1)$  is the hidden unit  $j$ , and  $j = H, \dots, (H + O - 1)$  is the output unit  $j - H$ . This representation has the following two characteristics:-

1. It allows direct connection from each input to each output units (we allow more than a single output unit in our representation).
2. It allows recurrent connections between the output units and themselves.

The vector  $\rho$  is of dimension  $H$ , where  $\rho_h \in \rho$  is a binary value used to indicate if hidden unit  $h$  exists in the network or not; that is, it works as a switch to turn a hidden unit on or off. The sum,  $\sum_{h=0}^H \rho_h$ , represents the actual number of hidden units in a network, where  $H$  is the maximum number of hidden units. This representation allows both training the weights in the network as well as selecting a subset of hidden units.

## Methods

We have a multi-objective problem with two objectives; one is to minimize the error and the other is to minimize the number of hidden units. The pareto-frontier tradeoff between the two objectives will result in a set of networks with different number of hidden units (note the definition of pareto-optimal solutions). However, sometimes the algorithm will return two pareto-networks with the same number of hidden units. This will only take place when the actual number of pareto-optimal solution in the population is less than 3. Because of the condition of having at least 3 parents - which are pareto solutions - in each generation, if there are less than three parents, the pareto optimal solutions are removed from the population and the population is re-evaluated. For example, assume that we have only 1 pareto optimal solution in the population. In this case, we need another 2. The process simply starts by removing the pareto optimal solution from the population and find the pareto optimal solutions in the remainder of the population. Those solutions dominating the rest of the population are added to the pareto list until the number of pareto solutions in the list is 3.

The algorithm consists of the following steps:

1. Create a random initial population of potential solutions. The elements of the weight matrix  $\Omega$  are assigned random values according to a Gaussian distribution  $N(0, 1)$ . The elements of the binary vector  $\rho$  are assigned the value 1 with probability 0.5 based on a randomly generated number according to a uniform distribution between  $[0, 1]$ ; otherwise 0.
2. Repeat
  - (a) Evaluate the individuals in the population and label those who are non-dominated.
  - (b) If the number of non-dominated individuals is less than 3 repeat the following until the number of non-dominated individuals is greater than or equal to 3:-
    - i. Find a non-dominated solution among those who are not labelled.
    - ii. Label the solution as non-dominated.
  - (c) Delete all dominated solutions from the population.

(d) Repeat

- i. Select at random an individual as the main parent  $\alpha_1$ , and two individuals,  $\alpha_2, \alpha_3$  as supporting parents.
- ii. With some probability  $Uniform(0, 1)$ , do

$$\omega_{ih}^{child} \leftarrow \omega_{ih}^{\alpha_1} + Gaussian(0, 1)(\omega_{ih}^{\alpha_2} - \omega_{ih}^{\alpha_3}) \quad (5.1)$$

otherwise

$$\omega_{ih}^{child} \leftarrow \omega_{ih}^{\alpha_1} \quad (5.2)$$

and With some probability  $Uniform(0, 1)$ , do

$$\omega_{ho}^{child} \leftarrow \omega_{ho}^{\alpha_1} + Gaussian(0, 1)(\omega_{ho}^{\alpha_2} - \omega_{ho}^{\alpha_3}) \quad (5.3)$$

otherwise

$$\omega_{ho}^{child} \leftarrow \omega_{ho}^{\alpha_1} \quad (5.4)$$

where each weight in the main parent is perturbed by adding to it a ratio,  $F \in Gaussian(0, 1)$ , of the difference between the two values of this variable in the two supporting parents. At least one variable must be changed.

- iii. If the child dominates the main parent, place it into the population.

(e) Until the population size is  $M$

3. Until termination conditions are satisfied, go to 2 above.

### 5.4.3 Applying the method in MANA

The inputs to the neural network will be the 12 inputs,  $x_1, \dots, x_{12}$  introduced in Chapter 4 and  $x_{13} = r_t$ ,  $x_{14} = \alpha_t$ , and  $x_{15} = \tau_t$ . The outputs will be  $x_{13} = r_{t+1}$ ,  $x_{14} = \alpha_{t+1}$ , and  $x_{15} = \tau_{t+1}$ .

At each time step  $t$ , the move equation will be calculated using the three parameters  $r_t, \alpha_t, \tau_t$ . At the following time step  $t + 1$ , the fourteen inputs  $x_1$  to  $x_{14}$  will be fed to the neural network and the corresponding outputs  $(r_{t+1}, \alpha_{t+1}, \tau_{t+1})$  are calculated. The run will continue until termination conditions (*ie.* the blue team win/lose) are reached. The fitness of the neural network which guided the run is calculated and evolution continues. In other words, the algorithm will be exactly as the one presented in the previous section with the evaluation step (2.a) undertaken by using the corresponding neural network in a number of complete MANA runs.

Notice that the evaluation is stochastic (the fitness will be different with different MANA runs). Therefore, one would expect to undertake at least 20-50 different MANA runs per network to get a reliable estimate of the network performance.

# Chapter 6

## Insight Extraction From The Learner

Rule extraction from neural networks [6] helps derive a symbolic description of a multi-layer feedforward artificial neural network (some rule extraction techniques work for recurrent neural networks). The aim is to generate a set of symbolic description that mimics the network's behavior in a concise and comprehensible form. The advantages of rule extraction include:

- provide explanation capabilities on the network's behavior,
- discover previously unknown dependencies in the input space,
- it helps to integrate connectionist systems with symbolic ones,
- it is a powerful tool for automated knowledge acquisition, and
- the rules sometimes generalize better than the networks from which they are extracted through the identification of regions in the input space that are now represented.

There are many techniques for rule extraction [6]; we will limit our discussion to three of these techniques in addition to a hybrid technique which balances comprehensibility and accuracy, and can be directly used for rule extraction.

### 6.1 Boolean Rule Extraction

A decompositional approach where the rules are extracted at the level of individual units (both hidden and output). The output of each unit is assumed to be binary (either yes or no). The idea is to identify positive regions in the weight space which results in an output of yes and excluding from these regions any sub-region which results in the reverse output. The algorithm works as follow:

- Select a positive weight  $P$

- Find the set  $N$  of negative weights so that if we add the negative weights to the positive weight in the set  $P$ , the result does not exceed the threshold
- Construct a rule of the form  
*If  $P$  and not  $N$  then True*

Let us take an example. Assume a node with input weights of 5, -1, -2, -3 and a bias of -1.5. The list  $P$  will include  $\{I1\}$  and the list  $N$  will include  $\{\{I4\},\{I3,I4\},\{I2,I4\},\{I2,I3,I4\}\}$ . Now, we can have a rule of the form *If  $P$  and not  $N$  then True*

This method is very simple and provides direct interpretability to the behavior of each node. However, the complexity is very high and some rules may be omitted if we try to restrict the number of antecedents to reduce the complexity.

## 6.2 Validity Interval Analysis

*Validity interval analysis* (VIA) extracts symbolic knowledge from artificial neural networks. A hypothesis is first associated with each rule then the rule is refined to prove/disprove the hypothesis. VIA can be applied to sparse networks as well as recurrent ones, and does not assume a specific training algorithm. The extracted rule are correct but not exact. Each rule takes the form

*If  $x_1$  in  $[a_1, b_1]$  and  $x_2$  in  $[a_2, b_2]$  then  $x_3$  in  $[a_3, b_3]$*

The algorithm starts with arbitrary intervals for all nodes, where each interval represents a constraint on the node. Two steps (forward and backward propagation) are then repeated until consistent (small) changes in the intervals occur. Let us take an example for a forward propagation step.

Assume a node with two inputs  $0 \leq x_1 \leq 1$  and  $0 \leq x_2 \leq 1$ , weights  $w_1 = 4, w_2 = 4$ , and bias of  $b = -6$ . Let us assume the weighted sum of a node is  $net_3$  and the output is  $x_3 = \sigma(net_3)$ . Therefore,

$$net_3 = w_1x_1 + w_2x_2 + b = 4x_1 + 4x_2 - 6$$

Let us also assume that the intervals associated with each node are as follows

$$x_1 = [0, 0.2], \quad x_2 = [0.8, 1]$$

Therefore, the validity interval for  $x_3$  is  $[-2.8, -1.2]$  and the output of the node is  $[0.057, 0.231]$ . We can therefore build a rule of the form

*if  $x_1 \leq 0.2$  and  $x_2 \geq 0.8$  then  $x_3 \in [0.057, 0.231]$ .*

This entails that the node has learnt correctly the inputs in the specified region. Let us take now an example of a backward phase. Assume the same network with a constraint on the output  $x_3 \geq 0.8$  and on one of the inputs  $x_2 \geq 0.8$ . Assuming that the interval for all variables  $0 \leq x_i \leq 1$ ; therefore

$$x_3 \geq 0.8$$

$$x_2 \geq 0.8$$

$$0 \leq x_1 \leq 1$$

propagate forward, we get  $-2.8 \leq net_3 \leq 2$ ; therefore,  $0.3 \leq x_3 \leq 0.8808$  where the constraint on  $x_3$  is not satisfied. Now backproject  $x_3$  to get  $1.386 \leq net_3 \leq 2$ , which means an increase on the lower bound of  $net_3$ . We cannot set bounds on  $x_1$  where we will find that  $0.8465 \leq x_1 \leq 1$ . This will lead to the rule

*If  $0.8465 \leq x_1 \leq 1$  and  $0.8 \leq x_2 \leq 1$  then  $0.8 \leq x_3 \leq 0.8808$ .*

VIA does not guarantee the most compact rule set. However, it provides a basis for other machine learning techniques to compact the rules.

## 6.3 Rulex

RULEX [7] is a technique developed at QUT’s Machine Learning Research Centre, for extracting rules from neural networks that have been trained by the “rapid-backprop” algorithm. Each node in the network is a local response unit (LRU) similar to radial basis function networks, except that the LRU is constructed from sigmoid functions rather than Gaussian. The network is trained by adjusting the centres, widths and steepness of the bumps (*ie.* intersections of sigmoid functions in  $n$  dimension) to minimise the output error. When training is complete, rules are extracted by a direct encoding of the response field of each hidden unit.

There is an additional rule-refinement phase which reduces and simplifies the rules to increase comprehensibility. The three refinement operations are negation, elimination, and absorption. If all possible values of an attribute but one occur within a rule, negation of the absent value is used instead. If all possible values of an attribute make the corresponding ridge active, that attribute is eliminated because it does not contribute to discrimination. Absorption refers to the elimination of an attribute’s negation when it is redundant.

In general, RULEX depends on the LRUs’ configuration. This has the disadvantage of rules being local by definition and global structure may be undetectable. Further, rules may not account for data in overlapping regions. Nevertheless, these techniques are easy and fast to train and the resulting refined rule set can be accurate and concise. Andrews [7] compared a number of rule extraction techniques and found that RULEX alone did not need parameter initialisation.

## 6.4 C-Net

C-Net is a simple novel algorithm, proposed for generating multivariate decision trees from ANNs [5]. The algorithm has three stages. Firstly, a single hidden layered ANN is trained on a suitable training set until performance is deemed to be satisfactory. Secondly, the training set is presented once more to the now-trained ANN but the outputs of the hidden units become the input feature vector to C5, with the target output still playing its usual role in controlling the hypothesis space. Thirdly, the univariate decision trees in the new feature space of hidden unit outputs is readily converted to a multivariate decision trees in the original input space. In the implementation, Quinlan's C5 (an enhancement of his earlier C4.5 [15]) is employed for constructing the univariate decision tree. The C-Net algorithm and its conceptual diagram are presented below.

- 
- *Train a neural network with  $\langle X_{training}, Y_{training} \rangle$ , until it reaches a satisfactory performance on  $\langle X_{validation}, Y_{validation} \rangle$ .*
  - *(1) Re-present  $\langle X_{training}, Y_{training} \rangle$ ,  $\langle X_{validation}, Y_{validation} \rangle$ , and  $\langle X_{testing}, Y_{testing} \rangle$  to the trained network and store  $\langle H_{training}, Y_{training} \rangle$ ,  $\langle H_{validation}, Y_{validation} \rangle$ , and  $\langle H_{testing}, Y_{testing} \rangle$ .*
    - (2) Train C5 with  $\langle H_{training}, Y_{training} \rangle$  and  $\langle H_{validation}, Y_{validation} \rangle$ .*
    - (3) Test C5 with  $\langle H_{testing}, Y_{testing} \rangle$ .*
  - *Replace each condition in the resultant UDT,  $(H_j \text{ op } RHS_j)$ ,  $op \in \{\leq, <, \geq, >, =\}$ , with  $(\sum_{i=1}^I w_{ij} X_i \text{ op } \sigma^{-1}(RHS_j))$ ,*
- 

Figure 6.1: The C-Net algorithm

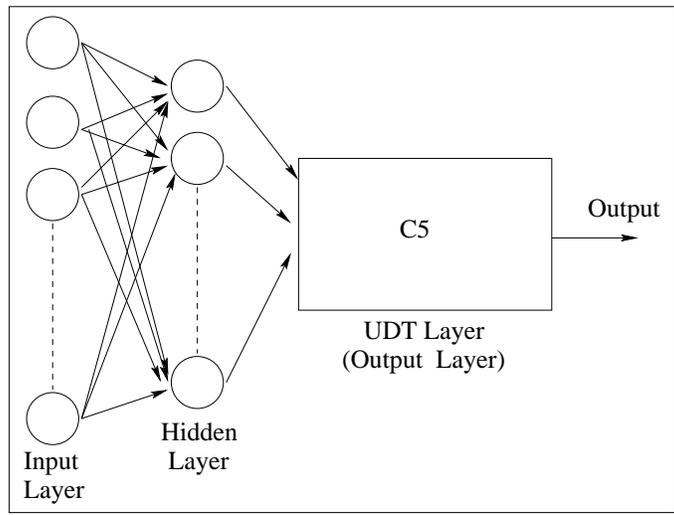


Figure 6.2: The C-Net conceptual representation. The output layer of the trained ANN is replaced with an UDT.

# Chapter 7

## Implementation - A Follow-on Study?

This report is a feasibility and implementation study on the use of adaptive learning for agent based distillations. The report includes a broad outline or design for how a machine learning algorithm - EANN - could be applied to selection of the  $r$ ,  $\alpha$  and  $\tau$  values in the MANA movement system.

This section details the action required if that broad design is to be taken to the next step and implemented. It discusses the key steps required, and some of the issues that will be faced. The following are considered to be the key steps in taking this report's preliminary design and modifying MANA on that basis:

1. Design Refinement - A more detailed design for the system and its components. In particular the software modules and how they link to the existing MANA system.
2. Training Set Design/Selection - The design and selection of a number of training scenarios which will be used to both train and evaluate the learning algorithm.
3. Implementation
  - (a) World Transform - The pre-processing system that transforms MANA's world into a set of inputs suitable for the learning algorithm.
  - (b) ANN System - A system to support Artificial Neural Networks.
  - (c) EANN System - Built on top of the ANN, this module allows the evolution and evaluation of an ANN.
  - (d) Middleware - The coupling of the MANA system with the three previously listed components.
4. Evaluation - A potentially iterative process by which the system performance is evaluated against agents without adaptive learning implemented. This may involve revisiting steps 2 and 3.

5. Insight Extraction - This is an optional step in which the ANNs trained in the previous step are analysed with the goal of extracting the rules they represent.

Most of the work required is coding, with some high-level design (software and experimental) and experimentation also. Whilst exact estimates of the amount of coding required fall outside the expertise of the authors, as an initial estimate it is felt that a suitably qualified programmer - a graduate in computer science (preferably with honors) and a strong mathematical or machine-learning background - could complete the work in 350 - 450 hours of work (two to two and a half months of full-time work). In addition, as already specified, the design and experimental phases of the project should be overseen by a qualified scientist or scientists. This would be far less involved, some tens of hours.

It is recommended that the implementation be performed as an iterative process. While the design and implementation of the software should be as complete as possible from the outset an exploratory experiment representing perhaps a single class of problems (e.g., flag always in top-right corner with enemy agents between) should be trailed first. That pilot, and the corresponding set of training examples, can then be expanded to more and more general situations.

## 7.1 Issues in Implementation

There are two important issues to be considered for any implementation of the design sketch found in this report. The first is the degree of coupling between MANA and the learning system to be implemented. The second is the training set - its size and scope.

There are a range of choices in how closely coupled (integrated) the adaptive learning system is with MANA. At one extreme the system is as decoupled as possible - the learning system is entirely separate to MANA. MANA is modified minimally - simply to output a world state each turn, and accept a set of  $r$ ,  $a$  and  $t$  values for each agent each turn. This has the advantage that MANA is modified minimally, which may be important for software engineering or even political reasons. On the other hand this could greatly slow the running of MANA - as much as one or two orders of magnitude slower perhaps. This is because the communication with the external learning system will be at the file (pipes) level - much slower than operations in main memory.

The other end of the spectrum sees the learning system tightly integrated into MANA. This tends to be a mirror case of the previous situation - there should be no appreciable impact on the speed of MANA's execution (the calculations required for an ANN to process a set of inputs are minimal). On the other-hand such an approach significantly alters the design and size of MANA and probably requires that the learning system be written in the same language as MANA.

Between these two extremes lie a number of compromises. The following list highlights the most obvious options:

1. All adaptation, including training, is performed inside MANA.
2. The training portion is performed outside MANA. MANA still incorporates the transform and ANN system.
3. Both training and ANN application are performed outside of MANA. MANA is modified to apply the pre-processing and output that via a pipe (file), as well as to accept  $r$ ,  $\alpha$  and  $\tau$  values from another pipe.
4. All machine-learning is performed outside MANA. MANA is modified to output the world status every turn to a pipe, as well as to accept  $r$ ,  $a$  and  $t$  values from another pipe.

The second important implementation issue is design and selection of one or more training sets. As was pointed out in a previous section, the particulars of a training set strongly define and constrain how well a final system will perform. If the training set is representative of the cases in which the system will be applied, has good statistical coverage of that range of cases, as well as include sufficient examples such that the parameters of the learning algorithm can be learnt; then the learning system will perform to its full potential. Designing such a training set, while still making the entire approach computationally tractable (an approach with 109 training instances is unlikely to be very useful) is a considerable challenge. For this reason it is strongly suggested that a bottom-up approach be taken in building one or more training sets - start from particular well-defined sub-problems and build up to further generality from there.

# Chapter 8

## Conclusion And Future Work

This project aimed at developing mathematical or computational techniques for adaptive learning within multi-agent simulations. The research objectives of the project and the corresponding actions are as follows:-

1. Objective: Initially focus on the mathematical technique of reinforcement learning and determine its suitability to enable adaptive learning within multi-agent simulations.

Action: The suitability of reinforcement learning is discussed in Chapter 3. We proposed evolutionary neural networks as an alternative and more useful platform.

2. Objective: Produce adaptive learning algorithms, either from reinforcement learning or from other appropriate mathematical or computational fields, which can be most easily implemented within multi-agent simulations.

Action: A description of three adaptive learning algorithms is given and a detailed algorithm is explained using evolutionary artificial neural network.

3. Objective: Detail the requirements for modifications, if any, to multi-agent simulations to enable the inclusion of the adaptive learning algorithms produced.

Action: an implementation plan is given in Chapter 7.

4. Objective: Provide a description of the potential uses and benefits of the adaptive learning algorithms developed for multi-agent simulations.

Action: a description of the potential uses and benefits of the adaptive learning algorithms developed for multi-agent simulations is given in Chapters 2 and 3.

5. Objective: The project deliverable is a written report detailing the research results against each of the four objectives above.

Action: This document represents the project deliverable.

In summary, the project was successful and the interaction between our group and the DSTO group lead by Dr. Gill was very productive and successful. The research reported here raised a number of questions described below.

## 8.1 Research Questions

A number of research questions are likely to arise during the implementation of the proposed system. Further, it is likely that the system will facilitate a number of research questions that could not be addressed previously.

The best design for, and organization of training sets for this particular problem is unknown. This not only has theoretical but practical implications. For instance it may be practical or desirable to have a single model for all classes of problems that a group of agents face. Alternatively it may be "better" to have separate models for separate types of problems - for instance one model for when the moving force is relatively clustered and another model for when the moving force is dispersed - and apply the model that is most appropriate for the problem. The second approach would require a clustering of training data into different sets and the training of separate models. Each new problem would then need to be classified before the most appropriate model could be applied. Determining whether this would be a better approach than a single "global" model for all cases would require experimentation.

A similar, but system-driven approach is to build a number of ANN learners for a number of different classes of problems. Those classes could then be automatically grouped on the basis of differences between the ANNs, rather than a human-assigned partitioning. This may yield new insights about the problem domain itself.

One critical question is the performance of the agents that incorporate adaptive learning. How different is their behavior contrasted with agents that lack learning? This can be measured in a number of different ways. At one level the variability in the alpha, tau, and r values that the learning algorithm generates can be examined. At a higher, and more interesting, level the performance of the agents can be measured. Given a set of scenarios the outcomes when adaptive learning agents are used can be quantified and contrasted against those same scenarios but using non-adaptive agents. Most interesting, but most challenging, would be to seek to contrast performance in a qualitative sense - for instance are the learning agents being "more manoeuvrist"?

Finally, the question of rule-extraction from the ANN(s) arises. It will be possible, with some loss of accuracy, to transform the system represented by the weights of the ANN into a set of decision-

tree like rules. The intriguing question of whether those rules can be understood and encoded in a succinct and human understandable form arises. This is an open possibility - though the very multi-dimensional aspect of the input parameter space, which necessitated an automated machine-learning approach, implies that the rule representation of the ANN weights will be difficult for a human-being to interpret.

# Bibliography

- [1] H.A. Abbass. A memetic pareto evolutionary approach to artificial neural networks. In Markus Stumptner, Dan Corbett, and Mike Brooks, editors, *AI2001: Advances in Artificial Intelligence*, LNAI2256, pages 1–12. Springer, 2001.
- [2] H.A. Abbass. An evolutionary artificial neural networks approach for breast cancer diagnosis. *Artificial Intelligence in Medicine*, 2002.
- [3] H.A. Abbass. Self-adaptive pareto differential evolution. In *IEEE Congress on Evolutionary Computation, USA*. IEEE Publishing, 2002.
- [4] H.A. Abbass, R. Sarker, and C. Newton. A pareto differential evolution approach to vector optimisation problems. In *IEEE Congress on Evolutionary Computation, Seoul, Korea*, volume 2, pages 971–978. IEEE Publishing, 2001.
- [5] H.A. Abbass, M. Towsey, and G. Finn. C-net: A method for generating non-deterministic and dynamic multivariate decision trees. *Knowledge and Information Systems: An International Journal (KAIS)*, 5(2):184–197, 2001.
- [6] R. Andrews and J. Diederich. Rules and networks. *Proceedings of the Rule Extraction from Trained Artificial Neural Network Workshop, University of Sussex, Brighton, U.K.*, 1996.
- [7] R. Andrews, J. Diederich, and A. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6):373–389, 1995.
- [8] S. Haykin. *Neural networks - a comprehensive foundation*. Printice Hall, New Jersey, USA, 2 edition, 1999.
- [9] A. Ilachinski. Irreducible semi-autonomous adaptive combat (isaac): An artificial-life approach to land combat. *Military Operations Research*, 5(3):29–47, 2000.
- [10] G.D.Forney Jr. The viterbi algorithm. *Proc. IEEE*, 61(3):263–278, 1973.
- [11] Michael K. Lauren. Characterising the difference between complex adaptive and conventional combat models. DOTSE Report 169 1335, New Zealands Defence Technology Agency, 1999.
- [12] Michael K. Lauren. Modelling combat using fractals and the statistics of scaling systems. *Military Operations Research, Warfare Analysis and Complexity Special Issue*, 5(3):47–59, 2000.

- [13] Michael K. Lauren and Roger T. Stephen. Modelling patrol survivability in a generic peace-keeping setting using isaac. DOTSE Report 177 1358, New Zealands Defence Technology Agency, 2000.
- [14] Michael K. Lauren and Roger T. Stephen. Map aware non-uniform automata, version 1.0. Users manual, New Zealands Defence Technology Agency, 6 2001.
- [15] J.R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufman, 1993.
- [16] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Mag.*, pages 4–16, 1986.