The Thesis Committee for Anand Subramoney
certifies that this is the approved version of the following thesis:

Evaluating Modular Neuroevolution in Robotic Keepaway Soccer

APPROVED BY

SUPERVISING COMMITTEE:

Risto Miikkulainen, Supervisor

Peter Stone

Evaluating Modular Neuroevolution in Robotic Keepaway Soccer

by

Anand Subramoney, B.Tech.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2012

Dedicated to my wife Ashwini.

Acknowledgments

I would like to thank Prof. Risto Miikkulainen, my supervisor, for his incredible patience, support and his sage guidance during the making of this thesis. Without his insightful guidance, suggestions and critiques this thesis would not have been possible. I would also like to express my gratitude to Prof. Peter Stone, for taking time out of his busy schedule and reviewing this thesis and providing very fast feedback. Finally I would like to thank my wife, Ashwini, my parents, and friends, in particular Srinath TV and Anand K for their constant support and advice.

Evaluating Modular Neuroevolution in Robotic Keepaway Soccer

Anand Subramoney, M.S.Comp.Sci.
The University of Texas at Austin, 2012

Supervisor: Risto Miikkulainen

Keepaway is a simpler subtask of robot soccer where three 'keepers' attempt to keep possession of the ball while a 'taker' tries to steal it from them. This is a less complex task than full robot soccer, and lends itself well as a testbed for multi-agent systems. This thesis does a comprehensive evaluation of various learning methods using neuroevolution with Enforced Sub-Populations (ESP) with the robocup soccer simulator. Both single and multi-component ESP are evaluated using various learning methods on homogeneous and heterogeneous teams of agents. In particular, the effectiveness of modularity and task decomposition for evolving keepaway teams is evaluated. It is shown that in the robocup soccer simulator, homogeneous agents controlled by monolithic networks perform the best. More complex learning approaches like layered learning, concurrent layered learning and co-evolution decrease the performance as does making the agents heterogeneous. The results are also compared with previous results in the keepaway domain.

Table of Contents

List of Figures

Chapter 1

Introduction

Robot soccer keepaway is a sub-component of robocup soccer that is not as complex as the full robot soccer, and provides an excellent testbed for machine learning in multi-agent systems[19]. The skills used in keepaway are very relevant to robot soccer, and hence it provides a good balance of manageable complexity and similarity to the full robot soccer game. The "Robocup Soccer Simulation League" uses the robocup soccer simulator[4].

Learning complex behavior, especially with minimal human input, is a challenging problem. Neuroevolution provides a method to learn this behavior. Neuroevolution as a method of training neural networks has been successfully used to solve large complex domains [26], [16], [5], [6]. Although computationally more intensive than back-propagation, it is less prone to stagnation and more efficient in searching complex landscapes. One of the more successful neuroevolution techniques is neuroevolution with Enforced Sub-Populations (ESP) ([7], [8]). ESP evolves sub-populations of neurons for each hidden node and allows neurons to specialize for the position they are in.

Complex tasks also require keeping track of several factors in the environment, such as multiple opponents, and executing several different behaviors

at once and in succession. Therefore a good way to encode control architectures would be to use modularity e.g. multiple components within a network, and training based on subtasks. In the past, several such methods have been proposed, and in this thesis, they are tested in a uniform, interesting platform of robocup soccer simulator.

One of the major approaches in this study is task decomposition. At a high level, many complex problems have a natural solution – split the large complex task into smaller manageable parts. Solving the parts may be easier than solving the entire problem at once and these smaller solutions then can be combined to give a solution for the entire problem. Task-decomposition is precisely this approach, where the domain is split up into smaller sub-tasks and later combined.

The performance of task decomposition with both ESP and multi-component ESP, and learning methodologies including layered learning, concurrent layered learning and co-evolution in the robot soccer domain is the primary focus of study. This thesis does a comprehensive study of various variants of the ESP neuroevolution algorithm used in various learning methodologies in the robot soccer keepaway task, using the robocup soccer simulator.

Part of this work has been inspired by the work done by Whiteson et al.[24]. Whiteson et al. studied the performance of various approaches, including layered learning and co-evolution, to solve the robot soccer domain using the SoccerBots simulator. Some of these experiments are recreated for the robocup soccer simulator. This study also includes comparison with multi-

component ESP and using heterogeneous agents.

It is found that using the robocup soccer simulator, the monolithic network does surprisingly well, in contrast to some earlier results. A possible explanation for this intriguing result is that the robocup simulator is a simpler domain in that kicking behaviour is separate and doesn't require learning complex movement strategies. The ball and the keepers are also equal in size, thus requiring less manoeuvring. So the agent is able to learn other tasks well.

Chapter 2 briefly describes other related studies. The approach is described in detail in Chapter 3 followed by a listing of the results in 4. Chapter 5 discusses the results, and possible extensions of this work, and Chapter 6 concludes the thesis.

Chapter 2

Background and Related Work

In this chapter, a brief description of previous work keepaway, neuroevolution, task-decomposition and multi-agent systems is given.The keepaway domain, task decomposition, and neuroevolution with ESP and multi-component ESP is also described in detail.

2.1 Keepaway

Keepaway is a sub-task of robot soccer that consists usually of three "keepers" and one or two "takers". The keepers are tasked with keeping possession of the ball, while the taker tries to snatch the ball from the keepers. The game is played within a rectangular (or circular) area of fixed dimensions. The game ends when either the taker gets the ball, or the ball goes outside the enclosing area. Both the keepers and the takers are allowed to move outside the enclosing area. For each pass completed, the keepers receive 1 point. In other studies of keepaway [18], [13], the time the ball remains in possession of the keeper is used as the fitness function rather than the number of passes. But in this thesis, for the purposes of comparison with the results in Whiteson et al., the number of completed passes is the fitness function. An illustration

of the keepaway setup is shown in Figure 3.1.

Each agent has access to visual information, received from the server, of the global positions and velocities of all the other agents, both teammates and opponents, the global position and velocity of the ball, and the position of the center of the keepaway rectangle. Each agent also has two types of actions it can perform – dash (move) with a particular power and angle, or kick the ball with a particular power and direction.

Whiteson et al. [24] studied the performance of various methods using layered learning. Layered learning is a mechanism of learning the separate subtask first, and combining them for the overall task. Whiteson et al. combined the sub-task networks with both hand coded decision trees that select the output of the appropriate network based on the outcomes of specific questions, and selection networks that select the output of one of the sub-networks. They also introduced one extension of layered learning – concurrent layered learning, that uses pre-trained networks, but evolves them simultaneously in the overall task. Co-evolution is also studied, wherein the entire network is co-evolved. All these methods are compared to a monolithic network, and a hand written script. When a decision tree was used, they found that co-evolution performs the best, followed by concurrent layered learning. Both of these performed much better than a hand-written script. The monolithic networks and layered learning with a decision tree did not perform as well as a hand-written script. When a switch network was used, concurrent layered learning performed much better than a hand-written script. While co-evolution, layered

learning and monolithic networks with a switch network performed well below a handwritten script.

While Whiteson et al. [24] implemented their experiments using the SoccerBots simulator, the implementation of keepaway used in this thesis is based on the keepaway client and trainer written by Gregory Kuhlmann an Peter Stone [21], along with the robocup soccer server [4] version 15.1.0. Unlike the robocup soccer simulator, the only action available to an agent in the SoccerBots simulator is to move with a specific magnitude and direction of velocity. Kicking is done by colliding head-on with the ball at the right velocity. The size of the players is relatively large compared to both the playing field and the ball. All the learning methods that used a switch network in Whiteson et al. – layered learning, concurrent layered learning, co-evolution, are evaluated. In addition to that, multi-component ESP and heterogeneous agents are evaluated in this thesis.

Multi-agent reinforcement learning using the robocup soccer simulator was studied in [17]. In this study, lower level behaviors were handcoded and the combining decision tree was learned. In [13], an evolutionary algorithm was used to learn a strategy for a single player in the keepaway domain.

In [18], reinforcement learning, in particular SMDP Sarsa(λ) is used to learn behaviors for the keepaway domain. In [20] the use of keepaway techiques to full soccer was explored.

Both heterogeneous agents – agents controlled by separate evolving

neural networks, and homogeneous agents – agents controlled by the same evolving neural network were tested in this domain. In this thesis, it is observed that homogeneous agents are able to perform significantly better than heterogeneous agents. Bryant and Miikkulainen [2] explored the ability of homogeneous agents to evolve heterogeneous roles using ESP, and showed that agents evolve adaptive behavior and division of tasks to solve the domain.

Waibel et al.[23] show that depending on the level of cooperation required in the task, homogeneous or heterogeneous agents perform better. In particular, heterogeneous agents perform very well in tasks that don't require much cooperation. Homogeneous agents perform significantly better in tasks that require a lot of cooperation. Similar results are shown by Campbell and Wu [3]. The results obtained in this thesis also verify these results – homogeneous teams of keepers significantly outperform heterogeneous teams of keepers.

2.2 Neuroevolution

Neuroevolution is a machine learning technique where networks are encoded as a gene using and evolved by evaluating the network in the given task, and selectively breeding the fittest individuals. The fittest individuals reproduce through crossover with or without mutation. In this thesis, the Enforced Sub-Population (ESP) neuroevolution algorithm ([7], [8]), and its extension, multi-component ESP [27] is used.

ESP was first proposed in [7] and [8], and has been successfully used in

many domains. In [8], ESP was used to evolve a controller for a finless rocket. Multi-agent ESP has had success in various other domains, notably predator-prey. In [27], multi-agent ESP (Enforced Sup-Populations) was used to co-evolve multiple networks for each set of inputs for a predator-prey task, and it was shown that this co-evolved network performs better than a monolithic network when there were multiple predators and prey involved. This work was extended in [14] to domains with different types of prey, and with individual and shared fitness, where cooperation between the agents was seen to evolve.

In ESP each hidden node in the network being evolved is associated with a subpopulation of neurons, as illustrated in 2.1. Each of these neurons is a genome encoding the input and output weights of the neuron. In each iteration, a random neuron from each subpopulation is selected and used as the hidden neuron at that position. The network is then evaluated, and the fitness at the end of the evaluation is equally shared between all the neurons comprising the network. In each generation, a 'trial' consists of multiple combinations of hidden neurons being chosen from the subpopulation and evaluated. At the end of these evaluations, the neurons within each subpopulation are sorted based on the average fitness the neuron got in the evaluations it participated in, and each neuron in the top 25% is recombined with a higher ranking neuron using 1-point crossover. The resulting children replace the bottom half of the population. Mutation also occurs during this phase. A small fraction of the neurons in each subpopulation are recombined with the neuron from the same hidden node in the generation best network.

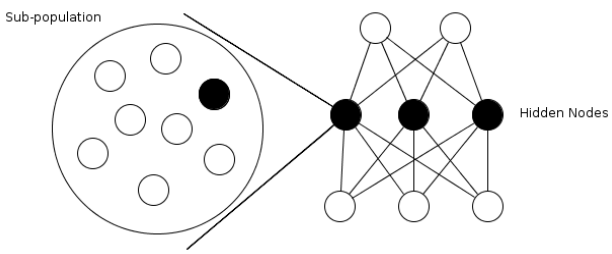


Figure 2.1: In ESP each hidden node is associated with a subpopulation of neurons. In each trial, a random neuron from a subpopulation is selected to occupy the hidden node at that position.

This sort of evolution allows each subpopulation to specialize for the particular node it belongs to, and allow the neurons to cooperate effectively. When performance begins to stagnate, delta-coding is applied in order to prevent premature convergence. Delta coding seeds the population with a cauchy perturbation of the strongest neuron in the population. Since the perturbation is cauchy, most neurons are very similar to the strongest neuron, although a few are radically different. This diversifies the population

Multi-component ESP [27] is an extension of standard ESP, where a separate network component is assigned to each agent in the domain, and the output of these sub-networks are combined using a combiner network. The sub-networks and the combiner networks are all evolved using ESP.

Yong and Miikkulainen [27] used multi-agent ESP (Enforced Sup-Populations) to co-evolve multiple networks for each set of inputs for a predator-prey task, and it was shown that this co-evolved network performs better than a monolithic network when there were multiple predators and prey involved. This

work was extended by Rajagopalan et al. [14] to domains with different types of prey, and with individual and shared fitness, where cooperation between the agents was seen to evolve.

Kohl and Miikkulainen [11] discuss the performance of unrestricted Neuroevolution of Augmenting Topologies (NEAT) as compared to other modifications of NEAT such as RBF-NEAT and Cascade-NEAT[10] and point out that while plain NEAT performs well in domains requiring reactive control, it doesn't perform well if fractured domains. RBF-NEAT which biases searches and Cascade-NEAT which constrains the searches were developed[10] to work well in such domains, and were evaluated both in the keepaway domain and in half-field soccer.

2.3 Task decomposition

Task decomposition is a general method of splitting up a complex task into separate specialised sub-tasks that are easier to learn. A separate subtask network is evolved to solve each of these simpler subtasks. These sub-task networks are later used, along with a selection network (or some combination mechanism) to solve the overall task. Chapter 3 explains the specific decomposition of tasks used in the keepaway domain. Task decomposition is illustrated in 2.2. The selection network chooses between the outputs of the subtask networks. In general the decomposition is done manually. The nature and exact mechanism of the decomposition can affect how well the network learns the overall task. The general idea is that by learning the smaller tasks separately,

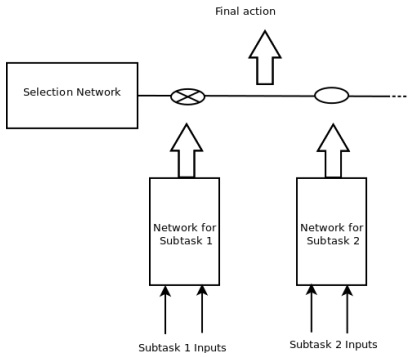


Figure 2.2: An illustration of task decomposition with a selection network. The sub-task networks learn the specialized sub-tasks, and the selection network chooses between the outputs of these sub-tasks in the overall task.

each network has to only contend with a simpler domain compared the overall complex task, thus learning the task in manageable pieces.

Lee [12] studied the task of finding a box in an enclosure and pushing it towards a light source by a robot, decomposing it into separate subtasks of finding the box, positioning the robot, and pushing the box in a straight line. Separate controller circuits were evolved in simulation for each of the sub-tasks, one at a time, using Genetic Programming (GP). Then higher level controller circuits were then evolved to select the appropriate sub-task controller based on the sensory inputs. Such a decomposition of the overall task into separate subtasks performed better than evolving a monolithic controller circuit. The current paper follows a similar approach but evolves neural networks using NEAT instead of controller circuits.

Jain et al. [9] used task decomposition to learn strategies for the predator-prey hunter domain. Neuro-evolution of augmenting topologies (NEAT)

was used as the neuroevolution algorithm and it was shown that as the complexity of the task increases, task decomposition performs better.

The performance of task decomposition in the keepaway domain is also evaluated in this thesis. The keepaway task is split into specialized subtasks, which are learned and later combined with a switch network for the overall task.

Chapter 3

Approach

In this chapter, the setup and approach used for the experiments are described in detail, along with the values of various parameters used. Section 3.1 describes the task decomposition done in keepaway and the overall task. Section 3.2 describes the various learning methods used.

3.1 Domain

The keepaway domain with three keepers and one taker is used in this study. The implementation of keepaway used is based on the keepaway client and trainer written by Kuhlmann and Stone [21], along with the robocup soccer server [4] version 15.1.0.

Each agent had access to visual information, received from the server, of the global positions and velocities of all the other agents, both teammates and opponents, the global position and velocity of the ball, and the position of the center of the keepaway rectangle. The agent had 360° vision and can see the position of all the keepers, takers and the ball at all times.

Each agent had two types of actions it can perform – dash (move) with a particular power and angle, or kick the ball with a particular power

and direction. For both dash and kick, the power can be between -100 and 100 and the angle can be between -180 and 180 (values specified in the server configuration). The ball can be kicked only when it is within a certain distance from the player. The soccer server updates the state of the playing field based on the actions taken by all the players at each time step, the overall update being a cumulative effect of all agents' actions. Although in the robocup soccer simulator dash and turn are separate actions, in this implementation, it is used as an atomic action at a higher abstraction. The agent decides to move in a specific direction with a specific power, and this is sent to the robocup server over two consecutive cycles as a turn action and a dash action.

3.1.1 Overall keepaway task

Figure 3.1: Configuration of agents at the beginning of the keepaway episode. Three keepers occupy three of the four corners of the keepaway rectangle with the ball in front of a randomly chosen keeper. The taker starts at the center of the field.

The setup of the overall keepaway task is shown in Figure 3.1. At the

beginning of the episode, the keepers are positioned at three random corners of the keepaway rectangle, with the ball in front of a random keeper. The taker is positioned at the center of the field. The goal of the keepers is to complete as many passes as possible among themselves. The episode ends when the ball goes outside the keepaway rectangle, or is intercepted by the taker.

The network inputs for each keeper for the overall task were: the relative position and angle of the ball – $Ball_r$ and $Ball_\theta$, the distance of the ball from the center of the field – $Center_r$, the relative positions and angles of the other keepers and the taker from the agent – $Keeper\,1_r$, $Keeper\,1_\theta$, $Keeper\,2_r$, $Keeper\,2_\theta$, $Taker_r$, $Taker_\theta$. The outputs of the network were Dash power and angle, and Kick power and angle. If the agent is within kickable distance from the ball, the kick power and angle outputs is used to kick the ball. Otherwise, the dash power and angle outputs are used. When a selection network is used, the sub-task network selected decides the action the agent takes, and it is not required to kick the ball when it is within kickable distance.

To compare the performance of the network to the one in [24] in the SoccerBots domain, one set of experiments with a network with only two outputs was also conducted. This network's two outputs were interpreted either as kick power and angle or dash power and angle depending on whether the agent was within kickable distance of the ball or not.

The network architectures that were evaluated for both the overall task and the sub-task networks (Section 3.1) was one of the following:

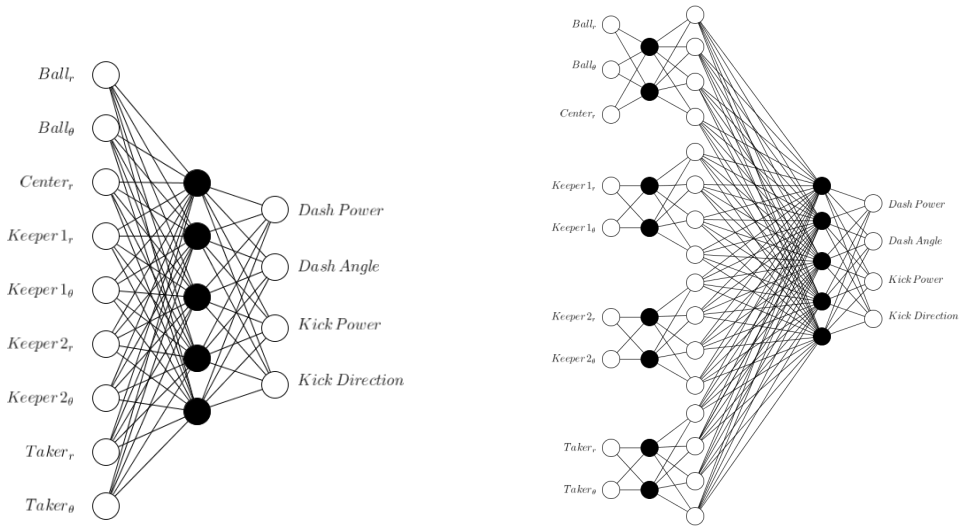
- **Single-component:** A "single-component" network has one input layer, one hidden layer and one output layer.

- **Multi-component:** A "multi-component" network has separate input, hidden and output layers that correspond to each agent on the playing field – one for each keeper, taker and the ball. A single 'combiner' network combines the outputs from these separate components into one output. Each keeper on the field has a separate multi-component network for control.

ESP was used to evolve all the networks. All networks were evaluated for 100 generations, with each neuron in a given subpopulation used in an average of 10 evaluation cycles, each evaluation cycle lasting for 10 episodes of the task/sub-task.

The single-component and multi-component networks used for the overall task are shown in Figures 3.2(a) and 3.2(b) respectively. In the case of the single-component network, five hidden neurons were used, and in the case of the multi-component network, two hidden neurons each was used for the separate components, and five hidden neurons were used for the combiner network.

When training a network for this task, for all cases, **Incremental evolution** [1], [6] was used. In incremental evolution the taker starts off by taking an action only 10% of the time (taking no action for the other 90% of the time). Each time the keepers complete an average of 2 passes in an evaluation cycle, the probability of the taker taking an action was increased by 5%. In effect,



(a) Single-component network for the overall task

(b) Multi-component network for the overall task

Figure 3.2: Single and multi-component network for the overall task. The multi-component network includes separate input, hidden and output layers for each agent, which are then combined.

the taker starts off at 10% speed of the keepers', and every time the mentioned criterion is satisfied, the taker's speed increases by 5% of the keepers' speed.

3.1.2 Sub-tasks

Separate specialized sub-tasks were defined and networks learned in these specialized subtasks were used for the overall task for layered learning and concurrent layered learning, described in Section 3.2. The definition of the sub-tasks in this study is very similar to the ones in Whiteson et al. [24] modified for the robocup soccer domain.

The three defined subtasks were:

- **Intercept:** In this subtask, the game consists on only one keeper and the ball. The keeper starts off in one randomly chosen corner of the keepaway rectangle and the ball is positioned at the center of the field. The ball is then propelled at the keeper at a random angle with a random velocity. The goal of the keeper is to intercept the ball before the ball leaves the keepaway rectangle or the episode times out, and the episode ends when this happens. The timeout is for cases where the initial velocity of the ball is too small for it to leave the keepaway rectangle, and is set at 100 cycles of simulation. An illustration of the starting setup in this subtask is shown in Figure 3.3

Figure 3.3: Configuration of agents at the beginning of the **Intercept** sub-task episode. The ball is propelled towards the keeper with a random magnitude and direction of velocity from the center.

The network inputs in this sub-task were the relative position and angle of the ball – $Ball_r$ and $Ball_\theta$, and the relative magnitude and direction of the velocity of the ball – $Ball\,Velocity_r$ and $Ball\,Velocity_\theta$. The outputs

were the dash power and angle.

The single-component network used for this sub-task is shown in Figure 3.4. Since this sub-task has only one other agent, the ball, a multi-component network is the same as a single-component network. The network had two hidden neurons.

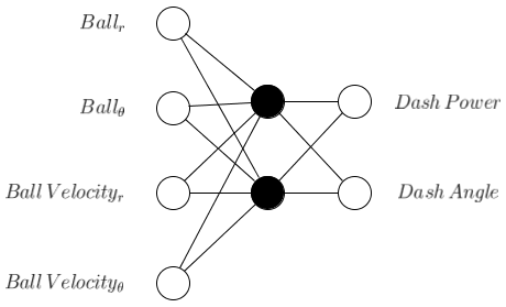


Figure 3.4: Single-component network for the **Intercept** sub-task

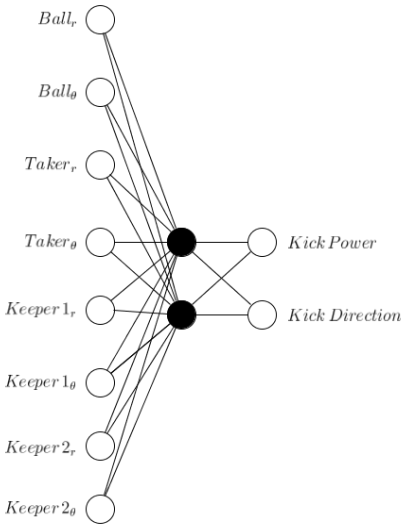
- **Pass:** In this subtask, there are three keepers and one taker. One keeper is positioned at the center with the ball in front of it. This keeper can only kick the ball, and cannot move around. The other two keepers are positioned at the corners on the other side. A taker is also positioned on the other side but closer to the ball. An illustration of this setup is shown in Figure 3.5. The goal of the keeper at the center is to kick the ball to one of the other two keepers without the ball being intercepted by the taker. While the keeper at the center evolves a network to achieve this goal, the other two keepers and the taker use a fixed network learned in the **Intercept** subtask.

Figure 3.5: Configuration of agents at the beginning of the **Pass** sub-task episode. The keeper at the center learns to kick the ball to the other two keepers at the opposite corners controlled by the fixed **Intercept** behaviour, while the taker tries to steal the ball.

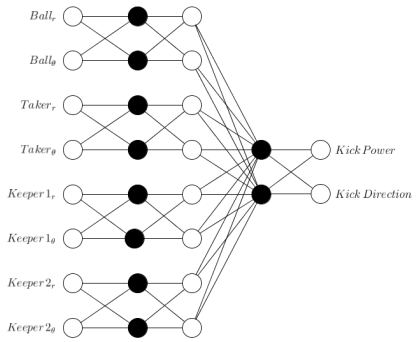
The network inputs for the keeper at the center in this subtask were: the relative position and angle of the ball – $Ball_r$ and $Ball_\theta$, the relative positions and angles of the other two keepers and the taker from the keeper – $Keeper\,1_r$, $Keeper\,1_\theta$, $Keeper\,2_r$, $Keeper\,2_\theta$, $Taker_r$, $Taker_\theta$. The outputs of the network were kick power and direction.

The single-component and multi-component networks used for this subtask are shown in Figures 3.6(a) and 3.6(b). The network had two hidden neurons.

- **Get Open:** In this subtask, the keepers, takers and the ball are positioned exactly as in the **Pass** subtask. The keeper at the center uses a fixed network learned from the **Intercept** subtask, while the other two keepers evolve a network to achieve the goal. The goal for the other two



(a) Single-component network for the **Pass** sub-task

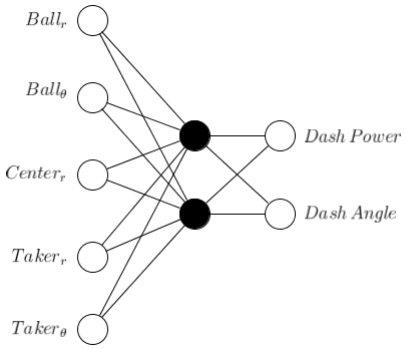


(b) Multi-component network for the **Pass** sub-task

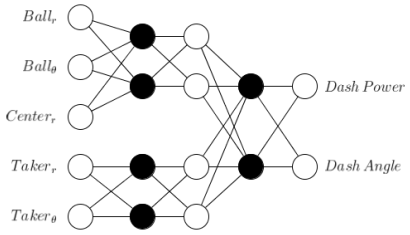
keepers in this subtask is to get to a position where they can successfully intercept the ball from first keeper. The taker uses a fixed network from the **Intercept** subtask to intercept the ball.

The network inputs for the evolving keepers in this subtask were: the relative position and angle of the ball – $Ball_r$ and $Ball_\theta$, the distance of the ball from the center of the field – $Center_r$, the relative position and angle of the taker from the agent – $Taker_r$, $Taker_\theta$. The outputs of the network were dash power and direction.

The single-component and multi-component networks used for this subtask are shown in Figures 3.6(c) and 3.6(d). The network had two hidden neurons.



(c) Single-component network for the **Get Open** sub-task



(d) Multi-component network for the **Get Open** sub-task

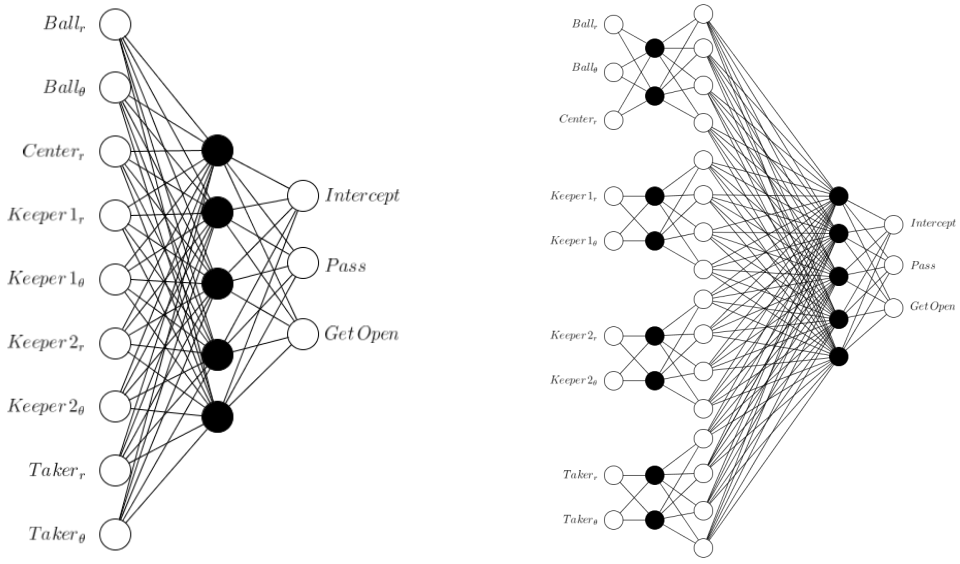
3.2 Learning

Four different learning methods were used to evolve the networks for the overall keepaway task. In all these methods, both single and multi-component networks were evaluated.

- **Monolithic learning:** In this method, a single network was evolved directly in the overall task domain.

- **Layered Learning:** In layered learning [22], the networks for the sub-tasks are first learned in their specialized domains. Then a selection network is evolved in the overall task. This selection network selects between the outputs of the fixed sub-task networks based on the inputs it gets. The final action taken by the agent is based on the network chosen by the selection network. For example, if the **Intercept** sub-task network is chosen by the selection network, then in that cycle, the action performed would be to intercept the ball based on the outputs of the **In-**

tercept network. The single-component and multi-component selection networks used are shown in Figures 3.6(e) and 3.6(f). The outputs of the selection network chooses one of the sub-task networks.



(e) Single-component selection network for the overall task

(f) Multi-component selection network for the overall task

Figure 3.6: The selection network for the overall task selects between the outputs of the three sub-task networks

- **Co-evolution:** This is similar to layered learning except that pre-trained sub-task networks are not used. Both the sub-task networks and the selection network are evolved from scratch in the overall domain. This is similar to the co-evolution approach described in [24].

- **Concurrent layered learning:** The subtask networks are first trained in their specialized task domains. The fittest networks from these sub-task networks are chosen, and used to seed the neuron sub-populations

of new sub-task networks using delta-coding [25]. These newly seeded sub-task networks are used while evolving the selection network in the overall domain like in layered learning. But these sub-task networks are also allowed to evolve simultaneously, to allow them to tune themselves in the overall task. The concurrent layered learning approach first described in [24] is the one used here.

3.3 Type of agents

Two types of agents were evaluated – homogeneous and heterogeneous.

- All the **homogeneous** keepers shared the same evolving network in all the tasks.
- Each **heterogeneous** keeper evolved a separate network, in isolation from the other two keepers.

One of the advantages of having homogeneous agents is that, since they all share the same network, each network is evaluated three times more than for heterogeneous agents. Homogeneous agents can also share the strategies they have learned. The heterogeneous agents have the advantage of being able to evolve specialized behavior for their position which is not easily possible for homogeneous agents. Comparisons between the performance of homogeneous and heterogeneous agents was also done to evaluate which of these factors affect the final performance and to what degree.

Chapter 4

Results

The experiments were run for both homogeneous and heterogeneous agents. Layered learning, co-evolution and concurrent layered learning were evaluated, with and without using multi-component ESP. The results are described in the next few sections. The fitness shown is the average fitness in each episode. For all the experiments, the results shown are an average of 10 runs, with standard error indicated in the graphs.

4.1 Layered Learning

The performance of monolithic network and layered learning using selection network is shown in 4.1. The monolithic network performs better than layered learning. The monolithic network reaches average fitness of 5 per episode, while layered learning levels out at average fitness of 2 per episode.

Experiments were also run by modifying the outputs of the monolithic network to emulate kicking behavior in the Soccerbots domain as described in Section 3.1. The comparison between the monolithic network with this output configuration with layered is shown in Figure 4.2. The performance of the monolithic network does not change much with this configuration.

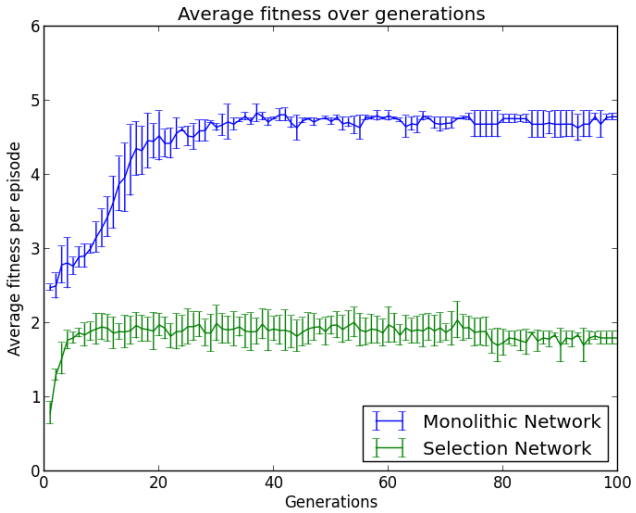


Figure 4.1: Comparison between performance of monolithic network and layered learning.

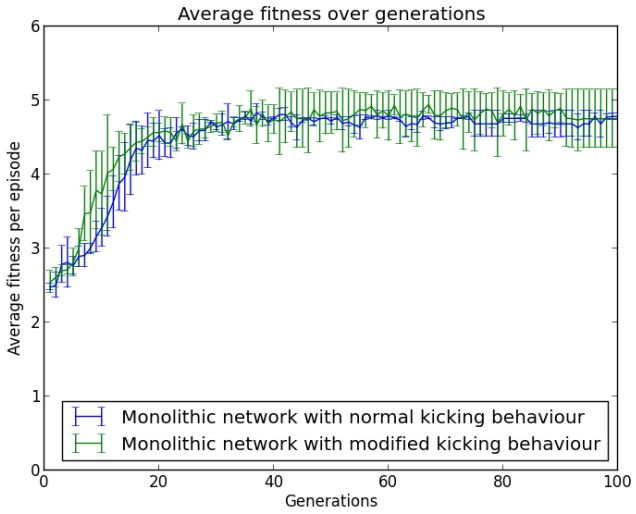


Figure 4.2: Comparison between performance of monolithic network and layered learning with modified kicking behavior.

4.2 Multi-component ESP

Multi-component ESP was evaluated for both the monolithic network and the selection network in layered learning. Figure 4.3 shows the comparison between the single and multi-component monolithic networks. The multi-component network performs slightly better than the single-component network.

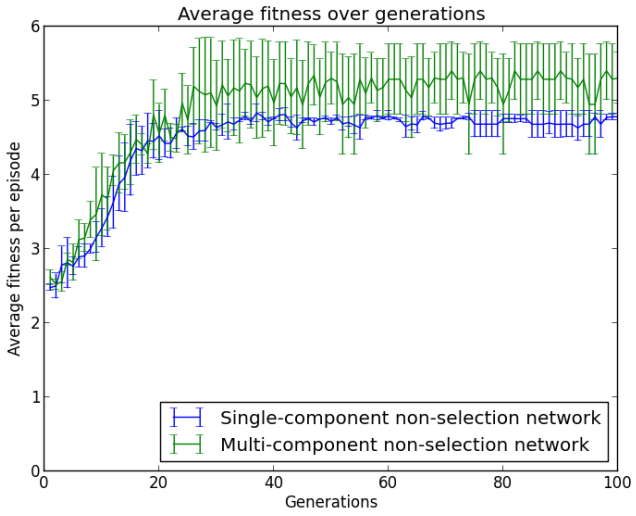


Figure 4.3: Comparison between performance of single and multi component monolithic networks.

In Figure 4.4 comparisons between the single and multi-component selection networks is shown. The multi-component network performs much better than the non-multi-component network, although its performance is still not as high as that of the monolithic network.

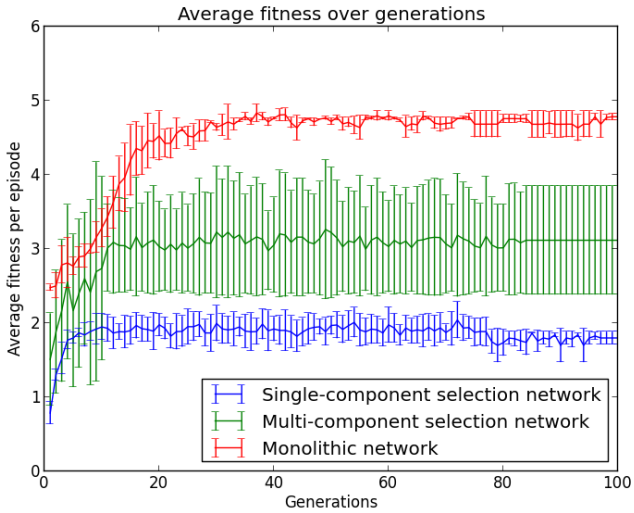


Figure 4.4: Comparison between performance of single and multi component selection networks, and monolithic network.

4.3 Co-evolution

Both the single and multi-component ESP selection networks were run with co-evolution. These networks use untrained sub-task networks, and all the networks – the sub-task networks and the selection network are evolved simultaneously. The results of using co-evolution is shown in Figure 4.5. Co-evolution increases the performance of the selection network significantly.

Co-evolution was also evaluated by replacing all the networks, both sub-task and selection, with networks that use multi-component ESP. The comparison between co-evolution with and without multi-component ESP is shown in Figure 4.6. Although for the monolithic network and layered learning, introduction of multi-component ESP improved the performance, in the case

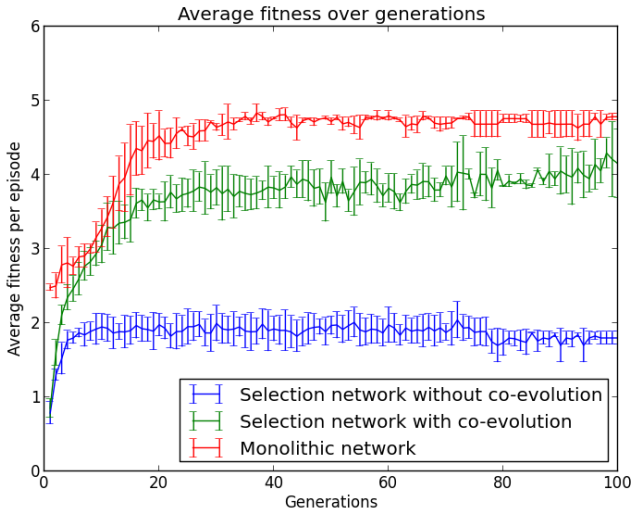


Figure 4.5: Comparison between performance of (single-component) selection network with and without co-evolution.

of co-evolution, the difference in performance is not significant.

4.4 Concurrent Layered learning

The selection networks were run with concurrent layered learning as described in Section 3.2. These networks use trained sub-task networks that continue to evolve during the overall task. The results of using concurrent layered learning is shown in Figure 4.7. Using concurrent layered learning significantly improves the performance as compared to layered learning.

Concurrent layered learning was also evaluated by replacing all the networks, both sub-task and selection, with networks that use multi-component ESP. The comparison between concurrent layered learning with and without

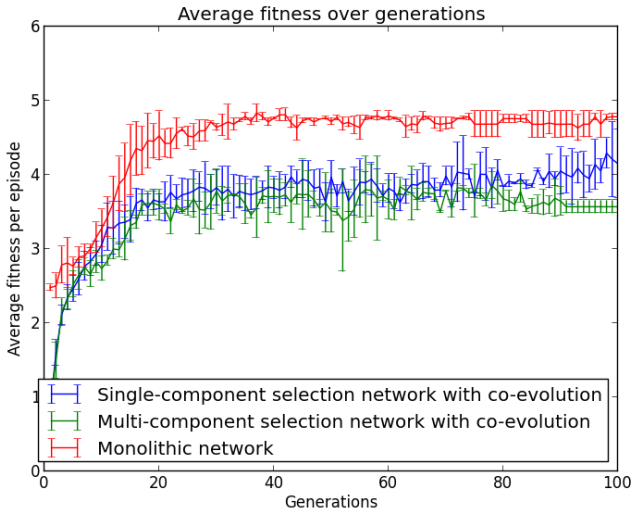


Figure 4.6: Comparison between performance of single and multi-component networks with co-evolution.

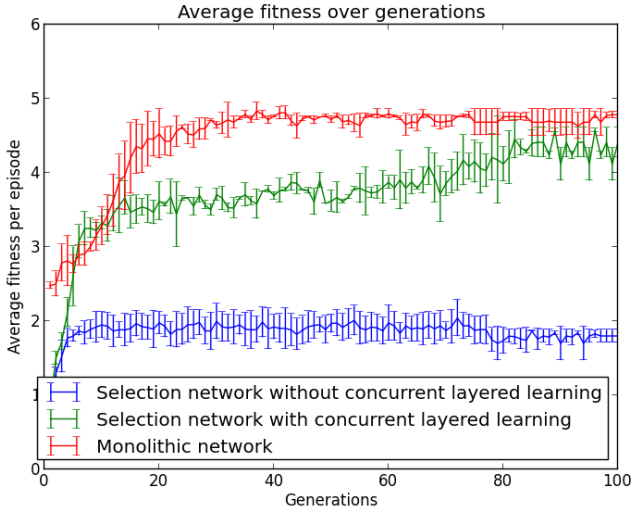


Figure 4.7: Comparison between performance of (single-component) selection network with and without concurrent layered learning.

multi-component ESP is shown in Figure 4.8. As for co-evolution, multi-component ESP does not significantly improve performance of the network.

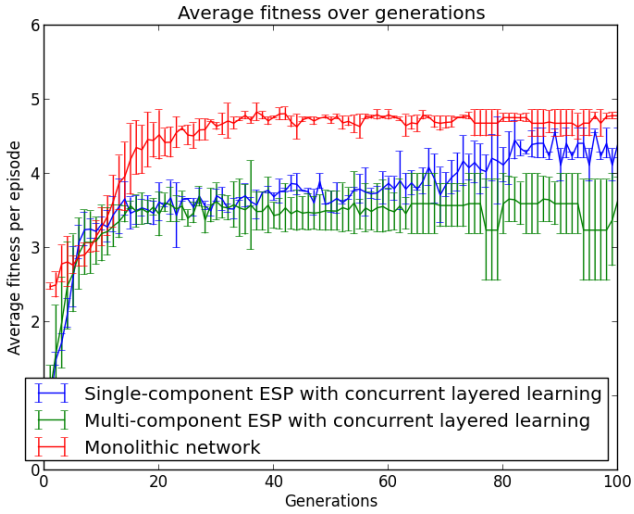


Figure 4.8: Comparison between performance of single and multi-component networks with concurrent layered learning.

The comparison between co-evolution and concurrent layered learning is shown in Figure 4.9. Concurrent layered learning performs much better at later generations than co-evolution. Both the methods still perform worse than the monolithic network.

4.5 Heterogeneous agents

Heterogeneous keepers that evolved separate networks in isolation from the other keepers, were also evaluated in the task of keepaway. All the preceding experiments were conducted using heterogeneous agents. It was found

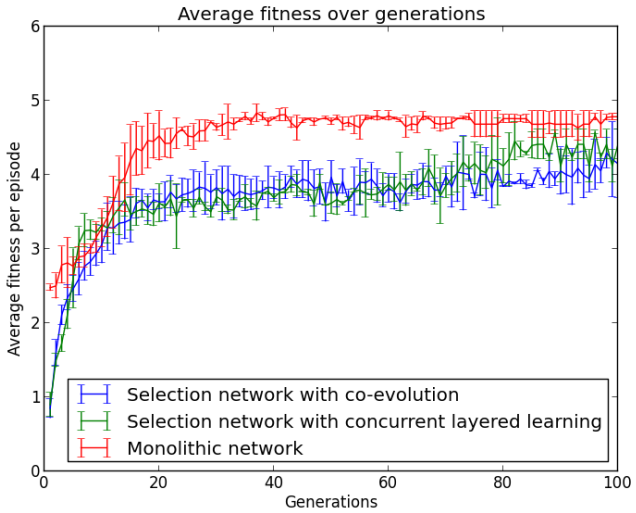


Figure 4.9: Comparison between performance of (single-component) selection network with co-evolution and concurrent layered learning.

that the ordering of the performance of various methods was the same as for homogeneous agents, but the absolute fitness achieved by the homogeneous agents was twice as much as heterogenous agents. Shown in Figures 4.10 and 4.11 are the performances of both the monolithic network and layered learning approaches for heterogeneous agents compared to homogeneous agents.

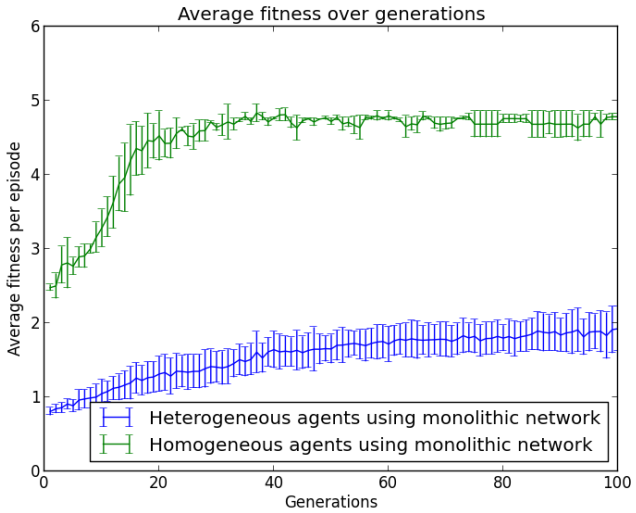


Figure 4.10: Comparison between performance of monolithic network between heterogeneous and homogeneous agents.

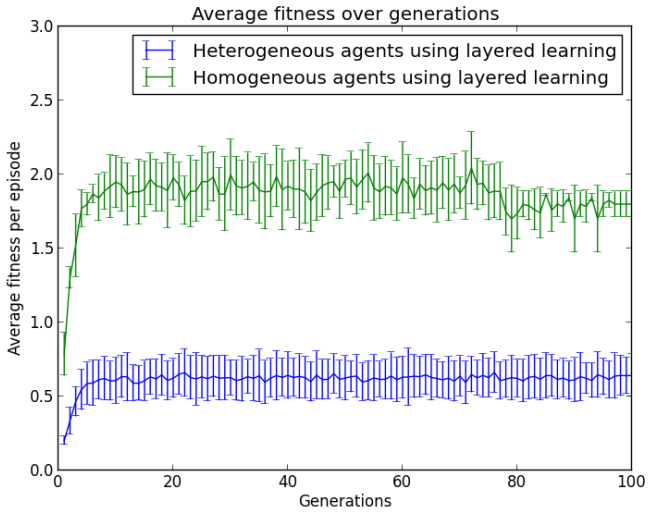


Figure 4.11: Comparison between performance of layered learning between heterogeneous and homogeneous agents.

Chapter 5

Discussion and Future Work

The results described in Chapter 4 are analysed in Section 5.1. Then a brief description of possible extensions of this work in the future is given in Section 5.2.

5.1 Discussion

The goal of this thesis was to perform a comprehensive study of various learning methodologies in the keepaway domain. As seen in the results in the previous sections, the monolithic network performs the best among all the methods evaluated. Multi-component ESP slightly improves the performance of the monolithic network, but all the other methods, including layered learning, co-evolution and concurrent layered learning does not perform as well as the monolithic network.

The separation of the move and kick action in the robocup soccer simulator makes it simple enough for the monolithic network to perform really well. Any extra machinery makes it harder for the agent to learn a successful strategy. Layered learning, co-evolution and concurrent layered learning also depend strongly on how the task decomposition is done, and the particular

means of combination. It is possible that a different set of sub-tasks could lead to better performance for these methods.

The fact that the monolithic network performs the best is also surprising compared to the results in Whiteson et al. [24]. Whiteson et al. used the SoccerBots simulator to evaluate the various methods. In Whiteson et al., concurrent layered learning performed the best, whereas the monolithic network performed the worst. The SoccerBots simulator is different from the robocup soccer simulator in that the players are much larger than the ball, and they have no explicit way to kick the ball. The players can only move around and for kicking the ball, collide with it with the right magnitude and direction of velocity. The player also has to position its front, which has a 'paddle' for kicking, to align the ball. So the player has to approach the ball from an appropriate angle and velocity to make a successful kick. This makes the task much harder compared to the robocup soccer simulator, where the player can kick the ball in an arbitrary direction and velocity if it is within kicking distance of the ball.

In the experiment with the modified kicking behavior, the player used the same two outputs for both dash and kicking. Although this seems similar to the behavior in SoccerBots, in the robocup soccer simulator, the player can come near the ball, and instantaneously change its outputs to kick the ball in the right direction, even if it is not facing the ball. In the SoccerBots simulator, this is not possible since the player has to approach the ball in a specific 'wind up' motion. This introduces an 'inertia' for the player and also explains why

the experiment with the modified kicking behavior did not significantly change the performance in the robocup soccer simulator.

Although the simulator used in Whiteson et al. [24] is significantly different from the one used in this theses, it is interesting to note that in absolute terms, the monolithic network in the robocup soccer simulator, which achieves an average fitness of 5.3 per episode, performs almost as well as the best method, concurrent layered learning with a switch network, in Whiteson et al., which achieves an average fitness of 5.5 per episode (or 55 per evaluation cycle).

In previous work by Padmini and Aditya, [15], [14], multi-component ESP was found to perform significantly better than single-component ESP. In keepaway, for layered learning, the multi-component ESP network performs significantly better. For the monolithic network, the difference is less significant. But overall, in all the experiments, multi-component ESP performs better by varying degrees.

In this thesis also multi-component ESP performs better than single-component ESP, but not to the degree see in the work done by Padmini and Aditya [15], [14]. This can be attributed to that fact that for harder domains, multi-component ESP makes a bigger difference, while for domains where the task is simpler, the extra complexity in the network does not compensate as much in performance.

Homogeneous agents perform much better than heterogeneous agents

in the keepaway domain. The ability to share strategies among themselves outweighs the ability to evolve specialized positional behavior when it comes to the final performance. The fact that homogeneous networks are also evaluated a greater number of times means that the evaluation is more thorough, which aids in the evolution of better networks. The combination of these two factors explains why the difference is performance between the homogeneous and heterogeneous networks is so significant.

5.2 Future Work

There are multiple avenues for extensions of this work. Broadly, these can be classified as (1) changing the definition of subtasks, (2) changing the type of networks itself, and (3) applying the approach to more complex domains. Apart from these three broad categories, automating task decomposition is another possible extension. These extensions are described briefly below.

The performance of all the learning methods depend heavily on the way the sub-tasks are decomposed. Evaluation of other ways of decomposing the task, and providing a more structured way to decompose subtasks would be an interesting extension to this study. For instance, if the fitness measure is possession time of the ball, then one of the sub-tasks could be how well the player can 'dribble' i.e. move around close to the ball without kicking it too far.

Recurrent neural networks could be used to give the agents a sort of

'memory', which would enable it to devise more complex strategies. These methods could also be applied to the more complex domain of full robot soccer. It would be interesting to see if the same performance ordering of the methods would hold when the task becomes much more complex.

Developing a method to partially or completely automate the task decomposition would help reduce the human input that is required right now to specify the sub-tasks. It would also help us understand which tasks are amenable to task decomposition and how decomposition contributes to complex behavior. This automation could take place based on observations of repeating sets of similar actions, or positions of the players.

It would also be interesting to compare the results in this thesis to previous work where the time the ball is in possession of the keeper is used as a fitness measure rather than the number of passes. Since the keepers are not required to kick the ball all the time, different strategies might evolve. Similarly, forcing the keeper to kick in the direction it is facing might make the task more complex, and similar to that in the Soccerbots simulator. It would be interesting to see if this makes the results more similar to the ones in Whiteson et al.

Chapter 6

Conclusion

A comprehensive study of various learning algorithms using both ESP and multi-component ESP were studied. Their performance was evaluated in the robot soccer keepaway domain using the robocup soccer simulator. It was found that a monolithic network performs the best compared more complex learning techniques. This is attributed to the fact that the dynamics of the simulator and the task itself is simple enough for a monolithic network to learn well, without the necessity for other complex machinery. The results were also compared with previous work in the keepaway domain, in particular Whiteson et al. [24]. It was found that although in absolute terms, the performances were comparable, the ordering of performances of the various methods were different. This was due to the differences in dynamics of the SoccerBots simulator used. If the task is relatively unconstrained, a monolithic network work seems to work quite well. But in the case of highly constrained tasks, the search for the solution network is more difficult. In that case, the task decomposition approach or multi-component network might perform better.

Bibliography

[1] David Andre and Astro Teller. Evolving team darwin united. In *In Minoru Asada and Hiroaki Kitano, editors, RoboCup-98: Robot Soccer World Cup II*, pages 346–351. Springer Verlag, 1999.

[2] Bobby D. Bryant and Risto Miikkulainen. Neuroevolution for adaptive teams. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, pages 2194–2201, Piscataway, NJ, 2003. IEEE.

[3] Adam Campbell and Annie S. Wu. Multi-agent role allocation: issues, approaches, and multiple perspectives. *Autonomous Agents and Multi-Agent Systems*, 22(2):317–355, 2011.

[4] Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrick Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. Users manual: Robocup soccer server manual for soccer server version 7.07 and later, 2003.

[5] D. Floreano and J. Urzelai. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, pages 431–443, 2000.

[6] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, pages 317–342, 1997.

[7] Faustino Gomez and Risto Miikkulainen. Learning robust nonlinear control with neuroevolution. Technical report, Technical Report AI01-292, Department of Computer Sciences, The University of Texas at Austin, 2001.

[8] Faustino Gomez and Risto Miikkulainen. Active guidance for a finless rocket using neuroevolution. In *Genetic and Evolutionary ComputationGECCO 2003*, pages 213–213. Springer, 2003.

[9] Ashish Jain, Anand Subramoney, and Risto Miikkulainen. Task decomposition with neuroevolution in extended predator-prey domain. In *Proceedings of Thirteenth International Conference on the Synthesis and Simulation of Living Systems*, East Lansing, MI, USA, 2012.

[10] Nate Kohl and Risto Miikkulainen. Evolving neural networks for strategic decision-making problems. *Neural Networks*, 22(3):326–337, 2009.

[11] Nate Kohl and Risto Miikkulainen. An integrated neuroevolutionary approach to reactive control and high-level strategy. *IEEE Transactions on Evolutionary Computation*, 2011.

[12] Wei-Po Lee. Evolving complex robot behaviors. *Information Sciences*, 121(1-2):1–25, 1999.

[13] Anthony Di Pietro, Lyndon While, and Luigi Barone. Learning in robocup keepaway using evolutionary algorithms. In *Proceedings of*

the Genetic and Evolutionary Computation Conference, pages 1065–1072. Morgan Kaufmann Publishers Inc., 2002.

[14] Padmini Rajagopalan, Aditya Rawal, Risto Miikkulainen, Marc A. Wiseman, and Kay E. Holekamp. The role of reward structure, coordination mechanism and net return in the evolution of cooperation. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2011)*, Seoul, South Korea, 2011.

[15] Aditya Rawal, Padmini Rajagopalan, and Risto Miikkulainen. Constructing competitive and cooperative agent behavior using coevolution. In *IEEE Conference on Computational Intelligence and Games (CIG 2010)*, Copenhagen, Denmark, August 2010.

[16] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, pages 653–668, 2005.

[17] P. Stone and R.S. Sutton. Scaling reinforcement learning toward robocup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 537–544, 2001.

[18] P. Stone, R.S. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[19] Peter Stone, Gregory Kuhlmann, Matthew E. Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In

Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer Verlag, Berlin, 2006.

[20] Peter Stone and David McAllester. An architecture for action selection in robotic soccer. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 316–323, New York, NY, 2001. ACM Press.

[21] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[22] Peter Stone and Manuela Veloso. Layered learning. In *Machine Learning: ECML 2000 (Proceedings of the Eleventh European Conference on Machine Learning)*, pages 369–381. Springer Verlag, Barcelona,Catalonia,Spain, May/June 2000.

[23] M. Waibel, L. Keller, and D. Floreano. Genetic team composition and level of selection in the evolution of cooperation. *Evolutionary Computation, IEEE Transactions on*, 13(3):648 –660, june 2009.

[24] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. Evolving soccer keepaway players through task decomposition. *Machine Learning*, 59(1-2):5–30, 2005.

[25] D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: An iterative search strategy for genetic algorithms. In *Proceedings of the Fourth*

International Conference on Genetic Algorithms, pages 77–84. Morgan Kaufmann, 1991.

[26] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423 –1447, sep 1999.

[27] Chern Han Yong and Risto Miikkulainen. Coevolution of role-based cooperation in multi-agent systems. *IEEE Transactions on Autonomous Mental Development*, 1:170–186, 2010.