CS365A Project
Report

Application of NEAT algorithm in PC Games

Submitted in partial fulfillment of
the requirements for the course CS365A

Submitted by

10114	Ankur Rai
13710	Sourya Basu
Group no. 9	

Under the guidance of
Prof. Harish Karnick

Indian Institute of Technology Kanpur
Kanpur, U.P. , India – 208 016

Abstract

Neuroevolution (NE) is the artificial evolution of neural networks using genetic algorithms. NE works better than some of the best fixed topology algorithms on challenging reinforcement learning tasks such as pole balancing and robot arm control[1]. In this project, we explore the possibilities of the Neuroevolution of augmenting topologies (NEAT) algorithm by using it in a PC game. The game is about two robots in a 2D plane learning to shoot each other and escape attacks at the same time. Initially , both the robots perform random tasks and using the feedback that they get, they evolve over several generations and learn to shoot with accuracy and escape each others attack. Experiments were performed for over 50 generations of Neural Evolution, the results of which shows good learning by the robots in the given environment.

Acknowledgments

Every year CS365A - Artificial Intelligence Programming Course is offered in IITK. The course involves a course project to be completed by a team of two students under the guidance of the course instructor. The course project gives the opportunity to the students to have experience in latest AI techniques and its applications. It would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

The authors would like to express their gratitude towards **Prof. Harish Karnick** for his kind co-operation and encouragement which helped us in the completion of this project.

Ankur Rai
Sourya Basu

April 2016
Indian Institute of Technology Kanpur

Problem Definition

The task is to develop a game using NEAT algorithm which learns to shoot each other and escape each others attack efficiently. The idea is to define a "Fitness" function which is used by the robots in the game to get feedback about their performance, based on which they learn to perform the task in a way which maximizes the fitness function.

Contents

List of Figures

1 Introduction

Neuroevolution (NE) is the artificial evolution of neural networks using genetic algorithms, which has shown excellent results in several complex reinforcement learning tasks. There is a "Fitness" function defined in the algorithm which gives a feedback using which the algorithm searches through the space of behaviours for a network that performs well at a given task.

The main advantage of the NeuroEvolution of Augmenting Topologies (NEAT) is that it minimizes the dimensionality of the search space of connecting weights. The structure in this algorithm is evolved such that the topologies are minimized and grown incrementally which results in significant gains in learning speed.

It is claimed in [3] that the increased efficiency is due to (1) employing a principled method of crossover of different topologies, (2) protecting structural innovation using speciation, and (3) incrementally growing from minimal structure.

In this work we have used the NEAT algorithm in a PC game for which an heuristic function, called the "Fitness" function is defined. Based on the feedback from this function the structure of the network evolves through several generations till the required fitness value is reached. The performance obtained and the variation in fitness and structure parameters thus formed after evolving is being studied.

2 Previous Work

NEAT algorithm is not the first algorithm that evolve both neural network topologies and weight, there was a lot of work done on this as mentioned in [5] - [11]. But there were several problems faced by such algorithms which was later improved by the NEAT algorithm.

One of the problem in such an Topology and Weight Evolving Artificial Neural Networks (TWEANNs) algorithm is the Competing Conventions Problem. Competing Conventions Problem is simply to have multiple solution to the weight optimization problem in neural networks. genomes representing the same solution must have the same representation or else it is likely to produce damaged offspring. This problem is taken care of by the NEAT using the historical information about the genes.

In TWEANNs innovation takes place by mutation when new structure is added. But adding new structures decreases the fitness initially which reduces the chances of the survival of the innovation when optimization takes place over a long number of generation. Even this problem is taken care of

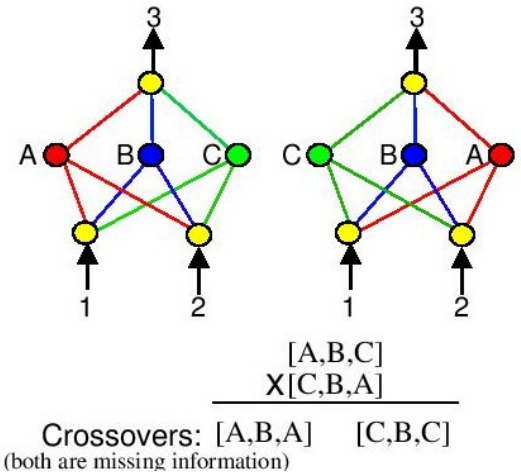

Figure 1: Example of Competing Conventions Problem (taken from [2])

by the NEAT algorithm by a method called speciation.

3 Methodology

To overcome the drawbacks in TWEANNs as mentioned earlier, the NEAT algorithm uses several innovative modifications. These are described in detail in the following subsections as mentioned in [2].

3.1 Genetic Encoding

Genomes are linear representations of network connectivity. Each genome includes a list of connection genes, each of which refers to two node genes being connected. Node genes provide a list of inputs, hidden node, and outputs that can be connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an innovation number, which allows finding corresponding genes.

The above figure demonstrates two kind of structural mutation that takes place, which are *add connection* and *add node*. In an *add connection* mutation, a single new connection gene is added connecting two previously unconnected nodes. Whereas in an *add node* mutation, an existing connection is split and the new node placed where the old connection used to be. The old connection is disabled and two new connections are added to the genome.

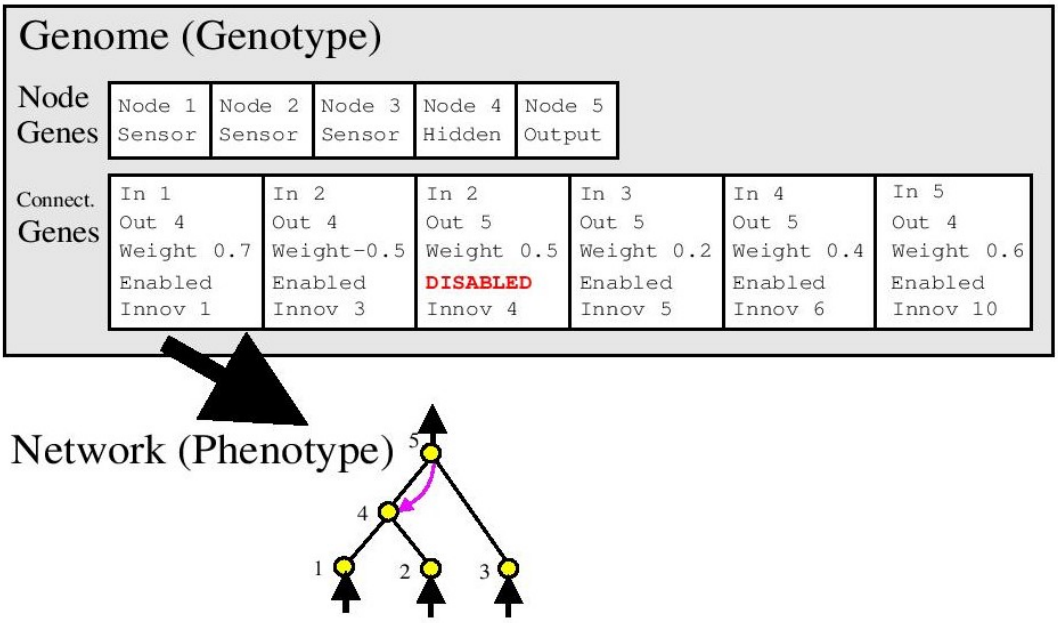

Figure 2: Different kind of genes present in a genome(taken from [2])

The new connection leading into the new node receives a weight of 1, and the new connection leading out receives the same weight as the old connection.

3.2 Tracking Genes through Historical Markings

NEAT tracks the origin of each gene by associating a global innovation number to each new gene formed. Thus, the innovation numbers represent a chronology of the appearance of every gene in the system.The historical markings give NEAT a powerful new capability, effectively solving the problem of competing conventions. The system now knows exactly which genes match up with which. When crossing over, the genes in both genomes with the same innovation numbers are lined up. These genes are called matching genes. Genes that do not match are either disjoint or excess, depending on whether they occur within or outside the range of the other parents innovation numbers. They represent structure that is not present in the other genome. In composing the offspring, genes are randomly chosen from either parent at matching genes, whereas all excess or disjoint genes are always included from the more fit parent.[2]

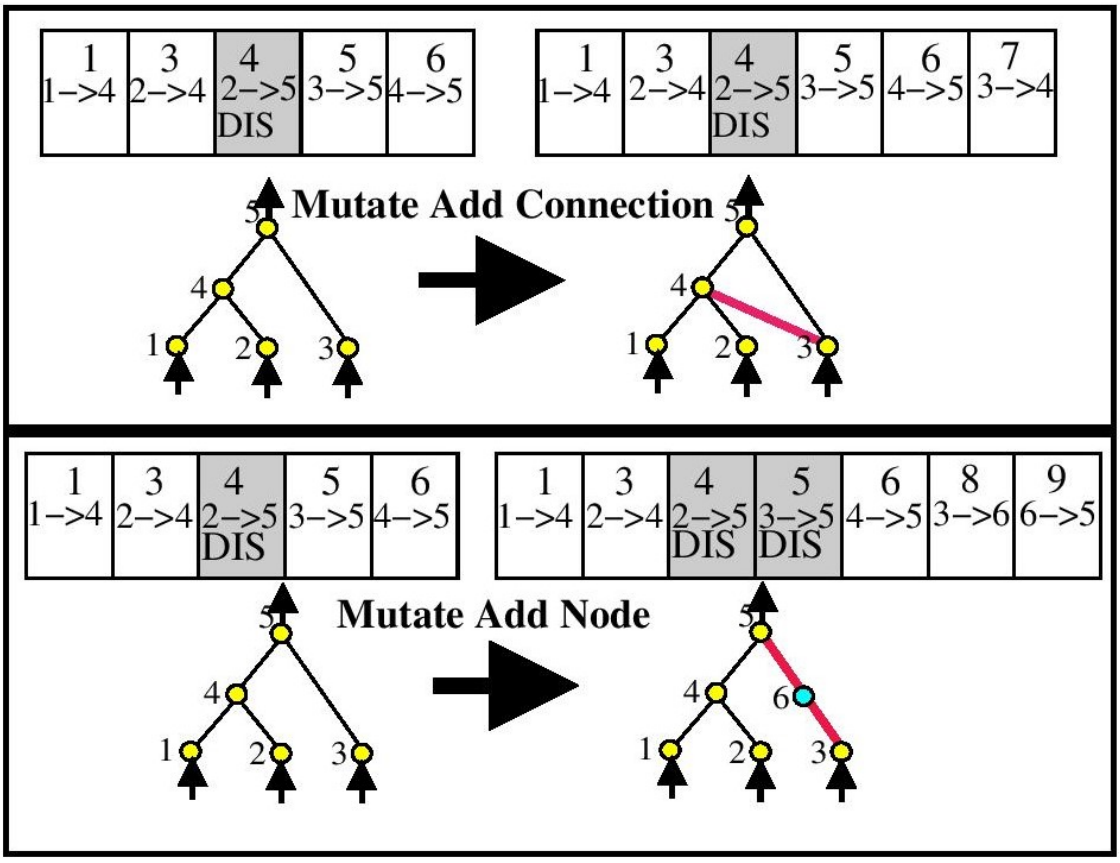

Figure 3: structural mutation in NEAT(taken from [2])

3.3 Protecting Innovation through Speciation

Speciating the population is a solution to one of the problems of the TWEANNs mentioned above. It allows the organism to compete within their own niches rather than the whole population at large. In this way, population innovation gets more time to optimize in their own niches and thus innovations are protected using speciation.

3.4 Minimizing Dimensionality through Incremental Growth from Minimal Structure

TWEANNs typically start with an initial population of random topologies in order to introduce diversity from the outset whereas, NEAT biases the search towards minimal dimensional spaces.This is done by starting out with a uniform population of networks with zero hidden nodes and structural mod-

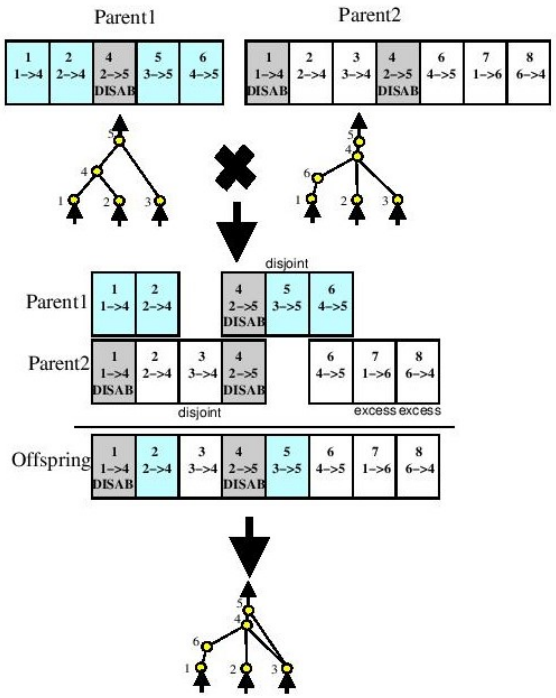

Figure 4: Matching up genomes for different network topologies using innovation numbers.(taken from [2])

ifications are done incrementally of which only those survive which are found useful through fitness evaluation. Since the population starts minimally, the dimensionality of the search space is minimized, and NEAT is always searching through fewer dimensions than other TWEANNs and fixed-topology NE systems.

4 Datasets and Algorithm Used

Datasets were self generated. A 2D region as shown in figure was formed where both the robots were allowed to move around and shoot in any possible way and get feedback thereafter. This was done for over 50 generation. Time taken by each generation varies from 0.8secs to 1000secs depending mainly on the complexity of the fitness function and the number of inputs and outputs. In our case, the inputs were the X and Y coordinate data of both the robots, and two outputs were taken which were used to form a fitness function. The

Figure 5: The red and the green circular objects are the two robots and the two smaller circles are the bullets that are shot by the robots.

idea behind the algorithm is that we define two different fitness function for the two robots. Clearly, since this is a competitive game, as the fitness of one robot increase the corresponding value for the other robot decreases. So, at a give time we define the overall fitness function as the minimum of the two fitness functions. Thus, whichever robot has lower fitness value evolves till it crosses the fitness of the competitor and then other robot evolves. This is done so that one robot doesn't start dominating the other robot and thus they both learn to shoot and escape in a healthy manner.

5 Code Used

The code was implemented in python for which the library used is neat-python.The documentation of this library is still under progress and is not complete because of which the implementation of XOR-example and pole balancing example was studied and modified to work in our situation. the link for the examples can be found at the following link.

- Neat-Python examples: https://github.com/CodeReclaimers/neat-python

6 Results

The results obtained are in the form of videos, each robot performing some basic tasks(the main task was divided into several subtasks which helped in achieving the main goal). Other than the videos, a graph was obtained

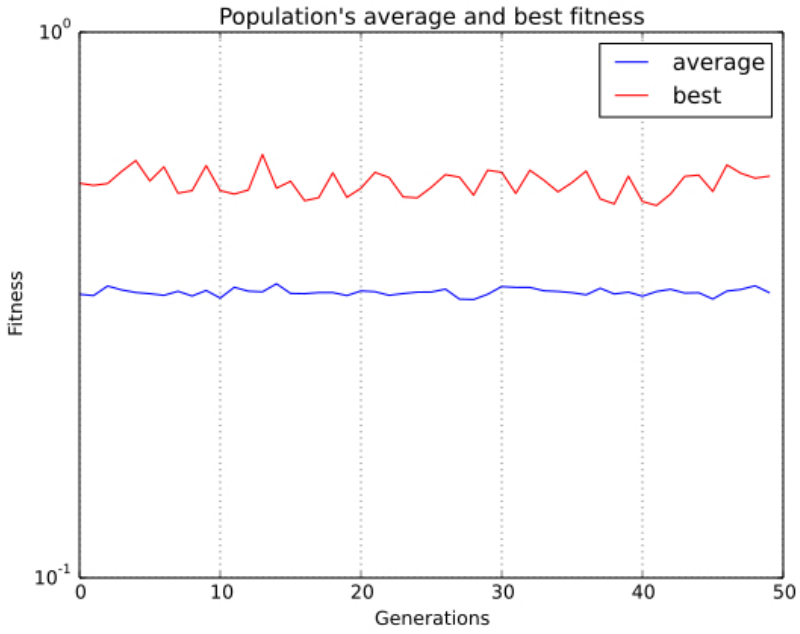

Figure 6: Average and best fitness value as a function of generations evolved

between the fitness function and the number of generation the network has evolved. Also a graph was obtained to observe the different species formed , their size and its variation over several generations.

6.1 Fitness evaluation

The average and the best fitness was evaluated over several generations as shown in the figure. Where the average fitness is found to be almost flat, the best fitness is found to have short peaks. The likely reason being the fact that the minimum of the two fitnesses is taken as the overall fitness. As the minimum of the two exceeds the other fitness value, the other robot starts to evolve which is accompanied by initial fall in fitness value and later builds on it to exceed the other value, and the process continues.

6.2 Speciation

In this subsection, an illustration on how the different species evolved and how populated were they in each generation is provided. Clearly, from the figure, it started with one single species and whenever any mutation resulted in any structure which had a distance greater than some threshold from all

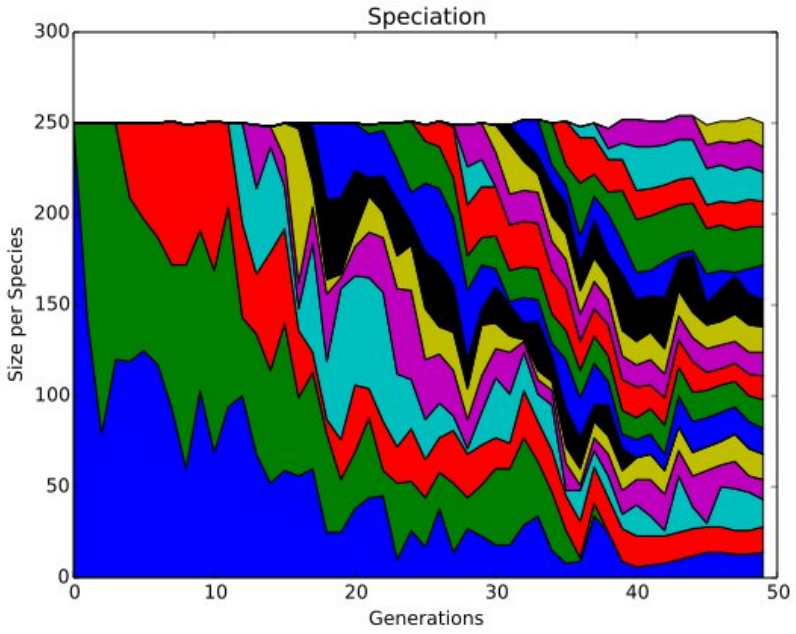

Figure 7: Evolution of different species over 50 generations

the other current species, a new species is formed. More about such structure and threshold is described in [2].

7 Future Work

In this work, we have used only one neural network structure for making both the robots learn to fight against each other. Further work can focus on learning using two different neural structure for the same work. This could be interesting because in the current version, at a time only the one with lower fitness value evolves.

References

[1] Stanley, Kenneth O., and Risto Miikkulainen. "Efficient reinforcement learning through evolving neural network topologies." Network (Phenotype) 1.2 (1996): 3.

[2] Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies." Evolutionary computation 10.2 (2002): 99-127.

[3] Stanley, Kenneth O., Bobby D. Bryant, and Risto Miikkulainen. "Evolving neural network agents in the NERO video game." Proceedings of the IEEE (2005): 182-189.

[4] Reddemann, Katie. "Evolving Neural Networks in NPCs in Video Games."

[5] Angeline, Peter J., and Jordan Pollack. "Evolutionary module acquisition." Proceedings of the second annual conference on evolutionary programming. 1993.

[6] Braun, Heinrich, and Joachim Weisbrod. "Evolving neural feedforward networks." Artificial Neural Nets and Genetic Algorithms. Springer Vienna, 1993.

[7] Dasgupta, Dipankar, and Douglas R. McGregor. "Designing application-specific neural networks using the structured genetic algorithm." Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on. IEEE, 1992.

[8] Fullmer, Brad, and Risto Miikkulainen. "Using marker-based genetic encoding of neural networks to evolve finite-state behaviour." Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life. 1992.

[9] Gruau, Frdric, Darrell Whitley, and Larry Pyeatt. "A comparison between cellular encoding and direct encoding for genetic neural networks." Proceedings of the 1st annual conference on genetic programming. MIT Press, 1996.

[10] Krishnan, Rajendra, and Victor B. Ciesielski. "Delta-gann: A new approach to training neural networks using genetic algorithms." University of Queensland. 1994.

[11] Pujol, Joo Carlos Figueira, and Riccardo Poli. "Evolving the topology and the weights of neural networks using a dual representation." Applied Intelligence 8.1 (1998): 73-84.