Game Bot using NEAT

Akshay Mohan (1301CS05) & Devansh Gupta (1301CS56)

1 Abstract

NeuroEvolution of Augmenting Topologies (NEAT) is a method of neuroevolution, which outperforms the best fixed-topology method on a challenging benchmark reinforcement learning task. The idea behind this project is to use NEAT approach in training neural networks to play several simple computer games. The main purpose is to demonstrate the capability of the approach in learning to play a game without having any prior information about the game.

2 Introduction

NEAT is an approach where neural network evolves artificially over several generations. It is found to be better than fixed topologies neural networks in many reinforcement learning tasks. The fitness function acts as a feedback to the algorithm and it tries to find the best neural network which can maximize the fitness function.
Features of NEAT:

- Genetic Encoding: NEAT's genetic encoding scheme is designed to allow corresponding genes to be easily lined up when two genomes cross over during mating.
- Tracking Genes through Historical Markings:- Whenever a new gene appears (through structural mutation), a global innovation number is incremented and assigned to that gene. The innovation numbers thus represent a chronology of the appearance of every gene in the system.
- Protecting Innovation through Speciation:- Speciating the population allows organisms to compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected in a new niche where they have time to optimize their structure through competition within the niche.
- Minimizing Dimensionality through Incremental Growth from Minimal Structure:- NEAT biases the search towards minimal-dimensional spaces by starting out with a uniform population of networks with zero hidden nodes (i.e., all inputs connect directly to outputs). New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. In other words, the structural elaborations that occur in NEAT are always justified.

The main advantage of using NEAT is that the topologies are minimized and grown incrementally during evolution resulting in higher learning speed than other neural evolution algorithms. The increase in speed is attributed due to the following reasons:

- It allows disparate topologies to crossover in meaningful ways. It is possible become of genetic encodings used in NEAT.
- It protects structural innovation through speciation.
- It minimizes the dimensionality of search space through incremental growth from minimal structure.

2.1 Literature survey

Seth Bling has implemented a game bot on famous game "Mario "using the NEAT approach. He named the game bot as "MARI/O ". He used a population size of 300 and his bot was able to beat the level after 34 generations and 24 hours of training, at which point the bot started playing significantly well as compared to a average human player.

3 Resources

The main idea of the NEAT approach is mentioned in the paper Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies."Evolutionary computation 10.2 (2002): 99-127.. It mentions the NEAT approach in detail and how it is better than the existing neural evolution approaches.
We have used the python library for NEAT. The documentation of the library is provided at NEAT Documentation
Code for Pong Pong Game
Code for MARI/O MARI/O
Emulator for mario BizHawk Emulator
Code for Atari Breakout Atari Breakout
Code for 2048 2048 Game

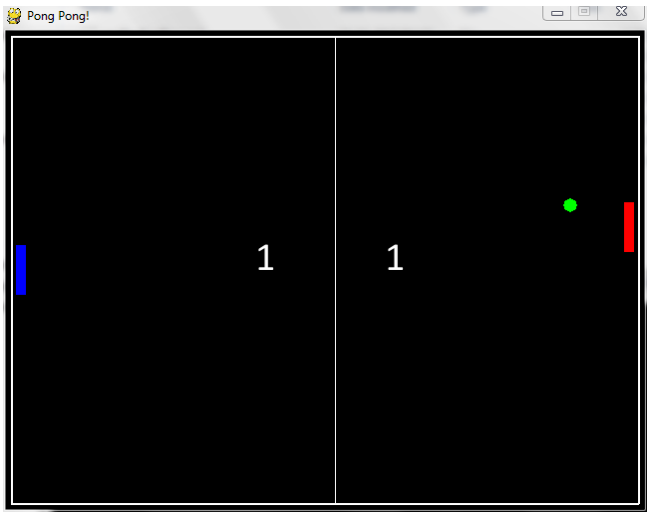
3.1 Work done

We have used NEAT approach in 3 main games:

- Pong - It is a table tennis sports game featuring simple two-dimensional graphics.
- 2048 - The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048.
- Atari Breakout - It is a classic arcade game. The objective of the game is to destroy all the blocks without losing the turn.

The github location for the bots of all the 3 games is : Game Bots using NEAT code
Pong Bot

Figure 1: Pong bot



- The inputs to the pong bot neural network were the x and y coordinates of the enemy tile, the x and y coordinates of the ball and the x and y coordinates of the self tile. Tanh activation function was used and the output was mapped to the two possible moves i.e. up or down.

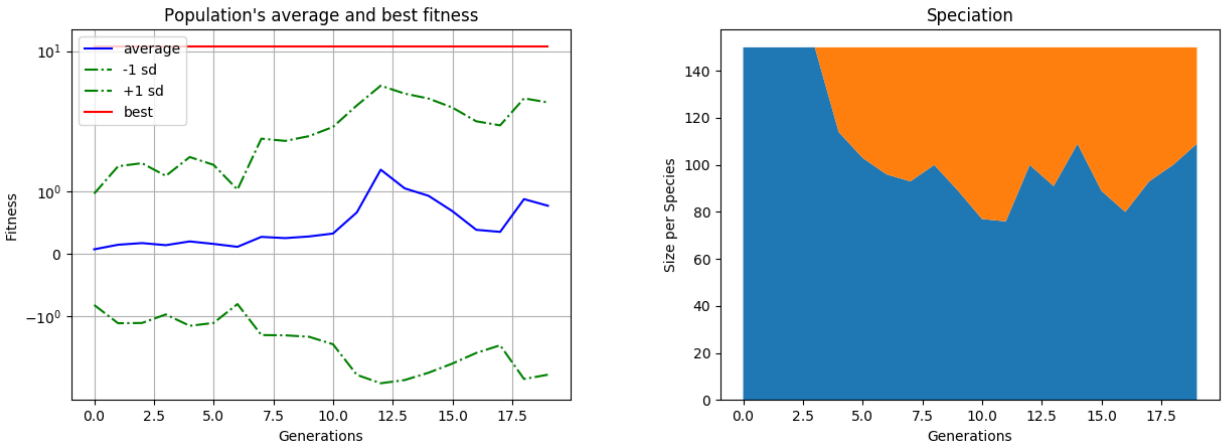
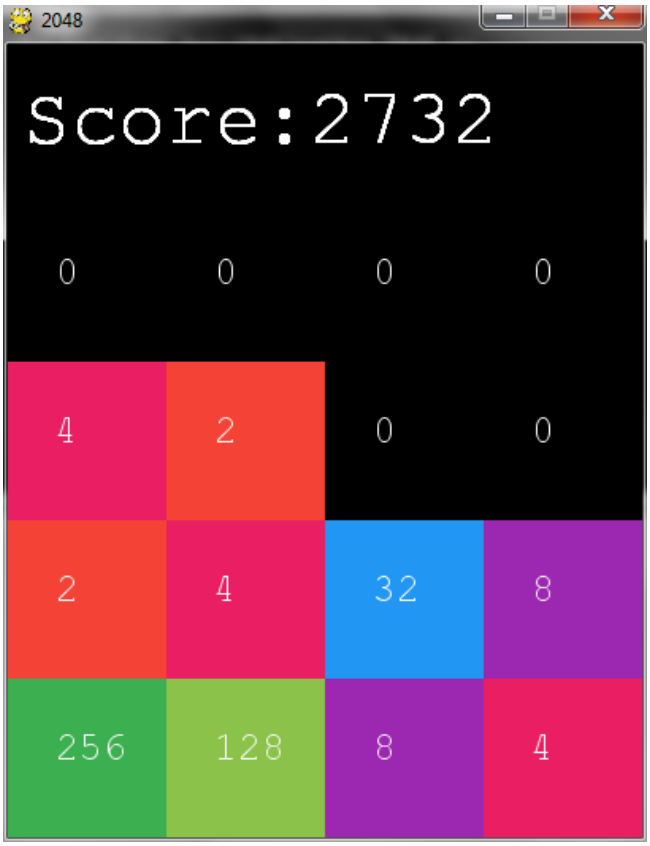


Figure 2: Fitness and species wrt generations for Pong

- Error plot validation set
- The pong bot was tested on two different structures. In the first one the inputs to the neural network were the y coordinate of the ball and the y coordinate of the self tile. In the other structure the inputs were as mentioned in item 1. The second structure learnt to play the game significantly faster than the first one.
- Figures

2048 Bot

Figure 3: 2048 bot



- The 2048 game was taken from the link mentioned in resources. The inputs to the neural network

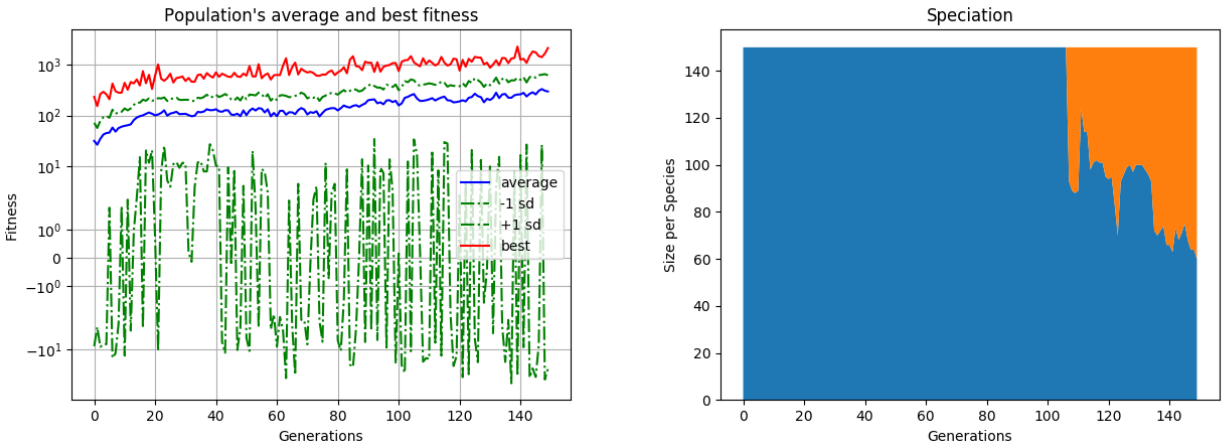


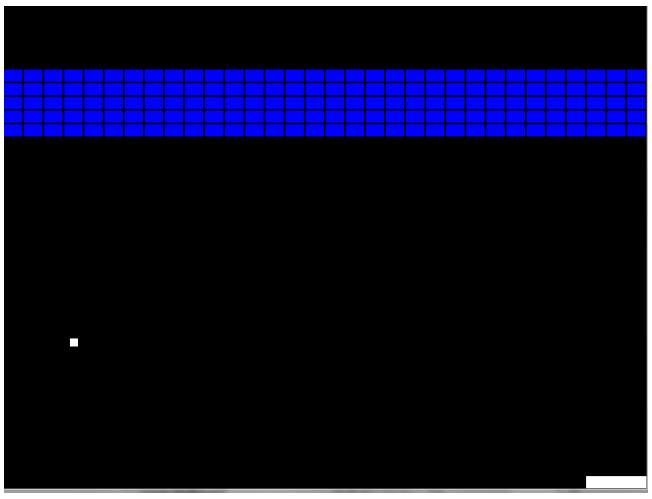
Figure 4: Fitness and species wrt generations for 2048

in NEAT were all the 16 tiles of the game and the corresponding output was the best possible move corresponding to the given state.

- Error plot validation set.
- Fitness function used in final architecture: total_score + 0.01*max_tile Here total_score indicates the total score achieved in the game and max_tile is the value of the maximum tile obtained in the game.
- Different fitness function and neural network structure gave different results. On applying fitness function as the total score obtained in the game without allowing the recurrent neural network the maximum score obtained in the game after training for 150 generations did not exceed 1000 whereas on applying fitness function as "total_score + 0.01*max_tile_value"and allowing recurrent neural networks the maximum score obtained after training for 150 generations exceeded 1500.

Atari Breakdown Bot

Figure 5: Atari Breakout bot



- In the game, a layer of bricks lines the top section of the screen. A ball travels across the screen, bouncing off the top and side walls of the screen. When a brick is hit, the ball bounces away and the brick is destroyed. The player loses when the ball touches the bottom of the screen. To prevent this from happening, the player has a movable paddle to bounce the ball upward, keeping it in play.

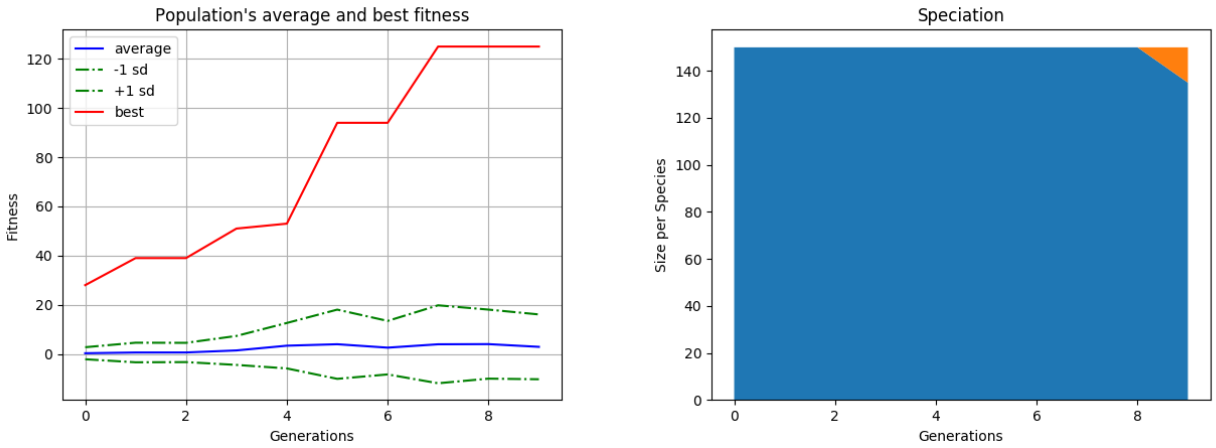


Figure 6: Fitness and species wrt generations for Atari Breakout

- The inputs to the neural network were the x coordinates of the ball and the paddle and the state of each block represented as 1 for present otherwise 0 for destroyed. Tanh activation function was used and the output decides the direction in which the paddle is moved i.e. left or right.

3.2 Future work

For future work we can try various fitness functions for the above mentioned games and compare their results. Also we can try and optimize the mutation parameters to achieve better results.