Using Artificial Neural Networks to Play Pong

Luis E. Ramirez

May 9th, 2014

Abstract

This paper examines the possibility of using Artificial Neural Networks to control AI for simple computer games, specifically pong. Simple games have always been popular, but AI opponents have routines that are easily predictable. Ultimately, the methods attempted in this paper did not create Artificial Neural Net controllers capable of controlling the paddle to hit the ball reliably.

1 Introduction

Pong is one of the first video games ever made. It was developed in 1972 by the Atari Corporation. The first commercially sucessful video game, it is widely regarded as the progenitor to the video game industry. Developers used a simple hard-coded controller for the opponent AI. In 1972, it was not feasible for the While AI and adaptive techniques have andvanced considerably since then, video game developers still often hard-code controllers for even simple games.

NeuroEvolution of Augmenting Topologies (NEAT) is a system introduced by Kenneth Stanley that evolves artificial neural network topologies simultaneously with the edge weights[3]. It has been used to create at least 1 game in the past [2].

Ms. Pac-Man is a classic arcade game from the 80's. A neural network controller to play the game was developed with mixed results [1]. However, Pac-Man is a much more complex game than pong, with intricate mazes unique to every level, and several antagonistic characters to avoid.

Cosmopolis is a free massively multiplayer online game developed for research purposes. It uses adaptive AI for non-player characters in order to maximize player enjoyment. Researchers use the game to learn AI characteristics that make good game user experiences.

Neural networks have been used for playing classic arcade games in the past. Neural networks have also been shown to work in developing novel games [2].It has been shown that humans prefer playing with more human-like AI [4], so the use of adaptive controllers that are able to learn and react to changing strategies has the potential to improve overall user satisfaction and enjoyment from games. Using NEAT, a controller for a paddle will be developed that can predict the trajectory of the ball and defeat hard-coded controllers.

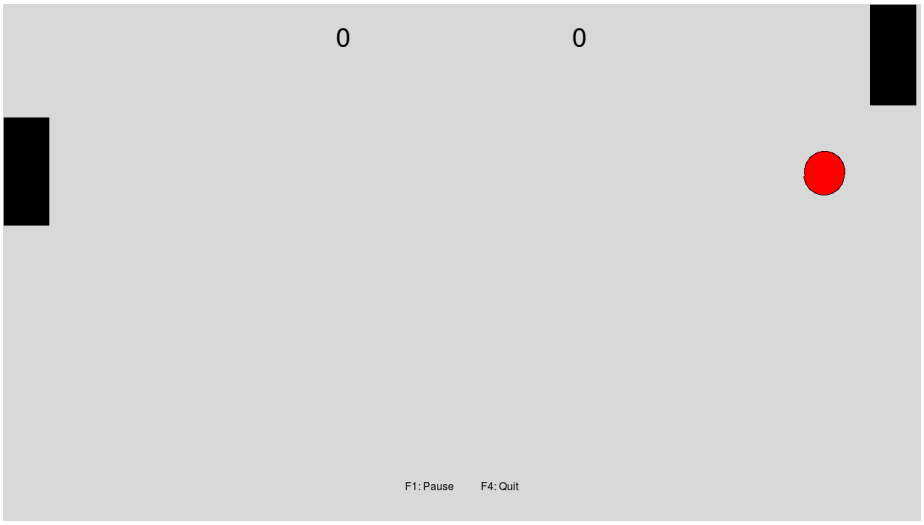

Figure 1: Python Implementation of pong used for the experiment. The hard-coded paddle is on the left, while the NEAT controlled paddle is on the right side of the image.

2 Methods

2.1 Game Implementation

Pong is a simple game in which 2 paddles on opposite sides of a rectangular map have the goal of preventing a ball from reaching the wall behind them. When a ball hits a paddle, it is sent back in the opposite direction. A player scores a point when the ball hits the opponent's wall. I developed a simple version of the classic game, complete with a hard-coded controller for the opponent paddle. This implementation of pong has a rectangular field of play, 1366x768 pixels. The ball has a diameter of 60 pixels, and is contained in a 60x60 bounding box. The paddles are 150 pixels high and 1/20th the width of the game box. The ball has a movement vector that is initialized to a random value between (5,5) and (-5,-5). When contacting the top or bottom wall, the Y value is multiplied by -1, upon hitting a paddle, the X value is multiplied by -1. Upon 5 successful volleys, the speed of the ball is increased by 1 in both the X and Y directions. A winner is determined when the score reaches 5.

2.2 Hard-Coded Opponent

The opponent was hard-coded to follow the ball precisely. Since this would cause the opponent to always win, a maximum travel speed was set. The maximum speed is a factor of 2, ranging from 2 to 20. Difficulty settings of 1-10 refer to these values.

2.3 Neural Network Controller

The NEAT algorithm is given 5 normalized inputs: The center of the paddle (Y value), the center of the ball (2 values), and the motion vector of the ball (2

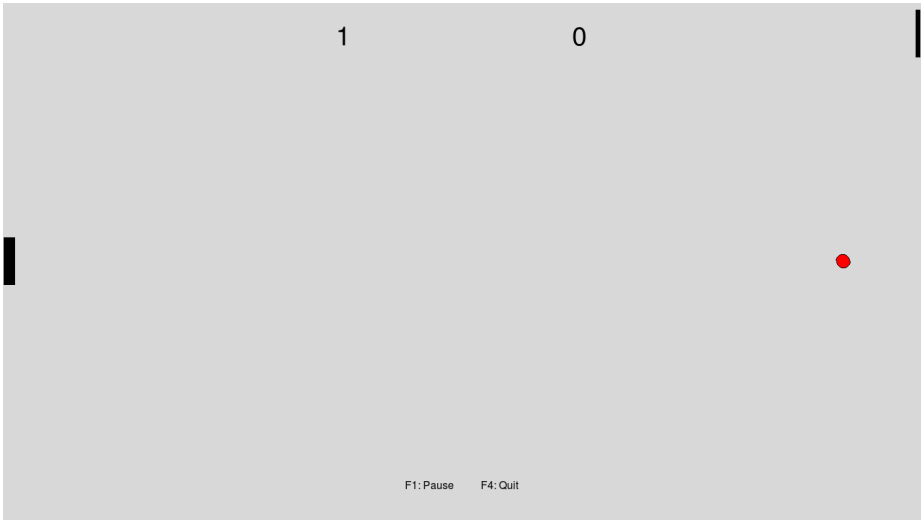

Figure 2: Modified game with smaller paddles and ball used for trials 5 and 6.

values). Except where otherwise stated, there is 1 output that is the movement direction and velocity of the paddle. The output ranges from 1 to -1. Within the range of 0.5 and - 0.5, it is ignored. Outside of this range, it is multiplied by 10 and used as the movement of the paddle for the next time step.

2.4 Evolution

The NEAT algorithm was used to evolve the Neural Network controller over time. Several trial runs were used to test several different fitness functions.

1. The first trial used a fitness function based on the percentage of sucessful hits.

$$F_n = \frac{hits}{hits + misses}$$

2. In the second trial, the distance traveled is calculated, and a penalty of inverse distance travelled is assesed for movement if the paddle hits the ball 5 times or more.

$$F_n = \frac{hits}{hits + misses} + 1 - \frac{PaddleDistance}{BallDistance} * \frac{1}{16}$$

3. In the 3rd trial, fitness is augmented by the final score of the game of pong.

$$F_n = \frac{hits}{hits + misses} + 1 - \frac{PaddleDistance}{BallDistance} * \frac{1}{16} + Score$$

4. In the 4th trial, 2 outputs are added, and the ANN is tasked with predicting the next position of the ball as well as playing the game. Prediction error is passed through a sigmoid and then added to the fitness function.

$$Error(T_n) = 2\sqrt{2} - Dist(Prediction(T_n), Actual(T_{n+1}))$$

$$F_n = \frac{hits}{hits + misses} + 1 - \frac{PaddleDistance}{BallDistance} * \frac{1}{16} + Score + \frac{1}{1 + e^{Error}}$$

5. In the 5th trial, the size of the ball and paddles is reduced significantly. The diameter of the ball is reduced to 20 pixels, and the paddle is only 70 pixels tall.

6. In the 6th trial, the fitness is penalized by a constant for every time step in which the paddle is hugging the edges of the screen.

$$F_n = \frac{hits}{hits + misses} + 1 - \frac{PaddleDistance}{BallDistance} * \frac{1}{16} + Score + ((\frac{1}{1 + e^{Error}} - 1) * \frac{1}{2} + 0.5) * 10^{-7} * Time_{edge}$$

Each individual was tested against hard-coded controllers at difficulties 1, 5, and 10. Averages over these 3 trials was used to compute the fitness for an individual. Balls were initialized to always head in the direction of the NEAT controller. The probabilities of adding a new connection and adding a new hidden node were increased from 15% to 25%.

2.5 Evaluation

Evaluation was similar to evolution, but values examined were the controller's score, the hit percentage, and total distance travelled. Additionally, balls could spawn and head in any direction, to better simulate actual gameplay.

3 Results

The performance of the evolved controllers was very noisy. None of the trials converged to a value or showed an upwards trend.

3.1 Trial 1

After 50 generations, trial 1 had not learned to follow the ball as it travels up and down. It resembles the hard-coded controller, but with a noticeable delay. It does not anticipate, but reacts to the changes in the ball's position. It hits the ball with swiping motions, as it travels from one edge of the screen to the other. It does not seem to react to changes in the ball's X position, and it can never hit a ball that is travelling laterally, and often misses balls with shallow angles.

3.2 Trial 2

The controller behaves similarly to that of Trial 1. After a few generations though, the paddle would begin to shake at the edges of the screen. Additionally, after generation 12, simulations take much longer to complete. With a generation taking a day or more to finish computing. For these reason, this trial was cut short.

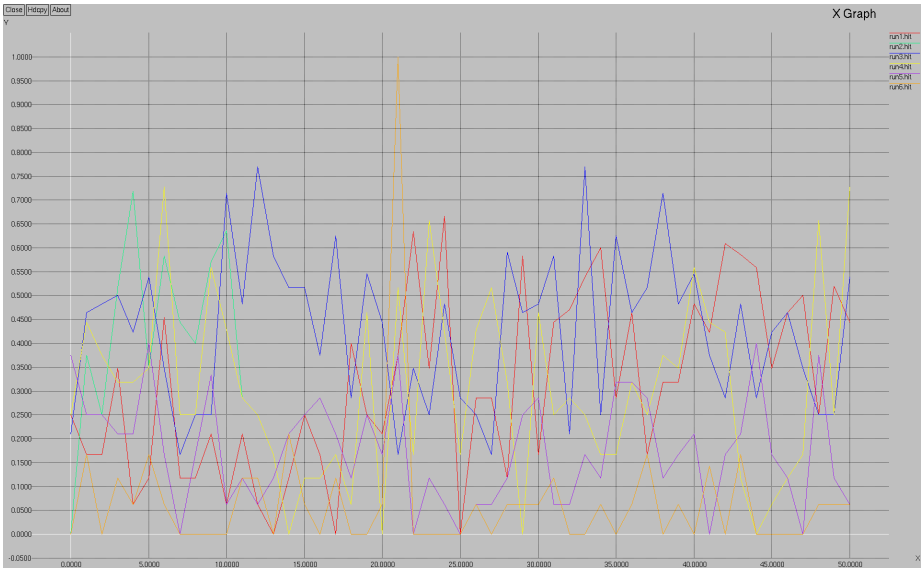

Figure 3: Hit percentage for different trials over time. The values are noisy and do not converge.

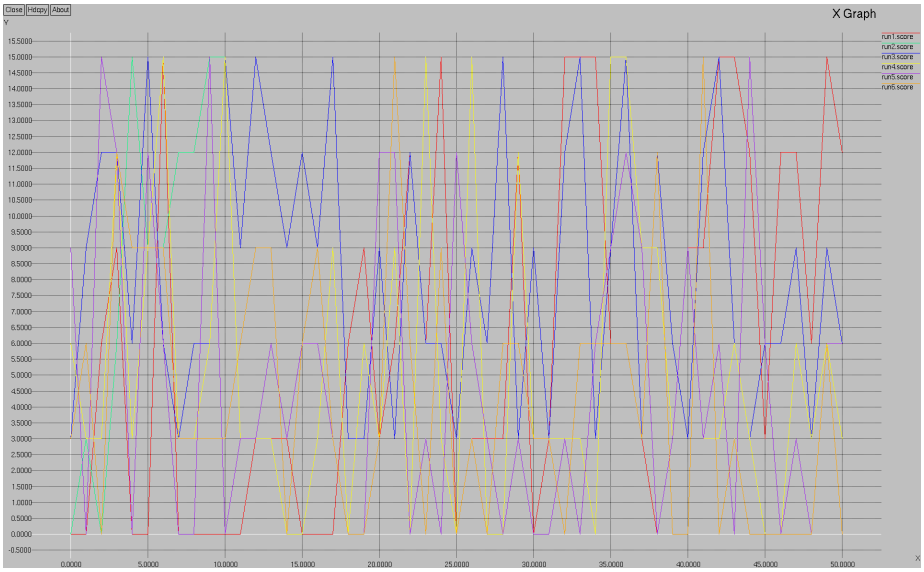

Figure 4: Controller Score over time. A score of 15 means that the controller won in all 3 trials.

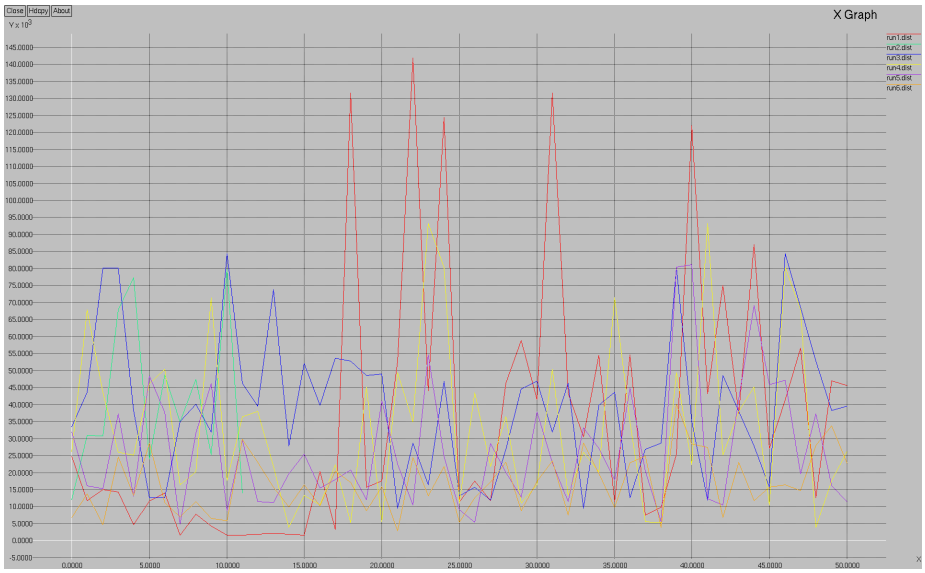

Figure 5: Total distance travelled by the paddle over time.

3.3 Trial 3

The controller behaves similarly to the one evolved in Trial 1 up until generation 15. It then stops ball tracking, lying in wait until the ball is near enough to be hit by the paddle. It will often go up to hit the ball, and then retreat to the nearest edge of the screen. It still misses about as often as the controller from Trial 1. However, at generation 17, the paddle becomes entirely unresponsive to the ball, and shakes in one of the corners indefinitely.

3.4 Trial 4

The controller is able to predict the ball movement more accurately than previous trials. After 25 generations, it began moving from one edge to the other before the ball hit the wall. 2 hidden nodes. 58.3% hit rate.

3.5 Trial 5

This trial behaved similarly to those before it, but was unable to utilize edge hugging as much as previous trials. More precisely, it continued to exhibit the behavior, but to much less effect than before. It demonstrated more of Trial 3's behavior (lying in wait for a ball moving at a shallow angle), around generation 25. However, it would still miss the ball more often than not. 16.6%

3.6 Trial 6

After 25 generations, the paddle was unable to find the ball. It was able to predict where the ball was going to go, behaving similarly to previous trials. The controller exhibited constant shaking, especially near the edges. It was able to anticipate change in direction before hitting a wall like Trial 5. generation

22 had a perfect hit percentage 3, which is almost certainly an outlier. After generation 25, it developed the shaking behavior of trial 3.

4 Discussion

4.1 Fitness Function Motivations

The trials did not yeild significant results, as show in 3, 4, 5. Trials were performed in a staggered pattern, overlapping each other. Problems with each trial were identified, and fitness functions were adjusted in an attempt to correct the erroneous behavior. The first trial developed the swiping technique, which was maladaptive, as the paddle would often move slightly too quickly or slowly, missing the ball. It also followed the ball as the ball's Y position changed. Trial 2 introduced the movement penalty in an attempt to rectify that issue. The movement penalty created indecision and a shaking behavior. After a certain point, the paddle gained more fitness from minimizing movement than from hitting the ball. Trial 3 attempted a different, giving a large boost in fitness to paddles that could actually score on the opponent. This was the most promising of the trials, showing increasing hit percentages until generation 17. However, the shaking behavior came back. In trial 4, the NEAT controller is explicitly trained to predict the ball's movement, in an attempt to get higher hit percentages. Trial 5 reduced the paddle and ball size, in order to disincentivize the edge hugging that is endemic to all of the trials. Trial 6 uses a penalty in an attempt to explicitly prevent edge hugging. This reintroduced the shaking behavior at the edges, which was not as pronounced in trial 5.

4.2 Possible Explanations

In this section, possible explanations for the observed behaviors are explored.

4.2.1 Shaking

All of the fitness functions revolve around the hit percentage. Since the majority of games are lost against the hard-coded controller, eventually, there is a point where

$$1 - \frac{PaddleDistance}{BallDistance} > \frac{hits + 1}{hits + 6} - \frac{hits}{hits + 5}$$

. Therefore, it becomes advantageous to not move, in order to minimize the distance penalty, even if it means not hitting the ball. This is difficult for the ANN to do, so it quickly modulates between positive and negative values. The shaking behavior was first developed when a distance penalty was applied. the

4.2.2 Edge Hugging and Swiping

As noted above, it seems difficult for the ANN to stay still, even though the range of possible values is (-0.5,0.5). In addition, the size of the ball and paddles, means that staying at the edge closest to where the ball will hit the wall is an effective strategy. 420 pixels of the 768 can be covered simply by alternating corners, that's over half of the field. If the switch is timed well enough, the rest of the wall can be protected as well.

4.2.3 Ball Following

It seems to be easy for the ANN to determine in which Y direction the ball is going, moving in that direction means that at least one edge will be covered. More often than not, this is enough to get a hit. Also, the task is such that increasing fitness happens in short bursts. Therefore, the controller will always perform the action that might increase fitness, because it does not detract from the fitness to do so.

4.2.4 Lateral Ball Blindness

Since the NEAT controller uses the ball's Y velocity to determine where to move, and it is not very good at staying still, when the ball is not moving in the Y direction at all, the controller gets confused. This is an uncommon occurrence at worst, so it is a scenario that can be ignored.

4.2.5 Evaluation Noise

All of the problems above contribute to the noise present in the evaluations. The random direction of the ball at start means that any individual reset can be a situation that which the controller is either able to handle very well or very poorly. Very positive or negative Y initial direction creates a situation in which the NEAT controller can perform very well in. So well, in fact, that the rest of the rounds of the game don't negatively affect fitness that much. 3 is a result that is very likely to be caused by overfitting. Increasing the complexity of the inputs and fitness function did little to hurt or help the results. In addition, the higher rates of added connections and nodes could cause high peaks and low valleys as good specimens are mutated into less fit ones because of changing topology.

4.3 Possible Remedies

The current inputs do not seem to be sufficient for NEAT to effectively predict where the paddle has to be. Perhaps using the Y values of 2 corners on the long side of the paddle will be enough information to guide the ball in between them. Also, a warning input could be added, to let the paddle know when the ball is immenently close. Penalizing movement after the ball is a certain distance away could allow for the NEAT controller to learn to anticipate the next region of the wall to defend.

Edge hugging and swiping seem to be caused by a difficulty to stay still. Increasing the range of ignored outputs might solve this issue. It would also reduce the occurrence of shaking. Reducing the size of the ball and paddles, as in trials 5 and 6, would also reduce the advantage of staying on one of the edges. This may also help with lateral ball blindness.

Perhaps the most important change that could be made is an increase of everything in the trials. Increasing the score to win would make the change in fitness more gradual as the controllers got better. It would also force the controllers to take every scenario seriously. If the score to win is 50, then it would greatly lower fitness to miss most of them, even if there were 1 or 2 very good volleys. Increasing the number of games that each individual had to take part in would also achieve this.

5 Conclusion

This experiment demonstrates how poignant of a problem overfitting is when training adaptive controllers. While it may take significantly longer, it is important to thoughroughly explore the search space of the possible inputs with each individual in the population. When it comes to mutation, more does not necessarily mean that results will be obtained more quickly.

It is important to note that a fitness function that trains well for the first few generations may not continue to do so down the line. This can happen for a multitude of reasons, but careful monitoring of resultant behavior is essential for determining when to change fitness functions.

6 Acknowledgements

I would like to thank my professor Lisa Meeden for support and guidance throughout the process of designing and implementing this experiment. John Zelle for the graphics library used to render the game. System administrator Jeff Knerr for an extension to the Zelle graphics library.

References

[1] Simon M Lucas. Evolving a neural network location evaluator to play ms. pac-man. In *CIG*. Citeseer, 2005.

[2] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. Evolving neural network agents in the nero video game. *Proceedings of the IEEE*, pages 182–189, 2005.

[3] Kenneth O Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. *Network (Phenotype)*, 1(2):3, 1996.

[4] Michael Zyda, Marc Spraragen, Balki Ranganathan, Bjarni Arnason, and Peter Landwehr. Designing a massively multiplayer online game/research testbed featuring ai-driven npc communities. In *AIIDE*, 2010.