Learning Breakout using NEAT

James T. Murphy III* Skyler Thomas†

December 15, 2017

Abstract

We present a replication of the learning of a version of Breakout via machine learning, specifically using an evolutionary algorithm known as NEAT. We show that given the current and past few game states as inputs, NEAT can evolve a neural network that can completely clear a game of Breakout, and we identify three eras of learning that occur during training.

1 Introduction

Recent advances in machine learning have spurred both research interest in and popular applications of neural networks. A popular application of particular interest is teaching neural networks to play video games. Such games often involve complex and potentially long-term strategies to play optimally. It was demonstrated in [5] and [6] that deep reinforcement learning can be used to train a neural network to play a wide array of Atari games. Similarly, YouTuber SethBling demonstrated that a neural network could learn via evolution to play the Super Nintendo game Super Mario World [10]. We took inspiration from these projects and have used the same evolutionary algorithm to train a neural network to play a version of the Atari game Breakout.

Breakout is a game by which the user moves a paddle to the left and right a fixed distance from the bottom of the screen in an attempt to destroying as many bricks as possible by bouncing a ball off the paddle. For each brick that is hit, a point is awarded. This continues until either the player misses the ball or all 520 bricks are hit.

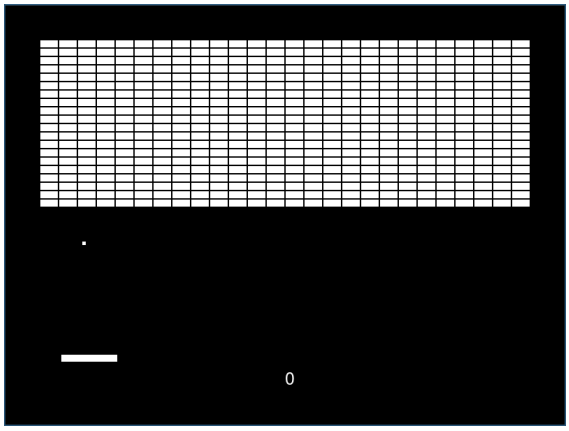

Figure 1: The version of Breakout we used.

The evolutionary algorithm we employ is NeuroEvolution of Augmenting Topologies (NEAT). NEAT is a type of genetic algorithm that follows biological metaheuristics. In NEAT, neural network nodes and weights are modeled as genes. Genomes are randomly mutated by adding/deleting nodes, adding/deleting connections, changing connection weights, etc. After mutation, individuals are ranked by their fitness at performing a certain task, in our case playing a game of Breakout. The best performing individuals are allowed to mate using a genetic crossover algorithm, producing children that compose the next generation of individuals. This process is repeated until some fitness threshold is reached, typically that the most fit individual succeeds at a particular task, in our case achieving a maximum score in Breakout.

*jamesmurphy@math.utexas.edu
†skylerthomas0@gmail.com

2 Results

We find there are qualitatively three eras of evolution of the network:

1. *Pre-ball-tracking era*: the network typically does nothing or makes a few small adjustments at the very beginning of the game, scoring a minimal number of points usually 0 to 30.

2. *Ball-tracking era*: the network definitively makes an effort to keep the paddle horizontally aligned with the ball. It clears a sizeable portion of the blocks, and only fails in edge cases or by infinitely looping without clearing some remaining blocks. It typically scores 100 to 400 points.

3. *Aiming era*: the network tracks the ball as before, but also makes sporadic adjustments to aim at remaining blocks. It clears all or nearly all of the blocks, achieving a score from 515 to the maximum 520.

A large percentage of the time, around 90% of attempts, the NEAT algorithm stagnates at a local maximum fitness somewhere in the first two eras. Simply repeating the experiment a sufficient number of times eventually yields a trained network that can achieve the maximum score. The resultant network is typically very small, having 5 or less non-input nodes and under 15 total connections.

It turned out to be critical to the success of the NEAT algorithm that several previous game states (3 is sufficient) were included as inputs to the neural networks. Giving only the current state, including the direction of the ball, as input was not sufficient in any of our experiments to produce a network that could even reliably track the ball.

Also, it was necessary to train the network with a fixed initial condition (initial ball speed and angle). We found that randomizing the initial condition made it too hard for evolution to select for individuals that could track the ball because, for instance, a network that pressed no buttons at all could earn sometimes 50 points with a very lucky initial condition. Although the initial condition ultimately used was deterministic, the final trained networks do not simply memorize a sequence of inputs that work for the initial condition. Indeed, the final networks turn out to be far too small for such memorization to be possible. Further, testing the most fit network with new initial conditions proved that its performance does generalize.

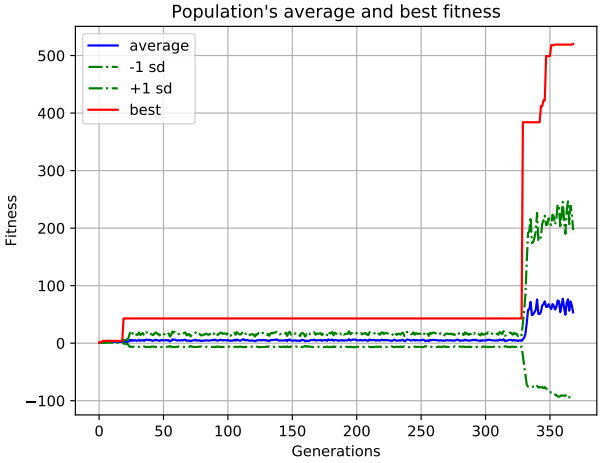

Figure 2: A successful training experiment. In this case, the pre-ball-tracking era lasts until around generation 325, where the network abruptly learns how to track the ball. The ball-tracking era then lasts until around generation 350, where the network learns to avoid missing stray blocks. The aiming era is reached in the last few generations. Generations, each of 200 individuals, took an average time of 2.835 seconds to evaluate running on an Intel Core i7-4790K @ 4.00GHz in parallel on 6 cores. The total runtime was around 17.5 minutes.

Another pitfall that was avoided was giving too many inputs to the networks. If the game state fed to the network included information about all 520 blocks, then regardless of whether the game history was given, none of our experiments produced a network that could reliably track the ball. The vast majority of networks trained in this manner did precisely nothing, making no button presses at all. On the other hand, training was able to produce an aiming era individual when the density of blocks remaining in a few predetermined regions were passed as inputs. Training also succeeded when the game state passed to the networks included no information about the remaining blocks whatsoever, i.e. a network that saw only the paddle and ball could still achieve a maximum score.

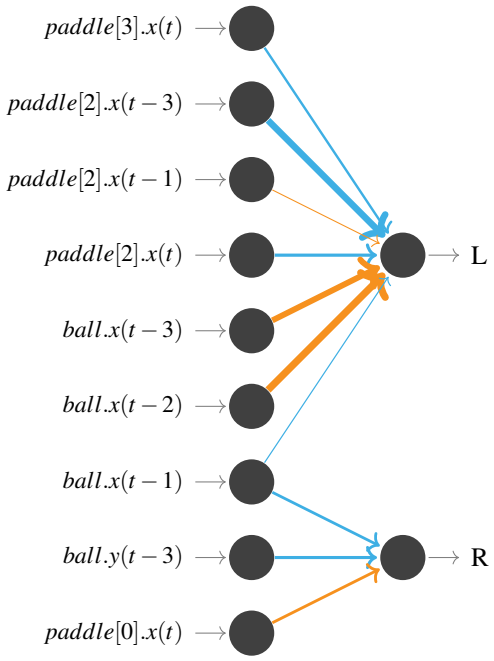

Figure 3: The most fit genome in one experiment with the network blind to the remaining blocks. Notice how it approximately hits left if the ball is left of the paddle, and it approximately hits right if the ball is right of the paddle. See the methods section for an explanation of the variables.

3 Methods

The source code and configuration of our experiments are publicly available at [7].

We modified a Pygame implementation (cf. [11]) of Breakout (cf. [9]) to allow human mouse input and human keyboard input (for testing purposes), or automated keyboard input. Then we used NEAT-Python (cf. [1]) to run the NEAT algorithm.

To allow the neural network to play Breakout, each frame a representation of the game state is computed and passed as the input layer to a neural network. The neural network has two outputs "L" and "R", which are binarized and taken to indicate whether the left or right arrows are being pressed by the neural network.

The representation of the game state includes the following information:

- $ball.x$, $ball.y$: the x and y coordinates of the ball, normalized as a percentage of the width of the game
- $ball.xdir$, $ball.ydir$: the x-direction ± 1 and y-direction ± 1 of the ball,
- $ball.angle$: the angle of the ball, normalized as a percentage of 2π
- $paddle[0].x$, $paddle[1].x$, $paddle[2].x$, $paddle[3].x$: the x coordinates of the left endpoints of each of the sections of the paddle, normalized as a percentage of the width of the game
- the history of the previous four up to a fixed number of time steps in the past
- (potentially) the density of remaining blocks in e.g. each of the four block quadrants or similar fixed configuration of predetermined regions of blocks.

After carefully choosing the parameters of the algorithm, a neural network is trained to play Breakout using NEAT with the final score of playing a game being the fitness function. Success of the training is *very* sensitive to changes in hyperparameters of NEAT, so the interested reader should view the sample training configuration file in [7].

4 Discussions

Although NEAT did eventually produce neural networks that can achieve the maximum score in Breakout, NEAT was very prone to stagnation at a local maximum fitness. We suspect that this was mainly due to the nature of how training was setup. Fitness was evaluated by allowing a network to play a whole game of Breakout start to finish. This means that small changes in the network could result in drastically different scores, making it hard to make the kind of incremental progress that evolution can select for easily. We suggest that anyone interested in repeating this experiment make the following changes to the training setup:

1. Generate initial board setup, paddle location, ball location, speed, and direction randomly.

2. Make the fitness of an individual the score that the network receives in 5 seconds of simulated gameplay, averaged over 10 trials.

3. Optionally, change the scoring mechanism to assign γ^n points for a block destroyed at frame n in training, where $\gamma < 1$ is very close to 1. This encourages networks not only to destroy the blocks, but to do so as quickly as possible.

We suggest the previous changes in order to make it easier for evolution to select for incremental progress. Indeed, in both [5] and [6] these aspects are already present, though no evolutionary algorithm is used, and apparently much greater success is achieved there than here. For Breakout, the changes were not strictly necessary, but originally we attempted to use an analogous setup to train neural networks to play Tetris. For Tetris, we had no success, but we believe that it may be possible to learn Tetris using NEAT once the suggested changes are made.

Future work could make a direct comparison between the performance of the most fit individual trained by NEAT along with several different methods such as Deep Q-learning or other evolutionary algorithms. More work is needed to elucidate what aspects of the learning setup are the most important to allow long-term strategy to develop. For instance, does evolving the network topology provide any benefit over choosing a fixed network topology?

5 References

[1] CodeReclaimers. Neat-python. `https://neat-python.readthedocs.io`, August 2017.

[2] J. J. Grefenstette, D. E. Moriarty, and A. C. Schultz. Evolutionary algorithms for reinforcement learning. *CoRR*, abs/1106.0221, 2011.

[3] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *CoRR*, abs/1711.00436, 2017.

[4] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *CoRR*, abs/1709.06009, 2017.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[7] J. Murphy and S. Thomas. Neat breakout. `https://github.com/kb100/NEAT_Breakout`, December 2017.

[8] S. Picek and M. Golub. Dealings with problem hardness in genetic algorithms. 8:747–756, 05 2009.

[9] F. Primerano. Breakout. `https://github.com/Max00355/Breakout`, December 2015.

[10] SethBling. Mari/o machine learning for video games. `https://www.youtube.com/watch?v=qv6UVOQ0F44`, June 2015.

[11] P. Shinners, R. Dudfield, M. von Appen, and B. Pendleton. Pygame. `https://www.pygame.org/`, January 2017.

[12] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 569–577. Morgan Kaufmann Publishers Inc., 2002.

[13] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.