Creating Diverse Fighters in Super Smash Bros. Melee using NSGA-II

Gabriel Richardson Alex Flores-Escarcega Carlos Valdez Jorge Garcia

Abstract

In recent years, massive improvements have been made surrounding game-playing artificial intelligence. Simple board games to complex multiplayer battle games, such as DotA, have been solved with reinforcement learning approaches. Although reinforcement learning has become the most popular method for solving these tasks, approaches that make use of neuroevolution have still shown promising results. In this work, we investigate the performance of evolutionary algorithms on the popular console fighting game Super Smash Bros. Melee (SSBM).

1 Introduction

Although machine learning may be the most popular approach when it comes to creating artificial intelligence in video games, it certainly isn't the only approach. Neuroevolution has shown to be just as promising, and in this paper we investigate its performance on the popular fighting game SSBM. Because we are trying to create a diverse set of agents that incorporate both offensive and defensive play into their fighting styles, we have to optimize multiple objectives. Offensive agents should deal more damage, and defensive agents should take less damage. As such, we want to maximize damage dealt and minimize damage taken. We make use of Non-dominated Sorting Genetic Algorithm (NSGA-II) in order to optimize the fitness values and maintain a diverse set of solutions.

2 The SSBM Environment

The SSBM environment poses several challenges that increase the complexity of evolving a set of agents.

Firstly, there is a wide range of diversity with a total of 26 characters and 29 stages. To combat this issue, all training takes place on a single stage, and agents are only trained against a single character.

Furthermore, the game is run on an emulator, making the training process very slow. This is not a chief concern as of now, but in the future we hope to make progress to speed this process up.

Lastly, SSBM is a multi-player game. This means that success is not an absolute measure given by the environment, but rather it is defined relative to an unpredictable opponent. Although we can't eliminate this variability completely, we can reduce it by pinning the agents against the game's more predictable built in CPUs

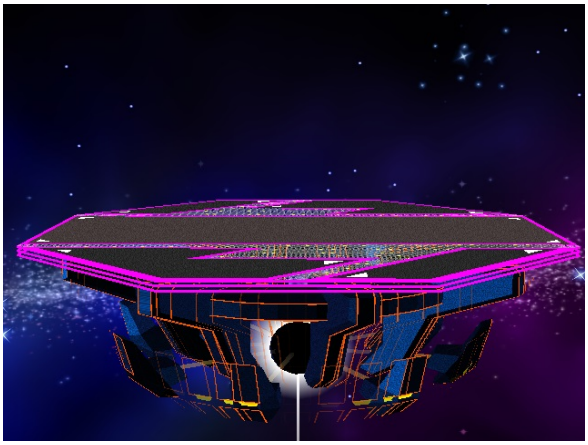

Figure 1: Final Destination, the stage where agent training takes place

2.1 Game State, Actions, and Rewards

In our approach, we are using a fully connected feed-forward neural network to control the agents actions. For our inputs we are simply reading in the x and y coordinates of the agent and its opponent, as well as the direction each character is facing. There are several other states we could read in from the game's memory such as velocity and action state. We chose to avoid these game states for now, as we don't want to over complicate the neural network.

The GameCube controller consists of two analog sticks, five buttons, two triggers, and a directional pad. To make things simpler, we reduce the inputs to 9 directions on the analog stick and 5 buttons. The outputs of our neural network are mapped to 28 discrete actions made from combinations of these controller inputs. This does not encompass all of the game's actions, however for the purposes of our project it is sufficient.

In SSBM, characters deal damage on each other in terms of percent. A character's percentage is initially at 0 when the game starts, and every time they get attacked the percent is increased according to the corresponding attack. As a character's percentage increases, each subsequent attack to the character will send them further away from the stage, thus decreasing their chance of survival. We use a multi-objective optimization approach, where we try to minimize the damage an agent receives, and maximize the damage it deals. In order to ensure diversity, we punish the agent for staying in one position for too long and we reward it for using a variety of moves. The agent is also punished when it turns its back to its opponent.

When a character is KOed, they spawn back in the middle of the stage. Traditionally, a game is over after a player is KOed 4 times. We set the game mode to infinite time and change agents after 1200 frames. When a new agent is swapped in, the game is restarted and percents are reset to 0. This ensures that all agents are on a level playing field, and it simplifies navigating through the SSBM menus.

3 Methods

Our main goal is to find a diverse set of agents that are optimized to minimize the damage they receive and maximize the damage they deal to their opponent. To do this, we make use of a fast and elitist multi-objective genetic algorithm: NSGA-II.

3.1 NSGA-II

NSGA-II is very effective at finding a diverse set of solutions when optimizing for multiple objectives. The algorithm follows as such:

1. Randomly initialize the population.
2. Chromosomes are sorted and placed into fronts based on Pareto non-dominated sets. Chromosomes are ranked based on euclidean between solutions, within a Pareto front. Non crowded solutions are given a higher preference during selection. This is done in order to diversify the solution set.
3. The best N (population) chromosomes are chosen from the population and placed into a mating pool.
4. In the mating pool, selection, mating, and mutation is done.
5. The mating pool and current population is combined. The resulting set is sorted, and the best N chromosomes make it into the new population.
6. Loop to step 2, unless you've reached the maximum number of generations.
7. The solution set is the highest ranked Pareto non dominated set from the latest population.

Our current implementation makes use of DEAP's toolbox algorithms for selection, mating, and mutation. We find the crowding distance using selNSGA2, and we select agents for the offspring using selTournamentDCD. Mating is done with cxSimulatedBinaryBounded which executes a simulated binary crossover that modifies the input individ-

uals in place. We also use mutPolynomialBounded, the polynomial mutation as implemented in the original NSGA-II algorithm for mutation.

4 Results

4.1 Benchmark

To check our NSGA-II implementation, we found several evaluation functions to test it against. Namely, the Zitzler-Deb-Thiele's (ZDT) Functions 1, 2, 3, and 4. All 4 functions converge to the optimized front within 250 generations.

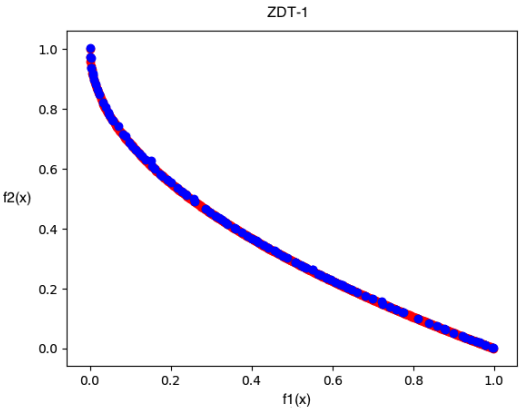

Figure 2: ZDT-1, converged

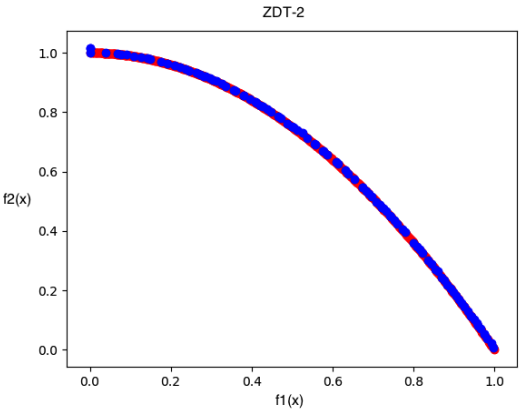

Figure 3: ZDT-2, converged

4.2 Agent Training

After solving the benchmark problems, we felt confident moving forward with implementing NSGA-II alongside SSBM. The evolution process remains the same as in the

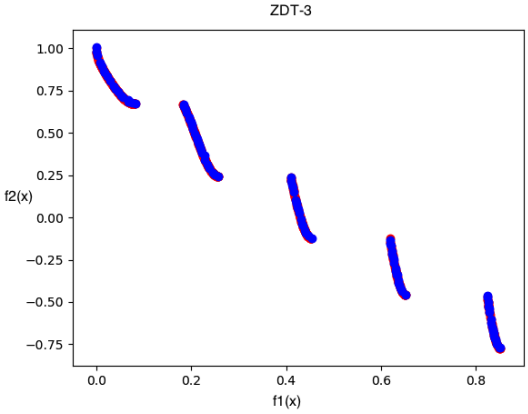

Figure 4: ZDT-3, converged

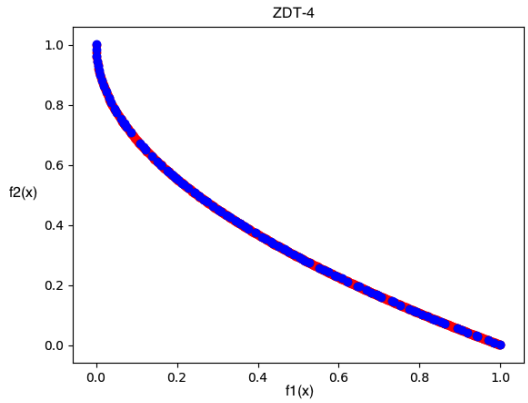

Figure 5: ZDT-4, converged

benchmark problems, and fitness evaluation is as previously detailed. While we were not able to create agents that were competitive with top level players, we did find success in creating a variety of agents with different play styles. Figure 6 shows this diverse set of solutions, along with their trade-offs.

5 Discussion and Future Work

There are still several areas of our work which can be improved upon. Right now, training speed is our biggest obstacle. Emulating the game is very slow, and it could be made more efficient if we could run multiple agents in parallel. It might also be more efficient to train agents against each other, instead of against the game's CPUs.

This brings us to issues with the competitive viability of our agents. Because they

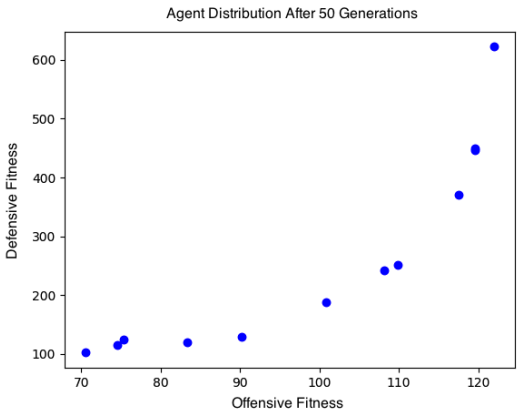

Figure 6: Distribution of agents after training for 50 generations

are only training against the game's CPUs, they are not fit to play against smarter opponents. One option for improving their competitiveness is to train them against the CPUs in 20XX, a modded version of the game with tougher CPUs.

As far as alternate implementations go, it would be interesting to see how performance improves with different neural networks. Recurrent neural networks have proven to be very effective with game-playing tasks, and I'd like to see how it handles a game as complex and variable as SSBM.

6 Conclusions

In this work, we introduce multi-objective optimization to a new and complex environment, Super Smash Bros. Melee. We discuss the advantages of NSGA-II, and its effectiveness in finding a diverse set of optimized solutions. We test our implementation by optimizing several different ZDT functions. Finally, we incorporate our algorithms alongside The SSBM environment, in order to train a diverse set of offensive and defensive fighters. While we did find a diverse set of solutions, the agents are not good enough fighters to compete with human players. Moving forward, the issue of competitive viability could possibly be solved by training the agents in parallel against more difficult opponents.

References

Kalyanmoy Deb. 2012. *Advances in Evolutionary Multi-objective optimization*, Springer, Berlin, Heidelberg.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, T. Meyarivan *A fast and elitist multiobjective genetic algorithm: NSGA-II.*

Vlad Firoiu, William F. Whitney, Joshua B. Tenenbaum 2017. *Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning*,