AI Agents for Pokémon Battle

Nisarg Patel

Khoury College of Computer Sciences, Northeastern University, Boston, MA
patel.nisargs@northeastern.edu

Abstract

In this project, I implemented various strategies to select best possible action for each turn of a Pokémon Battle and compared the performances. I implemented Greedy Agent, Minimax Agent, Expectimax Agent, and Approximate Q-Learning Agent. Out of these agents, the results showed that Expectimax performed well against opponent with Random Agent. Minimax performed well against opponent with Greedy Agent and Q-learning Agent performed well against both.

Introduction

Pokémon is a famous video game franchise where player owns creatures known as Pokémon and completes a storyline. One of the most important aspects of these games is Pokémon Battle, a 2 player turn based battle, which is also the format used to play the Pokémon Competitive.

Each Pokémon has a variety of features like types, base stats (health (HP), attack, defense, special attack, special defense, speed), abilities, and held items. Each move also has features like type, damage, accuracy, power points, etc. Currently there are more than 800 Pokémon and 800 moves and 18 different types in Pokémon games. Types of moves and Pokémon determines whether a move with a particular type will be resisted, normally damage or be super-effective to opponent Pokémon's type combination.

In a Pokémon Battle, each player has at most 6 Pokémon and each Pokémon knows 4 moves. Both players have 1 Pokémon in front. At every turn, a player can either select one of the moves to damage the opponent or switch to any of the other Pokémon which has not yet fainted. A Pokémon is fainted if it's HP drops down to 0. A player loses if every Pokémon of the player is fainted.

Given the player's and opponent's Pokémon and moves that each Pokémon knows, this project aim at finding the optimal strategy to maximize the probability of winning. Considering that actual Pokémon Competitive has a lot of factors and thus state space increasing exponentially, I included many constraints like only damaging moves, no abilities, no held items and more. Since each Move has various secondary effects, I only considered the damage output from each move.

Given these conditions, I developed various agents to select an action and compared performance of each of them. I also tuned some parameters to get the best possible accuracy for different AI agents.

This problem is popular due to the vast options of combinations of moves, Pokémon and their factors exist and Pokémon Competitive has also gained much popularity since its start. Along with it, there is only limited research done to create an optimal agent for Pokémon. The possible state space of Pokémon is extremely large just for a single turn considering all the Pokémon and moves.

Background

A rational agent decides an action from the list of available actions that results in the best possible outcome. For Pokémon Battle, at each turn, the possible actions are to attack using one of the 4 moves or to switch to any of the remaining Pokémon. In this project, I implemented four agents using different strategies.

A **Greedy Strategy** performs an action based on the current state of the game and always follows a given policy. The policy could be anything that leads to best outcome. A policy for Pokémon Battle could be to select a move that will do maximum damage or to switch to other Pokémon that will resist the damage most.

A **Minimax Search Strategy** takes an optimal action assuming that the opponent also plays optimally. A minimax agent will pick move that maximizes the player's reward given that opponent is playing to minimize it.

Minimax algorithm works well against opponents that always pick best move possible, but this is not observed a lot in real world. To factor in the probabilities of taking each action, an **Expectimax Search Strategy** selects an action that maximizes the expected value of the game.

In Reinforcement learning, the importance of each action during a state is learned from experience in a learning phase and similar actions are taken when playing a new game. **Approximate Q-Learning** with linear function ap-

proximation represents these Q-values as a linear sum of weighted features. A feature is a property of a particular state and action. In Pokémon Battle a feature can be the remaining HP of Pokémon or a particular stat of Pokémon like Attack or Defense.

Related Work

There are not many attempts made to create AI agents to play Pokémon Battle due to complexity of the problem. Many different condition-based games are attempted as a course project in different universities. Almost all the attempts use one or more of the four strategies that I implemented for this project.

Some different strategies could be applied to solve this problem including machine learning techniques of linear regression or neural networks. Many simple conditions Pokemon Battle can also be solved using uninformed search algorithms. If no switching is allowed, one can use Breadth First Search or A* with minimum damage heuristics to select the best possible damaging move for each turn.

Since the project aim to reduce the game conditions, BFS and A* cannot be used due to a very large state space. For the same reason Q-values were needed to be approximated instead of representing Q-values as a state-action pair table.

Artificial Neural Network for action selection has also shown positive results in Pokémon Battle but due to limited knowledge in machine learning and time constraints I was not able to implement it.

Approach

The project required to implement a simulator for Pokémon Battle from scratch. The first step was to collect the data about every Pokémon including their types, stats and moves they can learn. Data of moves and type matchups were also required. All the data required by the project was available at Kaggle.

I implemented an MVC console simulator that randomly selects a predefined number of Pokémon for each player with 4 random moves which they can learn. I implemented a Manual Agent that uses console for user input to select actions on each turn.

The damage is calculated in a similar way as it is calculated in actual Pokémon games. The faster Pokémon gets to attack first and if a Pokémon is fainted, a new Pokémon should be switched.

Following a working simulator, I implemented the following strategies to select an action:

Epsilon Greedy

This strategy follows a greedy strategy with a small chance of selecting a random action. The greedy strategy used was to select the move which will do most effective damage based on type matchups. The agents selects this move most of the time but with a small probability of (ε), the agent selects a random move. This the closest representation of how an average player plays a Pokémon Battle if he knows the type matchups. I used the value of (ε=0.2) for the purpose of result analysis, but the value can be changed before starting each game.

Minimax

The Minimax Agent used minimax algorithm with a depth of 2 to compute the best action. Different evaluation functions were tested like number of alive Pokémon or total health remaining of the current Pokémon. But a better evaluation function was the difference between the total damage given to the opponent and the total damage taken (DamageDiff).

$$eval(s) = \sum self.damage - \sum opponent.damage$$

Expectimax

The Expectimax Agent used expectimax algorithm with similar parameters as by minimax agent. Depth of 2 and evaluation function as the difference between the total damage given to the opponent and the total damage taken.

Approximate Q-Learning

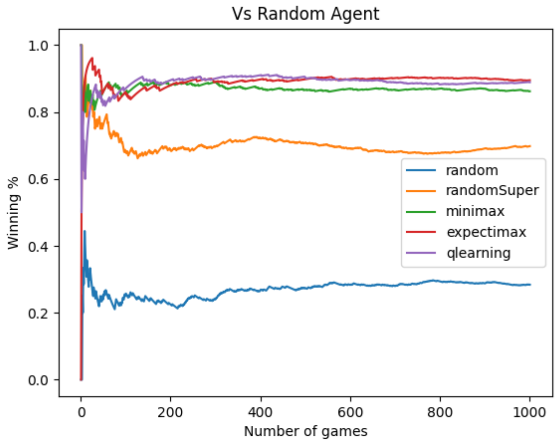
Many features were selected to represent the model and action in best possible way. Some of the features were player Pokémon's stats, opponent Pokémon's stats, type multiplier and damage. A total of 18 different features were selected. State-only rewards were used: 0 if game not over, 100 for player winning states and -100 for losing or drawing states. Number of learning games were set as 1000 but can be changed. Action selection in learning state used epsilon-greedy strategy based on current Q-values. The parameters used for result analysis were epsilon(ε)=0.3, learning rate(α) = 0.5 and discount factor(γ) = 0.7.

Experiments and Results

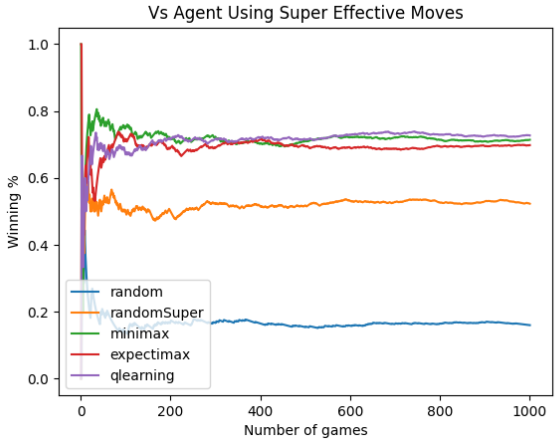
Performance

The four agents simulated 1000 games each against opponent with Random Agent and Epsilon-Greedy Agent. For the opponent with Random Agent, Expectimax Agent was able to perform the best with 90% wins. Q-learning Agent and Minimax Agent also performed well with 87% and 86% respectively while epsilon greedy agent won 70% of

games. With Epsilon-Greedy Opponent, Q-learning agent was able to perform best with 74%, followed by minimax and expectimax with 72% and 70%. Epsilon Greedy agent was able to win 52% of the games. Thus, there was not much of the difference observed in win percentage between Minimax, Expectimax or Q-learning. The Q-learning agent edged slightly ahead of the other two when combining stats of both the agents. Expectimax Agent performed better against Random Agent whereas Minimax Agent performed better against Greedy Agent. This is in accordance with the way minimax and expectimax are implemented.



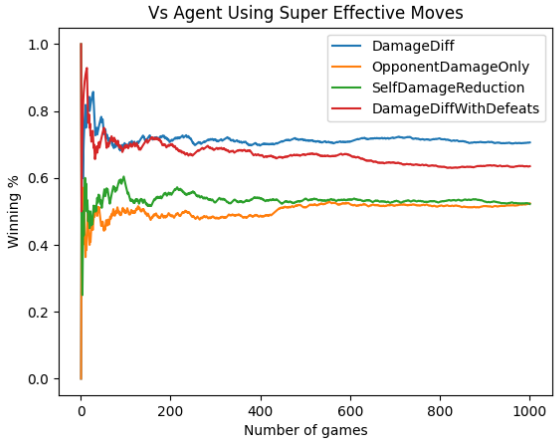
Performance versus Random Agent



Performance versus Epsilon Greedy Agent

Evaluation Function

I compared the win ratio of 4 different evaluation functions for Expectimax Agent vs Epsilon Greedy opponent. The SelfDamageReduction function considers only the remaining health of the player. OpponentDamageOnly considers only the total damage done to the opponent. DamageDiffWithDefeats also considers the number of pokemon defeated of the opponent. Out of these, the best win ratio was given by DamageDiff function. It considers both, the damage given to the opponent and damage taken by the pokemon thus taking into consideration the switching action as well.



Performance of Expectimax Agent using different Evaluation functions.

Conclusion

From the four agents, the Approximate Q-learning agent had the highest win ratio when counting all the games. Since a lot of factors affect win conditions including the Pokémon assigned, the win ratio of the Qlearning Agent, Minimax Agent and Expectimax Agent were all above average than of an average player. The Expectimax Agent(and similarly Minimax Agent) performed best with DamageDiff Evaluation function. Since this project considered the Battle with conditions applied to it, it can be further expanded to make it closer to the actual Pokémon Battle simulation and similar analysis can be made to compare the performance of four agents.

References

Norvig, Peter, and Stuart Russell. *Artificial Intelligence: A Modern Approach, Global Edition*. 4th ed., Pearson, 2021.

Sutton, Richard, and Andrew Barto. Reinforcement Learning: An Introduction. Second edition, Cambridge, MA, MIT Press, 2018.

"Complete Competitive Pokémon Dataset." *Kaggle*, 24 June 2018, www.kaggle.com/datasets/n2cholas/competitive-pokemon-dataset.

Kush Khosla, Lucas Li, Calvin Qi. Artificial Intelligence for Pokémon ´ Showdown. Stanford, CA: n.d. Web. 6 Dec. 2018.