Radboud University

Nash equilibrium estimation in competitive Pokémon using search and supervised learning

Thesis BSc
Artificial Intelligence

Author:
R. van der Heijden
s4822641

First supervisor:
Dr. J.H.P. Kwisthout

Second supervisor:
E. De Wolff, MSc

Abstract

The Pokémon main series role-playing video games revolve around catching and training different Pokémon to battle with. The competitive scene focusses solely on the game's battling aspect in a player vs player setting, transforming it into a zero-sum, non-deterministic, simultaneous-move strategy game with imperfect information. We propose a method to estimate Nash equilibria strategies for competitive Pokémon battles. By combining search with an evaluation network, we can set up a payoff matrix for any given turn within a Pokémon battle to be used for Nash equilibrium calculation. Our evaluation network trained on a gen-8-ou dataset was able to correctly predict the outcome of a battle for randomly sampled states with an overall accuracy of 0.740. In battles against an open-source heuristic expectiminimax agent by Patrick Mariglia, our agent using the same heuristic evaluation achieved average win rates of 0.173 (control) with regular competitive teams and 0.618 when using simplified teams. Our agent using the trained evaluation network achieved an average win rate of 0.295 with the regular competitive teams and did not battle within the simplified setting. The results indicate that an increase in evaluation accuracy leads to better Nash equilibria estimation, with our current evaluation network being the bottleneck of this method. Future experiments are required to determine whether a sufficient level of evaluation accuracy for our method can be achieved.

Contents

1. Introduction

Nearly everyone is familiar with Pokémon as a multimedia-franchise. Their core-series games are sold in countries all over the world. However, despite the game's popularity, few people are aware a competitive battling scene of Pokémon exists. While the official competitive circuit known as the video game championships (VGC) uses the mainline games, the heart of the scene is located on Pokémon Showdown [3] (simply known as 'Showdown'). This is an open-source online battle simulator that allows players to skip the exploration, catching and training aspects of the game and go straight into player vs player battling. The platform is equipped with an Elo rating system (preliminaries 2.1, Player rating) and the ability to save a game's replay, as well as some quality-of-life features for the battles themselves (for example, you can read what has happened during the previous turns). When considering all of this in combination with the download-free ease of access, it is clear why most of the competitive player base is set on Showdown.

1.1 The game

A standard Pokémon battle consists of two players, each with six Pokémon. It is a turn-based game where both players choose their move simultaneously at the beginning of every turn. The game allows for random events (e.g., a critical hit) to occur depending on the interaction between certain moves, abilities and items, making it a non-deterministic game. Each player chooses one active Pokémon at the start of a battle (preliminaries 2.1, Active Pokémon). During the move-selection phase of the battle, a player will also have the option to make a switch (changing their active Pokémon) and is forced to do so when their active Pokémon is knocked out. The game ends when one side has their entire team knocked out, or when a player decides to forfeit.

The six Pokémon a player brings to battle, also known as their team (preliminaries 2.1, Pokémon team), are chosen long before a battle is initiated (without knowing anything about the opponent) in a process known as teambuilding. A player gets to select their Pokémon, moves and other attributes in a way that suits the strategy they desire (preliminaries 2.1, Pokémon attributes). The result is a game where every battle is unique, although metagame trends will push the teams towards certain directions. It is important to note that players are not aware of the opponent's teambuilding choices during battle. All you see is which Pokémon it is you are facing, with some of the important attributes (e.g., moves) being revealed as the match progresses.

The thought process behind finding the *right* move in a competitive Pokémon battle is complicated. The players will have to combine the typical rock-paper-scissors-like 'do they know that I know that they know' type of logic together with long-term strategy. For every turn, you must mentally cross-reference your own actions with each potential move your opponent could use, somehow finding an acceptable balance between winning the long-term game and not losing immediately. Players often refer to a move being 'correct' when an opponent cannot reasonably afford to select their counterplay; for them, the risk of calling the turn wrong is too great in comparison to playing it safe and just accepting that a minor setback is probable. Though they may have the option to change their strategy, the fear of being read (like having your bluff called in a game of Poker) outweighs their potential reward, despite their conviction on how the turn would play out. Now imagine a scenario where you select (what many would believe to be) the correct move while your opponent decides to take a risk, do the unexpected and go for their best possible counterplay. As result, you are left with a reduced chance of winning the game. Does this mean your move was, in fact, not correct? Some simply view this as

an acceptable risk associated with their move choice or blame the opponent for not behaving in a 'rational manner'.

For simultaneous move games, this idea that a certain action is the correct choice based on assumptions of your opponent's move choice, despite the existence of opposing counterplay, is not in line with what game theory proposes. Instead, it suggests a Nash equilibrium (preliminaries 2.2) to be the optimal solution [1] [2]. This does make some intuitive sense when going back to our Pokémon battle setting: If player A has a 'correct' move available they should always go for it. Then if counterplay exists for player B, assuming they are aware which of player A's moves would be their 'correct' option, that counterplay should always be selected. This in turn should change the correctness of player A's move, initiating a recursive process until both players converge to a somewhat balanced strategy where each move is selected with a certain probability. Surprisingly, the idea of Nash equilibria strategy does not see much talk within the game's competitive communities. The decision-making process is viewed as a mix between strategy and predicting your opponent [12] despite to game theory being applicable to Pokémon [11]. The difficulty of Pokémon Nash equilibrium strategies in practice lies in being able to find one. With each game being different, the enormous search space and with many variables hidden to the players, it is simply infeasible to precisely compute a Nash equilibrium during a game of Pokémon.

1.2 Pokémon and artificial intelligence

Pokémon combines a lot of aspects challenging for artificial intelligence (AI) to solve. A standard Pokémon battle is a zero-sum, non-deterministic, imperfect information and simultaneous-move game. However, it might be the countless options of team customization that causes the most difficulties. This is the source of the game's imperfect information creating an enormous theoretical search space, while also leading to each game being unique.

With Pokémon being a turn-based game, it is not unreasonable to consider a search-based approach. The most prominent issues such approach would face is a limited search depth and difficulties in backpropagating a state's value, as the game contains simultaneous move selection. Min-max search methods such as Monte Carlo tree search [27] will not result in an optimal solution for simultaneous move games. These methods will assume that the opponent will always select their best counterplay. In a Pokémon setting this equates to your opponent reading your every move, which is not a realistic scenario. A min-max agent would only select their safest action (the one with the least amount of drawback in a worst-case scenario) regardless of how unlikely it may be for the opponent to punish the riskier move.

A more sophisticated approach to solving simultaneous move games is Counterfactual regret minimization (CFR), which is an iterative game-tree traversing algorithm that approximates Nash equilibrium strategies [26]. CFR can be used to solve large incomplete-information games, but there is a catch. To deal with extremely large games, the algorithm runs on an abstraction of the game. Even though we have not found any research to back this up, such abstractions just do not seem reasonable for competitive Pokémon. Each game really is different. The dynamics of a battle can be drastically different by changing just one of the Pokémon's moves. A single abstracted version of Pokémon would be like playing Poker with every card in the deck being the same; it would not translate to the real thing. Hence, there is a need for a new (approximation) method if Nash equilibrium strategies are to be applied to competitive Pokémon.

1.3 Rationale behind out method

Game theory states that for two-player finite zero-sum games, a Nash equilibrium is the optimal solution [1]. Such equilibrium will always exist given that mixed strategies are an option [2]. The idea of using a Nash equilibrium strategy is simple: in terms of expected payoff, the opponent can break even at best and once they make inaccuracies, you profit. Nash equilibria can be computed via a payoff matrix, which in essence is just a table with awarded scores for each combination of moves the players can choose from. Our approach is straightforward: we explore a search-tree for each combination of moves and evaluate the resulting game states. To avoid searching until an end-state has been reached, we use a trained neural network to assign a score to each future game state. The evaluation network is trained to predict the winner of a given state by assigning each a probability. Our assumption is that the strength of a player's position in the game directly correlates with their probability of winning, thus making it a valid metric to be used as value for a given state.

Our method was inspired by state-of-the-art Go and Chess AI [21] [22] [23] that combine machine learning and search as way to accurately evaluate positions in-game. Rather than going with a move-prediction model [17] [24], we pursue a Nash equilibrium approximation method in the hopes of achieving a similar level of un-exploitability as seen in state-of-the-art Poker AI [13] [25].

1.4 Research question

This research aims to investigate the viability of Nash equilibrium strategies for competitive Pokémon when the payoff values are estimated using search together with an evaluation neural network. More formally, our research question is:

Can the combination of search and supervised learning achieve accurate Nash equilibria estimates for competitive Pokémon battles?

We will attempt to answer our research question by answering four different sub-questions:

1. *Can competitive Pokémon battle outcomes be predicted with a neural network trained via supervised learning?*

2. *Does the performance of our Nash equilibrium estimation method improve when the evaluation accuracy increases?*

3. *Is our Nash equilibrium estimation method sufficient to outperform Mariglia's Safest[1]?*

1.4.1 Contributions

The amount of artificial intelligence research for Pokémon is extremely limited, especially when comparing to other game playing AI research such as for Chess and Poker. Therefore, we believe that for this subject, any well-documented machine learning approach will be a significant contribution. We introduce a new and easy to explore method for Pokémon-playing AI and draw more attention to Nash equilibria strategies

[1] Safest is the benchmark AI used in one of our experiments. Its details are covered in section 5.2.

as opposed to move-prediction based models. This research will contribute to the field of machine learning in Pokémon, providing a well-documented supervised learning game state evaluation approach for others to build upon.

2. Preliminaries

This chapter will define some concepts that frequently appear in this paper, as well as describe the core terms and mechanics of a competitive Pokémon battle. The main purpose of this chapter is to cover the context that is required for the reader to get a deeper understanding of the topic at hand. It may also serve as a helpful reference for readers already familiar with the material.

2.1 Pokémon terminology

Player rating
Pokémon Showdown makes use of a ladder rating system [14]. When we refer to a player's rating, we refer to their Elo rating. A player's rating starts at one thousand and typically, for most battle formats, players do not achieve a rating above two thousand.

Pokémon team
A player's team, also known as their *party*, describes all the Pokémon a player has brought to battle (Fig. 1).

Active Pokémon
With the standard single-battle format, each player has one active Pokémon (Fig. 1). Only an active Pokémon can select an attack and only an opposing active Pokémon may be targeted. The only two situations where a player does not have an active Pokémon are at the start of a battle (during team preview) or when a player's Pokémon has just been knocked out.

Reserve Pokémon
Each non-active Pokémon on a player's team is considered a reserve Pokémon.

Switch
A switch refers to a player choosing the switch action in battle, swapping their current active Pokémon with one of their reserve Pokémon.

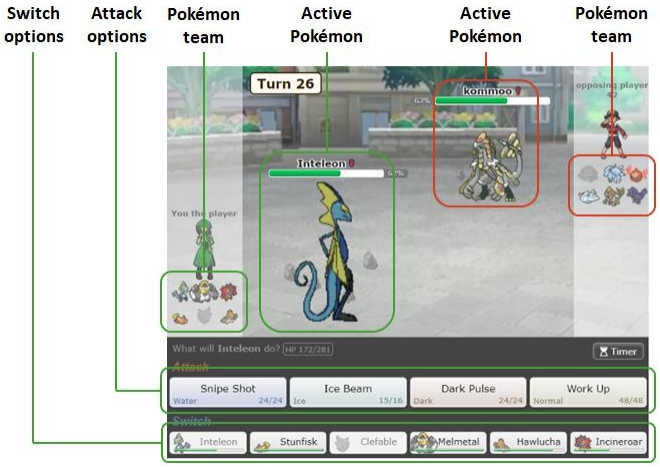

Figure 1: The Pokémon Showdown game interface during battle. The main attributes for the user (green) and the opponent (red) are highlighted.

Team preview
The term team preview is used to describe the active Pokémon-selection phase at the start of a battle: both players get to see which Pokémon the other player has got and select the active Pokémon they want to start the battle with.

Pokémon attributes
Before getting into a battle, each player constructs their team by selecting up to six different Pokémon, known as teambuilding. Depending on the ruleset you play with, there are roughly nine hundred different Pokémon to choose from (excluding form changes). Every Pokémon is made up out of multiple attributes such as their typing, base stats, abilities and moves (Fig. 2). Some of these attributes can be used for customization, with the five most impactful options being:

1. *Level*
   A Pokémon's level ranges from 1-100. Generally, there is no reason not to select the highest level available within a ruleset.

2. *Moves*
   A Pokémon can have up to four moves, selected from their individual move learnset.

3. *Ability*
   A Pokémon must have one ability, selected from their individual ability learnset.

4. *Item*
   A Pokémon can be given any one item.

5. *Stats*
   Small adjustments to a Pokémon's natural stats can be made by changing the Nature, EV's (effort values) and IV's (individual values).

Figure 2: The Pokémon Showdown Teambuilder interface for a single Pokémon. The most significant customizable attributes are highlighted.

Switching moves
In the game there exists a unique set of moves we refer to as switching moves. These are the moves that once used, the player is given the opportunity to make a switch mid-turn (e.g., U-turn).

Damage rolls
Part of Pokémon's non-determinism comes from the damage formula. For all damaging moves in-game, sixteen integers between 100% and 85% of the attack's maximum damage are generated, known as the damage rolls. The eventual damage done is a single randomly sampled damage roll.

2.2 Nash equilibrium

A Nash equilibrium is a set of strategies for each player where neither player can increase their expected payoff by deviating from their own strategy. At least one Nash equilibrium will always exist in a two-player finite zero-sum game so long as mixed strategies are available [2]. A Nash equilibrium is mixed if a player's moves are selected with a certain probability.

	A	**B**
A	*-5*, *-5*	*0*, *-10*
B	*-10*, *0*	*-1*, *-1*

Figure 3: A two-player payoff matrix containing a pure strategy Nash equilibrium. Both players (red, blue) selecting **A** is a Nash equilibrium.

	A	**B**
A	*-5*, *5*	*1*, *-1*
B	*1*, *-1*	*-1*, *1*

Figure 4: A two-player payoff matrix containing a mixed strategy Nash equilibrium. Both players (red, blue) selecting **A** with *P=0.25* and **B** with *P=0.75* is a Nash equilibrium.

3. Related work

3.1 Related game playing AI research

Recent years have seen big breakthroughs in game playing artificial intelligence with the likes of AlphaGo [21], OpenAI [30] and AlphaStar [31] making headlines. With Go, Chess and Pokémon all being a turn-based game, both AlphaGo and its successor AlphaZero [22] carry a lot of relevance as similar techniques may be explored for competitive Pokémon. AlphaGo combines Monte Carlo tree search (MCTS) with a supervised learning (SL) / reinforcement learning (RL) hybrid policy network and an RL value network. AlphaZero uses MCTS together with a single dual-headed RL policy and value network. An important takeaway is the fact that both programs combine search with a way of evaluating a player's in-game position and employ deep convolutional network architectures to do so. Convolutional layers are a logical choice for games like Go and Chess, as the boards can be represented as images with each tile representing a single pixel [21] [32]. For Pokémon, however, this might not be the case and different network architecture will have to be explored.

A (perhaps somewhat limited) comparison between Poker and Pokémon can be drawn as well. The games share a 'predict your opponent' type of aspect, and Poker AI deals with a similar challenge as Pokémon AI: incomplete information. State-of-the-art two-player Poker AI use Counterfactual regret minimization (CFR) variants on an abstraction of the game [16] [25]. These algorithms can approximate Nash equilibrium strategies for large incomplete-information games with great accuracies [26]. The result is a kind of strategy-lookup table where the Nash equilibrium strategy probabilities are stored for different scenarios in-game. While Pokémon's game-changing nature might make it unrealistic for these types of algorithms to be the solution, future Nash equilibria approximation methods may still hold potential.

3.2 Pokémon AI research

With the competitive scene being relatively small, especially when compared to the popularity of the mainline games, the amount of related artificial intelligence research is limited. While some machine learning attempts for Pokémon do exist, few are well-documented. Amongst those that are, *A Self-Play Policy Optimization Approach to Battling Pokémon* [17] by Huang and Lee is the most notable. The authors created, as the title may already suggest, a self-play-based reinforcement learning agent and is what we consider to be *the* state of the art for competitive Pokémon artificial intelligence. Their model of choice is a two-headed actor-critic neural network, combining both policy-based and value-based RL methods. An actor-critic neural network functions by predicting both a policy and value for a given state, then uses the value as estimated expected return when updating the policy prediction [18]. The performance of their model was measured against *Safest* by Mariglia [15] and was both trained and evaluated in two distinct settings: a random battles format (where each player's team is generated pseudo-randomly) and a format where the agent would rotate around three different teams. The actor-critic model significantly outperformed Mariglia's agent in the rotating-team format, presumably, because a model trained on a specific team will have learned the ins and outs of that playstyle. The random-battle trained model would only slightly outperform Mariglia's search-based agent.

The other well-documented machine learning attempt is *Gotta Train 'Em All: Learning to Play Pokémon Showdown with Reinforcement Learning* [24] by Chen and Lin which happens to be a policy optimization RL approach as well. The authors feature their Pokémon embedding pipeline in detail where they use graph data in combination with Node2Vec [28]. In contrast to the previous paper, Chen and Lin's agent did not achieve much success. It remained unable to outperform a min-max 1-step lookahead agent with a heuristic evaluation that considered the Pokémon's health only.

The last notable project goes by the name of FutureSightAI [29], created by Albert III. Albert's method combines search with both an evaluation network and a move-prediction model. The documentation, however, is limited to a Youtube video plus a website containing what is essentially the video's script. It does not cover any detail on the machine learning architectures and neither does it state exactly how the claimed accuracies were measured, making the project achievements difficult to verify.

4. Methods

To investigate the viability of our move-selection method a Pokémon Showdown agent has been created. Pokémon Showdown is an open-source battle simulator [3] and is the go-to platform to be used within the competitive scene. Unfortunately, due to time constraints, our method could not be extended to the standard imperfect information setting of a Pokémon battle. The original imperfect information method is briefly described in section 6.3. The method used for our experiments and described in the current section will focus on battles with complete information only.

Our method can be viewed as three separate modules: Search, Evaluation network and Nash equilibrium calculation. The search algorithm combines with the evaluation network to generate a payoff matrix. The Nash equilibrium calculation occurs at the final stage of this move-selection method.

1. *Search*
   A payoff matrix is created by calculating the expected value for every combination of actions available to the players.

2. *Evaluation network*
   An artificial neural network that receives a game state as input and yields a player's probability of winning as output.

3. *Nash equilibrium calculation*
   Calculates a single Nash equilibrium for a given payoff matrix.

These three modules will be expanded in the following sections.

4.1 Search

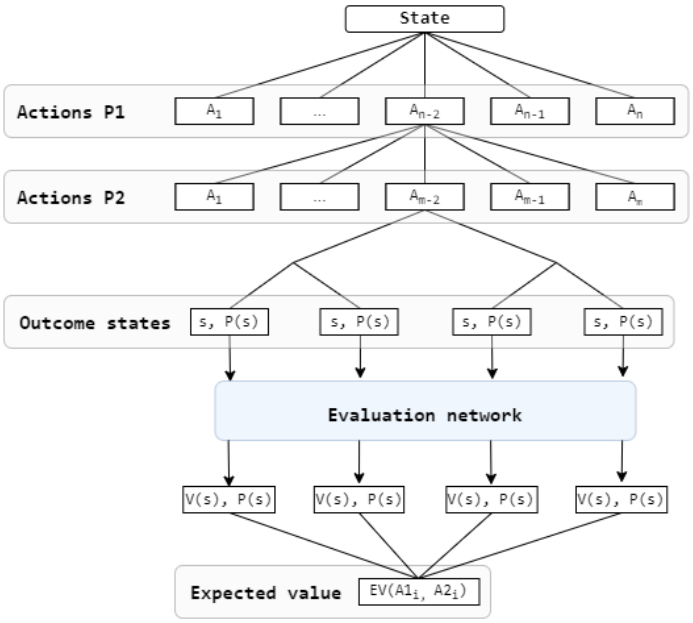

Figure 5: Our search algorithm. A payoff matrix is obtained by calculating the expected value for every action pair combination.

The search part of the method was implemented using Mariglia's Pokémon Showdown project [5]. This is an open-source Python-Showdown environment containing a state-simulation implementation. Although the project has a few limitations, including a couple of in-game features not being implemented and some edge-case scenarios causing wrong results, none of these limitations lead to anything game breaking.

The goal of our search algorithm is to set up a payoff matrix for given state. The main algorithm steps can be seen in Fig. 5. Starting at the current game state, each of the player's actions are cross-referenced. For each action combination (A_{p1}, A_{p2}) the possible turn outcomes are simulated with each outcome state s having a probability $P(s)$ and a value $V(s)$. $P(s)$ is the probability for that set of actions to result in state s. $V(s)$ is generated via the evaluation network and represents the agent's chance of winning. The expected value for (A_{p1}, A_{p2}) with n outcome states is calculated via $\sum_{i=1}^{n} P(s_i) \cdot V(s_i)$. This expected value is used as the payoff for the corresponding action combination (A_{p1}, A_{p2}) within our matrix.

One design choice worth mentioning is our representation of switching-moves. In both the search and payoff matrix, each switching-move is split as multiple different moves; one for each switch available. This is a way to circumvent the state-simulation environment being ill-equipped to deal with this type of moves, as well as to better represent the move for Nash equilibria calculation.

4.2 Evaluation network

4.2.1 Dataset

The dataset we used in this project was made available by the Pokémon Showdown staff. It consists of around 1.5 million .json files for the gen-8-ou format from Dec. 2019 till Feb. 2020, containing the player inputs, a game's event log and both Pokémon teams used by the players. After filtering the faulty game-logs and games for which Dynamax was still within the ruleset[2], we were left with just shy of 1.1 million rated battles. We created a train-test split before any states were sampled to prevent games from co-occurring in both the training and test datasets. Due to the large size of our dataset a 90/10 split was considered to be an adequate choice.

A. State extraction
Before any game states could be extracted from these files, the event log first had to be parsed and the events had to be tracked properly. This was done by creating an adapted version of the *battle* and *battle_modifier* classes that are present in Mariglia's Showdown environment. These files were edited in such a way that the program would track the game's events for both players via the event log, allowing accurate game state representations to be extracted for given turns.

The original AlphaGo paper [21] mentioned an increase in overfitting when multiple states are sampled from a single game, as successive positions are strongly correlated while sharing the same regression target. The proposed solution is to simply extract one state per game only. However, this would significantly reduce the size of our dataset, negatively impacting the performance of the network. In the hopes of having our cake and eating it too, we proposed these two middle-ground solutions:

[2] Dynamax is a game mechanic introduced in Pokémon generation 8. It was removed from the gen-8-ou ruleset on Dec. 17, 2019.

1. Extract *n* states per game where *n* is a function of the game's length (with a maximum of *m*).

2. Extract *n* states per game where *n* is fixed, and every state is extracted for games consisting of less than *n* states.

It seemed that having *n* as a function of the game's length would lead to an increased rate of overfitting when compared to the other sampling method. We hypothesize that an over-representation of the longer games is the cause. While the states sampled from a longer game are further spread out from each other than those coming from shorter games, the between-state correlation may primarily come from the Pokémon brought to a battle. Where in Go or Chess, a state from the beginning might not share as much similarity with one from the endgame. In Pokémon, these states would still share the same species, moves, abilities, items, and stats for both players. Thus, the second sampling method was chosen with *n=10*. The number is an arbitrary choice but felt appropriate given that the average number of states per game in our dataset is twenty-five.

Three other decisions are worth mentioning. First, team preview states were not included in the sampling process as they are not required for our agent to function. Second, the situations in-game where a player must select a switch-in after one of their Pokémon has been knocked out were also represented as separate states, despite not being at the start of a turn. Lastly, games made of less than three states have been excluded from the sampling process. These games are forfeit before any move is made past team preview.

Our final dataset contains a total of 8.711.156 states sampled out of the 942.004 games that remained. Roughly 8.000.000 states were used as train set with the remainder used for validation. It is important to note that the state files were not shuffled prior to creating the split. This means that our train and validation sets do not share any samples from identical games.

B. State transformation

The core idea for the evaluation network is that a player's win probability is an accurate measurement for how strong their in-game position is. With this assumption in mind, we used the game result a given state was sampled from as our output label. Although ties rarely occur in this game it was still implemented as one of the possible labels, giving a final output set of {0, 0.5, 1}.

Pokémon	Cosine similarity
Raichu	0.803
Raichu-Alola	0.554
Togedemaru	0.523
Seaking	0.486
Liepard	0.401

Table 1: Pikachu's five nearest neighbours after creating 64-dim entity embeddings for all Pokémon species.

N-dimensional entity embeddings have been created for each of the four major categorical Pokémon variables (species, ability, item and moves). These embeddings were created by first converting each Pokémon from our dataset to a string of their attributes. For a Pokémon to be extracted as datapoint, a minimum player rating of twelve hundred was set to portrait a more accurate image of the metagame at hand. After representing the species, moves, ability, item and typing of each Pokémon as sentence, a FastText [6] continuous-bag-of-words model was trained. This model can be used to predict a target (e.g., species) based on the context (e.g., moves and item), but also creates

an n-dimensional vector representation for each variable within a category. We used these vectors as embeddings for the categories mentioned above (table 2).

Feature	Type	Dims	Transformation
species	categorical	[1 x 1023]	64-dim embedding
ability	categorical	[1 x 263]	16-dim embedding
item	categorical	[1 x 197]	16-dim embedding
moves	categorical	[4 x 757]	16-dim embedding
typing	categorical	[1 x 18]	one-hot encoding
status	categorical	[1 x 7]	one-hot encoding
volatile status	categorical	[1 x 33]	one-hot encoding
is active	indicator	[1]	none
is alive	indicator	[1]	none
disabled moves	indicator	[4]	none
move pp	continuous	[4]	scaling
max move pp	continuous	[4]	scaling
n moves	continuous	[1]	scaling
level	continuous	[1]	scaling
health	continuous	[1]	scaling
stats	continuous	[6]	scaling
stat changes	continuous	[6]	scaling

Table 2: Complete list of features describing a single Pokémon in battle. These features are fed into the 'Pokémon layer' of the network (see section 4.2.2).

Each of the minor categorical variables, such as typing or status conditions, have been one-hot encoded. All numerical variables were scaled down to a range between 0 and 1. While a Pokémon's stats are theoretically limitless, ignoring rare cases it will normally not exceed four hundred; hence, we divided each stat by four hundred to obtain the desired range.

It is possible for battles to consist of players with less than six party members or for Pokémon to have one, two or three moves only (e.g., Ditto with Transform only). For these edge cases, 'empty' Pokémon and move slots containing all zeros were generated. Because of this, the attribute of a Pokémon which states whether it has fainted or not was changed to *is_alive* {1, 0}, so that an empty Pokémon slot of all zeros would correlate with a Pokémon being fainted, rather than the other way around.

C. Data augmentation

Three basic ways of state augmentation were applied to reduce the rate of overfitting. For each transformation:

1. The player point of view would be swapped with a probability 0.5.
2. Each player's reserve Pokémon positions are randomly shuffled.
3. all Pokémon move positions are randomly shuffled.

4.2.2 Network architecture

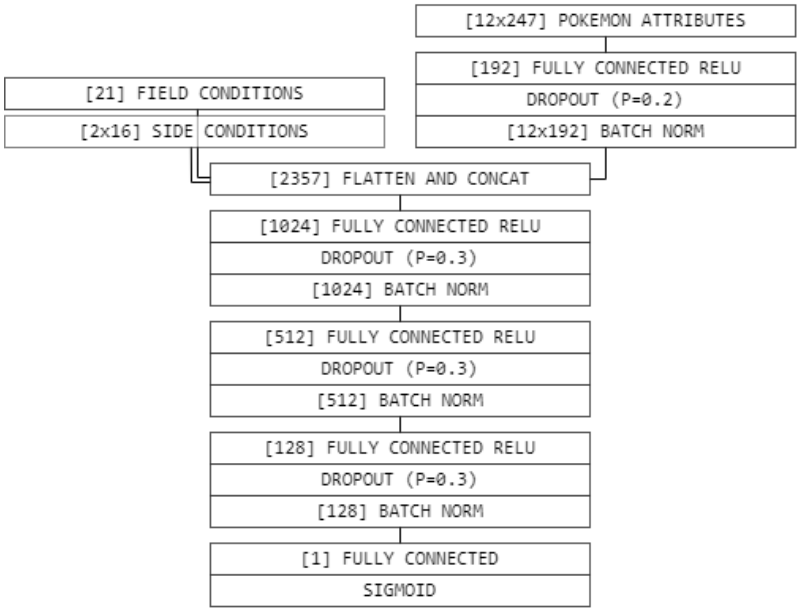

Figure 6: Evaluation network architecture. The Pokémon attributes, field and side conditions are extracted from a game state and together make up the input layer. The output is a single probability of winning the game.

The chosen model is a feedforward artificial neural network. Inspired by Huang and Lee [17], the network is made of a hierarchical structure where each Pokémon and its corresponding features first go through separate 'Pokémon layers'. Attempts were made to recreate their 'team pooling' layer as well, however, it did not lead to any increase in performance for our model. Multiple different convolutional 'Pokémon match-up' layers, like the convolutional architecture used in *Helping AI to Play Hearthstone using Neural Networks* [7] by Lukasz Grad, have been tried as well. Again, without any performance increase.

The input features are split into three categories: field conditions, each player's side conditions and each Pokémon's attributes. All twelve sets of Pokémon attributes are individually passed through a 'Pokémon layer' (Fig. 6). It is important to note that the same weights are used for each of the twelve Pokémon in battle. This layer serves as a type of Pokémon embedding and helps to reduce the amount of network parameters, given that the alternative is a large fully connected layer for all our input features. Next, all Pokémon layer outputs are concatenated with the field and side conditions and passed through multiple series of fully connected, rectified linear unit (ReLU), dropout and batch normalization layers in that order (Fig. 6). The output layer consists of a single fully connected layer with an out size of one. A Sigmoid activation function was used so that the output would represent a probability (of winning the game).

Our final network was created via PyTorch [19] and contains a total of 3.141.225 network parameters. It was trained for thirty-seven epochs using the Adam algorithm [20] and a batch size of 256. The learning rate was set to 0.0002 and a learning rate decay was used with $\gamma=0.95$ and a step size of 1. With the output of the network being what is essentially a binary classification between a win or loss (ties are extremely rare and can be ignored) a binary cross-entropy (BCE) loss is appropriate.

4.3 Nash equilibria calculation

Given the payoff matrix (created via the Search algorithm, section 4.1) a single Nash equilibrium is calculated using the QuantEcon Game theory package [8]. The function of choice was the method using the Lemke-Howson algorithm [9]. Because both the Lemke-Howson and the McLennan-Tourky [10] implementations gave comparable results when tested, we arbitrarily settled on the Lemke-Howson method. No extra criteria were used for matrices containing multiple Nash equilibria. The equilibrium of choice is the default output of the function.

4.3.1 Mid-turn switches

A Nash equilibrium is calculated at the start of a turn. During the turn, whenever the faster player uses a switching-move they must choose and act mid-turn: They have to switch into one of their reserve Pokémon. If the opposing Pokémon moves last and does not switch out, the player is presented with this as evidence: The player knows the opponent has not switched. This may result in scenarios where our agent must choose an action based on this new evidence and a previously computed Nash equilibrium. To do so, an assumption is made that the opponent's (mixed) Nash equilibrium strategy computed at the start of the turn translates to the current mid-turn position in-game[3]. This assumption leaves us with three scenarios:

1. *The opponent's strategy did not contain a switch*.
   The agent selects their best counterplay to the opposing strategy.

2. *The opponent's strategy contained switches only.*
   A new equilibrium is computed, and the agent selects their move based on the newly computed strategy.

3. *The opponent's strategy contained at least one switch and at least one attack*.
   The remainder of the opponent's strategy is normalized. The agent selects their best counterplay to the normalized opposing strategy.

[3] We reckon this may not be correct, but we could not find any relevant material for scenarios like this. This was a simple solution that allowed us to continue.

5. Experiments

5.1 Experiment 1: Predicting battle outcomes

This experiment attempts to measure the performance of the evaluation network. Our network should be able to accurately predict the winner of a battle at a given state. The predicted label is obtained by rounding the network output to the nearest integer {0: loss, 1: win}. Like the sampling process explained in section 4.2.1, games consisting of less than three game states are excluded from this experiment.

5.1.1 Experimental setup

To better understand the nature of Pokémon battle outcome prediction we measured the prediction accuracy for two categories: game rating[4] and game length. Each category is grouped into several ranges (*1000-1199*, *1200-1399*, *1400-1599*, *1600+* and *All ratings* for game rating, *2-9*, *10-19*, *20-29*, *30-39*, *40-49*, *50+* and *All lengths for* game length). For either category, the range as well as the number of ranges are arbitrary choices. For both (rating and length) we consider two measurements: overall prediction accuracy and prediction accuracy as function of the game's completion. It should be noted that the higher ranges (*1600+* for game rating, *40-49* and *50+* for game length) have a low representation within the dataset (Fig. 7, 8). Even though a set of *n* 'test states' is sampled from each game, the network may produce similar evaluations within each set due to correlation between those states. Hence, the results that correspond to these higher range ranges should be taken with a grain of salt.

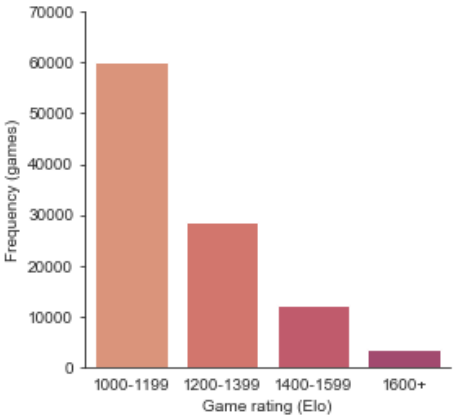

Figure 7: Frequency distribution of the game rating ranges within our test data.

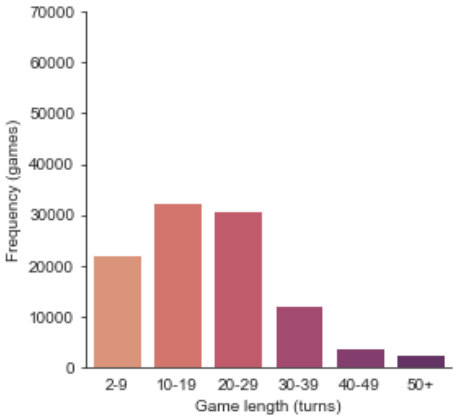

Figure 8: Frequency distribution of the game length ranges within our test data.

A. Overall prediction accuracy
The prediction accuracy was measured for each range within the category. Game states were sampled via the same method as explained in section 4.2.1: For each game we extracted ten states, and every state was extracted for games consisting of less than ten states. We used state sampling (as opposed to evaluating entire games) as way to prevent results from being skewed towards longer games.

[4] A game's rating is set to the lowest Elo rating of the two players in battle.

B. Prediction accuracy as function of the game's completion
To calculate the predicted results at different percentages of game completion, we did the following: For each game, every state was evaluated and converted to their corresponding label prediction. We divided the game into twenty segments {*2.5*, *7.5*, ..., *97.5*} representing the game's completion percentage, and mapped each prediction to the nearest segment based on the game completion at that state.

5.1.2 Results

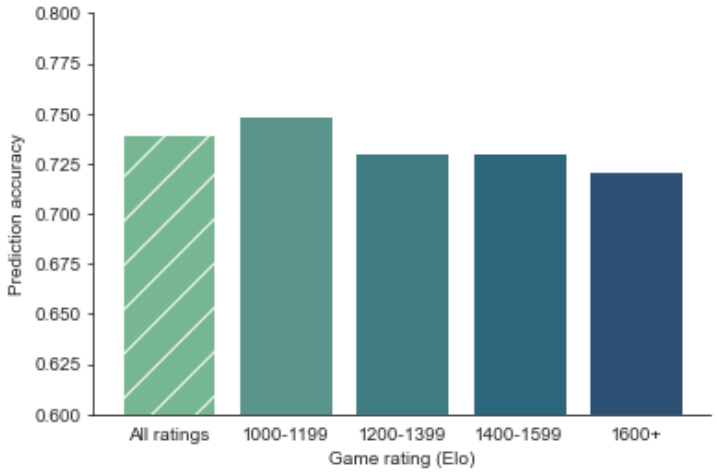

Figure 9: The overall prediction accuracy for different game rating ranges.

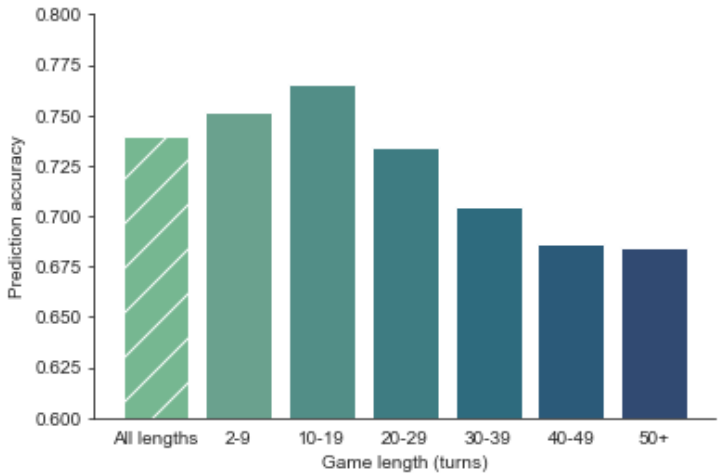

Figure 10: The overall prediction accuracy for different game length ranges.

The network achieved an overall prediction accuracy of 0.740 when not distinguishing between a game's rating or length (Fig. 9 *All ratings*, Fig. 10 *All lengths*). Prediction accuracies of 0.749, 0.730, 0.730 and 0.721 were measured for rating ranges of *1000-1199*, *1200-1399*, *1400-1599* and *1600+*, respectively (Fig. 9). Prediction accuracies of 0.752, 0.765, 0.734, 0.705, 0.686 and 0.684 were measured for length ranges of *2-9*, *10-19*, *20-29*, *30-39*, *40-49* and *50+*, respectively (Fig. 10).

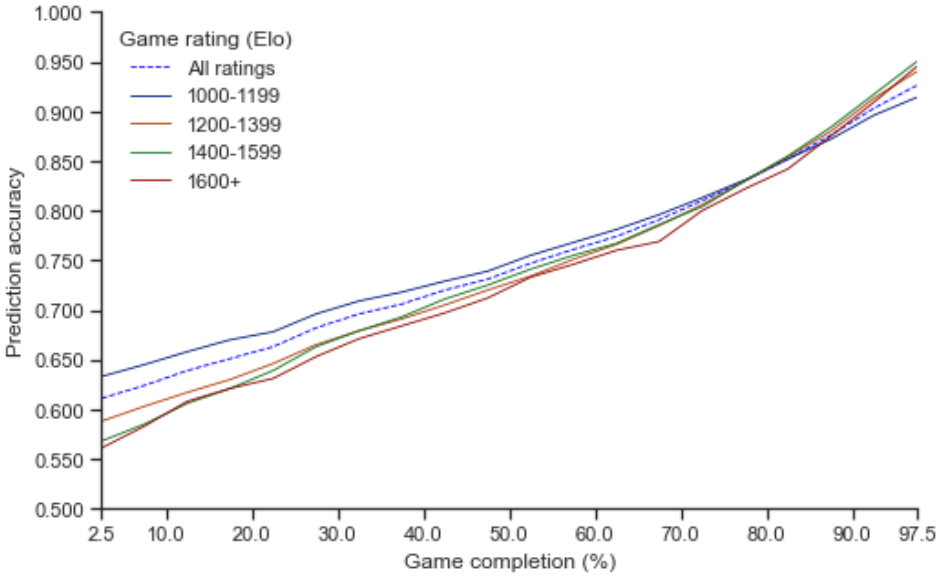

Figure 11: The prediction accuracy as a function of the game's completion for different game rating ranges.

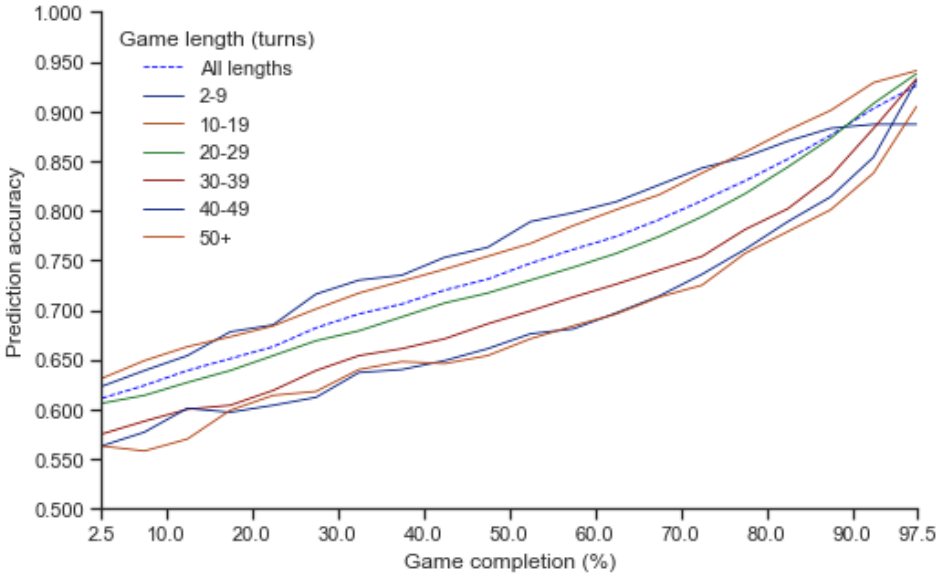

Figure 12: The prediction accuracy as a function of the game's completion for different game length ranges.

5.2 Experiment 2: Battling a benchmark AI

This experiment consists of battles between our agent and a chosen 'benchmark AI'. The benchmark AI of choice is *Safest* by Mariglia [15]. This is an open-source Showdown agent that uses expectiminimax for its move selection. As mentioned at the start of section 4, this experiment contains a complete-information setting: every agent has detailed knowledge of the other player's team.

5.2.1 Experimental setup

For our experiment we created two Nash equilibrium estimation agents:

1. *Network-agent*: uses the trained evaluation network.
2. *Heuristic-agent*: uses the same heuristic evaluation as used by *Safest*.

Both of our agents faced off against *Safest* in multiple series of two hundred battles. We created two distinct battle categories: battles with realistic (sample) teams and battles with simplified teams. During the experiment, *Safest* was set to only check the average damage roll, not to consider critical hits and had a maximum search depth of two. Our agents were set to only consider the minimum, average and maximum damage rolls, only consider the average critical hit roll and had a maximum search depth of one. With these settings, both *Network-agent* and *Safest* took roughly the same time for each decision. It may be worth mentioning that we considered creating a second agent for *Safest* as well (*Network-Safest*). However, due to the evaluation network time consumption we could not reasonably combine the two without lowering the search depth to one, ultimately defeating the purpose of this search-based agent.

A. Sample teams
Two sample teams (team A, team B) have been selected from an online archive [4]. The specifics of these two teams can be found in the appendix (A.2 Sample teams). For a sample team to be considered, all Pokémon must exist in our dataset from Dec. 2019 to Feb. 2020. Both agents (*Network-agent*, *Heuristic-agent*) faced off against *Safest* in each of the two team matchups (A vs B and B vs A). The mirror matchups (A vs A, B vs B) were left out as regular competitive Pokémon teams in a mirror setup showed some undesirable battle interactions, like (normally uncommon) speed ties or unprogressively drawn-out positions in-game.

B. Simplified teams
Two simplified teams (team A, team B) have been handcrafted with each having a difference in complexity. The specifics of these two teams can be found in the appendix (A.1 Simplified teams) but a quick overview is given down below. For this category of teams, the battles were held in a mirror-match setting (A vs A and B vs B). The goal of these simplified teams is game abstraction to a point where the heuristic function becomes an accurate method of evaluation. Since such settings do not occur in our dataset, we only used *Heuristic-agent* to face off against *Safest* during this part of the experiment.

> *Simplified team A*
> This team consists of three Pokémon and has the typical fire-water-grass type triangle, creating a rock-paper-scissors-like dynamic in battle. The Pokémon hold no items, carry just two moves, and have no EV investment. This setting creates a straightforward type-based positioning game and contains no major long-term strategy elements.

Simplified team B
This team consists of four Pokémon. The Pokémon hold items, carry three moves and each has their own style of EV investment: physically offensive, specially offensive, physically defensive and specially defensive. This setting contains no major long-term strategy elements.

5.2.2 Results

	Sample: A vs B	**Sample: B vs A**	**Simple: A vs A**	**Simple: B vs B**
Heuristic-agent	0.230	0.115	0.570	0.665
Network-agent	0.365	0.225	—	—

Table 3: The performance of *Heuristic-agent* and *Network-agent*. Each value represents the agent's win rate against *Safest* after two hundred battles.

Table 3 shows the win rates our agents achieved against *Safest* for each battle category. *Heuristic-agent* achieved an average win rate of 0.173 in the sample team battles and an average win rate of 0.618 in the simplified team battles. Network-agent achieved an average win rate of 0.295 in the sample team battles and did not play with the simplified team battle setup. As shown, *Network-agent* was able to outperform *Heuristic-agent* in terms of win rates with the sample teams, but neither agent was able to achieve a positive win rate against *Safest* in this setting. In the series with the simplified teams *Heuristic-agent* did manage to outperform *Safest*.

6. Discussion

6.1 Experiment 1 - Network prediction accuracy

Our trained evaluation network achieved an overall prediction accuracy of 0.740. The outcomes of higher rated games are shown to be more difficult for our model to predict (Fig. 9). Similarly, the model's prediction accuracy drops for longer lasting games, with the *2-9* range being the exception (Fig. 10). When considering the prediction accuracy as a function of a game's completion (Fig. 11, 12) we get a better idea as to why the accuracies vary for different rating/length ranges, with five observations we want to cover more in depth:

1. The prediction accuracy increases as the game progresses. This is to be expected, as both players will have roughly the same chance of winning at the start of a battle, whereas later in the game it should be clear which player has the upper hand.

2. The network already achieves prediction accuracies above chance level (0.500) within the first 5% of a game played. This might reaffirm the idea that a player's Pokémon selection plays a big part in their chance of winning, highlighting the importance of teambuilding. However, it remains unclear whether the network has learned the way two teams match up against each other. Rather, we believe it picks up on the fact that some Pokémon will simply have a higher/lower chance of winning in general, due their competitive strength or place within the metagame.

3. The prediction accuracy for the *2-9* game length range hits an early plateau at an accuracy of 0.887, which is the lowest accuracy for the last 5% of a game played when comparing to the other length ranges. We hypothesize that this results from multiple early (premature) forfeits ending up in this length range.

4. The network achieves higher prediction accuracies for the lower length ranges regardless of game completion (ignoring the *2-9* range). It could be possible that many longer games occur if neither player can get themselves in a dominant position at the start of a battle. You would expect it to be difficult to predict the winner of a game when both players maintain at equal positions for multiple turns.

5. The network achieves higher prediction accuracies for the lower rating ranges until at around 80% game completion, where the trend seems to reverse (apart from the *1600+* game rating range). The exact cause remains unclear, but we hypothesize the following:
    a. Lower rated players may have a higher tendency to use 'uncompetitive' Pokémon, making these games easier to predict early on. Simultaneously, these players may be more likely to prematurely forfeit a game (they might simply not care about the game or get frustrated when the game does not go their way), which leads to endgames being less predictable.
    b. Higher rated games may remain at an equal state for longer (neither player will make many mistakes), making these games difficult to predict early on. Simultaneously, these players carry the knowledge to properly play out a winning endgame position, making these games easier to predict at near the end.

    However, we must reiterate that this is all very speculative. We also do not have enough high rated games in our dataset to fully rule out noise being the culprit here. No conclusions should be drawn solely based on this observation.

6.1.1 Evaluation network limitations

It is hard to tell what the overall accuracy is representative of, as we do not have a good reference to what extend a Pokémon match can be predicted. We are certain that a near-perfect accuracy is unrealistic due to game's inherit randomness. While none of the well-documented Pokémon machine learning research focusses on battle prediction, the FutureSightAI project claims to be able to predict a Pokémon battle from any given turn with 'at most 81% accuracy', but as mentioned in the related work chapter (section 3.2), this project's achievements are hard to verify. Still, we do not believe that our model's accuracy comes close to what is achievable for competitive Pokémon battle outcome prediction.

There exists a variety of factors that are potentially holding back the performance of our model. For one, we have had limited computing power available when training the network. This means we have been unable to fully explore large hidden layer sizes (above four thousand nodes, for example). Another reason is the size of our dataset. We had noticed an increase in training samples coinciding with an increase in accuracy but did not have any capacity to collect more games as datapoints. Sampling multiple states from each individual game helped enlarge our train set but lead to extra overfitting in the process. In an ideal scenario your dataset is large enough for each state to be sampled from a separate game, as described in the original AlphaGo paper. Finally, related to the size of our dataset, is the lack of high rated training games. Considering the distribution of game ratings shown in Fig. 7, together with the fact that the ladder rating scales up to around two thousand, it is clear the bulk of our data comes from lower rated games. It is reasonable to assume a network trained on high rated games will have an increase in prediction accuracy when compared to our model, at least when measured against other higher rated games.

6.2 Experiment 2 - Performance against the benchmark AI

The results (Table 3) show that within the sample team battle setting, our *Heuristic-agent* and *Network-agent* achieved average win rates of 0.173 and 0.295, respectively. *Heuristic-agent* managed to get an average win rate of 0.618 against *Safest* in the simplified team setting. The main observation is the fact that *Heuristic-agent* outperformed *Safest* in the simplified setting for which we believe the evaluation method to be accurate, while unable to do so within the regular sample team setting. This indicates that our Nash equilibrium estimation method has the potential to perform well but that the bottleneck lies with the evaluation accuracy. It does, however, not tell us whether a required level of evaluation accuracy is achievable with our supervised learning approach.

Network-agent outperforming *Heuristic-agent* in terms of win rate from the sample team matches shows us that the evaluation network is indeed an upgrade from the heuristic evaluation, as well as reaffirm the idea that our method's performance is tied to the evaluation accuracy. *Network-agent's* inability to attain a positive win rate against *Safest* is in line with the believe that our current trained evaluation network has not yet reached a sufficient level of prediction accuracy (section 6.1.1), assuming that an accurate way of evaluation really is key for our method.

6.2.1 Limitations of our method

Currently, our method only functions in a full information setup of the game where both players have all details of their opponent's team. This means we have not been able to assess how the method performs in a normal battle setting. Simultaneously, it prevented us from executing the experiment via the Showdown ladder against human players, obtaining an Elo rating in the process which would have been a more optimal performance measurement.

A more technical limitation of our method is the current search depth of one. State-of-the-art Chess AI all utilize a deep search to better approximate a state's evaluation. The possibility exists to recursively apply our search algorithm, where the value for each non-leaf state will be the expected payoff of the corresponding Nash equilibrium. Of course, speed would become a limiting factor and smarter (perhaps Monte Carlo) search variations would have to be explored with it.

6.3 Extending the method for incomplete information battling

The original plan for this project was to develop a method that would function in a regular Pokémon battle without any information on the opposing team. Unfortunately, due to time constraints the method had to be cut short. This also means we have not been able to fully flesh out what this complete method would look like. Hence, why it will not be covered in depth. In short, this new method contains four separate modules and works as follows:

1. *Pokémon set model*
   This new module is responsible for attaching probabilities to each of the game's possible info-states (game states with the hidden information filled in, representing theoretically possible game states). The Pokémon set model would be initiated via the Pokémon teams within our dataset and updated as the game progresses, pruning info-states that would no longer be a possibility. This represents how a human player discovers information about the opposing team during battle.

2. *Search*
   Functions in the same way as described in section 4.1, but instead it goes over the combination of actions available to the players within a given info-state. The search algorithm will be called upon multiple times; once for each of the *n* most likely info-states.

3. *Evaluation network*
   Remains the same as described in section 4.2, although the training states may have to be changed to represent an incomplete-information perspective for a player in battle.

4. *Bayes-Nash equilibrium calculation*
   Replaces to the original Nash equilibrium calculation described in section 4.3. A single Bayes-Nash equilibrium will be calculated using *n* payoff matrices (one for each of the *n* most likely info-states).

7. Conclusion

We proposed a method that can estimate Nash equilibria in Pokémon battles by combining search with an evaluation network. This research aimed to investigate the method's viability for Nash equilibrium strategy estimation. We set to find out if a network trained on a gen-8-ou dataset could accurately predict the winner of a Pokémon battle, whether the Nash equilibrium approximation performance increases alongside increased evaluation accuracy and whether our method is sufficient to beat Mariglia's Safest. The network's outcome prediction accuracies were measured for various game length and rating ranges, achieving an overall accuracy of 0.740. We assessed the performance of our full method by setting up a series of battles against Mariglia's Safest within a complete information setting, using regular and simplified team setups. Only in the simplified setup did our method outperform this benchmark AI. The results suggest that the Nash equilibrium approximation performance is tied to the evaluation accuracy, also indicating the evaluation network likely to be the bottleneck for this method. Whether the evaluation accuracy can be improved upon using similar architectures with supervised learning remains unclear. Future attempts to increase the evaluation accuracy are required to really determine the potential of our approach.

7.1 Future work

Future research should first focus on applying the method to incomplete-information games as described in section 6.3. Next, the possibilities for an increased evaluation accuracy can be researched. To achieve this, we believe there are a few different paths that may be considered. To begin, an efficient way to extend the search depth should be explored. As mentioned in section 6.2.1, we can achieve this by applying a recursive adaptation of our current search algorithm. Furthermore, the evaluation network architecture could see a change. There exist many different network structures to experiment with for future builds, like by adding convolutional layers or by changing to a dual-headed outcome and move prediction network. In addition to this, it will be interesting to see how a model trained on a larger (and higher rated) dataset performs. As covered in section 6.1.1, we have only been able to train our model on low rated games. Future work will have to show to which extend higher rated data can lead to a performance increase. Finally, the possibility for reinforcement learning can be explored. This removes the need of an external dataset altogether and could, in theory, allow an agent to be trained using a specific team of Pokémon.

Bibliography

[1] J. F. Nash, Non-Cooperative Games, *Annals of Mathematics,* vol. 54, no. 2, p. 286, 1951.

[2] I. L. Glicksberg, A Further Generalization of the Kakuatni Fixed Point Theorem, with Application to Nash Equilibrium Points, 1952. [Online]. Available: http://ams.org/journals/proc/1952-003-01/s0002-9939-1952-0046638-5/home.html. [Accessed 19 6 2022].

[3] Pokémon Showdown, [Online]. Available: https://pokemonshowdown.com/.

[4] Pokeaimmd sample teams, [Online]. Available: https://www.pokeaimmd.com/overused. [Accessed June 2022].

[5] P. Mariglia, Showdown, [Online]. Available: https://github.com/pmariglia/showdown/tree/4e704ebe2b3f653c5bd6fe3ddf448b2e4d0b94e4. [Accessed June 2022].

[6] fastText, [Online]. Available: https://github.com/facebookresearch/fastText/. [Accessed June 2022].

[7] L. Grad, Helping AI to play hearthstone using neural networks, 2017. [Online]. Available: https://annals-csis.org/volume_11/drp/561.html. [Accessed 19 6 2022].

[8] QuantEcon Game theory, [Online]. Available: https://github.com/QuantEcon/QuantEcon.py/tree/master/docs/source/game_theory. [Accessed May 2022].

[9] C. E. Lemke and J. T. Howson, Equilibrium Points of Bimatrix Games, *Journal of The Society for Industrial and Applied Mathematics,* vol. 12, no. 2, pp. 413-423, 1964.

[10] A. McLennan and R. Tourky, From Imitation Games to Kakutani, 2005.

[11] S. Sagona, The Game Theory of Pokemon: AI implementing Nash Equilibrium, 2014.

[12] Introduction To Competitive Pokemon, [Online]. Available: https://www.smogon.com/dp/articles/intro_comp_pokemon. [Accessed June 2022].

[13] N. Brown, N. Brown and T. Sandholm, Superhuman AI for multiplayer poker, *Science,* vol. 365, no. 6456, pp. 885-890, 2019.

[14] Pokémon Showdown Rating System, [Online]. Available: https://pokemonshowdown.com/pages/ladderhelp [Accessed June 2022].

[15] P. Mariglia, Safest, [Online]. Available: https://github.com/pmariglia/showdown/tree/4e704ebe2b3f653c5bd6fe3ddf448b2e4d0b94e4/showdown/battle_bots/safest. [Accessed May 2022].

[16] M. Bowling, N. Burch, M. Johanson, O. Tammelin, Heads-up Limit Hold'em Poker is Solved, 2015.

[17] D. Huang and S. Lee, A Self-Play Policy Optimization Approach to Battling Pokémon, 2019.

[18] V. R. Konda and J. N. Tsitsiklis, Actor-critic algorithms, 2002. [Online]. Available: http://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf. [Accessed 19 6 2022].

[19] PyTorch, [Online]. Available: https://pytorch.org/. [Accessed May 2022].

[20] D. P. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, 2015. [Online]. Available: https://dare.uva.nl/search?identifier=a20791d3-1aff-464a-8544-268383c33a75. [Accessed 19 6 2022].

[21] Silver, D., Huang, A., Maddison, C. *et al*, Mastering the game of Go with deep neural networks and tree search, *Nature*, vol. 529, no. 7587, p. 484–489, 2016.

[22] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan and D. Hassabis, Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, *arXiv: Artificial Intelligence*, 2017.

[23] Silver, D., Schrittwieser, J., Simonyan, K. *et al*, Mastering the game of Go without human knowledge, *Nature*, vol. 550, no. 7676, p. 354–359.

[24] K. Chen and E. Lin, Gotta Train 'Em All: Learning to Play Pokémon Showdown with Reinforcement Learning, 2018.

[25] N. Brown and T. Sandholm, Libratus: the superhuman AI for no-limit poker, 2017. [Online]. Available: https://ijcai.org/proceedings/2017/772. [Accessed 6 2022].

[26] N. Brown, A. Lerer, S. Gross and T. Sandholm, Deep Counterfactual Regret Minimization, *arXiv: Artificial Intelligence*, 2018.

[27] R.Coulom, Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, 5th International Conference on Computer and Games, 2006.

[28] A. Grover and J. Leskovec, node2vec: Scalable Feature Learning for Networks, *arXiv: Social and Information Networks*, 2016.

[29] Future Sight AI, [Online]. Available: https://www.pokemonbattlepredictor.com/FSAI. [Accessed June 2022].

[30] OpenAI. *et al*, Dota 2 with Large Scale Deep Reinforcement Learning, 2019.

[31] Vinyals, O., Babuschkin, I., Czarnecki, W.M. *et al*, Grandmaster level in StarCraft II using multi-agent reinforcement learning, *Nature*, vol. 575, no. 7782, pp. 350-354, 2019.

[32] B. Oshri and N. Khandwala, Predicting Moves in Chess using Convolutional Neural Networks, 2015.

Appendix

A. Experiment Pokémon teams

This appendix contains the details of the Pokémon teams used in our experiment (5.2 Experiment 2: Battling a benchmark AI). The teams are a direct export from the Pokémon Showdown Teambuilder.

A.1 Simplified teams

Team A

Arcanine
Ability: Intimidate
EVs: 1 HP
- Flare Blitz
- Toxic

Tapu Bulu
Ability: Grassy Surge
EVs: 1 HP
- Horn Leech
- Leech Seed

Blastoise
Ability: Torrent
EVs: 1 HP
- Scald
- Flip Turn

Team B

Landorus-Therian (M) @ Choice Band
Ability: Intimidate
EVs: 4 HP / 252 Atk / 252 Spe
Adamant Nature
- U-turn
- Rock Slide
- Earthquake

Zapdos @ Leftovers
Ability: Pressure
EVs: 252 HP / 252 SpA
Modest Nature
IVs: 0 Atk
- Volt Switch
- Hurricane
- Toxic

Tapu Fini @ Light Clay
Ability: Misty Surge
EVs: 252 HP / 4 SpA / 252 SpD
Calm Nature
IVs: 0 Atk
- Scald
- Nature's Madness
- Light Screen

Tapu Bulu @ Lum Berry
Ability: Grassy Surge
EVs: 252 HP / 4 Atk / 252 Def
Impish Nature
- Horn Leech
- Leech Seed
- Substitute

A.2 Sample teams

Team A

Mamoswine @ Life Orb
Ability: Thick Fat
EVs: 252 Atk / 4 SpD / 252 Spe
Adamant Nature
- Earthquake
- Ice Shard
- Icicle Crash
- Stone Edge

Ninetales-Alola @ Choice Specs
Ability: Snow Warning
EVs: 4 Def / 252 SpA / 252 Spe
Timid Nature
IVs: 0 Atk
- Blizzard
- Freeze-Dry
- Aurora Veil
- Moonblast

Rotom-Heat @ Heavy-Duty Boots
Ability: Levitate
EVs: 244 HP / 16 SpA / 248 Spe
Timid Nature
IVs: 0 Atk
- Nasty Plot
- Overheat
- Volt Switch
- Pain Split

Togekiss @ Choice Scarf
Ability: Serene Grace
EVs: 4 Def / 252 SpA / 252 Spe
Timid Nature
IVs: 0 Atk
- Air Slash
- Flamethrower
- Dazzling Gleam
- Trick

Kommo-o @ Leftovers
Ability: Bulletproof
EVs: 248 HP / 200 Def / 60 Spe
Impish Nature
- Stealth Rock
- Body Press
- Earthquake
- Taunt

Corviknight @ Leftovers
Ability: Pressure
EVs: 248 HP / 48 Def / 212 SpD
Careful Nature
- Brave Bird
- U-turn
- Roost
- Defog

Team B

Inteleon @ Metronome
Ability: Torrent
EVs: 252 SpA / 4 SpD / 252 Spe
Timid Nature
IVs: 0 Atk
- Snipe Shot
- Ice Beam
- Dark Pulse
- Work Up

Melmetal @ Leftovers
Ability: Iron Fist
EVs: 224 HP / 32 Atk / 32 Def / 196 SpD / 24 Spe
Adamant Nature
- Body Press
- Double Iron Bash
- Thunder Punch
- Acid Armor

Incineroar @ Heavy-Duty Boots
Ability: Intimidate
EVs: 248 HP / 188 SpD / 72 Spe
Careful Nature
- Taunt
- Flare Blitz
- Knock Off
- Parting Shot

Stunfisk @ Leftovers
Ability: Static
EVs: 252 HP / 224 Def / 32 Spe
Bold Nature
IVs: 0 Atk
- Earth Power
- Stealth Rock
- Discharge
- Toxic

Clefable @ Leftovers
Ability: Magic Guard
EVs: 252 HP / 4 Def / 252 SpD
Calm Nature
IVs: 0 Atk / 30 Spe
- Wish
- Protect
- Teleport
- Moonblast

Hawlucha @ Rocky Helmet
Ability: Limber
EVs: 252 HP / 80 Atk / 176 Spe
Jolly Nature
- Defog
- U-turn
- Flying Press
- Roost