**THE VERY BEST LIKE NO ONE EVER WAS: PLAYING POKÉMON WITH DEEP REINFORCEMENT LEARNING**

**PROGRESS REPORT I:**

Kenan Rustamov Jake Baumbaugh

Sean Steinle Gordon Lu

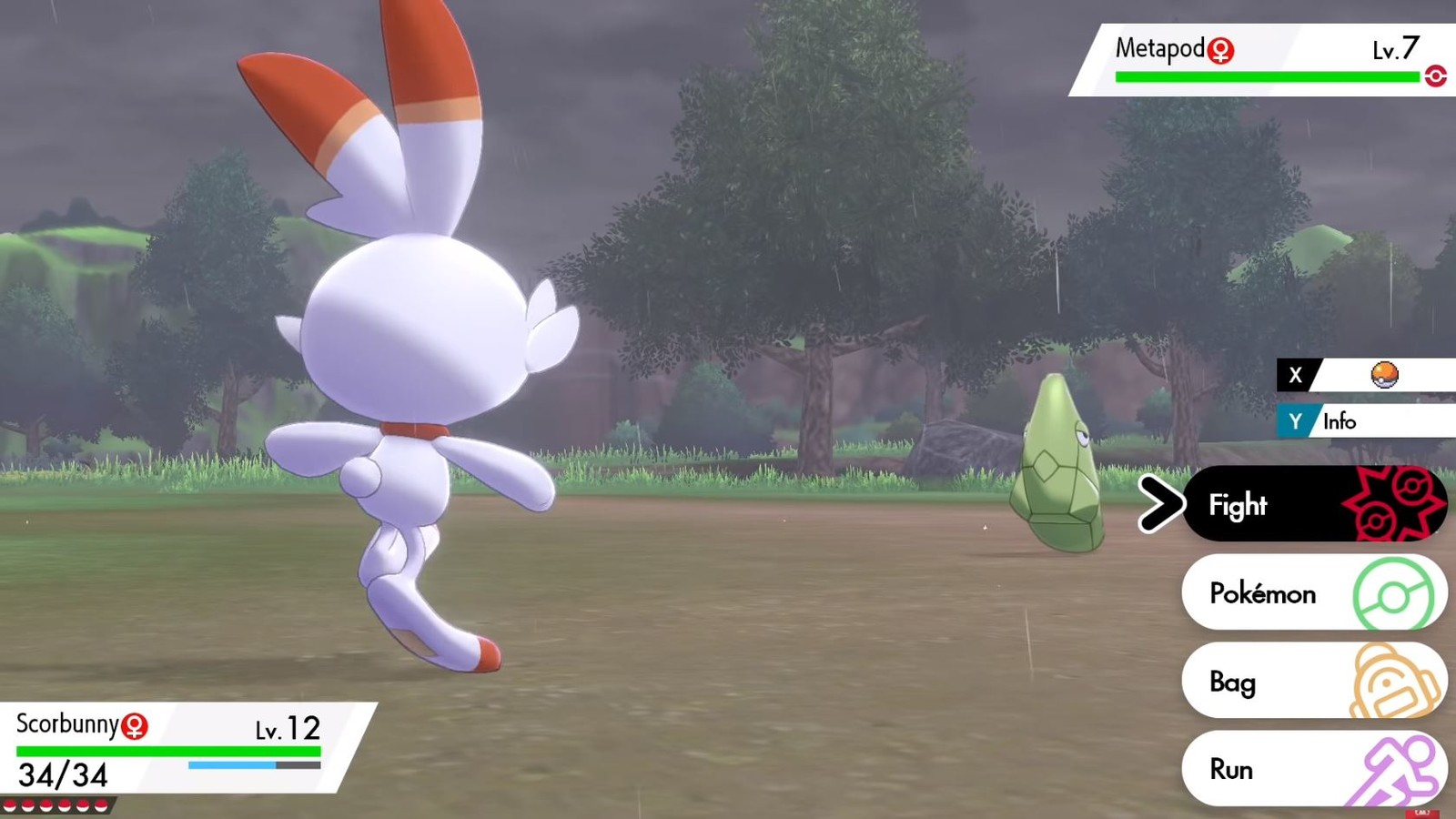
**PART I: BACKGROUND AND PREVIOUS WORK**

**BACKGROUND**

On a cold day in February in 1996, a computer did the unthinkable: it beat the world’s best chess player. This match between IBM’s Deep Blue and chess grandmaster Garry Kasparov came in the midst of a technological boom. The first websites were coming online, home computers were becoming accessible, and a new question was being asked—how can humans understand computers? 24 years later, our team answers that question the same way that IBM did: humans can best understand computers through their own games. Thus, our project will update this three decade old framework into more recent culture and technology. We will build a deep learning model that can best a human opponent in a Pokémon battle.

**THE WORLD OF POKÉMON**

Pokémon is a massively popular RPG game set around collecting and training Pokémon which are animal-like creatures that players can fight with. Each of these creatures have their own unique attributes, levels, names, and move sets that they can utilize against other Pokémon. As the game grew, new games were created and in each game a new generation of Pokémonwere introduced into it. This means that overtime the number of Pokémon, moves and types increased, which lead to increases in the number of outcomes in a Pokémon battle. To gain a better understanding of the game as a whole and competitive play, we are going to be focusing on Generation 8. This is the newest generation of Pokémon and has the most players that can be used for testing.



**BATTLE INFRASTRUCTURE**

To set this challenge into motion, we need a few tools. First, we need a place where the battle will be hosted. For many of us, that was on a Gameboy Advance with a two squared inch screen, but that won’t be compatible with our deep learning model. Instead, the existing approaches to this task have either used the Pokémon Showdown competitive battle website and its corresponding Python API, Poke-env, or they’ve implemented a battle interface from scratch. To maximize our time spent making non-trivial contributions to the task, we’re going to use the former.

Poke-env is a Python library that allows for deploying local Pokémon battle servers or for accessing the Pokémon Showdown online servers. In addition, the Poke-env library also provides robust and well-documented support for controlling battle game flow, which can be used to create autonomous rule-based or neural agents. We will use this toolkit to create simpler rule-based agents and to implement our reinforcement learning model. Finally, we will need opponents that our model can learn from, so we’ll train the model against simple rule-based agents on a local server, and we’ll also train and test the model against less predictable human agents via the online servers provided by Poke-env.

**LEARNING IMPLEMENTATIONS**

Now that we have detailed the different ways to conduct battles, let’s see the ways we can implement training a model. To our surprise, there are several tutorials and resources available, such as the extensive Poke-env documentation that includes a tutorial specifically for reinforcement learning, and a great post on Towards Data Science by Caleb Lewis that gives a brief overview of the task—with one particular q-learning based solution. Both of these resources suggest using the Poke-env library to implement all agent actions, and the Python library gym to implement the reinforcement learning algorithm that will guide our deep learning agent.

**LEARNING ALGORITHMS**

Despite a healthy number of online tutorials and programming support, this task has little academic notoriety. We were only able to find a few papers that work directly on this problem. The first is from a pair of undergraduates out of Stanford University, Kevin Chen and Elbert Lin, titled, “Gotta Train ‘Em All: Learning to Play Pokémon Showdown using Reinforcement Learning”. This paper was followed by and referenced by a triplet of Stanford undergraduates—Akshay Kalose, Kris Kaya, and Alvin Kim—in their more recent paper, “Optimal Battle Strategy in Pokémon using Reinforcement Learning”.

The defining algorithmic dichotomy of the Pokémon battle task is search algorithms or neural models. In most of today’s tasks, neural models are extremely effective and are thus the state-of-the-art models. However, in this task there is one very high-performing search algorithm—the max damage algorithm. This algorithm simply chooses the move/action that causes the highest damage to the opponent. This algorithm is impressively effective, winning 95% of the time against Chen and Lin’s model.

Even though the max damage algorithm is extremely effective, we know from domain knowledge that it is not always the optimal battle strategy. For example, some moves may disable the opponent for several moves, or damage the opponent at small increments for the remainder of the battle. While these moves have obvious utility, they are not employed by max damage algorithms, because they do not provide their utility as a one-time expression of damage. We see further evidence for the effectiveness of non-max damage strategies in the strategies of elite human players, who often use moves that increase their Pokémon’s abilities or decrease their opponents in the long term. It is critical to keep these simple-but-effective algorithms like max damage in mind because we want to build a neural model that has not just a higher success rate, but also a more robust strategy than a simple algorithm.

The most popular reinforcement learning algorithm that has been tried on this task is Deep Q-learning, which pairs action-state tuples with reward values to facilitate learning. Both Lewis and Kalose, Kaya, and Kim use this algorithm. Chen and Lin, however, opted instead for proximal policy optimization (PPO), an algorithm which incorporates new policy at each state that maximizes advantage while staying true to existing policy at some rate ε. Chen and Lin chose this algorithm in place of Q-learning because Q-learning can be insufficient in learning very complex functions, a strand of complexity that they believe Pokémon battle strategy falls under. Chen and Lin also considered using the asynchronous advantage actor-critic (A3C) algorithm—an algorithm which emphasizes stabilizing training—but they cited limited computational resources as another reason for using PPO.

**MODEL ARCHITECTURES**

As compared to the learning algorithms and the battle environments, both teams focused far less on the model architecture and the subsequent training and testing of their models. In fact, the Kalose, Kaya, and Kim team did not even specify the depth or width of their model. They did, however, detail the data they used as their input. They created two separate data structures to represent the game—Pokémon and moves. The Pokémon object includes attributes like speed, attack, level, type, and so on; the move object includes type, power, class, and more. They then used these data objects to inform their damage equation, which is a rough approximation of the one used in the Pokémon games and Pokémon Showdown. These two data objects and the model for the central action in battles comprised the state/action tuple used in Q-learning.

Chen and Lin did specify their architecture, which is a basic multi-level perceptron with three layers. There are 512 units in the hidden layer, and the activation function is ReLU. Chen and Lin’s model does not need to inform a state/action tuple like Kalose, Kaya, and Kim, but it does still need an input layer. For their input, Chen and Lin created embeddings from a rich Pokémon dataset. These embeddings include ten or so dimensions, including attack, speed, defense, legendary, type, and more. Chen and Lin checked these embeddings by clustering the Pokémon and confirming that the clusters did indeed share common traits.

**TRAINING AND TESTING THE MODEL**

Perhaps one of the least explored areas of this task is the qualitative and quantitative analysis of the task. Chen and Lin succeeded in creating multiple agents to train their model against. They created a random agent, which selects any move at random; a default agent, which selects any non-switching move at random; and a max damage agent, which selects the move that would inflict the most damage on the opponent. With these agents, Chen and Lin trained and tested three models—one for each agent. Their random agent model and default agent model both performed very well, succeeding at a 0.85 and 0.58 reward per epoch respectively. On the other hand, the max damage agent model performed very poorly, sporting a -0.90 reward per epoch. Chen and Lin posit that the reason the max damage agent model performed so poorly is that the max damage agent was simply too powerful. In fact, they say it was so good that the model believed “there is no move that would clearly be more beneficial in that game state.” Chen and Lin also encountered an anomaly in their model, wherein the model was extremely preferential to the fourth move available, no matter what the move was. They offered no explanation for this phenomenon, other than a curiosity to learn more.

Instead of training one algorithm against several agents, Kalose, Kaya, and Kim trained two algorithms against one random agent. They first trained an epsilon-greedy Q-learning model against the random agent--which won 60% of its battles—before moving on to train a softmax q-learning model—which won 65% of its battles. Although the two metrics (reward per epoch and win percentage) are not the same, it does seem as though Chen and Lin’s random agent model performed better. All this considered, both papers were clearly focused more on the algorithm design and the battle environment rather than analysis. This may be one way our model can be novel. Our model will focus on generalizing to a battle between trainers in a newer generation by having our Pokémon and our opponents' Pokémon be randomly generated by Pokémon Showdown.

**PART II: AN OUTLINE OF OUR MODEL**

**A NEW KIND OF Q-LEARNING**

As one of the best studied reinforcement learning algorithms, we would first like to explore the option of deep Q-learning. Chen and Lin were critical of this approach, and by the statistics laid out by both them and Kalose, Kaya, and Kim, they have valid reasons to believe this. While Chen and Lin’s epochs were not precisely defined, their 0.85 average reward against a random agent (translating to roughly a 92.5% win rate) greatly outweighs Kalose, Kaya, and Kim’s 65% after training on 5,000 battles. But, we believe that this may be due to other simplifications made by Kalose, Kaya, and Kim, rather than the choice of algorithm. Most notably, they excluded information about the remaining Pokémon on each team not currently active in battle. As such, they could not plan which Pokémon of theirs to reserve for a later match-up, or whether to play more defensively if they have no more Pokémon to switch to in the event that the current one dies. They also elected to group health percentages into groups of 10%, though this likely had much less of an impact on outcome.

Kalose, Kaya, and Kim made these simplifications with the overall goal of reducing the number of possible states the model needed to consider. At the center of basic Q-learning, sometimes called “vanilla Q-learning,” is the Q-table mapping action-state pairs to reward values as mentioned above. As such, the Q-table grows in size linearly as the number of possible states in the battle grows. Including more features to be considered in the program state will exponentially increase the cardinality of the set of states, so these simplifications were paramount in making their model viable. We believe that this simplistic model is simply not fit to handle a task with so many states such as Pokémon. We instead have opted for a deep Q-learning approach which uses a neural network to compute Q-values. With our deep Q-learning approach, our input will be a feature vector as is typical for most neural nets, so the input size need only grow linearly with additional features. Therefore, we can choose features for the network to include without significant concern of overflowing the memory capacity of the system or slowing the computation of actions to a halt.

**NETWORK FEATURES**

Pokémon games have a wild assortment of different variables and possibilities that work in combination to make battles feel unique. As such, we needed to decide which of these variables are absolutely relevant to the state of a battle. Ideally, we can simplify the model while avoiding any loss of information that would aid the model in playing the battle. Most obviously, statistics about the currently equipped Pokémon such as the health, attack, defense, special attack, and special defense will be required to choose the best move at hand. If health is low, our model will want to switch to a new Pokémon, so it would be useful to have these statistics about the other five Pokémon on our team as well. Past these statistics, every Pokémon has either one or two types, which influence its possible moveset and its weakness to certain moves. These types fall under categorical data, and as such we will represent them in the features as a one-hot vector of length 18, representing the 18 different Pokémon types. While this will have the effect of adding a significant number of features to the input space, type advantages are a significant part of the battle system, and cannot be overlooked. We will also want to know as much as we can about our opponent’s Pokémon team, though that information is limited. The opponent’s Pokémon currently on the battlefield will display its health, as well as its types, both of which we will want to keep track of (the types being represented in another one-hot vector). We can also be aware of these stats for Pokémon that the opponent previously had on the field, and we can assume that the health is 100% otherwise. The model has no way of knowing the types of yet unrevealed Pokémon, so the feature dimensions correlating to it will be a zero vector until revealed.



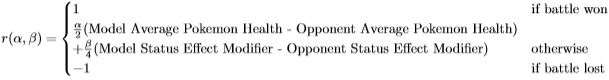
Another feature dimension to consider are the moves that each Pokémon has to choose from. It is important to consider the possible moves that your Pokémon have, as these can determine if the Pokémon to be switched to will have an advantage over the opponent’s current Pokémon on the battlefield. Knowledge of these moves is also imperative in battle, where the model must choose between these available moves. We will attempt to map these moves through a series of feature dimensions using scalar values and one-hot vectors where necessary. Each move has a type, power, accuracy, possible status effect, and possible weather effects. Status effects can influence either Pokémon on the field, providing some change to the stats or actions of the chosen Pokémon. The weather effects affect both Pokémon on the field, as well as the outcomes of the chosen moves. The type, status effect, and weather effect of the move can be one-hot vectors similar to that used for modeling the type of our Pokémon team. Power and accuracy can each be separate features represented by scalar values. These different values and features are considered heavily by top expert Pokémon players, so it is important for our model to attempt to emulate it.

**NETWORK OUTPUT**

With the input features to the network defined, it is critical to understand the output format as well. Following closely to the outputs laid out by Chen and Lin, we have ten possibilities: four moves to choose from, five Pokémon on our team we could switch to, and a final no-op move, which should be saved as a last resort if no other moves are possible. The network will use the features to decide what it believes to be the best of these moves to choose from, the highest of which will be the chosen course of action. In the event that the action chosen by the model is invalid given the current state, the next best valid action will be chosen according to the model’s output.

**MOVE EVALUATION**

Once the model has made its move, we need an evaluation function to determine the overall reward of the move. A common reward function that makes itself apparent is to return one if the model has won the battle, negative one if the model loses, and zero for all other battle states. This is the function that Chen and Lin implemented, and while it has the benefit of removing any bias towards what the best move might be, it significantly limits the amount of learning the model can perform during a battle. Kalose, Kaya, and Kim, on the other hand, toyed with the idea of an intermediate reward based on the relative healths of the Pokémon on the battlefield, providing reward when the model’s Pokémon was ahead. We decided to introduce into our reward function another intermediate reward based on what status effects each Pokémon has at the time, encouraging positive status effects for the model’s Pokémon and negative ones for the opponent. The effect of these intermediate rewards can be controlled by parameters ⍺ and 𝛽, which scale the health modifier and the status effect modifier, respectively. ⍺ and 𝛽 should each be bound in the range [0,1] to represent scaling, with the overall reward function returning between [-1,1]. This results in the final reward function:



**NETWORK ARCHITECTURE**

The remaining decisions to be made about the model are mostly just questions of hyperparameter values, as well as choice of activation functions and a loss function. While we cannot predict the ideal values of hyperparameters at this stage, we would like to approach the problem with a multi-layered neural network, allowing space to understand the intricacies of Pokémon battles dealing with type advantages, status effects, and more. The activation functions can be decently straightforward - using ReLU at the hidden layers, with an identity function at the output layer. This is beneficial in q-learning, as it best replicates the idea of a table of q-values, rather than transforming these values, limiting them to some range. Lastly we will use a squared error loss function for performing the gradient descent, finalizing the structure of our network.

**PART III: WHY IT’S UNIQUE**

**APPROACH JUSTIFICATION**

Previously Chen and Lin were critical of the use of Q-learning as an optimal algorithm in conducting Pokémon battles, but Kalose, Kaya, and Kim were able to achieve mild success with it still. By building off the previous success of Kalose, Kaya, and Kim with an improved feature space and a more generalized algorithm, we should be able to achieve greater performance against random or human agents. Deep Q-learning has also had success in other games and even showed that one model could be generalized to multiple games, most notably being applied to a number of Atari games in a 2013 paper by Volodymyr Mnih et al. This means that it should be able to handle a game like Pokémon, which requires a greater amount of generalization and complexity.

**NOVELTY IN APPROACH**

While Kalose, Kaya, and Kim tested their models against only random agents and Chen and Lin trained and tested on a variety of automated agents, the Poke-Env API for Pokémon Showdown, as mentioned above, provides a unique opportunity to test and potentially even train against other human players on the platform. Each of these papers outlined the fact that reaching win rates even moderately above average requires thousands of Pokémon battles to be played out and learned from. As such, we aren’t entirely confident in our ability to train our model to this level if playing only against human players which are notably slower than automated models. Even without this, we will be able to test our models against real players, an option we only found explored in one prior document. Written by Kush Khosla, Lucas Lin, and Calvin Qi, “Artificial Intelligence for Pokemon Showdown” is another paper from Stanford undergraduates seeking to apply neural networks and machine learning to Pokémon. After developing their four algorithms, they tested them on Pokémon Showdown servers to evaluate the individual performance of each, something we intend to explore.

Another way our approach is novel is that we are prioritizing the direction of training for our model. As mentioned before, we do not simply want to train a model that always selects the highest damage move, as there is already a simple algorithm that accomplishes this. Instead of a model that always picks the highest damage move, we want a model that will use a move with longer term payoffs--like a human would do. We know from watching human trainers play that there is an optimal strategy that can consistently beat the max damage algorithm: this is the strategy we want our model to employ.

To this end, we will leverage the construction of adversaries and the construction of Pokémon teams to guide our model’s learning. As mentioned earlier, the only agents that have been trained against are a completely random agent, a semi-random agent, a max damage agent, and human agents. We will expand this list to include rule-based agents that employ different strategies, such as an agent that prefers to fight longer battles with less high damage attacks. The agents we will create will be the random agent, the max damage agent, the status agent, the defense agent, and the offense agent. Finally, while we’re primarily going to train these agents with random sets of Pokémon, we’ll also be experimenting with tailored Pokémon lineups. For example, we will give the defense agent a set of high defense Pokémon.

Another dimension in which our approach is novel is that we will test the different agents and models we create against each other. In all of the previous work on this topic, the authors have not experimented with testing models trained on different agents against each other. Despite our prior expectation that some models will be significantly less competitive than others--the random model will likely not fare well against the max damage model--there is great potential for understanding the different models’ strategies. Perhaps the defense model will outlast the max damage model, or the status model will outflank the attack model. This approach will result in a large number of battle pairs to interpret, but that is the defining strength of this approach: we will have a huge amount of data that we can analyze to better understand the battle strategies of our models. After initial testing, we will have the different agents and models compete in a battle tournament, which may in part speak to the adaptability of the winning agent.

**PROJECT AMBITION**

The basis for our entire project is reinforcement learning. With the necessary scaffolding for reinforcement learning being covered late in the semester, it is certainly a major reason why our approach is ambitious. Due to our model using an advancement reinforcement learning algorithm, in our case deep Q-learning, the challenge in this task is far greater than using a simple neural network. As stated above, selecting features to include without concern of slowing the computation of actions to a halt, will certainly be a challenge, and in itself contributes to the ambition of the approach. Additionally, Pokémon itself is a difficult problem to attack. There are 932 total Pokémon, with each Pokémon having their own respective stats. For each Pokémon, each move would not only need to be selected based on the move that best suits their highest stat, but also to consider conditions such as attaining a status, or a change in stats. An important point to note is that in the Stanford paper by Chen and Lin, they trained against 3 different agents. On the other hand, another way our proposed approach is ambitious is that we seek to train against many agents.

With the paper by Kalose, Kaya, and Kim, their approach introduced simplifications, one of which was to limit the amount of Pokémon from 932 to the original 151 Pokémon, as well as limited the moveset of the Pokémon to be much sparser with 165 moves out of the total 826 moves available. Additionally, they did not keep in mind the other 10 Pokémon that were not in play, only considering the current Pokémon in play. With our approach, the plan is to disallow such restrictions, and to instead allow for the full 932 Pokémon and the total 826 possible moves to be considered. In this sense, our model will be much more flexible, and will consider much more possibilities, such as switching out Pokémon dependent on knowledge known from when the opponent switches in for a different Pokémon. Such decisions are quintessential and are the grand crux of Pokémon. If Pokémon was limited to the original 151, our problem would be trivial.

**EVALUATION OF PERFORMANCE**

As there aren’t too many outcomes of a Pokémon battle, namely winning or losing, we are somewhat limited in our options for evaluating our final model. As aforementioned, Kalose, Kaya, and Kim described their final model by its win rates against a random agent. Chen and Lin, on the other hand, provided additional details on the average reward over approximately 70 epochs against a variety of agents: random, default (focus on non-switching moves), and minimax, which performed with average epoch rewards of 0.85, 0.58, and -0.90, respectively. We will be able to evaluate our model similarly, finding an average win rate after some number of battles have been performed for training, at which point we can compare our model to these previously studied attempts at Pokémon AI.

Another significant metric we can use to measure our model’s skill is Pokémon Showdown’s Elo ranking system. Designed by Arpad Elo, the system is used in zero-sum games (most notably chess) to track players’ individual skills. As seen on the Pokémon Showdown website, at the time of writing, the current best player in random battles has an Elo of about 2500. Since we will have a model training on the site against other players and AI, we can compare its performance to other players using our resulting Elo rating. Assuming that we do not surpass all other players on the platform, after some unknown number of games our win rate will approach 50% and our Elo will stabilize, providing a final rating for the model’s performance.

**IN CONCLUSION**

Our deep learning model seeks to “be the very best, like no one ever was,” just as the Pokémon theme song states. While there has not been a great deal of research put into solving Pokémon battles, we want to better any previous efforts and produce a model that will yield a significant win rate against other decent players and models.

**REFERENCES**

1. Chen and Lin:<https://cs230.stanford.edu/projects_fall_2018/reports/12447633.pdf>
2. Kalose, Kaya, and Kim: <https://web.stanford.edu/class/aa228/reports/2018/final151.pdf>
3. Lewis: [https://towardsdatascience.com/poke-agent-pokemon-battling-reinforcement-learning-27ef10885c5c \](https://towardsdatascience.com/poke-agent-pokemon-battling-reinforcement-learning-27ef10885c5c%20/)
4. Poke-env Tutorial: [Reinforcement learning with the OpenAI Gym wrapper — Poke-env documentation (poke-env.readthedocs.io)](https://poke-env.readthedocs.io/en/stable/rl_with_open_ai_gym_wrapper.html)
5. Elo rating system:<https://en.wikipedia.org/wiki/Elo_rating_system>
6. Pokémon Showdown Ladder:<https://pokemonshowdown.com/ladder/gen8randombattle>
7. Khosla, Lin, and Qi:[https://docplayer.net/63514819-Artificial-intelligence-9for-pokemon-showdown.html](https://docplayer.net/63514819-Artificial-intelligence-for-pokemon-showdown.html)
8. Mnih et al.: <https://arxiv.org/pdf/1312.5602.pdf>