CS230

Learning Competitive Pokemon through Neural Network and Reinforcement Learning

Cyril Chun Him Tse
cyriltse@stanford.edu

Abstract

In this project, an AI agent has been trained to play competitive Pokémon. It was first trained by real players' battle records, aiming to learn embedding for the battle states. Then the network is transferred to Deep Q-Learning for optimizing long term rewards. After finishing the training, the AI successfully beats the two baseline agents and shows some capability to compete against real players.

1 Introduction

Pokémon is one of the most popular video game franchises in the world. However, its official competitive format, Pokémon Video Game Championship (VGC), is different from most of the in-game single format plays. It could be difficult for newcomers to learn the format due to its ever-increasing mechanics over the 20 years of franchise history. On average, it may take a new player some months to a year to reach a competitive level. Building an AI to show the best play to them is one good way to shorten the time and make the competitive scenes more accessible, though being challenging due to a large number of possible outcomes in each turn. To do so, I will first use real player data to do supervised learning, and then transfer the network to a reinforcement learning environment to play in simulated games.

2 Related work

2.1 AI in Gaming

Deep learning has made remarkable progress to surpass human performance in various board games and videos. Multiple AI has been developed to beat human in all sort of competitive games, from transitional games like GO[2] to video games like Dota. All those AI are built in a reinforcement learning setting. Given the similarity in large action space between GO and Pokémon, I decide to build an AI with a similar architecture.

2.2 Reinforcement Learning in Pokémon

There were attempts to build a Pokémon playing AI through reinforce-learning. Results have been done using DQL or policy-gradient method[4][5][6][8]. However, none of those AI is built to play the more complex VGC format. Nevertheless, experience from those projects should be transferable to this project. One of the most useful resources coming from those research is the architecture of simulating Pokémon battles. A python library called Poke-env has been created [7]. It communicates with a showdown server through JavaScript and simulates Pokémon battle for a reinforcement learning

agent. Though it requires bridging two programming languages and has speed implications, it can save us from building a complicated Pokémon simulator from scratch.

3 Dataset and Features

Data used in this project comes from Pokémon Showdown's public server, a fan-made online Pokémon battle platform. They have provided over 4 million battle records for the project. Amongst those battle records, only games from the top 20% ranked players are put into the Supervised learning model, so the network can focus to learn the top players' action. Also to prevent the model overfits to successive high-correlated game states in the same games, only one action and the corresponding game state are sampled in each game.

Records are originally stored in game-log (text description of the game in Json format) and were transformed into state features in Figure1. 700 thousands battle are transformed and 30 thousand of them will serves at dev-set. No test-set is needed as the model will be tested in the reinforcement stage. In the transformation process, categorical fields are transformed into embedding features by stats and type, instead of by one-hot key embedding. That is for shrinking the sizes of input features and prevent over-fitting. Even with this procedure, the game state transformation results in 4186 features.

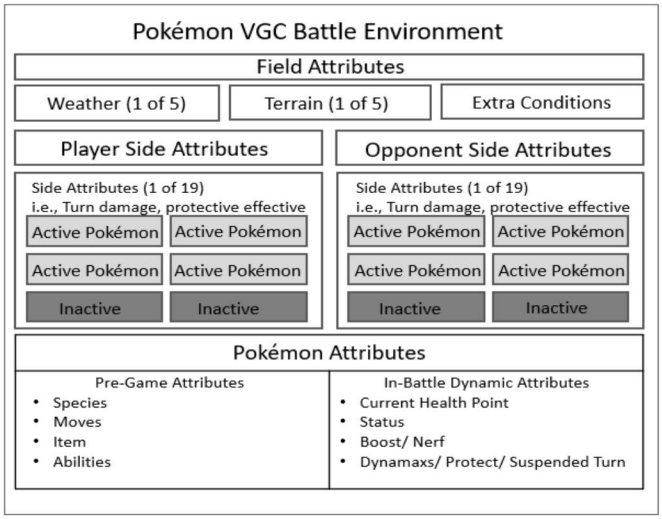


Figure 1: Features of battle state in VGC

4 Methodology and Network Architecture

The project is divided into two parts: 1) Supervised learning through real player data and 2) reinforcement learning on Monte-Carlo tree searching. It is designed in such a way to overcome the slow simulation of Pokémon games, of which speed is bottle-necked by communication with showdown simulation servers.

4.1 Supervised Learning (SL)

The main purpose of supervised learning is to build a policy network, which estimated a value function $v \epsilon[-1, 1]$ predicting battle outcome and a probability distribution of potential opponent action a through the battle state s. Its structure is identical to reinforcement agent policy, such that the reinforcement learning could have a quick start.

The SL network is a 6-layers ANN network with a final layer combining two softmax units outputting move probabilities $p(a|s)$, two numeric output specific targeting and a tanh unit outputting value $v(s)$. The network is trained on random sample pair of (a,s) using gradient descent to minimize loss function: $Loss = \alpha * CrossEntropy\_Loss(Move_1) + \alpha * CrossEntropy\_Loss(Move_2) + \beta * MSE\_loss(Target_1) + \beta * MSE\_loss(Target_2) + \gamma * MSE\_loss(Value)$. Note that here I use MSE to measure the value function, even though it is similar to the classification task on win/loss. The reason behind is that I want the output range includes negative values, so that the model can be transferred as part of the reward to the RL easily.

The first layer of the SL network is not fully-connected. All of the twelfth Pokémons have their own 1st-layer module, which will have the same parameters if the Pokémons are on the same player side. The idea is to use this layer to create embedding in order to reduce input features size.

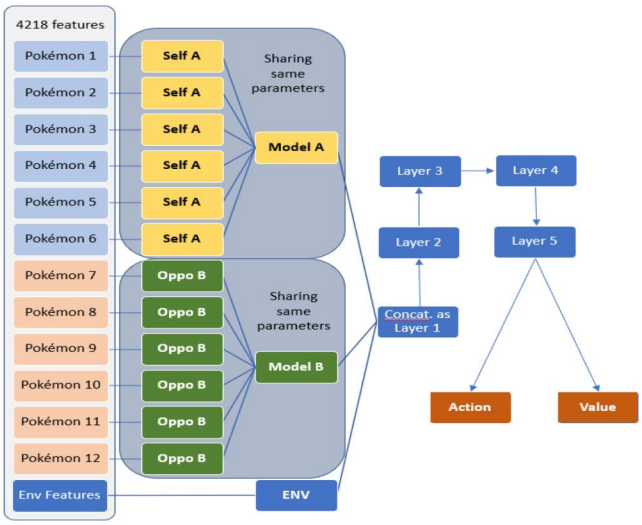


Figure 2: SL network structure

4.2 Reinforcement Learning (RL)

In the second stage of the project, the SL network (with only the action output) is transferred to a reinforcement learning environment to learn maximum the long term return of the agent. Conceptually it uses the moving probability from the top20% players as a proxy of the true Q-value of an optimal strategy. For the learning algorithm, I have adopted deep Q-learning here and use ε-greedy policy to learn action selection.

The agent then plays simulated battles in a locally hosted showdown server through the gym wrapper library. By playing enough games with different agents, the network can optimise the action selection while considering long-term reward of the game. The reward used here are:

- Outcome (+Win and -loss) of the simulated game
- % Hp of the active Pokemon on both side (+own side -opponent)
- Value function from the SL model (parameters will be fixed and not be trained in RL)

Each of the three rewards was scaled by a different hyperparameter. The outcome of the SL value function is used as empirical statistics here, aiming to measure how winnable a particular game state is. This should incentivize the agent to move to a better game state and learn the complex game easier.

5 Experiments Discussion

5.1 Supervised Learning

At the beginning of the supervised learning stage, I was using a more vanilla neural network to learn the data. However, I found that the model would inevitably overfit, even with Batch normalization and regulation techniques like dropout or L2 regularization. The overfitted model generally gave a train set accuracy over 90% , simultaneously returning an accuracy close to a random guess at dev-set.

To investigate the situation, I have picked samples from train-set and dev-set with similar game states to do some error analysis. From the analysis, I found that the network gave completely different results in game states that are similar numerically and conceptually, indicating that the model is memorizing the input instead of generalizing. Therefore, I decided to create the model with a first layer acting as embedding, which shrinks the size of the features from 4186 to 254. It turns out prevent the network to over-fit the model.

After finalizing the network structure, I start to perform hyperparameter searching. One of the more important hyperparameter searches is to decide the weight of different loss components. I set the value output as evaluating metric while putting the move accuracy as a satisfying metric (more accurate than a random guess) to evaluate the performance of different loss component weighting. The rationale behind is that value function cannot be improved in the reinforcement stage, so it is more important to prioritize it in the SL training. I eventually settle with α as 0.67 while β as 1.

result after 3 epochs	combination1	combination2	combination3
α	1	0.67	0.67
β	0.67	1	0.67
γ (set as 1 to prioritize)	1	1	1
move selection accuracy	0.21	0.19	0.18
move target accuracy	0.33	0.35	0.31
Value accuracy	0.51	**0.54**	0.54

After being trained with 90 epochs using the chosen hyperparameters, the model achieves the following accuracy on dev-set:

SL Result	Move action	Move target	Value(classification)
Accuracy	51.8%	84.6%	75.3%

5.2 Reinforcement Learning

The reinforcement learning agent is evaluated by two baseline agents. One of them is a random agent, while the other agent is a Max_Damage agent, which is essentially an agent maximize short-term return on battle damage.

After transferring the network from SL, the agent can already beat the two baseline agent 71% and 62% of times. It shows that learning from real player records can give the agent a solid start.

To begin reinforcement learning, I started to tune the hyperparameters of the reward. After doing some trials and taking references in the related study[8], I settled by using 0.05 on the hp reward and 1 on the other two rewards.

Using those hyperparameters at the reward function, the agent is then deployed in the simulator to play 100000 matches against the two agents. It have further learned to maximize the reward and achieve a better win rate.

Result of the agent after training:

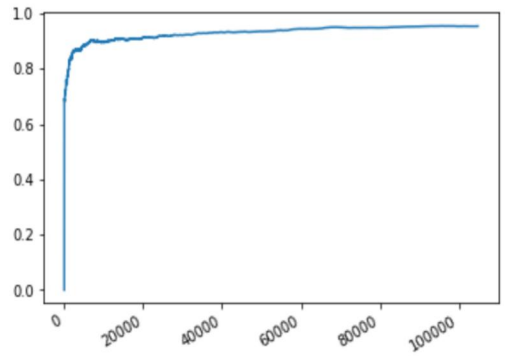


Figure 3: Win rate against random agent while training

RL Agent	vs Random Agent	vs MaxDamage Agent
Initial (SL) win rate%	71.7%	62.9%
Win rate% after 150k matches	96.6%	78.2%

I have also connected the agent to the showdown server to test with real players. It has won some matches on the lower rank ladder, showing that the AI has learned to play the game to some extent. However, the AI starts to struggle at average player ranking(1350). One thing I realize is that the AI tends to spam the same move in successive turns, which can be easily exploited by a real player. I believe it is caused by training with two unvarying opponents in the training process. Giving more time to train with a larger variety of different opponents or even self-play, this issue should be able to overcome. However, since simulating Pokemon battles is very time consuming and simulating 50000 takes around 6 hours, I could not have more training on the agent before the deadline of the project.

6 Conclusion and Future Works

By learning battle records from top-ranked players and then reinforcing the network in simulated games, the AI agent has learned to play Pokemon VGC battles to some extent. Even trained with limited resources and time, it has been able to beat the two baseline agents and can be competitive against humans in lower-ranking Pokemon battles. Given more time to train with a larger variety of opponent agents, the AI should be able to correct some of its deficiencies.

Another potential improvement is to migrate the Pokémon simulator to Python. Under the current approach, simulating Pokémon battles in JavaScript is a significant bottleneck. Also without the possibility to restart the at a specific game states, the AI waste resource to learn the exist path, and loss out the possibility to try more robust model like Monte-Carlo tree search.

7 Appendix

7.1 Library and open sourced code used

SL learning model : Pytorch
Pokemon simulation server: smogon pokemon-showdown-client at Git-hub
RL learning enviornment: poke-env
Code of RL model is inspired by leolellisr/poke_RL at Git-hub

7.2 Hyperparameter Choices in SL model

Hyperparameters	Selected value
Learning rate	1e-6
Learning rate decay rate	0.99 per epoch
layers	6
Activation function	tanh
α	0.67
β	1
γ	1

7.3 Hyperparameter selection

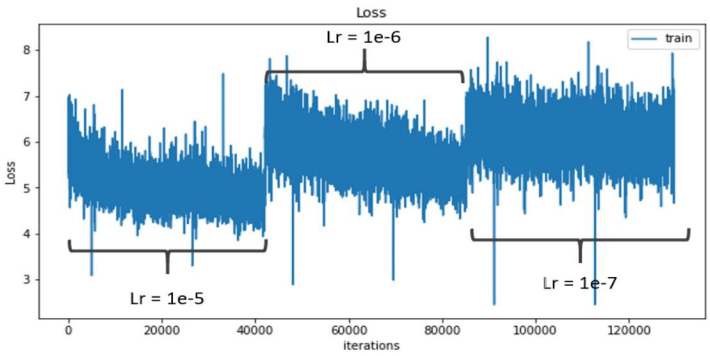
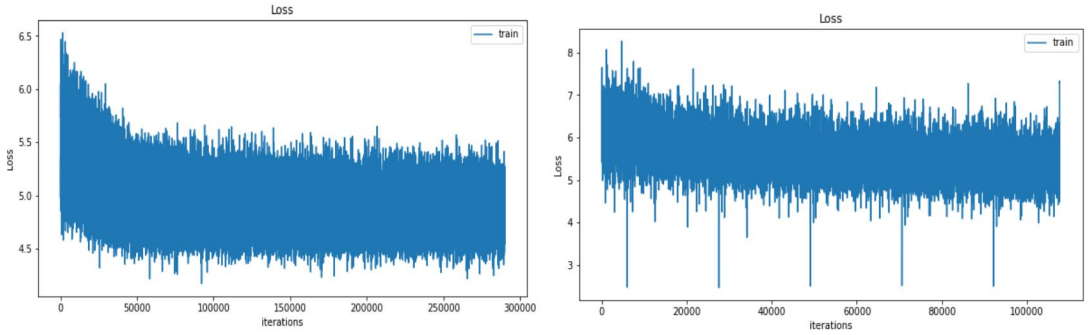


Figure 4: 1e-06 is chosen as it is the most stable one with least spikes on loss



(a) loss with tanh activation

(b) loss with Relu activation

Figure 5: Use tanh activation since it is more stable in loss plot

References

[1] Guo, X., Singh, S. P., Lee, H., Lewis, R. L. Wang, X. "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In Adv. Neural" Inf. Process. Syst. Vol. 27 (eds Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. Weinberger, K. Q.) 3338–3346 (2014)

[2] Silver, D., Schrittwieser, J., Simonyan, K. et al. Mastering the game of Go without human knowledge. Nature 550, 354–359 (2017). https://doi.org/10.1038/nature24270

[3] C. B. Browne et al., "A Survey of Monte Carlo Tree Search Methods," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 1, pp. 1-43, March 2012, doi: 10.1109/TCIAIG.2012.2186810.

[4] S. Lee and J. Togelius "Showdown AI competition," 2017 IEEE Conference on Computational Intelligence and Games (CIG), 2017, pp. 191-198, doi: 10.1109/CIG.2017.8080435.

[5] D. Simões, S. Reis, N. Lau and L. P. Reis, "Competitive Deep Reinforcement Learning over a Pokémon Battling Simulator," 2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), 2020, pp. 40-45, doi: 10.1109/ICARSC49921.2020.9096092.

[6] Elbert Lin Kevin Chen & Gotta Train. 'Em All: Learning to Play Pokemon Showdown with Reinforcement Learning. Stanford, CA.: n.d. Web. 17 May. 2022

[7] Poke-env: A python interface for training Reinforcement Learning pokemon bots. https://poke-env.readthedocs.io/en/stable/index.html#poke-env-a-python-interface-for-training-reinforcement-learning-pokemon-bots/. 17 May. 2022

[8] Pokémon RL. Git repository. https://github.com/leolellisr/poke_RL/blob/master/Notebook_Report_MO436_RL_P2.ipynb. 17 May. 2022