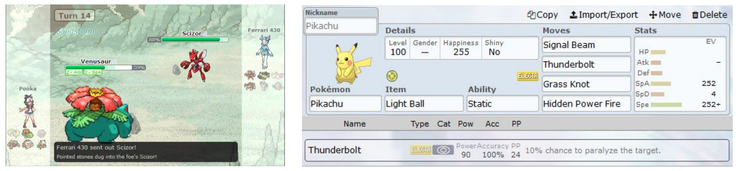
Playing Pokémon Showdown with Deep Reinforcement Learning

Kevin Chen (kvchen@stanford.edu), Elbert Lin (el168@stanford.edu)

Overview

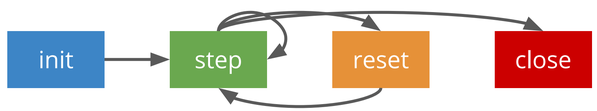
- Simultaneous zero-sum game in which players compete to defeat the opposing team's Pokémon
- Reinforcement learning methods shown to be effective on various games
- Train agents to compete in Pokémon battles against an adversarial opponent



Left: In-game screenshot from Pokémon Showdown (Guangcong Luo, 2018)
Right: Visual stats for a particular Pokémon. These statistics are used as input features to our network.

Environment

- Created a proxy server that wraps the Pokémon Showdown internals. Snapshots game state at each step to permit for deterministic rollouts at any point in the game.
- Client is exposed as an OpenAI Gym environment to allow arbitrary agents to play the game.
- Rewards: +1 if the game is won, -1 otherwise.



Feedback loop for agents in a gym environment. Agents take a step from some state and observe some new state and corresponding reward. If an episode is completed, the environment resets and a new game is started. The environment is closed and terminated at the end of an epoch.

- Three opponent agents:
  - *Random*: Selects a choice at random out of the available choices.
  - *Default*: Selects the first available choice. This will select a *move* action the majority of the time.
  - *Minimax*: Explores the game tree to a certain depth and selects the most favorable action based on a heuristic (difference between sums of fractional Pokémon healths)

Methods

Feature Extraction

- Created a wrapper around PS server to enable information extraction
- For each turn, obtained certain data about current game state
- General battle features: terrain, weather, each team's Pokémon
- Pokémon features: HP, max HP, stats, boosts, type, status, gender, level, moves
- Move features: PP, max PP, accuracy, base power, type, target

Proximal Policy Optimization

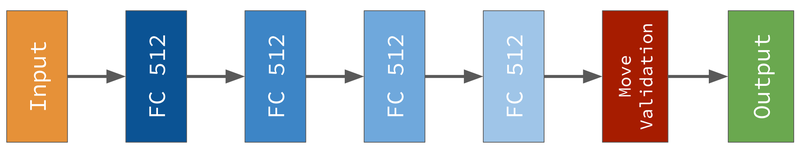
- New family of policy gradient methods, is simpler, more general, and has better sample complexity
- Uses novel objective function to apply soft constraint as penalty (see below, John Schulman et al., 2017)
- Optimizes objective by minimizing cost while maintaining small deviation from previous policy
- To simplify, uses clipping on probability ratio between old and new policies

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\right]$$

- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\hat{A}_t$ is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

Experiments

- Network architecture:

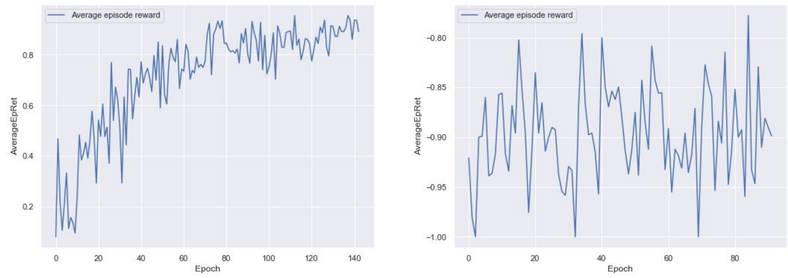


Our RL agent used a deep NN. Of note is the move validation step, in which the logits of invalid moves are set to negative infinity such that the final softmax probabilities of these invalid moves is zeroed out.

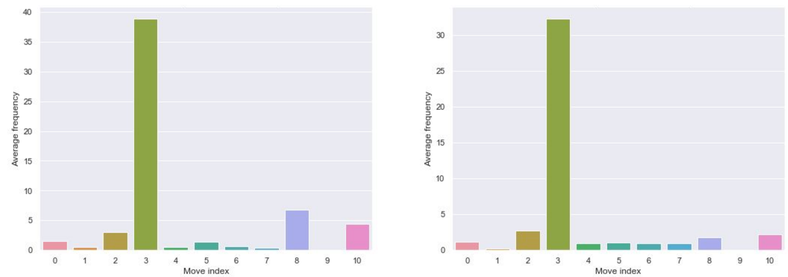
- Not all moves are valid at each timestep, so filtering network outputs was necessary.
- 250 epochs, 4000 steps per epoch
- Trained RL agent against each opponent agent and recorded average reward over each epoch

Results and Discussion

- RL agent learned to beat both random and default baseline agents
- Failed to perform well against minimax agent



Left: Rewards after training against a random agent. The RL agent quickly learns how to win almost 100% of the time.
Right: Rewards after training against a 1-ply minimax agent. The RL agent is unable to make much progress in learning how to beat this agent.



Left: Moves chosen by our trained agent against a random opponent.
Right: Moves chosen by our trained agent against a minimax opponent.

- Surprisingly, agent which does poorly against minimax performs well against random and default.

Future Work

- Optimize battle engine system to use immutable data structures, making snapshotting faster and easier to reason about.
- Train against minimax at a larger ply. Computational resources are the limiting factor here.
- Explore more complex network architectures
- Explore transfer learning - pretraining the network against an easier opponent agent, then switching over to the minimax agent

Works Cited

- Luo, Guangcong. "Pokémon Showdown." GitHub repository (2018), https://github.com/Zarel/Pokemon-Showdown.
- Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).
- Brockman, Greg, et al. "Openai gym." *arXiv preprint arXiv:1606.01540* (2016).
- Abadi, Martín, et al. "Tensorflow: a system for large-scale machine learning." *OSDI*. Vol. 16. 2016.
- Silver, David, et al. "Mastering the game of Go without human knowledge." *Nature* 550.7676 (2017): 354.