DEEP REINFORCEMENT LEARNING FOR POKEMON BATTLING

Kevin Zhang

PURDUE UNIVERSITY | Department of Computer Science

Battling Environment

Pokemon Battles offer unique challenges

- Stochastic: Accuracy, Status, Crits
- Partially-Observable: Stats, abilities, moves
- Very large state/search space: Embedding of above
- Additionally- Information may be entirely known by “testing” during battle

There are 611 Pokémon moves that are currently shown in the table.

Move Name	Type	PP	Pwr	Acc	Class	Description
Absorb	GRA	25	20	100		A nutrient-draining attack. The user's HP is restored by half the damage taken by the target.
Acid	POI	30	40	100		The opposing Pokémon are attacked with a spray of harsh acid. This may also lower their Sp. Def stats.
Acid Armor	POI	20	-	-		The user alters its cellular structure to liquefy itself, sharply raising its Defense stat.
Acid Spray	POI	20	40	100		The user spits fluid that works to melt the target. This harshly lowers the target's Sp. Def stat.
Acrobatics	FLY	15	55	100		The user nimbly strikes the target. If the user is not holding an item, this attack inflicts massive damage.
Acupressure	NOR	30	-	-		The user applies pressure to stress points, sharply boosting one of its or its allies' stats.
Aerial Ace	FLY	20	60	-		The user confounds the target with speed, then slashes. This attack never misses.
Aeroblast	FLY	5	100	95		A vortex of air is shot at the target to inflict damage. Critical hits land more easily.

Pokemon Showdown Environment

Turning Pokemon into an OpenAI Gym environment

- Pokemon Showdown offers an open-source method for generating and conducting battles

- Poke-Environment acts as an interface to Python for this data

- Select an action at each state, and use each turn as a transition
  - Follow Gymnasium's specification
  - Integrate Agents trained using Pytorch

Training Design

There are many options for training methodology

- Algorithm-
  - Optimization Method: DQN, PPO, GIGAθ, WPL, etc.
  - Opponent Design: Baseline, Selfplay, etc.

- Information Structure-
  - Embedding huge information space

- Reward Design-

Algorithm 1: Pseudo-code for a worker thread running GIGAθ using -greedy exploration.

Input: Globally, target network update period τ, on-line and target Q-networks, on-line and target policy networks, on-line and target average policy network weights, exploration rate ϵ, and maximum iterations $T_{\max}$. Locally, on-line Q, policy and average policy networks.

```
for iteration T ← 0, T_max do
    Synchronize local on-line networks as copies of global on-line networks
    Sample initial state
    repeat
        Execute random action with probability ε, otherwise execute action
        from policy network
        Sample new state and reward
        Compute Q-network targets with target Q-network
        Compute policy networks' targets with GIGA-WoLF's update equations
        Compute loss of local on-line Q, policy, and average policy networks
        Accumulate gradients by minimizing the loss
    until terminal state
    Update global on-line networks weights with the accumulated gradients of
    local on-line networks
    Synchronize global target networks as copies of global on-line networks
    every τ time-steps
end for
```

Output: A converged Q-network to approximate the value function as $Q(s, a, \theta)$, and a converged policy network to approximate the policy function as $\pi(s, a, \theta_\pi)$.

Competitive Deep Reinforcement Learning over a Pokémon Battling Simulator
(Simoes et. al)

Deep Reinforcement Learning

Q Networks

- Reinforcement Learning is based on the Bellman Equations

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

$$q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

- Storing these mappings is difficult: Learn them with a neural network
  - State -> Q-values for each action
- Deep Q Network (DQN) – Train a NN to select actions via argmax
  - Improvements such as Replay and Target networks improve stability

Human Level Control through Deep Reinforcement Learning
(Mnih et. al)

Strategies

After integrating with environment, test:

- Algorithms:
  - DQN, and improvements (n-step learning, priority replay)
  - GIGAθ, WPL
- Embeddings and Reward:
  - Naïve: Only take into account power & hp
  - Hand Designed: Take into account stats of bench and opponent
  - Future- Reward Network
- Opponent
  - Random & MaxPower Baseline
  - Selfplay

Results

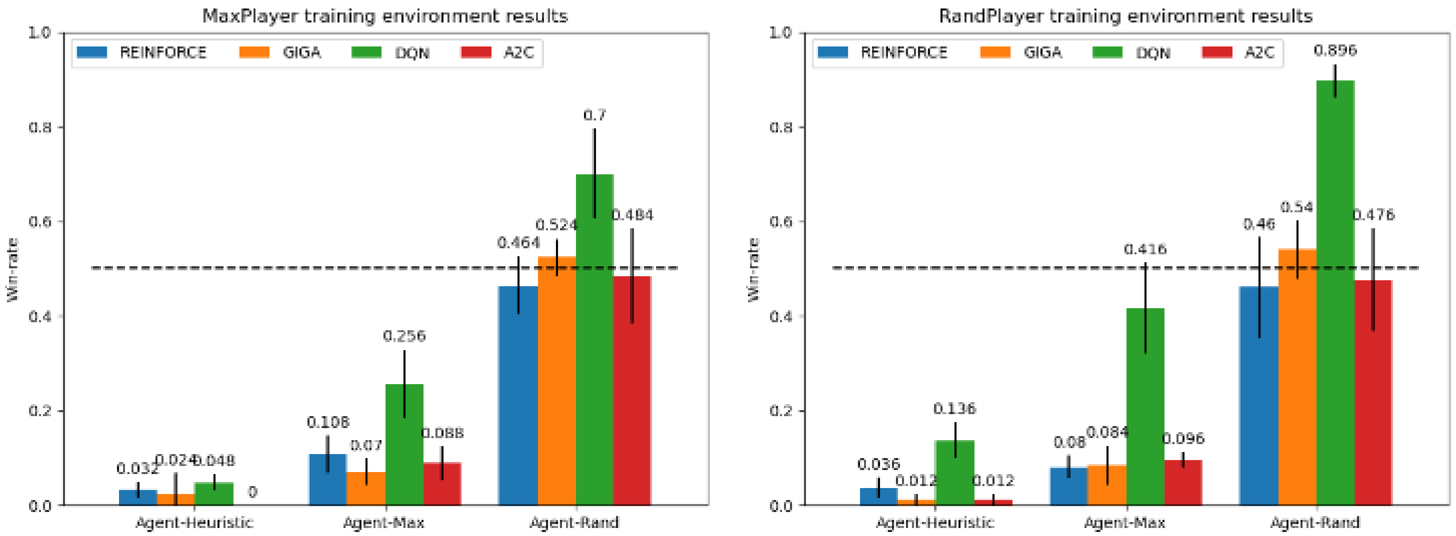
Initial Results-

- DQN with Random and MaxPower Baseline: 5 runs
  - Training over 10,000 env steps

Training against Max			Training against Rand		
Agent-Rand	Agent-Max	Agent-Heuristic	Agent-Rand	Agent-Max	Agent-Heuristic
0.9	0.38	0.1	0.58	0.2	0.02
0.94	0.4	0.12	0.82	0.22	0.04
0.84	0.44	0.14	0.76	0.38	0.06
0.9	0.56	0.12	0.64	0.26	0.06
0.9	0.3	0.2	0.7	0.22	0.06
0.896	0.416	0.136	0.7	0.256	0.048
0.035777088	0.095289034	0.038470768	0.09486833	0.072663608	0.017888544

- Beating similar student project's performance ~65% - 70% : Kalose et. al.

https://web.stanford.edu/class/aa228/reports/2018/final151.pdf



(a) Results when trained against a MaxPowerPlayer baseline.

(b) Results when trained against a RandomPlayer baseline.

Figure 5. Final win-rate results for a sample size of n=5 runs. Error bars show sample standard deviation. The x-axis corresponds with the opponent in testing, the two graphs shows the results of models trained against different baseline models. The dashed line shows a win-rate of 0.5 for approximate parity in strength.

	Win-rate		
Experiment	Agent-Rand	Agent-Max	Agent-Heuristic
MaxPower DQN	0.896 ± 0.036	0.416 ± 0.095	0.136 ± 0.0385
Heuristic DQN	0.66 ± 0.105	0.276 ± 0.056	0.072 ± 0.023
Adam DQN	0.712 ± 0.107	0.344 ± 0.0385	0.152 ± 0.048

Figure 6. Final win-rate results for a sample size of n=5 runs. Adam DQN is trained with an Adam optimizer against the MaxPower baseline agent. Errors show sample standard deviation.

Future Work

Directions for this Project

- Simple:
  - Develop Embeddings & Reward
  - Implement other algorithms/improvements (Rainbow is all you need)
  - Hyperparameter tuning
- Advanced:
  - Reward network
  - Develop novel training algorithms
    - Potentially tackle IL/BC or Few-shot

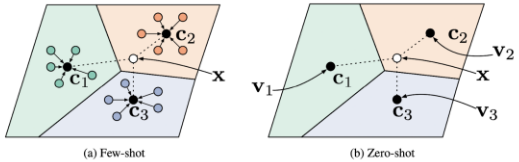


Figure 1: Prototypical Networks in the few-shot and zero-shot scenarios. **Left**: Few-shot prototypes $\mathbf{c}_k$ are computed as the mean of embedded support examples for each class. **Right**: Zero-shot prototypes $\mathbf{c}_k$ are produced by embedding class meta-data $\mathbf{v}_k$. In either case, embedded query points are classified via a softmax over distances to class prototypes: $p_\phi(y = k|\mathbf{x}) \propto \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))$.