I Choose You, Reinforcement Learning! Trained RL Agents For Pokémon Battles

Leonardo de Lellis Rossi[1,3,4], Bruno Souza[2,3,5], Maurício Pereira Lopes[2,3], Ricardo Ribeiro Gudwin[1,3,4], Esther Luna Colombini[2,3,4]

[1]Faculty of Electrical and Computer Engineering (FEEC)
[2]Institute of Computing (IC)
[3]Universidade Estadual de Campinas (Unicamp)
[4]Hub of Artificial Intelligence and Cognitive Architectures (H.IAAC)
[5]Reasoning for Complex Data (Recod.ai)

Campinas/SP – Brazil
{l261900, b234837}@dac.unicamp.br

Abstract. *Pokémon battles present a valuable training environment for Reinforcement Learning (RL) agents due to their inherent stochastic nature and adaptability to deterministic settings. However, this environment currently lacks a comprehensive benchmark of basic RL agent implementations suitable for training purposes. This project aims to fill this gap with an open-source benchmark of trained agents with classic RL methods and Deep Reinforcement Learning (DRL) techniques, to foster the development in the field and facilitate the entry of new researchers. We also propose a Markov Decision Process (MDP) environment, in which agents are trained and validated. The agents demonstrated effective learning and achieved robust performance during training.*
Keywords *Reinforcement Learning, Deep Learning, Tabular Methods, Pokémon, Benchmark.*

1. Introduction

Pokémon is a renowned franchise that involves capturing and training fictional creatures, battling in turns of attack [Nintendo. 2024]. The environment serves as an ideal testbed for RL, a machine learning technique that optimizes actions for rewards in dynamic environments [Sutton and Barto 2018]. In RL domain, tabular methods use tables to store and update values for state-action pairs to iteratively learn optimal policies in MDPs. While effective for small state spaces, they struggle with scalability in larger environments [Watkins 1989]. DRL methods address high-dimensional state spaces and complex decision-making problems [Arulkumaran et al. 2017]. This work focuses on applying diverse RL techniques within a Pokémon battle simulation framework. It serves both as a case study for understanding and applying RL in complex, dynamic environments and as a benchmarking platform of trained agents for future RL developments.[1] The proposed MDP is also available to be used for training and validation of RL agents.

[1]Open-Source Repository: **github.com/leolellisr/poke_RL**

2. Materials and Methods

This section outlines the formulation of the Pokémon Battle approach, the Pokémon Battle settings employed in the experiments, and the RL agents utilized.

2.1. MDP Formulation and Discretization Model

The Pokémon battle system operates within an MDP framework, including states (S), actions (A), a transition function (ϕ), and rewards (R). In our deterministic adaptation, moves have 100% accuracy, and no critical hits, removing stochastic transition function (ϕ) and randomness. For convenience, in this subsection, we use some abbreviations. [2]

(S) **States** represent six battle elements concatenated: *player's AP Index* [0-5], *opponent's AP Index* [0-5], *player's AM BP* (if NA, default to -1), *player's AM DM, number of player's RmP*[0-6], *number of opponent RmP* [0-6].

(A) **Actions** involve the available moves and Pokémon switches.

(R) **Rewards:** A reward r is acquired at the end of each turn in state (s) by taking the action (a). At each step, r is defined by: *Player's AP HP + Number of agent's RmP - Opponent's AP HP - Number of opponent's RmP - 2* (if player's AP fainted) *- 1* (if player's AP have a NSC) *+ 2* (if opponent's AP fainted) *+ 1* (if opponent's AP have a NSC) *+ 15* (if player won the CB) *- 15* (if player lost the CB).

2.2. Pokémon Teams and Environment

The selections aim to balance the Pokémon types between the player's teams and the opponent's teams. The Pokémon, their abilities, items, natures, moves (with effect, base power and accuracy) and possible switches are available on *teams* folder of the repository.[3] Deterministic teams have been configured to have no effects related to randomness and to have 100% accuracy. We use Pokémon Showdown, an open-source Pokémon simulator. [4] Figure 1 demonstrates a Pokémon Battle.

2.3. Agents

Our RL agents were trained against a MaxDamagePlayer (MP), which selects the move that will cause the most damage, and against a RandomPlayer (RP), which selects a random move. The RL agents we trained are divided into Tabular Agents and DRL Agents. In this subsection, we list the algorithms and their specificities in our implementations. The agents were implemented in Python language.

2.3.1 Tabular Agents: We carry out all implementation of all tabular algorithms. In each algorithm, we implemented two versions, the classic one and one with Function Approximation (FA). Classical tabular methods offer exact solutions, ideal for small state spaces. FA methods provide approximate solutions, fitter for larger state spaces. Each tabular agent has been trained for 10,000 battles.

[2]Abbreviations. **AP:** Active Pokémon, **AM:** Active Moves, **BP:** Base Power, **NA:** Not Applicable, **DM:** Damage Multipliers, **RmP:** Remaining Pokémon, **HP:** Health Points, **NSC:** Negative Status Condition and **CB:** Current Battle.

[3]Teams folder link: **github.com/leolellisr/poke_RL/tree/master/teams**

[4]Pokémon Showdown! battle simulator: **play.pokemonshowdown.com**

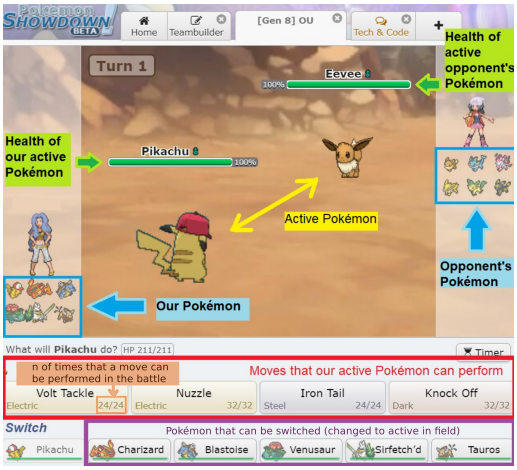

Figure 1. Example of one battle in Pokémon Showdown.

2.3.1.1 - Monte Carlo Control First-Visit (MCC): For exploration, ε-greedy strategy with $\varepsilon = \frac{N_0}{N_0+N(s)}$. **Classical:** employs a learning rate of $\alpha = \frac{1}{N(s,a)}$ and N_0 with three values: 10^{-4}, 10^{-3}, and 10^{-2}; **FA:** first, weights w and $N(s,a)$ are set to zero. The end of the battle triggers the first-time update for w and $N(s,a)$, integrating $\alpha \cdot \delta \cdot x(s,a)$ for w updates, where δ is calculated with $q_{\text{approx\_fn}}(s,a,w)$. The policies are updated with the new w and $N(s,a)$ values.

2.3.1.2 - Q-Learning: Exploration employs an ϵ-greedy strategy, where ϵ is $N_0/(N_0 + N(s))$. Three distinct N_0 values are evaluated: 10^{-4}, 10^{-3}, and 10^{-2}. **Classical:** The foundational TD update rule: $Q^{new}(S_t) \leftarrow Q(S_t) + \alpha\left(R_{t+1} + \gamma * max(Q(S_{t+1})) - Q(S_t)\right)$, where α operates promptly at the transition to S_{t+1} and receipt of reward R_{t+1}; **FA:** initialization sets a weight array w to random values and $N(s,a)$ to zero, with keys representing states s and values as arrays reflecting the action space size (9 in our case)

2.3.1.3 - SARSA(λ): Learning rate $\alpha = \frac{1}{N[s,a]}$, and $\gamma = 0.75$. An ϵ-greedy strategy is employed with $\epsilon = \frac{N_0}{N_0+N(s)}$, exploring three N_0 values and six λ values. Validation focuses on specific N_0 values 10^{-4}, 10^{-3}, 10^{-2}. **Classical:** updates considering reward (R), next state (s'), and $(E(s,a))$ and iteratively refined $(Q(s,a))$ during each turn of the battle episode; **FA:** a defaultDict $N(s,a)$ is initialized with weight storage w random, and eligibility trace e at zero.

2.3.2 DRL Agents: We implement all DRL algorithms with adaptations based on the original code of each one. DQN, Double DQN, and PPO were trained with 10k steps per epoch, totaling approximately 333 battles. Training spanned two durations, of 300,000 steps (30 epochs, approximately 10,000 battles) and 900,000 steps (90 epochs, approximately 30,000 battles). REINFORCE was trained with 10,000 and 30,000 battles. Throughout the process, the best models were saved. We conducted training over 900k steps / 30k battles to verify the continuous learning progress of the agents. We used the Adam optimizer with a Learning Rate of 2.5×10^{-4} and a discount factor (γ) of 0.75. For DQN and DDQN we use a linear annealing ϵ-greedy exploration strategy with $\epsilon = 0.1$. DQN and DDQN models include 128 neurons in the first hidden layer.

2.3.2.1 - Deep Q-Learning (DQN) [Mnih et al. 2013, Osband et al. 2016, Arulkumaran et al. 2017]: Operated "Off-Policy", the DQN agent updates Q-Values under the assumption that the best action was chosen, regardless of the action taken. The Q-Value calculation incorporates both the immediate reward and the maximum Q-Value of the next state.

2.3.2.2 - Double Deep Q-Learning (DDQN) [Osband et al. 2016, Arulkumaran et al. 2017]: If action a in state s has a higher value than action b, the secondary model is employed for Q-Value calculation. Double DQN dynamically selects between the models based on their respective action values. Our "Off-Policy" method updates Q-Values assuming the best action was chosen.

2.3.2.3 - Proximal Policy Optimization (PPO) [Schulman et al. 2017]: We use a Neural Network consisting of a two-layered multi-layer perceptron (MLP) with each layer containing 64 neurons.

2.3.2.4 - REINFORCE [Williams 1992, Zhang et al. 2021]: Updates parameters by stochastic gradient ascent. The goal is to compute $\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_d [v_{\pi_\theta}(S)]$. Applying the Policy Gradient Theorem on MCC Policy Gradient, computing above equal is equivalent to iteratively update θ as follows: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta (s_t, a_t) v_t$.

3. Results and Discussion

The graphical results are available on Neptune for Tabular and DRL methods.[5] The results of PPO are available in Github.[6] Graphical results are also available in folder *images/report* of the repository.[7] Trained models are available on a Google Drive repository.[8] We provide guidance on how we save models and how they can be loaded in the README file of our repository. Table 1 presents a performance comparison in the validation set across the Tabular and DRL agents, along some approaches found in the literature.

Tabular methods (IDs 1-18): **Classical MCC** (1-3) performed poorly against MP, struggling to adapt to aggressive strategies caused by end-of-episode reward sampling and table updating. **MCC FA** (4-6) performed better, with broad generalization and strategic Pokémon switches; **Q-Learning** (7-9) excelled effectively using Pokémon abilities. **Q-Learning FA** outperformed the tabular version, showing a deeper understanding of rewards and optimal state generalization; **SARSA(λ)** (13-15) was successful in both forms but performed slightly worse than **SARSA(λ) FA** (16-18) against stochastic RP, due to its bootstrapped nature.

DRL methods (IDs 19-26): **DQN** (21-22) performance varied with learning rate and environments; **DDQN** (19-20) improved upon the DQN in both environments; **PPO** (23-24) had high performance, showcasing strategic switches and a nuanced understanding of the reward structure; **REINFORCE** (25-26) displayed behavior similar to the PPO agent.

[5]Results Tabular (MC Control, Q-Learning, SARSA) and DRL (DQN, Double-DQN) - Neptune: **acesse.dev/nepPkRL**, Results SARSA - Neptune: **acesse.dev/nepPkRL2** and Results REINFORCE - Neptune: **acesse.dev/nepPkRL3**

[6]Results PPO - Github: **acesse.dev/git-PPO**

[7]Graphic results - Github: **encr.pw/git-imgs**

[8]Trained models - Google Drive: **acesse.one/svdMdls**

Table 1. Results for 15 methods and 31 agents. Each agent faces as opponents a MaxDamagePlayer (MP) and a RandomPlayer (RP). FA is Function Approximation. Values found in the literature are also presented. Win rates shown in percentages. Results above 80% are highlighted in green. Results below 50% are highlighted in red. Each Tabular agent was trained for 10k epochs and validated by 2k epochs. Each DRL agent was trained for 10k / 30k epochs and validated by 2k / 6k epochs.

ID		No	Stochastic			Deterministic		
		-	γ	vs. MP [%]	vs. RP [%]	γ	vs. MP [%]	vs. RP [%]
1	Monte Carlo Control	0.0001	-	37.86	90.79	-	41.40	91.57
2		0.001	-	31.89	93.40	-	40.77	90.22
3		0.01	-	34.98	93.91	-	36.03	89.92
4	Monte Carlo Control FA	0.0001	-	83.14	99.37	-	60.67	99.37
5		0.001	-	83.89	99.1	-	59.47	99.19
6		0.01	-	83.35	99.2	-	60	99.31
7	Q-Learning	0.0001	-	45.3	93.7	-	43.38	84.04
8		0.001	-	34.68	85.78	-	32.31	84.01
9		0.01	-	45.09	90.64	-	43.83	88.03
10	Q-Learning FA	0.0001	-	83.98	99.37	-	57.76	99.37
11		0.001	-	84.37	99.1	-	59.2	99.25
12		0.01	-	83.17	99.2	-	59.23	99.4
13	SARSA(λ)	0.0001	0.2	56.17	94.54	0.2	52.18	89.17
14		0.001	0	54.13	89.68	0.2	54.13	89.02
15		0.01	0	54.07	88.6	0.2	52.48	90.25
16	SARSA(λ) FA	0.0001	1	83.53	57.43	0.8	61.9	98.95
17		0.001	0.6	83.2	57.37	0.4	60.82	98.95
18		0.01	1	83.65	57.13	0.8	60.91	99.49
19	Double DQN - 300k steps	-	0.75	67	98	0.75	77.08	99.37
20	Double DQN - 900k steps	-	0.75	83.94	99.24	0.75	77.92	98.98
21	DQN - 300k steps	-	0.75	69.57	98.59	0.75	71.71	98.5
22	DQN - 900k steps	-	0.75	60.97	99.28	0.75	74.62	98.89
23	PPO - 300k steps	-	0.75	**88.93**	99.94	0.75	74.86	**99.73**
24	PPO - 900k steps	-	0.75	88.21	99.67	0.75	**82.75**	99.49
25	REINFORCE - 10k battles	-	0.75	87.52	99.43	0.75	60.16	99.13
26	REINFORCE - 30k battles	-	0.75	85.07	99.14	0.75	75.97	99.55
27	[Rill-Garcıa 2018]	0.1	-	-	58	-	-	-
28	[Kalose et al. 2018] softmax	-	-	-	68	-	-	-
29	[Kalose et al. 2018] ϵ-greedy	-	-	-	60	-	-	-
30	[Huang and Lee 2019]	-	-	85.07	99.14	-	75.97	99.55
31	GIGAΘ [Simoes et al. 2020]	-	-	-	**99.9**	-	-	-

4. Conclusion

This project establishes an open-source benchmark for RL agents trained for Pokémon battles, with classical tabular RL methods and DRL methods. By implementing it with stochastic and deterministic adaptive environments, we aim to improve accessibility and promote collaborative progress in the field. However, we acknowledge some limitations, such as potential bias introduced by fixed team compositions. The work will continue with the refine of DRL algorithms and experiments using variations in team compositions and strategies. We thank H.IAAC, MCTI/Softex, CNPq and CEPID/BRAINN. [9] We also thank our colleague Henrique L. C. Oliveira for his help in developing and training the algorithms.

[9]This project was supported by the brazilian Ministry of Science, Technology and Innovations, with resources from Law nº 8,248, of October 23, 1991, within the scope of PPI-SOFTEX, coordinated by Softex and published Arquitetura Cognitiva (Phase 3), DOU 01245.003479/2024 -10. E. L. Colombini is partially funded by CNPq PQ-2 grant (315468/2021-1), R.R. Gudwin is partially funded by CEPID/BRAINN (Proc. FAPESP 2013/07559-3), L. L. Rossi is funded by MCTI/Softex.

References

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep Reinforcement Learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Huang, D. and Lee, S. (2019). A self-play policy optimization approach to battling Pokémon. In *Proceedings of the 2019 IEEE Conference on Games (CoG)*, pages 1–4.

Kalose, A., Kaya, K., and Kim, A. (2018). Optimal Battle Strategy in Pokémon using Reinforcement Learning. *Stanford University*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with Deep Reinforcement Learning. *DeepMind Technologies*.

Nintendo. (2024). Pokémon official site. `https://www.pokemon.com`.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Rill-Garcıa, R. (2018). Reinforcement Learning for a Turn-Based Small Scale Attrition Game.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization algorithms.

Simoes, D., Reis, S., Lau, N., and Reis, L. P. (2020). Competitive Deep Reinforcement Learning over a Pokémon battling simulator. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 40–45. IEEE.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An introduction*. MIT press, Cambridge, MA.

Watkins, C. (1989). Learning From Delayed Rewards. *British Library. Thesis (Ph. D.)*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist Reinforcement Learning. *Machine learning*, 8:229–256.

Zhang, J., Kim, J., O'Donoghue, B., and Boyd, S. (2021). Sample efficient Reinforcement Learning with REINFORCE. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10887–10895.