Playing Pokémon Red with Reinforcement Learning

2021-03-06

Joseph Flaherty, Aaron Jimenez, Bahareh Abbasi

California State University Channel Islands

Playing Pokémon Red with Reinforcement Learning

Joseph Flaherty *, Aaron Jimenez *, Bahareh Abbasi

California State University Channel Islands, Camarillo, United States

Abstract

Through Reinforcement Learning (RL), superhuman performance has been achieved in every Atari 2600 game available in OpenAI's Gym library. To this day, however, no reinforcement learning agent has been capable of completing Pokémon Red *despite its massive popularity. Between the game's over 25-hour length, contrasting overworld and battle systems, randomness, and non-linear gameplay,* Pokémon Red *is undoubtedly a daunting task for current RL algorithms. In this paper, we investigate how to employ RL techniques to develop a game AI agent for playing* Pokémon Red. *To best tackle this challenge, we conducted two experiments. First, we compared the performance of two RL algorithms: the Advantage-Actor Critic (A2C) and Deep Q-Networks (DQN). Next, we looked into whether to use image or RAM observations for training our agent. We found that A2C outperformed DQN due to its ability to run far more timesteps of training in the same amount of time. We also found that image observations surpassed RAM observations despite taking longer per timestep, thanks to the increased effectiveness of training.*

1. Introduction

A great amount of attention has been given to the field of Artificial Intelligence (AI) in media. In particular, the terms *"Machine Learning"* and *"Reinforcement Learning"* are thrown around a great deal; however, context is not usually given as to the actual meaning of these words. In essence, the idea behind Machine Learning is to create AI systems that can learn to do a specific task without the need to explicitly program said behavior into your system. Reinforcement Learning is a sub-field of Machine Learning in which an AI agent learns to perform a specific task through feedback from repeated trials and errors, much like how humans learn.

Reinforcement Learning has been extensively used for policy extraction in many Atari games such as Breakout, Space Invaders, Montezuma's Revenge, Private Eye, and etc. [1–4]. Through Reinforcement Learning, Agent57 [5] has surpassed the human benchmark for every Atari 2600 game in OpenAI's Gym library [6]. Role-Playing Games (RPGs) are widely appreciated but are seldom the subject of Artificial Intelligence (AI) research due to their inherent challenge. The most similar previous work is by Kalose et al. [7] which uses RL to find an optimal strategy for *Pokémon* battles. While their work focuses on a more discrete sub-problem in the game, our goal is to create an agent that is proficient in all aspects of the game.

While most Atari 2600 games offer simplistic "easy to learn, difficult to master" experiences, *Pokémon Red* [8] instead presents a marathon of assorted challenges. The game's massive and non-linear world, variance through randomness, hidden key items, and various puzzles together form a game that takes average players over 25 hours to complete. With such a massive state space, a large amount of training is required. To make the process of training as time-effective as possible, several design aspects of the agent must be examined.

The goal of this study is to analyze and compare the most effective methods for training an AI agent to play in a complex game environment, such as *Pokémon Red*, using free and open-source tools. This goal is founded upon the idea that the knowledge gained from this study could be applied to further the uses of reinforcement learning not only in other challenging games but also in fields such as robotics, where the environments are incredibly complex and contain a large degree of randomness.

This paper is organized as follows: Section 2 provides an introduction of *Pokémon Red*. Next, an overview of the Reinforcement Learning algorithms used in this work is given in Section 3. The setups for the experiments and their results are presented in Section 4 and Section 5 respectively. Finally, we conclude the paper in Section 6.

*First two authors contributed equally to this work.

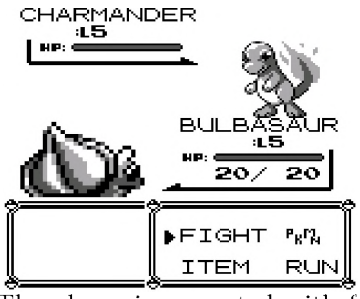

Figure 1: The player is presented with four options in a pokémon battle. They can use an attack, switch to another pokémon, use an item (such as a potion to restore HP), or attempt to flee from the battle.

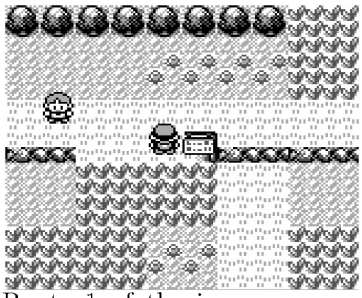
Figure 2: Route 1 of the in-game overworld. The player navigates this overworld throughout the game to reach their next challenge.

2. Basics of Pokémon Red

To beat *Pokémon Red*, a player must defeat *The Elite Four* and the *Champion*, in a series of back-to-back pokémon battles. A pokémon battle is won by using your own pokémon's moves to reduce all of the opponent's pokémon's hit points (HP), a measure of a pokémon's health, to zero. An example battle is shown in Fig. 1. Before the player is allowed to challenge the Elite Four, they must travel across numerous cities and routes within the in-game overworld, as shown in Fig. 2, and collect eight gym badges. A gym badge is earned by defeating a gym leader in a pokémon battle, usually preceded by other trainers and a puzzle.

In order to reach each gym, the player explores various areas, encountering new pokémon along the way. To get stronger, players can catch new pokémon to add them to their party and level up their pokémon through battles. On their quest to defeat the *Elite Four* at the *Pokémon League*, the player must also stop a group of villains known as *Team Rocket*.

3. Reinforcement Learning

Reinforcement Learning algorithms learn to solve complex problems through repeated experimentation. To accomplish this, several components are needed to replace the extensive data sets often used by other approaches. The most central component of Reinforcement Learning is the agent, which examines the current state and decides what to do in response. Thus, the agent takes the place of a human player. The agent operates within an environment that represents the challenge the agent is tasked with. The environment, in this case, is *Pokémon Red*. The agent can influence the environment through *actions*. The *Pokémon Red* agent has the following actions: Up, Down, Left, Right, A, B, Start, Select, and None. Whenever an action is taken, the agent receives a reward that helps lead the agent in the right direction. Finally, to produce the output of an action, the agent uses the observations (feedback) from the environment.The observation is an abstract representation of the environment which conveys the current state to the agent.

Two of the most popular RL algorithms are Deep Q-Networks (DQN) and Advantage Actor-Critic (A2C). In our work, we examine both for their performance within the environment of *Pokémon Red*.

3.1. Deep Q-Networks (DQN)

DQN is an extension of the Q-Learning algorithm [9] which operates on many of the same core principles. The core of Q-learning is the Q-values ($Q(s,a)$), which represent the expected utility of a given state and action pairing. As the agent experiences the environment in training, the following equation is used to update Q-values:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a} Q(s',a) - Q(s,a)]$$

where s is the current state, a is the action, α is the learning rate which adjusts the importance of new updates, R is the reward function, s' is the next state, and γ is the discount factor which alters the weight given to future rewards. While relatively simple problems can perform well with Q-learning by storing Q-values in a table, this solution does not scale well. In a table of Q-values, an entry is needed for every state. When considering the state space of a game such as *Pokémon Red*, there are simply too many possible states to store in any table. To solve this, function approximation can be used, which is done most frequently through the use of Deep Neural Networks, resulting in a DQN.

Although DQNs have seen some experimentation with asynchronous variants, such as with the work of Mnih et al. [10], it is not commonly used in tandem with multiprocessing. As such, the Stable Baselines library does not offer support of multiprocessing with the DQN algorithm, and only singular environments are compatible [11].

3.2. Advantage Actor-Critic (A2C)

In actor-critic algorithms, there are two key functions used to facilitate learning. The actor makes decisions based on its current policy function which it updates as it receives feedback from the critic. The critic attempts to improve the agent by giving feedback in line with the value function that it approximates. One issue with a Q actor-critic agent is the variability that Q-values can have. A2C addresses this by using advantage instead of raw Q-values. Advantage is the Q-value of a state-action pair subtracted by the value of the state. In other words:

$$A(s, a) = Q(s, a) - V(s)$$

where Q is the Q-value of a given state-action pair and V is the value function. Advantage is, in essence, a measure of how much better an action is in a certain situation, in the long run than randomly choosing any of the available actions. A2C was preceded by an asynchronous variant of the algorithm, A3C as seen in Mnih et al. [10]. This version featured several "workers" that worked separately on multiple environments to form one policy. As detailed in OpenAI's research blog [12], it was ultimately determined that a synchronous variant of A3C performed better overall, noting that it more effectively took advantage of GPUs by generating larger batch sizes.

3.3. Observation Comparisons

Well-thought-out input is pivotal in simplifying the problem for an agent. Ideally, the agent should receive enough information to make an optimal decision, but not more than that to avoid making the problem more difficult with extraneous data. Within the Gym Retro library that the *Pokémon Red* environment relies upon, there are two main options when it comes to observations: image observations and RAM observations [13]. Image observations take the form of the rendered screen output of the game as an RGB image. This is easily enough information to beat the game, since it is what is shown to human players, but may clutter the agent with superfluous, decorative information. In contrast, RAM observations are merely the memory of the game, which sidestep having to recognize onscreen values but at the cost of not having the information visualized as originally intended.

4. Experiments

To determine the best setup to achieve maximal performance with our agent, we devised two experiments. The first experiment (Experiment 1) aims to determine whether A2C or DQN will provide better performance for the task of *Pokémon Red*. The second experiment (Experiment 2) looks into whether image observations perform better than RAM observations. With these two questions answered, the agent will be able to train much more efficiently. Since the observation tests still need an algorithm for their comparison, A2C was used for both. Similarly, the RAM was chosen as the observation method for Experiment 1.

4.1. Experimental Setup

For our experiments, the game runs within a Gym environment [14] using OpenAI's Gym Retro library. To run *Pokémon Red*, a GameBoy hardware emulator is utilized with the Gym environment created. By default, Gym Retro supports GameBoy games, however, it does not have an integration for *Pokémon Red*. A custom integration was created to enable functionality for the game within Gym Retro. To give the agent access to the necessary information about the game, memory addresses of variables in the GameBoy emulator's system memory [15] are enumerated in a JSON file. These values are then utilized in a custom Lua script [16] to define the reward functions and end state conditions for the environment. To avoid unnecessary hassle with the main menu, the agent begins the game from a save state that is set to when the player is first actionable.

Rather than writing everything from scratch, the algorithms used are from Stable Baselines [17], a fork of OpenAI's Baselines repository [18]. The decision to use Stable Baselines over OpenAI's Baselines revolves around its superior documentation, well-commented code, TensorBoard support, and ease of use.

In each experiment, the performance is evaluated by running the model in an evaluation script, producing a cumulative reward [19]. The aforementioned Lua script holds the reward function for the agent which is pivotal in this evaluation. The reward function differs when the agent is in the overworld versus when the agent is in a battle. This is determined by reading which audio bank the game is currently using to access music, as in-battle and overworld music use different audio banks. The overworld portion rewards factors such as not standing in the same spot, reaching a new world map, and the number of pokémon in the agent's party. The battle portion of the reward function focuses on a ratio of the agent's pokémon's HP to the opponent's as well as a level difference.

When creating the models for all of our experiments, we used an MLP (Multilayer Perceptron) network, [20] trained for a single continuous episode. The values of the hyperparameters were set to the default

Table 1: Comparison of time needed for A2C and DQN to complete a certain amount of timesteps of training and their corresponding ratios.

Timesteps	A2C	DQN	Ratio
1,000	1.0500s	2.1000s	2
5,000	3.5650s	120.92s	33.920
10,000	7.0078s	257.11s	36.688
Average Ratio (exclude 1k)			35.304

values provided by Stable Baselines for both the DQN [11] and A2C [21] algorithms.

4.2. Evaluation

Two key elements to choosing the best algorithm and observation method are effectiveness and efficiency. The algorithm and the observation method need to be effective in consistently improving performance as training goes on. They should also be efficient, as this allows more training to take place.

To put the two algorithms and observation methods on an even playing field for efficiency, first a test was performed to examine the amount of time that each needed to complete training for an arbitrary amount of timesteps. Between three tests, a ratio of time was calculated to relate how much time was needed to complete the number of time steps in relation to each other. Once these tests are completed, an approximate average ratio will be used to determine a *fair* number of timesteps for each algorithm and observation method, resulting in a similar amount of training time for both.

Once the models are trained for their respective number of fair timesteps, it is saved. This saved model is evaluated for 100,000 timesteps, producing a cumulative reward and a reward per step. These measures are then contrasted to determine if one algorithm is superior to the other in the context of *Pokémon Red*. Reward was chosen as the primary evaluation metric since it is the best predictor of the agent's competency.

5. Results and Discussion

In this section, we present our results for both experiments.

5.1. Experiment 1: A2C Vs. DQN

The result of the time comparisons between A2C and DQN is provided in Tab. 1. The 1,000 timestep test was deemed to be an outlier in comparison with the other two tests, so it was excluded from the calculation of the average ratio. This helped to yield a more fair result. The average ratio indicates that A2C can run approximately 35.304 times more timesteps in a given time period. This is thanks to the ability to use vectorized environments with the Stable Baselines implementation of A2C. This ratio also means that DQN needs to be much more effective per timestep in order to outperform A2C.

Table 2: Comparison of time needed for models with Image and RAM observations to train for a certain amount of timesteps and their corresponding ratios.

Timesteps	IMG	RAM	Ratio
10,000	14.400s	7.6808s	1.8748
50,000	66.489s	36.352s	1.8290
100,000	117.45s	63.892s	1.8382
Average Ratio			1.8473

Table 3: Comparison of the performance of models trained using the A2C and DQN algorithms.

RESULTS	A2C	DQN
Training Time	219.71s	244.73s
Cumulative reward	-24348105.68	-101262330.19
Reward per step	-243.48	-1012.62

DQN was arbitrarily given 10,000 timesteps for the final test. Based on the average ratio, A2C was given 350,000 timesteps of training for its final model. As seen in Tab. 3, the A2C model performs much better than the DQN model. Ultimately, the efficiency of A2C is enough to separate it from DQN by a wide margin. This is also despite the fact that A2C ended up being trained for less time than DQN. Although neither model performs particularly well due to having a limited amount of training time, it is clear that A2C will achieve a competent agent much quicker than DQN. Hence, it can be concluded that A2C is preferable with this setup for *Pokémon Red*. The sheer number of timesteps that A2C can train allows it to demonstrate excellent performance.

Table 4: Comparison of the performance of models trained with image-based observations and RAM-based observations.

RESULTS	IMAGE	RAM
Training Time	299.31s	296.18s
Cumulative reward	–7558272.25	-94121697.33
Reward per step	-75.58	-941.21

5.2. Experiment 2: RAM Vs. Image

The results of the time comparison between the Image and RAM Observations are displayed in Tab. 2. An average ratio was calculated to create a fair amount of timesteps for each observation type to run for a similar amount of time. The average ratio shown in Tab. 2 shows that the RAM observation model trains almost twice as fast as models with image observations.

An arbitrary number of 250,000 timesteps was chosen for an image-based model to train for the final test. To create a fair comparison, the RAM-based observation model was chosen to run for 462,500 timesteps, based on the aforementioned ratio.

It may appear that RAM observations perform better than image observations solely based on their training speed, however, this is not the only metric to consider. Surprisingly, the model trained with image observations had a far greater real-world performance compared to RAM. Despite having less efficient training, image observations were much more effective. As seen in Tab. 4, the image-based model ended up obtaining a cumulative reward nearly ten times better than the RAM-based model.

6. Conclusion

To achieve the best possible performance from a Reinforcement Learning-based *Pokémon Red* agent, we have experimented with two key factors in the design: the algorithm and the observation type. Our results suggest that for the best performance, an agent is best off using A2C and image observations. A2C was able to outperform DQN through its superior efficiency, allowing it to train for far more timesteps in the same amount of time. Conversely, image observations were able to outperform RAM observations because of their effectiveness. Despite each timestep taking nearly twice as long, the image observations allow the agent to perform better by improving faster on a per-timestep basis.

The use of A2C and image observations with our agent has enabled us to maximize the value of training. As a result, we have quickly been able to create an agent that is capable of acquiring its first *pokémon*, defeating its rival in a *pokémon* battle, and begin travelling towards the next town. Moreover, this work provides a strong foundation for further work towards an agent that can defeat *The Elite Four*.

These results are relevant to anyone looking into RL tasks involving RPG-style games. Furthermore, in any task that involves a long series of challenges or randomness, these results will also apply to help achieve the most efficient and effective training.

References

[1] Hosu I, Rebedea T. Playing atari games with deep reinforcement learning and human checkpoint replay. CoRR 2016;abs/1607.05077. URL http://arxiv.org/abs/1607.05077.

[2] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing atari with deep reinforcement learning. arXiv preprint arXiv13125602 2013;.

[3] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. Human-level control through deep reinforcement learning. nature 2015; 518(7540):529–533.

[4] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30. 2016; .

[5] Badia AP, Piot B, Kapturowski S, Sprechmann P, Vitvitskyi A, Guo D, Blundell C. Agent57: Outperforming the atari human benchmark, 2020.

[6] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W. Openai gym, 2016.

[7] Kalose A, Kaya K, Kim A. Optimal battle strategy in pokemon using reinforcement learning. Web httpsweb stanford educlassaa228reports2018final151 pdf 2018;.

[8] https://bulbapedia.bulbagarden.net/wiki/Pok%C3%A9mon_Red_and_Blue_Versions.

[9] Watkins CJ, Dayan P. Q-learning. Machine learning 1992;8(3-4):279–292.

[10] Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In International conference on machine learning. 2016; 1928–1937.

[11] https://stable-baselines.readthedocs.io/en/master/modules/dqn.html.

[12] https://openai.com/blog/baselines-acktr-a2c/.

[13] https://retro.readthedocs.io/en/latest/python.html.

[14] https://openai.com/blog/gym-retro/.

[15] http://datacrystal.romhacking.net/wiki/PokÃ©mon_Red/Blue:RAM_map,.

[16] https://git.io/JtL07,.

[17] https://stable-baselines.readthedocs.io/en/master/guide/algos.html.

[18] Dhariwal P, Hesse C, Klimov O, Nichol A, Plappert M, Radford A, Schulman J, Sidor S, Wu Y, Zhokhov P. Openai baselines. https://github.com/openai/baselines, 2017.

[19] Flaherty J, Jimenez A. Pokemon-red-ai. https://github.com/JFlaherty347/Pokemon-Red-AI, 2021.

[20] Noriega. Multilayer perceptron tutorial ;.

[21] https://stable-baselines.readthedocs.io/en/master/modules/a2c.html.