Reinforcement Learning in Strategy-Based and Atari Games: A Review of Google DeepMind's Innovations

Abdelrhman Shaheen
Computer Science Engineering Undergraduate Student
Egypt Japan University of Science and Technology
Alexandria, Egypt
abdelrhman.shaheen@ejust.edu.eg

Anas Badr
Computer Science Engineering Undergraduate Student
Egypt Japan University of Science and Technology
Alexandria, Egypt
anas.badr@ejust.edu.eg

Ali Abohendy
Computer Science Engineering Undergraduate Student
Egypt Japan University of Science and Technology
Alexandria, Egypt
ali.abohendy@ejust.edu.eg

Hatem Alsaadawy
Computer Science Engineering Undergraduate Student
Egypt Japan University of Science and Technology
Alexandria, Egypt
hatem.alsaadawy@ejust.edu.eg

Nadine Alsayad
Computer Science Engineering Undergraduate Student
Egypt Japan University of Science and Technology
Alexandria, Egypt
nadine.alsayad@ejust.edu.eg

***Abstract*—Abstract**

***Index Terms*—Deep Reinforcement Learning, Google DeepMind, AlphaGo, AlphaGo Zero, MuZero, Atari Games, Go, Chess, Shogi,**

I. Introduction

Introduction

II. Background

Reinforcement Learning (RL) is a key area of machine learning that focuses on learning through interaction with the environment. In RL, an agent takes actions (A) in specific states (S) with the goal of maximizing the rewards (R) received from the environment. The foundations of RL can be traced back to 1911, when Thorndike introduced the Law of Effect, suggesting that actions leading to favorable outcomes are more likely to be repeated, while those causing discomfort are less likely to recur [1].

RL emulates the human learning process of trial and error. The agent receives positive rewards for beneficial actions and negative rewards for detrimental ones, enabling it to refine its policy function—a strategy that dictates the best action to take in each state. That's said, for a give agent in state u, if it takes action u, then the immediate reward r can be modeled as $r(x, u) = \mathbb{E}[r_t \mid x = x_{t-1}, u = u_{t-1}]$.

So for a full episode of T steps, the cumulative reward R can be modeled as $R = \sum_{t=1}^{T} r_t$.

A. Markov Decision Process (MDP)

In reinforcement learning, the environment is often modeled as a **Markov Decision Process (MDP)**, which is defined as a tuple (S, A, P, R, γ), where:

- S is the set of states,
- A is the set of actions,
- P is the transition probability function,
- R is the reward function, and
- γ is the discount factor.

The MDP framework is grounded in **sequential decision-making**, where the agent makes decisions at each time step based on its current state. This process adheres to the **Markov property**, which asserts that the future state and reward depend only on the present state and action, not on the history of past states and actions.

Formally, the Markov property is represented by:

$$P(s' \mid s, a) = \mathbb{P}[s_{t+1} = s' \mid s_t = s, a_t = a]$$

which denotes the probability of transitioning from state s to state s' when action a is taken.

The reward function R is similarly defined as:

$$R(s, a) = \mathbb{E}[r_t \mid s_{t-1} = s, a_{t-1} = a]$$

which represents the expected reward received after taking action a in state S.

B. Policy and Value Functions

In reinforcement learning, an agent's goal is to find the optimal policy that the agent should follow to maximize cumulative rewards over time. To facilitate this process, we need to quantify the desirability of a given state, which is done through the **value function** $V(s)$. Value function estimates the expected cumulative reward an agent will receive starting from state s and continuing thereafter. In essence, the value function reflects how beneficial it is to be in a particular state, guiding the agent's decision-making process. The value function is defined as:

$$V(s) = \mathbb{E}[G_t \mid s_t = s]$$

where G_t is the cumulative reward from time step t onwards. From here we can define the **action-value function** under policy π $Q_\pi(s, a)$, which again, estimates the expected cumulative reward from the state s and taking action a and then following policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a]$$
$$= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s, a_t = a]$$

where γ is the discount factor, which is a decimal value between 0 and 1 that detemines how much we care about immediate rewards versus future reward rewards [2].

C. Reinforcement Learning Algorithms

There are multiple reinforcement learning algorithms that have been developed that falls under a lot of categories. But, for the sake of this review, we will focus on the following algorithms that have been used by the Google DeepMind team in their reinforcement learning models.

*1) **Monte Carlo Algorithm**:* The Monte Carlo Algorithm is a model-free reinforcement learning algorithm used to estimate the value of states or state-action pairs under a given policy by averaging the returns of multiple episodes. This method alternates between two main steps: policy evaluation and policy improvement.

- **Policy Evaluation:** The algorithm evaluates the value of a state or state-action pair by averaging the returns from multiple episodes that include that state or state-action pair.
- **Policy Improvement:** The algorithm improves the policy by selecting the action that maximizes the value of the state-action pair.

Since the algorithm is model-free and does not assume any prior knowledge of the environment's dynamics, it focuses on estimating state-action pair values ($Q(s, a)$) rather than just state values. The Monte Carlo Algorithm typically follows these steps:

1) Initialize the state-action pair values $Q(s, a)$ arbitrarily.
2) Start at a random state and take a random action, ensuring the possibility of visiting all state-action pairs over time.
3) Follow the current policy π until a terminal state is reached.
4) Update the value of each state-action pair $Q(s, a)$ encountered in the episode using the observed returns.
5) For each state visited in the episode, update the policy $\pi(s)$ to select the action that maximizes $Q(s, a)$:

$$\pi(s) = \arg\max_a Q(s, a).$$

This algorithm is particularly well-suited for environments that are *episodic*, where each episode ends in a terminal state after a finite number of steps.

III. AlphaGo

AlphaGo

IV. AlphaGo Zero

AlphaGo Zero

V. MuZero

MuZero

VI. Advancements

Advancments

VII. Challenges and Future Directions

Challenges and Future Directions

VIII. Conclusion

Conclusion

Acknowledgment

References

Please number citations consecutively within brackets [**?**]. The sentence punctuation follows the bracket [**?**]. Refer simply to the reference number, as in [**?**]—do not use "Ref. [**?**]" or "reference [**?**]" except at the beginning of a sentence: "Reference [**?**] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be

cited as “unpublished” [**?**]. Papers that have been accepted for publication should be cited as “in press” [**?**]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [**?**].

REFERENCES

[1] L. Thorndike and D. Bruce, Animal Intelligence. Routledge, 2017.

[2] R. S. Sutton and A. Barto, Reinforcement learning : an introduction. Cambridge, Ma ; London: The Mit Press, 2018.

[3] A. Kumar Shakya, G. Pillai, and S. Chakrabarty, “Reinforcement Learning Algorithms: A brief survey,” Expert Systems with Applications, vol. 231, p. 120495, May 2023