Reinforcement Learning in Videogames

David Fuentes Insa

13 d'abril de 2025

Resum– Resum del projecte, màxim 10 línies. ....
.. ..
... .
....
....
....
....
....
....
....
....

Paraules clau– Paraules clau del treball, màxim 2 línies . ...
....

Abstract– Versió en anglès del resum . ...
....
....
....
....
....
....
....
....
....
....

Keywords– Versió en anglès de les paraules clau. ....
....

1 INTRODUCTION

REINFORCEMENT learning is one of the most promising approaches of artificial intelligence, especially in the videogames sector. Meanwhile other AI approaches use supervised training with labeled data, RL agents learn by interacting with the enviroment throught trial and error, and improve based on rewards or penalties for taking actions.

• E-mail de contacte: david.fuentesinsa@gmail.com
• Menció realitzada: Computació
• Treball tutoritzat per: Jordi Casas Roma
• Curs 2024/25

The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Inthe most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics —trial-and-error search and delayed reward— are the two most important distinguishing features of reinforcement learning [1].

In 2013, the company DeepMind released a paper regarding how their RL models could surpass a human expert playing Atari 2600 classic games. The model beat the human expert in 3 of 6 games tested [3]. In 2015, they released a more extended paper tested on 49 games. The model got to the level of a human professional [4]. The most impressive archievement is AlphaGo, the model who beat the

European Go champion Fan Hui by 5 games to 0 [5].

Videogames provide an ideal enviroment to test the models because of the clear objectives and controlled actions. However, reinforcement learning applications extend not only in videogames, also in real life problems such as robotics, autonomous vehicles, healthcare, resource management, and much more.

2 Objectives

The main objective of this project is fully understanding Single-agent Reinforcement Learning and what can be acomplished with it using existing libraries but also how it works from zero. The next five objectives define the goal by the end of the project.

1. Learn RL Fundamentals: Study core RL concepts such as Markov Decision Processes (MDPs), value functions, and policy optimization. Implement simple algorithms like Monte Carlo or Dynamic Programming to build a solid base.

2. Learn RL Advanced Methods: Such as Temporal Difference learning (Q-learning and SARSA) and deep RL approaches like DQNs.

3. Implementing my RL model from zero: Without using libraries.

4. Experiment with self-play training in a videogame: Using a ping pong enviroment, train the agents from both sides of the game.

5. Analyze and Compare every model: With empiric data.

3 Methodology and Planning

The methodology chosen is Kanban, an agile approach that has a clear visualization of the workflow, the evolution is continuous and is very flexible.

The tool used for the implementation of Kanban is Clickup [2], a free tool for project management.

The planning is divided in five phases of three or four weeks each.

- Phase 1: Reinforcement learning fundamentals and the State of Art
  - Task 1: Research and document core RL concepts (agents, environments, states, actions, rewards)
  - Task 2: Study tabular methods (Dynamic Programming, Monte Carlo, Q-learning, SARSA)
  - Task 3: Study non-tabular methods (DQN)
  - Task 4: Document the State of Art
- Phase 2: Enviroment setup and algorithms
  - Task 5: Study gymnasium library
  - Task 6: Implement non-tabular methods
  - Task 7: Implement tabular methods
  - Task 8: Document the implementations
- Phase 3: Implement RL from zero
  - Task 9: Select the algorithm to implement
  - Task 10: Program from zero the algorithm
  - Task 11: Test the algorithm
  - Task 12: Document the algorithm
- Phase 4: Test the algorithms in enviroments and compare
  - Task 13: Select and implement the enviroments for testing
  - Task 14: Test the algorithms in the enviroments
  - Task 15: Train a self-play model in an Atari game enviroment based in images
  - Task 16: Compare empirically all the algorithms
- Phase 5: Final inform and conclusions
  - Task 17: Document the conclusions
  - Task 18: Final inform
  - Task 19: Project presentation

4 State of Art

4.1 Reinforcement Learning Fundamentals

These are the basic components a reinforcement learning problem has.

An agent — it's the entity that makes decisions. It's objective is to learn a policy ($\pi(a|s)$) that maximizes the accumulated reward. It can be a robot, a player in a videogame, etc. Enviroment — responds to the agent actions. It provides the states and the rewards, and changes based on the agent actions. States — Represent the actual enviroment situation by the perspective of the agent. Actions — The possible decissions an agent can do in a determinate state s. Choosing an action makes a transition to a new state. Rewards — Numeric values that indicate the immediate utility of an action. The agent must maximize the rewards at long term. An immediate reward r_t is a reward obtained in the time t. The return G_t, is the future rewards sum with a discount factor γ, which determines how much the agent values future rewards. If its near 0, the agent only cares for immediate rewards. If it's near 1, the agent will appreciate much more the future rewards.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Politic — Defines the agent behavior. It's the strategy the agent follows. $\pi(a|s) = P(A_t = a \mid S_t = s)$

What would these components be in a reinforcement learning problem applied in the videogame of Space Invaders? The agent, is the spaceship that moves horizontally and shoots the aliens. The enviroment, would be the aliens, the visual scenary, the alien bullets and all the game interface like lives, points. The state could include: position of the agent spaceship, position of all the aliens, position of every bullet, the lives remaining and the image frame. The actions: move left, move right, shoot, doing nothing. The rewads: +1 for shooting an enemy, -1 for loosing a live and 0 for not doing anything for example. The politic: If an enemy is just up the agent, shoot.

4.2 Markov Decision Process (MDP)

A Markov Decision Process is the math model that formalizes a RL problem. It is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ is the set of all possible actions, $P(s'|s,a)$ is the probability of transitioning to state s' when taking action a in state s, $R(s,a)$ is the expected immediate reward received after taking action a in state s, and $\gamma \in [0,1]$ is the discount factor.

The Markov property states that the probability of transitioning to the next state depends only and only on the current state and action, not on the sequence of previous states and actions:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots)$$

4.3 Tabular methods

Ideal when the enviroment is small and manageable. State-action values can be stored in tables.

4.3.1 Dynamic Programming

Based on a simple idea, if we know exactly how the enviroment works (what will happen with a probability when we take an action in a specific state), we can calculate which decisions are better in long term. This is very limited, because in the real world we don't know complete knowledge of the enviroment [6].

There are three algorithms used in DP. Policy evaluation: Calculates how good is the current policy using Bellman equation in every state using that specific policy.Policy iteration: Firstly uses policy evaluation and then improves the current policy choosing the best actions based on the calculated values.Values iteration: This method updates the values of every state with a simplified Bellman equation, to get the optimal policy.

4.3.2 Monte Carlo

Does not need to know the model of the enviroment. Learns based on experience.

The monte carlo algorithms works like this: The agent follows a politic [7]. Plays an episode till the end. At the end of the episode, calculates how much has won in total from every state. Updates the value estimates. Improves the politic, to take better actions in the next episodes

What is an episode? A sequence of states starting from one state to the end state when a condition is met.

There are two main objectives in Monte Carlo algorithms. Prediction: Follow a specified policy and evaluate how good the policy is. Control: To learn the best actions the agent can do, keep adjusting the policy.

And there are two approaches on how to update what the agent learnt. First-visit: Updates the value of the state only the first time it appears on an episode. Every-visit: Updates the value of state every time it appears

4.3.3 Temporal Difference

Learn from previous experience and don't need to end an episode like Monte Carlo algorithms. They are ideal for tasks that do not have an end (like mantaining the temperature of an oven that is always on, or to move a character in a videogame on an open world without a clear objective). Ccombines ideas from Monte Carlo and Dynamic Programming [8].

TD algorithms update the value of a state based on future estimates. When the agent changes the state and get a reward, the state value gets immediately updated

There are two main algorithms based on TD: SARSA — means State-Action-Reward-State-Action. To learn, must know the state, the chosen action, the reward obtained, the next state and the next chosen action. Learns from what you are doing, even if that is not the best possible. $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$. Q-learning — is almost the same, but learns from the best action possible. $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

4.4 Non-Tabular methods

Methods where every action-state value cannot be stored on a table because it will be infinite or impossible to manage.

4.4.1 DQN

Deep Q-Network (DQN) combines Q-learning with deep neural networks [4]. The q table is replaced for a neural network, that can estimate the q values.

How do DQN Networks work? You send the actual state to the network input (for example, the game image or the needed variables) and the network outputs the estimate q-values for every action possible. Then, the greatest q-value is chosen and executed, and get the new state and the reward. Finally, the DQN Network is updated.

REFERÈNCIES

[1] Sutton & Barto (2018). Reinforcement learning: an introduction.

[2] https://clickup.com/about

[3] Mnih, Volodymyr and Kavukcuoglu, Koray and Silver, David and Graves, Alex and Antonoglou, Ioannis and Wierstra, Daan and Riedmiller, Martin. Playing Atari with Deep Reinforcement Learning. (2013).

[4] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015).

[5] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016).

[6] Bellman, R. (1957). A Markovian decision process. Journal of Mathematics and Mechanics, 6(5), 679–684.

[7] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. IBM Journal of Research and Development, 3(3), 210–229.

[8] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine Learning, 3, 9–44.

[9] Rummery, G. A., & Niranjan, M. (1994). On-line Q-learning using connectionist systems (Tech. Rep. No. CUED/F-INFENG/TR 166). Cambridge University Engineering Department.

[10] Watkins, C. J. C. H. (1989). Learning from Delayed Rewards (PhD thesis). University of Cambridge.