Multi-Agent Reinforcement Learning Project

Anirudh Tikoo 20043
Suchetan GU 21342
Manish Gayen 21161

December 2024

1 Introduction

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take but instead must discover which actions yield the most reward by trying them. In the 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.

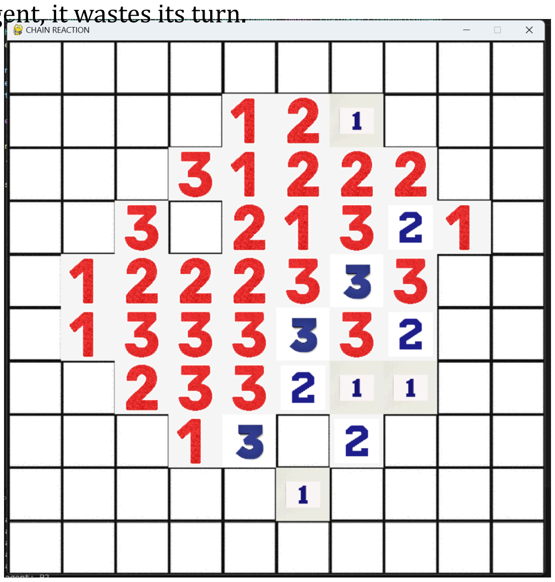
Reinforcement learning (RL) is a type of machine learning process that focuses on autonomous agents' decision-making. An autonomous agent is any system that can make decisions and act in response to its environment independent of direct instruction by a human user. Robots and self-driving cars are examples of autonomous agents. In reinforcement learning, an autonomous agent learns to perform a task by trial and error in the absence of any guidance from a human user. It particularly addresses sequential decision-making problems in uncertain environments and shows promise in artificial intelligence development. Multi-Agent Reinforcement Learning (MARL) is a subfield of reinforcement learning that studies how multiple agents interact in a common environment. When these agents interact with the environment and one another, can we observe them collaborate, coordinate, compete, or collectively learn to accomplish a particular task. It can be further broken down into three broad categories: Cooperative, Competitive, and a mixture of both.

2 Problem statement

Teaching autonomous agents to play chain reaction games through multi-agent reinforcement learning algorithms.

3 Environment

The chain reaction environment consists of a *5x5 board*. 2 agents are competing against each other to remove all opponent pieces from the board. Each agent can place their pieces anywhere on the board except for the places on the board occupied by the opponent player. A player can stack up their pieces on the board. When the square on the board reaches a critical mass, the pieces of the square explode in such a way that they can occupy all the nearby places at once. If the place is already occupied by itself, it adds up 1 more to that square, or if an opponent occupies the position, it converts all opponent pieces into its pieces and adds another piece to it. The explosion is limited to the cardinal directions if they are available on the board. Similarly the critical mass of the square is defined by the number of places that are available for it to explode to. For example, a piece located at the edge of the board can explode in 3 directions, so it's critical mass is 2 , as soon as the 3rd mass is added to the square, the square explodes. In this way, players can eat up opponents' pieces and create a chain reaction of exploding squares. This is a highly dynamic game in which even if the opponent has a single piece on the board, the opponent still has a chance to win. The environment is deterministic, and agents act alternatingly. During training, the agents take every other step but learn and maintain a buffer for each step. Action masking is done during action-taking, but agents can try to take actions on squares that are occupied by opponent during training. Rewards are in the form of a zero-sum game; that is, each agent gets +100 for winning and -100 for losing. On reducing the opponent's pieces from the board, the agent gets a positive reward equal to the number of pieces it took, and the opponent gets a similar negative reward. If the agent tries to take action on a square occupied by the opponent agent, it wastes its turn.



Example of Chain Reaction Game Board State

4 Approach

Previous approaches to this problem involved using a min-max algorithm to essentially search through the possible states and find out the state that provided the maximum return. Our approach implements MARL algorithms like MADDPG and PPO in order to induce strategies within the agents through the use of neural networks. Our implementation is discussed further as follows:

4.1 MADDPG

It extends the Deep Deterministic Policy Gradient (DDPG) algorithm to multiagent settings, enabling decentralized agents to learn a centralized critic based on the observations and actions of all agents. Architecture: It works using an Actor-Critic Framework. Each agent has its actor (policy network), and there is a critic (value network) that is common to both agents. The critic network is trained with access to all agents' observations and actions, facilitating coordinated learning. However, during deployment, each agent operates independently using only its local observations. In the Chain Reaction game, MADDPG allows agents to learn optimal strategies by considering both their actions and the potential actions of other agents, leading to effective competition.Agents are taking actions every other step while the buffer is being maintained for every step.

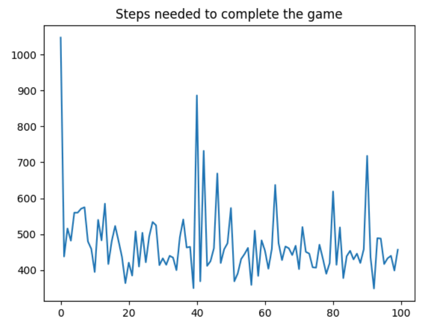
4.2 PPO

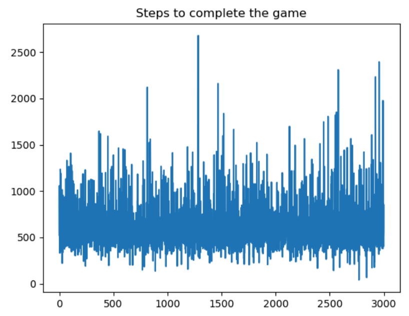
It is a policy gradient method that optimizes a surrogate objective function to improve training stability and sample efficiency.

Architecture: It involves a clipping mechanism that constrains policy updates to prevent large deviations, enhancing training stability. It also uses a clipped objective function to balance exploration and exploitation. In the Chain Reaction game scenario, PPO is implemented by sharing independent policies, allowing agents to learn effective strategies in the competitive setting. In PPO each agent is updating the environment every other step and training after a given amount of steps.

5 RESULTS

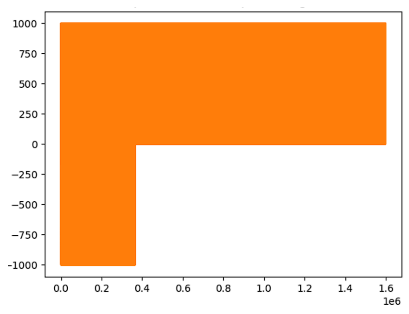
We implemented both MADDPG and PPO trained on two separate agents and tested the trained agents against an agent that used random policy. The following are the results:



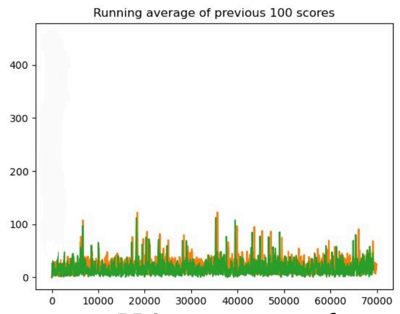


Random moves-based algorithm steps

PPO steps per game



MADDPG running avg of previous 100 episodes



PPO running avg of previous 100 episodes