Reinforcement Learning for Pacman

Kleon Karapas, Youssef Raoui

June 5, 2023

1 Introduction

Reinforcement learning has revolutionized the field of artificial intelligence by enabling agents to learn and make decisions through interactions with their environment. This project focuses on the application of reinforcement learning techniques to develop an intelligent agent for playing PacMan. Indeed, by training the agent to navigate the maze, avoid ghosts, and maximize scores, it explores the potential of reinforcement learning in solving complex gaming challenges.

2 Reinforcement Learning

2.1 Markov Decision Process

In reinforcement learning, the interactions between the agent and the environment are often described by an infinite-horizon, discounted Markov Decision Process (MDP). MDP consists of a set of states, actions, transition probabilities, rewards, and a discount factor. At each state, an agent selects an action, which transitions the system to a new state according to probabilities. The agent receives a reward based on the chosen action and the resulting state. The transition probabilities and rewards are determined by the dynamics of the environment. The objective is to find a policy that maximizes the expected cumulative discounted rewards over time, considering the uncertain nature of the system.

2.2 Stochastic Policy Gradient

Stochastic policy gradient is a reinforcement learning algorithm that aims to learn an optimal policy for decision-making in stochastic environments. It uses gradient ascent to iteratively update the policy parameters based on the expected return. By sampling actions according to the current policy and estimating the expected return using Monte Carlo methods, the algorithm calculates the policy gradient and updates the parameters to maximize the expected return.

As a part of our project, we've chosen to chose to work with REINFORCE Policy gradients which correponds to the following expressions :

$$\nabla V_{\pi_\theta}(\mu) = E_{\pi_\theta}\left[\sum_{t=0}^{\infty} R(\tau)\nabla \log \pi_\theta(a_t|s_t)\right] = E_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \cdot R(t) \cdot \nabla \log \pi_\theta(a_t|s_t)\right] \tag{1}$$

2.3 Softmax Policies / Log-Linear policies

During our project, in order to implement our stochastic policy gradient we started by working on a grid experimental set-up. We chose a softmax policy which corresponds to the following expression :

$$\pi_\theta(a|s) = \frac{\exp(\theta_{s,a})}{\sum_{a'} \exp(\theta_{s,a'})} \tag{2}$$

where the parameter space is $\Theta = R^{|S||A|}$.

When we started working on our real Pacman setup, we noticed that the softmax was not efficient anymore. Indeed, our state space was too large to be stored ($\theta = S_{\text{pacman}} \times S_{\text{ghost}} \times \text{Action} \times \{0,1\}^{100} = 100 \times 100 \times 4 \times 2^{100}$) . To solve that, we've tried to work on a log-linear policy. For any state-action pair s, a, suppose we have a feature mapping $\varphi_{s,a} \in R^d$. Let us consider the policy class

$$\pi_\theta(a|s,\phi) = \frac{\exp(\theta^T \phi_{s,a})}{\sum_{a'} \exp(\theta^T \phi_{s,a'})} \tag{3}$$

We've started to work on the following features : Φ_1 = Collected coins, Φ_2 = Action chosen leads to coin, Φ_3 = Distance to closest coin and Φ_4 = Distance to closest ghost. However, due to lack of time, the code for the features could not be completed.

2.4 What have we learned during this project ?

Overall, a bachelor project in Reinforcement Learning for Pacman offers the opportunity to delve into core RL concepts, algorithm implementation, environment modeling, feature engineering. As a matter of fact, you can study and implement various RL algorithms such as value iteration, policy iteration, stochastic policy gradient... We've developped a model of the Pacman environment, including the game mechanics, walls, and interactions with ghosts and coins. Thus, we've explored the design of informative features for Pacman agents.

---

3 Appendix

3.1 Notation

- A state space S, which may be finite or infinite. For mathematical convenience, we will assume that S is finite or countably infinite.
- An action space A, which also may be discrete or infinite. For mathematical convenience, we will assume that A is finite.
- A transition function $P : S \times A \to \Delta(S)$, where $\Delta(S)$ is the space of probability distributions over S (i.e., the probability simplex). $P(s'|s,a)$ is the probability of transitioning into state s' upon taking action a in state s. We use $P_{s,a}$ to denote the vector $P(\cdot|s,a)$.
- A reward function $r : S \times A \to [0,1]$. $r(s,a)$ is the immediate reward associated with taking action a in state s. More generally, $r(s,a)$ could be a random variable (where the distribution depends on s,a). While we largely focus on the case where $r(s,a)$ is deterministic, the extension to methods with stochastic rewards is often straightforward.
- A discount factor $\gamma \in [0,1)$, which defines a horizon for the problem.
- An initial state distribution $\mu \in \Delta(S)$, which specifies how the initial state s_0 is generated.

3.2 References

1. Alekh Agarwal, Nan Jiang, Sham M. Kakade, Wen Sun. "Reinforcement learning: theory and algorithm".
2. Richard S. Sutton, Andrew G. Barto. "Reinforcement learning".