Reinforcement Learning HW 3

Mert Bilgin (7034879)
mert.bilgin@student.uni-tuebingen.de

Lalitha Sivakumar (6300674)
lalitha.sivakumar@student.uni-tuebingen.de

Kevin Van Le (7314700)
kevin-van.le@student.uni-tuebingen.de

November 5, 2025

1 Recap

a)

A Markov reward process (MRP) is just a Markov chain with rewards and a discount factor. Instead of moving between states, each transition gets some reward and we care about the long-term expected return from each state. Formally, an MRP is defined as a tuple $(\mathcal{S}, P, R, \gamma)$ where:

- $\mathcal{S}$ is a finite set of states,
- $P(S_{t+1} \mid S_t)$ is the state transition probability matrix,
- $R(s)$ is the expected reward on transition,
- $\gamma \in [0, 1]$ is the discount factor.

b)

A MDP could be reduced to MRP when the decision-making part is removed. This happens when we fix a policy $\pi(a \mid S_t)$ that determines which action to take in each state.

c)

MRPs can be solved in closed form as they only have states, transitions, and rewards which we can describe as a system of linear equations. However in MDPs include actions, so the agent must choose among different options that affect future states and rewards. We have a max operation in the Bellman optimality equation, making it non-linear. Therefore, it can't be solved in closed form and require iterative methods for optimal solution.

2 TD(λ)

2.1 a)

$$\begin{aligned} G_t^{(n)} &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \\ G_{t+1}^{(n-1)} &= R_{t+2} + \gamma R_{t+3} + \cdots + \gamma^{n-2} R_{t+n} + \gamma^{n-1} V(S_{t+n}) \\ G_t^{(n)} &= R_{t+1} + \gamma\big(R_{t+2} + \gamma R_{t+3} + \cdots + \gamma^{n-2} R_{t+n} + \gamma^{n-1} V(S_{t+n})\big) \\ &= R_{t+1} + \gamma G_{t+1}^{(n-1)} \end{aligned}$$

b)

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

$$\text{Using part (a)}, \quad G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\big(R_{t+1} + \gamma G_{t+1}^{(n-1)}\big)$$

$$= R_{t+1}(1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} + \gamma(1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t+1}^{(n-1)}$$

By the geometric series,

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \frac{1}{1-\lambda}$$

$$\begin{aligned} G_t^\lambda &= R_{t+1} + \gamma(1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t+1}^{(n-1)} \\ &= R_{t+1} + \gamma(1-\lambda)\sum_{n=0}^{\infty} \lambda^{n} G_{t+1}^{(n)} \\ &= R_{t+1} + \gamma\Big[(1-\lambda)V(S_{t+1}) + \lambda G_{t+1}^\lambda\Big] \end{aligned}$$

$$\boxed{G_t^\lambda = R_{t+1} + \gamma\Big[(1-\lambda)V(S_{t+1}) + \lambda G_{t+1}^\lambda\Big]}$$

3 Policy Iteration

(b)

With noise $n = 0.2$ or $n = 0.5$: **2 policy iterations** until the start state becomes non-zero.
Without noise $n = 0$: **8 policy iterations** until the start state becomes non-zero.

(c)

Noise Level	Iterations to Converge
$n = 0$	8
$n = 0.2$	2
$n = 0.5$	2

Explanation for $n = 0$

The reason it is 8 policy iterations and not 12 policy iterations from start to finish is because at the beginning the policy is initialized with taking the action going north for all states. For some states this is already the optimal action, hence for a state s_t where the optimal action is going north and the next state s_{t+1} have a non-zero value then state s_t will also be updated with a non-zero value given that the discount is greater than 0. This value will propagate backwards to the subsequent states as long as the policy already takes the optimal action in a given state where the next state has a non-zero value.

Explanation for $n > 0$

It converges even faster with noise, because due to noise all actions will be taken with a non-zero probability and thus the optimal action is included which leads to multiple non-zero value updates and therefore faster convergence.

(d)

Advantages of Policy Iteration:

1. converges faster as can be seen in 3c. Value Iteration only updates one subsequent state, i.e. it will take 12 iterations in the GridMaze for the start state to update with a non-zero value.

2. due to faster convergence, it also needs less compute

Disadvantages of Policy Iteration:

1. if the environment is really small, i.e. number of Value Iteration == number of Policy Iteration, then Value Iteration could be cheaper to compute as a single iteration in Policy Iteration is more expensive than in Value Iteration

2. Policy Iteration is more complex, as not only the value function is tracked but also the policy