Week 2 Report

Mu Junrong

Abstract

This report focuses on the Markov Decision Process (MDP), the difference between Reinforcement Learning (RL) and Supervised Learning, the real-world application of Reinforcement Learning, and the objectives of Reinforcement Learning. This report also discusses the basic policy gradient derivation and baseline for the policy gradient. Lastly, an algorithm for the policy gradient is presented.

1 Markov Decision Process (MDP)

The Markov Decision Process (MDP) is a mathematical model of the decision-making process, extensively used in the field of Reinforcement Learning (RL) (Sutton & Barto (2018)). It formally describes a fully observable environment where an agent performs actions, and outcomes are partly random. In other words, Markov Decision Process (MDP) is a Markov Reward Process (MRP) with decisions.

It is under the assumption that the underlying structure of state transitions follow the Markov Property. The process is a decision process because it involves decision-making that influence the state transitions.

1.1 Markov Property

Markov Property describes the state in which the future is independent of the past, given the present. A state S_t is Markov if and only if

$$P[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ldots, S_t]$$

When the state S_t is Markov, the state captures all relevant information from the history. Therefore, once the state is known, the history can be discarded (i.e. the state is a sufficient statistic of the future).

1.2 Markov Process (Markov Chain)

The Markov Process is a memoryless stochastic process, consisting of a sequence of random states $S_1, S_2, \ldots$ with the Markov Property explained above.

A Markov Process (or Markov Chain) is a tuple $\langle S, P \rangle$ where S is a (finite) set of states, P is a state transition probability matrix such that

$$P = \begin{bmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{bmatrix}$$

For a Markov state s and successor state s', the state transition probability is defined by

$$P_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

1.3 Markov Reward Process (MRP)

A Markov reward process is a Markov chain with values. The agent does not choose actions. It is just adding values to sequences of actions.

1.3.1 Definition of Markov Reward Process

MRP is a tuple $\langle S, P, R, \gamma \rangle$, where S is a finite set of states, P is a state transition probability matrix,

$$P_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

R is a reward function,

$$R_s = \mathbb{E}[R_{t+1} \mid S_t = s]$$

γ is a discount factor,

$$\gamma \in [0, 1]$$

1.3.2 Return value of Markov Reward Process

The return G_t is the total discounted reward from time-step t.

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

The discount $\gamma \in [0, 1]$ is the present value of future rewards, and the value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.

MRP is discounted because it represents delayed reward. For example, when applying MRP on financial problem, the immediate rewards are more favourable since more interest can be earned than delayed rewards.

1.3.3 Value function of Markov Reward Process

The value function $v(s)$ gives the long-term value of state s. The state value function $v(s)$ of an MRP is the expected return starting from state s:

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

By Bellman Equation for MRP, the value function can be decomposed into two parts: immediate reward R_{t+1}, discounted value of successor state $\gamma v(S_{t+1})$, where

$$\begin{aligned} v(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots) | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \\ &= R_s + \gamma \sum_{s' \in S} P_{ss'} v(s') \end{aligned}$$

The value function of the current step = current step rewards + sum of product of probability of next step and reward of next step.

1.4 Markov Decision Process (MDP)

A Markov Decision Process (MDP) is a Markov Reward Process with the agent making decisions. It is under an environment where all states are Markov (i.e. every state has the Markov property).

1.4.1 Definition of Markov Decision Process (MDP)

MDP is a tuple $\langle S, A, P, R, \gamma \rangle$, where S is a finite set of states, A is a finite set of actions, P is a state transition probability matrix, which is defined as $P_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$, R is a reward function with the reward R_s at state s being $R_s = \mathbb{E}[R_{t+1} \mid S_t = s]$, and $\gamma \in [0, 1]$ is a discount factor.

1.4.2 Policy

A policy π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

The policy fully defines the behaviour of the agent (i.e. the policy is the action taken by the agent). The MDP policies depend only on the current state instead of the history, and the policies are stationary (time-independent).
Given an MDP $M = \langle S, A, P, R, \gamma \rangle$ and a policy π, the state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $\langle S, P^\pi, R^\pi, \gamma \rangle$ where

$$P^\pi_{s,s'} = \sum_{a \in A} \pi(a|s) P^a_{ss'}$$

$$R^\pi_s = \sum_{a \in A} \pi(a|s) R^a_s$$

1.4.3 Value function of policy

The state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right] \\ &= \sum_{a \in A} \pi(a|s) \left(R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s') \right) \end{aligned}$$

The action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right] \\ &= R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a' \in A} \pi(a'|s') q_\pi(s', a') \end{aligned}$$

1.4.4 Optimal value function and policy

The optimal state-value function $v_*(s)$ is the maximum value function attainable over all policies:

$$v_*(s) = \max_\pi v_\pi(s)$$

The optimal action value function $q_*(s, a)$ is the maximum action-value function attainable over all policies:

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

To find the optimal policy, a partial ordering over the policies is defined:

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

For any Markov Decision Process: There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$. All optimal policies achieve the optimal value function and the optimal action-value function. An optimal policy can be found by maximizing over $q_*(s, a)$:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in A}{\operatorname{argmax}}\, q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

2 Differences between Reinforcement Learning and Supervised Learning

Both Reinforcement Learning and Supervised Learning are schemes of Machine Learning (ML). They are different in ultimate goals and learning signals, data and outputs, as well as use cases.

2.1 Ultimate goals and learning signals

Supervised Learning aims to learn a mapping from inputs to outputs. $f : \mathcal{X} \to \mathcal{Y}$ To achieve this, A loss function is used in Supervised Learning (Goodfellow et al. (2016)), to measure the inaccuracy of the model's prediction, compared to the true label. For exmaple, one value function used is the mean square error:

$$\mathcal{L}(f(x), y) = (f(x) - y)^2$$

The mapping from inputs to outputs is learnt by minimizing the loss function. Training is typically done using gradient descent.

On the other hand, in Reinforcement Learning, the agent learns a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes expected return:

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

through interaction with the environment (Sutton & Barto (2018)).
To get the optimal policy, we want to learn the **optimal policy** π^* that maximizes expected return from any state s:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}[G_t \mid S_t = s]$$

To do this, we define:

- **State-Value Function:**

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

- **Action-Value Function (Q-function):**

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

- **Bellman Optimality Equations** define recursive relationships:

$$V^*(s) = \max_{a}\left[R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s')\right]$$

2.2 Data and outputs

Supervised Learning includes a pre-labeled dataset (i.e. pairs of inputs and their respective correct outputs)

$$D = \{(x_i, y_i)\}_{i=1}^{N}$$

This requires large amount of manual effort to label the data.

The output will be a prediction of the input value (e.g., classification or regression of the input).

On the other hand, Reinforcement Learning does not include labels. It allows the agent to interact with an environment to maximize cumulative data rewards. The output returns a policy that maps from states to actions.

2.3 Use cases

Supervised learning foes not require exploration since data given is fixed. It can be used in the scenario of prediction and classification. For example, in the case of spam email detection (using KNN algorithm). On the other hand, reinforcement learning requires exploration (i.e. agent must explore actions). It can be used in cases such as recommendation systems and Artificial Intelligence (AI) gaming.

3 Real-world case of Reinforcement Learning

Reinforcement Learning is widely used in the areas of robotics, healthcare, gaming and Natural Language Processing (NLP). One such example is training the self-driving car. In this case, the agent is the Artificial Intelligence (AI) used in the self-driving car. It is able to take actions such as accelerating and decelerating. The environment that the agent is in is the road, including cars, pedestrians, other vehicles and traffic signals. The tuple $\langle S, A, P, R, \gamma \rangle$ represents the following respectively:

- State ($s \in S$): each state may include the current speed, the distance to the nearest car or pedestrian, the color of the traffic light, the weather conditions, the time of the day and the current condition of the car.

- Actions ($a \in A$): at each state, the agent can choose actions including accelerating, decelerating, braking and turning left or right. Such actions can change the state of the environment.

- State transition possibility matrix ($P(s'|s, a)$): The possibility matrix describes how the agent make actions. For example, if the agent take the action of braking, it leads to a decelerating state.

- Reward Function ($R(s, a)$): The reward function can be designed and guides the agent's behaviour. For example, ten points to be added for proceeding at green light while ten points to be deducted for running a red light. If it collides with another vehicle or hits a pedestrian, one hundred points will be deducted. If it successfully reaches the destination, one hundred points will be awarded.

3.1 Value function

The value function ($V^{\pi}(s)$) determines how good a state s is under the policy. For example, a state with clear roads nearer to the destination has high value, while a state with traffic congestion has low value.

3.2 Policy

The policy ($\pi(a|s)$) determines which action to be taken in a state. For example, the agent should decelerate when it is in a state that is close to the pedestrian (e.g., 50 meters from the pedestrian). The optional policy (π^*) is the best driving strategy to be taken in this environment. It maximizes the total expected reward

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right]$$

By Reinforcement Learning, the car can learn to follow traffic rules, avoid traffic incidences and reach the destination safely and efficiently.

4 Objective of Reinforcement Learning

The main objective of Reinforcement Learning (RL) is to find an optimal policy π that maximizes the reward (i.e. maximizes the state-value function and action-value function):

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right]$$

5 Policy gradient

Given the objective of Reinforcement Learning, which is to maximize the total expected reward:

$$\pi^* = \arg\max \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | \pi \right]$$

the expected return is maximized by adjusting the policy parameters. This is done by defining the objective function and deriving the basic policy gradient (Sutton et al. (1999)).

5.1 Objective function

Let $\pi_\theta(a|s)$ be a stochastic policy parameterized by θ (where θ can be parameters of the distribution), and let τ represent a trajectory (i.e., the sequence of states, actions, rewards):

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots)$$

The objective function (which is the mathematical definition of the goal of the optimization problem) can be defined as

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

Based on the definition of expectation, let $p_\theta(\tau)$ be the probability of trajectory τ under policy π_θ.

$$J(\theta) = \int p_\theta(\tau) R(\tau) \, d\tau$$

5.2 Derivation of basic policy gradient

Based on the chain rule in calculus, If $f(\theta) > 0$, then

$$\frac{d}{d\theta} \log f(\theta) = \frac{1}{f(\theta)} \cdot \frac{df(\theta)}{d\theta}$$

Multiplying both sides by $f(\theta)$

$$\frac{df(\theta)}{d\theta} = f(\theta) \cdot \frac{d}{d\theta} \log f(\theta)$$

When $f(\theta) = p_\theta(\tau)$

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \cdot \nabla_\theta \log p_\theta(\tau)$$

Thus, by differentiating the objective function $J(\theta)$:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \int p_\theta(\tau) R(\tau)\, d\tau \\ &= \int \nabla_\theta p_\theta(\tau) R(\tau)\, d\tau \\ &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau)\, d\tau \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log p_\theta(\tau) R(\tau)\right]\end{aligned}$$

5.3 Basic policy gradient

The trajectory probability under the policy is given by

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

Since $p(s_{t+1}|s_t, a_t)$ is independent of the variable θ, and a trajectory is a sequence of actions $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$. Thus, $\pi_\theta(\tau) = \pi_\theta(a_0|s_0) \cdot \pi_\theta(a_1|s_1) \cdots \pi_\theta(a_T|s_T)$. By taking the logarithm on both sides

$$\log \pi_\theta(\tau) = \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t)$$

Therefore,

$$\log p_\theta(\tau) = \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) + \mathrm{C}$$

where C is a constant. By differentiating both sides,

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

By substituting into the objective function, the basic policy gradient formula is derived

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right) R(\tau)\right]$$

6 Baselines in policy gradient

From section 5 Policy gradient, the policy gradient is derived such that

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right) R(\tau)\right]$$

If the agent encountered a high reward $R(\tau)$, the agent has a higher probability of making that action. However, if it encounters a low reward, it is less likely to take the action. However, value updates can be noisy and unstable due to factors such as randomness in the environment.

Thus, a baseline can be added to ensure faster and more stable policy learning.

6.1 Baseline

A baseline is a reference value that is subtracted from the return (or reward) when calculating the policy gradient. It measures how much better or worse an action is compared to expected outcomes. The baseline $b(s_t)$ is subtracted from the return

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot (G_t - b(s_t))\right]$$

where G_t is return from time t, and $b(s_t)$ is baseline (Williams (1992)). Subtracting the baseline does not change the expected value of the policy gradient, since the integration of a probability integrates to 0.

$$\begin{aligned}\mathbb{E}_{a_t}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] &= \nabla_\theta \int \pi_\theta(a_t \mid s_t)\, da_t \\ &= \nabla_\theta 1 \\ &= 0\end{aligned}$$

Therefore,

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot b(s_t)\right] = 0$$

This proves that by subtracting the baseline, the expectation is not changed, while the variance is reduced, reducing the noise.

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t\right] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot (G_t - b(s_t))\right]\end{aligned}$$

6.2 Common baseline forms

The common baseline forms are Zero baseline ($b(s_t) = 0$, which does not reduce variance), Constant baseline ($b(s_t) = average(G_t)$), as well as Learned baseline ($b(s_t) = \hat{V}_\phi(s_t)$). The most common one is the on-policy value function

$$b(s_t) = V^\pi(s_t)$$

7 Algorithm for policy gradient method

The goal of the policy gradient is to train a policy$\pi_\theta(a|s)$ to maximize the expected total reward the agent get over time. At each time step t, the agent gets a reward R_t. The reward-to-go from t is

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

To achieve this, Vanilla policy gradient with ready-to-go is used (Levine (2023)).

7.1 Algorithm

1. Initialize a policy $\pi_\theta(a \mid s)$:
use a neural network that takes a state and outputs a probability distribution over actions. The parameters of this network are θ.

2. Collect trajectories using current policy, and for each trajectory:
Reset the environment: `state = env.reset()`
Repeat for T steps or until convergence: Sample action $a_t \sim \pi_\theta(\cdot \mid s_t)$ Apply action: get next state and reward (s_{t+1}, r_t) Store (s_t, a_t, r_t)

3. Compute reward-to-go R_t for each time step by $G_t = R_{t+1} + \gamma R_{t+2} + \ldots$

4. Estimate policy gradient by $\nabla_\theta J(\theta) = \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) R_t$

5. Update policy parameter θ by $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta)$ Where α is the learning rate.

7.2 Pseudocode

Algorithm 1 Vanilla Policy Gradient with Reward-to-Go

Input: Policy $\pi_\theta(a \mid s)$ parameterized by θ; learning rate α; number of trajectories N; horizon T; discount factor γ
while not converged **do**
  Collect a batch of N trajectories $\{\tau_1, \ldots, \tau_N\}$ by running policy π_θ
  for each trajectory $\tau_i = (s_0, a_0, r_0, \ldots, s_T, a_T, r_T)$ **do**
    for each time step t **do**
      Store (s_t, a_t, r_t)
      Compute reward-to-go:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T$$

    end for
  end for
  Estimate policy gradient:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t$$

  Update policy parameters using gradient ascent:

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta)$$

end while

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL https://www.deeplearningbook.org.

Sergey Levine. Cs285: Deep reinforcement learning. https://rail.eecs.berkeley.edu/deeprlcourse/, 2023. Lecture materials from UC Berkeley.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018. URL http://incompleteideas.net/book/RLbook2020.pdf.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (NeurIPS 1999)*, pp. 1057–1063. MIT Press, 1999.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696.