Notes on Reinforcement Learning Theory (ongoing)

[Siyu ZHOU]

Academic Year 2022

Contents

1 References

1. Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.
2. Introduction to Reinforcement Learning with David Silver [https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver].
3. Dong, Hao, et al. Deep Reinforcement Learning. Springer Singapore, 2020.

2 Lecture 2: Markov Decision Processes

2.1 Markov Processes

Markov decision processes formally describe an environment for reinforcement learning, where the environment is **fully observable**, i.e., the current state completely characterises the process. Almost all RL problems can be formalised as MDPs, e.g., optimal control primarily deals with continuous MDPs; Partially observable problems can be converted into MDPs; Bandits are MDPs with one state.

Markov Property: "The future is independent of the past given the present."
State Transition Matrix: For a Markov state s and successor state s' , the state transition probability is defined by

$$\mathcal{P}_{SS'} = \mathbb{P}[S_{t+1} = s' | S_t = s] \tag{1}$$

Definition 1 (Markov Process)**.** *A Markov Process (or Markov Chain) is a tuple* $\langle \mathcal{S}, \mathcal{P} \rangle$*, where* $\mathcal{S}$ *is a (finite) set of states and* $\mathcal{P}$ *is a state transition probability matrix, e.g.,* $\mathcal{P}_{SS'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$.

2.2 Markov Reward Processes

Definition 2 (Markov Reward Process)**.** *A Markov Reward Process is a tuple* $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$*, where* $\mathcal{S}$ *is a (finite) set of states;* $\mathcal{P}$ *is a state transition probability matrix, e.g.,* $\mathcal{P}_{SS'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$*;* $\mathcal{R}$ *is a reward function,* $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$ [1]*;* γ *is a discount factor,* $\gamma \in [0, 1]$.

Definition 3 (Return)**.** *The return* G_t *is the total discounted reward from time-step* t.

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

Definition 4 (Value Function)**.** *The state value function* $v(s)$ *of an MRP is the expected return starting from state* s

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

Bellman Equation for MRPs
The value function can be decomposed into two parts: immediate reward R_{t+1} and discounted value of successor state $\gamma v(S_{t+1})$

$$\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v\left(S_{t+1}\right) \mid S_t = s\right] \\
&= \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')
\end{aligned}$$

Bellman Equation in Matrix Form
The Bellman equation can be expressed concisely using matrices,

$$\boldsymbol{v} = \boldsymbol{\mathcal{R}} + \gamma \boldsymbol{\mathcal{P}} \boldsymbol{v}$$

where $\boldsymbol{v}$ is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \ldots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \ldots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} \tag{2}$$

Solving the Bellman Equation
The Bellman equation is a linear equation. It can be solved directly:

$$\begin{aligned}
\boldsymbol{v} &= \boldsymbol{\mathcal{R}} + \gamma \boldsymbol{\mathcal{P}} \boldsymbol{v} \\
(\boldsymbol{I} - \gamma \boldsymbol{\mathcal{P}}) &= \boldsymbol{\mathcal{R}} \\
\boldsymbol{v} &= (\boldsymbol{I} - \gamma \boldsymbol{\mathcal{P}})^{-1} \boldsymbol{\mathcal{R}}
\end{aligned}$$

[1]only related with S_t (current state), but obtained at next time point (see Def. 8)

But the computational complexity is $O(n^3)$ for n states, direct solution is therefore only possible for small MRPs. There are many **iterative methods** for large MRPs, e.g.

- Dynamic programming
- Monte-Carlo evaluation
- Temporal-Difference learning

2.3 Markov Decision Processes

Definition 5 (Markov Decision Process)**.** *A Markov Reward Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a (finite) set of states; $\mathcal{A}$ is a finite set of actions; $\mathcal{P}$ is a state transition probability matrix, e.g., $\mathcal{P}^a_{SS'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$; $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, , A_t = a]$; γ is a discount factor, $\gamma \in [0, 1]$.*

Definition 6 (Policy)**.** *A policy π is a distribution over actions given states,*

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

The introduction of action redefines transition probability $\mathcal{P}$ and reward function $\mathcal{R}$
Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π;
The state sequence $S_1, S_2, \ldots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
The state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
where

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

Definition 7 (State Value Function)**.** *The state value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π*

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Definition 8 (Action Value Function)**.** *The action value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π*

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Corollary 1 (Bellman Expectation Equation for v^π)**.**

$$\begin{aligned} v_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_\pi(s') \right) \end{aligned}$$

Corollary 2 (Bellman Expectation Equation for q^π)**.**

$$\begin{aligned} q_\pi(s, a) &= \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_\pi(s') \\ &= \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a') \end{aligned}$$

Bellman Expectation Equation (Matrix Form)
The Bellman expectation equation can be expressed concisely using the induced MRP,

$$\boldsymbol{v}_\pi = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \boldsymbol{v}_\pi$$

with direct solution

$$\boldsymbol{v}_\pi = (\boldsymbol{I} - \gamma \boldsymbol{\mathcal{P}}^\pi)^{-1} \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}}$$

Definition 9 (Optimal Value Function)**.** *The optimal state-value function $v_\star(s)$ is the maximum value function over all policies*

$$v_\star(s) = \max_\pi \; v_\pi(s)$$

The optimal action-value function $q_\star(s,a)$ is the maximum action-value function over all policies

$$q_\star(s,a) = \max_\pi \; q_\pi(s,a)$$

Finding an Optimal Policy

- All optimal policies achieve the optimal value function, $v_{\pi\star}(s) = v_\star(s)$ and action-value function, $q_{\pi\star}(s,a) = q_\star(s,a)$
- There is always a deterministic optimal policy for any MDP.
- If we know $q_\star(s,a)$, we immediately have the optimal policy

Corollary 3 (Bellman Optimality Equation for $v_\star$)**.**

$$\begin{aligned} v_\star(s) &= \max_a q_\star(s,a) \\ &= \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\star(s') \end{aligned}$$

Corollary 4 (Bellman Optimality Equation for $q_\star$)**.**

$$\begin{aligned} q_\star(s,a) &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\star(s') \\ &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_\star(s',a') \end{aligned}$$

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration, Policy Iteration, Q-learning, Sarsa

2.4 Extensions to MDPs

Infinite and continuous MDPs; Partially observable MDPs; Undiscounted, average reward MDPs.

2.4.1 Infinite MDPs

......

2.4.2 POMDPs

A Partially Observable Markov Decision Process is an MDP with hidden states. It is a hidden Markov model with actions.

Definition 10 (POMDPs)**.** *A POMDPs is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$*

- *$\mathcal{S}$ is a finite set of states*
- *$\mathcal{A}$ is a finite set of actions*
- *O is a finite set of observations*
- *$\mathcal{P}$ is a state transition probability matrix, e.g., $\mathcal{P}_{SS'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$*

- $\mathcal{R}$ *is a reward function,* $\mathcal{R}_s^a = \mathbb{E}[R_{t+1}|S_t = s', A_t = a]$
- $\mathcal{Z}$ *is an observation function,* $\mathcal{Z}_{s'o}^a = \mathbb{P}[O_{t+1} = o|S_{t+1} = s', A_t = a]$
- γ *is a discount factor,* $\gamma \in [0, 1]$.

Definition 11 (history)**.** *A history H_t is a sequence of actions, observations and rewards,*

$$H_t = A_0, O_1, R_1, \ldots, A_{t-1}, O_t, R_t$$

Definition 12 (belief state)**.** *A belief state $b(h)$ is a probability distribution over states, conditioned on the history h*

$$b(h) = (\mathbb{P}[S_t = s^1|H_t = h], \ldots, \mathbb{P}[S_t = s^n|H_t = h]) \tag{3}$$

......

2.4.3 Average Reward MDPs

......

3 Lecture 3: Planning by Dynamic Programming

3.1 Introduction

What is Dynamic Programming?
A method for solving complex problems
By breaking them down into subproblems: Solve the subproblems; Combine solutions to subproblems

Requirements for Dynamic Programming
Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure
- Overlapping subproblems

Planning by Dynamic Programming

- For prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
  - Output: value function $\boldsymbol{v}_\pi$
- Or for control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function $\boldsymbol{v}_\star$ and: optimal policy $\pi_\star$

Summary of DP Algorithms*–*Synchronous Dynamic Programming Algorithms
Complexity Disscusion???...

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Table 1: Synchronous Dynamic Programming Algorithms

3.2 Policy Evaluation

- *Problem*: evaluate a given policy π
- *Solution*: iterative application of **Bellman expectation** backup
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$ (**Convergence to v_π**)
- Using **synchronous** backups: At each iteration $k+1$; For all states $s \in \mathcal{S}$; Update $v_{k+1}(s)$ from $v_k(s')$, where s' is a successor state of s.
- **Convergence to v_π** will be proven at the end of the lecture.

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$
$$\boldsymbol{v}^{k+1} = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \boldsymbol{v}^k$$

3.3 Policy Iteration

How to Improve a Policy

Given the policy π

- Evaluate the policy π
$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$
- Improve the policy by acting greedily with respect to v_π
$$\pi' = \text{greedy}(v_\pi)$$

This process of policy iteration always converges to $\pi_\star$, as shown in Figure.1.

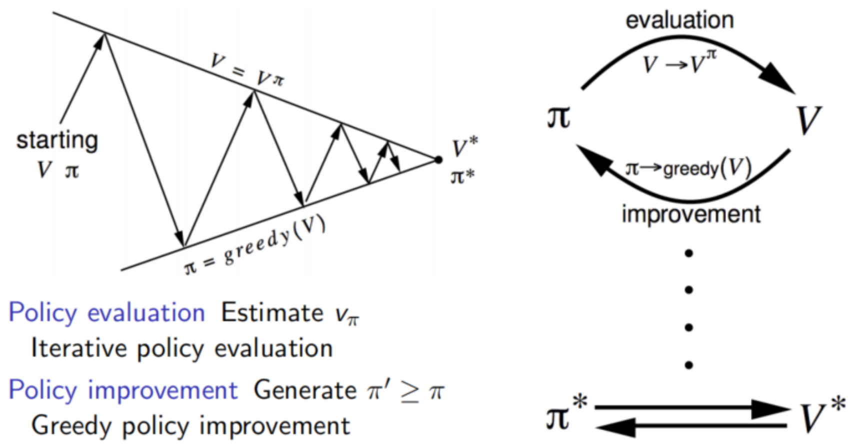

starting
V π
V = Vπ
π = greedy(V)
V*
π*
Policy evaluation Estimate v_π
Iterative policy evaluation
Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement
evaluation
V → V^π
π
V
π→greedy(V)
improvement
π*
V*

Figure 1: Policy Iteration.

Policy Improvement

1. Consider a deterministic policy, $a = \pi(s)$
2. We can improve the policy by acting greedily
$$\pi'(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s, a)$$
3. This improves the value from any state s over one step,
$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$
4. It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$
prove......
5. If improvements stop,
$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$
6. Then the Bellman optimality equation has been satisfied
$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$
7. Therefore $v_\pi(s) = v_\star(s)$, $\forall s \in \mathcal{S}$
8. so π is an optimal policy.

Modified Policy Iteration (Shown in Figure.2).

- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition, e.g., ϵ-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy
- Why not update policy every iteration? i.e., stop after $k = 1$.
  This is equivalent to ***value iteration*** (next section)

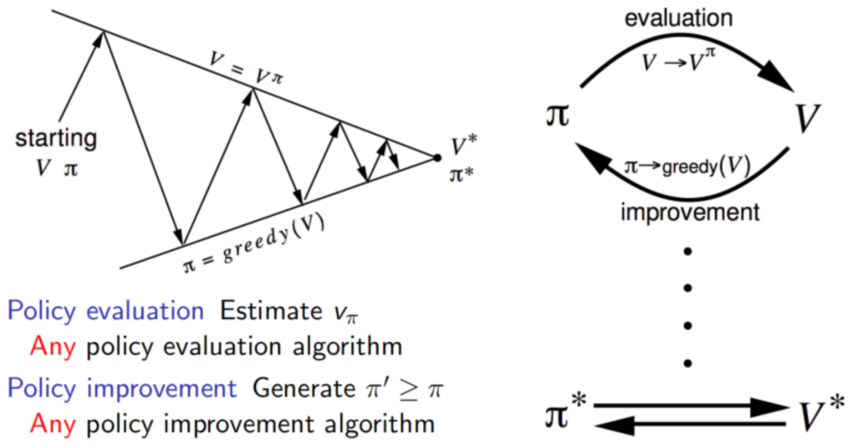

starting
V π
$V = V^\pi$
$\pi = \mathrm{greedy}(V)$
V^*
π^*
Policy evaluation Estimate v_π
Any policy evaluation algorithm
Policy improvement Generate $\pi' \geq \pi$
Any policy improvement algorithm
evaluation
$V \rightarrow V^\pi$
π
V
$\pi \rightarrow \mathrm{greedy}(V)$
improvement
π^*
V^*

Figure 2: Generalised Policy Iteration.

3.4 Value Iteration

- *Problem*: find optimal policy π
- *Solution*: iterative application of **Bellman optimality** backup
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\star$ (**Convergence to $v_\star$**)
- Using **synchronous** backups: At each iteration $k+1$; For all states $s \in \mathcal{S}$; Update $v_{k+1}(s)$ from $v_k(s')$.
- **Convergence to $v_\star$** will be proven later.
- Unlike policy iteration, there is no explicit policy.
- Intermediate value functions may not correspond to any policy.

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$
$$\boldsymbol{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^a + \gamma \boldsymbol{\mathcal{P}}^a \boldsymbol{v}^k$$

Theorem 1 (Principle of Optimality). *A policy $\pi(a|s)$ achieves the optimal vlue from state s, $v_\pi(s) = v_\star(s)$, if and only if for any state s' reachable from s, π achieves the optimal value from state s', $v_\pi(s') = v_\star(s')$.*

Explanation: Any optimal policy can be subdivided into two components:

- An optimal first action $A_\star$
- Followed by an optimal policy from successor state S'

Deterministic Value Iteration
If we know the solution to subproblems $v_\star(s')$. Then solution $v_\star(s)$ can be found by one-step lookahead

$$v_\star(s) \leftarrow \max_{a\in\mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\star(s')$$

The idea of value iteration is to apply these updates iteratively.
Intuition: start with final rewards and work backwards.
Still works with loopy, stochastic MDPs.

3.5 Extensions to Dynamic Programming

Asynchronous Dynamic Programming

- DP methods described so far used synchronous backups, i.e., all states are backed up in parallel
- Asynchronous DP backs up states individually, in any order. For each selected state, apply the appropriate backup. Can significantly reduce computation. Guaranteed to converge if all states continue to be selected.
- Three simple ideas for asynchronous dynamic programming:
  - In-place dynamic programming
  - Prioritised sweeping
  - Real-time dynamic programming

In-Place Dynamic Programming
Synchronous value iteration stores two copies of value function

$$v_{\text{new}}(s) = \max_{a\in\mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_{\text{old}}(s') \right) \quad \forall s \in \mathcal{S}$$

In-place value iteration only stores one copy of value function

$$v(s) = \max_{a\in\mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right) \quad \forall s \in \mathcal{S}$$

Prioritised Sweeping
......

Real-Time Dynamic Programming
......

Full-Width Backups vs Sample Backups
subsequent lectures......

Approximate Dynamic Programming
......

3.6 Contraction Mapping

Some Technical Questions

- How do we know that value iteration converges to $v_\star$?
- Or that iterative policy evaluation converges to v_π?
- And therefore that policy iteration converges to $v_\star$?
- Is the solution unique?

- How fast do these algorithms converge?
- These questions are resolved by contraction mapping theorem

Definition 13 (contraction mapping)**.** *Let $(\mathcal{X}, d)$ be a metric space. A mapping $T: \mathcal{X} \to \mathcal{X}$ is a contraction mapping, or contraction, if there exists a constant c, with $0 \le c < 1$, such that*

$$d(T(x), T(y)) \le c\, d(x, y) \quad \forall x, y \in \mathcal{X}.$$

Theorem 2 (contraction mapping)**.** *If $T: \mathcal{X} \to \mathcal{X}$ is a contraction mapping on a complete metric space $(\mathcal{X}, d)$, then there is exactly one solution $x \in \mathcal{X}$ of $T(x) = x$; T converges to a unique fixed point.*

Bellman Expectation Backup is a Contraction

- Consider the vector space $\mathcal{V}$ over value functions.
- There are $|\mathcal{S}|$ dimensions.
- Each point in this space fully specifies a value function $v(s)$.
- What does a Bellman backup do to points in this space?
- We will show that it brings value functions closer
- And therefore the backups must converge on a unique solution

We will measure distance between state-value functions $\boldsymbol{u}$ and $\boldsymbol{v}$ by the $\infty - norm$, i.e., the largest difference between state values,

$$\|\boldsymbol{u} - \boldsymbol{v}\| = \max_{s \in \mathcal{S}} |u(s) - v(s)|$$

Define the Bellman expectation backup operator T^π,

$$T^\pi(\boldsymbol{v}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \boldsymbol{v}$$

This operator is a γ-contraction, i.e., it makes value functions closer by at least γ,

$$\begin{aligned}
\|T^\pi(u) - T^\pi(v)\|_\infty &= \|(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi \boldsymbol{u}) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi \boldsymbol{v})\|_\infty \\
&= \|\gamma \mathcal{P}^\pi (\boldsymbol{u} - \boldsymbol{v})\|_\infty \\
&\le \|\gamma \mathcal{P}^\pi \|\boldsymbol{u} - \boldsymbol{v}\|_\infty\|_\infty \\
&\le \gamma \|\boldsymbol{u} - \boldsymbol{v}\|_\infty
\end{aligned}$$

Convergence of Iter. Policy Evaluation, Policy Iteration and Value Iteration

- The Bellman expectation operator T^π has a unique fixed point, v_π is a fixed point of T^π (by Bellman expectation equation). By contraction mapping theorem, Iterative policy evaluation converges on v_π; Policy iteration converges on $v_\star$
- The Bellman optimality operator $T^\star$ has a unique fixed point, $v_\star$ is a fixed point of $T^\star$ (by Bellman expectation equation). By contraction mapping theorem, Value iteration converges on $v_\star$;

[Reference]

1. https://www.math.ucdavis.edu/~hunter/book/ch3.pdf
2. https://zhuanlan.zhihu.com/p/26214408

4 Lecture 4: Model-Free Prediction

4.1 Introduction

- Last lecture:
  - Planning by dynamic programming
  - Solve a known MDP
- This lecture:
  - Model-free prediction
  - Estimate the value function of an unknown MDP
- Next lecture:
  - Model-free control
  - Optimise the value function of an unknown MDP

4.2 Monte-Carlo Learning

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from complete episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs, e.g., All episodes must terminate

4.2.1 Monte-Carlo Policy Evaluation

Goal: learn v_π from episodes of experience under policy π.

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

Recall that the return is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Monte-Carlo policy evaluation uses empirical mean return instead of expected return.

First-Visit/Every-Visit Monte-Carlo Policy Evaluation

- To evaluate state s
- The first time-step t/Every time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

4.2.2 Incremental Monte-Carlo

Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x1, x2, \ldots$ can be computed incrementally,

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\
&= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right) \\
&= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})
\end{aligned}$$

Incremental Monte-Carlo Updates

Update $V(s)$ incrementally after episode $S_1, A_1, R_2, \ldots, S_T$. For each state S_t with return G_t

$$\begin{aligned}
N(S_t) &\leftarrow N(S_t) + 1 \\
V(S_t) &\leftarrow V(S_t) + \frac{1}{N(S_t)}\left(G_t - V(S_t)\right)
\end{aligned}$$

In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

4.3 Temporal-Difference Learning

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes, by bootstrapping
- TD updates a guess towards a guess

4.3.1 Temporal-Difference Policy Evaluation (compared with MC prediction)

Goal: learn v_π online from experience under policy π

- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

- Simplest temporal-difference learning algorithm: $TD(0)$
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right),$$

    where S_t and S_{t+1} are s and s' respectively.
  - $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error

4.4 MC vs. TD

4.4.1 Learning Mode

TD can learn before knowing the final outcome, e.g., TD can learn online after every step; MC must wait until end of episode before return is known
TD can learn without the final outcome

- TD can learn from incomplete sequences; MC can only learn from complete sequences
- TD works in continuing (non-terminating) environments; MC only works for episodic (terminating) environments

4.4.2 Bias/Variance Trade-Off

- MC has high variance, zero bias, e.g., Good convergence properties (even with function approximation); Not very sensitive to initial value; Very simple to understand and use
- TD has low variance, some bias, e.g., Usually more efficient than MC; $TD(0)$ converges to $v_\pi(s)$ (but not always with function approximation); More sensitive to initial value
- For example,
  - Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$ is unbiased estimate of $v_\pi(S_t)$
  - True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is unbiased estimate of $v_\pi(S_t)$;
    TD target $R_{t+1} + \gamma V(S_{t+1})$ is biased estimate of $v_\pi(S_t)$
  - TD target is much lower variance than the return:
    * Return depends on many random actions, transitions, rewards
    * TD target depends on one random action, transition, reward

4.4.3 Batch MC and TD

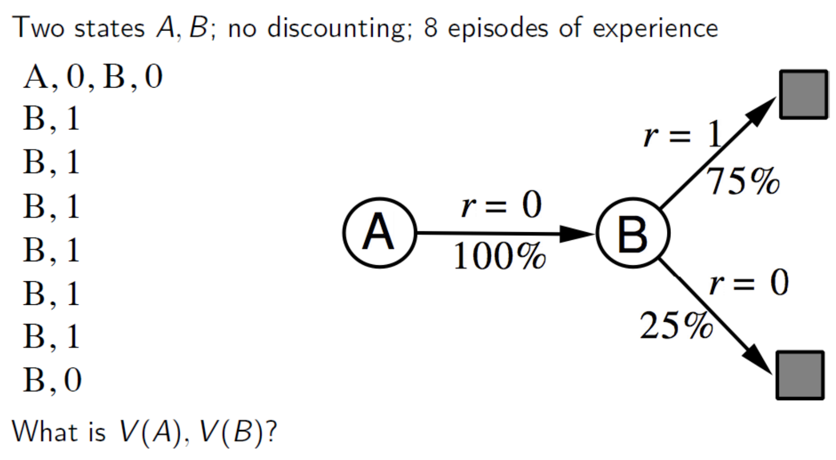
- TD exploits Markov property, e.g., Usually more efficient in Markov environments
- MC does not exploit Markov property, e.g., Usually more effective in non-Markov environments

Background:
MC and TD converge: $V(s) \to v_\pi(s)$ as experience $\to \infty$
But what about batch solution for finite experience?

$$\begin{array}{l} s_1^1, a_1^1, r_2^1, \ldots, s_{T_1}^1 \\ \vdots \\ s_1^K, a_1^K, r_2^K, \ldots, s_{T_K}^K \end{array}$$

e.g., Repeatedly sample episode $k \in [1, K]$; Apply MC or TD(0) to episode k.

AB Example:


Two states A, B; no discounting; 8 episodes of experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

What is $V(A), V(B)$?

Figure 3: AB Example.

Certainty Equivalence:

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

$$\sum_{k=1}^{K}\sum_{t=1}^{T_k}(G_t^k - V(s_t^k))^2$$

  - In the AB example, $V(A) = 0$
- TD(0) converges to solution of max likelihood Markov model (model the MDP first, then solve the MDP)
  - Solution to the MDP $\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ that best fits the data

$$\hat{\mathcal{P}}_{s,s'}^a = \frac{1}{N(s,a)}\sum_{k=1}^{k}\sum_{t=1}^{T_k}\mathbf{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s,a)}\sum_{k=1}^{k}\sum_{t=1}^{T_k}\mathbf{1}(s_t^k, a_t^k = s, a)r^k$$

  - In the AB example, $V(A) = 0.75$

4.5 Unified View (DP,MC,TD)

Unified View of Reinforcement Learning is shown in Figure. 4.
Comparison between DP, MC and TD, as shown in Figure. 5.

Bootstrapping and Sampling

- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP and TD bootstraps
- Sampling: update samples an expectation
  - MC and TD samples
  - DP does not samples

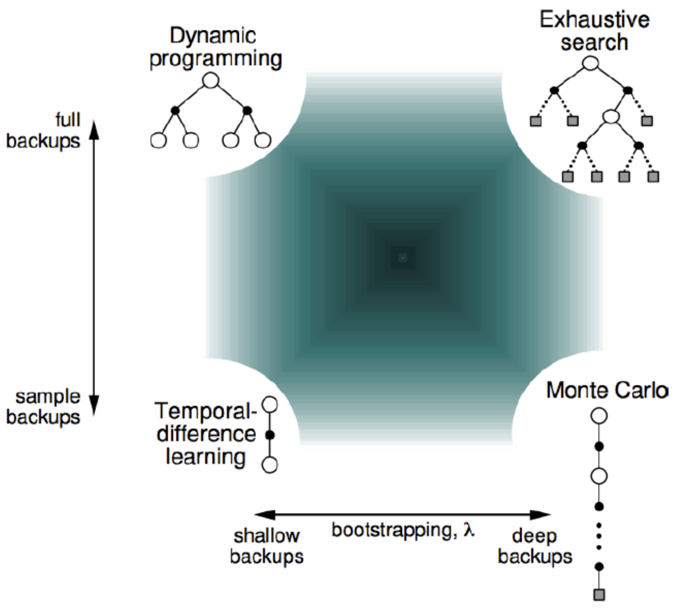


Figure 4: Unified View of Reinforcement Learning.

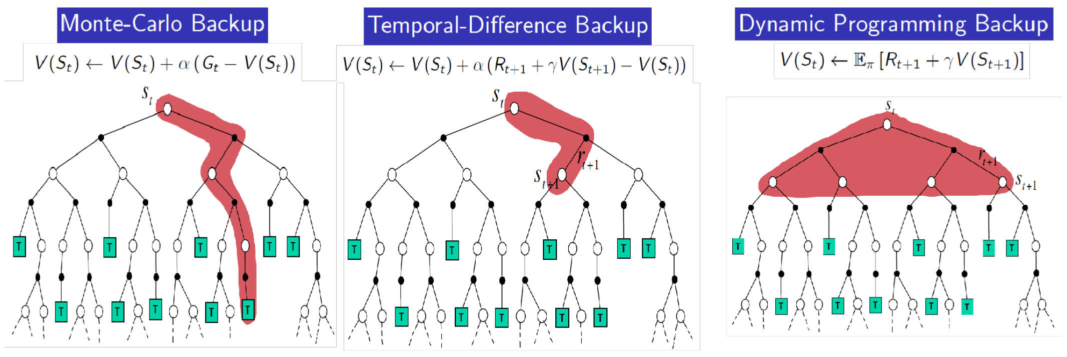

Monte-Carlo Backup
$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$
Temporal-Difference Backup
$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
Dynamic Programming Backup
$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$

Figure 5: Comparison between DP, MC and TD.

4.6 $TD(\lambda)$

4.6.1 n-Step TD

n-Step Prediction: Let TD target look n steps into the future, as shown in Figure.6.
n-Step Return: Consider the following n-step returns for $n = 1, 2, \infty$:

$$\begin{array}{lll} n=1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma V(S_{(t+1)}) \\ n=2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{(t+1)}) \\ \vdots & & \vdots \\ n=\infty & (MC) & G_t^{(1)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T \end{array}$$

Define the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{(t+n)})$$

n-step temporal-difference learning

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{(n)} - V(S_t) \right),$$

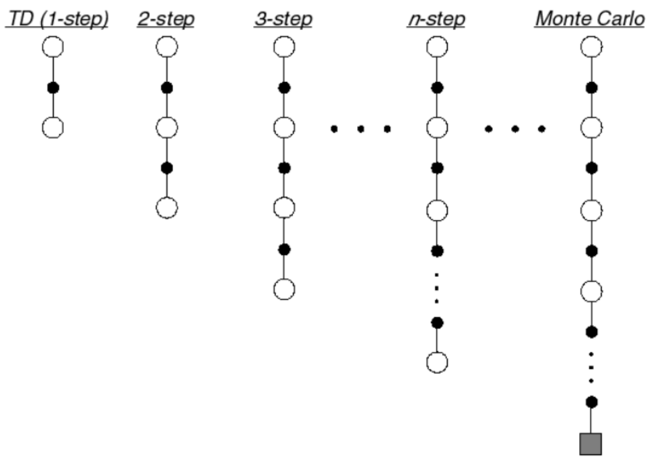


Figure 6: Comparison between DP, MC and TD.

4.6.2 λ-return

Averaging n-Step Returns:
We can average n-step returns over different n, e.g., average the 2-step and 4-step returns

$$\frac{1}{2}G^{(2)} + \frac{1}{4}G^{(4)}$$

Combines information from two different time-steps
Can we efficiently combine information from all time-steps?

λ-return:
The λ-return G_t^{λ} combines all n-step returns $G_t^{(n)}$. Using weight $(1-\lambda)\lambda^{n-1}$ (geometric weighting, whose sum is equal to 1 as shown in the Figure.7)

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

Why use geometric weighting? It is computational efficient, the cost for $TD(0)$ and $TD(\lambda)$ are the same.

4.6.3 Forward-view TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\lambda} - V(S_t)\right).$$

Update value function towards the λ-return. Forward-view looks into the future to compute G_t^{λ}. Like MC, can only be computed from complete episodes.

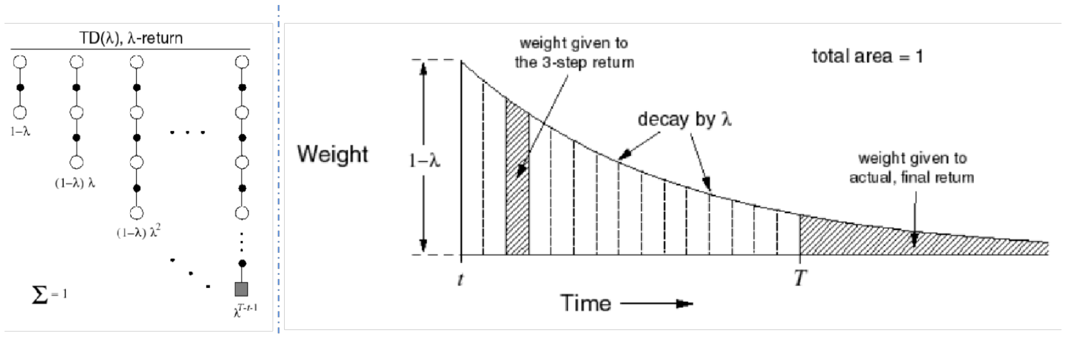


Figure 7: $TD(\lambda)$ Weighting Function (Geometric Weighting).

4.6.4 Backward View TD(λ)

- Forward view provides theory
- Backward view provides mechanism
- Update online, every step, from incomplete sequences

Eligibility Traces
Eligibility traces combine both Frequency and Recency heuristics

- Frequency heuristic: assign credit to most frequent states
- Recency heuristic: assign credit to most recent states

$$\begin{aligned} E_0(s) &= 0 \\ E_t(s) &= \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s) \end{aligned}$$

as shown in Figure.8.

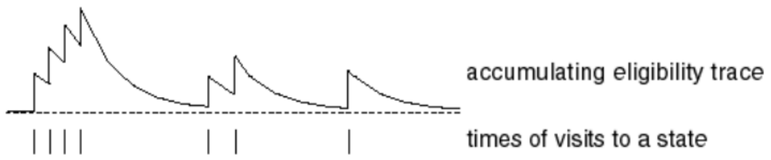


Figure 8: Eligibility Traces.

Backward View TD(λ)
Keep an eligibility trace for every state s. Update value $V(s)$ for every state s (at every time step?). In proportion to TD-error δ_t and eligibility trace $E_t(s)$

$$\begin{aligned} \delta_t &= \underline{R_{t+1} + \gamma V(S_{t+1})} - V(S_t) \\ V(s) &\leftarrow V(s) + \alpha\delta_t \underline{E_t(s)} \end{aligned}$$

where Forward view $TD(\lambda)$ modifies the TD target in the TD error δ_t; Backward view $TD(\lambda)$ adds the eligibility trace.

4.6.5 Relationship for Forward, Backward TD(λ), TD(0) and MC; Online and offline update

$TD(\lambda)$ ***and*** $TD(0)$
When $\lambda = 0$, only current state is updated

$$\begin{aligned} E_t(s) &= \mathbf{1}(S_t = s) \\ V(s) &\leftarrow V(s) + \alpha\delta_t E_t(s) \end{aligned}$$

This is exactly equivalent to $TD(0)$ update

$$V(s) \leftarrow V(s) + \alpha\delta_t$$

$TD(\lambda)$ ***and*** MC
Over the course of an episode, total update for $TD(1)$ is the same as total update for MC.
......

5 Lecture 5: Model-Free Control

5.1 Introduction

- Last lecture:
  - Model-free prediction
  - **Estimate** the value function of an unknown MDP
- This lecture:
  - Model-free control
  - **Optimise** the value function of an unknown MDP

Applicable scenarios for Model-Free Control:

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

On and Off-Policy Learning

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

5.2 On-Policy Monte-Carlo Control

Refresh that there are Policy evaluation and Policy improvement two steps in Generalised Policy Iteration, as shown in Figure.2.
The choice for state value function and action value function in the policy improvement step

- Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \operatorname*{arg\,max}_{a \in \mathcal{A}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over $Q(s,a)$ is model-free

$$\pi'(s) = \operatorname*{arg\,max}_{a \in \mathcal{A}} Q(s,a)$$

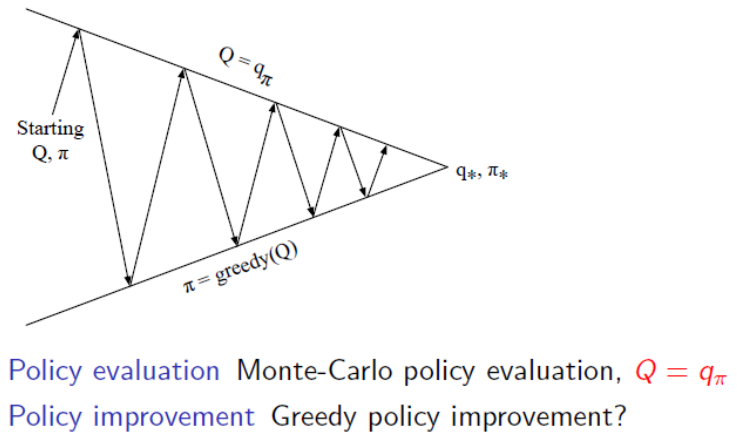
Hence, the Monte-Carlo policy evaluation is applicable in the Generalized Policy Iteration with Action-Value Function (Figure.9), which does not require the model of MDP in the policy improvement step.

Monte Carlo Estimation of Action Values: "The policy evaluation problem for action values is to estimate $q_\pi(s,a)$, the expected return when starting in state s, taking action a, and thereafter following policy π. The Monte Carlo methods for this are essentially the same as just presented for state values, except now we talk about visits to a state–action pair rather than to a state. A state–action pair s, a is said to be visited in an episode if ever the state s is visited and action a is taken in it." Sutton and Barto (2018)

ϵ-Greedy Exploration Simplest idea for ensuring continual exploration, all $|\mathcal{A}(S_t)|$ actions are tried with non-zero probability. With probability $1-\epsilon$ choose the greedy action; With probability ϵ choose an action at random.

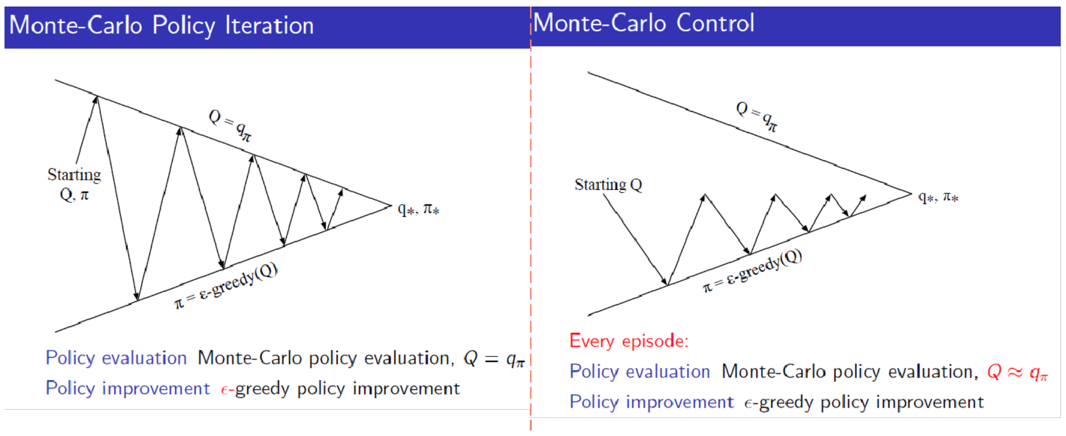
$$\pi(a|s) = \begin{cases} \epsilon/|\mathcal{A}(S_t)| + 1 - \epsilon & if\ a^\star = \operatorname*{arg\,max}_{a \in \mathcal{A}} Q(s,a) \\ \epsilon/|\mathcal{A}(S_t)| & otherwise \end{cases}$$

ϵ-Greedy Policy Improvement



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$
Policy improvement Greedy policy improvement?

Figure 9: Generalized Policy Iteration with Action-Value Function.



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$
Policy improvement ϵ-greedy policy improvement

Every episode:
Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$
Policy improvement ϵ-greedy policy improvement

Figure 10: Monte-Carlo Policy Iteration and Monte-Carlo Control.

Theorem 3 (ϵ-Greedy Policy Improvement). *For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$*

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s,a) \\
&= \epsilon/|\mathcal{A}(S_t)| \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \utilde{\max_{a \in \mathcal{A}} q_\pi(s,a)} \\
&\gtrsim {\color{green}\epsilon/|\mathcal{A}(S_t)|} \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \utilde{\sum_{a \in \mathcal{A}} \frac{\pi(a|a) - {\color{green}\epsilon/|\mathcal{A}(S_t)|}}{1-\epsilon} q_\pi(s,a)} \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) = v_\pi(s)
\end{aligned}
$$

Therefore from policy improvement theorem, $v_{\pi'} \geq v_\pi(s)$.

Monte-Carlo Policy Iteration and Monte-Carlo Control are shown in Figure.10

GLIE

Definition 14 (Greedy in the Limit with Infinite Exploration (GLIE)). *All state-action pairs are explored infinitely many times,*

$$\lim_{k \to \infty} N_k(s,a) = \infty$$

The policy converges on a greedy policy,

$$\lim_{k\to\infty} \pi_k(a|s) = 1(a = \arg\max_{a' \in \mathcal{A}} Q_k(s, a'))$$

For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE Monte-Carlo Control
(Policy evaluation) Sample kth episode using $\pi : \{S_1, A_1, R_2, \ldots, S_T\} \sim \pi$. For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$$

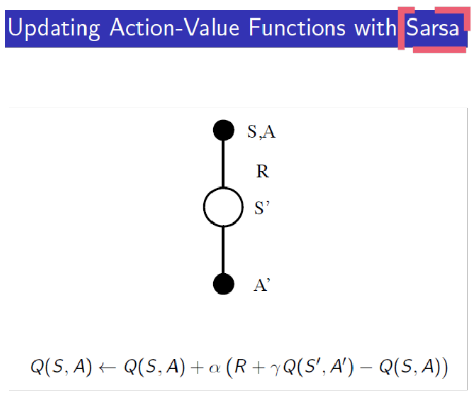
(Policy improvement) Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

5.3 On-Policy Temporal-Difference Learning

5.3.1 Sarsa (on-policy TD control)

Updating Action-Value Functions with Sarsa and On-Policy Control With Sarsa are shown in Figure.11.


Updating Action-Value Functions with Sarsa
S,A
R
S'
A'
$Q(S,A) \leftarrow Q(S,A) + \alpha\,(R + \gamma Q(S',A') - Q(S,A))$

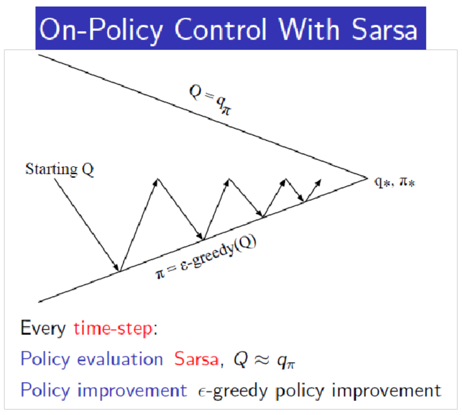

On-Policy Control With Sarsa
$Q = q_\pi$
Starting Q
q_*, π_*
$\pi = \epsilon\text{-greedy}(Q)$
Every time-step:
Policy evaluation Sarsa, $Q \approx q_\pi$
Policy improvement ϵ-greedy policy improvement

Figure 11: Updating Action-Value Functions with Sarsa and On-Policy Control With Sarsa.

Algorithm 1 Sarsa (on-policy TD control) for estimating $Q \approx q_\star$

```
Algorithm parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
for each episode do
  Initialze $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy) comparing with Algorithm.2
  for each step of episode do
    Take action $A$, observe $R, S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'; A \leftarrow A'$;
  end for
end for
```

Theorem 4 (Convergence of Sarsa)**.** *Sarsa converges to the optimal action-value function, $Q(s,a) \to q_\star(s,a)$, under the following conditions:*

- *GLIE sequence of policies $\pi_t(a|s)$*
- *Robbins-Monro sequence of step-sizes α_t*

$$\sum_{t=1}^{\infty} \alpha_t = \infty; \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

5.3.2 Sarsa(λ)

***n*-Step Sarsa** (Comparable with Section.4.6.1): Consider the following n-step returns for $n = 1, 2, \infty$:

$$\begin{array}{lll} n = 1 & (Sarsa) & q_t^{(1)} = R_{t+1} + \gamma Q(S_{(t+1)}) \\ n = 2 & & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{(t+1)}) \\ \vdots & & \vdots \\ n = \infty & (MC) & q_t^{(1)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T \end{array}$$

Define the n-step Q-return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

n-step Sarsa updates $Q(s, a)$ towards the n-step Q-return

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right),$$

Forward View Sarsa(λ):
The q_t^λ return combines all n-step Q-returns $q_t^{(n)}$. Using weight $(1 - \lambda)\lambda^{n-1}$ (geometric weighting, whose sum is equal to 1 as shown in the Figure.7)

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

Forward View Sarsa(λ)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^\lambda - Q(S_t, A_t) \right).$$

Backward View Sarsa(λ):
Just like $TD(\lambda)$, we use eligibility traces in an online algorithm. But $Sarsa(\lambda)$ has one eligibility trace for each state-action pair

$$\begin{aligned} E_0(s) &= 0 \\ E_t(s, a) &= \gamma \lambda E_{t-1}(s, a) + \mathbf{1}(S_t = s, A_t = a) \end{aligned}$$

$Q(s, a)$ is updated for every state s and action a. In proportion to TD-error δ_t and eligibility trace $E_t(s, a)$.

$$\begin{aligned} &\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \\ &Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a) \end{aligned}$$

The pseudo-code is shown in Algorithm???.

5.4 Off-Policy Learning

Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$, while following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

Why is this important?

- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
- Learn about optimal policy while following exploratory policy

- Learn about multiple policies while following one policy

Importance Sampling:
Estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X)\\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X)\\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

Importance Sampling for Off-Policy Monte-Carlo:

1. Use returns generated from μ to evaluate π
2. Weight return G_t according to similarity between policies
3. Multiply importance sampling corrections along whole episode
$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})}\cdots\frac{\pi(A_T|S_T)}{\mu(A_T|S_T)}G_t$$
4. Update value towards corrected return
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\pi/\mu} - V(S_t))$$
5. Cannot use if μ is zero when π is non-zero
6. Importance sampling can dramatically increase variance

Importance Sampling for Off-Policy TD:

1. Use TD targets generated from μ to evaluate π
2. Weight TD target $R + \gamma V(S')$ by importance sampling
3. Only need a single importance sampling correction
$$V(S_t) \leftarrow V(S_t) + \alpha(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1} + \gamma V(S_{t+1})) - V(S_t))$$
4. Much lower variance than Monte-Carlo importance sampling
5. Policies only need to be similar over a single step

5.4.1 Q-Learning: Off-policy TD Control

- We now consider off-policy learning of action-values $Q(s,a)$ [2]
- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\right).$$
(where <u>two actions</u> need to be taken in the control method).

[2]"Off-policy": the optimal action-value function, independent of the policy being followed. (always selecting the action with the largest q-value)

Off-Policy Control with Q-Learning:

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s,a)$

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g., ϵ-greedy w.r.t. $Q(s,a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ =& R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a')) \\ =& R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

Q-Learning Control Algorithm defined as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right).$$

which is shown in Figure.12.

Theorem 5. *Q-learning control converges to the optimal action-value function, $Q(s,a) \to q_\star(s,a)$*

(Comparing with the value iteration of dynamic programming, $v(s) \to v_\star(s)$, in Section 3.4.)

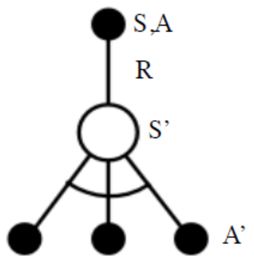


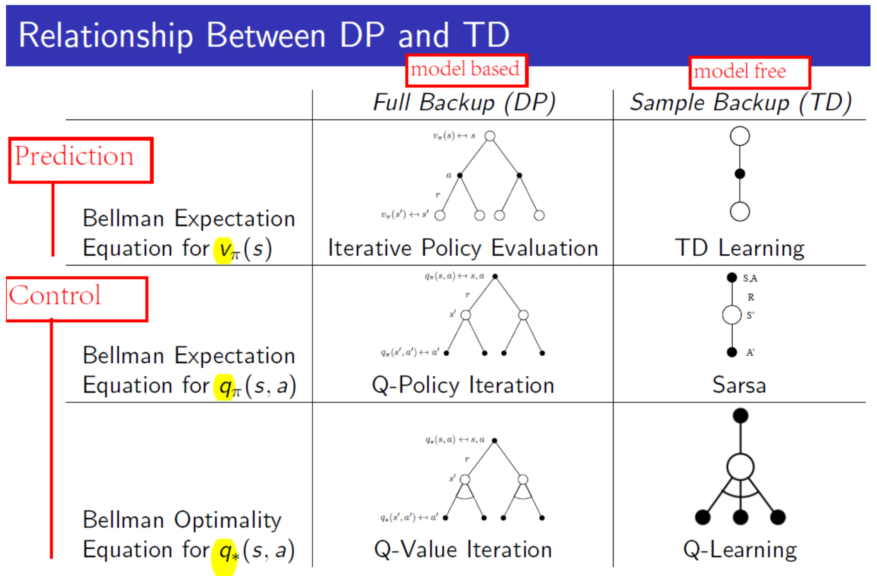
Figure 12: Q-Learning Control Algorithm.

Algorithm 2 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_\star$

```
Algorithm parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
for each episode do
  Initialze $S$
  for each step of episode do
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy) comparing with Algorithm.1
    Take action $A$, observe $R, S'$
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  end for
end for
```

5.5 Summary

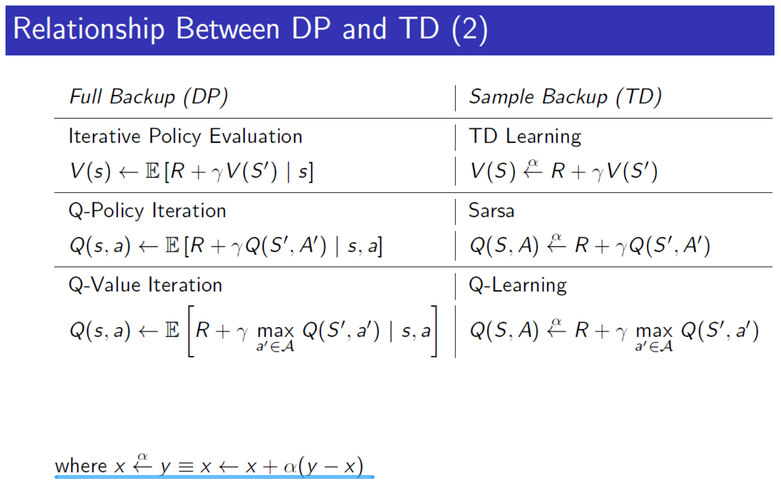
Relationship Between DP and TD is shown in Figure.13 and 14.



Relationship Between DP and TD

	model based *Full Backup (DP)*	model free *Sample Backup (TD)*
Prediction Bellman Expectation Equation for $v_\pi(s)$	Iterative Policy Evaluation	TD Learning
Control Bellman Expectation Equation for $q_\pi(s,a)$	Q-Policy Iteration	Sarsa
Bellman Optimality Equation for $q_*(s,a)$	Q-Value Iteration	Q-Learning

Figure 13: Relationship Between DP and TD(1).



Relationship Between DP and TD (2)

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	TD Learning $V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}[R + \gamma Q(S',A') \mid s,a]$	Sarsa $Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S',A')$
Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	Q-Learning $Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

Figure 14: Relationship Between DP and TD(2).

6 Lecture 6: Value Function Approximation

6.1 Introduction

Large-Scale Reinforcement Learning:
Reinforcement learning can be used to solve large problems, e.g.

- Backgammon: 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

How can we scale up the model-free methods for prediction and control from the last two lectures?

Value Function Approximation:

- So far we have represented value function by a lookup table
  - Every state s has an entry $V(s)$

    - Or every state-action pair s, a has an entry $Q(s,a)$

- Problem with large MDPs
    - There are too many states and/or actions to store in memory
    - It is too slow to learn the value of each state individually
- Solution for large MDPs:
    - Estimate value function (under a policy) with function approximation

$$\begin{aligned} \hat{v}(s,w) &\approx v_\pi(s) \\ \text{or } \hat{q}(s,a,w) &\approx q_\pi(s,a) \end{aligned} \tag{4}$$

    - Generalise from seen states to unseen states
    - Update parameter $\boldsymbol{w}$ using MC or TD learning

Different Types of Value Function Approximation are shown in Figure.15.

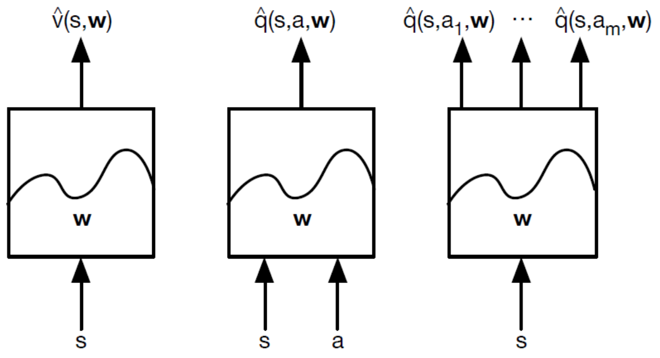


Figure 15:   Types of Value Function Approximation.

Which Function Approximator?
We consider **differentiable** function approximators, e.g.

- **Linear combinations of features**
- **Neural network**
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

Furthermore, we require a training method that is suitable for **non-stationary, non-iid** data

6.2 Gradient Descent

Let $J(\boldsymbol{w})$ be a differentiable function of parameter vector $\boldsymbol{w}$;
Define the gradient of $J(\boldsymbol{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

To find a local minimum of $J(\boldsymbol{w})$;
Adjust $\boldsymbol{w}$ in direction of -ve gradient

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \tag{5}$$

where α is a step-size parameter.

6.3 Incremental Methods

Canonical Form: Value Function Approx. By Stochastic Gradient Descent (Assuming $v_\pi(S)$ is given)
Goal: find parameter vector $\boldsymbol{w}$ minimising mean-squared error between approximate value fn $\hat{v}(s, \boldsymbol{w})$ and true value fn $v_\pi(s)$.

$$J(\boldsymbol{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right] \tag{6}$$

Gradient descent finds a local minimum

$$\begin{aligned} \Delta \boldsymbol{w} &= -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) \right] \end{aligned} \tag{7}$$

Stochastic gradient descent *samples* the gradient

$$\Delta \boldsymbol{w} = \alpha \left(v_\pi(S) - \hat{v}(S, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) \tag{8}$$

Expected update is equal to full gradient update.

6.3.1 Incremental Prediction Algorithms

Have assumed true value function $v_\pi(s)$ given by supervisor. But in RL there is no supervisor, only rewards. In practice, we substitute a target for $v_\pi(s)$

- For MC, the target is the return G_t

$$\Delta \boldsymbol{w} = \alpha \left(G_t - \hat{v}(S, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) \tag{9}$$

- For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) \tag{10}$$

- For forward view TD(λ), the target is the λ-return G_t^λ

$$\Delta \boldsymbol{w} = \alpha \left(G_t^\lambda - \hat{v}(S, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) \tag{11}$$

Monte-Carlo with Value Function Approximation

TD Learning with Value Function Approximation

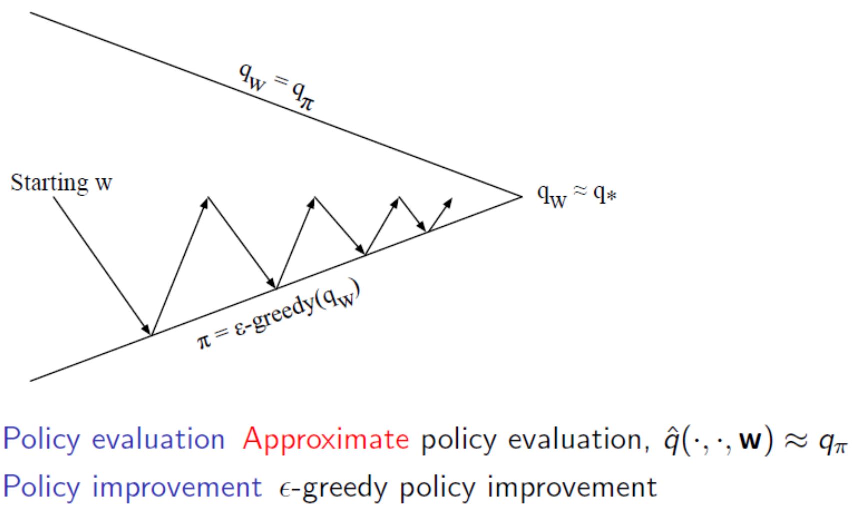
TD(λ) with Value Function Approximation

6.3.2 Incremental Control Algorithms

Control with Value Function Approximation Figure.16.

Action-Value Function Approximation
Approximate the action-value function

$$\hat{q}(S, A, \boldsymbol{w}) \approx q_\pi(S, A)$$



$q_w = q_\pi$
Starting w
$q_w \approx q_*$
$\pi = \varepsilon\text{-greedy}(q_w)$
Policy evaluation Approximate policy evaluation, $\hat{q}(\cdot,\cdot,\mathbf{w}) \approx q_\pi$
Policy improvement ϵ-greedy policy improvement

Figure 16: Control with Value Function Approximation.

Minimise mean-squared error between approximate action-value fn $\hat{q}(S, A, \boldsymbol{w})$ and true action-value fn $q_\pi(S, A)$.

$$J(\boldsymbol{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \boldsymbol{w}))^2 \right] \tag{12}$$

Use stochastic gradient descent to find a local minimum

$$\begin{aligned} \Delta \boldsymbol{w} &= -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \\ &= \alpha \left(q_\pi(S, A) - \hat{q}(S, A, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{q}(S, A, \boldsymbol{w}) \end{aligned} \tag{13}$$

Incremental Control Algorithms
Like prediction, we must substitute a target for $q_\pi(S, A)$

- For MC, the target is the return G_t

$$\Delta \boldsymbol{w} = \alpha \left(G_t - \hat{q}(S_t, A_t, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t, \boldsymbol{w}) \tag{14}$$

- For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w})$

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t, \boldsymbol{w}) \tag{15}$$

- For forward-view TD(λ), the target is the action-value λ-return

$$\Delta \boldsymbol{w} = \alpha \left(q_t^\lambda - \hat{q}(S_t, A_t, \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t, \boldsymbol{w}) \tag{16}$$

- For backward-view TD(λ), equivalent update is

$$\begin{aligned} \delta &= R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w}) \\ E_t &= \gamma \lambda E_{t-1} + \nabla_w \hat{q}(S_t, A_t, \boldsymbol{w}) \\ \Delta \boldsymbol{w} &= \alpha \delta_t E_t \end{aligned}$$

6.3.3 Convergence

On/Off-Policy; MC/TD(0)/TD(λ) ......

6.4 Batch Methods (offline RL)

Gradient descent is simple and appealing, but it is **not sample efficient**.
Batch methods seek to find the best fitting value function, given the agent's experience ("training data").

TIPS – Optimization problem:
Batch Methods: Least Squares Prediction $LS(\boldsymbol{w})$ 17.
Incremental Methods: mean-squared error $J(\boldsymbol{w})$ 12.

Least Squares Prediction
Given value function approximation $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$;
And *experience* $\mathcal{D}$ consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \cdots, \langle s_T, v_T^\pi \rangle\}$$

Which parameters $\boldsymbol{w}$ give the best fitting value fn $\hat{v}(s, \boldsymbol{w})$?
Least squares algorithms find parameter vector $\boldsymbol{w}$ minimising sum-squared error between $\hat{v}(s_t, \boldsymbol{w})$ and target values v_t^π,

$$\begin{aligned} LS(\boldsymbol{w}) &= \sum_{t=1}^{T} \left(v_t^\pi - \hat{v}\left(s_t, \boldsymbol{w}\right)\right)^2 \\ &= \mathbb{E}_{\mathcal{D}}\left[\left(v^\pi - \hat{v}(s, \mathbf{w})\right)^2\right] \end{aligned} \tag{17}$$

Stochastic Gradient Descent with Experience Replay – Steps
Given experience consisting of $\langle state, value \rangle$ pairs;

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \cdots, \langle s_T, v_T^\pi \rangle\}$$

Repeat:

1. Sample state, value from experience
$$\langle s, v^\pi \rangle \sim \mathcal{D}$$
2. Apply stochastic gradient descent update

$$\Delta \boldsymbol{w} = \alpha \left(v_\pi(S) - \hat{v}(S, \boldsymbol{w})\right) \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) \tag{18}$$

Converges to least squares solution
$$\mathbf{w}^\pi = \underset{\boldsymbol{w}}{\operatorname{argmin}} LS(\mathbf{w})$$

<u>Example</u>: Experience Replay in Deep Q-Networks (DQN) DQN uses experience replay and fixed Q-targets

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters $\boldsymbol{w}^-$[3]
- Optimise MSE between Q-network and Q-learning targets
$$\mathcal{L}_i\left(w_i\right) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; w_i^-\right) - Q\left(s, a; w_i\right)\right)^2\right]$$
- Using variant of stochastic gradient descent

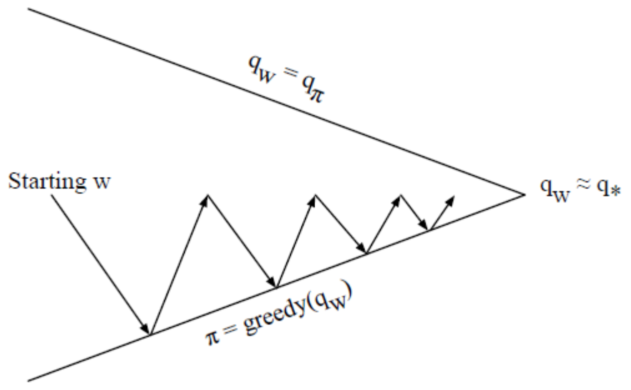
[3]Two neural networks, the one working as target would not be updated at every step

6.4.1 Least Squares Prediction

illustrate in Section 6.5.4, just about substitution.

6.4.2 Least Squares Control

Least Squares Policy Iteration, shown in Figure.17.



Policy evaluation Policy evaluation by least squares Q-learning

Policy improvement Greedy policy improvement

Figure 17: Least Squares Policy Iteration.

Least Squares Action-Value Function Approximation

Least Squares Q-Learning

......

Least Squares Policy Iteration Algorithm

......

6.5 Linear Methods

6.5.1 Feature Vectors

Represent state by a feature vector

$$\boldsymbol{x}(S) = \begin{pmatrix} \boldsymbol{x}_1(S) \\ \vdots \\ \boldsymbol{x}_n(S) \end{pmatrix}$$

For example:

- Distance of robot from landmarks
- Trends in the stock market
- Piece and pawn configurations in chess

6.5.2 Linear Value Function Approximation

Represent value function by a linear combination of features

$$\hat{v}(S, \boldsymbol{w}) = \boldsymbol{x}(S)^\top \boldsymbol{w} = \sum_{j=1}^{n} \boldsymbol{x}_j(S)\boldsymbol{w}_j \tag{19}$$

6.5.3 Applied in incremental methods

Objective function is quadratic in parameters $\boldsymbol{w}$

$$J(\boldsymbol{w}) = \mathbb{E}_\pi \left[\left(v_\pi(S) - \boldsymbol{x}(S)^\top \boldsymbol{w} \right)^2 \right] \tag{20}$$

Stochastic gradient descent converges on global optimum;
Update rule is particularly **simple**

$$\begin{aligned} \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) &= \boldsymbol{x}(S) \\ \Delta \boldsymbol{w} &= \alpha(v_\pi(S) - \hat{v}(S, \boldsymbol{w}))\boldsymbol{x}(S) \end{aligned} \tag{21}$$

Update = step-size × prediction error × feature value

6.5.4 Applied in batch methods

(*Linear Least Squares* ***Prediction***)
Experience replay finds least squares solution, but it may take many iterations.
Using linear value function approximation $\hat{v}(s, \boldsymbol{w}) = \boldsymbol{x}(s)^\top \boldsymbol{w}$, we can solve the least squares solution directly.
At minimum of $LS(\boldsymbol{w})$, the expected update must be zero

$$\begin{aligned} \mathbb{E}_{\mathcal{D}}[\Delta \mathbf{w}] &= 0 \\ \alpha \sum_{t=1}^{T} \mathbf{x}(s_t) \left(v_t^\pi - \mathbf{x}(s_t)^\top \mathbf{w} \right) &= 0 \\ \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^\pi &= \sum_{t=1}^{T} \mathbf{x}(s_t) \mathbf{x}(s_t)^\top \mathbf{w} \\ \mathbf{w} &= \left(\sum_{t=1}^{T} \mathbf{x}(s_t) \mathbf{x}(s_t)^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^\pi \end{aligned} \tag{22}$$

For N features, direct solution time is $O(N^3)$
Incremental solution time is $O(N^2)$ using Shermann-Morrison

Linear Least Squares Prediction Algorithms (Substitution)
We do not know true values v_t^π
In practice, our "training data" must use noisy or biased samples of v_t^π

- LSMC – Least Squares Monte-Carlo uses return $v_t^\pi \approx G_t$
- LSTD – Least Squares Temporal-Difference uses TD target $v_t^\pi \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$
- LSTD(λ) Least Squares TD(λ) uses λ-return $v_t^\pi \approx G_t^\lambda$

In each case solve directly for fixed point of MC / TD / TD(λ).
Just substitute in formulation 22, similar to 6.3.1.

Convergence of Linear Least Squares Prediction Algorithms
......

6.5.5 Linear Action-Value Function Approximation

Represent state and action by a feature vector

$$\boldsymbol{x}(S, A) = \begin{pmatrix} \boldsymbol{x}_1(S, A) \\ \vdots \\ \boldsymbol{x}_n(S, A) \end{pmatrix}$$

Represent action-value fn by linear combination of features

$$\hat{q}(S, A, \boldsymbol{w}) = \boldsymbol{x}(S, A)^\top \boldsymbol{w} = \sum_{j=1}^{n} \boldsymbol{x}_j(S, A)\boldsymbol{w}_j$$

Stochastic gradient descent update

$$\begin{aligned} \nabla_{\boldsymbol{w}} \hat{q}(S, A, \boldsymbol{w}) &= \boldsymbol{x}(S, A) \\ \Delta \boldsymbol{w} &= \alpha(q_\pi(S, A) - \hat{q}(S, A, \boldsymbol{w}))\boldsymbol{x}(S, A) \end{aligned} \tag{23}$$

6.5.6 Table Lookup Features – a special case of linear value function approximation

Table lookup is a special case of linear value function approximation;
Using table lookup features

$$\boldsymbol{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}$$

Parameter vector $\boldsymbol{w}$ gives value of each individual state

$$\hat{v}(S, \boldsymbol{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_n \end{pmatrix}$$

6.6 Summary

7 Lecture 7: Policy Gradient

7.1 Introduction

Policy-Based Reinforcement Learning

In the last lecture we approximated the value or action-value function using parameters θ,

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

A policy was generated directly from the value function, e.g., using ϵ-greedy.
In this lecture we will directly parameterize the policy

$$\pi_\theta(s,a) = \mathbb{P}[a \mid s, \theta]$$

We will focus again on model-free reinforcement learning

We consider methods for learning the policy parameter based on the gradient of some scalar performance measure $J(\theta)$ with respect to the policy parameter. These methods seek to maximize performance, so their updates approximate gradient ascent in J:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)} \tag{24}$$

where $\widehat{\nabla J(\theta_t)} \in \mathbb{R}^{d'}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument θ_t

Value-Based and Policy-Based RL (Figure.18)

- Value Based: Learnt Value Function; Implicit policy (e.g. ϵ-greedy)
- Policy Based: No Value Function; Learnt Policy
- Actor-Critic: Learnt Value Function; Learnt Policy

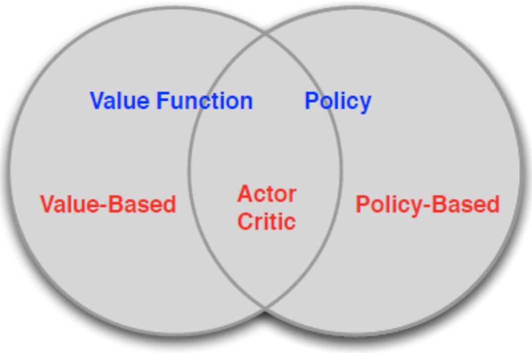


Figure 18: Value-Based and Policy-Based RL.

Advantages of Policy-Based RL

- Advantages:
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

7.2 Policy Approximation and its Advantages

In policy gradient methods, the policy can be parameterized in any way, as long as $\pi(a|s,\boldsymbol{\theta})$ is differentiable with respect to its parameters, that is, as long as $\nabla\pi(a|s,\boldsymbol{\theta})$ (the column vector of partial derivatives of $\pi(a|s,\boldsymbol{\theta})$ with respect to the components of θ) exists and is finite for all $s \in \mathcal{S}, a \in \mathcal{A}(s), \boldsymbol{\theta} \in \mathbb{R}^{d'}$.

Here, we introduce the most common parameterization for discrete action spaces. If the action space is discrete and not too large, then a natural and common kind of parameterization is to form parameterized numerical preferences $h(s,a,\boldsymbol{\theta}) \in \mathbb{R}$ for each state–action pair. The actions with the highest preferences in each state are given the highest probabilities of being selected, for example, according to an **exponential soft-max distribution**:

$$\pi(a|s,\boldsymbol{\theta}) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,a,\theta)}}$$

where e is the base of the natural logarithm. Note that the denominator here is just what is required so that the action probabilities in each state sum to one. We call this kind of policy parameterization **soft-max in action preferences**.

The action preferences themselves can be parameterized arbitrarily. For example, they might be computed by a deep artificial neural network (ANN), where $\boldsymbol{\theta}$ is the vector of all the connection weights of the network. Or the preferences could simply be linear in features,

$$h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}(s,a),$$

using feature vectors $\boldsymbol{x}(s,a) \in \mathbb{R}^{d'}$.

7.3 The Policy Gradient Theorem

The episodic and continuing cases[4] define the performance measure, $J(\boldsymbol{\theta})$, differently and thus have to be treated separately to some extent.

In this section we treat the episodic case, for which we define the performance measure as the value of the start state of the episode. We can simplify the notation without losing any meaningful generality by assuming that every episode starts in some particular (non-random) state s_0. Then, in the episodic case we define performance as

$$J(\boldsymbol{\theta}) = v_{\pi_\theta}(s_0)$$

where v_{π_θ} is the true value function for π_θ, the policy determined by $\boldsymbol{\theta}$.

The ***policy gradient theorem***, which provides an analytic expression for the gradient of performance with respect to the policy parameter. The policy gradient theorem for the episodic case establishes that

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla\pi(a|s,\boldsymbol{\theta}), \tag{25}$$

where the gradients are column vectors of partial derivatives with respect to the components of $\boldsymbol{\theta}$, and π denotes the policy corresponding to parameter vector $\boldsymbol{\theta}$. The symbol $\propto$ here means "proportional to". The distribution μ here (as in Chapters 9 and 10) is the on-policy distribution under π (see page 199).

Proof of the Policy Gradient Theorem (episodic case)......

7.4 REINFORCE: Monte Carlo Policy Gradient

We are now ready to derive our first policy-gradient learning algorithm. Recall our overall strategy of stochastic gradient ascent, which requires a way to obtain samples such that the expectation of the sample gradient is proportional to the actual gradient of the performance measure as a function of the parameter. The sample gradients need only be proportional to the gradient because any constant of proportionality can be absorbed into the step size α, which is otherwise arbitrary. The policy gradient theorem gives an exact expression proportional to the gradient; all that is needed is some way of sampling whose expectation equals or approximates this expression. Notice that the right-hand side of the policy gradient theorem is a sum over states weighted by how often the states occur under the target

[4]more explanations?

policy π; if π is followed, then states will been countered in these proportions. Thus

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$$
$$= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta}) \right].$$

We continue our derivation of REINFORCE by introducing At in the same way as we introduced S_t, by replacing a sum over the random variable's possible values by an expectation under π, and then sampling the expectation. The equation above involves an appropriate sum over actions, but each term is not weighted by $\pi(a|S_t,\boldsymbol{\theta})$ as is needed for an expectation under π. So we introduce such a weighting, without changing the equality, by multiplying and then dividing the summed terms by $\pi(a|S_t,\boldsymbol{\theta})$, we have

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) \quad & \propto \mathbb{E}_\pi \left[\textstyle\sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})} \right] & \\ & = \mathbb{E}_\pi \left[q_\pi(S_t,A_t) \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] & (\text{replacing} a \text{by the sample} A_t \sim \pi) \\ & = \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] & (\text{because} \mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t)) \end{aligned}$$

where G_t is the return as usual. The final expression in brackets is exactly what is needed, a quantity that can be sampled on each time step whose expectation is proportional to the gradient. Using this sample to instantiate our generic stochastic gradient ascent algorithm 24 yields the REINFORCE update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

Note that REINFORCE uses the complete return from time t, which includes all future rewards up until the end of the episode. In this sense **REINFORCE is a Monte Carlo algorithm** and is well defined only for the episodic case with all updates made in retrospect after the episode is completed (like the Monte Carlo algorithms in Chapter 5). This is shown explicitly in Algorithm.3.

Algorithm 3 REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for $\pi_\star$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameters: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)
for each episode do
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  for each step of episode $t = 0, 1, \ldots, T-1$ do
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$
  end for
end for
```

where $\nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$ is the compact expression for the fractional vector $\frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$

7.5 REINFORCE with Baseline

The policy gradient theorem 25 can be generalized to include a comparison of the action value to an arbitrary baseline $b(s)$:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \left(q_\pi(s,a) - b(s) \right) \nabla \pi(a|s,\boldsymbol{\theta}), \tag{26}$$

The baseline can be any function, even a random variable, as long as it does not vary with a; the equation remains valid because the subtracted quantity is zero:

$$\sum_a b(s) \nabla \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla \sum_a \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla 1 = 0$$

The policy gradient theorem with baseline 26 can be used to derive an update rule using similar steps as in the previous section. The update rule that we end up with is a new version of REINFORCE that includes a general baseline:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(G_t - b(S_t)\right) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \tag{27}$$

Because the baseline could be uniformly zero, this update is a strict generalization of REINFORCE. In general, the baseline leaves the expected value of the update unchanged, but it can have a large effect on its variance. *In some states all actions have high values and we need a high baseline to differentiate the higher valued actions from the less highly valued ones; in other states all actions will have low values and a low baseline is appropriate.*

One natural choice for the baseline is an estimate of the state value, $\hat{v}(S_t, \boldsymbol{w})$, where $\boldsymbol{w} \in \mathbb{R}^d$ is a weight vector learned by one of the methods presented in previous chapters. *Because REINFORCE is a Monte Carlo method for learning the policy parameter, $\boldsymbol{\theta}$, it seems natural to also use a Monte Carlo method to learn the state-value weights, $\boldsymbol{w}$.* A complete pseudocode algorithm for REINFORCE with baseline using such a learned state-value function as the baseline is given in Algorithm.4.

Algorithm 4 REINFORCE with Baseline (episodic) for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_\star$

```
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s, \boldsymbol{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0,\ \alpha^{\boldsymbol{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\boldsymbol{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
for each episode do
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  for each step of episode $t = 0, 1, \ldots, T-1$ do
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta \leftarrow G - \hat{v}(S_t, \boldsymbol{w})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \delta \nabla \hat{v}(S_t, \boldsymbol{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
  end for
end for
```

7.6 Actor–Critic Methods

In **REINFORCE with baseline**, the learned state-value function estimates the value of the only the first state of each state transition. This estimate sets a baseline for the subsequent return, but is made prior to the transition's action and thus cannot be used to assess that action. In **actor–critic methods**, on the other hand, the state-value function is applied also to the second state of the transition. The estimated value of the second state, when discounted and added to the reward, constitutes the one-step return, $G_{t:t+1}$, which is a useful estimate of the actual return and thus is a way of assessing the action. As we have seen in the TD learning of value functions throughout this book, the one-step return is often superior to the actual return in terms of its variance and computational congeniality, even though it introduces bias. When the state-value function is used to assess actions in this way it is called a critic, and the overall policy-gradient method is termed an actor–critic method.

First consider one-step actor–critic methods, the analog of the TD methods introduced in Chapter 6 such as TD(0), Sarsa(0), and Q-learning. The main appeal of one-step methods is that they are **fully online and incremental**, yet avoid the complexities of eligibility traces. They are a special case of the eligibility trace methods, but easier to understand. One-step actor–critic methods replace the full return of REINFORCE 27 with the one-step return (and use a learned state-value function as the baseline) as follows:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \hat{v}(S_t, \boldsymbol{w})\right) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \\
&= \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})\right) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \\
&= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}
\end{aligned} \tag{28}$$

......

- MC methods: REINFORCE; REINFORCE with baseline
- TD methods: Actor-Critic Methods

8 Notes on Deep Deterministic Policy Gradient (DDPG)

Policy-Based Reinforcement Learning

9 Chapter 2 Introduction to Reinforcement Learning

9.1 2.1 Introduction

9.2 2.2 Bandits

9.3 2.3 Markov Decision Process

9.4 2.4 Dynamic Programming

9.5 2.5 Monte Carlo

9.6 2.6 Temporal Difference Learning

9.7 2.7 Policy Optimization

9.7.1 2.7.1 Overview

9.7.2 2.7.2 Value-Based Optimization

9.7.3 2.7.3 Policy-Based Optimization

Gradient-Based Optimization

Stochastic Policy Gradient

Lemma 1 (Log-Derivative Trick).

$$\nabla_\theta \log p(\tau|\theta) = \frac{\nabla_\theta p(\tau|\theta)}{p(\tau|\theta)} \Rightarrow \nabla_\theta p(\tau|\theta) = p(\tau|\theta)\nabla_\theta \log p(\tau|\theta) \tag{29}$$

Lemma 2 (Probability of Trajectory). *Considering a parameterized stochastic policy $\pi_\theta(a|s)$, we then have the probability of trajectory*

$$p(\tau|\pi) = \rho_0(S_0)\prod_{t=0}^{T} p(S_{t+1}|S_t, A_t)\pi(A_t|S_t), \tag{30}$$

for MDP process with $\rho_0(S_0)$ as initial state distribution, we can get the logarithm of the probability of trajectory with parameterized policy π_θ as:

$$\log p(\tau|\theta) = \log \rho_0(S_0) + \sum_{t=0}^{T} \left(\log p(S_{t+1}|S_t, A_t) + \log \pi_\theta(A_t|S_t)\right). \tag{31}$$

Therefore we can get the derivative of the log-probability of a trajectory as:

$$\begin{aligned}\nabla_\theta \log p(\tau|\theta) &= \nabla_\theta \log \rho_0(S_0) + \sum_{t=0}^{T} \left(\nabla_\theta \log p(S_{t+1}|S_t, A_t) + \nabla_\theta \log \pi_\theta(A_t|S_t)\right) \\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t)\end{aligned} \tag{32}$$

where the terms containing $g\rho_0(S_0)$ and $p(S_{t+1}|S_t, A_t)$ are removed because they do not depend on parameters θ, although unknown.

Lemma 3 (Exchanging between Expectation and Sum and Derivative).

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R_t\right] = \sum_{t=0}^{T} \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R_t], \tag{33}$$

producing two ways to calculate the gradients

Lemma 4 (Expected Grad-Log-Prob (EGLP) – which is commonly used in policy gradient optimization, introducing baseline in the gradient,e.g., REINFORCE with Baseline; Actor Critic Methods). *Suppose that p_θ is a parameterized probability distribution over a random variable, x. Then:*

$$\mathbb{E}_{x\sim p_\theta}\left[\nabla_\theta \log P_\theta(x)\right] = 0 \tag{34}$$

Proof. Recall that all probability distributions are normalized:

$$\int_x p_\theta(x) = 1 \tag{35}$$

Take the gradient of both sides of the normalization condition:

$$\nabla_\theta \int_x p_[\theta](x) = \nabla_\theta 1 = 0. \tag{36}$$

Use the log derivative trick to get:

$$\begin{aligned} 0 &= \nabla_\theta \int_x p_\theta(x) \\ &= \int_x \nabla_\theta(x) \\ &= \int_x p_\theta \nabla_\theta \log p_\theta(x) \\ \text{Hence, } 0 &= \mathbb{E}_{x\sim p_\theta}\left[\nabla_\theta \log p_\theta(x)\right] \end{aligned} \tag{37}$$

Learning objective: Recall that the **learning objective** is to maximize the expected cumulative reward:

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R_t\right] = \sum_{t=0}^{T} \mathbb{E}_{\tau\sim\pi_\theta}[R_t] \tag{38}$$

where $\tau = (S_0, A_0, R_0, \ldots, S_T, A_T, R_T, S_{T+1})$ and $R(\tau) = \sum_{t=0}^{T} R_t$. We can directly perform gradient ascent on the parameters of the policy θ to gradually improve the performance of the policy π_θ.

There are two ways to calculate the gradient
Taking the expectation of the reward at a certain time step (lemma.3)
Note that R_t only depends on τ_t, where $\tau_t = (S_0, A_0, R_0, \ldots, S_t, A_t, R_t, S_{t+1})$.

$$\begin{aligned} \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R_t] &= \nabla_\theta \int_{\tau_t} R_t p(\tau_t|\theta) d\tau_t && \text{Expand expectation} \\ &= \int_{\tau_t} R_t \nabla_\theta p(\tau_t|\theta) d\tau_t && \text{Exchange gradient and integral} \\ &= \int_{\tau_t} R_t p(\tau_t|\theta) \nabla_\theta \log p(\tau_t|\theta) d\tau_t && \text{Log-derivative trick} \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[R_t \nabla_\theta \log p(\tau_t|\theta)] && \text{Return to expectation form} \end{aligned} \tag{39}$$

Plug the above formula back to $J(\pi_\theta)$38,

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R_t \nabla_\theta \log p(\tau_t|\theta)\right]. \tag{40}$$

where $\nabla_\theta \log p(\tau_t|\theta)$ can be easily derived by replacing the $\tau = \tau_{0:T}$ in lemma.2 to be $\tau_t = \tau_{0:t}$, which gives:

$$\nabla_\theta \log p(\tau_t|\theta) = \sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}). \tag{41}$$

Therefore,

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R_t \nabla_\theta \sum_{t'=0}^{t} \log \pi_\theta(A_{t'}|S_{t'})\right] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t`=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \sum_{t=t'}^{T} R_t\right]. \end{aligned} \tag{42}$$

Here the last equality is simply by rearranging the summation.

Taking the expectation of the cumulative reward along the whole trajectory: (lemma.3)

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} R(\tau) \\
&= \nabla_\theta \int_\tau p(\tau|\theta) R(\tau) d\tau && \text{Expand expectation} \\
&= \int_\tau \nabla_\theta p(\tau|\theta) R(\tau) d\tau && \text{Exchange gradient and integral} \\
&= \int_\tau p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) R(\tau) d\tau && \text{Log-derivative trick} \\
&= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log p(\tau|\theta) R(\tau)] && \text{Return to expectation form}
\end{aligned} \tag{43}$$

We may have

$$\begin{aligned}
\Rightarrow \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t) \sum_{t'=0}^{T} R_{t'}\right]
\end{aligned} \tag{44}$$

A careful reader may notice that the second result in Eq.44 is slightly different from the first result as in Eq.42. Specifically, the time scales of the cumulative reward are different. The first result uses only the cumulative future rewards $\sum_{t=t'}^{T} R_t$ after action A_t to evaluate the action, while the second result uses the cumulative rewards on the whole trajectory $\sum_{t=0}^{T} R_t$ to evaluate each action A_t on that trajectory, including the rewards before choosing that action. Those past rewards can be simply dropped in the derived policy gradient to have Eq.42, which is called the "reward-to-go" policy gradient. The equivalence of the two (Eq.44 and 42) has been proved [5].

We also notice that the expectation in Eq.44 can be estimated with the sample mean. If we collect a set of trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ where each trajectory is obtained by letting the agent act in the environment using the policy π_θ the policy gradient can be estimated with

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau\in\mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t) R(\tau) \tag{45}$$

From the EGLP lemma.4 we can directly derive that:

$$\mathbb{E}_{A_t\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A_t|S_t) b(S_t)\right] = 0 \tag{46}$$

where $b(S_t)$ is called a baseline and is independent of the future trajectory the expectation is taken over. The baseline is any function dependent only on the currents state, without affecting the overall expected value in the optimization formula.

Could b be a constant?

In the above formulas the optimization goal is finally (from Eq.44):

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t`=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) R(\tau)\right]. \tag{47}$$

We can also modify the reward for total trajectory $R(\tau)$ to be reward-to-go G_t following time step t (from Eq.42):

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t`=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) G_t\right]. \tag{48}$$

With the EGLP lemma.4, the expected return can be generalized to be:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t`=0}^{T} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \Phi_t\right]. \tag{49}$$

[5] Proof of equivalence of two versions of stochastic policy gradient: https://spinningup.openai.com/en/latest/spinningup/extra_pg_proof1.html.

where $\Phi_t = \sum_{t'=t}^{T}(R(S_{t'}, a_{t'}, S_{t'+1}) - b(S_t))$ [6].

Actually ϕ_t could be the following formats for more practical usage:

$$\Phi_t = Q^{\pi_\theta}(S_t, A_t) \tag{50}$$

or,

$$\Phi_t = A^{\pi_\theta}(S_t, A_t) = Q^{\pi_\theta}(S_t, A_t) - V^{\pi_\theta}(S_t) \tag{51}$$

which are both proven to be identical to the original format in the expected value, just with different variances in practice. The proof of these requires law of iterated expectations: $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]$ for two random variables (discrete or continuous). And this is easy to prove. The rest of the proof is given below:

Proof.......

Example: REINFORCE Algorithm

REINFORCE is an algorithm using stochastic policy gradient method as in Eq.49, where $\Phi_t = Q^\pi(S_t, A_t)$ and it is estimated with sampled rewards along the trajectory $G_t = \sum_{t'=t}^{\infty} R_{t'}$ (or discounted version $G_t = \sum_{t=t'}^{\infty} \gamma^{t'-t} R_{t'}$) in REINFORCE. The gradients for updating the policy are:

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty}\sum_{t'=t}^{\infty} R_{t'} \nabla_\theta \log \pi_\theta(A_t|S_t)\right] \tag{52}$$

Deterministic Policy Gradient

What has been described above belongs to stochastic policy gradient (SPG), and it works for optimizing the stochastic policy $\pi(a|s)$, which represents the action as a probabilistic distribution based on the current state. The contrary case to the stochastic policy is the deterministic policy, where $a = \pi(s)$ is a deterministic action instead of probability [7].

The **performance objective for the deterministic policy** following the same expected discounted reward definition in stochastic policy gradient:

$$\begin{aligned} J(\mu) &= \mathbb{E}_{S_t \sim \rho^\mu, A_t = \mu(S_t)}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R(S_t, A_t)\right] \\ &= \int_{\mathcal{S}}\int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} \rho_0(s) p(s' \mid s, t, \mu) R(s', \mu(s'))] \mathrm{d}s\, \mathrm{d}s' \\ &= \int_{\mathcal{S}} \rho^\mu(s) R(s, \mu(s)) \mathrm{d}s \end{aligned} \tag{53}$$

where $p(s' \mid s, t, \mu) = p(S_{t+1}|S_t, A_t)p^\mu(A_t|S_t)$, the first probability is the transition probability and the second is the probability of the action choice. Since it is deterministic policy, we have $p^\mu(A_t|S_t) = 1$ and therefore $p(s'|s, t, \mu) = p(S_{t+1}|S_t, \mu(S_t))$. Also, the state distribution in above formula is $\rho^\mu(s') := \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} \rho_0(s) p(s'|s, t, \mu) \mathrm{d}s$

Another way for defining performance objective. As the value function could be defined as

$$V^\mu(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R(S_t, A_t) | S_1 = s; \mu\right] = \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p(s'|s, t, \mu) R(s', \mu(s')) \mathrm{d}s' \tag{54}$$

following the same definition in stochastic policy gradient except for applying the deterministic policy, we can also derive that

$$\begin{aligned} J(\mu) &= \int_{\mathcal{S}} \rho_0(s) V^\mu(s) \mathrm{d}s \\ &= \int_{\mathcal{S}}\int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} \rho_0(s) p(s' \mid s, t, \mu) R(s', \mu(s'))] \mathrm{d}s\, \mathrm{d}s' \end{aligned} \tag{55}$$

which is the same as the above representation 53 directly using discounted rewards. The relationships here also hold for stochastic policy gradient, just with the deterministic policy $\mu(s)$ replaced by the stochastic policy $\pi(a|s)$.

[6] for $b(S_t)$, according to lemma.4,the integral for the expression including this item is zero

[7] In the following part of this section, we use $\mu(s)$ instead of $\pi(s)$ as previously defined to represent the deterministic policy

For deterministic policy, we have $V^{\mu}(s) = Q^{\mu}(s, \mu(s))$ as the Q-value is an expectation over the action distribution for stochastic policy, but there is no action distribution but a single value for the deterministic policy. Therefore we also have the following representation for deterministic policy,

$$\begin{aligned} J(\mu) &= \int_{\mathcal{S}} \rho_0(s) V^{\mu}(s) \mathrm{d}s \\ &= \int_{\mathcal{S}} \rho_0(s) Q^{\mu}(s, \mu(s)) \mathrm{d}s \end{aligned} \tag{56}$$

The different formats of performance objective will be used in the proof of DPG theorem, as well as several conditions. We list the conditions here without a detailed derivation process, which can be checked in the original paper by Silver et al. (2014):

- **C.1 The Existence of Continuous Derivatives**: $p(s'|s,a), \nabla_a p(s'|s,a), \mu_\theta(s), \nabla_\theta \mu_\theta(s), R(s,a), \nabla_a R(s,a), \rho_0(s)$ are continuous in all parameters and variables s, a, s' and x.
- **C.2 The Boundedness Condition**: there exist a, b, and L such that $\sup_s \rho_0(s) < b, \sup_{a,s,s'} p(s'|s,a) < b, \sup_{a,s} R(s,a) < b, \sup_{a,s,s'} \|\nabla_a p(s'|s,a)\| < L, \sup_{a,s} \|\nabla_a R(s,a)\| < L$.

Theorem 6 (Deterministic Policy Gradient)**.** *suppose that the MDP satisfies conditions C.1 for the existence of* $\nabla_\theta \mu_\theta(s)$, $\nabla_a Q^{\mu}(s,a)$ *and the deterministic policy gradient, then,*

9.7.4 2.7.4 Combination of Policy-Based and Value-Based Methods

10 Chapter 3 Taxonomy of Reinforcement Learning Algorithms

10.1 3.1 Model-Based and Model-Free

10.2 3.2 Value-Based and Policy-Based

10.3 3.3 Monte Carlo and Temporal Difference

10.4 3.4 On-Policy and Off-Policy

11 Fall 2019

This line is red

and this line in yellow

Back to red !

11.1 Fri, Sept 6: Phenomenology of Microscopic Physics

- Newtonian mechanics (i.e., $\mathbf{F} = m\mathbf{a}$) is an excellent theory; it applies to the vast majority of human-scale (and even interplanetary-scale) physics.

- Apart from relativistic effects at very high velocities (special relativity) or in very strong gravitational fields (general relativity), Newtonian mechanics accurately describes a huge range of phenomena, but around the end of the Nineteenth Century people became aware of some physical effects for which there is no sensible Newtonian explanation.
- Examples include:
  - the **double slit experiment** (done with light by Thomas Young in 1801, and with electrons by Tonomura in 1986)
  - the photoelectric effect (analyzed by Einstein in 1905 — in fact his Nobel-winning work)
  - the "quantum Venn diagram" puzzle, involving the overlaps of three polarizing filters
  - the stability of the hydrogen atom (i.e., the fact that the electron doesn't lose energy and spiral inward toward the proton).
  - Test Levine et al. (2020)
  - equation 57

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{H} \gamma^t \mathcal{R}(s_t, a_t) \right]. \tag{57}$$

Remark 1. *How now, brown cow?*

Definition 15. *The* Feynman kernel *is given by*

$$K(x_b, t_b; x_a, t_a) = \int_{x(t_a)=x_a}^{x(t_b)=x_b} e^{(i/\hbar)S[x(t)]}\, \mathcal{D}x(t).$$

11.2 Mon, Sept 9: Review of Newtonian Mechanics

- A *Newtonian trajectory* $\mathbf{x}(t)$ ($t \in \mathbb{R}$) is given by solutions of the second order ODE

$$m\,\ddot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t)),$$

  where $m > 0$ is a basic parameter associated with a given Newtonian particle, called its *mass*.

- The force field $\mathbf{F}(\mathbf{x})$ — which we take to be static (i.e., not intrinsically dependent on time) for simplicity — is said to be *conservative* if there is a *potential function* $V(\mathbf{x})$ such that

$$\mathbf{F}(\mathbf{x}) = -\nabla V(\mathbf{x}).$$

  Here, '∇' denotes the *gradient operator*,

$$\nabla V = \left(\frac{\partial V}{\partial x}, \frac{\partial V}{\partial y}, \frac{\partial V}{\partial z} \right).$$

- For a conservative force field, we can find a *conserved quantity* along the Newtonian trajectories, namely the *total mechanical energy*.

$$E = H(\mathbf{x}, \mathbf{p}) := \frac{1}{2m}\mathbf{p}^2 + V(\mathbf{x}).$$

  Here, $\mathbf{p}^2 := \mathbf{p} \cdot \mathbf{p} = \|\mathbf{p}\|^2$, and $\mathbf{p} := m\mathbf{v} := m\dot{\mathbf{x}}$ is the *momentum*.

11.3 Tue, Sept 10: Alternative Formulations of Newtonian Mechanics

- The **Hamiltonian formulation**:
$$\dot{\mathbf{x}} = \frac{\partial H}{\partial \mathbf{p}}, \qquad \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{x}}.$$
- The **Lagrangian formulation**:
$$\delta S[\mathbf{x}(t)] = 0,$$
where the *action* on the time interval $[t_a, t_b]$ is given by
$$S[\mathbf{x}(t)] := \int_{t_a}^{t_b} \left[\frac{m}{2}\,\dot{\mathbf{x}}(t)^2 - V(\mathbf{x}(t))\right] dt.$$
- Etc.

References

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643.*

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

12 Spring 2020