# Reinforcement Learning Notes

# Lec 1 Introduction

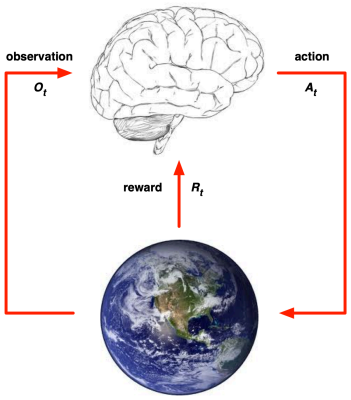
## The RL settings

### Rewards

- A reward $R_t$ is a **scalar** feedback signal
- **Reward Hypothesis**:
  - All goals can be described by the maximization of expected culumative reward
  - Notes:

        - **maximization**
        - **Expectation**
        - **Cumulative**
        - How to deal with multiple goals?
- Sequential Decision Making: to maximize total future reward
    - Reward may be delayed, which implies that sometimes immediate reward may be sacrificed got more long-term reward

## Agent and Environment




At each timestep $t$

the agent:

- Do action $a_t$
- Observe $o_t$
- Receice $r_t$

the env:

- Receive $a_t$
- Emits $o_t$ and $r_t$

## State

**History**:

- the sequence of observations, actions, rewards

  $o_1, r_1, a_1, o_2, r_2, a_2, \cdots, o_t, r_t$
- The agent uses history to decide $a_t$

**State**:

- Generally, the agent don't use all the information in the history to make decisions
- **State** is the information used to determine what happens next
- i.e. The state is a **sufficient statistic** for the future
- Formally, state is a function of the history
  - $S_t = f(H_t)$

**Environment State and Agent State**:

- environment state $S_t^e$: environment's private representation
    - usually not visible to the agent
- agent state $S_t^a$: agent's internal representation

**Information State**:

- contains all useful information from the history
- aka **Markov State**
    - Given $S_t$, the future is independent of the history $H_t$
- Once the state is known, the history may be thrown away

**Observability**:

- Fully Obervation Environment:
    - $O_t = S_t^a = S_t^e$
    - **Markov Decision Process(MDP)**
- Partial observability:
    - **POMDP**
    - Agent must construct its own state representation
    - How?
        - Using complete history
        - Using **beliefs** of environment state

        - Using RNN

## RL agent

#### major components

- Policy: makes decision
- Value function: evaluates each state and/or action
- Model: model of the environment

#### Policy

- Tells the agent how to behave
- maps from state to action
    - Deterministic: $a_t = \pi(s_t)$
    - Stochastic: $\pi(a|s) = P(A_t = a|S_t = s)$

#### Value Function

- Evaluate states/actions
- State value functions:

$$v_\pi(s) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right] \tag{1}$$

- State-action value functions:

$$v(a|s) = \mathbb{E} \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid A_t = a, S_t = s \right] \tag{2}$$

**Model**:

- the model of environment interaction
- consists of two parts
  - $\mathcal{P}$: predicts the state, i.e. state transision probability matrix
  - $\mathcal{R}$: predict the reward, i.e. the reward distribution

**Example: Maze**

**Categorizing RL agents**

- With/without Model: Model based/model free
- Value based: value function and no policy
- Policy based: policy and no value function
- Actor-Critic: value function and policy

# Problems within RL

## Learning and planning

- Learning:
    - the model is unknown
    - interacts with the environment while improving its policy
- Planning
    - model is known
    - do not interact with the environment while improving its policy

## Exploration and Exploitation

- Reinforcement learning is like trial-and-error
- The agent should learn about the environment as well as gain more reward.
- Exploration finds more information about the environment
- Exploitation uses known information to maximise reward

## Prediction and Control

- Prediction: predict the state given a policy

- Control: find the best policy given the state

# Lec 2 MDP

**Why do we study Markov Decision Process in Reinforcement Learning?**

- Markov Property is a **good** property for RL
    - Simplify the history with the current state
- When the environment is fully observable, it can be described as a Markov Decision Process
- When it is partially observable, it can be converted into MDPs(POMDPs).

# Markov Process

## Markov Property

- The future is independent of the past given the present

A state $S_t$ is Markov if and only if

$$\mathbb{P}\left[S_{t+1} \mid S_t\right] = \mathbb{P}\left[S_{t+1} \mid S_1, \ldots, S_t\right] \tag{3}$$

## Markov Process

Markov Process (or Markov Chain) is a tuple $< \mathcal{S}, \mathcal{P} >$

- $\mathcal{S}$: state space
- $\mathcal{P}$: state transition probability matrix

## Markov Reward Process

Markov Reward Process is a tuple $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$: state space
- $\mathcal{P}$: state transition probability matrix
- $\mathcal{R}$: reward function. $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- $\gamma$: discount factor

**Return** $G_t$: the total discounted reward from time-step $t$

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4}$$

Why Discount?

- Mathematically: for convergence and convenience
- Financially: interest rate
- Uncertainty about the future and preference for immediate reward

Also, for episodic MDPs, $\gamma$ can be 1.

#### Value Function

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right] \tag{5}$$

# Markov Decision Process

## Definition

Markov Decision Process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$: state space
- $\mathcal{A}$: action space
- $\mathcal{P}$: state transition probability matrix
  $\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$: reward function. $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- $\gamma$: discount factor

#### Policy

Policy $\pi$ is a distribution over actions given states.

$$\pi(a|s) = P(A_t = a|S_t = s) \tag{6}$$

With a fixed policy, MDP is reduced to Markov Reward Process $< \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma >$

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

## Value Function and Bellman Equation

state-value function

$$v_\pi(s) = \mathbb{E}_\pi \left[ G_t \mid S_t = s \right] \tag{7}$$

action-state value function

$$q_\pi(s, a) = \mathbb{E}_\pi \left[ G_t \mid S_t = s, A_t = a \right] \tag{8}$$

Relationship between $v_\pi(s)$ and $q_\pi(s, a)$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) \tag{9}$$

**Bellman Expectation Equation**

Because

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi \left( s' \right) \tag{10}$$

Using (7) (8) we have

$$\begin{aligned} v_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi \left( s' \right) \right) \\ q_\pi(s, a) &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi \left( a' \mid s' \right) q_\pi \left( s', a' \right) \end{aligned} \tag{11}$$

the matrix form

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi \tag{12}$$

## Optimal Policy and Value Function

- Define **Optimal**
    - Define a partial ordering over policies $\pi > \pi'$ if
      $v_\pi(s) \geqslant v_{\pi'}(s), \forall s \in \mathcal{S}$

- Theorem: For any Markov Decision Process
  - There exists an optimal policy
  - All optimal policies achieve the optimal value function and action-value function
  - There is always a deterministic optimal policy, that is
    -

$$v_*(s) = \max_a q_*(s, a) \tag{13}$$

    -
    -

$$\pi_*(a \mid s) = \begin{cases} 1 & \text{if } a = \operatorname*{argmax}_{a \in \mathcal{A}} q_*(s, a) \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Bellman Optimality Function

$$\begin{aligned} v_*(s) &= \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*\left(s'\right) \\ q_*(s, a) &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*\left(s', a'\right) \end{aligned} \tag{15}$$

## Extensions to MDPs

- Infinite MDPs
  - Countably infinite state and/or action spaces

    - Continuous state and/or action spaces
        - Closed form for linear quadratic model (LQR)
    - Continuous time
        - Partial Differential Equations
        - Hamilton-Jacobi-Bellman Equation
        - Limiting case of Bellman Equation as $\Delta t \to 0$
- POMDP(Partially Oberservable Markov Decision Process)
    - POMDP is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma >$
        - $\mathcal{S}$: state space
        - $\mathcal{A}$: action space
        - $\mathcal{O}$: observation space
        - $\mathcal{P}$: state transition probability matrix
          $\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
        - $\mathcal{R}$: reward function. $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
        - $\mathcal{Z}$ : obervation function
          $\mathcal{Z}^a_{s'o} = \mathbb{P}\left[O_{t+1} = o \mid S_{t+1} = s', A_t = a\right]$
        - $\gamma$: discount factor
    - Belief States
        - History $H_t$: $H_t = A_0, O_1, R_1, \ldots, A_{t-1}, O_t, R_t$
        - A belief state $b(h)$ is a probability distribution over states conditioned on the history $h$,
          $b(h) = \left(\mathbb{P}\left[S_t = s^1 \mid H_t = h\right], \ldots, \mathbb{P}\left[S_t = s^n \mid H_t = h\right]\right)$
        - The history $H_t$ satisfies the Markov Property
        - The belief state $b(H_t)$ satisfies the Markov property

    - So POMDP can be reduced to an history tree or a belief state tree
- Ergodic MDP
    - An ergodic Markov process has a stationary distribution $d^{\pi}(s)=\sum_{s' \in \mathcal{S}} d^{\pi}\left(s'\right) \mathcal{P}_{s's}$
    - An MDP is ergodic if the Markov chain induced by any policy is ergodic.
    - Average reward $\rho^{\pi}=\lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}\left[\sum_{t=1}^{T} R_t\right]$

# Lecture 3 Dynamic Programming

# Recap: MDP

## Markov Process

**A Markov Process ( or Markov Chain ) is a tuple** $<\mathcal{S}, \mathcal{P}>$

### Markov Property

$$\mathbb{P}\left[S_{t+1} \mid S_t\right]=\mathbb{P}\left[S_{t+1} \mid S_1, \ldots, S_t\right] \tag{16}$$

### Transition Matrix

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right] \tag{17}$$

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

## Markov Reward Process

Add a reward function in a markov process

### Definition

**A Markov Reward Process is a tuple** $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- $\gamma$ is a discount factor

Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{18}$$

Value Function:

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right] \tag{19}$$

### Bellman Equation

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

$$
\mathbf{v} = \mathcal{R} + \gamma \mathcal{P} \mathbf{v} \tag{20}
$$

## Markov Decision Process

Add actions in a Markov Decision Process

### Definition

**A Markov Decision Process is a tuple** $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{A}$ is a finite set of actions, which determines the transition probability and the reward
- $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$

Policy:

- $\pi \in \Pi = \{\pi : \mathcal{S} \mapsto \mathcal{A}\}$
- $\pi(a \mid s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$

Given $\pi$,  MDP is a Markov Decision Process with $< \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma >$

In the same way, we can define the **value function.**

$$v_{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s\right] \tag{21}$$

Moreover, we can define the **action-value function**

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s, A_t = a\right] \tag{22}$$

The relationship between value function and action-value function

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_{\pi}(s, a) \tag{23}$$

## Bellman Expectation Equation

Substitute $\mathbf{v}, \mathcal{R}, \mathcal{P}$ with $\mathbf{v}^{\pi}, \mathcal{R}^{\pi}, \mathcal{P}^{\pi}$

$$\mathbf{v}^{\pi} = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} \mathbf{v}^{\pi} \tag{24}$$

## Optimal Policy and Value Function

- How to evaluate a state?
    - Its value function
- How to evaluate a policy?
    - Its value function of every state

Define a partial ordering over policy

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s \tag{25}$$

- There exists an optimal policy $\pi_*$
- All optimal policies achieve the optimal value function and action-value function
- There is always a deterministic optimal policy.

## Bellman Optimality Equation

$$v_*(s) = \max_a q_*(s,a) \tag{26}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*\left(s'\right) \tag{27}$$

To seperate $v(s)$ and $q(s,a)$

$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_* \left(s'\right) \tag{28}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_* \left(s', a'\right) \tag{29}$$

Non-linear and no closed-form solution.

# Introduction to Dynamic Programming

A paradigm for solving problems by breaking into subproblems

## When to use DP

- Optimal substructure
    - Bellman Optimality Equation
- Overlapping subproblems
    - Bellman Expectation Equation

### DP in Markov Decision Process

- DP requires full knowledge of MDP
- model-based method

#### Prediction: Policy Evaluation

- Given policy $\pi$, the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ is a MRP $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$
- Prediction focuses on evaluating the policy $\pi$ and calculating the value function $\mathbf{v}_{\pi}$
- Using Bellman Expectation Equation

#### Control: Policy Improvement

- Control focuses on how to improve the policy and find the optimal policy and its value function
    - Policy Iteration
    - Value Iteration
- Using both Bellman Expectation Equation and Bellman Optimality Equation

## Policy Evaluation

- Input: $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy $\pi$
    - $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
- Output: $\mathbf{v}_\pi$

Using Bellman Expectation Equation

$$\mathbf{v}_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}_\pi \tag{30}$$

- Non-iterative method $\mathbf{v}_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$
- Iterative method

$$\mathbf{v}_\pi^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}_\pi^k \tag{31}$$

## Algorithm

- Input: $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
- Initial: $\mathbf{v}_\pi^0$
- At each Iteration $k$
    - $\mathbf{v}_\pi^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}_\pi^k$
- Output: $\mathbf{v}_\pi$

Convergence can be guaranteed because all the eigenvalues of $\gamma \mathcal{P}^{\pi}$ are between $-1$ and $1$

# Policy Iteration

Policy Iteration can be divided into two parts

- Policy evaluation
- Policy Improvement

## Policy Improvement

Question: Given $\pi$ and $\mathbf{v}_{\pi}$, find a better policy $\pi'$

Strategy: Greedy Search

- Given $\pi$ and $\mathbf{v}_{\pi}$, $q_{\pi}(s, a)$
- Consider a deterministic policy $\pi'$
    - Because $v(s) = \sum_a \pi(a \mid s) q_{\pi}(s, a) \leqslant \max_a q_{\pi}(s, a)$

- By setting $\pi'(s) = \text{argmax}_a q_\pi(s, a)$,
- $v_{\pi'}(s) = \max_a q_\pi(s, a) \geqslant v_\pi(s)$

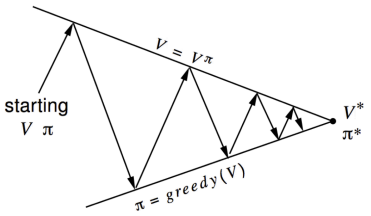
## Algorithm

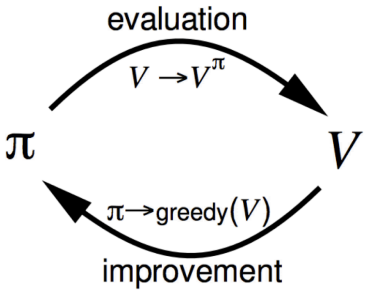
- Input:  MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
- Initial: $\pi_0$
- At each iter $k$
  - Evaluate the policy $\pi_k$, calculate $q_{\pi_k}(s, a)$
  - Update policy $\pi_{k+1}(s) = \text{argmax}_a q_{\pi_k}(s, a)$
- Output: Optimal policy and value function $\pi_*$ and $\mathbf{v}_*$

## Notes

- Proof of Convergence
  - The value function is improved at each iteration
  -

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi\left(s, \pi'(s)\right) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi\left(S_{t+1}, \pi'\left(S_{t+1}\right)\right) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi\left(S_{t+2}, \pi'\left(S_{t+2}\right)\right) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s)
\end{aligned}
$$

    - And the improvement stops only if it achieves the optimal.
- Discussion of stopping condition
- The algorithm contains two levels of iteration
    - The policy iteration
        - policy evaluation
    - Why not update policy after each iteration of evaluation
        - which leads to **value iteration**
- Generalization:
    - 

    - 

# Value Iteration

Problem: Try to directly find the optimal value function instead of the optimal policy.

## Recap: Bellman Optimality Equation

It consists of two parts:

- Maximisation:

$$v_*(s) = \max_a q_*(s, a) \tag{32}$$

- Expectation:

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_* \left(s'\right) \tag{33}$$

### Algorithm

- Input:  MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
- Initial: $\mathbf{v}_0$ and $q_0(s,a)$
- At each iter $k$
  - $v_{k+1}(s) = \max_a q_k(s,a)$
  - $q_{k+1}(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{k+1}(s')$
- Output: the optimal value function $\mathbf{v}_*$

## Discussion

### Async Dynamic Programming

- In-place Dynamic Programming
  - Sync:
  - $$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{old}(s') \right) \quad (34)$$
  - Async:

- 

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v\left(s'\right) \right) \tag{35}$$

- Prioritised Sweeping:
  - Use magnitude of Bellman error to guide state selection:
    - 

$$\left| \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v\left(s'\right) \right) - v(s) \right| \tag{36}$$

  - Update the state with bigggest error
- Real-Time Dynamic Programming
- Full-Width Backups and Sample Backups

## Convergence: contraction mapping

> “
> Contraction Mapping Theorem:
>
> For any metric space $\mathcal{V}$ that is complete (i.e. closed) under an operator $T(v)$ , where $T$ is a $\gamma$-contraction,
>
> 1. $T$ converges to a unique fixed point

2. At a linear convergence rate of $\gamma$

Bellman Expectation Operator

- Define the Bellman expectation backup operator $T^\pi$
  - $T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v$
- This operator is a $\gamma$-contraction
  -

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

Bellman Optimality Operator

- Define the Bellman optimality backup operator $T^*$
  - $T^*(v) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v$

- This operator is a $\gamma$ -contraction
    - $\|T^*(u) - T^*(v)\|_\infty \leq \gamma \|u - v\|_\infty$

### Summary

- MDP and Bellman Equation
- Use DP for prediction and control
- Policy Evaluation
- Policy Iteration
- Value Iteration

# Lec 4 MC–TD

Recap:

- Last Lecture: Dynamic Programming

    - Model-based Prediction and Control
- This Lecture: model-free prediction(evaluation)
    - Estimate the value function of an unknown MDP
    - Monte-Carlo Learning
    - Temporal-Difference Learning
- Next Lecture: model-free control
    - optimize the value function of an unknown MDP

# Monte-Carlo Learning

- model free: no knowledge of MDP
- learn directly from episodes of experience
- learns from complete episodes: no bootstrapping
- can only be applied to episodic MDPs

### First-Visit Policy Evaluation

To evaluate state $s$

- The first time that $s$ is visited in an episode

    - $N(s) \leftarrow N(s) + 1$
    - $S(s) \leftarrow S(s) + G_t$
- $V(s) = S(s)/N(s)$

## Every-Visit Policy Evaluation

To evaluate state $s$

- Every time that $s$ is visited in an episode
    - $N(s) \leftarrow N(s) + 1$
    - $S(s) \leftarrow S(s) + G_t$
- $V(s) = S(s)/N(s)$

## Incremental Update of Mean

- $N(S_t) \leftarrow N(S_t) + 1$
- $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$ or in non-stationary problems $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$

# Temporal–Difference Learning

- model free: no knowledge of MDP
- learn directly from episodes of experience
- learns from incomplete episodes: bootstrapping
- updates a guess based upon a guess

## TD(0)

- In MC, we use empirical return to estimate $G_t$
- In TD(0), we use $G_t = R_{t+1} + \gamma V(s_{t+1})$ to estimate $G_t$, where
  - $R_{t+1}$ uses the empirical reward
  - $V(s_{t+1})$ uses the estimated value function
- Update value function based on estimation

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)\right) \quad (37)$$

  - $R_{t+1} + \gamma V(s_{t+1})$ : TD target
  - $R_{t+1} + \gamma V(s_{t+1}) - V(S_t)$ : TD error

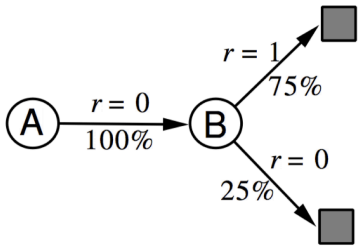
## MC vs. TD

- Bias/Variance Trade-Off
  - Empirical Return $G_t$ is unbiased estimator of $v_\pi(S_t)$
  - $R_{t+1} + v_\pi(S_{t+1})$ is also unbiased
  - $R_{t+1} + \hat{v}(S_{t+1})$ is biased, but asymptotic unbiased.
  - But $R_{t+1} + \hat{v}(S_{t+1})$ reduces the variance
    - Because it depends only on $a_t, S_{t+1}, R_{t+1}$
- MC:
  - unbiased, high variance
  - Good Convergence(even with function approximation)
  - Not sensitive to initial value
  - only learn from complete sequences
    - episodic MDPs
    - can not learn only
- TD
  - biased. low variance
  - TD(0) converges(but not with function approximation)
  - Sensitive to initial value
  - learn from incomplete sequences
    - non-terminating MDPs
    - can learn online

AB Example:

> Two state A and B, 8 episodes, $\gamma = 1$
>
> 1. A,0,B,0
> 2. B,1
> 3. B,1
> 4. B,1
> 5. B,1
> 6. B,1
> 7. B,1
> 8. B,0
>
> Estimate $v(A)$ and $v(B)$

- Using MC, $v(A) = 0, v(B) = 6/8 = 0.75$
- Using TD, the model is like




so $v(A) = v(B) = 0.75$

Discussion:

- MC doesn't account for the Markov property (state transition)
  - converges to the model with minimum mean-squared error
  - Fit the model to minimize
  -

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} \left(G_t^k - V\left(s_t^k\right)\right)^2 \tag{38}$$

- TD(0) consider the Markov property
  - converges to the model with max likelihood
  - Fit the MDP model
  -

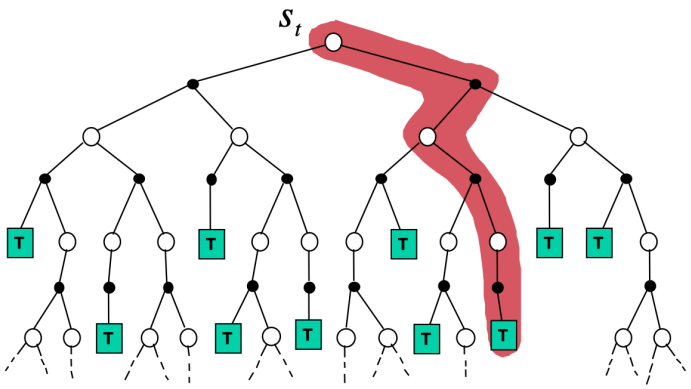
$$\hat{\mathcal{P}}_{s,s'}^a = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}\left(s_t^k, a_t^k, s_{t+1}^k = s, a, s'\right)$$

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

- If the environment is Markov and the dataset is big enough, MC and TD(0) converges to the same result.

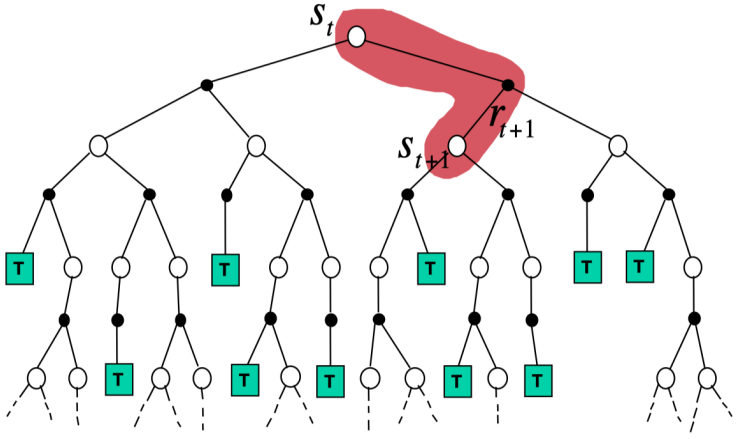
Monte-Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{39}$$



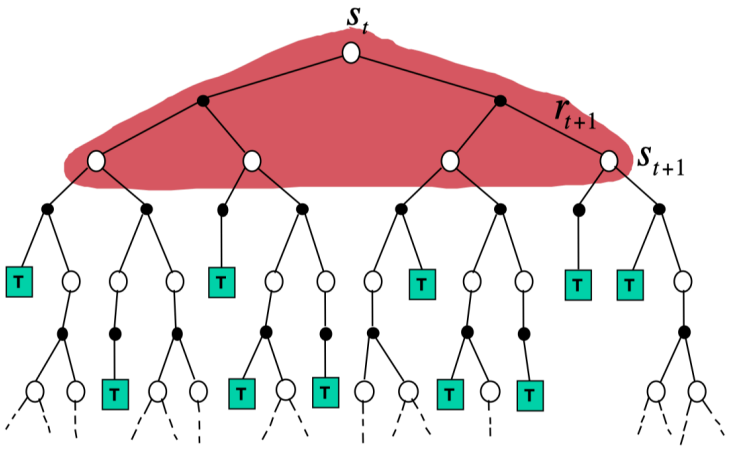

Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{40}$$




Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[ R_{t+1} + \gamma V(S_{t+1}) \right] \tag{41}$$




- Boostrapping: Estimate based on estimation (Depth)
- Sampling: Using one episode instead of all to update (Width)

## TD(λ)

### n-step TD target

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \tag{42}$$

$$\begin{aligned} G_t^{(1)} &= R_{t+1} + \gamma V(S_{t+1}) \\ G_t^{(2)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\ &\vdots \\ G_t^{(\infty)} &= R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T \end{aligned} \tag{43}$$

when $n \to \infty$ TC turns into MC

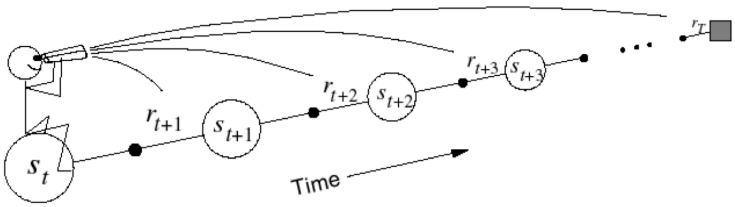
### Averaging n-Step returns: λ-return

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)} \tag{44}$$

**Forward-view TD(λ)**

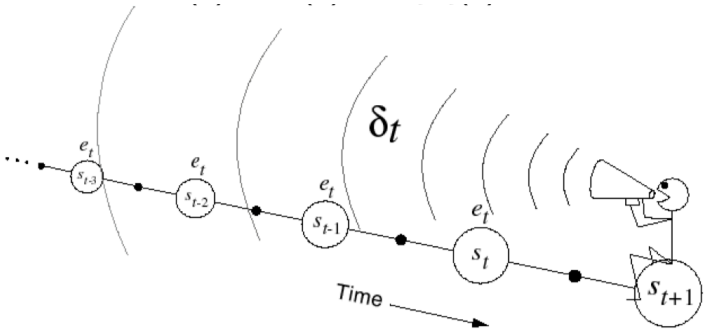
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right) \tag{45}$$

- Update value function use λ-return
- Use the future to compute $G_t^\lambda$
- can only be computed from complete episodes like MC




**Backward-view TD(λ)**

- Forward-view provides theory while backward provides mechanism
- Update online from incomplete sequences




**Eligibility Traces**

- Credit Assignment Problem
    - Frequency Heuristic
    - Recency Heuristic
- Combine Frequecy and Recency: Eligibility Trace
    - $E_0(s) = 0, E_t(s) = \gamma\lambda E_t(s) + \mathbf{1}(S_t = s)$
    - $\alpha = \gamma\lambda = 1$: frequency; 0, recency

**Apply Eligibility Trace to Backward-view TD(λ)**

- Maintian an eligibility trace $E_t(s)$ for every state $s$
- Update value $V(s)$ online for every state $s$ after each step
- The Increment comes from
    - The TD-error $\delta_t = R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)$
    - The eligibility trace $E_t(s)$
- $V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$

**Forward vs. Backward**

- $\lambda = 0$： backward is equivalent to forward TD(0)
- $\lambda = 1$： TD(1) is roughly equivalent to every-visit MC

    - If value function is only updated offline(at the end of episode), TD(1) is the same as MC
- General λ:
    - TD errors telescopes to λ-error $G_t^\lambda - V(S_t)$

**Offline vs. Online**

Online updates:

- Updates are applied online at each step
- Forward and backward are slightly different

Offline updates:

- Updates are cumulated within episode
- but applied only at the end of episode

**Summary**

| Offline updates | $\lambda = 0$ | $\lambda \in (0,1)$ | $\lambda = 1$ |
|---|---|---|---|
| Backward view | TD(0) | TD($\lambda$) | TD(1) |
| | $\Vert$ | $\Vert$ | $\Vert$ |
| Forward view | TD(0) | Forward TD($\lambda$) | MC |
| Online updates | $\lambda = 0$ | $\lambda \in (0,1)$ | $\lambda = 1$ |
| Backward view | TD(0) | TD($\lambda$) | TD(1) |
| | $\Vert$ | $\not\Vert$ | $\not\Vert$ |
| Forward view | TD(0) | Forward TD($\lambda$) | MC |
| | $\Vert$ | $\Vert$ | $\Vert$ |
| Exact Online | TD(0) | Exact Online TD($\lambda$) | Exact Online TD(1) |

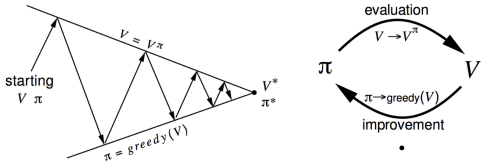
# Lec 5 Model–free Control

### On-Policy vs. Off-Policy

- Policy we try to learn(optimize) π
- Policy we use to learn from expererience μ
- On-Policy: $\pi = \mu$; Off-policy: $\pi \neq \mu$

# On–Policy Monte–Carlo Control

## Generalized Policy Iteration




**Policy Evaluation**: Estimate $v_\pi$

**Policy Improvement**: Find $\pi' \geqslant \pi$

In Monte-Carlo control,

- For policy evaluation, use Monte-Carlo evaluation
- For policy improvement, use greedy policy improvement

Greedy policy improvement over $V(s)$

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s') \tag{46}$$

But it requires MDP model, and Greedy policy improvement over action-value function $Q(s,a)$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s,a) \tag{47}$$

$\varepsilon-$greedy Exploration

- With prob $1-\varepsilon$ choose greedy action
- With prob ε choose a random action

$$\pi(a \mid s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases} \tag{48}$$

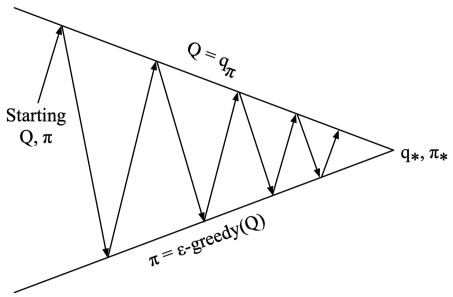
Improvement Theorem

For any ε-greedy policy $\pi$, the ε-greedy policy $\pi'$ using $q_\pi$ satisfies $v'_\pi(s) \geqslant v_\pi(s)$

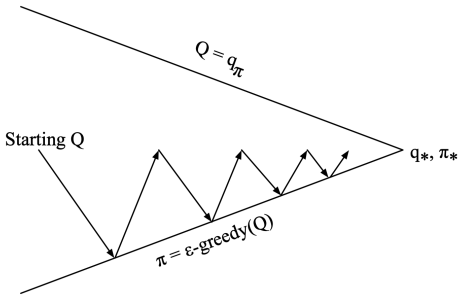
$$\begin{aligned} q_\pi\left(s, \pi'(s)\right) &= \sum_{a \in \mathcal{A}} \pi'(a \mid s) q_\pi(s,a) \\ &= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \max_{a \in \mathcal{A}} q_\pi(s,a) \\ &\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a \mid s) - \epsilon/m}{1-\epsilon} q_\pi(s,a) \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s,a) = v_\pi(s) \end{aligned}$$

## Monte Carlo Control

### Monte Carlo Policy Iteration




### Monte-Carlo Control




## Convergence

How to make sure it converges to $\pi^*, q^*$?

We have GLIE (Greedy in the Limit with Infinite Exploration)

- 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
  $\lim_{k\to\infty} \pi_k(a \mid s) = \mathbf{1}\left(a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, Q_k\left(s, a'\right)\right)$

For example, if we let ε reduces to 0 like $\varepsilon_k = \frac{1}{k}$, the ε-greedy is GILE

#### GLIE Monte-Carlo control

At iter $k$

- Sample episode using $\pi$ $\{S_1, A_1, R_2, \ldots, S_T\} \sim \pi$
- For each state $S_t$ and action $A_t$:
  - $N\left(S_t, A_t\right) \leftarrow N\left(S_t, A_t\right) + 1$

    - $Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right)+\frac{1}{N\left(S_t, A_t\right)}\left(G_t-Q\left(S_t, A_t\right)\right)$
- Improve policy based on $Q(S, A)$
    - $\varepsilon \leftarrow 1/k$
    - $\pi \leftarrow \varepsilon-$greedy(Q)

Theorem:

GLIE Monte-Carlo control converges to the optimal action-value function $q_*(s, a)$

# On–Policy Temporal–Difference Control

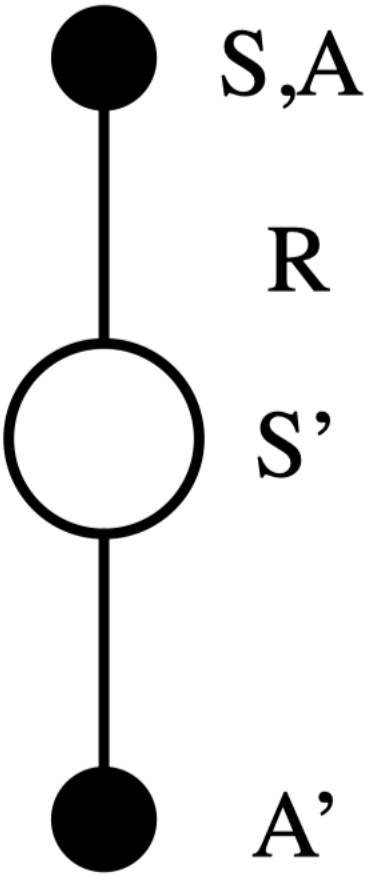
TD has several advantages over MC

- lower variance
- online
- incomplete sequences

Apply TD instead of MC for control, we have Sarsa(λ)

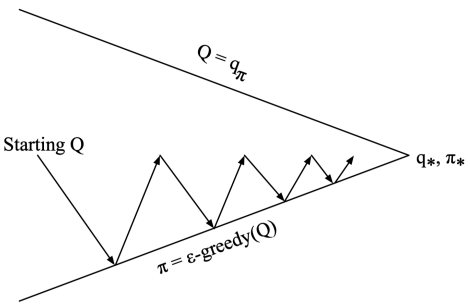
# Sarsa

### Updating Q(s,a)




$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q\left(S', A'\right) - Q(S,A)\right) \tag{49}$$

### On-Policy Control with Sarsa




Every **time-step**(In MC,we do that every episode)

- Policy evaluation: Sarsa update
- Policy Improvement: ε-greedy

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

## Convergence

Sarsa converges to the optimal action value function under the following conditions:

- GLIE sequence of policies $\pi_t(a|s)$
- Robbins-Monro sequence of step-size $\alpha_t$
  - $\sum_{t=1}^{\infty} \alpha_t = \infty$
  - $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

## Sarsa(λ)

n-step return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q\left(S_{t+n}\right) \tag{50}$$

λ-return

$$q_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)} \tag{51}$$

**Forward View Sarsa(λ)**

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha \left(q_t^{\lambda} - Q\left(S_t, A_t\right)\right) \tag{52}$$

**Backward View Sarsa(λ)**

Keep an eligibility trace for each state-action pair

- $E_0(s,a) = 0$
- $E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}\left(S_t = s, A_t = a\right)$

Update $Q(s,a)$ from TD error and eligibility trace

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

**Algorithm**

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize $S$, $A$
  Repeat (for each step of episode):
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$
      $E(s, a) \leftarrow \gamma \lambda E(s, a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until $S$ is terminal

# Off-policy Learning

- Episode: $\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$
- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$

Why do we need off-policy learning?

- Learn from other agents
- Re-use experience
- 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

- Use returns generated from μ to evaluate π
- Importance Sampling
-

$$G_t^{\pi/\mu} = \frac{\pi\left(A_t \mid S_t\right)}{\mu\left(A_t \mid S_t\right)} \frac{\pi\left(A_{t+1} \mid S_{t+1}\right)}{\mu\left(A_{t+1} \mid S_{t+1}\right)} \cdots \frac{\pi\left(A_T \mid S_T\right)}{\mu\left(A_T \mid S_T\right)} G_t \quad (53)$$

- Update value $V\left(S_t\right) \leftarrow V\left(S_t\right)+\alpha\left(G_t^{\pi/\mu}-V\left(S_t\right)\right)$
- Problem
  - $\mu$ may be zero
  - increase variance

### Importance Sampling for off-policy TD

- Use TD targets generated from μ to evaluate π
- Importance Sampling
-

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(\frac{\pi\left(A_t \mid S_t\right)}{\mu\left(A_t \mid S_t\right)}\left(R_{t+1}+\gamma V\left(S_{t+1}\right)\right)-V\left(S_t\right)\right)$$

- lower variance

## Q–Learning

- We use importance sampling to update value function $V(S)$
- Now consider off-policy learning with $Q(s,a)$
- We don't need importance sampling

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right)+\alpha\left(R_{t+1}+\gamma Q\left(S_{t+1}, A'\right)-Q\left(S_t, A_t\right)\right) \tag{54}$$

- $A'$ is decided by μ

#### Off-Policy Control With Q-Learning

- Behaviour Policy μ
- Target Policy π

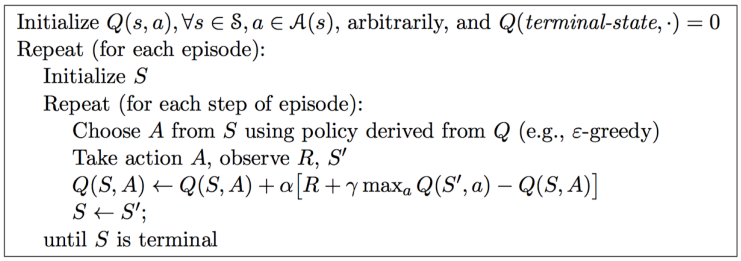
- If we allow both μ and π to improve
  - improve π: greedy $\pi\left(S_{t+1}\right)=\underset{a'}{\operatorname{argmax}} Q\left(S_{t+1}, a'\right)$
  - improve μ: ε-greedy

### Q-Learning Control Algorithm

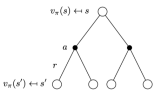
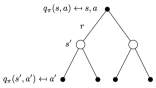
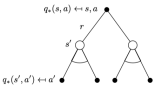
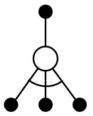
$$Q(S, A) \leftarrow Q(S, A)+\alpha\left(R+\gamma \max_{a'} Q\left(S', a'\right)-Q(S, A)\right) \tag{55}$$

Theorem:

Q-Learning control converges to the optimal action-value function




## DP vs. TD

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

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

# Lec 6 Value Function Approximation

# Introduction

## RL for large-scale Problems

How does Reinforcement Learning solve large problems?

For small-scale problems, we use a lookup table

- $s \rightarrow V(s)$
- $(s, a) \rightarrow V(s, a)$

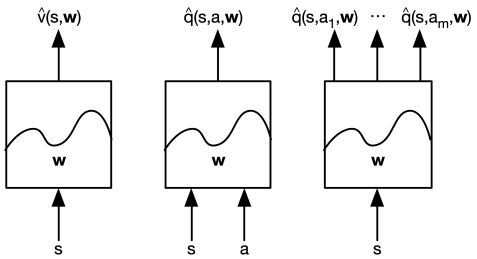
Problems with large MDPs

- Too many states or state-action pairs to store in memory
- Too slow to learn the value of each state or state-action pair

Solutions:

- Esitimate value function with a function approximation $\hat{v}(s, w) \approx v_\pi(s)$ or $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$
- Generalize from seen states to unseen states
- Use parameter $\mathbf{w}$ using MC or TD learning

## Types of Value Function Approximation




Function Approximators:

- **Linear Regression**
- **Decision Tree**
- Nearest Neighbor
- Fourier/wavelet bases
- Neural Network

In RL, we prefer the function approximator to be **differentiable**.

Furthermore, it requires a training method for **non-stationary, non-iid** data.

# Incremental Methods

## Gradient Descent

### Gradient Descent

- Let $J(\mathbf{w})$ be a differentiable function
- The gradient is

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix} \tag{56}$$

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

#### Value Function Approximation By Stochastic Gradient Descent

- Goal: find parameter vector $\mathbf{w}$ minimizing mean-squared error between target function $\hat{v}(s, \mathbf{w})$ and true value function $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[ (v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right] \tag{57}$$

- Stochastic Gradient Descent

$$\Delta \mathbf{w} = \alpha \left( v_\pi(S) - \hat{v}(S, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \tag{58}$$

## Linear Function Approximation

#### Feature Vectors

- Represent state by a feature vector

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix} \tag{59}$$

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

**Linear Function Approxmation**

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j \tag{60}$$

Objective Function

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[ \left( v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w} \right)^2 \right] \tag{61}$$

Stochastic Gradient Descent

$$\Delta \mathbf{w} = \alpha \left( v_\pi(S) - \hat{v}(S, \mathbf{w}) \right) \mathbf{x}(S) \tag{62}$$

Table Lookup is a special case of linear function approximation

$$\mathbf{x}^{\text{table}}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \tag{63}$$

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix} \tag{64}$$

## Incremental Prediction Algorithms

- Problem: we don't know the true $v_\pi(s)$
- Solution: find a target to replace $v_\pi(s)$
- In MC, the target is the return $G_t$

$$\Delta \mathbf{w} = \alpha \left( G_t - \hat{v} \left( S_t, \mathbf{w} \right) \right) \nabla_{\mathbf{w}} \hat{v} \left( S_t, \mathbf{w} \right) \tag{65}$$

  - $G_t$ is unbiased
  - Apply supervised learning to training data
    $\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, \ldots, \langle S_T, G_T \rangle$
  - Linear model converges to global optimum
  - Non-linear model converges to local optimum
- In TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v} \left( S_{t+1}, \mathbf{w} \right)$

$$\Delta \mathbf{w} = \alpha \left( R_{t+1} + \gamma \hat{v} \left( S_{t+1}, \mathbf{w} \right) - \hat{v} \left( S_t, \mathbf{w} \right) \right) \nabla_{\mathbf{w}} \hat{v} \left( S_t, \mathbf{w} \right) \tag{66}$$

  - TD target is biased
  - Apply supervised learning to training data
    $\langle S_1, R_2 + \gamma \hat{v} \left( S_2, \mathbf{w} \right) \rangle, \langle S_2, R_3 + \gamma \hat{v} \left( S_3, \mathbf{w} \right) \rangle, \ldots, \langle S_{T-1}, R_T \rangle$
  - Linear model converges close to global optimum
- In TD(λ), the target is the λ-return $G_t^\lambda$

$$\Delta \mathbf{w} = \alpha \left( G_t^\lambda - \hat{v} \left( S_t, \mathbf{w} \right) \right) \nabla_{\mathbf{w}} \hat{v} \left( S_t, \mathbf{w} \right) \tag{67}$$

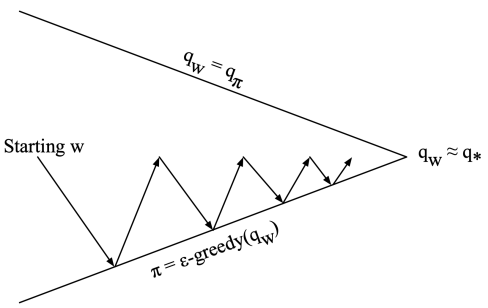
- λ-return is biased
- Apply supervised learning to training data
  $\left\langle S_1, G_1^\lambda \right\rangle, \left\langle S_2, G_2^\lambda \right\rangle, \ldots, \left\langle S_{T-1}, G_{T-1}^\lambda \right\rangle$
- Forward view TD(λ)

$$\Delta \mathbf{w} = \alpha \left( G_t^\lambda - \hat{v} \left( S_t, \mathbf{w} \right) \right) \nabla_{\mathbf{w}} \hat{v} \left( S_t, \mathbf{w} \right) \tag{68}$$

- Backward View TD(λ)

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma \hat{v} \left( S_{t+1}, \mathbf{w} \right) - \hat{v} \left( S_t, \mathbf{w} \right) \\ E_t &= \gamma \lambda E_{t-1} + \nabla_{\mathbf{w}} \hat{v} \left( S_t, \mathbf{w} \right) \\ \Delta \mathbf{w} &= \alpha \delta_t E_t \end{aligned}$$

## Incremental Control Algorithms



- Policy Evaluation: Approxmate Policy Evaluation
- Policy Improvement: ε-greedy policy improvement

#### Action-value function approximation

- Goal: Find $\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$
- Objective Function: $J(\mathbf{w}) = \mathbb{E}_\pi \left[ \left( q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}) \right)^2 \right]$
- SGD: $\Delta \mathbf{w} = \alpha \left( q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$
- Linear model: $\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$
- Also, we should find a target to replace $q_\pi(s, a)$
  - In MC, $\Delta \mathbf{w} = \alpha \left( G_t - \hat{q} \left( S_t, A_t, \mathbf{w} \right) \right) \nabla_{\mathbf{w}} \hat{q} \left( S_t, A_t, \mathbf{w} \right)$
  - In TD(0),
    $\Delta \mathbf{w} = \alpha \left( R_{t+1} + \gamma \hat{q} \left( S_{t+1}, A_{t+1}, \mathbf{w} \right) - \hat{q} \left( S_t, A_t, \mathbf{w} \right) \right) \nabla_{\mathbf{w}} \hat{q} \left( S_t, A_t, \mathbf{w} \right)$
  - In Forward-View TD(λ):
    $\Delta \mathbf{w} = \alpha \left( q_t^\lambda - \hat{q} \left( S_t, A_t, \mathbf{w} \right) \right) \nabla_{\mathbf{w}} \hat{q} \left( S_t, A_t, \mathbf{w} \right)$
  - In Backward-View TD(λ):

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

## Convergence

| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | TD(0) | ✓ | ✓ | ✗ |
| | TD($\lambda$) | ✓ | ✓ | ✗ |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | TD(0) | ✓ | ✗ | ✗ |
| | TD($\lambda$) | ✓ | ✗ | ✗ |

Problem: TD doesn't follow the gradient of any objective function → Gradient TD

| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | TD | ✓ | ✓ | ✗ |
| | Gradient TD | ✓ | ✓ | ✓ |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | TD | ✓ | ✗ | ✗ |
| | Gradient TD | ✓ | ✓ | ✓ |

### Convergence of Control Algorithms

| Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|
| Monte-Carlo Control | ✓ | (✓) | ✗ |
| Sarsa | ✓ | (✓) | ✗ |
| Q-learning | ✓ | ✗ | ✗ |
| Gradient Q-learning | ✓ | ✓ | ✗ |

# Batch Methods

- Problem: Low Sample Efficiency
- Solution: Given the agent's experience as input("training data"), find the best fitting value function

## Least Square Prediction

- Given value function approximation model $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And experience $\mathcal{D}$ consisting of <state,value> pairs
  $\mathcal{D} = \left\{ \left\langle s_1, v_1^\pi \right\rangle, \left\langle s_2, v_2^\pi \right\rangle, \ldots, \left\langle s_T, v_T^\pi \right\rangle \right\}$
- Least Sqaures Algorithms find parameters minimizing

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

#### Stochastic Gradient Descent with Experience Replay

Repeat

1. Sample state, value form experience $\left\langle s, v^\pi \right\rangle \sim \mathcal{D}$
2. Apply stochastiv gradient descent update
   $\Delta \mathbf{w} = \alpha \left( v^\pi - \hat{v}(s, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$

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

**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 mini-batch of transitions $(s, a, r, s')$ form $\mathcal{D}$
- Compute Q-learning targets w.r.t. old,fixed parameters $w^-$
- Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i\left(w_i\right)=\mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(r+\gamma\max_{a'}Q\left(s',a';w_i^-\right)-Q\left(s,a;w_i\right)\right)^2\right] \quad (69)$$

**Linear Least Square Prediction**

- Experience replay may take many iterations
- If we use linear model to approximate value function $\hat{v}(s,\mathbf{w})=\mathbf{x}(s)^\top\mathbf{w}$, it has a closed form solution

$$\mathbf{w}=\left(\sum_{t=1}^{T}\mathbf{x}\left(s_t\right)\mathbf{x}(s_t)^\top\right)^{-1}\sum_{t=1}^{T}\mathbf{x}\left(s_t\right)v_t^\pi \quad (70)$$

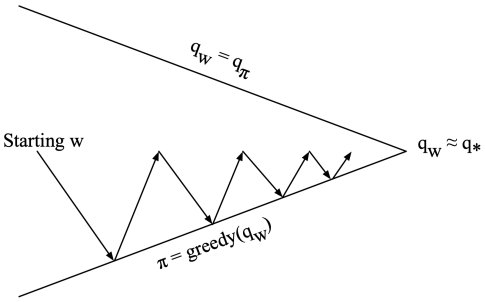
- Direct solution is $O(N^3)$, incremental solution time is $O(N^2)$ using Shermann-Morrison

- LSMC: $\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}\left(S_t\right) \mathbf{x}(S_t)^\top\right)^{-1} \sum_{t=1}^{T} \mathbf{x}\left(S_t\right) G_t$
- LSTD: $\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}\left(S_t\right) \left(\mathbf{x}\left(S_t\right) - \gamma \mathbf{x}\left(S_{t+1}\right)\right)^\top\right)^{-1} \sum_{t=1}^{T} \mathbf{x}\left(S_t\right) R_{t+1}$

### Convergence

| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | LSMC | ✓ | ✓ | - |
| | TD | ✓ | ✓ | ✗ |
| | LSTD | ✓ | ✓ | - |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | LSMC | ✓ | ✓ | - |
| | TD | ✓ | ✗ | ✗ |
| | LSTD | ✓ | ✓ | - |

## Least Square Control




- Policy Evaluation: Least Square Q-learning
- Policy Improvement: Greedy Policy Improvement

### Least Square Action-Value Function Approximation

- Using linear model to approximate action-value function
  $\hat{q}(s, a, \mathbf{w}) = \mathbf{x}(s, a)^\top \mathbf{w} \approx q_\pi(s, a)$
- From experience generated under policy π:
  $\mathcal{D} = \left\{ \langle (s_1, a_1), v_1^\pi \rangle, \langle (s_2, a_2), v_2^\pi \rangle, \ldots, \langle (s_T, a_T), v_T^\pi \rangle \right\}$

#### Least Square Control

- For policy evaluation, wa want to use all experience
- But experience is generated form many policies
- So we must learn off-policy
  - Use experience generated by old policy
    $S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{\text{old}}$
  - Consider alternative successor action $A' = \pi_{new}(S_{t+1})$
  - Update $\hat{q}\left(S_t, A_t, \mathbf{w}\right) = R_{t+1} + \gamma \hat{q}\left(S_{t+1}, A', \mathbf{w}\right)$

#### Least-Square Q-learning

- Linear Q-learning update

$$\delta = R_{t+1} + \gamma \hat{q}\left(S_{t+1}, \pi\left(S_{t+1}\right), \mathbf{w}\right) - \hat{q}\left(S_t, A_t, \mathbf{w}\right)$$
$$\Delta \mathbf{w} = \alpha \delta \mathbf{x}\left(S_t, A_t\right)$$

- LSTDQ algorithm

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}\left(S_t, A_t\right)\left(\mathbf{x}\left(S_t, A_t\right) - \gamma \mathbf{x}\left(S_{t+1}, \pi\left(S_{t+1}\right)\right)\right)^{\top}\right)^{-1} \sum_{t=1}^{T} \mathbf{x}\left(S_t, A_t\right) R_{t+1} \quad (71)$$

### Least Square Policy Iteration Algorithm

**function LSPI-TD**$(\mathcal{D}, \pi_0)$
  $\pi' \leftarrow \pi_0$
  **repeat**
    $\pi \leftarrow \pi'$
    $Q \leftarrow$ **LSTDQ**$(\pi, \mathcal{D})$
    **for all** $s \in \mathcal{S}$ **do**
      $\pi'(s) \leftarrow \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s, a)$
    **end for**
  **until** $(\pi \approx \pi')$
  **return** $\pi$
**end function**

### Convergence of Control algorithms

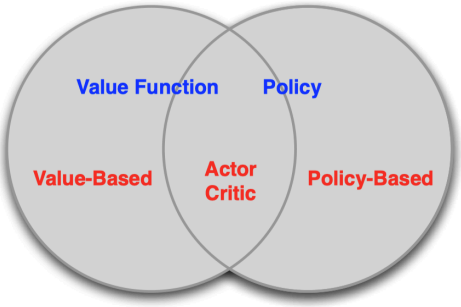
| Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|
| Monte-Carlo Control | ✓ | (✓) | ✗ |
| Sarsa | ✓ | (✓) | ✗ |
| Q-learning | ✓ | ✗ | ✗ |
| LSPI | ✓ | (✓) | - |

# Lec 7 Policy Gradient

- Last Lecture
  - we approximated the value or action-value function using parameters θ,
  - $$V_\theta(s) \approx V^\pi(s)$$
    $$Q_\theta(s,a) \approx Q^\pi(s,a)$$
  - And policy is generated directly from the value function (e.g. using ε-greedy)
- This Lecture
  - We will focus on **model-free** reinforcement learning
  - $$\pi_\theta(s,a) = \mathbb{P}[a \mid s, \theta] \qquad (72)$$

# Introduction

Value-Based v.s. Policy-Based




Advantages of Policy-Based RL

- better convergence
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages

- Local Optimum
- Evaluating a policy is inefficient and high variance

## Policy Search

Goal: give policy $\pi_\theta(s,a)$ with parameters $\theta$, find best $\theta$

How do we measure the quality of $\pi_\theta$

- In episodic environments we use the start value:
  - $J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$
- In continuing environments we use average value

- $J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$
- or average reward per time-step
- $J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) \mathcal{R}_s^a$
- where $d^{\pi_\theta}(s)$ is the stationary distribution of Markov Chain

Given $\pi_\theta(s,a)$ and $J(\theta)$, policy-based reinforcement learning is an optimization problem.

Optimization methods

- Non-gradient methods
- Gradient methods
    - gradient descent

# Finite Difference Policy Gradient

- To evaluate policy gradient of $\pi_\theta(s,a)$
- For each dimension $k \in [1,n]$
    - $\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta+\epsilon u_k)-J(\theta)}{\epsilon}$
- Simple, noisy, inefficient

# Monte–Carlo Policy Gradient

## Score Function

Assume policy $\pi_\theta$ is differentiable whenever it is non-zero

$$\begin{aligned}\nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\ &= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\end{aligned}$$

Score Function $\nabla_\theta \log \pi_\theta(s,a)$

For softmax policy

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^\top \theta} \tag{73}$$

So the score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)] \tag{74}$$

For Gaussian Policy(in continuous action spaces) $a \sim \mathcal{N}\left(\mu(s) = \phi(s)^T\theta, \sigma^2\right)$

$$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a-\mu(s))\phi(s)}{\sigma^2} \tag{75}$$

## Policy Gradient Theorem

Consider a simple one-step MDP

- Starting state $s \sim d(s)$
- Terminating after one time-step $r = R_{s,a}$

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\
&= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \mathcal{R}_{s,a} \\
\nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \mathcal{R}_{s,a} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) r\right]
\end{aligned}
$$

**Policy Gradient Theorem**

For any differentiable policy $\pi_\theta(s,a)$

the policy gradient is $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right]$

## REINFORCE

- Update parameters by stochastic gradient descent
- Using return $v_t$ as an unbiased sample of $Q^{\pi_\theta}\left(s_t, a_t\right)$
- $\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta\left(s_t, a_t\right) v_t$

**function REINFORCE**
    Initialise $\theta$ arbitrarily
    **for** each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        **for** $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        **end for**
    **end for**
    **return** $\theta$
**end function**

# Actor–Critic Policy Gradient

Why do we need a critic?

- Problem of MC-PG: high variance due to $v_t$ as an estimator of $Q^{\pi_\theta}(s, a)$
- Instead we use a critic to estimate it $Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$

Actor-critic algorithm maintains

- Critic: Updates action-value function parameters $w$
- Actor: Updates policy parameters $\theta$, in direction suggested by critic

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta} \left[ \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a) \right] \tag{76}$$

## Critic Design

- How to evaluate action-value function (Policy Evaluation)
  - MC learning
  - TD learning
  - function approxmation
- Example: QAC
  - Using linear function approximation $Q_w(s,a) = \phi(s,a)^\top w$
  - Critic: Updates $w$ by linear TD(0)
  - Actor: Update $\theta$ by policy gradient
  - **function** QAC
    Initialise $s$, $\theta$
    Sample $a \sim \pi_\theta$
    **for** each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    Sample action $a' \sim \pi_\theta(s', a')$
    $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a', s \leftarrow s'$
    **end for**
    **end function**

## Bias in Actor–critic

- Bias comes from value function approximation

- A biased policy gradient may lead to the wrong direction
- Luckily, if we choose function approximation carefully, we can avoid the bias.

#### Compatible Function Approximation Theorem

If the following conditions are satisfied,

- Value Function Approximator is compatible to the policy $\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$
- Value Function parameters minimize the mean-squared error $\varepsilon = \mathbb{E}_{\pi_\theta}\left[\left(Q^{\pi_\theta}(s,a) - Q_w(s,a)\right)^2\right]$

Then the policy gradient is exact,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)\right] \tag{77}$$

## Advantage Funtion

#### Reducing Variance using baseline

- We want to find a baseline function $B(s)$ only related to state $s$, which satisfies $\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) B(s)\right] = 0$
- A good baseline is $B(s) = V^{\pi_\theta}(s)$
- Then we can define the advantage function

$$
\begin{aligned}
A^{\pi_\theta}(s,a) &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right]
\end{aligned}
$$

### Estimating Advantage Function

- We can estimate both $V^{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s,a)$

$$
\begin{aligned}
V_v(s) &\approx V^{\pi_\theta}(s) \\
Q_w(s,a) &\approx Q^{\pi_\theta}(s,a) \\
A(s,a) &= Q_w(s,a) - V_v(s)
\end{aligned}
$$

- Notice that the TD-error $\delta^{\pi_\theta}$ is an unbiased estimator of the advantage function

$$
\begin{aligned}
\mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta} \mid s,a\right] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V^{\pi_\theta}(s') \mid s,a\right] - V^{\pi_\theta}(s) \\
&= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\
&= A^{\pi_\theta}(s,a)
\end{aligned}
$$

- We can only use TD error to compute policy gradient

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \delta^{\pi_\theta}\right] \\
\delta_v &= r + \gamma V_v(s') - V_v(s)
\end{aligned}
\tag{78}
$$

- And this way we only use parameters $v$.

## Policy Gradient with Eligibility Traces

- In Critic, we can use MC, TD(0), forward-view/backward-view TD(λ)
- In Actor, we can also use these methods
  - Monte-Carlo policy gradient

    $\Delta\theta = \alpha \left(v_t - V_v \left(s_t\right)\right) \nabla_\theta \log \pi_\theta \left(s_t, a_t\right)$
  - Actor-Critic Policy Gradient

    $\Delta\theta = \alpha \left(r + \gamma V_v \left(s_{t+1}\right) - V_v \left(s_t\right)\right) \nabla_\theta \log \pi_\theta \left(s_t, a_t\right)$
  - Policy Gradient with Eligibility Traces
  -

$$
\begin{aligned}
\delta &= r_{t+1} + \gamma V_v \left(s_{t+1}\right) - V_v \left(s_t\right) \\
e_{t+1} &= \lambda e_t + \nabla_\theta \log \pi_\theta(s, a) \\
\Delta\theta &= \alpha \delta e_t
\end{aligned}
$$

## Natural Policy Gradient

- Vanilla gradient descent is ususlly sensitive to parameters.
- To avoid this, we can use Natural Policy Gradient
-

$$
\nabla_\theta^{nat} \pi_\theta(s, a) = G_\theta^{-1} \nabla_\theta \pi_\theta(s, a) \tag{79}
$$

- where $G_\theta$ is the Fisher information matrix

$$G_\theta = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T\right] \tag{80}$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right] \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T w\right] \\
&= G_\theta w \\
\nabla_\theta^{nat} J(\theta) &= w
\end{aligned}$$

### Summary

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^w(s,a)\right] && \text{Q actor-critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^w(s,a)\right] && \text{Advantage actor-critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \delta\right] && \text{TD actor-critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \delta e\right] && \text{TD}(\lambda)\text{ actor-critic} \\
G_\theta^{-1} \nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}$$

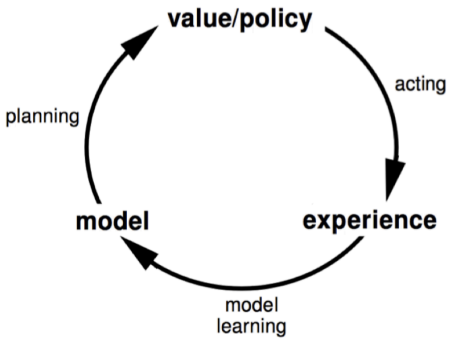
# Lec 8: Integrating Learning and Planning

## Model–Based Reinforcement Learning

Model-based vs model-free RL

- Model-Free RL

    - no model
    - Learn value function or policy from experience
- Model-based RL
    - Learn a model from experience
    - Plan value function and/or policy from model




Advantages:

- Efficiency: supervised learning methods
- Uncertainty can be discussed

Disadvantages:

- Two error sources
    - learning a model
    - find its value function

## Learning a model

A model $\mathcal{M}$ is a representation of an MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} >$, parametrized by $\eta$

- Assume $\mathcal{S}, \mathcal{A}$ are known
- To learn a model is to learn $\mathcal{P}_\eta, \mathcal{R}_\eta$

Goal: estimate model $\mathcal{M}_\eta$ from experience ${S_1,A_1,R_2,\cdots,S_T}, which is a supervised learning problem

- $s, a \rightarrow r$ is a regression problem
- $s, a \rightarrow s'$ is a density estimation problem

**Types of Model**

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model

### Table Lookup Model

$$\begin{aligned}\hat{\mathcal{P}}_{s,s'}^{a} &= \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}\left(S_t, A_t, S_{t+1} = s, a, s'\right) \\ \hat{\mathcal{R}}_{s}^{a} &= \frac{1}{N(s,a)} \sum_{t=1}^{T} 1\left(S_t, A_t = s, a\right) R_t\end{aligned} \tag{81}$$

## Planning with a Model

### Sample-based Planning

- Simple but Powerful
- Use model to generate samples
- Sample Experience from model

$$\begin{aligned}S_{t+1} &\sim \mathcal{P}_{\eta}\left(S_{t+1} \mid S_t, A_t\right) \\ R_{t+1} &= \mathcal{R}_{\eta}\left(R_{t+1} \mid S_t, A_t\right)\end{aligned}$$

- Apply model-free RL to samples

### Planning with an inaccurate model

- When the model is inaccurate, the planning will lead to a suboptimal policy.
- Solution:
    - use model-free
    - reason about model uncertainty

# Integrated Architectures

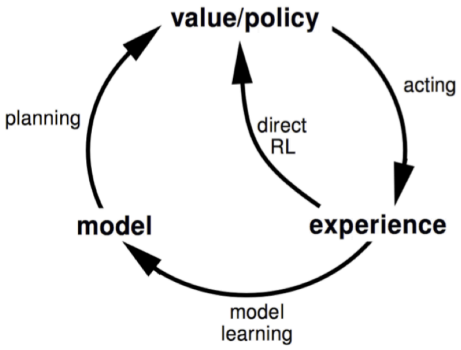
## Dyna

#### Real and Simulated Experience

- Real Experience: from real environment
- Simulated Experience: from model

#### Dyna

- Learn a model from real experience
- Learn and plan value function and policy from real and simulated experience




### Dyna-Q

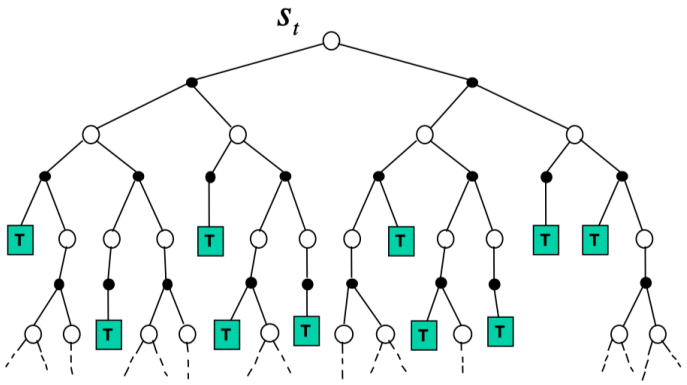
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy$(S,Q)$
(c) Execute action $A$; observe resultant reward, $R$, and state, $S'$
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Repeat $n$ times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in $S$
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

# Simulation–Based Search

### Forward Search

- Select the best action by lookahead
- build a search tree with the current state $s_t$ at the root

- Use a model to look ahead
    - Not the complete MDP model
    - Just sub-MDP starting from now




**Forward Search with simulated experiences**

- Simulate episodes of experience from $s_t$ with the model
- Apply model-free RL to simulated episodes
    - Monte-Carlo control: Monte-Carlo Search
    - Sarsa: TD search

## Monte-Carlo Search

**Simple Monte-Carlo Search**

- Given a model $\mathcal{M}_{\mathcal{V}}$ and a simulation policy $\pi$
- For each state $a \in \mathcal{A}$
    - Simulate $K$ episodes from current state $s_t$

$$\left\{s_t, a, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, \ldots, S_T^k\right\}_{k=1}^K \sim \mathcal{M}_\nu, \pi \tag{82}$$

    - Evaluate actions by mean return

$$Q\left(s_t, a\right) = \frac{1}{K} \sum_{k=1}^K G_t \overset{P}{\rightarrow} q_\pi\left(s_t, a\right) \tag{83}$$

- Select current action with maximum value

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q\left(s_t, a\right) \tag{84}$$

**Monte-Carlo Tree Search**

- Given a model $\mathcal{M}_{\mathcal{V}}$
- Simulate $K$ episodes from current state $s_t$ using current policy $\pi$

$$\left\{s_t, a, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, \ldots, S_T^k\right\}_{k=1}^K \sim \mathcal{M}_\nu, \pi \tag{85}$$

- Build a search tree containing visited states and actions

- Evaluate states $Q(s, a)$ by mean return of episodes from $(s, a)$

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{k=1}^{K} \sum_{u=t}^{T} \mathbf{1}\left(S_u, A_u = s, a\right) G_u \xrightarrow{P} q_\pi(s, a) \quad (86)$$

- In MCTS, the simulation policy $\pi$ improves
- Each simulation consists of two phases
  - Tree policy: pick actions to maximize $Q(s, a)$
  - Default policy: pick actions randomly
- Repeat:
  - Evaluare states $Q(s, a)$, by Monte-Carlo evaluation
  - Improve tree policy by ε-greedy
- Monte-Carlo control applied to simulated experience

Advantages of MCTS

- Highly selective best-first search
- Evaluate states dynamically
- Use sampling to break curse of dimensionality
- Works for black-box models
- Computationally efficient and parallelizable

## Temporal Difference Search

- Simulation Based Search
- Using TD instead of MC (uses bootstrapping)
    - MC tree search applies MC control to sub-MDP from now
    - TD search applies Sarsa to sub-MDP from now

#### TD Search

- Simulate episodes from current state $s_t$
- Estimate action-value function $Q(s, a)$
    - For each step of simulation, update action-values by Sarsa
    - $\Delta Q(S, A) = \alpha \left( R + \gamma Q \left( S', A' \right) - Q(S, A) \right)$
- Select actions using $Q(s, a)$

#### Dyna-2

- In Dyna-2, the agent stores two sets of feature weights

- Long-term memory: updated from real experience; general domain knowledge
- Short-term memory: updated from simulated experience; specific local knowledge

# Lec 9 Exploration and Exploitation

## Introduction

#### Exploration vs. Exploitation Dilemma

- Online Decision-making involves a fundamental choice
  - Exploitation: make the best decision given current information
  - Exploration: gather more information
- Long-term strategy may sacrifice short-term interests

#### Principles

- **Naive Explorartion**

    - Add noise to greedy policy
- **Optimistic Initialization**
    - Assume the best until proven otherwise
- **Optimism in the Face of Uncertainty**
    - Prefer actions with uncertain values
- **Probability Matching**
    - Select actions according to probability they are best
- **Information State Search**

# Multi–Armed Bandit

A multi-armed bandit is a tuple $< \mathcal{A}, \mathcal{R} >$

- $\mathcal{A}$ is a known set of $m$ actions (arms)
- $\mathcal{R}^a(r) = \mathbb{P}(r|a)$ is an unknown probability distribution over rewards
- At each time-step $t$
    - the agent selects an action $a_t \in A$
    - the environment generates a reward $r_t$
- Goal is to maximize cumulative reward $\sum_{t=1}^{T} r_t$

## Regret

- Action-value function $Q(a) = \mathbb{E}[r \mid a]$
- optimal value $V^* = Q\left(a^*\right) = \max_{a \in \mathcal{A}} Q(a)$
- Regret: opportunity loss for one step $I_t = \mathbb{E}\left[V^* - Q\left(a_t\right)\right]$
- Total Regret:
  - $L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q\left(a_\tau\right)\right]$
- Maximize cumulative reward = minimize total regret

### Counting Regret

- Count: $N_t(a)$ expected number of selections for action $a$
- Gap: $\Delta_a = V^* - Q(a)$

$$
\begin{aligned}
L_t &= \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q\left(a_\tau\right)\right] \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_t(a)\right]\left(V^* - Q(a)\right) \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_t(a)\right] \Delta_a
\end{aligned}
$$

**Exploration-Exploitation trade-off** in the perspective of total regret

- If an algorithm forever explores, it will have linear total regret
- But if it never explores, still linear total regret
- Problem: find a balance between **Exploration-Exploitation** to achieve sublinear total regret

## Algorithms and total regret

- Greedy: linear total regret
- ε-greedy: linear total regret
- Optimistic Initialization
    - Initialize $Q(a)$ to high value
    - Encourage Systematic Exploration
    - Still lock onto suboptimal action
    - greedy/ε-greedy + optimistic initialization: linear total regret

- Decaying $\varepsilon_t$-greedy Algorithm

$$
\begin{aligned}
&c > 0 \\
&d = \min_{a|\Delta_a > 0} \Delta_i \\
&\epsilon_t = \min\left\{1, \frac{c|\mathcal{A}|}{d^2 t}\right\}
\end{aligned} \tag{87}
$$

#### Lower Bound

- The performance is determined by similarity between optimal arm and others

#### Theorem

Asymptotic total regret is at least logarithmic in number of steps

$$
\lim_{t \to \infty} L_t \geq \log t \sum_{a|\Delta_a > 0} \frac{\Delta_a}{KL\left(\mathcal{R}^a || \mathcal{R}^{a^*}\right)} \tag{88}
$$

## Upper Confidence Bounds

#### Optimism in the Face of Uncertainty

- Estimate an upper confidence $\hat{U}_t(a)$ for each action-value
  - Small $N_t(a)$ leads to large $\hat{U}_t(a)$
- Select action to max Upper Confidence Bound(UCB)

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}_t(a) + \hat{U}_t(a) \tag{89}$$

### Determine the Upper Bound

> Hoeffding's Inequality:
>
> Let $X_1, \cdots, X_n$ be i.i.d. random variables in $[0, 1]$, and $\hat{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$ is the sample mean.
>
> Then
>
> $$\mathbb{P}[\mathbb{E}[X] \geqslant \hat{X}_n + u] \leqslant e^{-2nu^2} \tag{90}$$

Using that, we have

$$\mathbb{P}\left[Q(a) > \hat{Q}_t(a) + U_t(a)\right] \leq e^{-2N_t(a)U_t(a)^2} = p \tag{91}$$

So $U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$

We can reduce $p$ as time step increases like $p = t^{-4}$

So $U_t(a) = \sqrt{\frac{2 \log t}{N_t(a)}}$

### UCB1

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(a) + \sqrt{\frac{2 \log t}{N_t(a)}} \tag{92}$$

Theorem: The UCB1 algorithm achieves asymptotic logarithmic total regret.

$$\lim_{t \to \infty} L_t \leq 8 \log t \sum_{a | \Delta_a > 0} \Delta_a \tag{93}$$

## Bayesian Bandits

- exploits prior knowledge of rewards
- compute posterior distribution of rewards
- Use posterior to guide exploration
    - UCB
    - Probability matching
- Better performance with accurate prior

### Bayesian UCB

- Assume reward distribution is Gaussian $\mathcal{R}_a(r) = \mathcal{N}(r; \mu_a, \sigma_a^2)$

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \mu_a + \frac{c\sigma_a}{\sqrt{N_t(a)}} \tag{94}$$

### Probability Matching

$$\pi\left(a \mid h_t\right) = \mathbb{P}\left[Q(a) > Q\left(a'\right), \forall a' \neq a \mid h_t\right] \tag{95}$$

- Select action according to probability that $a$ is optimal
    - can be optimistic to uncertainty
- But may be difficult to compute

#### Thompson Sampling

- Implements Probability Matching

$$\begin{aligned}\pi\left(a \mid h_t\right) &= \mathbb{P}\left[Q(a) > Q\left(a'\right), \forall a' \neq a \mid h_t\right] \\ &= \mathbb{E}_{\mathcal{R}|h_t}\left[\mathbf{1}(a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(a))\right]\end{aligned}$$

- Use Bayes law to compute posterior distribution $p[\mathcal{R}|h]$
- Sample a reward distribution $\mathcal{R}$ from posterior and compute action-value function $Q(a) = \mathbb{E}[\mathcal{R}_a]$
- Select action maximizing value on sample

## Information State Search

#### Value of Information

- Quantify the value of information
- Help Trade-off between exploration and exploitation

#### Information State Space

- View bandits as sequential decision-making problems
- Define a information state $\tilde{s} = f(h_t)$
    - and induce a MDP

- Example: Bernoulli Bandits

#### Solving information state space bandits

- Model-free RL
    - e.g. Q-learning
- Bayesian-adaptive RL

#### Bayes-Adaptive Bernoulli Bandits

- Bernoulli Bandits:
    - $\mathcal{R}^a = \mathcal{B}(\mu_a)$
    - Find which arm has the highest $\mu_a$
- Information state $\hat{s} =< \alpha, \beta >$
    - $(\alpha_a, \beta_a)$ the number of success/fails in arm $a$
- Prior: beta distribution
- Compute posterior
    - transition matrix
- Can be solved by dynamic programming
    - Gittins index
- Or use simulation based search

# Contextual Bandits

a tuple $< \mathcal{A}, \mathcal{S}, \mathcal{R} >$

- $\mathcal{A}$ is a known set of $m$ actions (arms)
- $\mathcal{S} = \mathbb{P}[s]$ is an unknown distribution over states
- $\mathcal{R}_s^a(r) = \mathbb{P}(r|s,a)$ is an unknown probability distribution over rewards
- At each time-step $t$
  - the environment generates state $s_t$
  - the agent selects an action $a_t \in A$
  - the environment generates a reward $r_t$
- Goal is to maximize cumulative reward $\sum_{t=1}^{T} r_t$

Use Linear Regression to estimate $Q(s,a)$

- $Q_\theta(s,a) = \phi(s,a)^\top \theta \approx Q(s,a)$
- Using Least Squares Regression to estimate $Q(s,a)$ and its variance $\sigma^2(s,a)$

$$
\begin{aligned}
A_t &= \sum_{\tau=1}^{t} \phi\left(s_\tau, a_\tau\right) \phi(s_\tau, a_\tau)^\top \\
b_t &= \sum_{\tau=1}^{t} \phi\left(s_\tau, a_\tau\right) r_\tau \\
\theta_t &= A_t^{-1} b_t \\
\sigma_\theta^2(s, a) &= \phi(s, a)^\top A^{-1} \phi(s, a)
\end{aligned}
$$

Using UCB

$$
a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_\theta\left(s_t, a\right) + c\sqrt{\phi(s_t, a)^\top A_t^{-1} \phi\left(s_t, a\right)} \tag{96}
$$

# MDPs

Same principles can be also applied to MDPs.

## Optimistic Initialization

**model-free RL**

- initialize action-value function $Q(s, a)$ to $\frac{r_{max}}{1-\gamma}$
- Run model-free RL algorithm
- Encourages systematic exploration of states and actions

#### model-based RL

- Construct an optimistic model of MDP
- Initialize transition to go to heaven
    - i.e.  transition to terminal state with $r_{max}$ reward
- Solve MDP by planning algorithm
- e.g. RMax algorithm

## Optimism in the Face of Uncertainty

#### Model-free RL: UCB

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q\left(s_t, a\right) + U_1\left(s_t, a\right) + U_2\left(s_t, a\right) \tag{97}$$

- $U_1(s_t, a)$: uncertainty in policy evaluation (easy to estimate)
- $U_2(s_t, a)$: uncertainty in policy improvement (hard)

#### Model-based RL: bayesian

- maintain posterior distribution
- Use posterior to guide exploration
    - Bayesian UCB
    - Probability matching (Thompson Sampling)

#### Thompson Sampling

$$\begin{aligned}\pi\left(s, a \mid h_t\right) &= \mathbb{P}\left[Q^*(s, a) > Q^*\left(s, a'\right), \forall a' \neq a \mid h_t\right] \\ &= \mathbb{E}_{\mathcal{P}, \mathcal{R} \mid h_t}\left[\mathbf{1}\left(a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^*(s, a)\right)\right]\end{aligned}$$

- Use Bayes' Law to compute posterior
- Sample an MDP from posterior
- Solve sampled MDP using planning algorithm

## Information State Search

- Augmented Information State Space
- $\tilde{\mathcal{M}} = \langle \tilde{\mathcal{S}}, \mathcal{A}, \tilde{\mathcal{P}}, \mathcal{R}, \gamma \rangle$
    - Bayesian-adaptive MDP
- Simulation-based search