Multi-Agent Systems

Introduction to Reinforcement Learning

Model-free Methods:

Monte Carlo, SARSA and Q-learning

Eric Pauwels (CWI & VU)

November 30, 2021

Reading

- Sutton & Barto: chapters 5 & 6

Outline

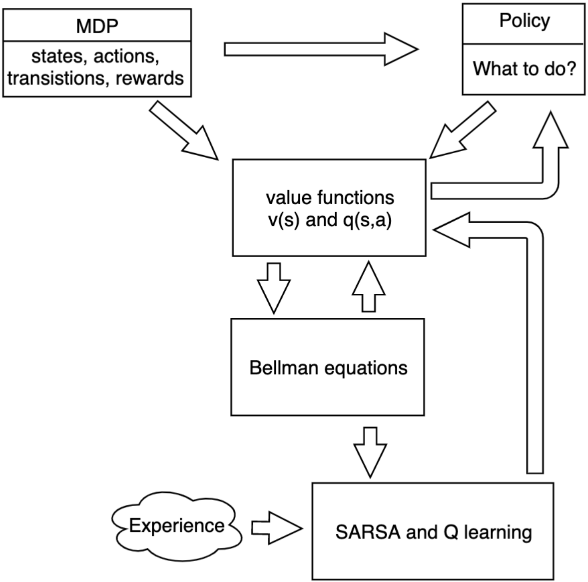
Model-based versus Model-free

	Prediction *Estimation:* *Given π, what is v?*	**(Optimal) Control** *Optimisation:* *What is optimal π?*
model-based (MDP given)	Policy evaluation using Dyn. Programming (DP)	Policy improvement ($+$ Policy evaluation) $=$ Policy iteration
model-free (MDP unknown)	Monte Carlo (MC) Temporal Diff^{ing} (TD) $=$ "impatient MC" *bootstrapping!*	Q-learning and Generalized Policy Iteration *"simultaneous"*

Solving model-free RL problems

- The agent has **no prior knowledge** about states, actions, rewards, transitions!
- By **acting** in the world, the agent **gains experience**. An experience can be expressed as a 4-tuple: (s, a, r, s')
- Over time the agent collects a list of experiences that he uses to find better policies . . . HOW?
  - **Directly:** policy improvement
  - **Indirectly:** estimate value functions, improve policy using greedification;

Overview


MDP
states, actions,
transistions, rewards
Policy
What to do?
value functions
v(s) and q(s,a)
Bellman equations
Experience
SARSA and Q learning

Definition of Greedification

- For a given action-value function $q(s, a)$, a corresponding **greedy policy** is a *deterministic* policy that picks (one of the) actions that maximise the action value:

$$\pi_g(s) = a^* := \arg\max_a q(s, a).$$

Greedification results in policy improvement

- To improve the policy, apply successive iteration steps:

$$\pi_k \quad \xrightarrow{eval} \quad v_k, q_k \quad \xrightarrow{greedify} \quad \pi_{k+1}$$

- Greedification implies deterministic action choice at each s:

$$\pi_{k+1}(s) = a^* \qquad \text{iff} \qquad q_k(s, a^*) = \max_a q_k(s, a)$$

- Hence value function increases, i.e. policy has been improved!

$$\begin{aligned} v_{k+1}(s) &= q_k(s, a^*) && (\pi_{k+1} \text{ is deterministic at } s) \\ &= \max_a q_k(s, a) \\ &\geq v_k(s) && (= \textstyle\sum_{a'} \pi_k(a' \mid s) q_k(s, a')) \end{aligned}$$

Greedification requires $q(s, a)$! model-free vs. model-based

Greedification: $\pi(s) := \arg\max_a q(s, a)$

Model-based: (a.k.a. planning, search)

- Suffices to estimate value function $v(s)$:
- $q(s, a)$ computed with Bellman's one-step look-ahead (i.e. use back-up diagram):

$$q(s, a) = \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma v(s') \right]$$

Model-free: (a.k.a. (optimal) control)

- $q(s, a)$ needs to be estimated from collected experiences;
- algorithms shift focus from $v(s)$ to $q(s, a)$;

Conclusion: for **model-free** RL

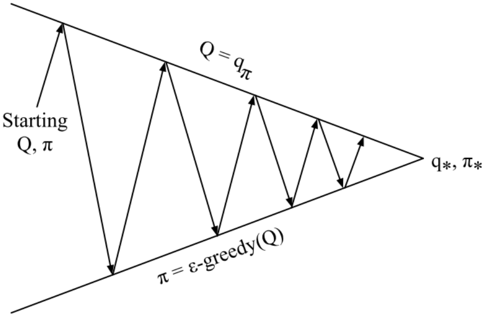
In contrast with to model-based, model-free RL is distinct:

- Focus on $q(s, a)$ rather than $v(s)$
- Make sure all state-action pairs (s,a) are sampled: importance of exploration!

Policy Iteration (PI) for Model-free RL

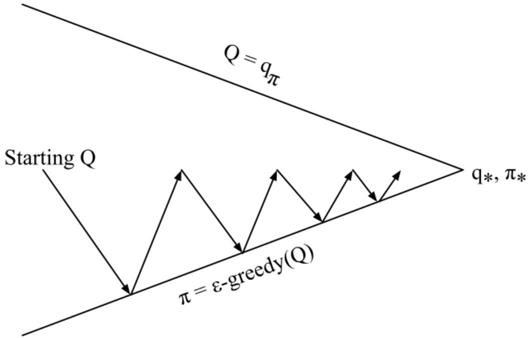
Goal: Find optimal policy π^* (aka optimal control)

- PI: Combine policy evaluation with policy improvement
  - Make Q-function consistent with policy π, i.e. $Q = q_\pi$;
  - Greedify policy: $\pi = \text{greedy}(Q)$;
- Introduce exploration in policy (e.g. $\pi = \varepsilon\text{-greedy}(Q)$);

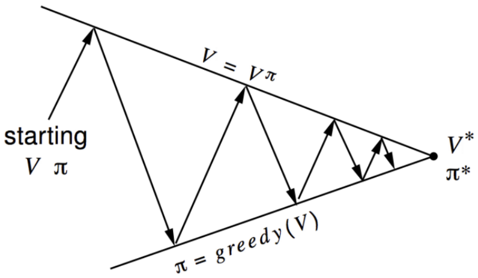


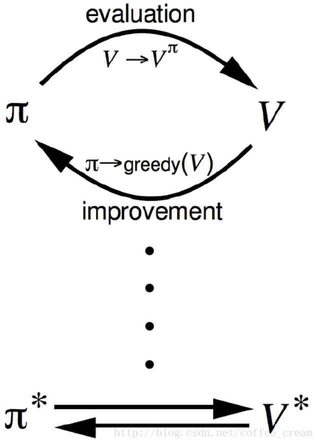
Generalized Policy Iteration (GPI)

- Impatient greedification: No insistence on making Q-function fully consistent with current policy π;
  Hence $Q \approx q_\pi$, but not necessarily $Q = q_\pi$;
- Ensure exploration: use $\pi = \varepsilon$-greedy(Q), not $\pi =$ greedy(Q);
- Works if both processes continue updating all states;



Generalized Policy Iteration (GPI) schematically





Policy evaluation Estimate v_π

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

Outline

Monte Carlo methods

- **Monte Carlo methods** are versatile statistical techniques for estimating properties of complex systems via **random sampling**;
- Example 1: if $X \sim p(x)$ and φ some function, then for any (sufficiently large) i.i.d. sample: $X_1, X_2, \ldots, X_n$:

$$E(\varphi(X)) = \int \varphi(x)\, p(x) dx \quad \approx \quad \frac{1}{n} \sum_{X_i \sim p} \varphi(X_i)$$

- Example 2: **Kullback-Leibler**

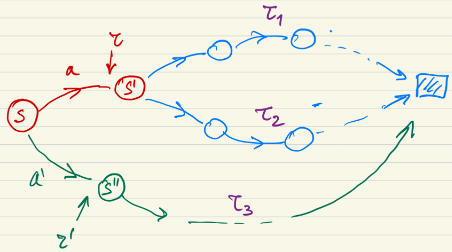
$$D_{KL}(f; g) = \int f(x) \log \frac{f(x)}{g(x)}\, dx \quad \approx \quad \frac{1}{n} \sum_{X_i \sim f} \log \frac{f(X_i)}{g(X_i)}$$

Model-free **policy evaluation**: MC-based estimation of v_π and $q_\pi(s, a)$

- Pick a starting state s (at random or systematic sweep)
- **Estimation of** $v_\pi(s)$
  - Use policy π to generate the initial and all subsequent actions (till terminal state)
  - Compute total return $r_{tot} = r_1 + r_2 + \ldots + r_T$ along path:
  - r_{tot} yields one sample value for $v_\pi(s)$
- **Estimation of** $q_\pi(s, a)$
  - Apply action a in state s, observe reward r_1 and new state s'
  - Use policy π to generate all subsequent actions (from s' till terminal state)
  - Compute total return $r_{tot} = r_1 + r_2 + \ldots + r_T$ along path:
  - r_{tot} yields one sample value for $q_\pi(s, a)$

Monte-Carlo for model-free **policy evaluation**

Monte Carlo: using **sampled** episodes to estimate $q(s, a)$!



$$\left.\begin{aligned} G(\tau_1) &= \sum_{\tau_1} r_i \\ G(\tau_2) &= \sum_{\tau_2} r_i \end{aligned}\right\} \rightarrow q_\pi(s,a)$$

$$G(\tau_3) = \sum_{\tau_3} r_i \longrightarrow q_\pi(s,a')$$

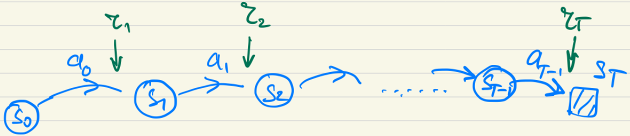
If $a, a' \sim \pi$:

$$G(\tau_1), G(\tau_2), G(\tau_3) \rightarrow v_\pi(s)$$

MC estimation of $q(s, a)$ along trajectory

$$\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2 \ldots\ldots s_{T-1}, a_{T-1}, r_T, s_T\}$$

↳ trajectory

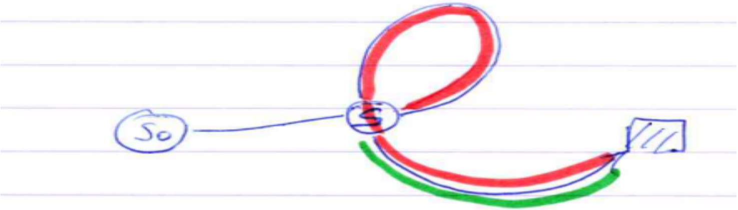


$$\begin{array}{cccc} G_0 \longrightarrow & G_1 \longrightarrow & G_2 \longrightarrow & \cdots \\ \downarrow & \downarrow & \downarrow & \\ q_\pi(s_0,a_0) & q_\pi(s_1,a_1) & q_\pi(s_2,a_2) & \cdots \end{array}$$

MC policy evaluation: **First** versus **every visit** estimate

? ? ?

- **First-visit MC**: average returns only for the first time s is visited in an episode
- **Every-visit MC**: average returns for every time s is visited in an episode:
  - More sample efficient: more samples per episode;
  - Samples no longer independent
- Both converge asymptotically 渐进的

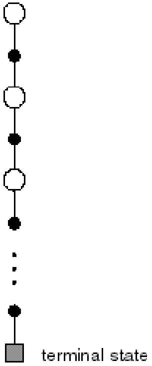


Monte-Carlo estimation of q-values

- Can learn q_π by averaging returns obtained when following π after taking action a in state s
- Converges asymptotically if every (s, a) visited infinitely often
  - Requires explicit exploration of actions not favored by π
  - Possible solutions:
    - **Exploring starts**: every (s, a) has a non-zero probability of being the starting pair
    - **Soft policies**: $\pi(a \mid s) > 0$ for all (s, a). E.g. ϵ-greedy

Monte-Carlo backup diagram

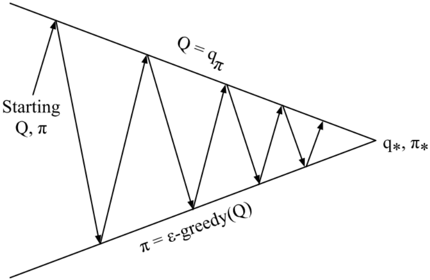
- Unlike dynamic programming, MC has only one choice at each state (i.e. the one actually taken!)
- Unlike dynamic programming, entire episode included: MC does not bootstrap ? ?
- Bellman equations are **NOT** used!



Model-free MC: From **Evaluation** to **Control**

Control: Find optimal policy π^*:

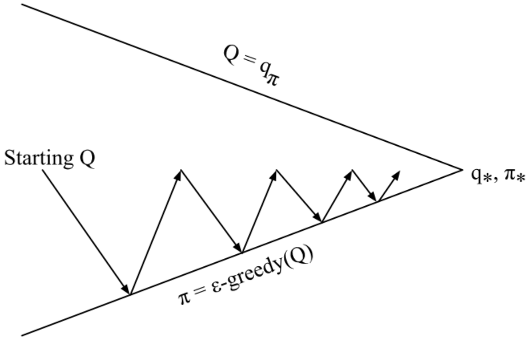
- Combine MC-based policy evaluation with improvement (e.g. greedification)
- Introduce exploration in policy
    - (e.g. ϵ-greedy)



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

Policy improvement ϵ-greedy policy improvement

Monte Carlo for Model-free Control (2)



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Monte-Carlo for **model-free estimation and control**

- MC provides one way to perform **model-free reinforcement learning** (RL): finding optimal policies without an explicit model for the MDP;
- MC for RL learns from **complete sample returns** in episodic tasks:
- Computes **value functions** using **direct sampling** **rather than Bellman equations**;

Convergence condition: Greedy in the Limit with Infinite Exploration (GLIE)

Def (GLIE)

- All state-action pairs are explored infinitely often:

$$\lim_{t\to\infty} N_t(s,a) = +\infty.$$

- The policy converges to a greedy policy:

$$\lim_{t\to\infty} \pi(a \mid s) = \begin{cases} 1 & \text{if} \quad a = \arg\max_{a'} q(s,a') \\ 0 & \text{otherwise} \end{cases}$$

Example: ϵ-greedy is GLIE iff $\epsilon \downarrow 0$.

GLIE Monte Carlo Model-free Control

- Sample kth episode using π: $\{S_1, A_1, R_2, ..., S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

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

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$

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

Theorem

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

Control based on MC+exploring starts: Algorithm

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
- $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
- $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
- $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
- Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
- Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$
- Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
    - Append G to $Returns(S_t, A_t)$
    - $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
    - $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Notice: exploration based on exploring starts, not ϵ-greedy!

Outline

Model-based versus Model-free

Monte Carlo for Model-free Policy Evaluation

Temporal Difference Methods for Model-free Prediction

SARSA and Q-Learning for Model-free Control

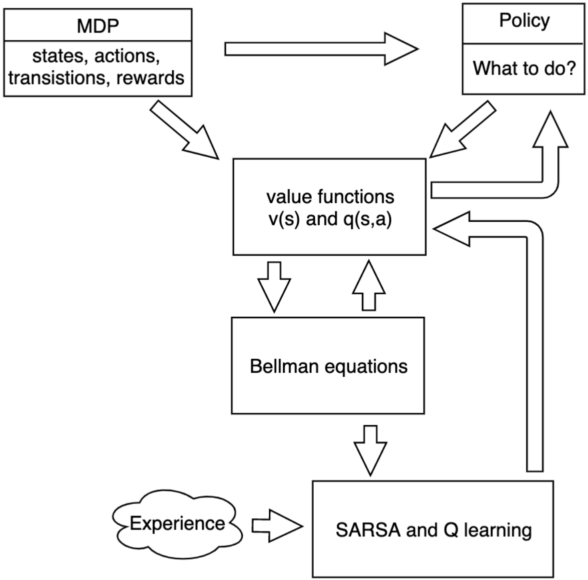
Integrating Planning and Learning

Solving model-free RL using **Temporal Differencing**

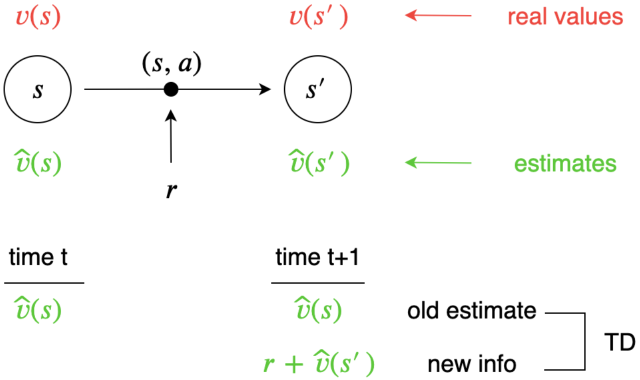
- By **acting in the world**, the agent gains experience.
  An experience can be expressed as a 4-tuple: (s, a, r, s')
- Over time the agent collects a list of experiences that he uses to find value functions (and corresponding policy) ... HOW?
- **Key observations:**
  - Bellman eqs. link values in neighbouring states along paths!
  - This backing-up improves learning efficiency!
  - Basis for RL algo's (SARSA, Q-learning, etc)

Temporal Differencing (TD): Use Bellman eqs to propagate values!

Exploiting the Bellman equations



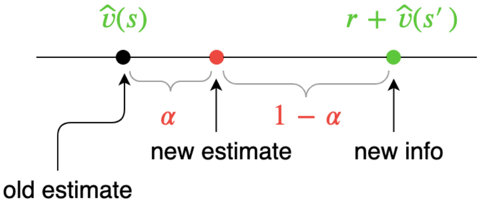
Temporal-differencing: Exploiting Bellman eqs.



The 1-step reward r is new information gained from experience.

Temporal differencing for v: Exploiting Bellman eqs.

time t	time t+1		
$\hat{v}(s)$	$\hat{v}(s)$	old estimate	TD
	$r + \hat{v}(s')$	new info	



$$\hat{v}_{\text{new}}(s) = (1 - \alpha)\hat{v}_{old}(s) + \alpha\big(r + \hat{v}(s')\big)$$

Temperal Differencing (TD) versus Monte Carlo (MC)

Given learning rate α

- **MC update rule**:

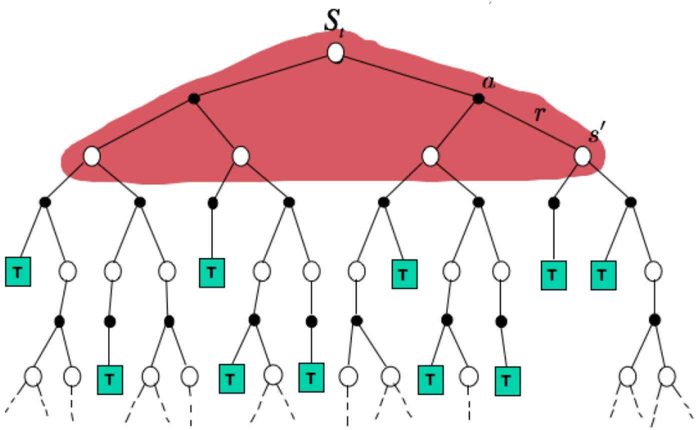
$$v_\pi^{new}(S_t) \leftarrow (1-\alpha)v_\pi^{old}(S_t) + \alpha G_t(S_t)$$
$$v_\pi^{new}(S_t) \leftarrow v_\pi^{old}(S_t) + \alpha \left[G_t(S_t) - v_\pi^{old}(S_t)\right]$$

- **TD update rule:** uses a different **update target**:

$$v_\pi^{new}(S_t) \leftarrow (1-\alpha)v_\pi^{old}(S_t) + \alpha[R_{t+1} + \gamma v_\pi(S_{t+1})]$$
$$v_\pi^{new}(S_t) \leftarrow v_\pi^{old}(S_t) + \alpha[R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi^{old}(S_t)]$$

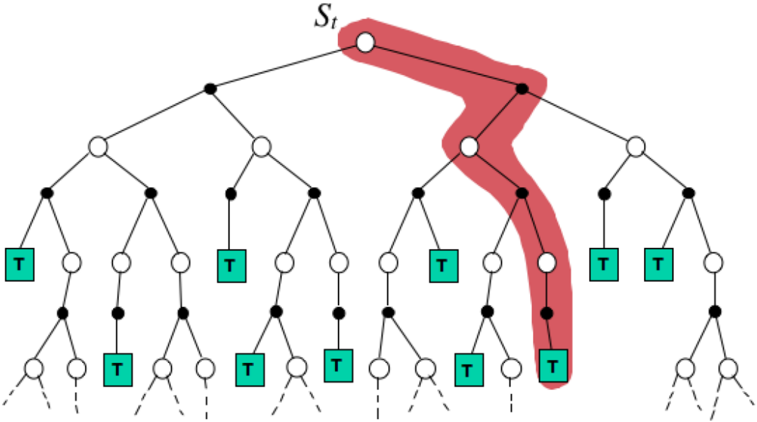
- TD is a **bootstrapping method**:
  bases updates on existing estimates, like DP

Dynamic Programming (DP): Backup diagram



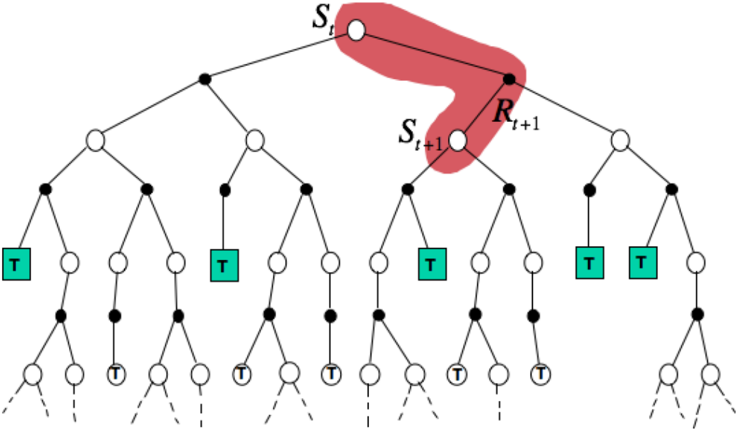
Monte-Carlo (MC): Backup diagram

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



Temporal Difference (TD): Backup diagram

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



Temporal-difference (TD) methods

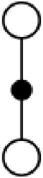
- **DP** exploits Bellman equation but **requires model**
- **MC** doesn't require model but **doesn't exploit Bellman** equation
- **TD methods** can get the best of both worlds: exploit Bellman equation without requiring a model
- TD is therefore **core algorithm** for **model-free RL**

Policy evaluation: Estimating v_π using TD(0)

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        $a \leftarrow$ action given by π for s
        Take action a; observe reward, r, and next state, s'
        $V(s) \leftarrow V(s) + \alpha\big[r + \gamma V(s') - V(s)\big]$
        $s \leftarrow s'$
    until s is terminal

TD(0) backup diagram

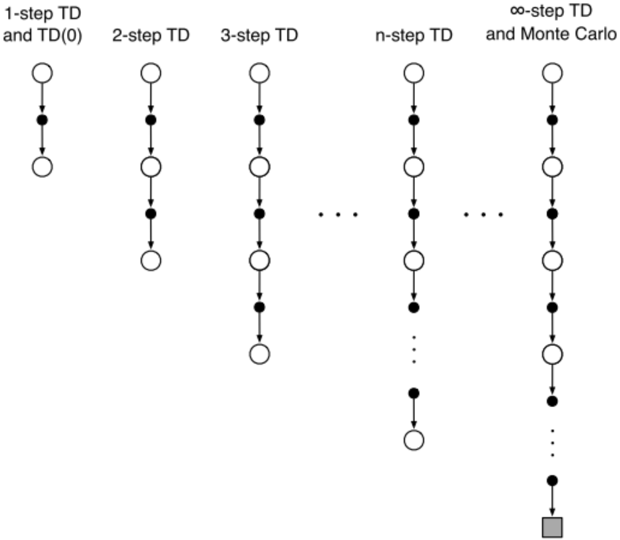
- Sampling: unlike DP but like MC, only one choice at each state
- Bootstrapping: like DP but unlike MC, use estimate from next state



Advantages of TD prediction methods

- TD methods require only experience, not a model
- TD, but not MC, methods can be fully incremental
- Learn before knowing the final outcome:
  - Efficient: less memory and peak computation
  - Learn from incomplete sequences
- Both MC and TD converge but TD tends to be faster

n-step TD: Spectrum between TD(0) and Monte Carlo



Issues when addressing model-free RL problems

1. **We need to compute $q(s, a)$ rather than $v(s)$**
   - To **improve policy**, for every s, need to know $\max_a q(s, a)$:

     construct $\pi : s \mapsto a_{max}$ where $q(s, a_{max}) = \max_{a'} q(s, a')$

     - **Model-based:** $q(s, a)$ can be computed from $v(s)$:

       $$q(s, a) = r(s, a, s') + v(s')$$

     - **Model-free:** $q(s, a)$ needs to be estimated explicitly!

   Hence: SARSA and Q-learning focus on $q(s, a)$

2. **Keep exploring!**
   - Balance exploration versus exploitation
   - E.g. ϵ-greedy, soft-max, etc.

Outline

Apply TD methods for estimating $q(s, a)$

TD methods Exploit correlations between successor states:

- **State value function** $v_\pi(s)$

$$v_\pi(S_t) \quad \longleftarrow \quad v_\pi(S_t) + \alpha(\underbrace{R_{t+1} + \gamma v_\pi(S_{t+1})}_{\text{new info}} - v_\pi(S_t))$$

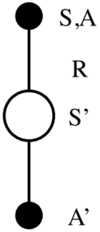
- **State value function** $q_\pi(s, a)$**:**
  - **SARSA:** policy evaluation

$$q_\pi(s, a) \leftarrow ??$$

  - **Q-learning:** optimal state-action value

$$q^*(s, a) \leftarrow ??$$

SARSA: TD estimation of Q-values



- **SARSA** In state s, take action a, transition to state s', use policy π to select next action a'
- **Bellman:** Two estimate for state-action value:
  $q_\pi(s,a)$ and $r(s,a,s') + q_\pi(s',a')$

SARSA: TD for action values

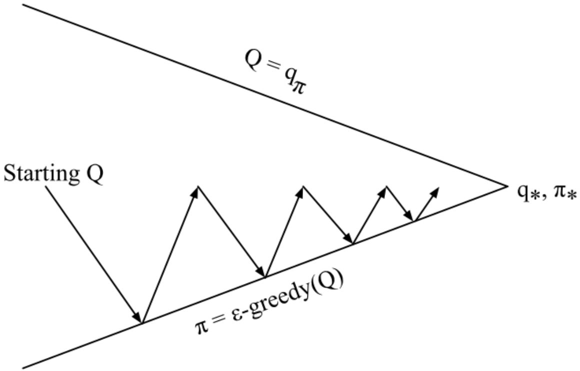
- **TD:** bootstrap update for **value function** v_π

$$v_\pi(s) \quad \longleftarrow \quad v_\pi(s) + \alpha(\underbrace{r(s,a,s') + \gamma v_\pi(s')}_{\text{new info}} - v_\pi(s))$$

- **SARSA:** bootstrap update for **action value function** q_π

$$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha(\underbrace{r(s,a,s') + q_\pi(s',a')}_{\text{new info}} - q_\pi(s,a))$$

SARSA model-free control: Schematically



Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

SARSA: Pseudo-code for model-free control

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

SARSA convergence

Theorem

Sarsa converges to the optimal action-value function, $Q(s, a) \to q_*(s, a)$, under the following conditions:

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

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

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

Applying TD to model-free RL problems

Recall: Model-free, hence focus on $q(s, a)$ and use Bellman!

1. **SARSA**: Evaluating a given policy π
   - $q_\pi(s, a) = \sum_{s'} p(s' \mid s, a)\Big[r(s, a, s') + \sum_{a'} \pi(a' \mid s') q_\pi(s', a')\Big]$
   - **Sample:** $q_\pi(s, a) = r(s, a, s') + q_\pi(s', a')$
   - SARSA update rule (sample-based):

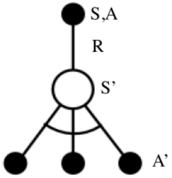
   $$q_\pi(s, a) \leftarrow (1 - \alpha) q_\pi(s, a) + \alpha(r(s, a, s') + q_\pi(s', a'))$$

2. **Q-learning:** Estimating the **optimal** value function $q^*(s, a)$
   - $q^*(s, a) = \sum_{s'} p(s' \mid s, a)\Big[r(s, a, s') + \max_{a'} q^*(s', a')\Big]$
   - **Sample:** $q^*(s, a) = r(s, a, s') + \max_{a'} q^*(s', a')$
   - Q-learning update rule (sample-based):

   $$q^*(s, a) \leftarrow (1 - \alpha) q^*(s, a) + \alpha(r(s, a, s') + \max_{a'} q^*(s', a'))$$

Q-learning:

Bootstrap with **best** action, not actual action



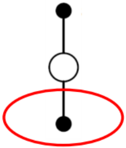
$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S',a') - Q(S,A) \right)$$

Convergence when TD-error vanishes:
Recall: optimality equation for q^*:

$$q^*(s,a) = \sum_{s'} p(s' \mid s,a) \left[r(s,a,s') + \gamma \max_{a'} q^*(s',a') \right]$$

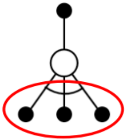
SARSA vs. Q-learning: On-Policy vs. Off-Policy

Sarsa:



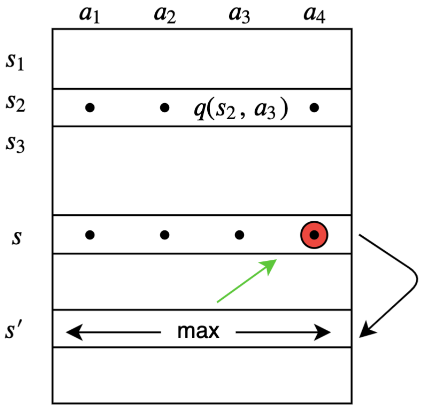
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Q-learning:

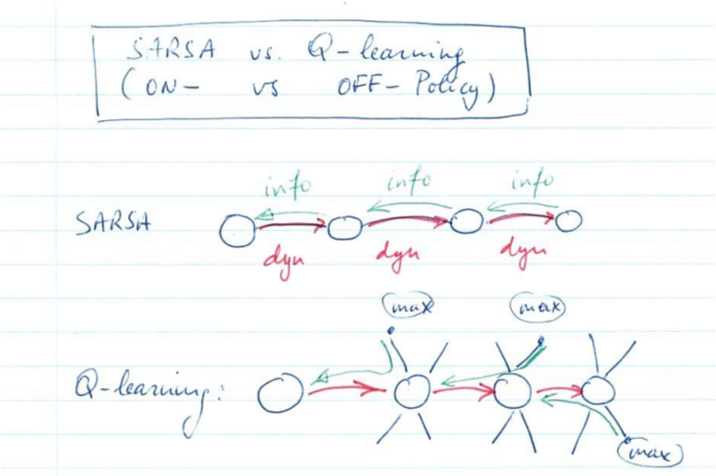


$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

Q-learning



SARSA (on-policy) vs. Q-Learning (off-policy)



Q-learning: dynamics policy different from backup policy

ON-policy vs. OFF-policy

ON-policy

- Info gathered to improve policy, depends on that policy;

$$\text{Feedback loop:} \quad \text{policy} \longleftrightarrow \text{data}$$

- Sample inefficient: experiences (s, a, r, s', a') cannot be re-used when policy changes since a' depends on actual policy!

OFF-policy

- Improved sample efficiency: can re-use all samples for training;
- E.g. in Q-learning: use data generated by dynamics policy π to learn value function for optimal policy π^*.

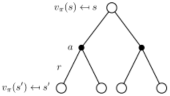
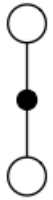
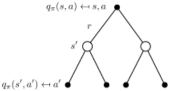
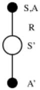
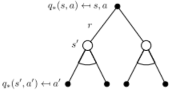
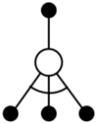
Q-learning ALGO: off-policy TD control

Make TD off-policy: bootstrap with best action, not actual action:

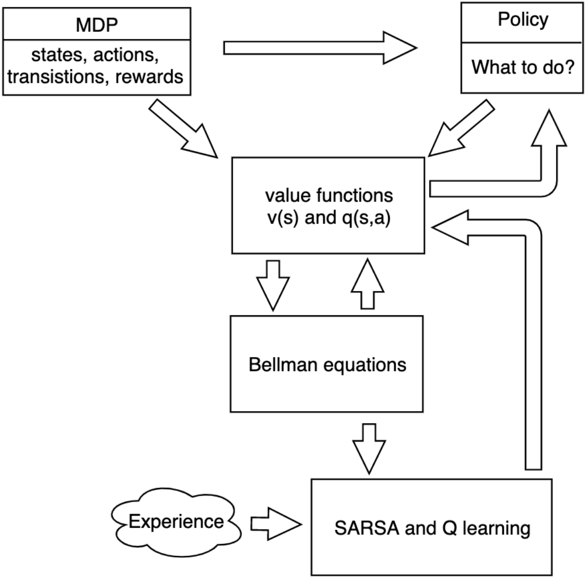
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$;
  until s is terminal

Dynamic Program^{ing} (DP) vs. Temporal Diff^{ing} (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

Summary


MDP
states, actions,
transistions, rewards
Policy
What to do?
value functions
v(s) and q(s,a)
Bellman equations
Experience
SARSA and Q learning

Summary: Model-based versus Model-free

	Prediction *Estimation:* *Given π, what is v?*	**(Optimal) Control** *Optimisation:* *What is optimal π?*
model-based (MDP given)	Policy evaluation using Dyn. Programming (DP)	Policy improvement (+ Policy evaluation) = Policy iteration
model-free (MDP unknown)	Monte Carlo (MC) Temporal Diff^{ing} (TD) = "impatient MC" *bootstrapping!*	Generalized Policy Iteration *"simultaneous"*

Outline

Model-based versus Model-free

Monte Carlo for Model-free Policy Evaluation

Temporal Difference Methods for Model-free Prediction

SARSA and Q-Learning for Model-free Control

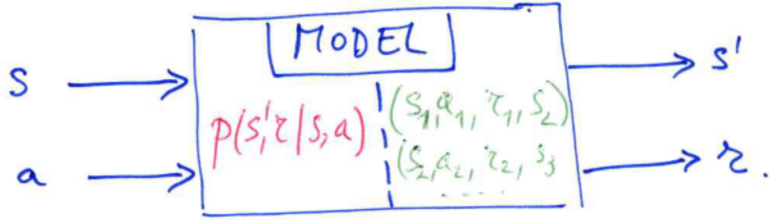
Integrating Planning and Learning

Dyna-Q; Integrating planning and learning

Model: tells agent what will happen next . . .

- **Model-based**: planning
- **Model-free**: learning

Distributional vs. Sample-based Model

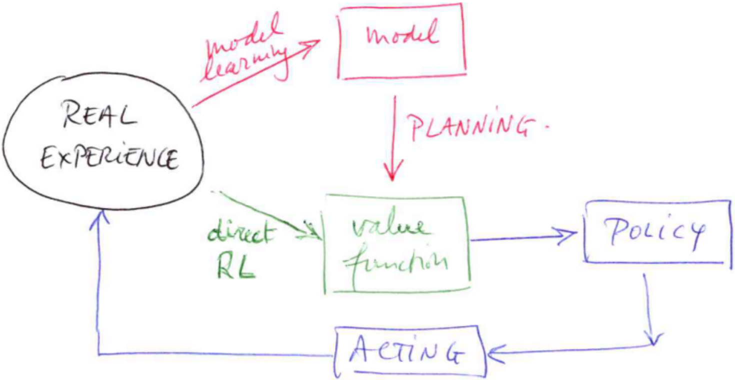


Model-based learning

- Until now: Model = fully specified MDP!
- More generally: anything that helps the agent to plan:
- More accurate models are more effective (myths vs. science):
  - Folklore and weather saying:
    A wet and windy May fills the barn with corn and hay.
  - Meteorological models running on supercomputer

Dyna-Q; Integrating planning and learning

Real experience can be used in two ways:



DYNA-Q: Algorithm

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) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Repeat n times:

$S \leftarrow$ random previously observed state

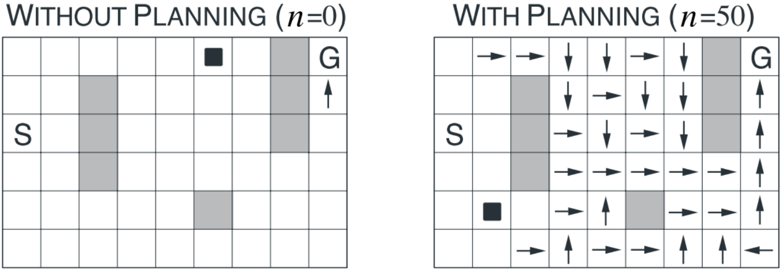
$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

DYNA-Q: Maze example

Greedy policy midway through 2nd episode:



More info: Sutton and Barto, sections 8.2-8.3