Reinforcement Learning

Amy Zhang

Machine Learning Summer School 2024

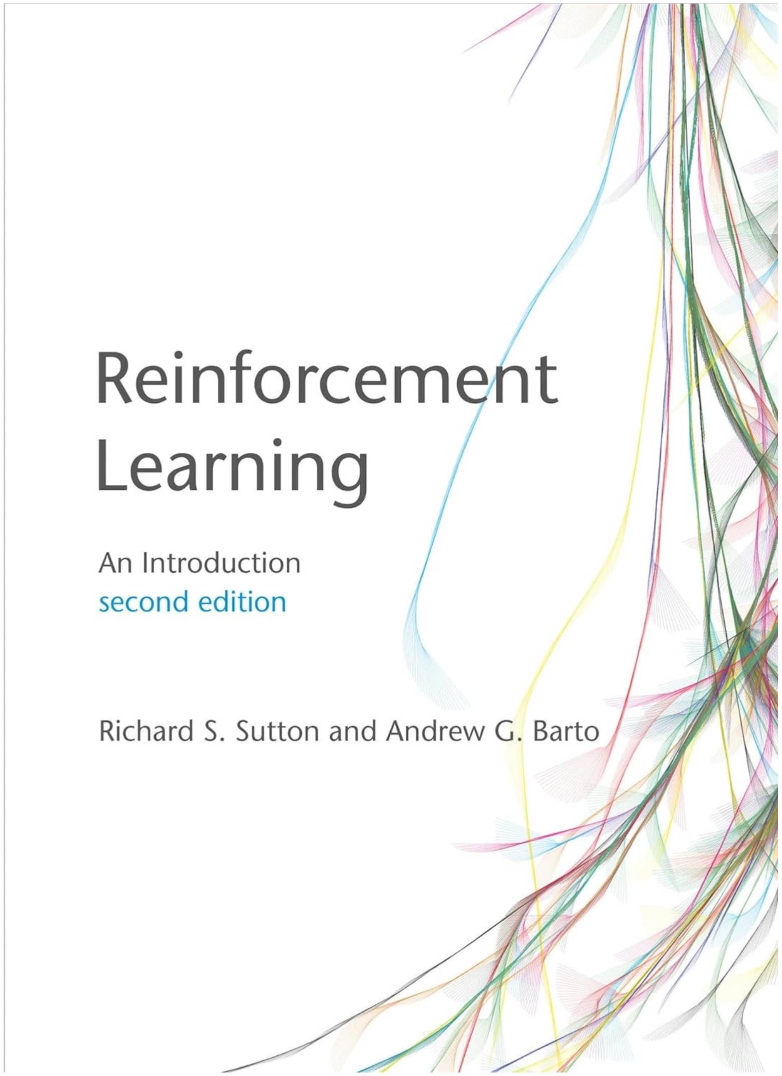
OIST

Outline: First Half

- What is Reinforcement Learning and when should I use it?
- Finite Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Planning
- Deadly Triad

Outline: Second Half

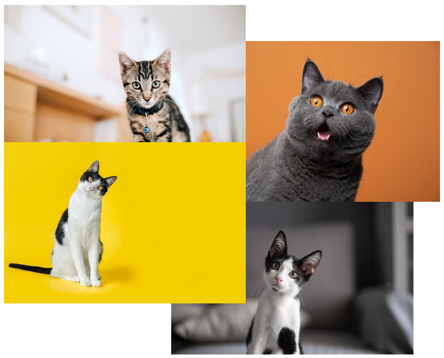
- Function Approximation: Model-free Methods
  - DQN
  - REINFORCE and Policy gradient
  - Actor-Critic Methods
- Function Approximation: Model-based Methods
  - Dyna
  - MBPO
  - PETS
- Advanced Topics
  - Abstractions and Generalization
  - Leveraging Structure in RL
  - Self-supervised RL



- http://incompleteideas.net/book/the-book-2nd.html

- **CS394R/ECE381V: Reinforcement Learning: Theory and Practice -- Spring 2024**
  https://www.cs.utexas.edu/~pstone/Courses/394Rspring24/

From a Supervised Learning Lens

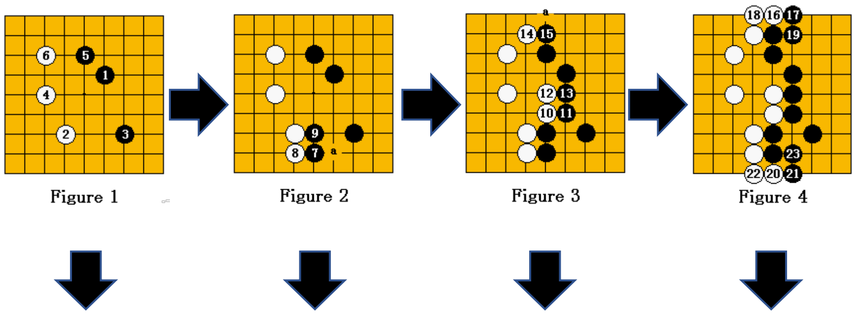


cat

What about sequential data?

dog

Still can be represented as a supervised learning problem

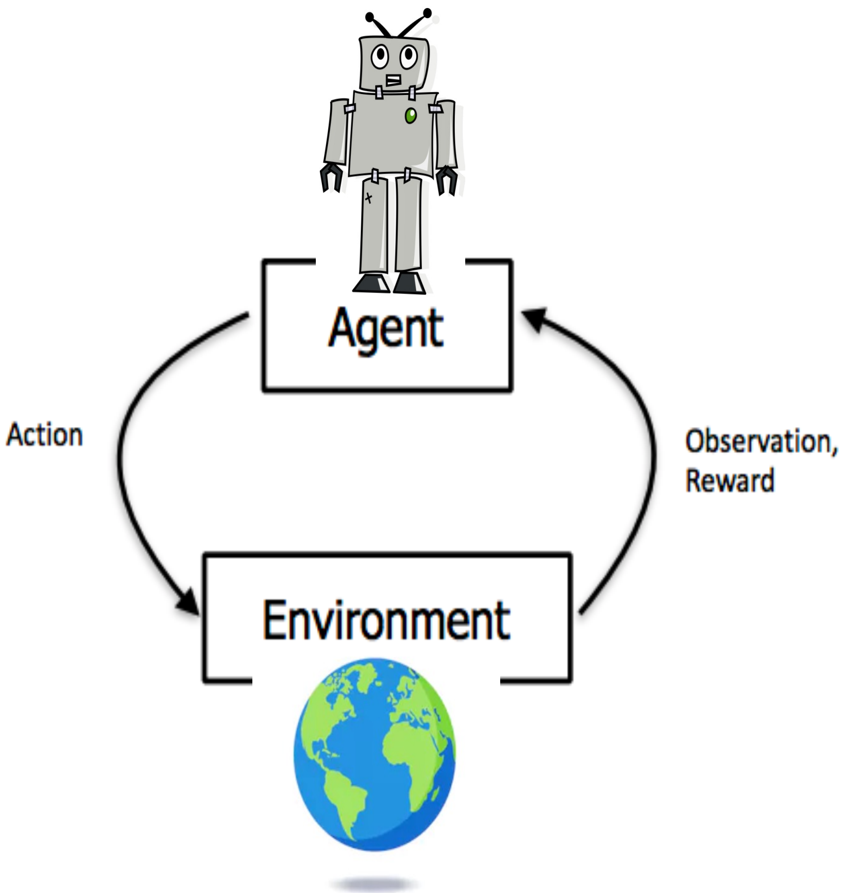


Desired action | Desired action | Desired action | Desired action

What if we don't know the best moves to take?

Learn via *trial and error:*

Learn from a reward signal and try to maximize that reward

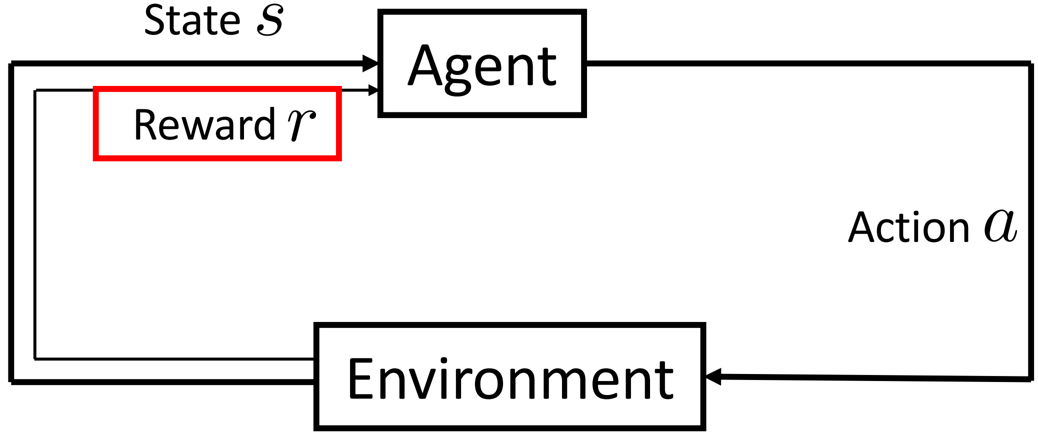


Reinforcement Learning Framework

Assumption: Environment is a Markov Decision Process

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ a set of actions,
- $p_0(\mathcal{S})$ is the initial state distribution,
- $T(s_{t+1}|s_t, a_t)$ is the probability of transitioning from state $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ after action $a_t \in \mathcal{A}$,
- $R(r_{t+1}|s_t, a_t)$ is the probability of receiving reward $r_{t+1} \in R$ after executing action a_t while in state s_t,
- $\gamma \in [0, 1)$ is the discount factor.

Goal of Reinforcement Learning



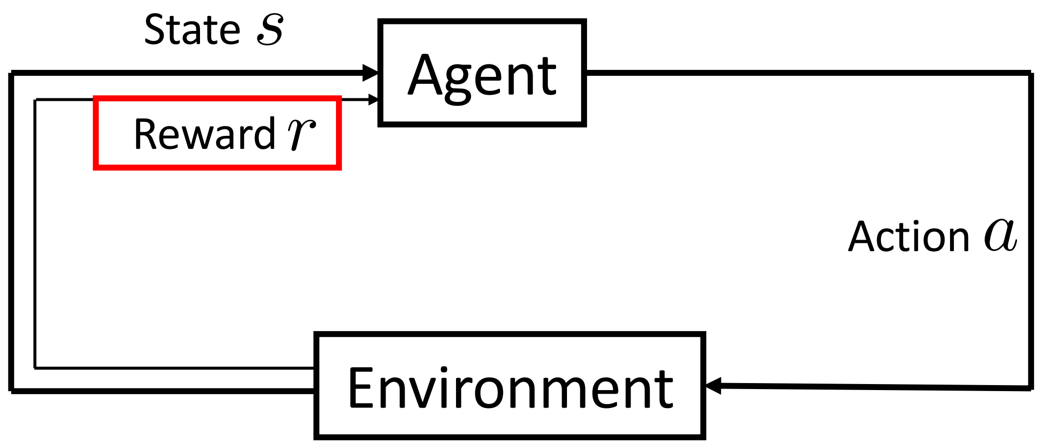
γ Discount factor

$$s_t \xrightarrow{a_t,\ r_t} s_{t+1} \xrightarrow{a_{t+1},\ r_{t+1}} s_{t+2} \xrightarrow{a_{t+2},\ r_{t+2}} \bullet\bullet\bullet \longrightarrow$$

τ Trajectory

Maximize long-term reward: $\max \sum_{t'=t}^{+\infty} \gamma^{t'-t} r_t$

Key Quantities

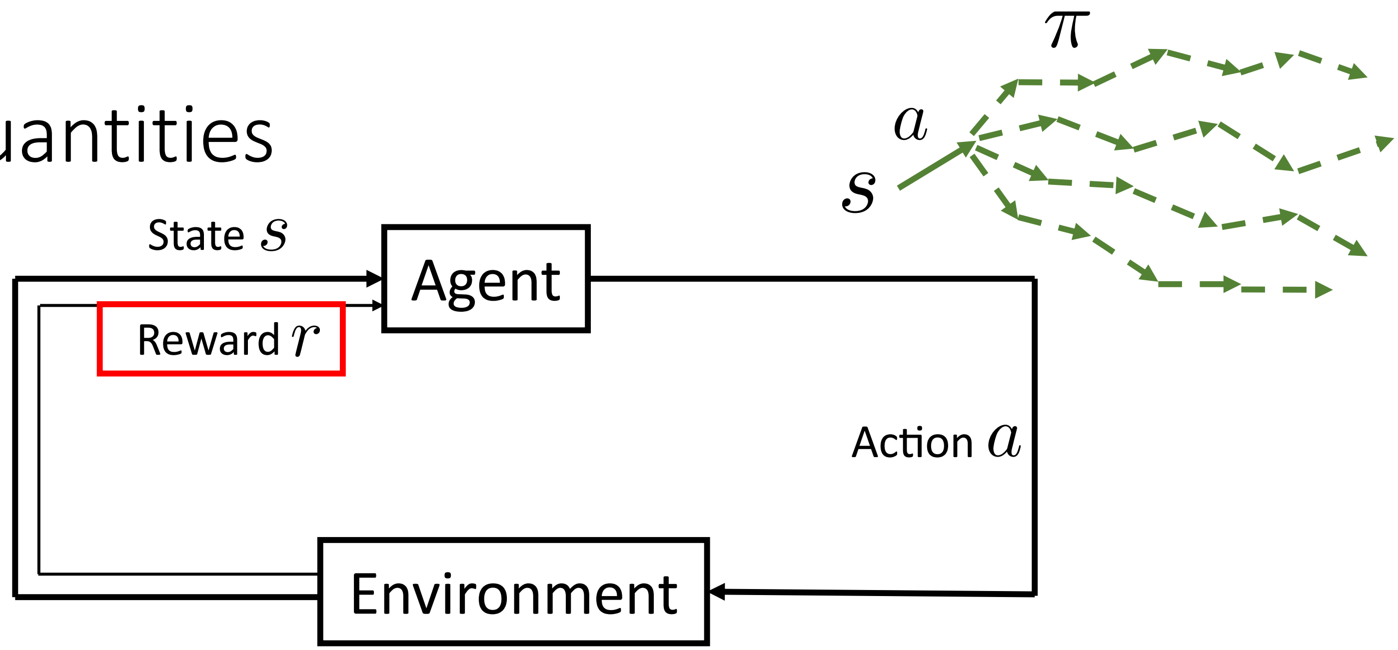


$V^*(s)$ Maximal reward you can get starting from state s

$Q^*(s, a)$ Maximal reward starting from s after taking action a

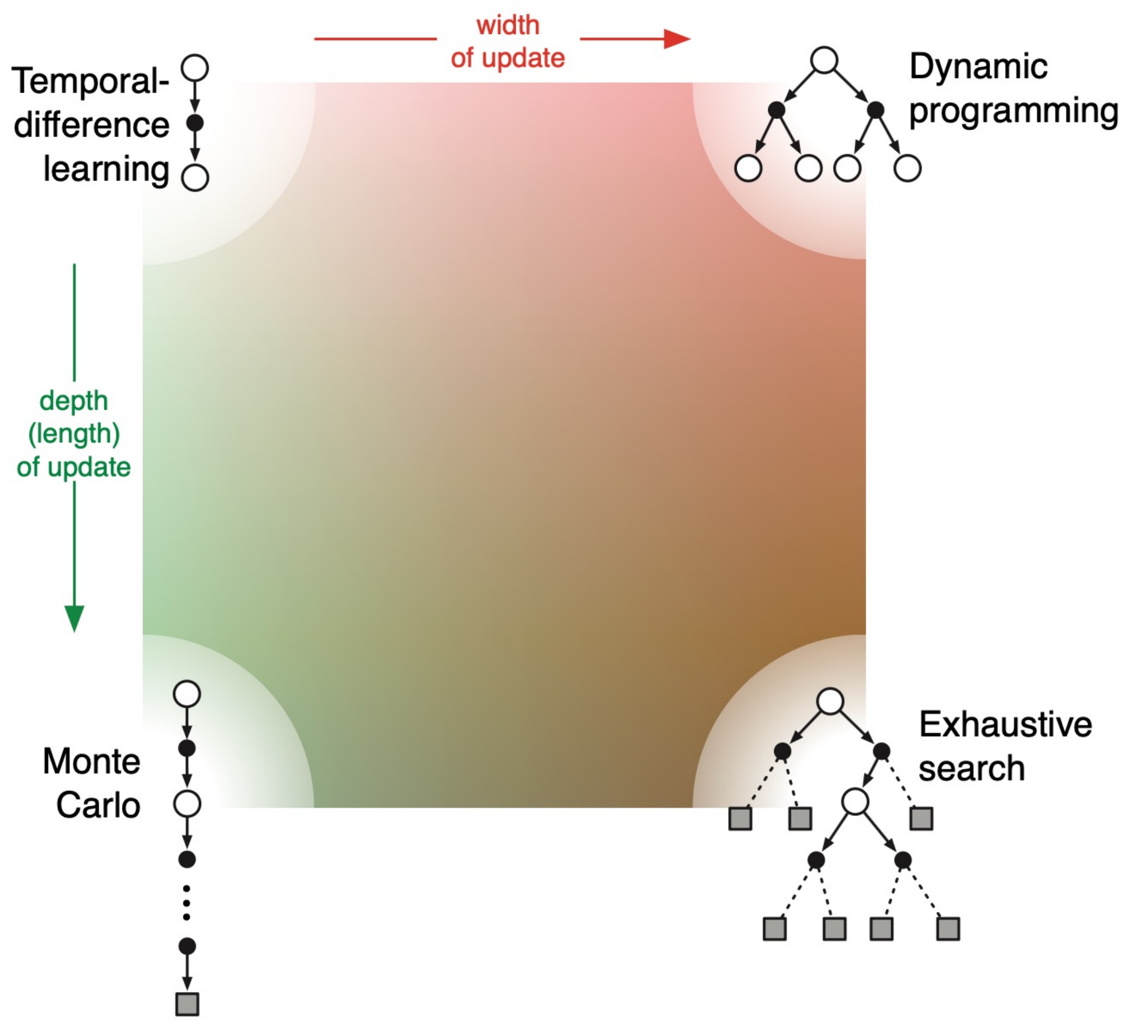
$\pi(a|s)$ Probability of taking action a given state s

Key Quantities



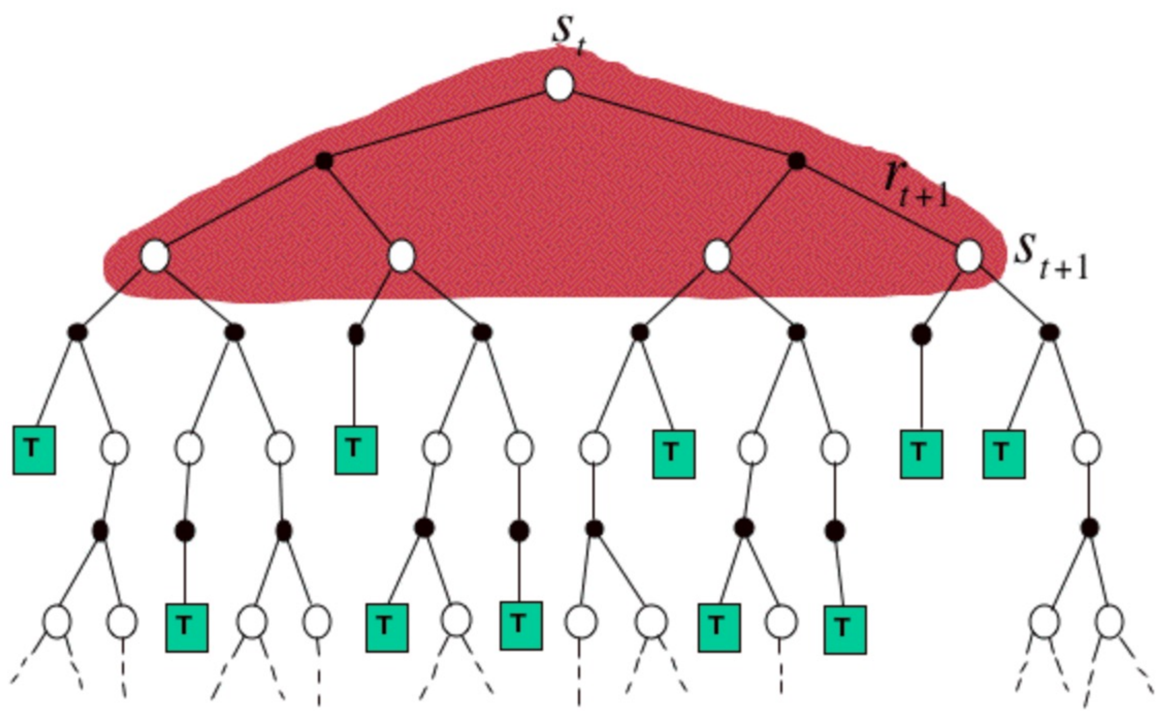
$V^{\pi}(s)$ Reward you can get, starting from s following policy π

$Q^{\pi}(s, a)$ Reward starting from s after taking action a and following π


width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search

cf. Dynamic Programming

$$V(s_t) \leftarrow E_\pi \left\{ r_{t+1} + \gamma\, V(s_t) \right\}$$



R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

Iterative Policy Evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(\textit{terminal})$ to 0

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Dynamic Programming: Policy Improvement

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_*(s') \Big], \end{aligned}$$

or

$$\begin{aligned} q_*(s, a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \ \Big|\ S_t = s, A_t = a \Big] \\ &= \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma \max_{a'} q_*(s', a') \Big], \end{aligned}$$

Policy Improvement Theorem

- If for all states

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

- Then for all states

$$v_{\pi'}(s) \geq v_\pi(s).$$

- A guarantee in the tabular setting that updates will always lead to improved policies, until convergence at the optimal value function.

Policy Improvement Theorem Proof

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \bigm| S_t = s\big] \\
&\le \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \bigm| S_t = s\big] \\
&\ \vdots \\
&\le \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \bigm| S_t = s\big] \\
&= v_{\pi'}(s).
\end{aligned}
$$

Takeaway

$$\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned}$$

$$\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma v_{\pi'}(s')\Big].
\end{aligned}$$

Policy improvement thus must give us a strictly better policy except when the original policy is already optimal.

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

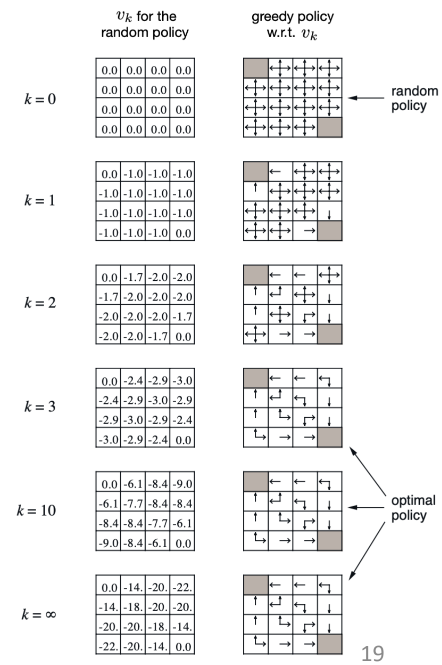
Policy evaluation

Policy improvement

Policy iteration drawbacks

- Convergence of policy evaluation can be expensive
- Maybe we don't need to converge policy evaluation to find the optimal policy

Convergence of iterative policy evaluation in a gridworld



Value Iteration

- We can speed up the previous algorithm by truncating the policy evaluation step:

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$

Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

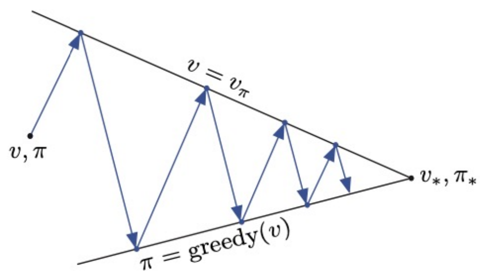
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

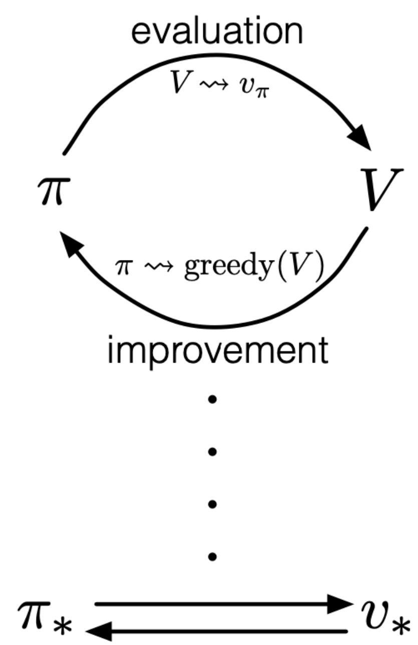
Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

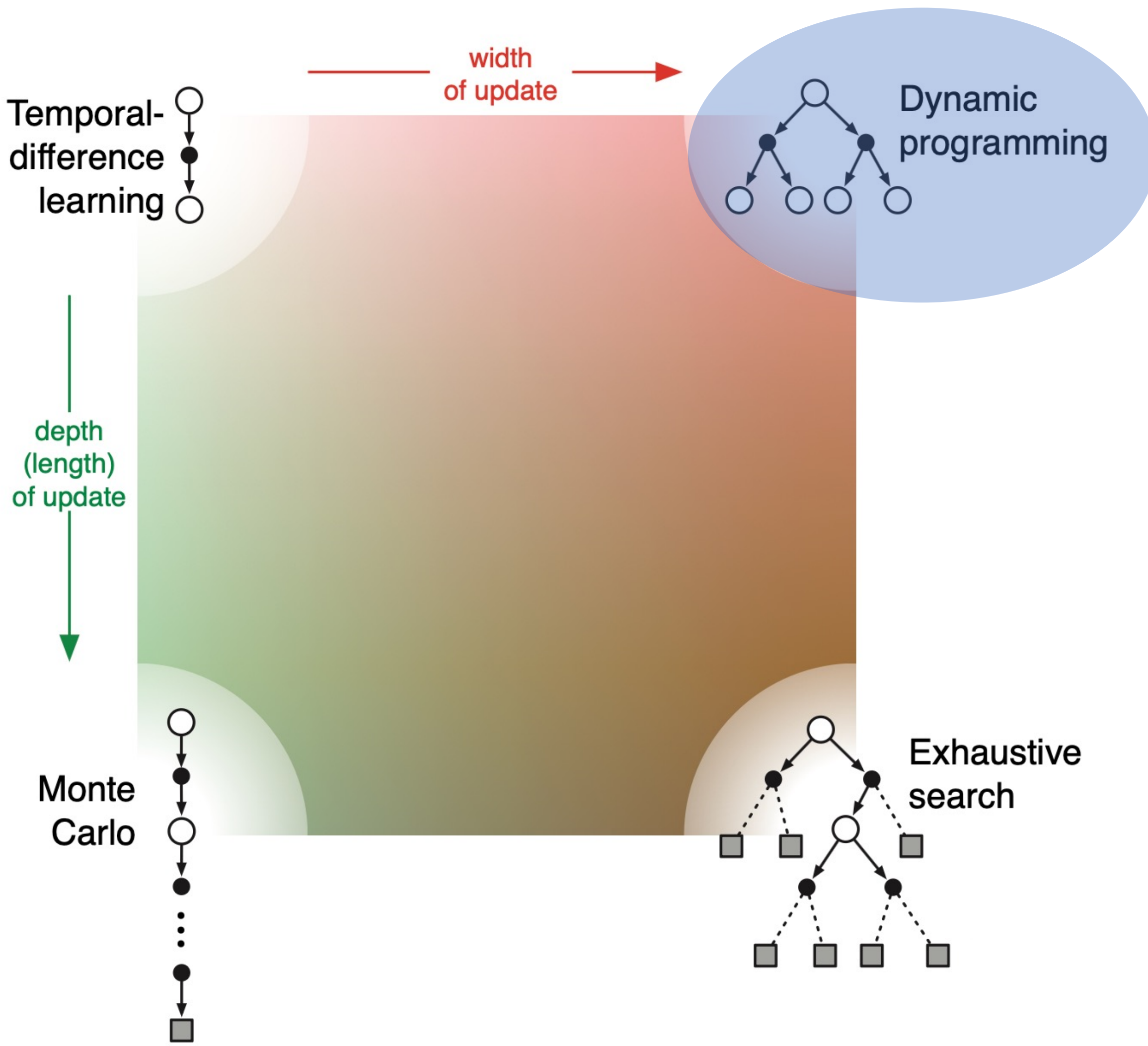
Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$

Generalized Policy Iteration

- We use the term generalized policy iteration (GPI) to refer to the general idea of letting policy-evaluation and policy-improvement processes interact, independent of the granularity and other details of the two processes.
- Almost all reinforcement learning methods are well described as GPI.






width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search

Monte Carlo Methods

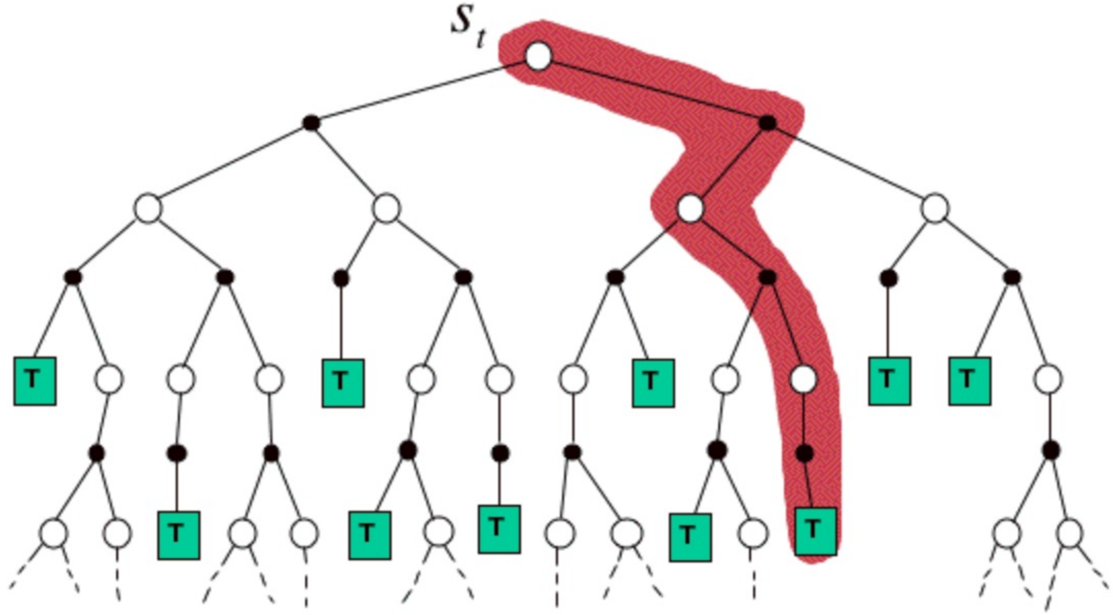
- Previous methods computed value functions using knowledge of the MDP
- Impractical assumption in most use cases

- How can we learn value functions from sample (Monte Carlo) returns instead?

Simple Monte Carlo

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

where R_t is the actual return following state s_t.



R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

Policy Evaluation Setting

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

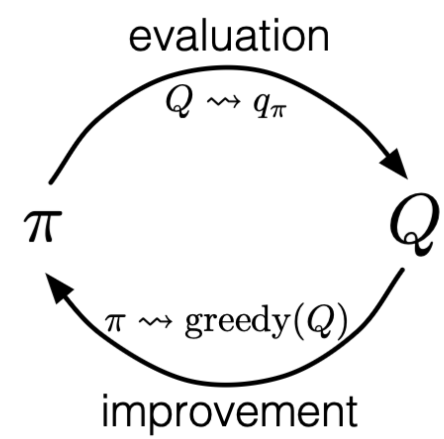
Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Monte Carlo Control

where E denotes a complete policy evaluation and I denotes a complete policy improvement.



$$\pi_0 \xrightarrow{\mathrm{E}} q_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} q_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} q_*,$$

Differences between DP and MC methods

DP methods:

- Require environment dynamics p(s’, r | s, a)
- Difficult to acquire in practice

MC methods:

- Don’t need environment dynamics p(s’, r | s, a)
- Only need environment samples!

Off-policy Prediction via Importance Sampling

- Simplest setting:
  - prediction problem
  - Both target and behavior policies are fixed

- Required assumptions:
  - Coverage assumption: Every action taken under the target policy is also taken under the behavior policy.

Off-policy Prediction via Importance Sampling

- Importance sampling: technique for estimating expected values under one distribution given samples from another.
- Probability of a state-action trajectory under any policy pi:

$$
\begin{aligned}
&\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&\quad = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&\quad = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\end{aligned}
$$

Relative probability of the trajectory under target and behavior policies

$$
\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}
$$

Dynamics cancel out - doesn’t depend on MDP!

Off-policy Prediction via Importance Sampling

Recall: wish to estimate expected returns under target policy, but only have returns under behavior policy.

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

$$\mathbb{E}[\rho_{t:T-1}G_t \mid S_t] = v_\pi(S_t)$$

Computing this expectation in practice requires a scaling term

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}.$$

Number of steps

Ordinary importance sampling

Off-policy Prediction via Importance Sampling

Recall: wish to estimate expected returns under target policy, but only have returns under behavior policy.

$$\rho_{t:T-1} \doteq \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t] = v_\pi(S_t)$$

Computing this expectation in practice requires a scaling term

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

Weighted importance sampling

Trade-offs of on-policy vs. off-policy

- On-policy method are simpler

- Off-policy methods:
  - Because the data is due to a different policy, off-policy methods are often of greater variance and are slower to converge.

  - More powerful and general. They include on-policy methods as the special case in which the target and behavior policies are the same.

Incremental Implementation of MC Prediction

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s, a) \in \mathbb{R}$ (arbitrarily)
    $C(s, a) \leftarrow 0$

Loop forever (for each episode):
    $b \leftarrow$ any policy with coverage of π
    Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
        If $W = 0$ then exit For loop

Off-policy Monte Carlo Control

Off-policy MC control, for estimating $\pi \approx \pi_*$

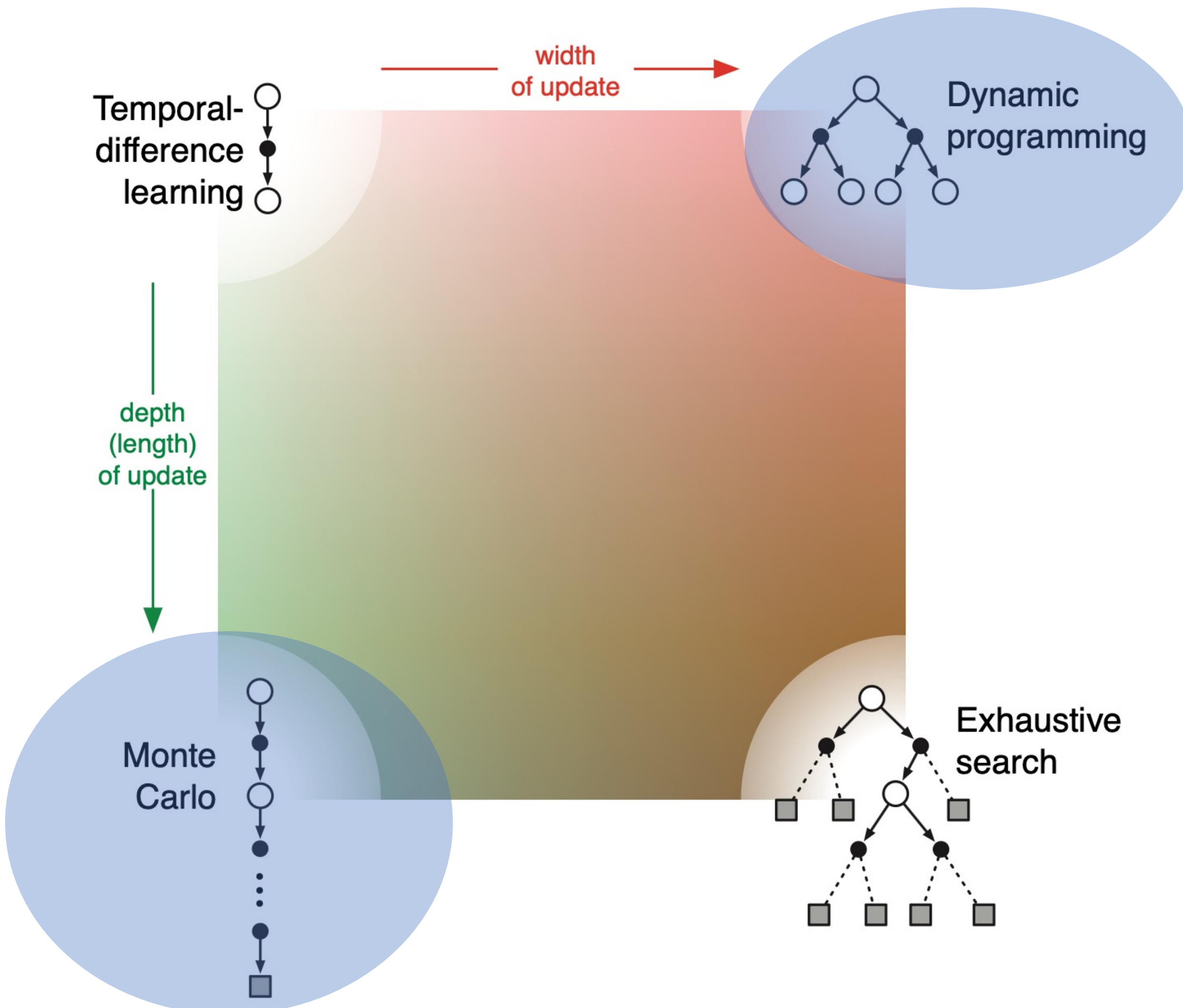
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s, a) \in \mathbb{R}$ (arbitrarily)
  $C(s, a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit For loop
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

Add a step of policy improvement

Recap: MC vs. TD

- MC doesn't need a (full) model
  - Can learn from actual or simulated experience
- DP takes advantage of a full model
  - Doesn't need any experience
- MC expense independent of number of states
- No bootstrapping in MC
  - Not harmed by Markov violations


width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search

TD Prediction

Policy Evaluation (the prediction problem):
for a given policy π, compute the state-value function V^{π}

Recall: Simple every - visit Monte Carlo method :

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

target: the actual return after time t

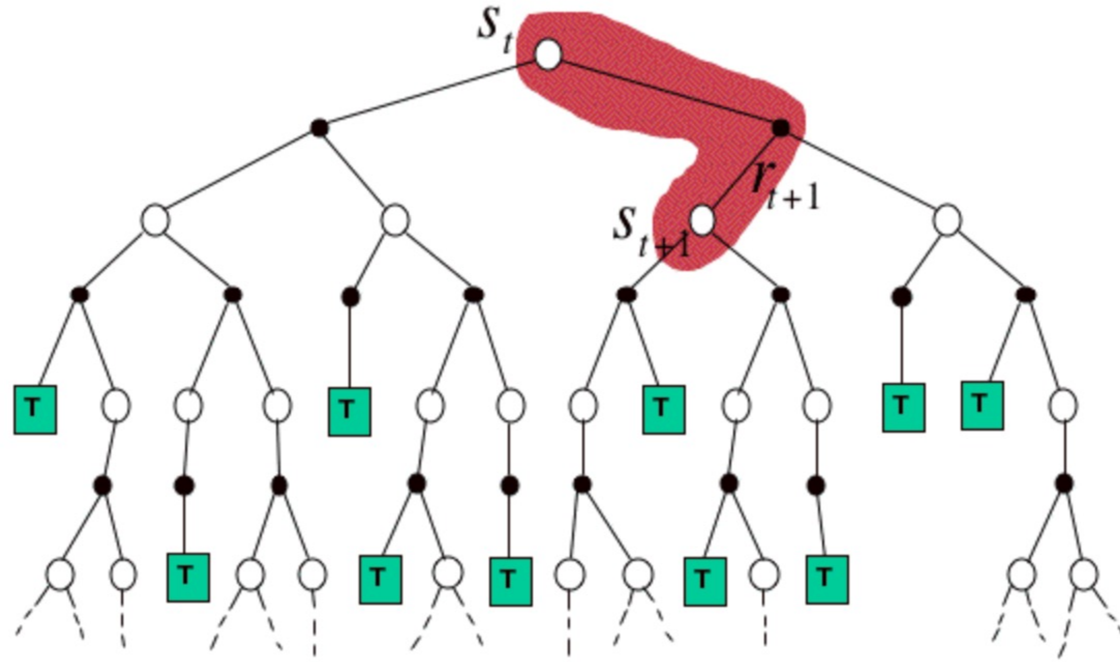
The simplest TD method, TD(0) :

$$V(s_t) \leftarrow V(s_t) + \alpha\left[\underbrace{r_{t+1} + \gamma V(s_{t+1})} - V(s_t)\right]$$

target: an estimate of the return

Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$



R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

TD methods bootstrap and sample

- Bootstrapping: update involves an *estimate*
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update does not involve an *expected value*
  - MC samples
  - DP does not sample
  - TD samples

Policy Evaluation Setting

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences
- Both MC and TD converge (under certain assumptions to be detailed later), but which is faster?

Sarsa: On-policy TD control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

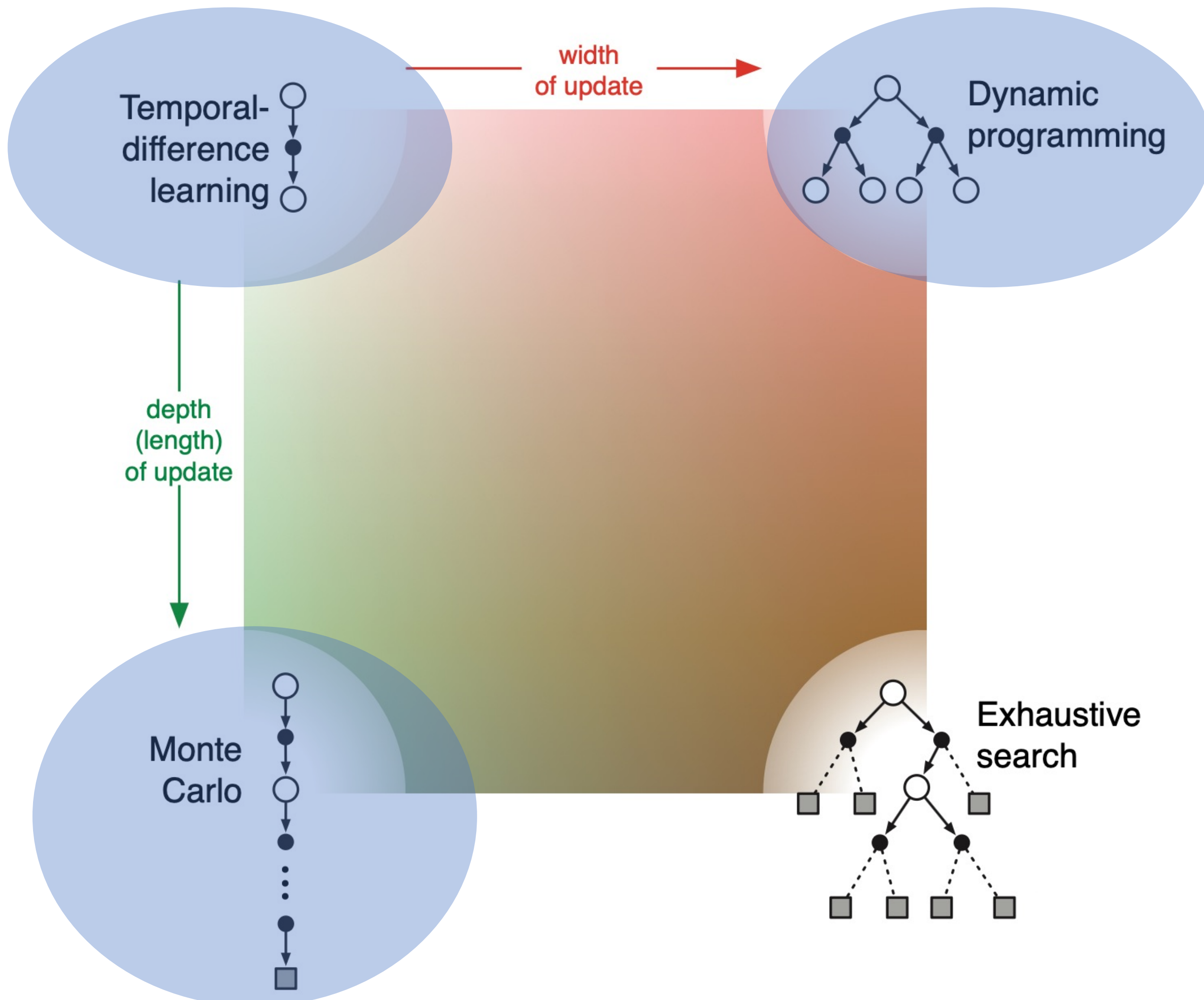
Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning: Off-policy TD Control

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

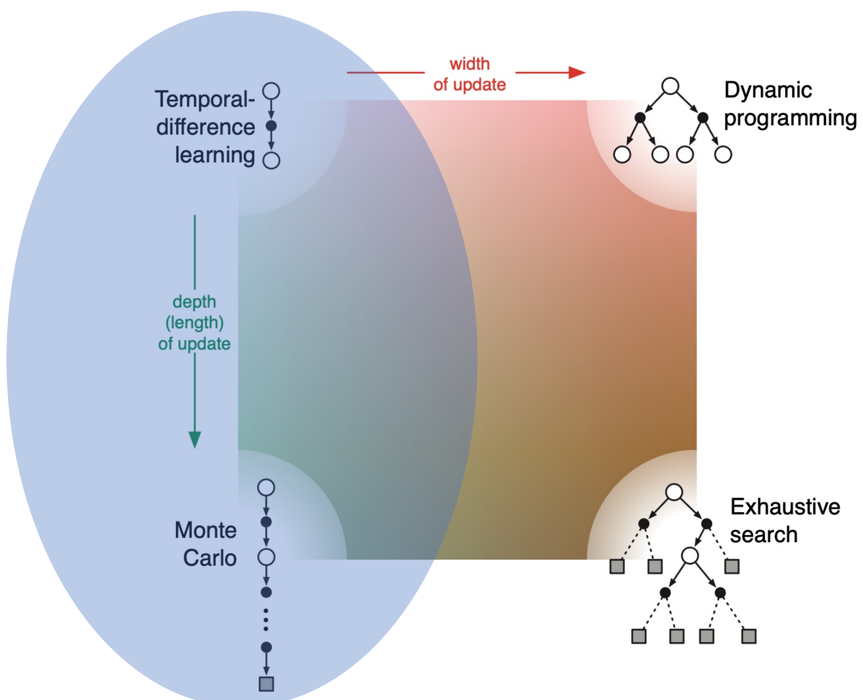
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\textit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal


width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search

N-step Bootstrapping

- Unifying MC and TD methods



Previously: MC and TD

- Monte Carlo update:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

- 1-step TD update:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

- n-step TD update:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

N-step TD prediction

- The space of methods between Monte Carlo and TD. This gives us the following state-value learning algorithm:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\big[G_{t:t+n} - V_{t+n-1}(S_t)\big], \qquad 0 \leq t < T,$$

- while the values of all other states remain unchanged:

$$V_{t+n}(s) = V_{t+n-1}(s), \text{ for all } s \neq S_t$$

The control problem: n-step Sarsa

- Let's construct an on-policy TD control method.
- Previously: Sarsa -> one-step Sarsa or Sarsa(0)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

- We redefine n-step returns in terms of estimated action-values:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T - n$$

New update:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right]$$

N-step tradeoffs

- More accurate, but fewer and slower updates.

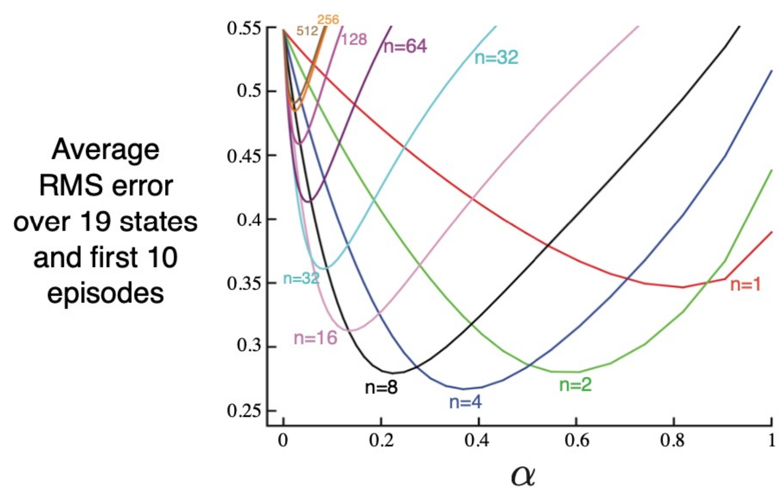
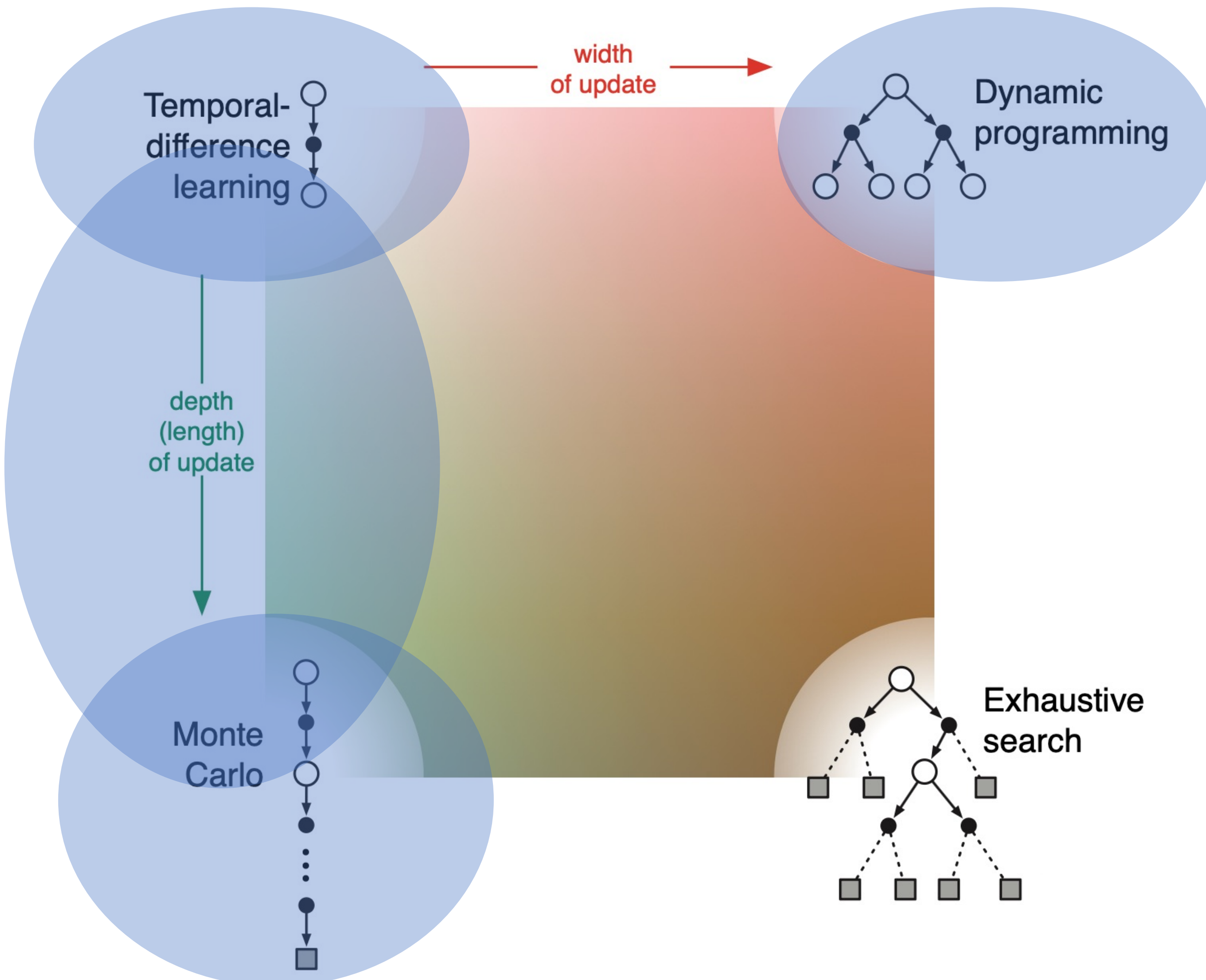
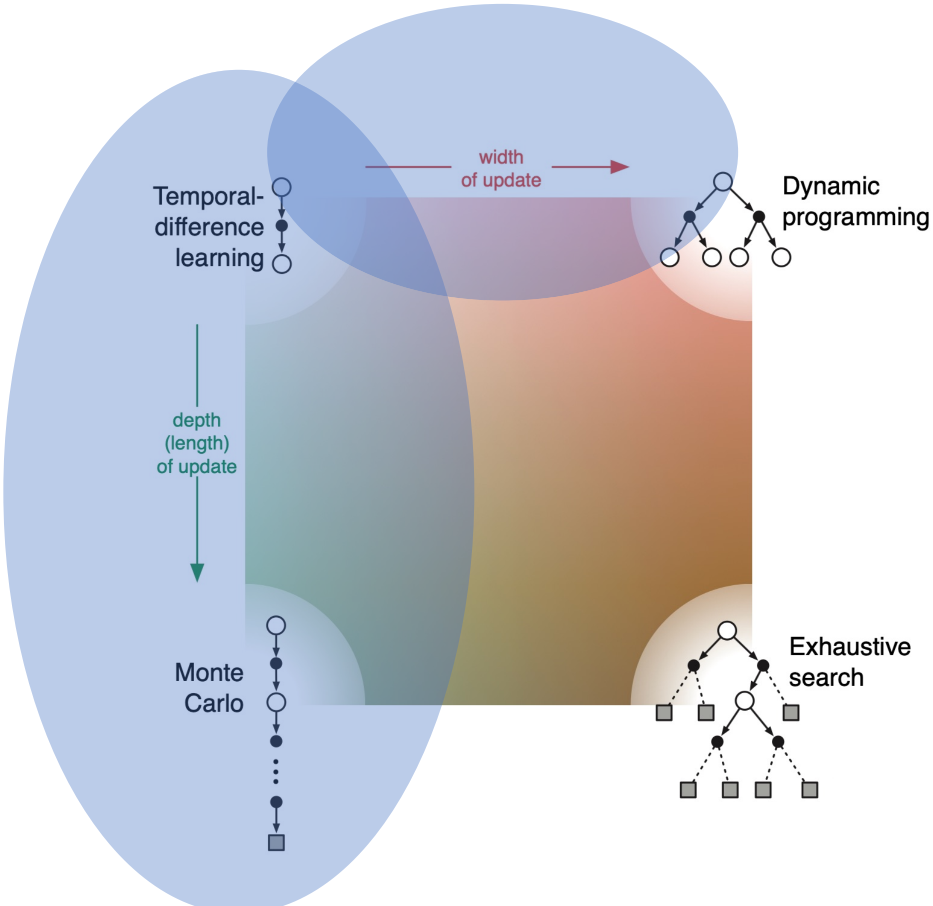


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■


width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search

Bridging Methods

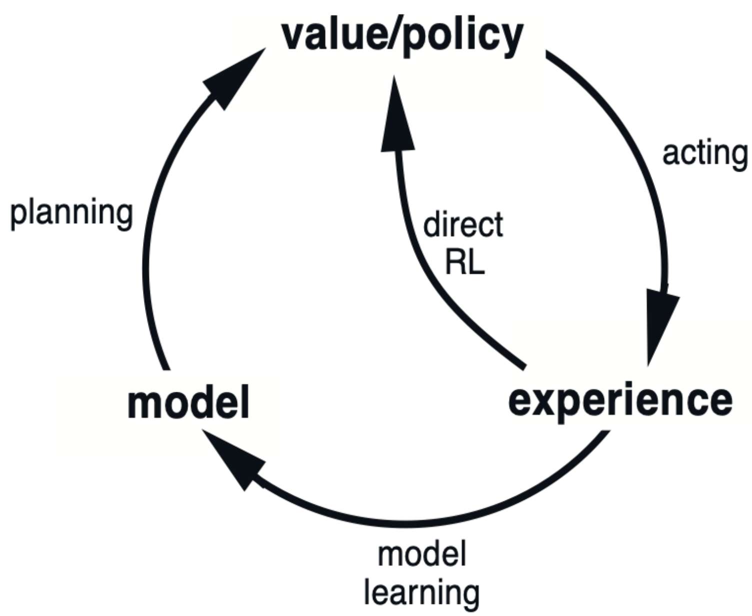
- n-step methods bridge TD and MC
  - TD(0) –→ MC
  - All online (model-free)
- Now we talk about bridging to DP (model-based)
  - TD,MC –→ DP (e.g. VI)
  - Also called learning vs. planning
  - Model-based RL does both
  - computational efficiency vs. sample efficiency



Two Types of Planning

- Model-based learning
  - e.g. Dyna
- Lookahead search
  - e.g. Monte Carlo Tree Search (MCTS)

Dyna: Integrated Planning, Acting, and Learning



Tabular Dyna-Q

Random-sample one-step tabular Q-planning

Loop forever:
1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S,Q)
(c) Take 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) Loop repeat n times:
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in S
    $R, S' \leftarrow Model(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

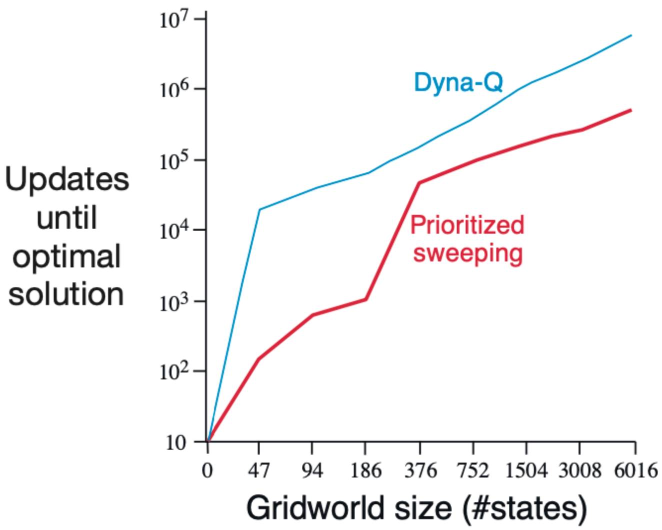
Dyna

- Downsides: uniform sampling is inefficient
- Planning can be much more efficient if simulated transitions and updates are focused on particular state–action pairs.
- Search might be usefully focused by working backward from goal states.

Prioritized Sweeping

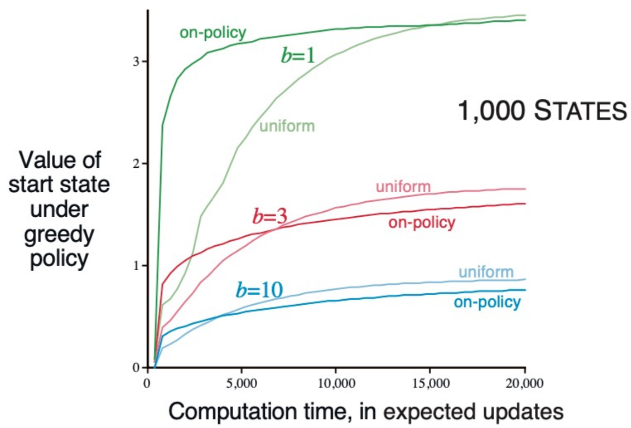
Prioritized sweeping for a deterministic environment

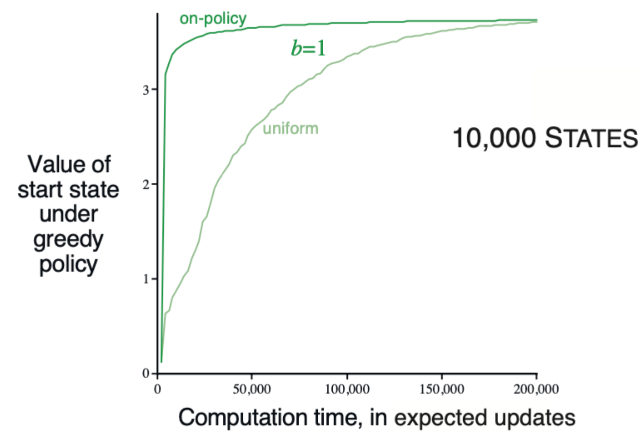
Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S, Q)$
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Model(S, A) \leftarrow R, S'$
(e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$.
(f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
(g) Loop repeat n times, while $PQueue$ is not empty:
$S, A \leftarrow first(PQueue)$
$R, S' \leftarrow Model(S, A)$
$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
Loop for all $\bar{S}, \bar{A}$ predicted to lead to S:
$\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
$P \leftarrow |\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})|$.
if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P



Trajectory Sampling

- Two ways of distributing updates:
  - Exhaustive sweeps over entire state or state-action space (e.g. dynamic programming)
  - Sampling from a distribution
    - Uniformly (Dyna-Q)
    - On-policy distribution (Trajectory sampling)

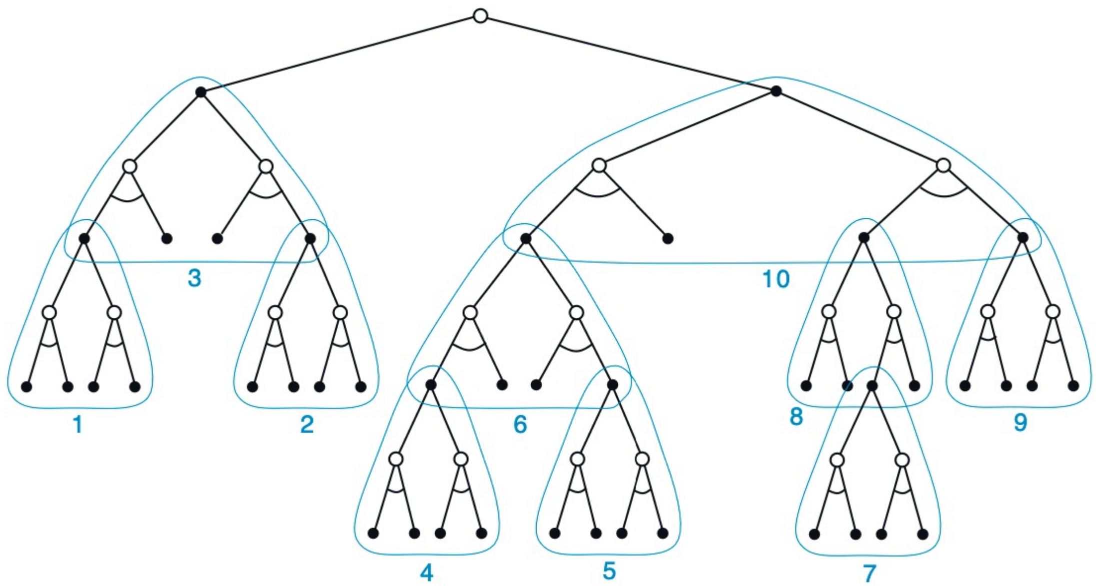




Planning at Decision Time

- Previous methods all use planning at training time.
- What about decision time planning?

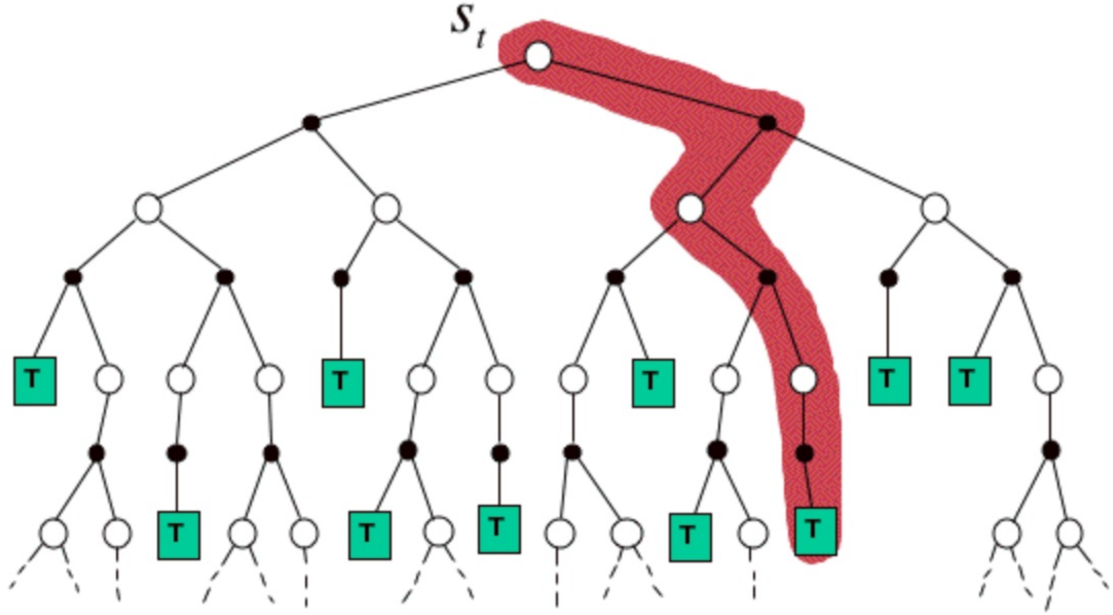
Heuristic Search



Sequence of one-step updates in a specific order (selective depth-first search).

Rollout Algorithms

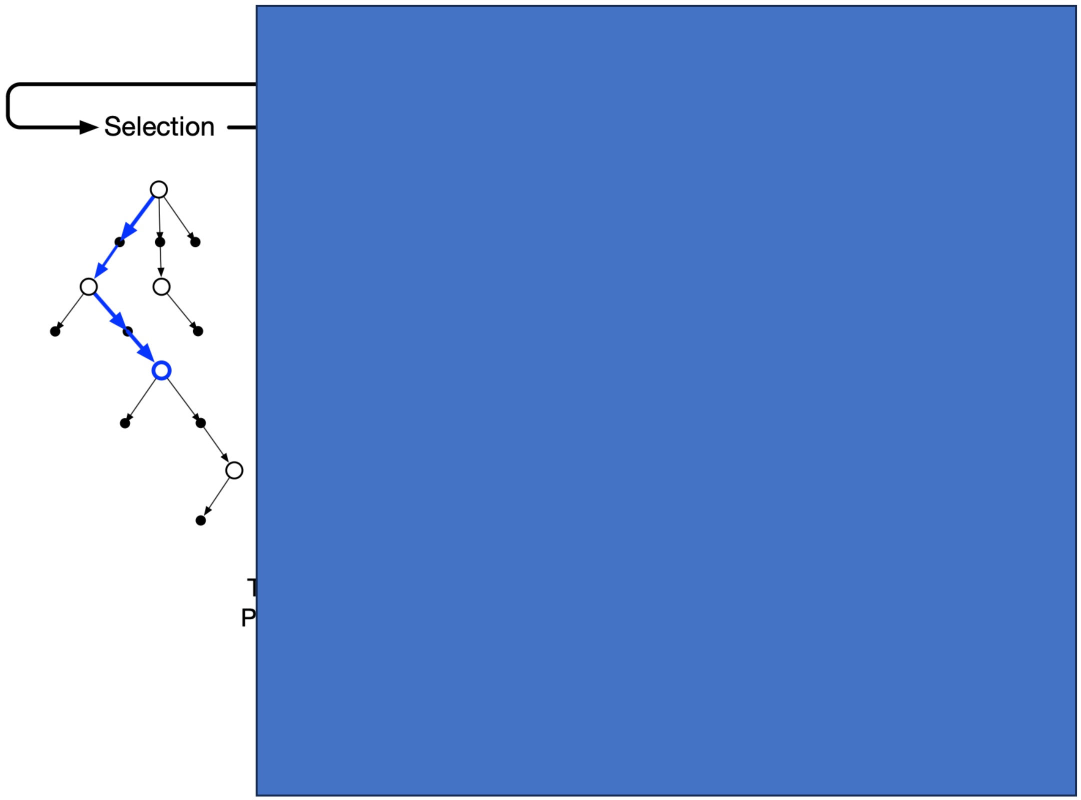
- Produce Monte Carlo estimates of action values only for each current state and for a given policy usually called the rollout policy.



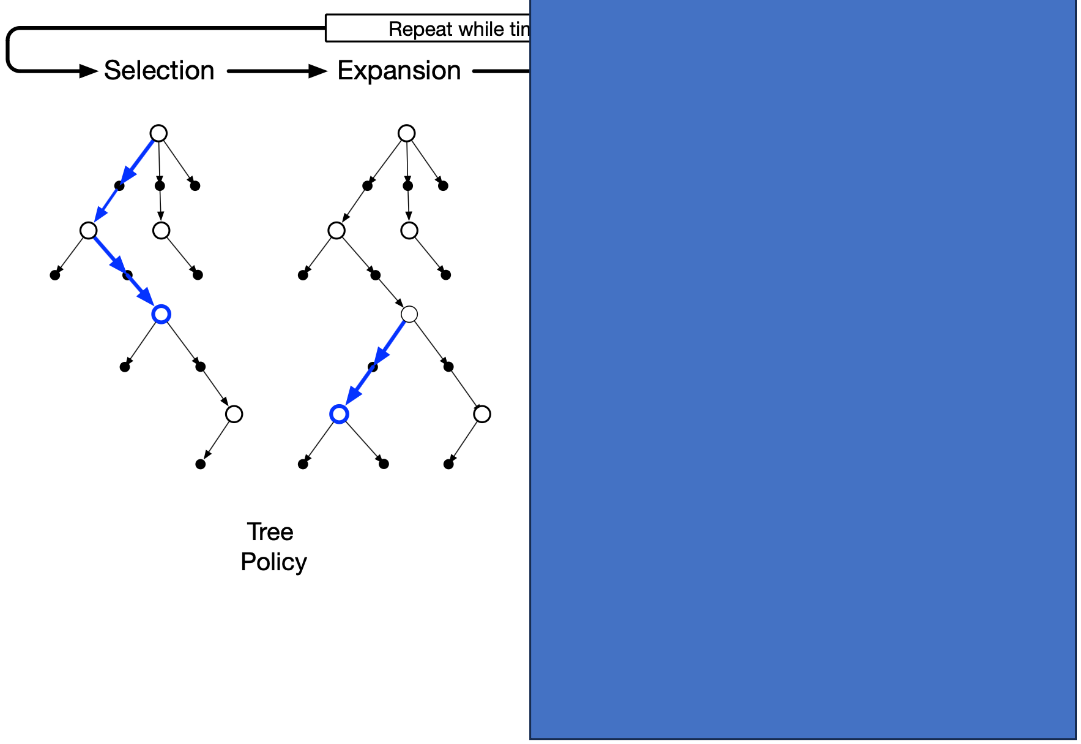
Monte Carlo Tree Search

- MCTS is a rollout algorithm
  - enhanced by the addition of a means for accumulating value estimates obtained from the Monte Carlo simulations in order to successively direct simulations toward more highly-rewarding trajectories.
- Largely responsible for the improvement in computer Go from a weak amateur level in 2005 to a grandmaster level (6 dan or more) in 2015

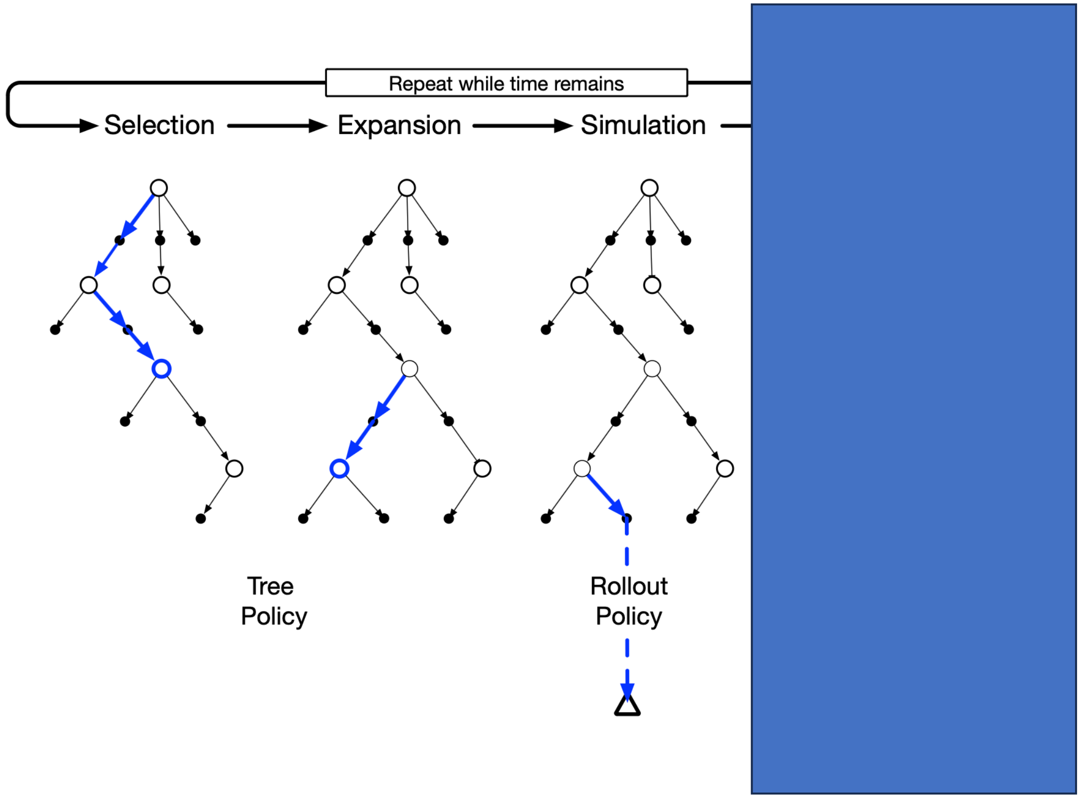
Monte Carlo Tree Search



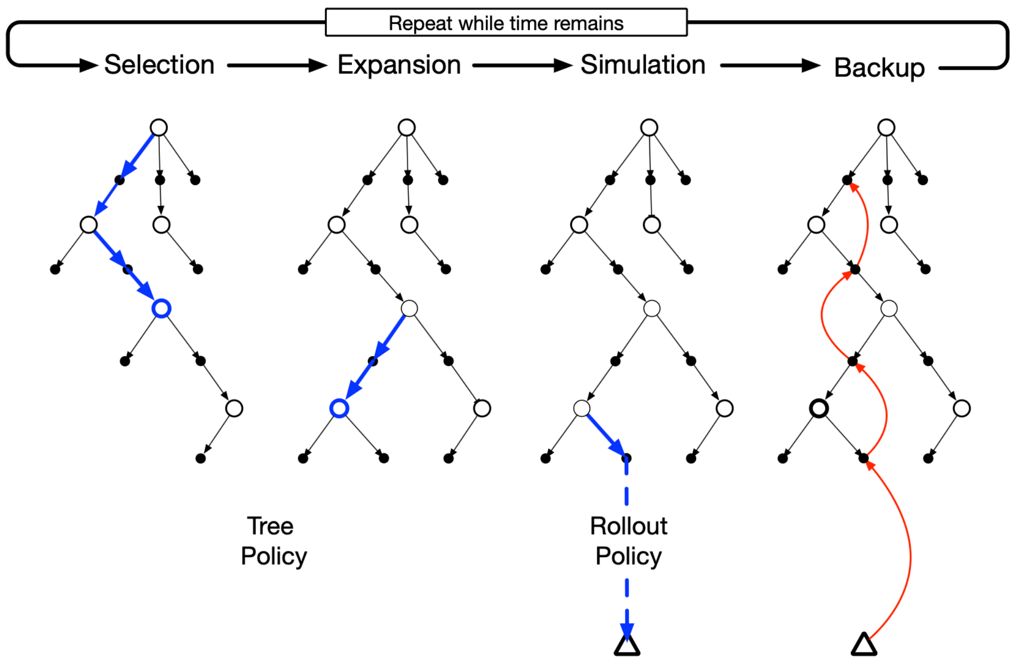
Monte Carlo Tree Search



Monte Carlo Tree Search



Monte Carlo Tree Search



What about function approximation?

The deadly triad

Divergence is possible when all 3 parts of the deadly triad are present:

- Function approximation
- Bootstrapping
- Off-Policy training

The deadly triad

Divergence is possible when all 3 parts of the deadly triad are present:

- Function approximation
- Bootstrapping
- Off-Policy training

The deadly triad

Divergence is possible when all 3 parts of the deadly triad are present:

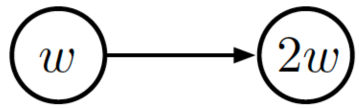
- Function approximation
- Bootstrapping
- Off-Policy training

The deadly triad

Divergence is possible when all 3 parts of the deadly triad are present:

- Function approximation
- Bootstrapping
- Off-Policy training

Off-policy semigradient methods



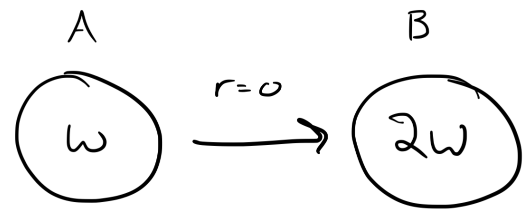
Stability of semigradient methods depends on on-policy distribution of updates. Why?

Imagine only updating one state S over and over again (i.e. off-policy):

- In tabular case, updating one state's value leaves all others changed
- With function approx + MC, multiple state values are updated, but V(S) is estimated independently of them via rewards only
- With function approx + TD (semigradient), multiple values are updated, which are then used to help estimate V(S) via bootstrapping, which are then updated again, which are then used to help estimates V(S)…

On-policy distribution forces state values to be "grounded" to something real

Examples of Off-policy Divergence

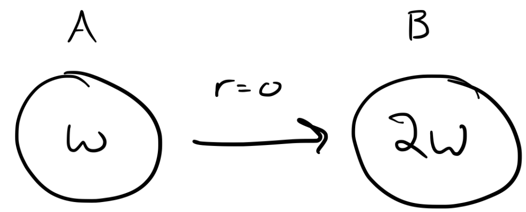


Init: $w_0 = 10$ Thus: $\hat{v}(A) = 10$, $\hat{v}(B) = 20$

Assume $\alpha = 0.5$, $\gamma = 0.9$

observe Transition from A to B

$$w_{t+1} = w_t + \alpha \rho \left[R_t + \gamma \hat{v}(B) - \hat{v}(A) \right] \nabla \hat{v}(A)$$



Init: $w_0 = 10$ Thus: $\hat{v}(A) = 10$, $\hat{v}(B) = 20$

Assume $\alpha = 0.5$, $\gamma = 0.9$

observe Transition from A to B

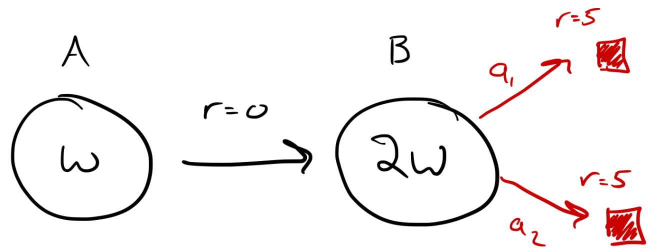
$$w_{t+1} = w_t + \alpha \rho \left[R_t + \gamma \hat{v}(B) - \hat{v}(A) \right] \nabla \hat{v}(A)$$

$$= 10 + (0.5)(1)\left[0 + .9(20) - 10 \right] \cdot 1$$

$$= 10 + 4$$

$$= 14$$

Thus: $\hat{v}(A) = 14$, $\hat{v}(B) = 28$



- off policy ignores transition from B and diverges!

- on-policy uses transitions from B, which lowers $\hat{v}(B)$ and $\hat{v}(A)$ and converges

Init: $w_0 = 10$ Thus: $\hat{v}(A) = 10$, $\hat{v}(B) = 20$

Assume $\alpha = 0.5$, $\gamma = 0.9$

observe Transition from A to B

$$w_{t+1} = w_t + \alpha \rho \left[R_t + \gamma \hat{v}(B) - \hat{v}(A) \right] \nabla \hat{v}(A)$$

$$= 10 + (0.5)(1)\left[0 + .9(20) - 10 \right] \cdot 1$$

$$= 10 + 4$$

$$= 14$$

Thus: $\hat{v}(A) = 14$, $\hat{v}(B) = 28$

Baird's Counterexample

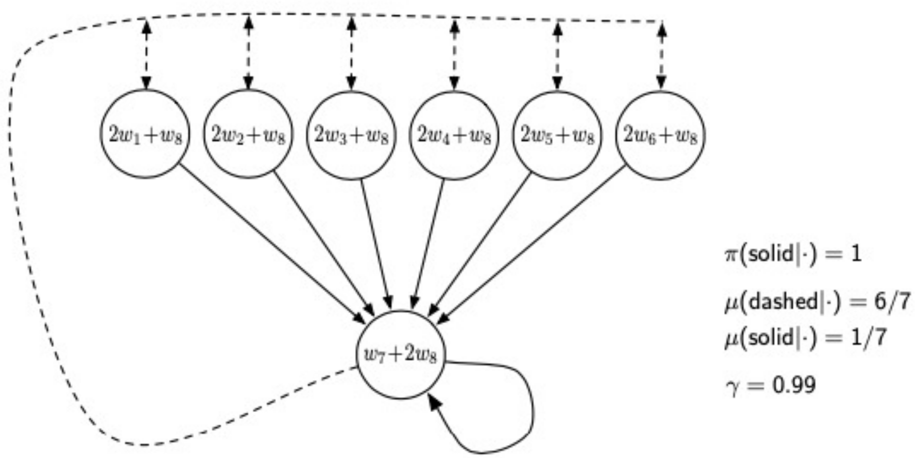
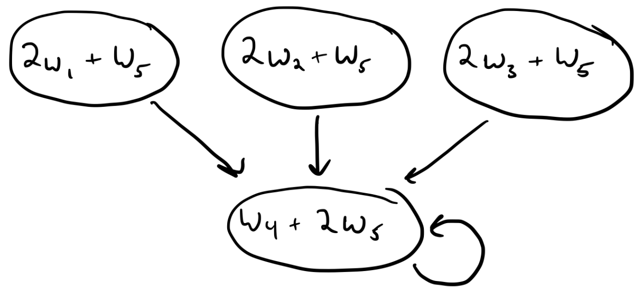


Figure 11.1: Baird's counterexample. The approximate state-value function for this Markov process is of the form shown by the linear expressions inside each state. The **solid** action usually results in the seventh state, and the **dashed** action usually results in one of the other six states, each with equal probability. The reward is always zero.

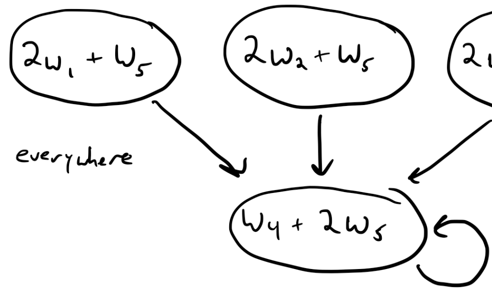


$\Rightarrow$

$$X: \begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

if $w_0 = \begin{bmatrix} 1 & 1 & 1 & 10 & 1 \end{bmatrix}^T$ then,

$$\hat{V} = \begin{bmatrix} 3 & 3 & 3 & 12 \end{bmatrix}^T$$



$r = 0$ everywhere

$$\Rightarrow \quad x: \begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

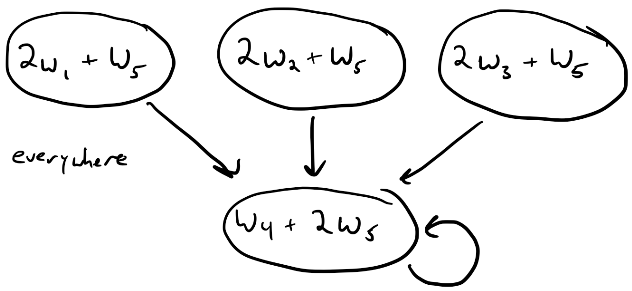
if $w_0 = \begin{bmatrix} 1 & 1 & 1 & 10 & 1 \end{bmatrix}^T$ then,

$$\hat{V} = \begin{bmatrix} 3 & 3 & 3 & 12 \end{bmatrix}^T$$

DP update:

$$w_{t+1} = w_t + \frac{\alpha}{|S|} \sum_s \Big(E_\pi[R_{t+1} + \gamma \hat{v}(s_{t+1})] - \hat{v}(s) \Big) \nabla v(s)$$

Step size alpha=1
Discount factor gamma = 1



$r = 0$ everywhere

$$\Rightarrow \quad x: \begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

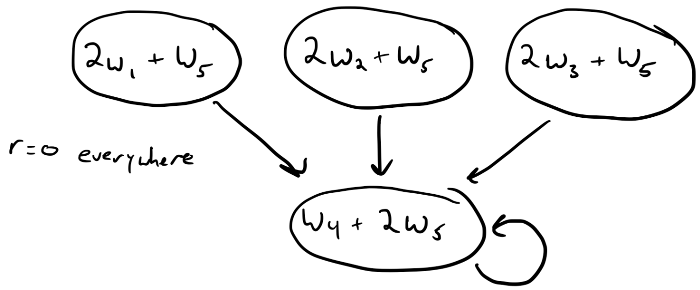
if $w_0 = [1 \ \ 1 \ \ 1 \ \ 10 \ \ 1]^T$ then,

$$\hat{V} = [3 \ \ 3 \ \ 3 \ \ 12]^T$$

DP update:

$$w_{t+1} = w_t + \frac{\alpha}{|S|} \sum_s \left(E_\pi[R_{t+1} + \gamma \hat{v}(s_{t+1})] - \hat{v}(s) \right) \nabla v(s)$$

$$= w_t + 1/4 \Big[(12-3)[2 \ 0 \ 0 \ 0 \ 1]^T + (12-3)[0 \ 2 \ 0 \ 0 \ 1]^T + (12-3)[0 \ 0 \ 2 \ 0 \ 1]^T + (12-12)[0 \ 0 \ 0 \ 1 \ 2]^T \Big]$$



$\Longrightarrow$

$$X: \begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

if $w_0 = [1 \ 1 \ 1 \ 10 \ 1]^T$ then,

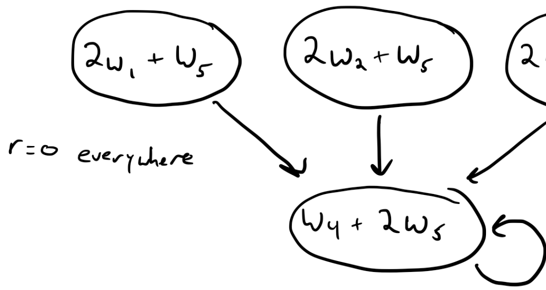
$$\hat{V} = [3 \ 3 \ 3 \ 12]^T$$

DP update:

$$w_{t+1} = w_t + \frac{\alpha}{|S|} \sum_s \left(E_\pi[R_{t+1} + \gamma \hat{v}(S_{t+1})] - \hat{v}(s) \right) \nabla v(s)$$

$$= w_t + 1/4 \Big[(12-3)[2 \ 0 \ 0 \ 0 \ 1]^T + (12-3)[0 \ 2 \ 0 \ 0 \ 1]^T + (12-3)[0 \ 0 \ 2 \ 0 \ 1]^T + (12-12)[0 \ 0 \ 0 \ 1 \ 2]^T \Big]$$

$$= [1 \ 1 \ 1 \ 10 \ 1]^T + 1/4 \cdot [18 \ 18 \ 18 \ 0 \ 54]^T$$



$\Rightarrow$ x: $\begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$

if $w_0 = [1 \ 1 \ 1 \ 10 \ 1]^T$ then,

$\hat{V} = [3 \ 3 \ 3 \ 12]^T$

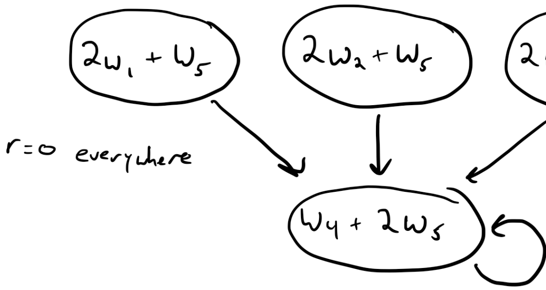
DP update:

$$w_{t+1} = w_t + \frac{\alpha}{|S|} \sum_s \left(E_\pi[R_{t+1} + \gamma \hat{v}(s_{t+1})] - \hat{v}(s) \right) \nabla v(s)$$

$$= w_t + 1/4 \Big[(12-3)[2 \ 0 \ 0 \ 0 \ 1]^T +$$
$$(12-3)[0 \ 2 \ 0 \ 0 \ 1]^T +$$
$$(12-3)[0 \ 0 \ 2 \ 0 \ 1]^T +$$
$$(12-12)[0 \ 0 \ 0 \ 1 \ 2]^T \Big]$$

$$= [1 \ 1 \ 1 \ 10 \ 1]^T + 1/4 \cdot [18 \ 18 \ 18 \ 0 \ 54]^T$$

$$= [5.5 \ 5.5 \ 5.5 \ 10 \ 15]^T$$



$\Longrightarrow$ $X: \begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$

if $w_0 = [1 \ 1 \ 1 \ 10 \ 1]^T$ then,

$\hat{V} = [3 \ 3 \ 3 \ 12]^T$

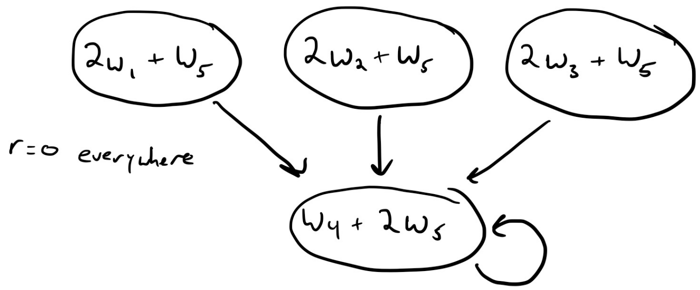
DP update:

$$w_{t+1} = w_t + \frac{\alpha}{|S|} \sum_s \left(E_\pi[R_{t+1} + \gamma \hat{v}(S_{t+1})] - \hat{v}(s) \right) \nabla v(s)$$

$$= w_t + 1/4 \Big[(12-3)[2 \ 0 \ 0 \ 0 \ 1]^T + (12-3)[0 \ 2 \ 0 \ 0 \ 1]^T + (12-3)[0 \ 0 \ 2 \ 0 \ 1]^T + (12-12)[0 \ 0 \ 0 \ 1 \ 2]^T \Big]$$

$$= [1 \ 1 \ 1 \ 10 \ 1]^T + 1/4 \cdot [18 \ 18 \ 18 \ 0 \ 54]^T$$

$$= [5.5 \ 5.5 \ 5.5 \ 10 \ 15]^T \Longrightarrow \hat{V} = [26 \ 26 \ 26 \ 40]^T$$



$$\Longrightarrow \quad x: \begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

if $w_0 = [1 \;\; 1 \;\; 1 \;\; 10 \;\; 1]^T$ then,

$$\hat{v} = [3 \;\; 3 \;\; 3 \;\; 12]^T$$

DP update:

$$w_{t+1} = w_t + \frac{\alpha}{|S|} \sum_s \Big(E_\pi\big[R_{t+1} + \gamma \hat{v}(S_{t+1})\big] - \hat{v}(s)\Big) \nabla v(s)$$

$$= w_t + \tfrac{1}{4}\Big[(12-3)[2\;0\;0\;0\;1]^T + (12-3)[0\;2\;0\;0\;1]^T + (12-3)[0\;0\;2\;0\;1]^T + (12-12)[0\;0\;0\;1\;2]^T \Big]$$

$$= [1\;\;1\;\;1\;\;10\;\;1]^T + \tfrac{1}{4}\cdot[18\;\;18\;\;18\;\;0\;\;54]^T$$

$$= [5.5\;\;5.5\;\;5.5\;\;10\;\;15]^T \quad \Longrightarrow \quad \hat{v} = [26\;\;26\;\;26\;\;40]^T$$

- DP updates with off-policy state dist (uniform), but policy π always follows solid lines
- Updates top states more often than it should
- Bottom state looks better and raises value of top state
- ... which raises value of bottom state, which ...

Outline: First Half

- What is Reinforcement Learning and when should I use it?
- Finite Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Planning
- Deadly Triad

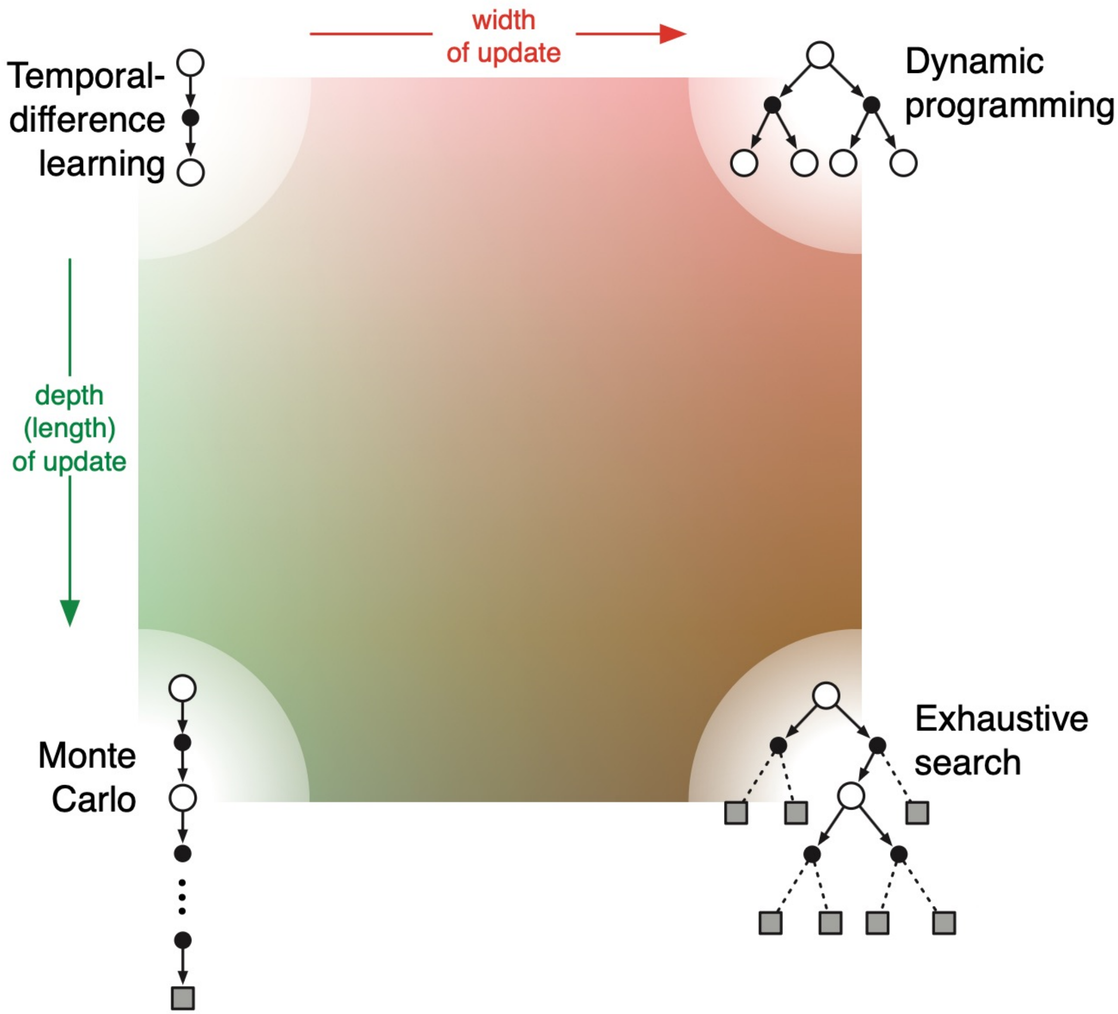
Coffee Break!

Outline: First Half

- What is Reinforcement Learning and when should I use it?
- Finite Markov Decision Processes
- Dynamic Programming
- Monte Carlo Methods
- Temporal-Difference Learning
- Planning

Outline: Second Half

- Function Approximation: Model-free Methods
  - DQN
  - REINFORCE and Policy gradient
  - Actor-Critic Methods
- Function Approximation: Model-based Methods
  - Dyna
  - MBPO
  - PETS
- Advanced Topics
  - Abstractions and Generalization
  - Leveraging Structure in RL
  - Self-supervised RL



width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search

A Coarse Breakdown of Model-free Methods

- Q-Learning $\boxed{Q^*(s,a)} = r(s,a) + \gamma \max_{a'} \boxed{Q^*}(s'(s,a),a')$
- Policy Gradient $J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[r(\tau)\right]$
- Actor-Critic

Algorithm

Iteratively table filling

Tabular Q-learning

$$Q^{(n)}(s,a) \leftarrow r(s,a) + \gamma \max_{a'} Q^{(n-1)}(s'(s,a),a')$$

Value Iteration

$$V^{(n)}(s) \leftarrow \max_{a} r(s,a) + \gamma V^{(n-1)}(s'(s,a))$$

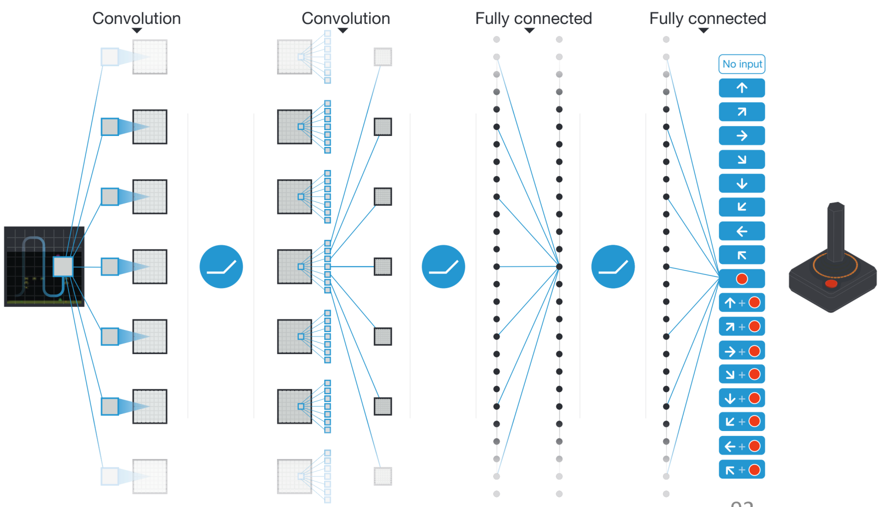
As long as we can enumerate ***all possible*** states and actions

Deep Q-Network (DQN)

Parametric function

Q-learning

$$Q_\theta(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$$



$Q_\theta(s, a)$ now have generalization capability

How could you take the gradient w.r.t θ ?
Note that θ appears on both sides.

[*Mnih et al.* Human-level control through deep reinforcement learning, *Nature 2015*]

DQN

Q-learning

$$Q_\theta(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a')$$

Old fixed parameters

Target network

Fixing RHS and learn θ from LHS.

[*Mnih et al.* Human-level control through deep reinforcement learning*, Nature 2015]*

DQN

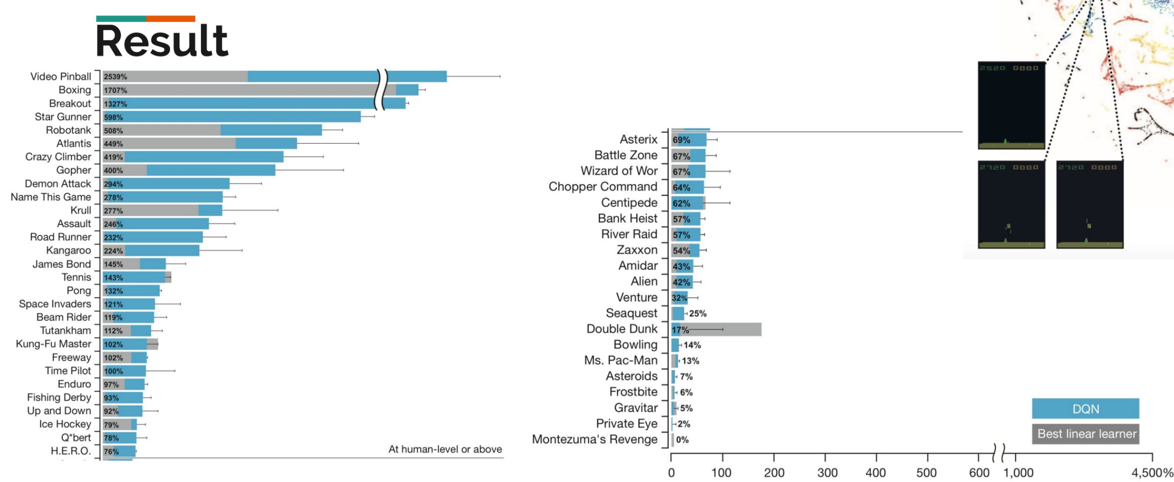
Q-learning (make the target even smoother)

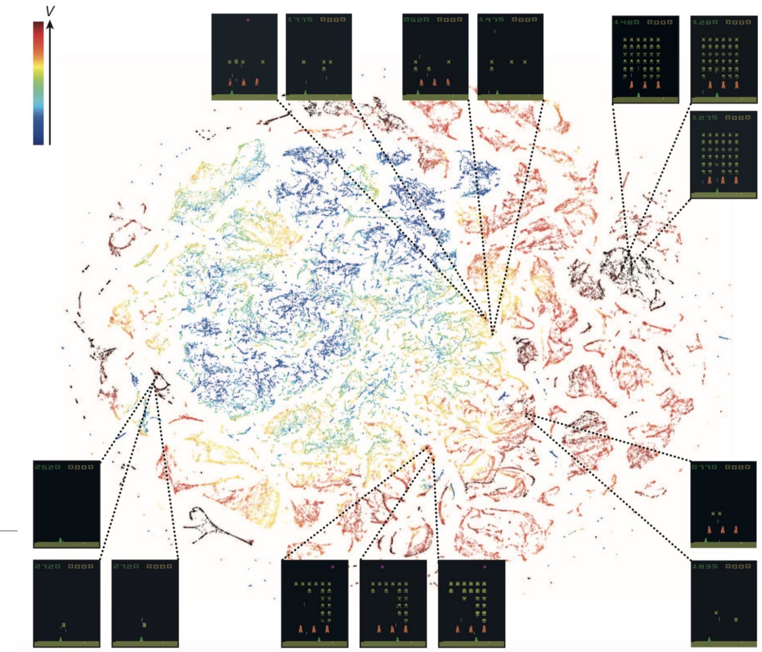
$$Q_\theta(s_t, a_t) \leftarrow (1-\alpha)Q_\theta(s_t, a_t) + \overset{\textit{Smoothing factor}}{\alpha} \left[r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a') \right]$$

$$\Delta Q_\theta(s_t, a_t) \propto \underbrace{r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a') - Q_\theta(s_t, a_t)}_{\textit{Temporal Difference (TD) Error } \delta}$$

[*Mnih et al.* Human-level control through deep reinforcement learning*, Nature 2015]*

DQN Results





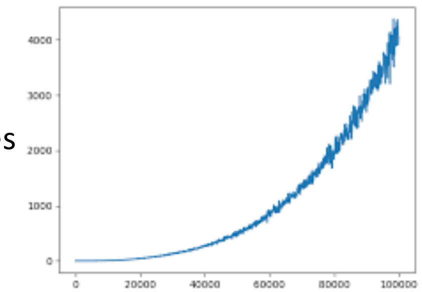
t-SNE embedding

100 * (DQN score - random play score)/ (human score - random play score)

[*Mnih et al.* Human-level control through deep reinforcement learning*, Nature 2015]*

DQN Issue

- Exploding Q values

Q values


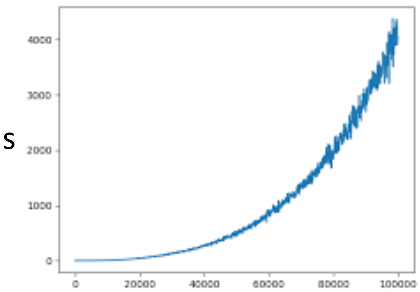
$$Q^*(s,a) = r(s,a) + \gamma \max_{a'} Q^*(s'(s,a), a')$$

[*Hado van Hasselt.* Double Q-learning. NeurIPS 2010]

DQN Issue

- Exploding Q values
- Fix: Double Q-Learning

Q values



Algorithm 1 Double Q-learning

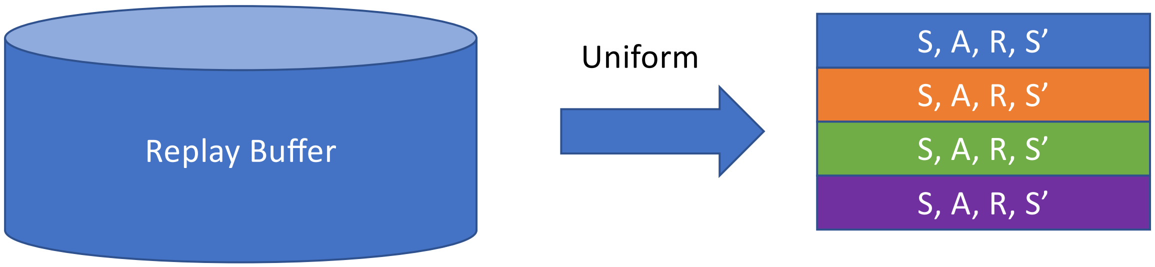
Initialize Q^A,Q^B,s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s', a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s', a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s', a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s', b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

- Use two Q networks instead of one to reduce bias.
  - One model get the optimal action
  - The other returns the Q value.

[*Hado van Hasselt*. Double Q-learning. NeurIPS 2010]

DQN Tricks

- Prioritized Experience Replay

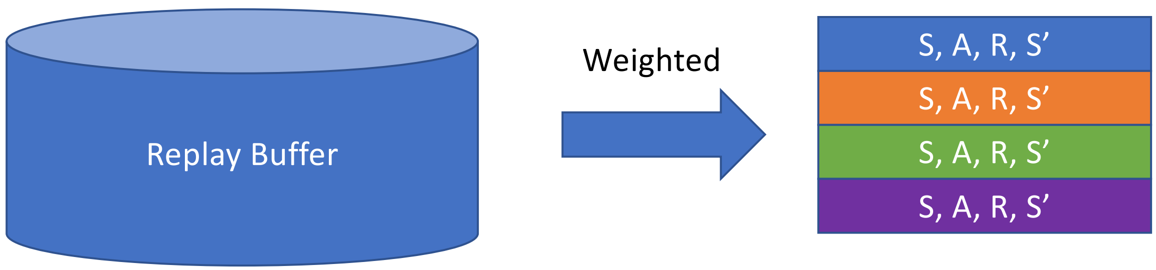


Standard replay

[*Schaul et al.* Prioritized Experience Replay. ICLR 2016]

DQN Tricks

- Prioritized Experience Replay



Prioritized replay

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Scoring mechanism: TD error

$$\delta_i = r_t + \gamma \max_{a \in \mathcal{A}} Q_{\theta^-}(s_{t+1}, a) - Q_\theta(s_t, a_t)$$

For vanilla DQN

$$\delta_i = r_t + \gamma Q_{\theta^-}(s_{t+1}, \mathbf{argmax}_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a)) - Q_\theta(s_t, a_t)$$

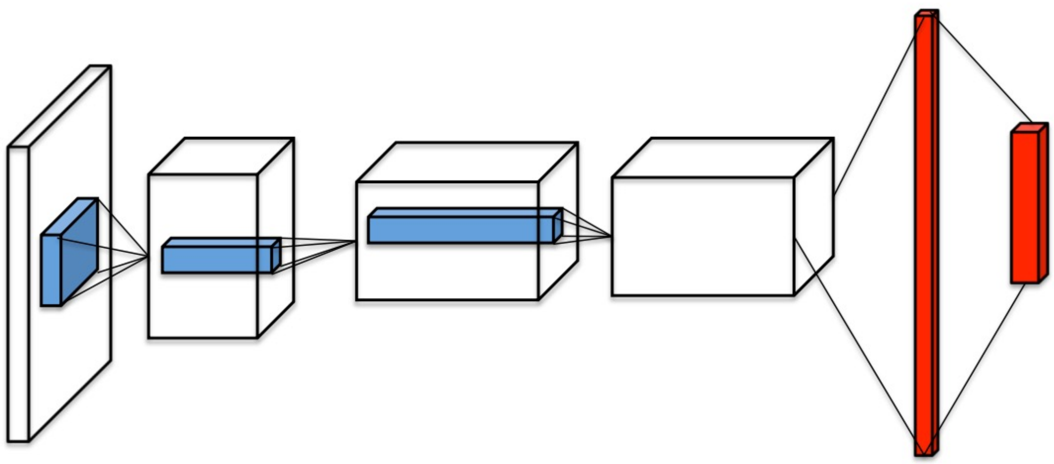
For Double DQN

[*Schaul et al.* Prioritized Experience Replay. ICLR 2016]

DQN Tricks

• Dueling networks

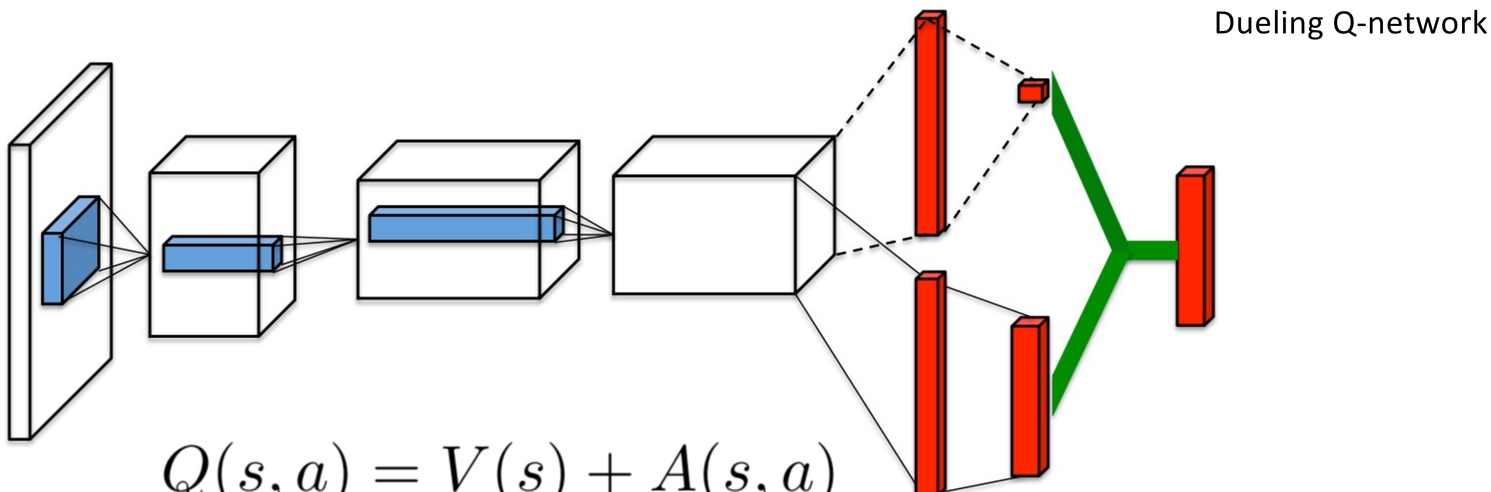
Standard Q-network



[*Wang et al.* Dueling Network Architectures for Deep Reinforcement Learning. ICML 2016]

DQN Tricks

- Dueling networks



$$Q(s,a) = V(s) + A(s,a)$$

Where $A(s,a) := Q(s,a) - V(s)$ is the advantage function.

[*Wang et al.* Dueling Network Architectures for Deep Reinforcement Learning. ICML 2016]

DQN Tricks

- Multi-step learning (also known as n-step returns)

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$$

- Multi-step variant of DQN loss:

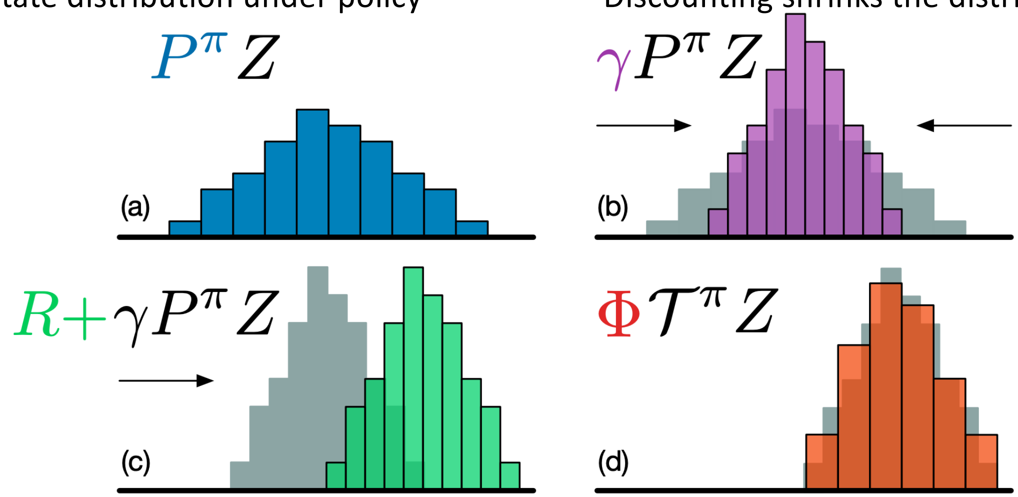
$$(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_{\overline{\theta}}(S_{t+n}, a') - q_\theta(S_t, A_t))^2$$

DQN Tricks

- Distributional reinforcement learning
  - Model the *value distribution* rather than the expected value.

Next state distribution under policy

Discounting shrinks the distribution towards 0



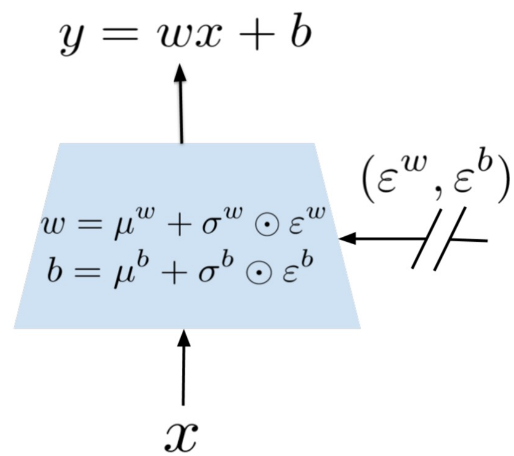
The reward shifts it

Projection step

[*Bellemare et al.* A distributional perspective on reinforcement learning. ICML 2017]

DQN Tricks

- Noisy Nets: Noisy linear layers for exploration



[*Fortunato et al.* Noisy networks for exploration. ICLR 2018]

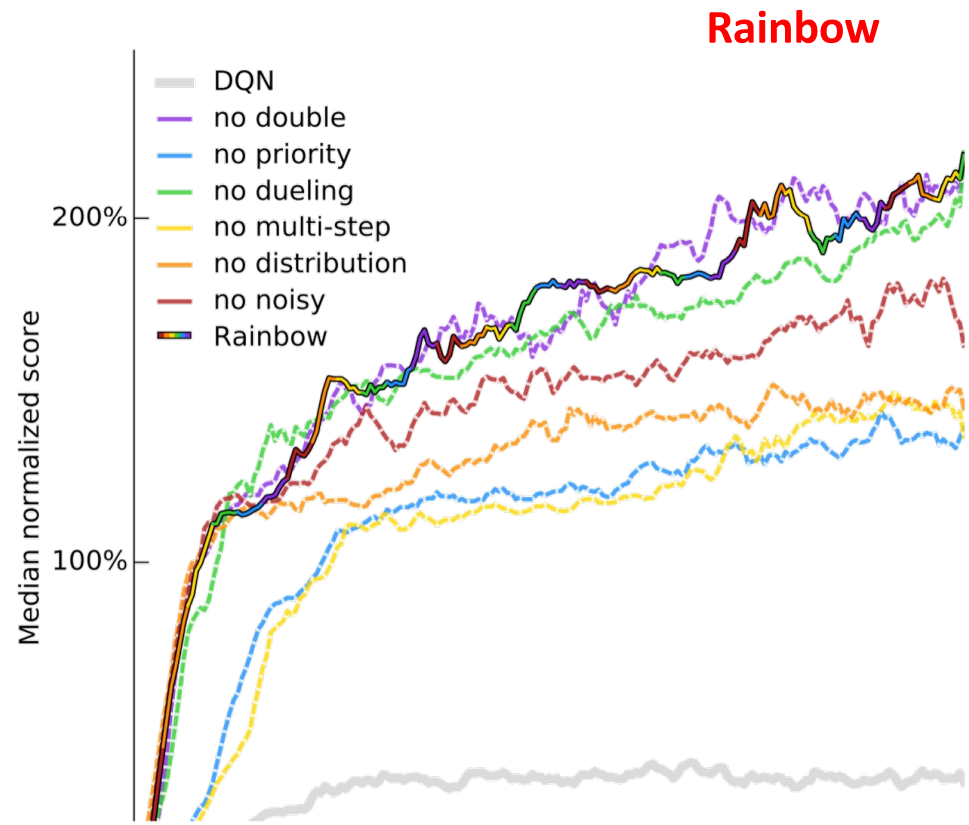
Rainbow DQN: Which tricks are most important?

Prioritized DQN

Distributional DQN

Multi-step learning

Noisy Nets



[Rainbow: Combining Improvements in Deep Reinforcement Learning, Hessel et al, 2017]

Downsides of Q-Learning

- Danger of instability and divergence caused by the Deadly Triad:
  1. Function approximation
  2. Bootstrapping
  3. Off-policy training

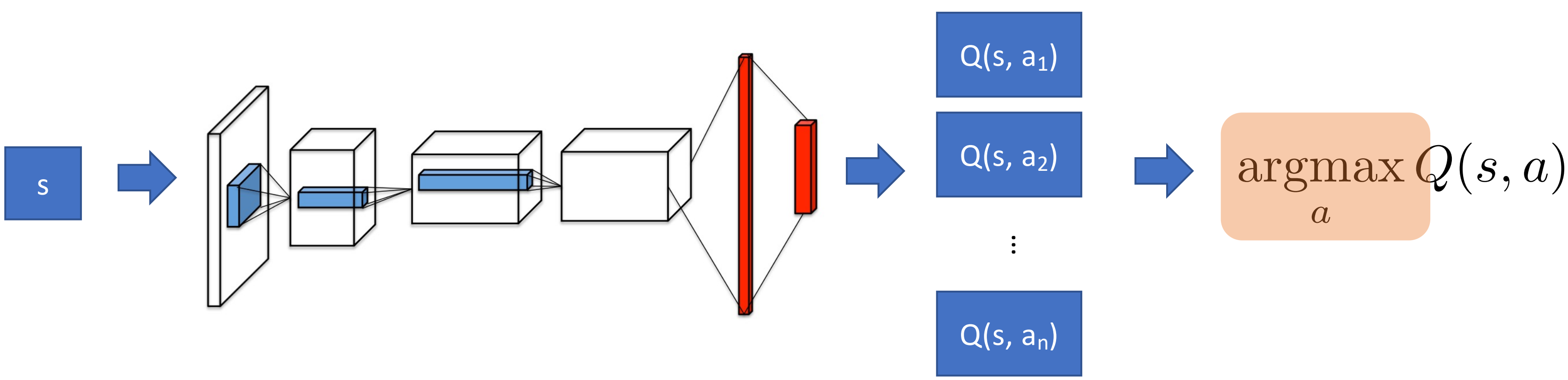
Hence, the use of tricks to get things working with deep neural networks!

Sutton & Barto. Reinforcement Learning: An Introduction.

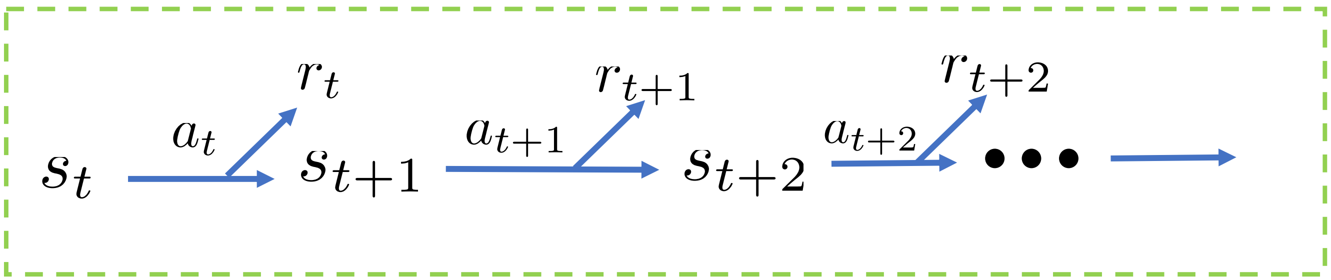
Downsides of Q-Learning

- Difficult to implement for continuous action spaces

How do we extract a policy from a Q function?



Policy Gradient



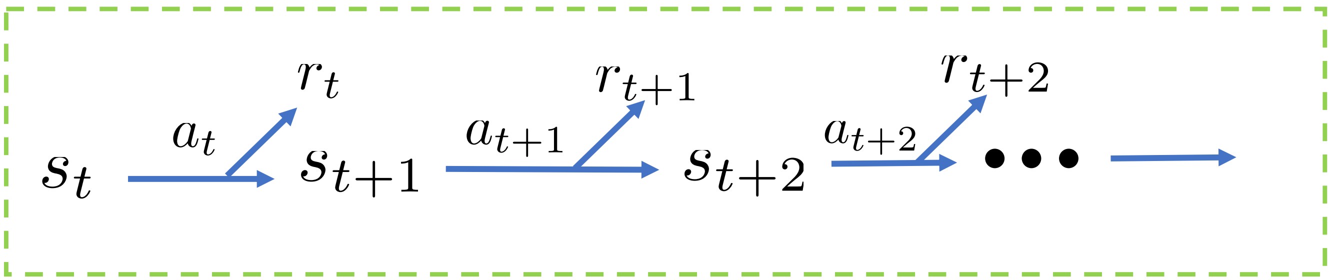
Trajectory $\boldsymbol{\tau}$

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \right]$$

$\pi_\theta(a|s)$ Probability of taking action a given state s

$r(\tau)$ Cumulative reward along a trajectory τ

Policy Gradient



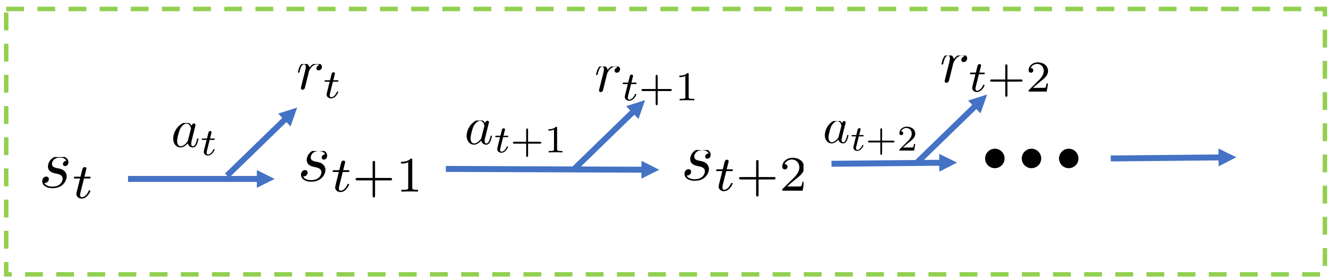
Trajectory τ

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[r(\tau)\right]$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau$$

$$= E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

Policy Gradient



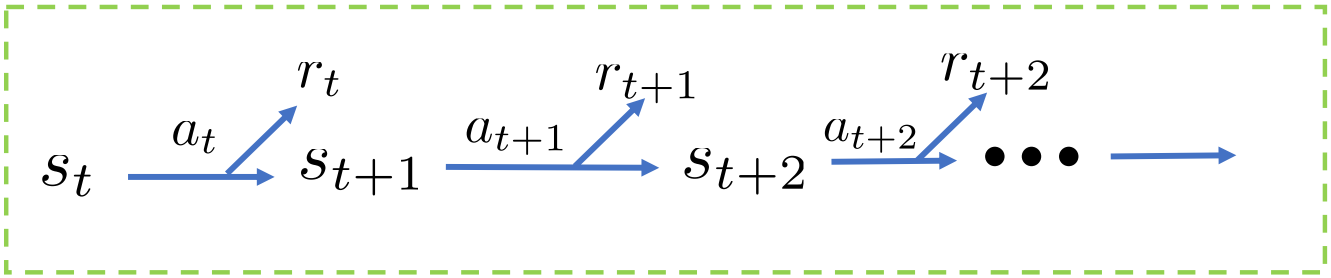
Trajectory $\boldsymbol{\tau}$

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \right]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \underline{\nabla_\theta \log p_\theta(\tau)} \right]$$

Score function

Policy Gradient



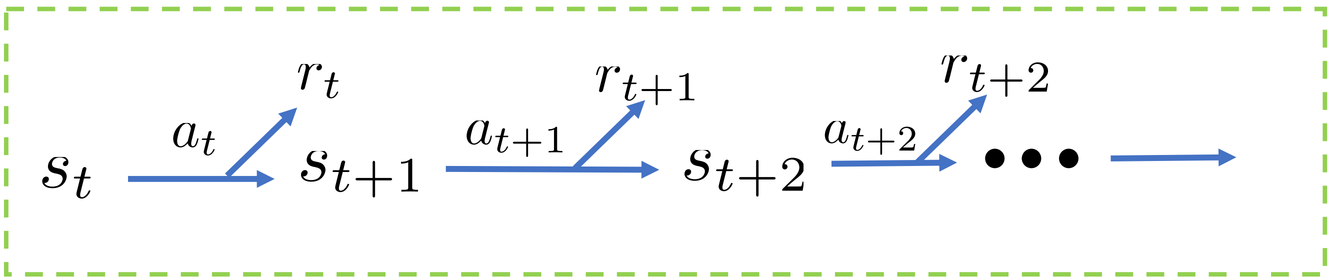
Trajectory $\boldsymbol{\tau}$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[r(\tau)\nabla_\theta \log p_\theta(\tau)\right]$$

$$\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1}|s_t, a_t)$$

Independent of θ

Policy Gradient



Trajectory $\boldsymbol{\tau}$

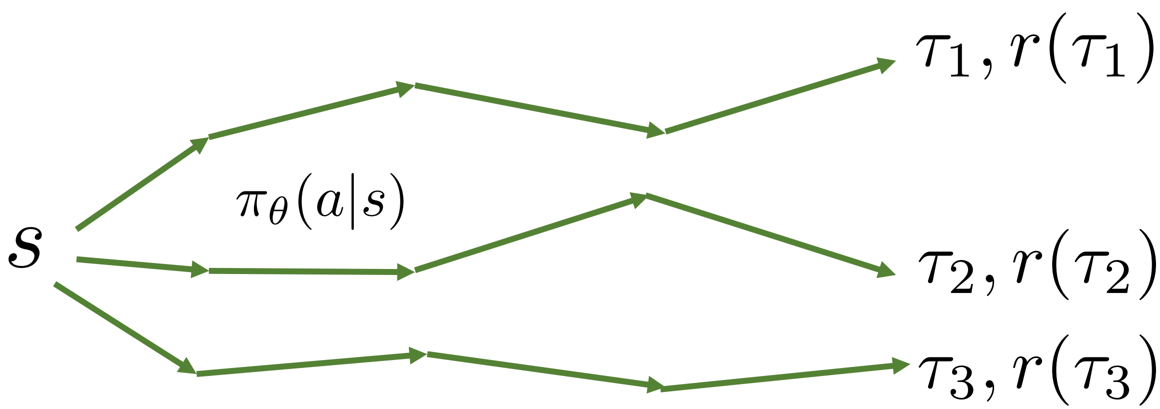
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) \right]$$

Estimated by sampling $\tau_i \sim \pi_\theta(a|s)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)$$

REINFORCE

$r(\tau)$ Actual reward obtained by rolling out from s



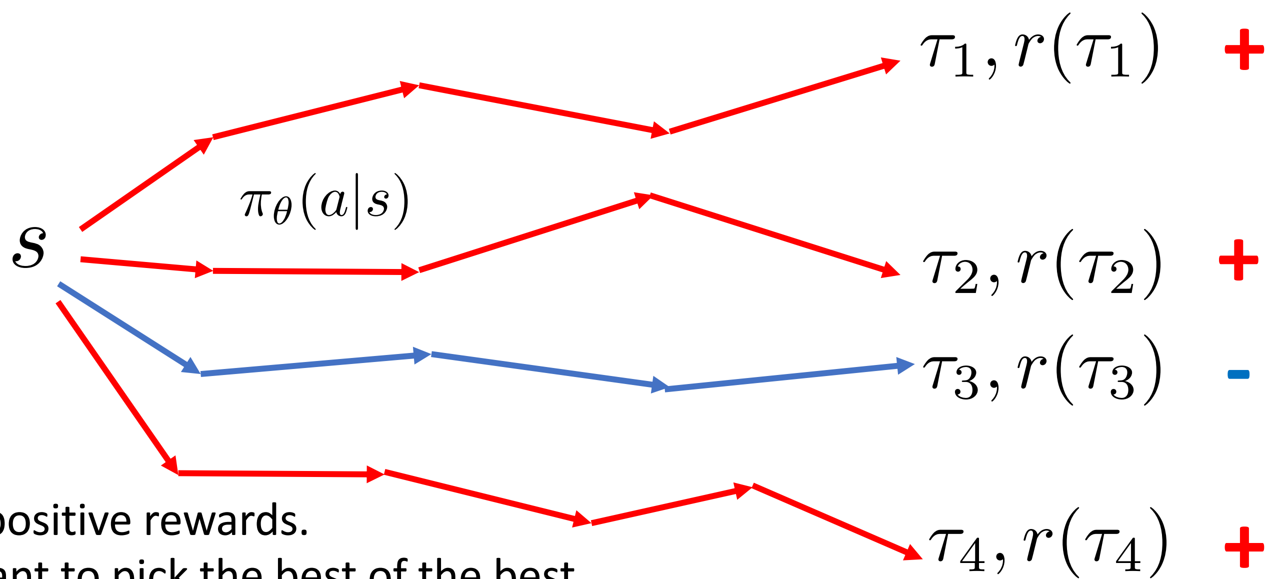
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[r_i(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right]$$

REINFORCE

1. Initialize the policy parameter θ at random.
2. Generate one trajectory on policy π_θ: $S_1, A_1, R_2, S_2, A_2, \dots, S_T$.
3. For t=1, 2, ... , T:
    1. Estimate the the return G_t;
    2. Update policy parameters: $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$

REINFORCE

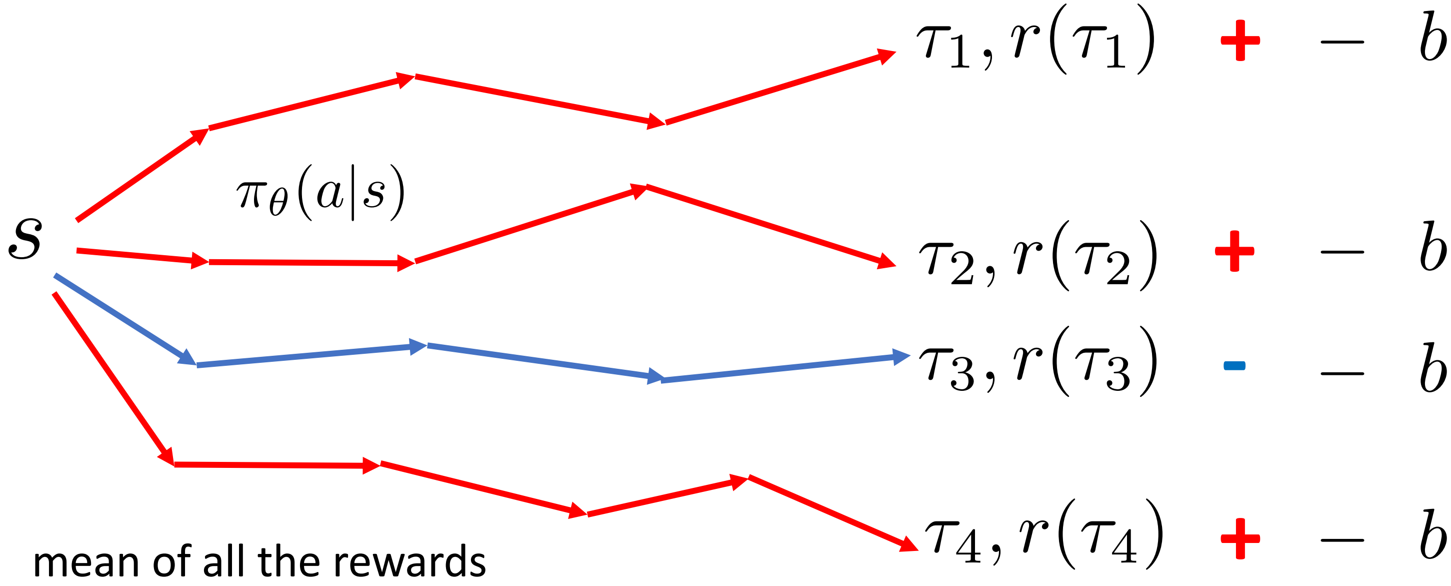
$r(\tau)$ Actual reward obtained by rolling out from s



Too many positive rewards.
We only want to pick the best of the best.

REINFORCE + baseline

$r(\tau)$ Actual reward obtained by rolling out from s



b mean of all the rewards

Can be any function that only depends on state.

Off-Policy Policy Gradient

Policy Gradient:

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{x_t, a_t \sim \pi}\left[\nabla_\theta \log \pi_\theta(a_t|x_t) Q^\pi(x_t, a_t)\right]$$

Off-policy Policy Gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x_t, a_t \sim \beta}\left[\rho_t \nabla_\theta \log \pi_\theta(a_t|x_t) Q^\pi(x_t, a_t)\right]$$

$$\rho_t = \frac{\pi(a_t|s_t)}{\beta(a_t|s_t)}$$ Importance sampling factor

Pros: We could now use off-policy data!
Cons: the factor might explode, when
we sample rare experience w.r.t. the behavior

Issues with Policy Gradient

- If data are on-policy, then it learns quite fast.

- Data need to be on-policy
  - Massive real-time simulations needed!
  - Low sample efficiency (you throw samples away immediately after using them)

- Reward estimation is not accurate
  - Random rollout
  - Tail value is not accurate.

Actor-Critic Models

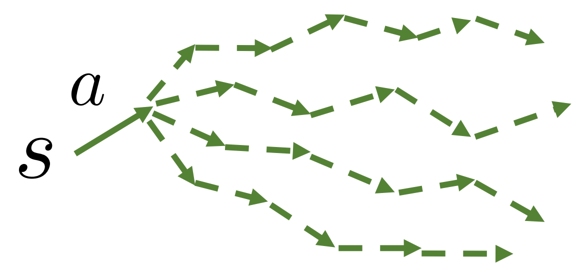
Actor-critic methods consist of two models, which may optionally share parameters:

- **Critic** updates the value function parameters w and depending on the algorithm it could be action-value $Q_w(a|s)$ or state-value $V_w(s)$.
- **Actor** updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic.

1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \ldots T$:
    1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
    2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
    3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
    4. Compute the correction (TD error) for action-value at time t:
       $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
       and use it to update the parameters of action-value function:
       $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
    5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

Actor-Critic Models

Rollout to the end or not?



$r(\tau) \approx Q^{\pi}_{\theta}(s,a)$ Rollout return as a parametric function (critic)

$b(s) = V_{\theta}(s)$ Use the value function as the baseline

$$r(\tau) - b(s) \approx Q^{\pi}_{\theta}(s,a) - V_{\theta}(s) = \boxed{A^{\pi}_{\theta}(s,a)}$$

Advantage function

“Advantageous Actor-Critic”

Policy Gradient in practice

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[r(\tau)\right] = \mathbb{E}_{s \sim \rho_{\pi_\theta}} \mathbb{E}_{a \sim \pi_\theta(\cdot|s)} Q^{\pi_\theta}(s,a)$$

$$= \mathbb{E}_{s \sim \rho_{\pi_\theta}} \sum_a \pi_\theta(a|s) Q^{\pi_\theta}(s,a)$$

$$\approx \mathbb{E}_{s \sim \boxed{\rho_{\pi_{\theta_0}}}} \sum_a \pi_\theta(a|s) Q^{\boxed{\pi_{\theta_0}}}(s,a)$$

Use old and fixed parameters

$$= \mathbb{E}_{s \sim \rho_{\pi_{\theta_0}}, a \sim \pi_{\theta_0}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_0}(a|s)} Q^{\pi_{\theta_0}}(s,a)\right]$$

Trust Region Policy Optimization (TRPO)

Advantage also works here

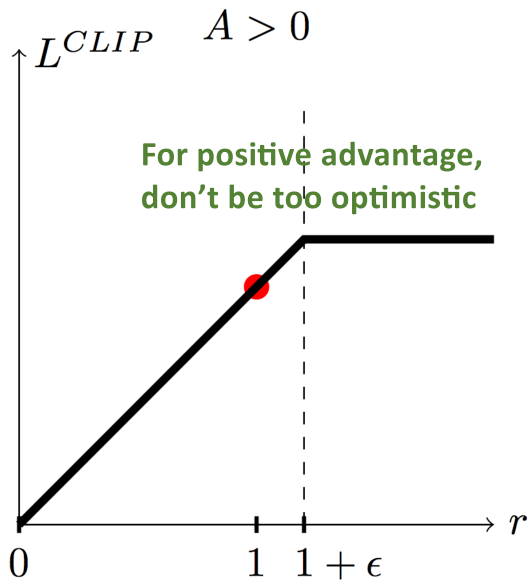
$$\max_{\theta} \; \mathbb{E}_{s\sim\rho_{\pi_{\theta_0}},a\sim\pi_{\theta_0}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_0}(a|s)}\boxed{Q^{\pi_{\theta_0}}(s,a)}\right]$$

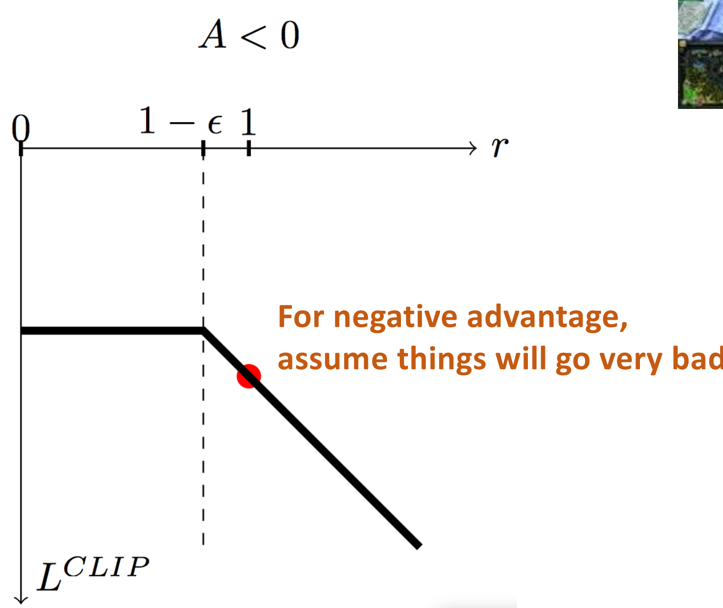
$$\text{s.t.} \qquad \mathbb{E}_{s\sim\rho_{\pi_0}}\underline{\left[D_{KL}(\pi_{\theta_0}(\cdot|s)\|\pi_\theta(\cdot|s))\right]\leq\delta}$$

Take baby steps, make sure the approximation is not off.

[Schulman et al, Trust Region Policy Optimization]

Proximal Policy Optimization (PPO)





Dota 2

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

[Schulman et al, Proximal Policy Optimization Algorithms]

Deterministic Policy Gradient (DPG)

Objective

$$J(\mu_\theta) = \mathbb{E}_{s \sim \rho^\mu} \left[Q^\mu(s, \mu_\theta(s)) \right]$$

$\mu_\theta(s)$ Deterministic policy function (e.g., if action is continuous)

$Q^\mu(s, a)$ Q-function following policy μ

[Deterministic Policy Gradient Algorithms, D. Silver et al.]

Deterministic Policy Gradient (DPG)

Taking Derivative w.r.t θ:

No need to take gradient w.r.t μ
Because of deterministic policy
gradient theorem

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \underbrace{\nabla_a Q^\mu(s,a)|_{a=\mu(s)}}_{\text{Scalar gradient at } (s,\, a)}\right]$$

Sample state from the distribution, $s_i \sim \rho^\mu$:

$$\nabla_\theta J(\mu_\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \mu_\theta(s_i) \nabla_a Q^\mu(s_i,a)|_{a=\mu(s_i)}$$

[Deterministic Policy Gradient Algorithms, D. Silver et al.]

DDPG (Deep Deterministic Policy Gradient)

- Use deep networks to represent policy / Q.
- Generate trajectories with current policy + noise
  - Since the policy is deterministic

- Save trajectories into replay buffer and sample from it (Off-policy!)
- Learn Q^{μ} via DQN using target network
- Learn μ using the slide above.

[Continuous Control With Deep Reinforcement Learning, Lillicrap et al. ICLR 2016]

Distributed Distributional Deterministic Policy Gradients (D4PG)

- Distributional version of DDPG

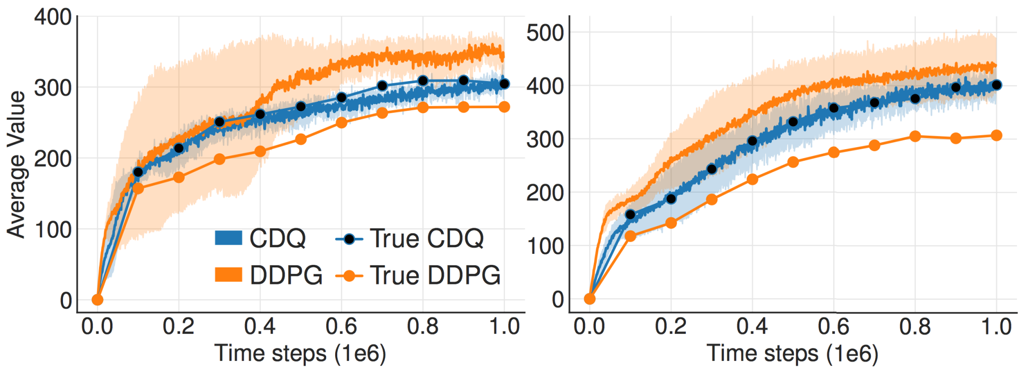
1. Distributional critic
2. N-step returns
3. Multiple distributed parallel actors
4. Prioritized experience replay

Twin Delayed DDPG (TD3)

Clipped Double Q-learning (CDQ)

Two independent models θ_1, ψ_1 and θ_2, ψ_2

$$y_1 = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \underbrace{\pi_{\phi_1}(s')}_{\textbf{Policy}})$$



(a) Hopper-v1 (b) Walker2d-v1

Delayed update of Target and Policy Networks

Target Policy Smoothing

[Addressing Function Approximation Error in Actor-Critic Methods, S Fujimoto et al, ICML 2018]

Soft Actor-Critic (SAC)

Objective:

Maximum entropy objective

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \underline{\alpha \mathcal{H}(\pi(\cdot | \mathbf{s}_t))} \right]$$

Improve policy diversity

Quantities to Learn simultaneously:

$$V_\psi(\mathbf{s}_t) \qquad Q_\theta(\mathbf{s}_t, \mathbf{a}_t) \qquad \pi_\phi(\mathbf{a}_t | \mathbf{s}_t)$$

[Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, T. Haarnoja et al, ICML 2018]

Learning Value function

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi} \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t | \mathbf{s}_t) \right] \right)^2 \right]$$

Match values with Q and policy

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(\mathbf{s}_t) \left(V_\psi(\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t) + \log \pi_\phi(\mathbf{a}_t | \mathbf{s}_t) \right)$$

[Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, T. Haarnoja et al, ICML 2018]

Learning Q-function

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \hat{Q}(\mathbf{s}_t,\mathbf{a}_t)\right)^2\right]$$

Target:

$$\hat{Q}(\mathbf{s}_t,\mathbf{a}_t) = r(\mathbf{s}_t,\mathbf{a}_t) + \gamma\,\mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V_{\bar{\psi}}(\mathbf{s}_{t+1})\right]$$

DQN-like step. Look one step ahead!

[Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, T. Haarnoja et al, ICML 2018]

Learning Policy without Policy Gradient

Matching policy with the current Q-value using KL-divergence

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathrm{D_{KL}} \left(\pi_\phi(\,\cdot\,|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(Q_\theta(\mathbf{s}_t,\,\cdot\,)\right)}{Z_\theta(\mathbf{s}_t)} \right) \right]$$

With deterministic action $a_t = f_\phi(\epsilon_t; s_t)$, things are simpler:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} \left[\alpha \log \pi_\phi(f_\phi(\epsilon_t; \mathbf{s}_t)|\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, f_\phi(\epsilon_t; \mathbf{s}_t)) \right]$$

Policy gradient is avoided so that it can work on off-policy data (replay buffer)

[Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, T. Haarnoja et al, ICML 2018]

SAC with Automatically Adjusted Temperature

SAC is brittle with respect to the temperature parameter. Unfortunately it is difficult to adjust temperature, because the entropy can vary unpredictably both across tasks and during training as the policy becomes better.

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_t}[-\alpha \log \pi_t(a_t \mid s_t) - \alpha \mathcal{H}_0]$$

[Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, T. Haarnoja et al, ICML 2018]

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters

$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights

$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool

for each iteration **do**

  for each environment step **do**

    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy

    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment

    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool

  end for

  for each gradient step **do**

    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters

    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ TD3 components ▷ Update policy weights

    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature

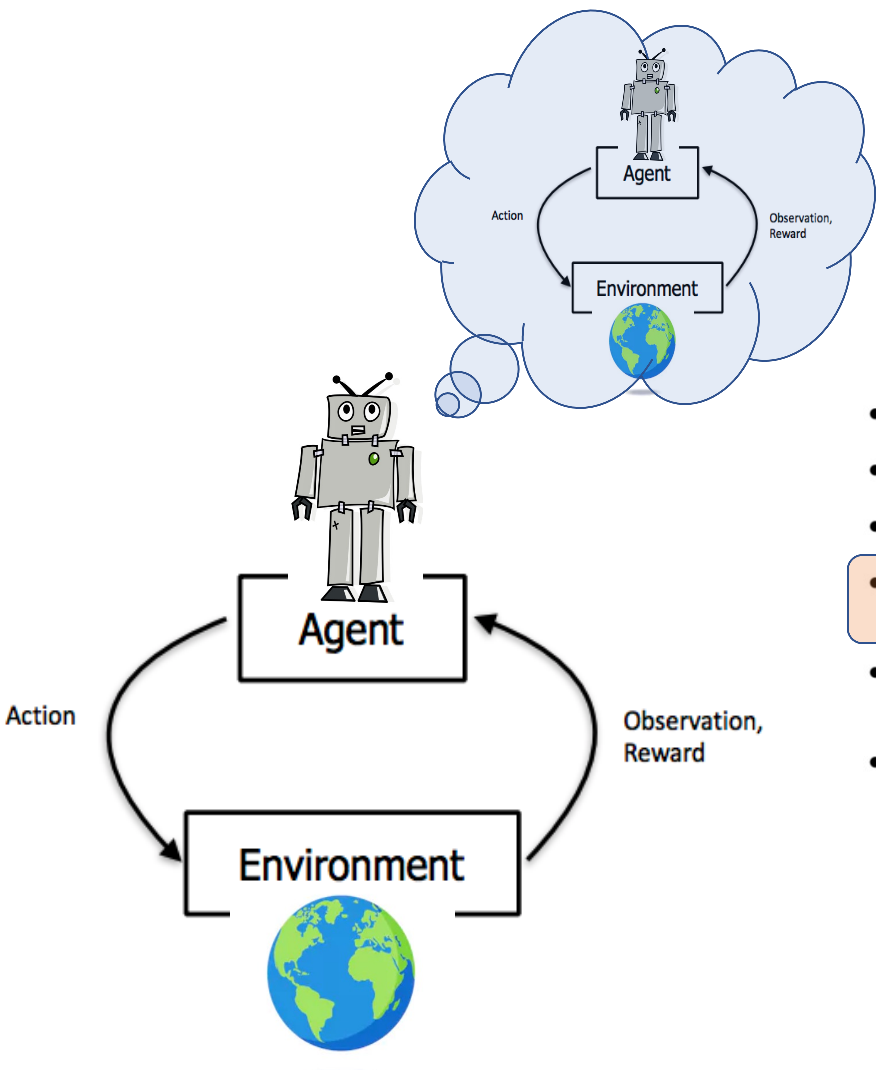
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights

  end for

end for

Output: θ_1, θ_2, ϕ ▷ Optimized parameters

Function Approximation: Model-based Methods

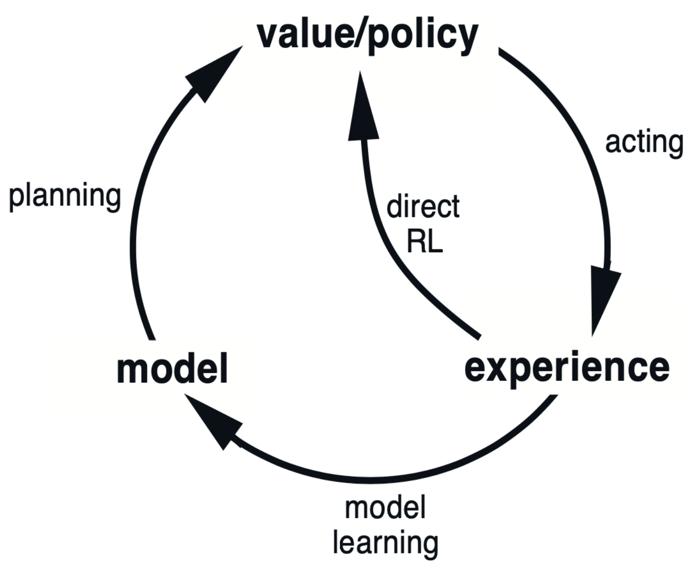


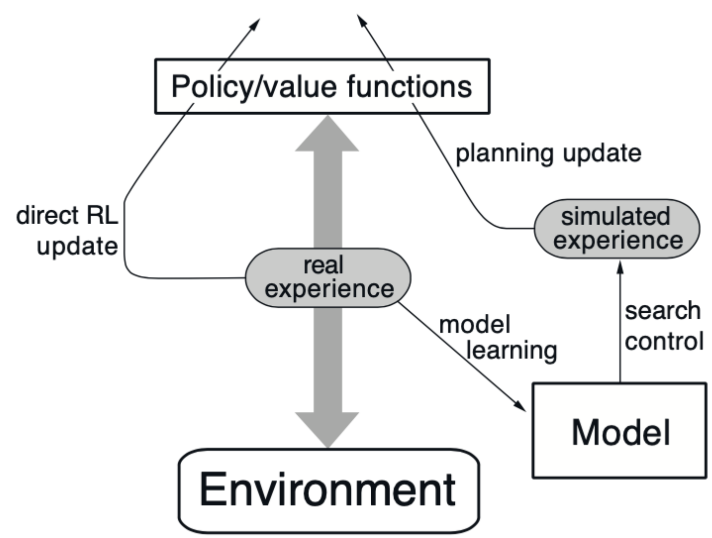
Model-Based Reinforcement Learning

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ a set of actions,
- $p_0(\mathcal{S})$ is the initial state distribution,
- $T(s_{t+1}|s_t, a_t)$ is the probability of transitioning from state $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ after action $a_t \in \mathcal{A}$,
- $R(r_{t+1}|s_t, a_t)$ is the probability of receiving reward $r_{t+1} \in R$ after executing action a_t while in state s_t,
- $\gamma \in [0, 1)$ is the discount factor.

Supervised learning problem!

Dyna: Integrating Planning, Acting, and Learning

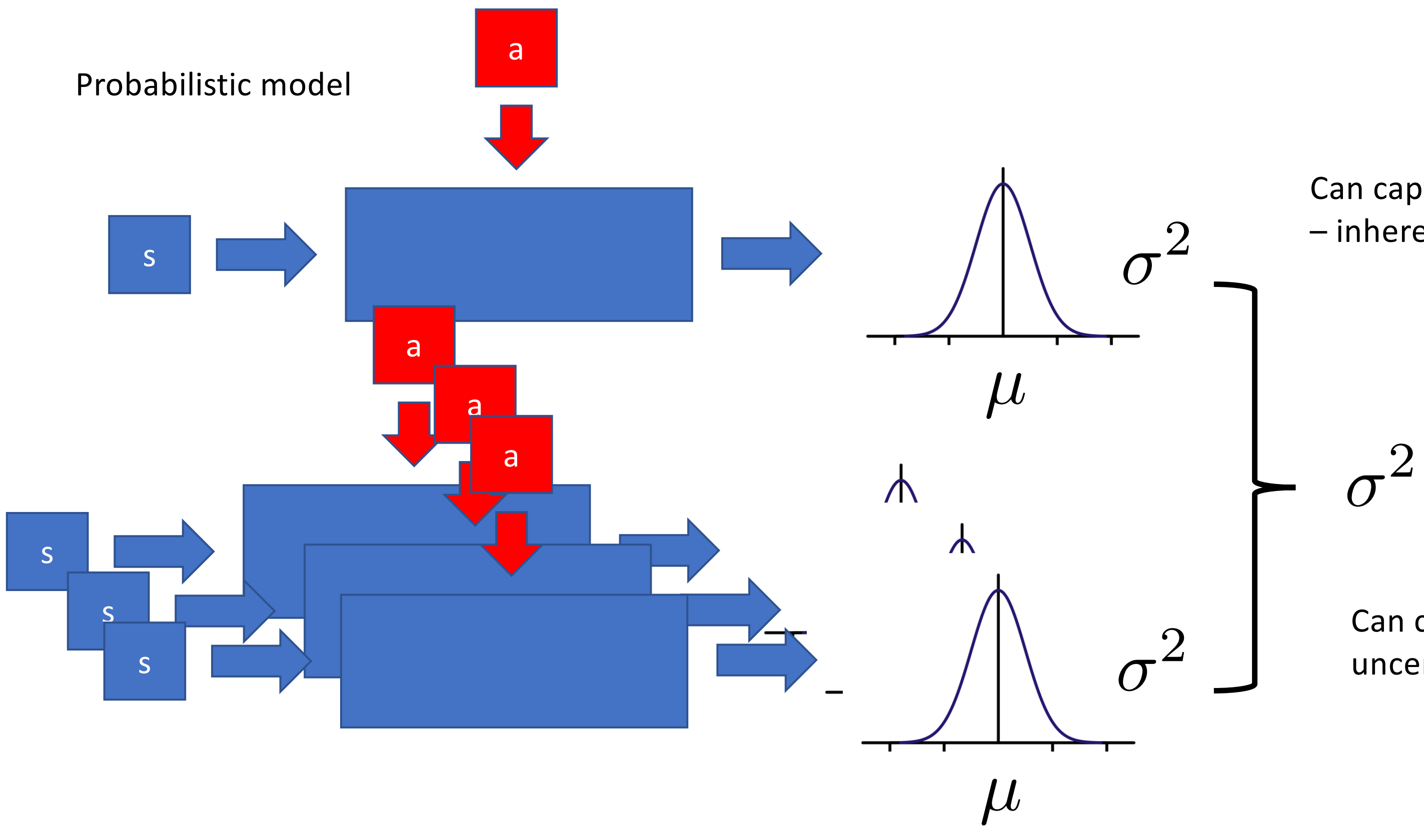




Sutton & Barto. Introduction to Reinforcement Learning.

Model Type

Probabilistic model



Can capture *aleatoric* uncertainty – inherent noise in the system

Can capture *epistemic* uncertainty – uncertainty due to scarcity of data

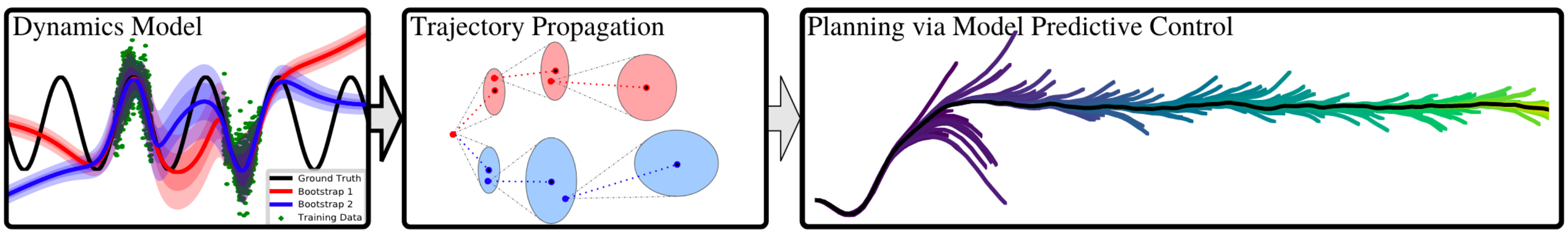
Model-Based Policy Optimization (MBPO)

MBPO (high-level)

1. Collect environment trajectories; add to $\mathcal{D}_{\text{env}}$
2. Train model ensemble on environment data $\mathcal{D}_{\text{env}}$
3. Perform *k*-step model rollouts branched from $\mathcal{D}_{\text{env}}$; add to $\mathcal{D}_{\text{model}}$
4. Update policy parameters on model data $\mathcal{D}_{\text{model}}$

Janner et al. When to Trust Your Model: Model-Based Policy Optimization. NeurIPS 2019.

Probabilistic Ensembles with Trajectory Sampling (PETS)



Chua et al. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. NeurIPS 2018.

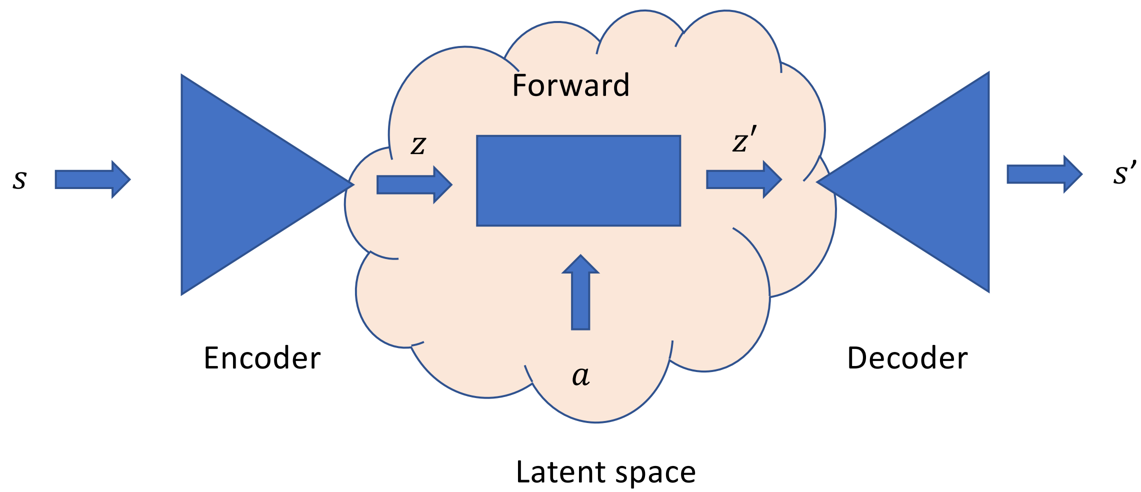
MBRL-Lib: A Modular Library for Model-based Reinforcement Learning

Luis Pineda Brandon Amos Amy Zhang Nathan O. Lambert
Roberto Calandra
Facebook AI Research
University of California, Berkeley
{lep,bda,amyzhang,rcalandra}@fb.com, nol@berkeley.edu

MBRL-Lib

`mbrl` is a toolbox for facilitating development of Model-Based Reinforcement Learning algorithms. It provides easily interchangeable modeling and planning components, and a set of utility functions that allow writing model-based RL algorithms with only a few lines of code.

Model-Based RL with Latent Models

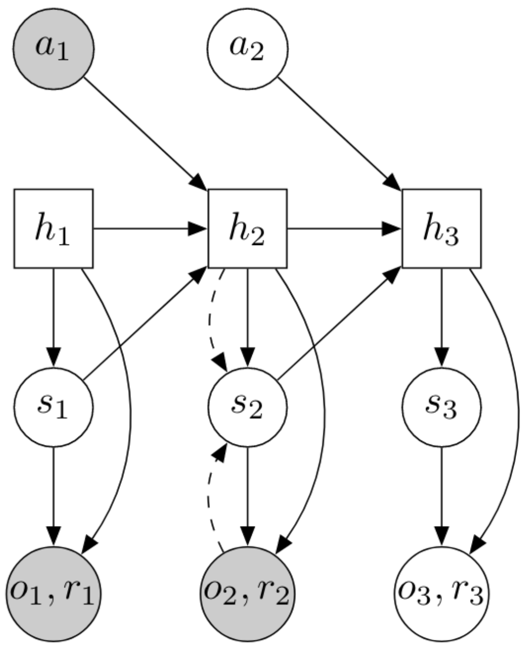


Deep Planning Network (PlaNet)

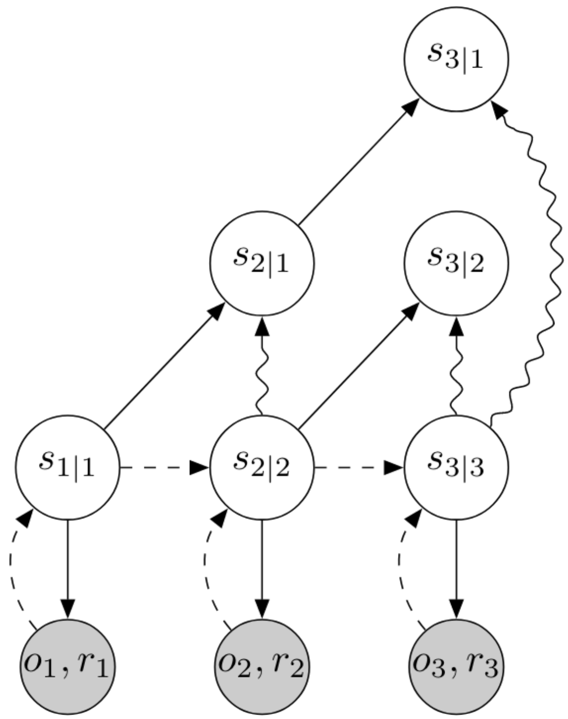
https://blog.research.google/2020/03/introducing-dreamer-scalable.html

Deep Planning Network (PlaNet)

(Like PETS)



Recurrent state-space model (RSSM)



Latent overshooting for planning

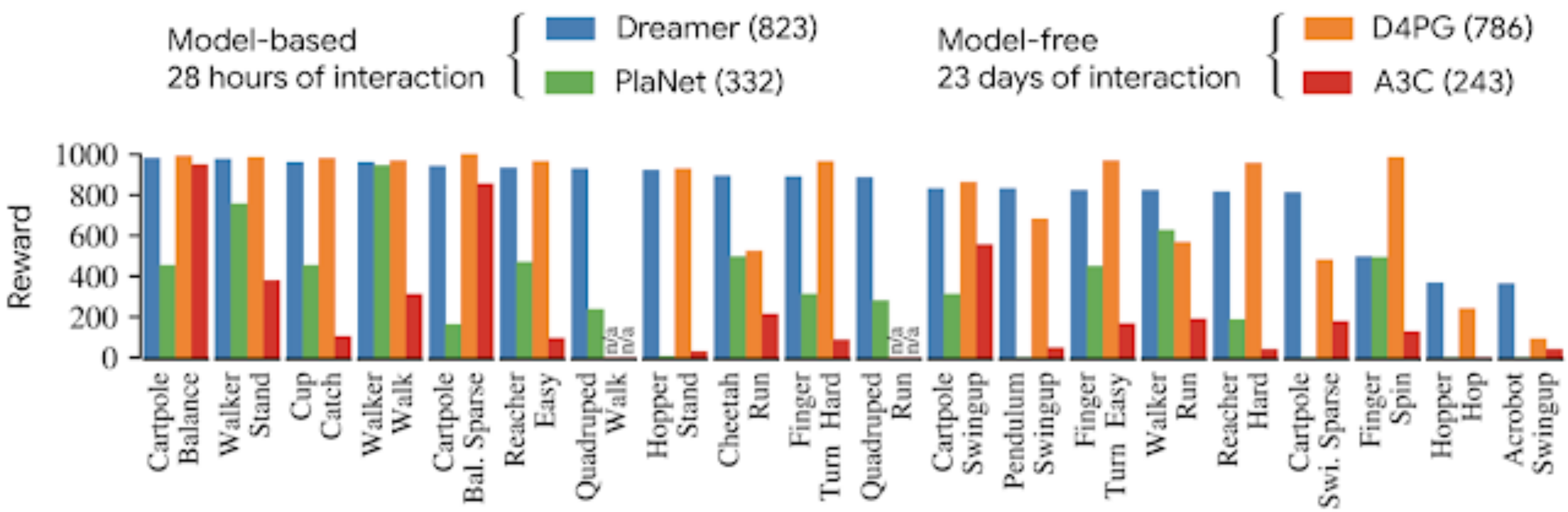
Hafner et al. Learning Latent Dynamics for Planning from Pixels. ICML 2019.

Dreamer

o_1

Hafner et al. DREAM TO CONTROL: LEARNING BEHAVIORS BY LATENT IMAGINATION. ICLR 2020.

Dreamer Results



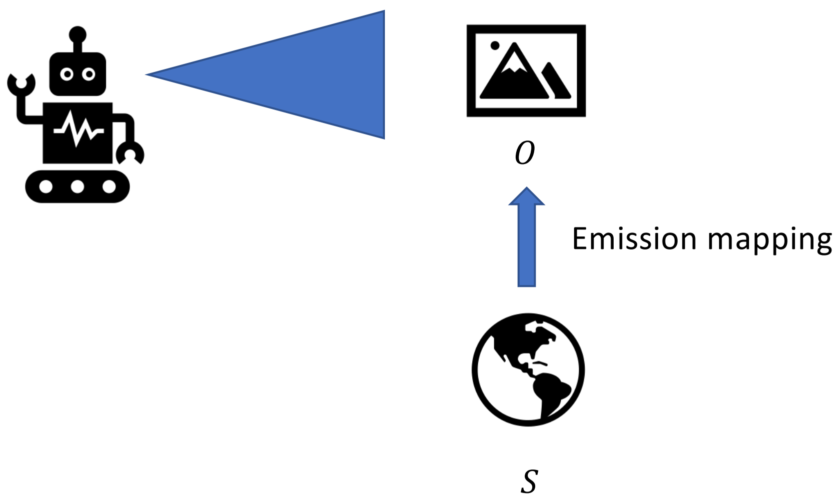
Hafner et al. DREAM TO CONTROL: LEARNING BEHAVIORS BY LATENT IMAGINATION. ICLR 2020.

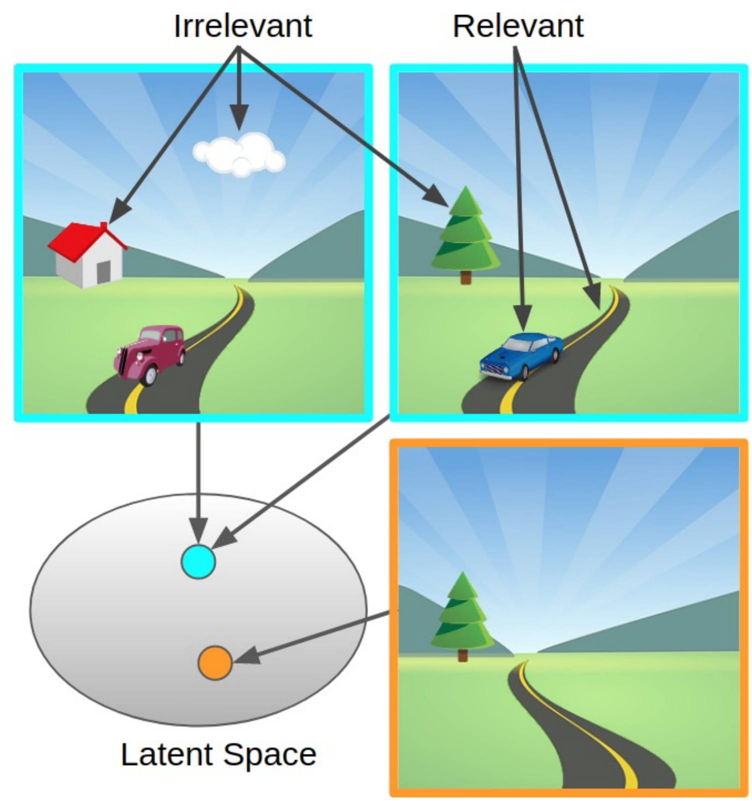
Outline: Second Half

- Function Approximation: Model-free Methods
  - DQN
  - REINFORCE and Policy gradient
  - Actor-Critic Methods
- Function Approximation: Model-based Methods
  - Dyna
  - MBPO
  - PETS
- Advanced Topics
  - Abstractions and Generalization
  - Leveraging Structure in RL
  - Self-supervised RL

Abstractions

A realistic additional assumption





Goal: Generalization to new observations *where the underlying MDP is the same*
Solution: Ignore irrelevant information

Block MDPs

A Block MDP family can be described by

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Reward function $\mathcal{R}$
- Discount factor γ
- Observation space $\mathcal{X}$
- Rendering mapping $q : \mathcal{S} \mapsto \mathcal{X}$

Assumption

Each observation x uniquely determines its generating state s. That is, the observation space $\mathcal{X}$ can be partitioned into disjoint blocks $\mathcal{X}_s$, each containing the support of the conditional distribution $q(\cdot|s)$.

Learning Invariant Representations for Reinforcement Learning without Reconstruction. AZ, R. McAllister, R. Calandra, Y. Gal, S. Levine. ICLR 2021 (Oral)

State Abstractions and Bisimulation

State abstractions have been studied as a way to distinguish relevant from irrelevant information in order to create a more compact representation for easier decision making and planning.

Definition

Given an MDP $\mathcal{M}$, an equivalence relation B between states is a bisimulation relation if, for all states $\mathrm{s}_i, \mathrm{s}_j \in \mathcal{S}$ that are equivalent under B (denoted $\mathrm{s}_i \equiv_B \mathrm{s}_j$) the following conditions hold:

$$\mathcal{R}(\mathrm{s}_i, \mathrm{a}) = \mathcal{R}(\mathrm{s}_j, \mathrm{a}) \quad \forall \mathrm{a} \in \mathcal{A}, \tag{1}$$

$$\mathcal{P}(G|\mathrm{s}_i, \mathrm{a}) = \mathcal{P}(G|\mathrm{s}_j, \mathrm{a}) \quad \forall \mathrm{a} \in \mathcal{A}, \quad \forall G \in \mathcal{S}_B, \tag{2}$$

where $\mathcal{S}_B$ is the partition of $\mathcal{S}$ under the relation B (the set of all groups G of equivalent states), and $\mathcal{P}(G|\mathrm{s}, \mathrm{a}) = \sum_{\mathrm{s}' \in G} \mathcal{P}(\mathrm{s}'|\mathrm{s}, \mathrm{a})$.

Learning Invariant Representations for Reinforcement Learning without Reconstruction. AZ, R. McAllister, R. Calandra, Y. Gal, S. Levine. ICLR 2021 (Oral)

Bisimulation Metric

Learn a representation where L1 distance between any two states is their bisimilarity:

Definition

Given a finite MDP $\mathcal{M} : (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, let $c \in (0, 1)$ be a discount factor. Let $\mathfrak{met}$ be the space of bounded pseudometrics on $\mathcal{S}$ equipped with the metric induced by the uniform norm. Define $F : \mathfrak{met} \mapsto \mathfrak{met}$ by

$$F(\mathrm{s}_i, \mathrm{s}_j) = \max_{\mathrm{a} \in \mathcal{A}} (1 - c)|r^{\mathrm{a}}_{\mathrm{s}_i} - r^{\mathrm{a}}_{\mathrm{s}_j}| + cW(\mathcal{P}^{\mathrm{a}}_{\mathrm{s}_i}, \mathcal{P}^{\mathrm{a}}_{\mathrm{s}_j}). \tag{3}$$

Then F has a unique fixed point $\tilde{d}$ which is the bisimulation metric.

Learning Invariant Representations for Reinforcement Learning without Reconstruction. AZ, R. McAllister, R. Calandra, Y. Gal, S. Levine. ICLR 2021 (Oral)

On-Policy Bisimulation Metrics

Let's modify the previous definition to get rid of the max over actions:

Definition

Given a finite MDP $\mathcal{M} : (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, let $c \in (0, 1)$ be a discount factor. Let $\mathfrak{met}$ be the space of bounded pseudometrics on $\mathcal{S}$ equipped with the metric induced by the uniform norm. Define $F : \mathfrak{met} \mapsto \mathfrak{met}$ by

$$F(\mathsf{s}_i, \mathsf{s}_j; \pi) = \mathbb{E}_\pi\left[(1 - c)|r^{\mathsf{a}}_{\mathsf{s}_i} - r^{\mathsf{a}}_{\mathsf{s}_j}| + cW(\mathcal{P}^{\mathsf{a}}_{\mathsf{s}_i}, \mathcal{P}^{\mathsf{a}}_{\mathsf{s}_j})\right]. \tag{4}$$

Then F has a unique fixed point $\tilde{d}_\pi$ which is the on-policy bisimulation metric.

Scalable methods for computing state similarity in deterministic Markov Decision Processes. P. Castro. AAAI 2020.

Generalization to new observations and rewards

Theorem: Connections to causal feature sets

If we partition observations using the bisimulation metric, those clusters (a bisimulation partition) correspond to the causal feature set of the observation space with respect to current and future reward.

Theorem: Task Generalization

Given an encoder $\phi : O \mapsto S$ that maps observations to a latent bisimulation metric representation where $||\phi(s_i) - \phi(s_j)||_2 := \tilde{d}(s_i, s_j)$, S encodes information about all the causal ancestors of the reward $AN(R)$.

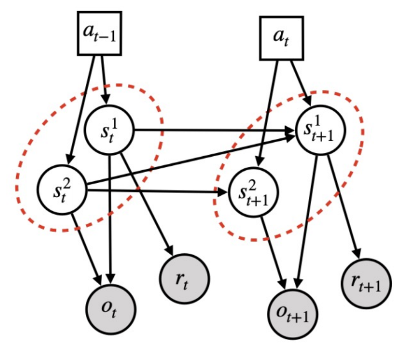
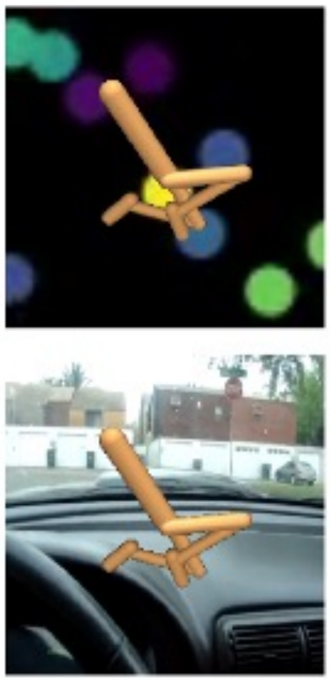
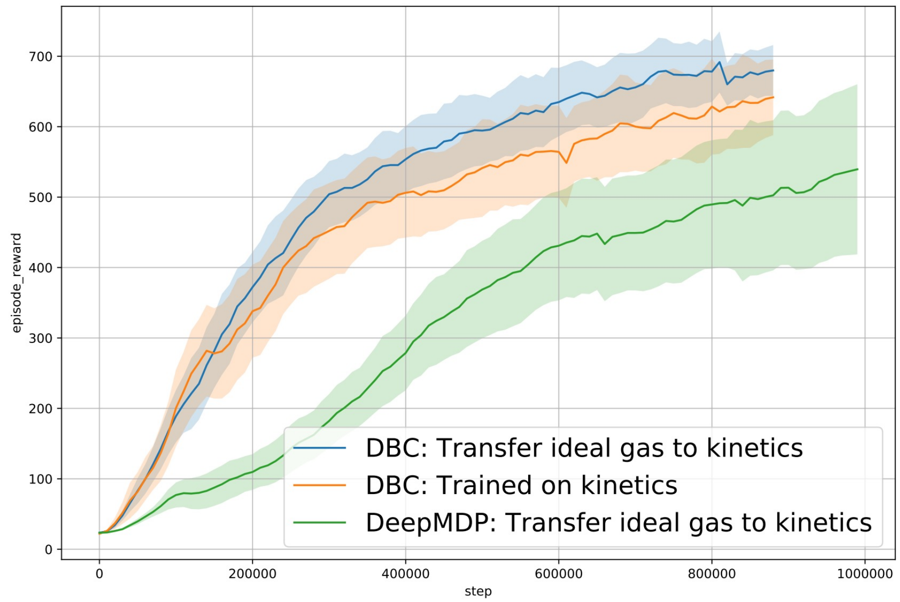


Figure 3: Causal graph of two time steps. Reward depends only on $\mathbf{s}^1$ as a causal parent, but $\mathbf{s}^1$ causally depends on $\mathbf{s}^2$, so AN(R) is the set $\{\mathbf{s}^1, \mathbf{s}^2\}$.

Learning Invariant Representations for Reinforcement Learning without Reconstruction. AZ, R. McAllister, R. Calandra, Y. Gal, S. Levine. ICLR 2021 (Oral)

Generalization to new observations





Walker Walk

Learning Invariant Representations for Reinforcement Learning without Reconstruction. AZ, R. McAllister, R. Calandra, Y. Gal, S. Levine. ICLR 2021 (Oral)

Generalization

- Deep RL has had many successes

Pinpointing some failures

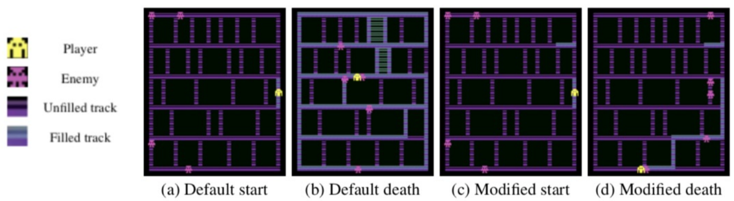


Figure: Train and Test on Atari proposed by Witty et al. 2018

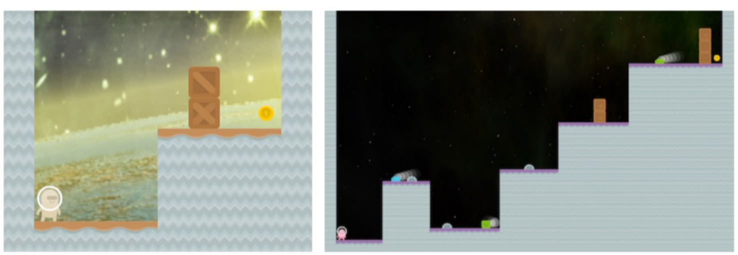


Figure: Train and Test on CoinRun proposed by Cobbe et al. 2019

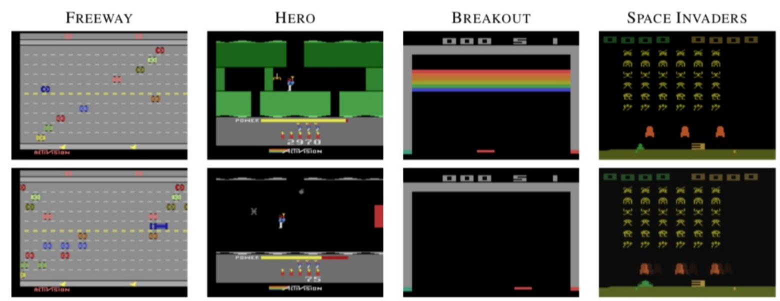
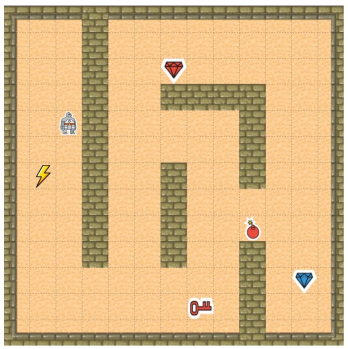
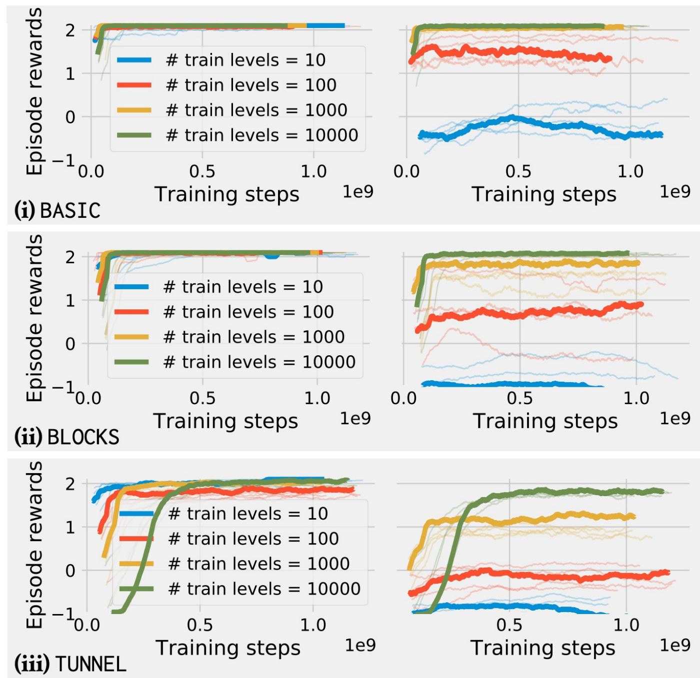


Figure: Train and Test on Atari proposed by Farebrother, Machado, and Bowling 2018[2] .

Generalization



(a) An example of a TUNNEL maze.

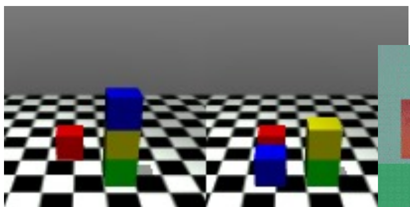


[C. Zhang et al. A Study on Overfitting in Deep Reinforcement Learning.]

Why the discrepancy?

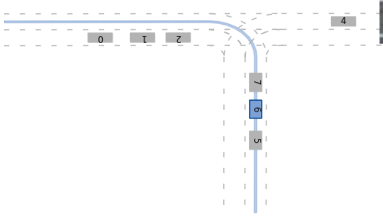
- Deep RL works really well in single task settings in simulation with millions of transitions.
- Works less well in visually complex and natural settings – we don't see the same generalization performance we're getting in computer vision and NLP.

Open Problems: Compositionality



ProcGen

Useful
Assumptions?

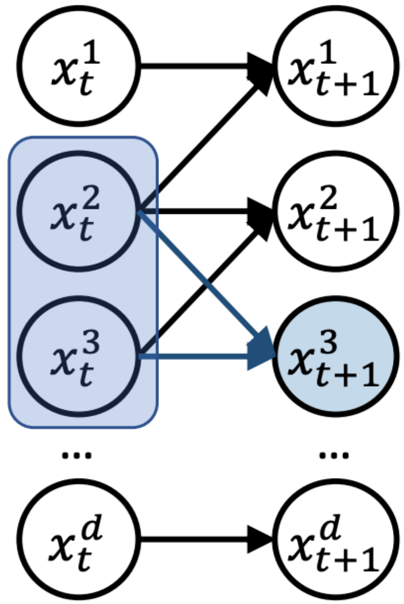


Bark Simulator

Factored MDP

- State space is made up of discrete variables:

$$\mathcal{X} := \{\mathcal{X}^1, \mathcal{X}^2, \ldots, \mathcal{X}^d\}$$



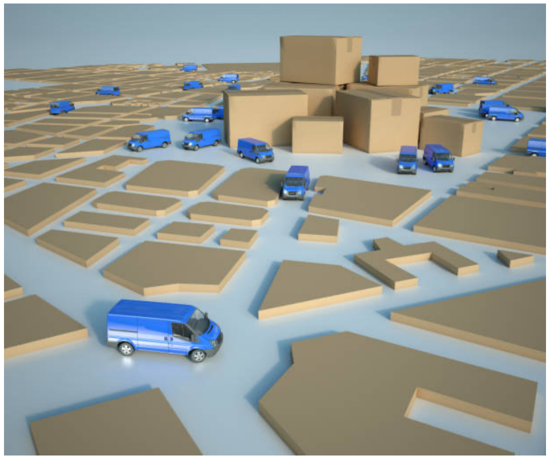
Assumption: Factored Transitions

For given full state vectors $x_t, x_{t+1} \in \mathcal{X}$, action $a \in \mathcal{A}$, and x_i denoting the i^{th} dimension of state x we have $P(x_{t+1}|x_t, a) = \prod_i P(x_{t+1}^i|x_t, a)$.

Assumption: Factored Rewards

For given full state vectors $x_t, x_{t+1} \in \mathcal{X}$, action $a \in \mathcal{A}$, and x_i denoting the i^{th} dimension of state x we have $R(x_{t+1}|x_t, a) = \sum_i R(x_{t+1}^i|x_t, a)$.

Relational MDP



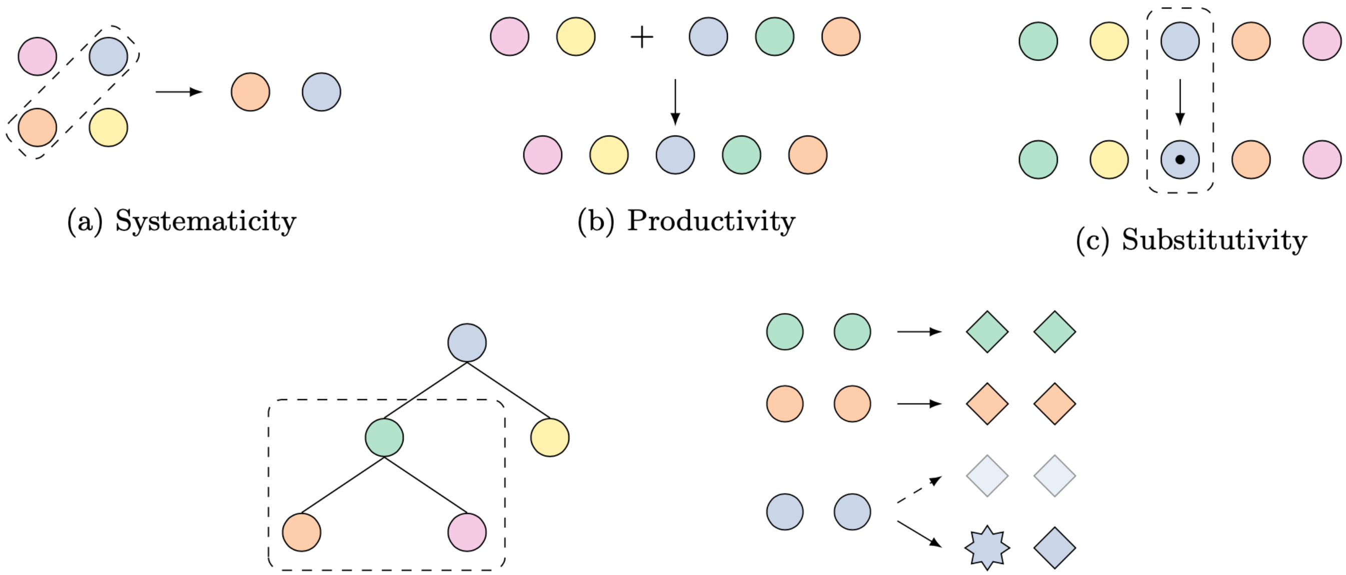
A Relational MDP family can be described by

- C: Set of classes denoting different types of object
  e.g., {Box, Truck, City}
- F: Set of function schemata that take objects as input
  e.g., {Bin(Box, City), On(Box, Truck)}
- A: Set of action schemata that operate on objects
  e.g., {Unload(Box,Truck,City), Load(Box, City, Truck), Drive(Truck,City,City)}
- D: Set of domain objects, each associated with a single type from C
- T: Transition function
- R: Reward model

C, F, and A are sets of relational schemata.

Boutilier, Reiter, & Price 2001

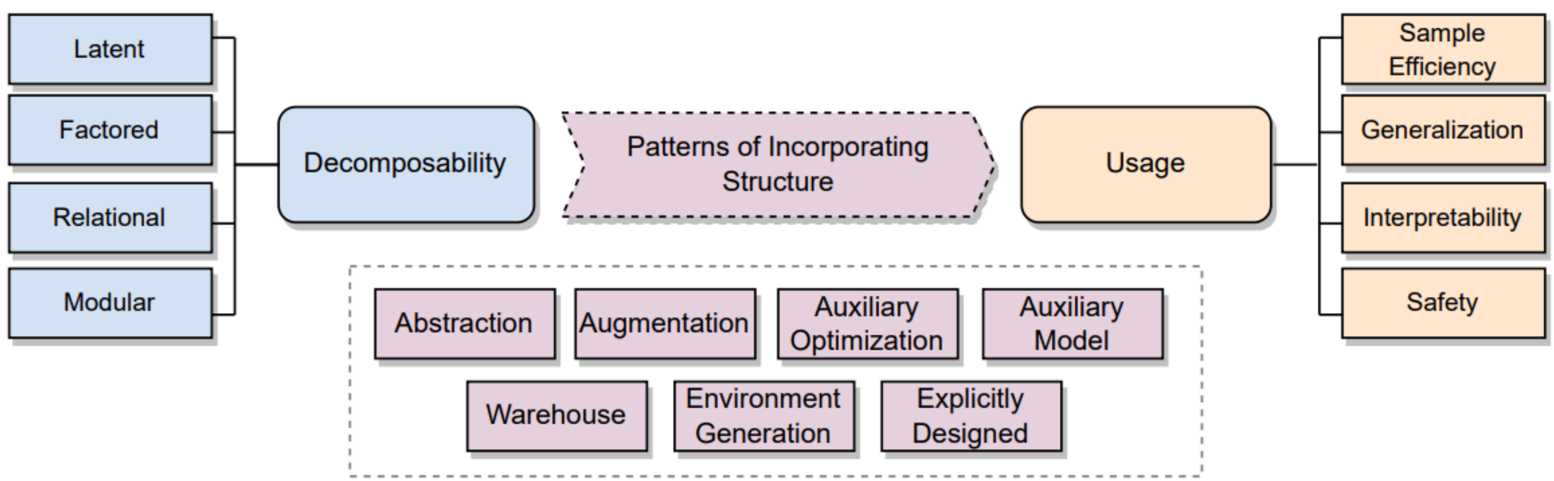
Forms of Compositional Generalization



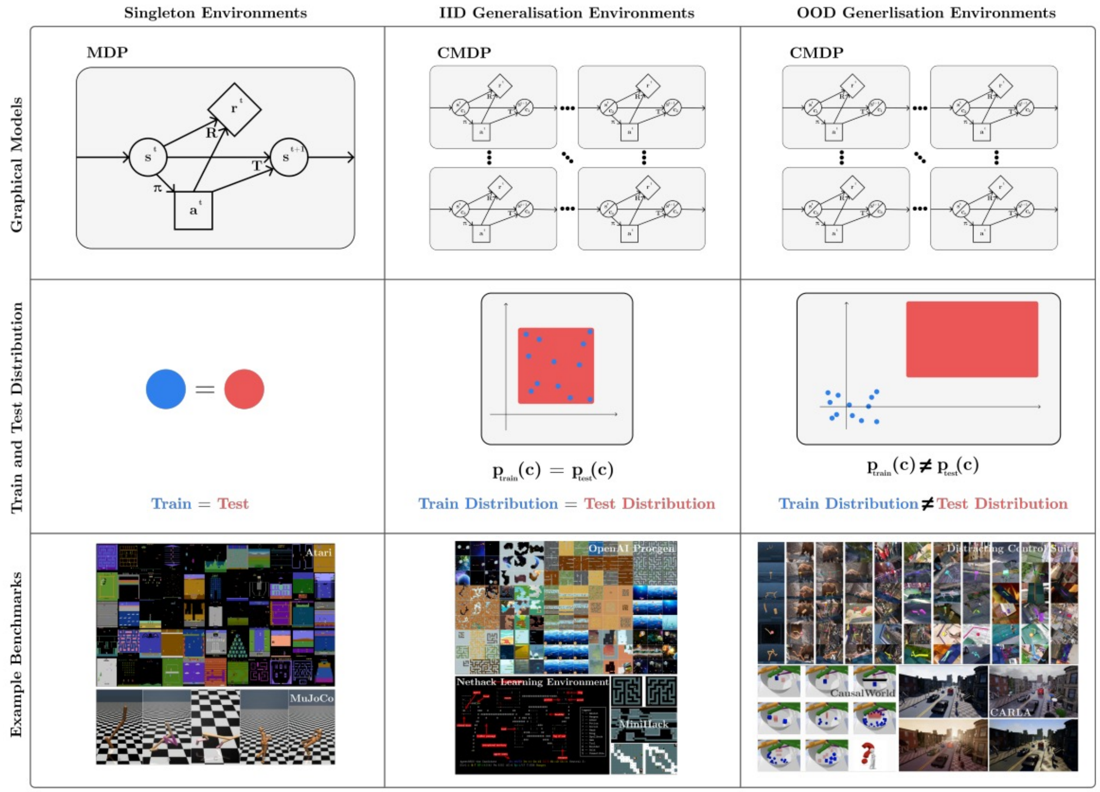
(a) Systematicity (b) Productivity (c) Substitutivity (d) Localism (e) Overgeneralisation

Figure from: Compositionality decomposed: how do neural networks generalize? Hupkes et al. 2020.

Structure in Reinforcement Learning: A Survey and Open Problems

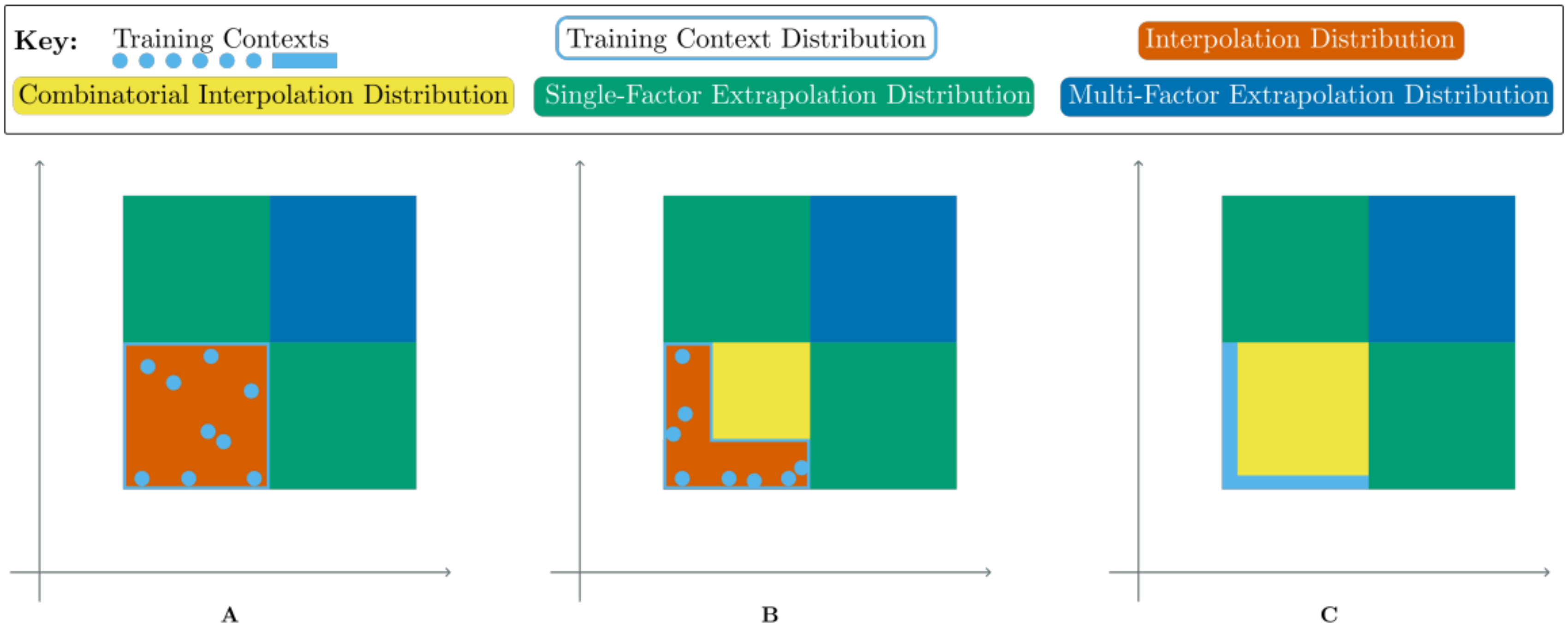


Defining Generalization Types



[Kirk, **AZ**, et al. A Survey of Zero-shot Generalisation in Deep Reinforcement Learning.]

Evaluating Generalization



[Kirk, **AZ**, et al. A Survey of Zero-shot Generalisation in Deep Reinforcement Learning.]

Self-Supervised RL

- What is self-supervised RL?
  - Assume no access to “labels” (reward)

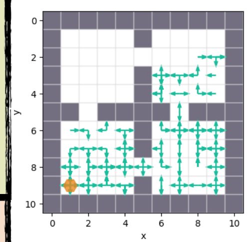
- Why should we care?
  - Pre-training and zero-shot/finetuning regime
  - What can we do with unlabeled sequential data, and how will it help with downstream tasks?
  - Hint: video generation models and LLMs!

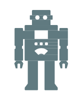
Yann Ollivier

LEARNING ONE REPRESENTATION TO OPTIMIZE ALL REWARDS

MOTIVATION

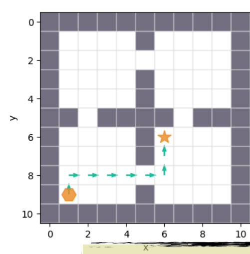
- You have an environment in which you can perform actions but no reward signal
- Your mission is to build a summary of the environment



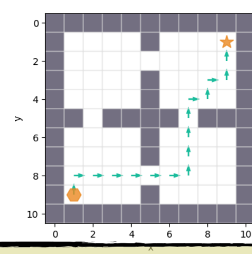


- Such that as soon as I describe a reward function, you immediately know what to do to maximize your reward.

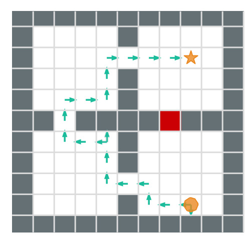
Task 1



Task 2



Task 3



Build fully controllable agents able to follow instructions such as "reach this state while avoiding that place"

Slide Credit: Ahmed Touati

MAIN RESULT

- Informal Theorem: There exists a representation of an environment on which we can directly read all optimal policies of all reward functions:

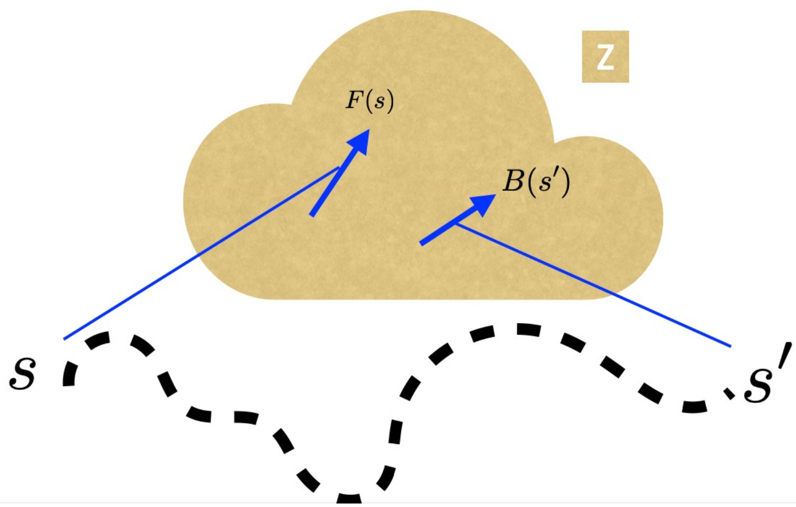
The forward-backward (FB) representation

- It is learnable from reward-free interactions, off policy.
- At test time, reward functions may be specified
  - either explicitly ("reach this state"),
  - or as a function over states,
  - or by reward samples as in classical RL.

Slide Credit: Ahmed Touati

IDEA

- Idea: F(s) represents the future of a state s, and B(s) represents the past of a state.
- Training criterion: If it is easy to reach s' from s (in many steps), then $F(s)^\top B(s')$ is large
- This training criterion is guaranteed to provide optimal policies.



Slide Credit: Ahmed Touati

OUTLINE OF THE METHOD

Unsupervised phase

- Choose a representation space $Z = \mathbb{R}^d$
- Learn two representations

$$F : \text{States} \times \text{Actions} \times Z \to Z$$
$$B : \text{States} \to Z$$

according to some unsupervised criterion (next slide)

Task identification phase

- Once rewards are accessible, compute

$$z_R = \mathbb{E}[r(s)B(s)]$$

- For instance, $z_R = B(s)$ if the reward is located at state s

Exploitation phase

- Apply policy $\pi_{z_R}(s) = \arg\max_a F(s, a, z_R)^\top z_R$

No Planning!

Slide Credit: Ahmed Touati

UNSUPERVISED PHASE

- Theorem: If F and B optimize their training criterion perfectly, then the obtained policy is guaranteed to be optimal, whatever the reward.
- Finite representation dimension => approximate training => approximate policies with controlled error.

Unsupervised training criterion: for all s, a, s', z,

$$F(s,a,z)^\top B(s') \approx \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s' \mid s, a, \pi_z)$$

$$\pi_z(s) = \arg\max_a F(s,a,z)^\top z$$

- Learn an occupancy model for many behaviours
- "Model-based lite": no synthesis of states or trajectories

: Ahmed Touati

FB TRAINING

- F and B satisfy a Bellman equation

$$F(s_0, a_0, z)^\top B(s') = \underbrace{\delta_{s_0}(s)}_{\textit{Infinitely sparse reward !!}} + \gamma \mathbb{E}_{s_1 \sim P(\cdot|s_0,a_0)}\left[F(s, \pi_z(s_1), z)^\top B(s')\right]$$

- Can be learned by TD-style from Blier-Tallec-Ollivier 2021 accounting for Dirac exactly even with continuous states

[1] Learning Successor States and Goal-Dependent Values: A Mathematical Viewpoint. Léonard Blier, Corentin Tallec, Yann Ollivier, arxiv 2021

Slide Credit: Ahmed Touati

CONCLUSION AND PERSPECTIVES

Take-home message:

There exists a learnable representation of an environment on which we can read all optimal policies of all reward functions (with arbitrary precision by increasing the dimension).

- Incorporating priors is possible (on rewards, relevant features)
- For a single, fixed environment.
- Long-range dependencies are captured well but local blurring of details in the reward.
- Allows for zero-shot extraction of the optimal policy for any downstream reward function.

Slide Credit: Ahmed Touati

Many research areas I did not cover

- Exploration
- Offline setting
- Arbitrary data
- Different learning signals
- Safety
- Interpretability
- Transfer
- Meta RL

Outline: Second Half

- Function Approximation: Model-free Methods
  - DQN
  - REINFORCE and Policy gradient
  - Actor-Critic Methods
- Function Approximation: Model-based Methods
  - Dyna
  - MBPO
  - PETS
- Advanced Topics
  - Abstractions and Generalization
  - Leveraging Structure in RL
  - Self-supervised RL