Lecture: Reinforcement Learning

http://bicmr.pku.edu.cn/~wenzw/bigdata2017.html

Acknowledgement: this slides is based on Prof. David Silver's lecture notes

Thanks: Mingming Zhao for preparing this slides

Outline

- Introduction of MDP
- Dynamic Programming
- Model-free Control
- Large-Scale RL
- Model-based RL

What is RL

Reinforcement learning

- is learning what to do–how to map situations to actions–so as to maximize a numerical reward signal. The decision-maker is called the *agent*, the thing it interacts with, is called the *environment*.
- A reinforcement learning task that satisfies the *Markov* property is called a Markov Decision process, or MDP
- We assume that all RL tasks can be approximated with Markov property. So this talk is based on MDP.

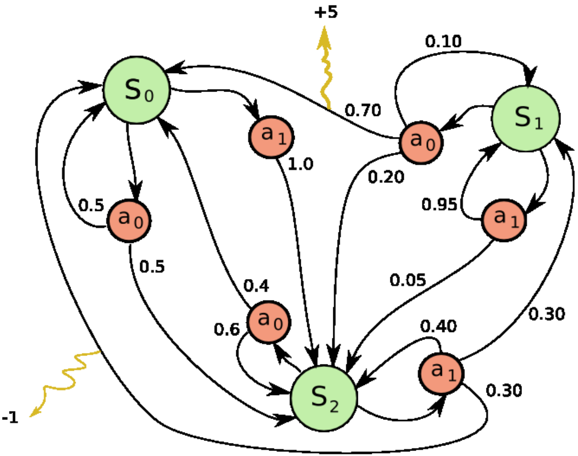
Definition

A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$:

- $\mathcal{S}$ is a finite set of states, $s \in \mathcal{S}$
- $\mathcal{A}$ is a finite set of actions, $a \in \mathcal{A}$
- $\mathcal{P}$ is the transition probability distribution.
  probability from state s with action a to state s': $P(s'|s, a)$
  also called the model or the dynamics
- r is a reward function, $r(s, a, s')$
  sometimes just $r(s)$
  or r_t after time step t
- $\gamma \in [0, 1]$ is a discount factor, why discount?

Example

A simple MDP with three states and two actions:



Markov Property

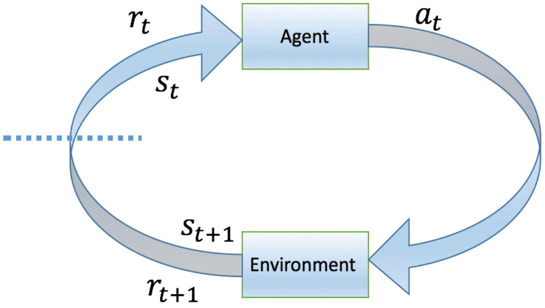
A state s_t is Markov iff

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, ..., s_t)$$

- the state captures all relevant information from the history
- once the state is known, the history may be thrown away
- i.e the state is a sufficient statistic of the future

Agent and Environment

- The agent selects actions based on the observations and rewards received at each time-step t
- The environment selects observations and rewards based on the actions received at each time-step t



Policy

$$\pi(a|s) = \mathbb{P}(a_t = a | s_t = s)$$

- policy π defines the behaviour of an agent
- for MDP, the policy depends on the current state(Markov property)
- deterministic policy: $a = \pi(s)$

Value function

Denote

$$G(t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$$
$$= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

state-value function

- $V_\pi(s)$ is the total amount of reward expected to accumulate over the future starting from the state s and then following policy π
- $V_\pi(s) = E_\pi(G(t)|s_t = s)$

action-value function

- $Q_\pi(s, a)$ is the total amount of reward expected to accumulate over the future starting from the state s, taking aciton a, and then following policy π
- $Q_\pi(s, a) = E_\pi(G(t)|s_t = s, a_t = a)$

Bellman Equation

$$
\begin{aligned}
V_\pi(s) =& E_\pi(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s) \\
=& E_\pi(r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \ldots) | s_t = s) \\
=& E_\pi(r_{t+1} + \gamma G(t+1) | s_t = s) \\
=& E_\pi(r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s)
\end{aligned}
$$

- For state-value function

$$
\begin{aligned}
V_\pi(s) =& E_\pi(r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s) \\
=& \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)[r(s,a,s') + \gamma V_\pi(s')]
\end{aligned}
$$

- Similarly,

$$
\begin{aligned}
Q_\pi(s,a) =& E_\pi(r(s) + \gamma Q_\pi(s',a') | s,a) \\
=& \sum_{s' \in \mathcal{S}} P(s'|s,a)[r(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a')]
\end{aligned}
$$

Bellman Equation

- The Bellman equation is a linear equation, it can be solved directly, but only possible for small MDP
- The Bellman equation motivates a lot of iterative methods for MDP,

  Dynamic Programming
  Monte-Carlo evaluation
  Temporal-Difference learning

Optimal value function

- The optimal state-value function $V_*(s)$ is

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

- The optimal action-value function $q_*(s,a)$ is

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

- The goal for any MDP is finding the optimal value function

- Or equivalently an optimal policy π^*

  for any policy π, $V_{\pi^*}(s) \geq V_\pi(s), \forall s \in \mathcal{S}$

Bellman Optimal Equation

$$V_*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)[r(s,a,s') + \gamma V_*(s')]$$

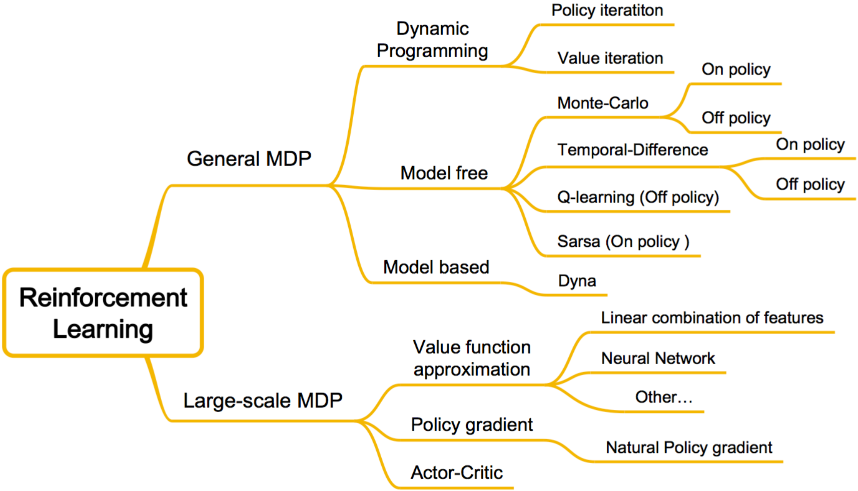
$$q_*(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)[r(s,a,s') + \gamma \max_{a' \in \mathcal{A}} q_*(s',a')]$$

- Bellman optimal equation is non-linear
- many iterative methods

  value iteration
    Q-learning

Structure of RL

- Basis: Policy Evaluation + Policy Improvement



Policy iteration

- Policy evaluation
  - for a given policy π, evaluate the state-value function $V_\pi(s)$ at each state $s \in \mathcal{S}$
  - iterative application of Bellman expectation backup

$$V_\pi(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)[r(s,a,s') + \gamma V_\pi(s')]$$

- Policy improvement
  - consider deterministic policy(greedy policy)

$$\pi(s) \leftarrow arg\max_a \sum_{s' \in \mathcal{S}} P(s'|s,a)[r(s,a,s') + \gamma V_\pi(s')]$$

  - ϵ-greedy policy

Policy iteration

Policy Iteration

Require:

Initial π, $V(s)$, for each $s \in \mathcal{S}$

STEP 1. Policy evaluation:

While $\Delta \neq 0$

for each $s \in \mathcal{S}$ **do**

$\delta \leftarrow V(s)$;

$V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|a,s)[r(s,a,s') + \gamma V(s')]$;

$\Delta \leftarrow |\delta - V(s)|$

end for

End

STEP 2. Policy improvement:

$\omega = 1$;

for each $s \in \mathcal{S}$ **do**

$\Omega \leftarrow \pi(\cdot|s)$;

$a^* \leftarrow arg\max_a \sum_{s' \in \mathcal{S}} P(s'|a,s)[r(s,a,s') + \gamma V(s')]$, and $\pi(a|s) = \begin{cases} 1, & \text{if } a = a^* \\ 0, & \text{if } a \neq a^* \end{cases}$,

$w \leftarrow \min(\omega, \Omega = \pi(a|s))$

end for

IF ω, output $\pi, V(s)$; else back to STEP 1.

return

Value iteration

- One-step truncated policy evaluation
- Iterative application of Bellman optimality backup

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$$

- Learn optimal value function directly
- Unlike policy iteration, there is no explicit policy

Value iteration

Value Iteration

Require:

Initial $V(s)$, for each $s \in \mathcal{S}$

Repeat

While $\Delta \neq 0$;

for each $s \in \mathcal{S}$ **do**

$\delta \leftarrow V(s)$;

$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|a, s)[r(s, a, s') + \gamma V(s')]$;

$\Delta \leftarrow |\delta - V(s)|$

end for

End

output a deterministic policy π such that

$a \leftarrow arg\max_a \sum_{s' \in \mathbb{S}} P(s'|a, s)[r(s, a, s') + \gamma V(s')], and\ \pi(a|s) = 1,$

return

State-value function space

- Consider the vector space $\mathcal{V}$ over state-value functions
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a state-value function $V(s)$
- For any $U, V \in \mathcal{V}$, we measure the distance between U, V

$$||U - V||_\infty = \max_{s \in \mathcal{S}} |U(s) - V(s)|$$

Convergence

Define the Bellman expectation backup operator T^{π}, for any state s

$$T^{\pi}(V(s)) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$$

then

$$\begin{aligned} \|T^{\pi}(V(s)) - T^{\pi}(U(s))\|_{\infty} =& \|\gamma \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)[V(s') - U(s')]\|_{\infty} \\ \leq& \gamma \max_{s' \in \mathcal{S}} |V(s') - U(s')| \end{aligned}$$

Convergence

- thus

$$\|T^\pi(V) - T^\pi(U)\|_\infty \leq \gamma \|V - U\|_\infty$$

- we call the operator T^π is a γ-contraction

Contraction Mapping Theorem

For any metric space $\mathcal{V}$ that is complete under an operator $T(v)$, where T is a γ-contraction, then

- T converges to a unique fixed point
- At a linear convergence rate of γ

Convergence

- Both the Bellman expectation operator T^{π} and the Bellman optimality operator T^{*} are γ-contraction
- Bellman equation shows V_{π} is the fixed point of T^{π}
- Bellman optimality equation shows V_{*} is the fixed point of T^{*}
- Iterative policy evaluation converges on V_{π}, policy iteration converges at V_{*}
- Value iteration converges at V_{*}

Model-free Algorithm

- Previous discussion
  planning by dynamic programming
  the model and dynamics of a MDP are required
  curse of dimensionality
- Next discussion
  model-free prediction and control
  estimate and optimize the value function of an unknown MDP
- In RL, we always deal with a unknown model, exploration and exploitation are needed

Model-free prediction

- Goal: learn $V_\pi(s)$ from episodes of experience under policy π
- On policy learning
- No requirement of MDP transitions or rewards
- Monte-Carlo learning
- Temporal-Difference learning

Monte-Carlo

- Update $V_\pi(s)$ incrementally from episodes of experience under policy π
- Every(or the first) time-step t that state s is visited in an episode

$$
\begin{aligned}
N(s) &\leftarrow N(s) + 1 \\
V(s) &\leftarrow \frac{(N(s)-1)V(s) + G_t}{N(s)} \\
or,\ \ V(s) &\leftarrow V(s) + \alpha(G_t - V(s))
\end{aligned}
$$

- By the law of large numbers, $V(s) \to V_\pi(s)$ as $N(s) \to \infty$
- MC must learns from complete episodes, no bootstrapping.

Temporal-Diference

- The simplest: TD(0)
- Update $V_\pi(s)$ incrementally from episodes of experience under policy π
- The update only requires one step episode

$$V(s) \leftarrow V(s) + \alpha(r(s,a,s') + \gamma V(s') - V(s))$$

- TD target: $r(s,a,s') + \gamma V(s')$
- TD error: $r(s,a,s') + \gamma V(s') - V(s)$
- $r(s,a,s')$ is the observed reward from one step forward simulation
- TD(0) learns from incomplete episodes, by bootstrapping.

Comparison between MC and TD(0)

- TD(0) can learn before knowing the final outcome, even without final outcome.
- MC must wait until the end of episode before return is known
- In MC, return G_t is unbiased estimate of $V_\pi(s_t)$, it depends an many random actions, transitions and rewards, which yields high variance
- However, TD(0) target is biased estimate of $V_\pi(s_t)$, it depends an one random action, transition and reward, which yields low variance
- MC has good convergence properties, and not very sensitive to initial value of $V(s)$
- TD(0) also converges , and more sensitive to initial value of $V(s)$. TD(0) is more efficient than MC

TD(λ)

- Consider n-step reward, $n = 1, 2, ...$, define

$$G_t^n = r_{t+1} + \gamma r_{t+2} + ... + \gamma^n r_{t+n+1} + \gamma^{n+1} V(s_{t+n+1})$$

- n-step TD(0) learning

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^n - V(s_t))$$

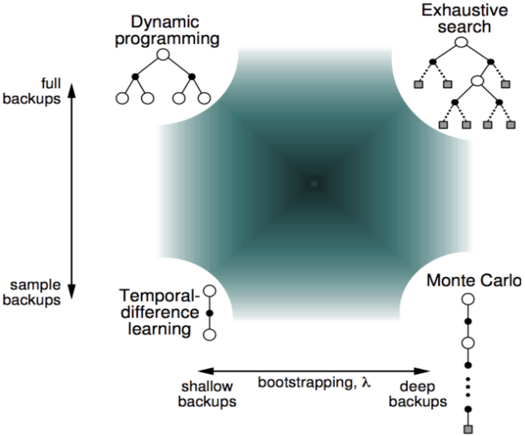
- TD(0)$\rightarrow$MC, as $n \rightarrow \infty$
- λ-return G_t^λ combines all n-step returns G_t^n

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

- Forward-view TD(λ)

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^\lambda - V(s_t))$$

Unified view of RL



Policy Improvement

After evaluate the value function of policy π, now improve policy

- greedy policy improvement

$$\pi^{'}(s) = arg \max_a \sum_{s' \in \mathcal{S}} P(s'|s,a)(r(s,a,s') + \gamma V_\pi(s'))$$
$$\pi^{'}(s) = arg \max_a Q_\pi(s,a)$$

- it is model-free and easier to obtain policy from $Q_\pi(s,a)$

- $\epsilon - greedy$ policy improvement

$$\pi^{'}(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + 1 - \epsilon, & a = arg \max_a Q(s,a) \\ \frac{\epsilon}{|\mathcal{A}|}, & o.w \end{cases}$$

- ϵ-greedy policy ensures continual exploration, all actions are tried

ϵ-Greedy Policy Improvement

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy $\pi^{'}$ w.r.t Q_π is an improvement, i.e $V_{\pi^{'}}(s) \geq V_\pi(s)$, $\forall s \in \mathcal{S}$.

Pf:

$$
\begin{aligned}
Q_\pi(s, \pi^{'}(s)) =& \sum_{a \in \mathcal{A}} \pi^{'}(a|s) Q_\pi(s,a) \\
=& (1-\epsilon) \max_{a \in \mathcal{A}} Q(s,a) + \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} Q_\pi(s,a) \\
\geq& (1-\epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}|}}{1-\epsilon} Q(s,a) + \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} Q_\pi(s,a) \\
=& \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s,a) = V_\pi(s)
\end{aligned}
$$

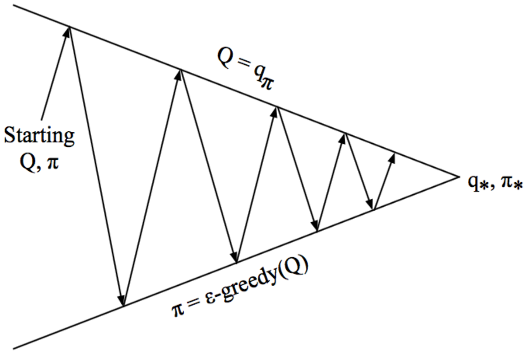
ϵ-Greedy Policy Improvement (Proof)

On the other hand,

$$
\begin{aligned}
Q_\pi(s, \pi^{'}(s)) =& E_{\pi'}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s] \\
\leq& E_{\pi'}[r_{t+1} + \gamma Q_\pi(s_{t+1}, \pi^{'}(s_{t+1})) | s_t = s] \\
=& E_{\pi'}[r_{t+1} + \gamma E_{\pi'}[r_{t+2} + \gamma V_\pi(s_{t+2})] | s_t = s] \\
=& E_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V_\pi(s_{t+2}) | s_t = s] \\
\leq& E_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 Q_\pi(s_{t+2}, \pi^{'}(s_{t+2})) | s_t = s] \\
& \vdots \\
\leq& E_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... | s_t = s] \\
=& V_{\pi'}(s)
\end{aligned}
$$

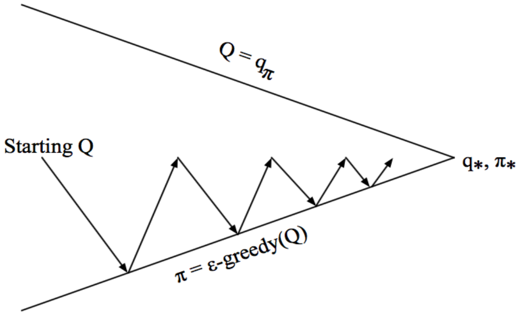
Thus, $V_{\pi'}(s) \geq V_\pi(s)$.

Monte-Carlo Policy Iteration



- Policy evaluation: Monte-Carlo
- Policy improvement: ϵ-greedy policy
- TD Policy iteration is similar

Monte-Carlo Control



For each episode:

- Policy evaluation: Monte-Carlo
- Policy improvement: ϵ-greedy policy

GLIE

Definition

A sequence of policy $\{\pi_k\}_{k=1}^{\infty}$ is called GLIE (Greedy in the limit with infinite exploration), if:

- all state-action pairs are explored infinitely many times,

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

- the policy converges on a greedy policy

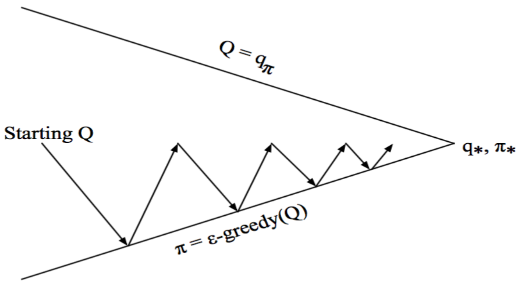
$$\lim_{k\to\infty} \pi_k(a'|s) = 1(a' = arg\max_a Q_k(s,a))$$

- For example: ϵ-greedy policy is GLIE if ϵ reduce to zero as $\epsilon_k = \frac{1}{k}$

Theroem

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

TD control–Sarsa



For each time-step:

- apply TD to evaluate $Q_\pi(s, a)$, then use ϵ-greedy policy improvement

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r(s, a, s') + \gamma Q(s', a') - Q(s, a))$$

- action a' is chosed from s' using policy derived from Q (eg, ϵ-greedy)

Convergence of Sarsa

Theroem

Sarsa converges to the optimal action-value function, i.e $Q(s,a) \rightarrow q^*(s,a)$, under the following conditions:

- GLIE sequence of policies $\pi_k(a|s)$
- Robbins-Monro sequence of step-size α_k

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \ \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

Off-policy learning

- we have assumed that the episode is generated following the learning policy π

  on policy learning
  that is the learning policy and behavior policy is coincident

- consider following behavior policy $\mu(a|s)$

  $\{s_1, a_1, s_2, a_2, ..., s_T\} \sim \mu$
  importance sampling
  off-policy learning

- why is this important?

  Re-use experience generated from old policies
  learn about optimal policy while following exploratory policy
  learn about multiple policies while following one policy

Importance sampling

- To estimate the expectation $E_{X\sim P}[f(X)]$, we estimate a different distribution instead

$$
\begin{aligned}
E_{X\sim P}[f(X)] &= \sum P(X)f(X) \\
&= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\
&= E_{X\sim Q}[\frac{P(X)}{Q(X)}f(X)]
\end{aligned}
$$

- Q is some simple or known distribution

Off-policy version of MC and TD

- MC

  use rewards generated from μ to evaluate π

$$G_t^{\pi/\mu} = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \frac{\pi(a_{t+1}|s_{t+1})}{\mu(a_{t+1}|s_{t+1})} \dots \frac{\pi(a_T|s_T)}{\mu(a_T|s_T)} G_t$$
$$V(s) \leftarrow V(s_t) + \alpha(G_t^{\pi/\mu} - V(s_t))$$

- TD

  use rewards generated from μ to evaluate π

$$V(s) \leftarrow V(s_t) + \alpha(\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}(r(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1})) - V(s_t))$$

  much lower variance than MC importance sampling

Q-learning

- Off-policy learning of action-values $Q(s,a)$

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

- Why Q-learning is considered Off-policy?

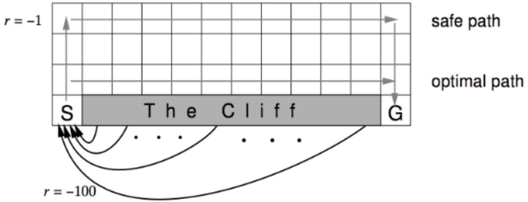
  the learned Q directly approximates q_*
  while behavior policy π may not optimal

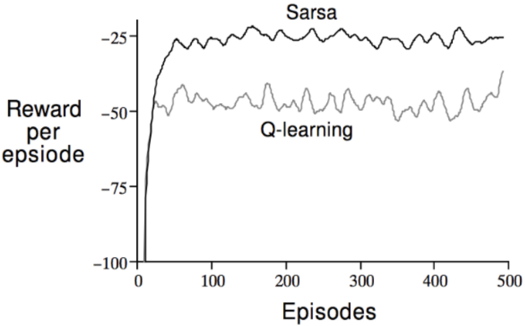
- Comparing with Sarsa

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r(s,a,s') + \gamma Q(s',a') - Q(s,a))$$

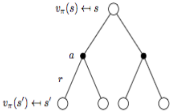
- Sarsa prefers to learn carefully in an environment where exploration is costly, while Q-learning not

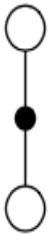
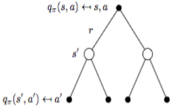
Q-learning V.S Sarsa

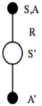


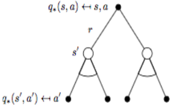


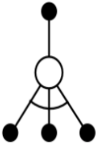
Summary

<table>
<tr><th></th><th>Full Backup (DP)</th><th>Sample Backup (TD)</th></tr>
<tr><td>Bellman Expectation Equation for $v_\pi(s)$</td><td>

Iterative Policy Evaluation</td><td>
TD Learning</td></tr>
<tr><td>Bellman Expectation Equation for $q_\pi(s,a)$</td><td>

Q-Policy Iteration</td><td>

Sarsa</td></tr>
<tr><td>Bellman Optimality Equation for $q_*(s,a)$</td><td>

Q-Value Iteration</td><td>
Q-Learning</td></tr>
</table>

Summary

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

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

Value function approximation

- So far, we have represented value function by a lookup table

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

- For large scale MDPs

  two many state s (or state-action pair s, a) to store
  too slow to learn the value individually

- Estimate value function with function approximation

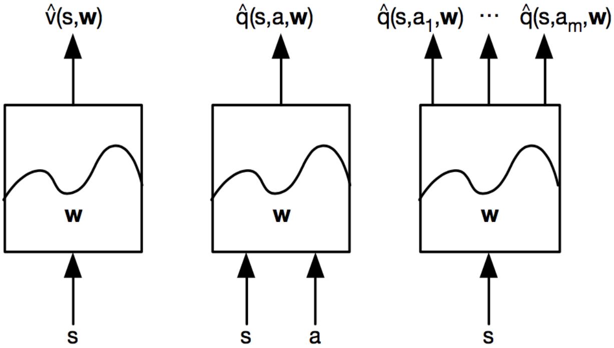
$$\hat{V}(s, w) \approx V_\pi(s)$$
$$\hat{Q}(s, a, w) \approx Q_\pi(s, a)$$

  generalize from seen states to unseen states
  update parameter w by MC or TD learning

Types of function approximation

- Function approximators, e.g

  Linear combinations of features, Neural Network, Decision tree, Nearest neighbor, Fourier/wavelet bases...



Basic idea

- Goal: find parameter vector w minimizing mean-square error between approximate value $\hat{V}(s, w)$ and true value $V_\pi(s)$

$$J(w) = E_\pi[(V_\pi(s) - \hat{V}(s, w))^2]$$

- Gradient descent (α is stepsize)

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w) = \alpha E_\pi[(V_\pi(s) - \hat{V}(s, w))\nabla_w \hat{V}(s, w)]$$

- Stochastic gradient descent

$$\Delta w = \alpha(V_\pi(s) - \hat{V}(s, w))\nabla_w \hat{V}(s, w)$$

Linear case

- $\hat{V}(s, w) = x(s)^T w = \sum_{i=1}^{N} x_i(s) w_i$, where $x(s)$ is a feature vector
- Objective function is quadratic in w

$$J(w) = E_\pi[(V_\pi(s) - x(s)^T w)^2]$$

- Update is simple

$$\Delta w = \alpha(V_\pi(s) - \hat{V}(s, w))x(s)$$

Practical algorithm

- The update requires true value function
- But in RL, it is unavailable, we only have rewards from experiences
- In practice, we substitute a target for $V_\pi(s)$, e.g

  for MC, the target is G_t
  for TD(0), the target is $r(s, a, s') + \gamma \hat{V}(s', w)$
  for TD(λ), the target is the λ-return G_t^λ
- Similarly, we can generate approximators of action-value function in the same way
- Proximate value function methods suffer from a lack of strong theoretical performance guarantees[1]

[1] Sham Kakade, John Langford. Approximately Optimal Approximate Reinforcement Learning

Batch methods

- Gradient descent is simple and appealing, but the sample is not efficient
- Experience $\mathcal{D}$ consisting of $< state, value >$ pairs

$$\mathcal{D} = \{< s_1, V_1^\pi >, < s_2, V_2^\pi >, ..., < s_T, V_T^\pi >\}$$

- Minimising sum-squared error

$$\begin{aligned} LS(w) &= \sum_{t=1}^{T} (V_t^\pi - \hat{V}(s_t, w))^2 \\ &= E_{\mathcal{D}}((V^\pi - \hat{V}(s, w))^2) \end{aligned}$$

SGD with experience replay

- Given experience $\mathcal{D}$ consisting of $< state, value >$ pairs

$$\mathcal{D} = \{< s_1, V_1^\pi >, < s_2, V_2^\pi >, ..., < s_T, V_T^\pi >\}$$

- Repeat:

  1. sample state, value from experience: $< s, V^\pi > \sim \mathcal{D}$
  2. apply SGD update: $\Delta w = \alpha(V^\pi - \hat{V}(s, w))\nabla_w \hat{V}(s, w)$

- Converges to least square solution

$$w^\pi = arg \min_w LS(w)$$

- In practice, we use noisy or biased samples of V_t^π, e.g.

  LSMC (Least Square Monte-Carlo)
  use return $G_t \approx V_t^\pi$

Policy Gradient

- So far, we are discussing value-based RL

  learn value function
  implicit policy (e.g. ϵ-greedy)

- Now, consider parametrize the policy

$$\pi_\theta(a|s) = \mathbb{P}(a|s, \theta)$$

- Policy-based RL

  no value function
  learn policy

Policy-based RL

- Advantages:

  Better convergence properties
  effective in high-dimensional or continuous action spaces
  can learn stochastic policy

- Disadvantages:

  Typically converge to a local rather than global optimum
  evaluating a policy is typically inefficient and high variance

Measure the equality of a policy π_θ

- In episodic environments:

  s_1 is the start state in an episodic

$$J_1(\theta) = V_{\pi_\theta}(s_1)$$

- In continuing environments:

$$J_{avV}(\theta) = \sum_s p_{\pi_\theta}(s) V_{\pi_\theta}(s)$$

$$or,\ J_{avR}(\theta) = \sum_s p_{\pi_\theta}(s) \sum_a \pi_\theta(a|s) r_s^a$$

- $p_{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

$$p_{\pi_\theta}(s) = \mathbb{P}(s_1 = s) + \gamma \mathbb{P}(s_2 = s) + \gamma^2 \mathbb{P}(s_3 = s) + \ldots$$

Policy Gradient

- Goal: find θ to maximize $J(\theta)$ by ascending the gradient of the policy, w.r.t θ

$$\Delta\theta = \alpha\nabla_\theta J(\theta),\ where\ \nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\theta_N} \end{pmatrix}$$

- How to compute gradient?
- Perturbing θ by small amount ϵ in k^{th} dimension, $\forall k \in [1, N]$

$$\frac{\partial J(\theta)}{\theta_k} \approx \frac{J(\theta + \epsilon e_k) - J(\theta)}{\epsilon}$$

- Simple, but noisy, inefficient in most cases

Score function

- Assume policy π_θ is differentiable whenever it is non-zero
- Likelihood ratios exploit the following identity

$$\nabla_\theta \pi_\theta(a|s) = \pi_\theta(a|s)\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} = \pi_\theta(a|s)\nabla_\theta \log \pi_\theta(a|s)$$

- The score function is $\nabla_\theta \log \pi_\theta(a|s)$
- For example, consider a Gaussian policy: $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- Mean is linear combination of state features $\mu(s) = \phi(s)^T\theta$
- The score function is

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

One-step MDPs

- Consider one-step MDPs

  start with $s \sim p(s)$, and terminate after one step with reward r_s^a

$$
\begin{aligned}
J(\theta) &= \sum_s p(s) \sum_a \pi_\theta(a|s) r_s^a, \\
\nabla_\theta J(\theta) &= \sum_s p(s) \sum_a \nabla \pi_\theta(a|s) r_s^a \\
&= \sum_s p(s) \sum_a \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) r_s^a \\
&= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) r_s^a]
\end{aligned}
$$

Policy Gradient Theorem

- Consider the multi-step MDPs, we can use likelihood ratio to obtain the similar conclusion:

Theorem

For any differentiable policy $\pi_\theta(a|s)$, and for any of the policy objective function $J = J_1, J_{avR}$ or $\frac{1}{1-\gamma}J_{avV}$, the policy gradient for policy objective function $J(\theta)$ is

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_{\pi_\theta}(s, a)]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Apply stochastic gradient ascent
- Experience an episode

$$\{s_1, a_1, r_2, s_2, a_2, r_3, ..., s_{T-1}, a_{T-1}, r_T, s_T\} \sim \pi_\theta$$

- Use return v_t as unbiased sample of $Q_{\pi_\theta}(s_t, a_t)$, $t = 1, .., T-1$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(a|s) v_t$$

- High variance

Actor-Critic

- Reduce the variance
- Use a critic to estimate the action-value function

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

- Actor-Critic algorithms maintains two set of parameters

  Critic: update action-value function parameters w
  Actor: update policy parameters θ in direction suggested by Critic

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_w(s, a)]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(a|s) Q_w(s, a)$$

- Can we avoid any bias by choosing action-value function approximation carefully?

Compatible Function Approximation Theorem

Theorem

If the action-value function approximator $Q_w(s,a)$ satisfies the following two conditions:

$$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(a|s),$$
$$w = arg\min_{w'} E_{\pi_\theta}[(Q_{\pi_\theta}(s,a) - Q_{w'}(s,a))^2]$$

then the policy gradient is exact, i.e.

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_w(s,a)]$$

- The function approximator is compatible with the policy in the sense that if we use the approximations $Q_w(s,a)$ in lieu of their true values to compute the gradient, then the result would be exact

Proof

Pf: Denote $\epsilon = E_{\pi_\theta}[(Q_{\pi_\theta}(s,a) - Q_w(s,a))^2]$, then

$$\begin{aligned}
\nabla_w \epsilon =& 0 \\
E_{\pi_\theta}[(Q_{\pi_\theta}(s,a) - Q_w(s,a))\nabla_w Q_w(s,a)] =& 0 \\
E_{\pi_\theta}[(Q_{\pi_\theta}(s,a) - Q_w(s,a))\nabla_\theta \log \pi_\theta(a|s)] =& 0 \\
E_{\pi_\theta}[Q_{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)] =& E_{\pi_\theta}[Q_w(s,a)\nabla_\theta \log \pi_\theta(a|s)].
\end{aligned}$$

Baseline

- Reduce the variance
- A baseline function $B(s)$

$$
\begin{aligned}
& E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a,a)B(s)] \\
& = \sum_s p_{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(a|s)B(s) \\
& = \sum_s p_{\pi_\theta}(s)B(s)\nabla_\theta \sum_a \pi_\theta(a|s) = 0
\end{aligned}
$$

- A good baseline $B(s) = V_{\pi_\theta}(s)$
- Advantage function

$$
A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)
$$

Policy gradient

$$
\begin{aligned}
\nabla_\theta J(\theta) =& E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_{\pi_\theta}(s,a)] \\
=& E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A_{\pi_\theta}(s,a)] \\
\Delta\theta =& \alpha \nabla_\theta \log \pi_\theta(a|s) A_{\pi_\theta}(s,a)
\end{aligned}
$$

- The advantage function can significantly reduce variance of policy gradient
- Estimate both $V_{\pi_\theta}(s)$ and $Q_{\pi_\theta}(s,a)$ to obtain $A_{\pi_\theta}(s,a)$
- e.g. TD learning

Estimation of advantage function

- Apply TD learning to estimate value function
- TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r(s, a, s') + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)$$

- is an unbiased estimate of advantage function

$$\begin{aligned} E_{\pi_\theta}[\delta^{\pi_\theta}|s, a] =& E_{\pi_\theta}[r(s, a, s') + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)|s, a] \\ =& E_{\pi_\theta}[r(s, a, s') + \gamma V_{\pi_\theta}(s')|s, a] - V_{\pi_\theta}(s) \\ =& Q_{\pi_\theta}(s, a) - V_{\pi_\theta}(s) = A_{\pi_\theta}(s, a) \end{aligned}$$

- thus the update

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(a|s) \delta^{\pi_\theta}$$

Natural Policy Gradient [2]

- Consider

$$\max_{d\theta} J(\theta + d\theta)$$
$$s.t. \ \|d\theta\|_{G_\theta} = a$$

where a is a small constant, and $\|d\theta\|^2_{G_\theta} = (d\theta)^T G_\theta (d\theta)$.

- The optimal

$$(d\theta)^* := \nabla^{nat} J(\theta) = G_\theta^{-1} \nabla_\theta J(\theta)$$

- Take the metric matrix G_θ as Fisher information matrix

$$G_\theta = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^T]$$

[2] Sham Kakade. A Natural Policy Gradient, In Advance in Neural Information Processing Systems, pp. 1057-1063. MIT press, 2002.

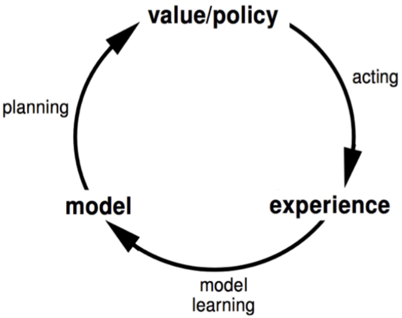
Natural Actor-Critic

- Use compatible function function $Q_w(s,a) = \nabla_\theta \log \pi_\theta(a|s)^T w$
- $w = arg\,\min_{w'} E_{\pi_\theta}[(Q_{\pi_\theta}(s,a) - Q_{w'}(s,a))^2]$
- then

$$\begin{aligned}\nabla_\theta J(\theta) =& E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_w(s,a)] \\ =& E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^T w] \\ =& E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^T] w \\ =& G_\theta w\end{aligned}$$

- so, $\nabla^{nat} J(\theta) = G_\theta^{-1} \nabla_\theta J(\theta) = w$
- i.e. the natural policy gradient update Actor parameters θ in direction of Critic parameters w

Model-based RL



- Learn a model directly from experience
- Use planning to construct a value function or policy

Learn a model

- For an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$
- A model $\mathcal{M} = < \mathcal{P}_\eta, \mathcal{R}_\eta >$, parameterized by η
- $\mathcal{P}_\eta \approx \mathcal{P}$, $\mathcal{R}_\eta \approx \mathcal{R}$

$$s' \sim \mathcal{P}_\eta(s'|s,a), \; r(s,a,s') = \mathcal{R}_\eta(s,a,s')$$

- Typically assume conditional independence between state transitions and rewards

$$P(s', r(s,a,s')|s,a) = P(s'|s,a)P(r(s,a,s')|s,a)$$

Learn a model

- Experience $s_1, a_1, r_2, s_2, a_2, r_3, ...s_T$

$$
\begin{aligned}
s_1, a_1 &\rightarrow r_2, s_2 \\
s_2, a_2 &\rightarrow r_3, s_3 \\
&\vdots \\
s_{T-1}, a_{T-1} &\rightarrow r_T, s_T
\end{aligned}
$$

- Learning $s, a \rightarrow r$ is a regression problem
- Learning $s, a \rightarrow s'$ is a density estimation problem
- find parameters η to minimise empirical loss

Example: Table Lookup Model

- count visits $N(s,a)$ to each pair action pair

$$\hat{P}(s'|s,a) = \frac{1}{N(s,a)} \sum_{t=1}^{T} 1(s_t = s, a_t = a, s_{t+1} = s')$$

$$\hat{r}_s^a = \frac{1}{N(s,a)} \sum_{t=1}^{T} 1(s_t = s, a_t = a) r_t$$

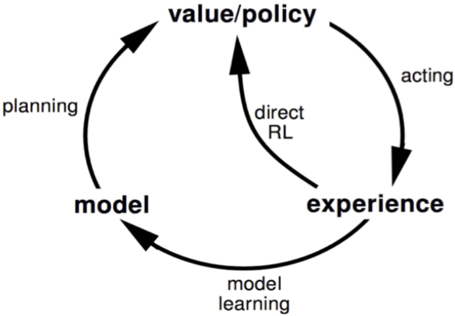
Planning with a model

- After estimating a model, we can plan with algorithms introduced before
- Dynamic Programming

  Policy iteration
  Value iteration...
- Model-free RL

  Monte-Carlo
  Sarsa
  Q-learning...
- Performance of model-based RL is limited to optimal policy for approximate MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta)$
- when the estimated model is imperfect, model-free RL is more efficient

Dyna Architecture



- Learn a model from real experience (true MDP)
- Learn and plan value function (and/or policy) from real and simulated experience