Notes for Bolei Zhou's RL Course

Yanjie Ze, 2021

Lecture 2 MDP

Bellman Expectation Backup

Bellman expectation backup for a particular policy

$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_t(s')) \tag{15}$$

Or if in the form of MRP $< \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}, \gamma >$

$$v_{t+1}(s) = R^\pi(s) + \gamma P^\pi(s'|s) v_t(s') \tag{16}$$

Bellman Optimality Function

1. The optimal value functions are reached by the Bellman optimality equations:

$$v^*(s) = \max_a q^*(s,a)$$
$$q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^*(s')$$

thus

$$v^*(s) = \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^*(s')$$
$$q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} q^*(s',a')$$

Value Iteration

Algorithm of Value Iteration

1. Objective: find the optimal policy π
2. Solution: iteration on the Bellman optimality backup
3. Value Iteration algorithm:
   1. initialize $k = 1$ and $v_0(s) = 0$ for all states s
   2. For $k = 1 : H$
      1. for each state s

$$q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_k(s') \quad (22)$$
$$v_{k+1}(s) = \max_a q_{k+1}(s,a) \quad (23)$$

      2. $k \leftarrow k + 1$
   3. To retrieve the optimal policy after the value iteration:

$$\pi(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{k+1}(s') \quad (24)$$

Policy Iteration

2. Policy iteration: Given a known MDP, compute the optimal policy and the optimal value function
   1. Policy evaluation: iteration on the Bellman expectation backup

$$v_i(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_{i-1}(s'))$$

   2. Policy improvement: greedy on action-value function q

$$q_{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{\pi_i}(s')$$

$$\pi_{i+1}(s) = \arg\max_a q_{\pi_i}(s,a)$$

Difference between PI and VI

Difference between Policy Iteration and Value Iteration

1. Policy iteration includes: **policy evaluation** + **policy improvement**, and the two are repeated iteratively until policy converges.
2. Value iteration includes: **finding optimal value function** + **one policy extraction**. There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged).
3. Finding optimal value function can also be seen as a combination of policy improvement (due to max) and truncated policy evaluation (the reassignment of v(s) after just one sweep of all states regardless of convergence).

Summary for prediction and control

Table: Dynamic Programming Algorithms

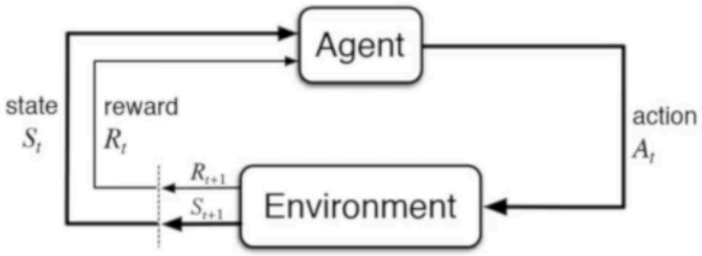
Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Lecture 3 Model-free prediction and control

Model-free RL

Model-free RL: Learning by interaction

1. Model-free RL can solve the problems through interaction with the environment



2. No more direct access to the known transition dynamics and reward function
3. Trajectories/episodes are collected by the agent's interaction with the environment
4. Each trajectory/episode contains $\{S_1, A_1, R_1, S_2, A_2, R_2, ..., S_T, A_T, R_T\}$

Monte-Carlo Policy Evaluation

Monte-Carlo Policy Evaluation

1. Return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$ under policy π
2. $v^\pi(s) = \mathbb{E}_{\tau \sim \pi}[G_t | s_t = s]$, thus expectation over trajectories τ generated by following π
3. MC simulation: we can simply sample a lot of trajectories, compute the actual returns for all the trajectories, then average them
4. MC policy evaluation uses empirical mean return instead of expected return
5. MC does not require MDP dynamics/rewards, no bootstrapping, and does not assume state is Markov.
6. Only applied to episodic MDPs (each episode terminates)

Incremental Mean

Mean from the average of samples $x_1, x_2, \ldots$

$$
\begin{aligned}
\mu_t &= \frac{1}{t} \sum_{j=1}^{t} x_j \\
&= \frac{1}{t} \left(x_t + \sum_{j=1}^{t-1} x_j \right) \\
&= \frac{1}{t} (x_t + (t-1)\mu_{t-1}) \\
&= \mu_{t-1} + \frac{1}{t} (x_t - \mu_{t-1})
\end{aligned}
$$

Advantages of MC over DP

1. MC works when the environment is unknown
2. Working with sample episodes has a huge advantage, even when one has complete knowledge of the environment's dynamics, for example, transition probability is complex to compute
3. Cost of estimating a single state's value is independent of the total number of states. So you can sample episodes starting from the states of interest then average returns

Temporal Difference Learning

Temporal-Difference (TD) Learning

1. Objective: learn v_π online from experience under policy π
2. Simplest TD algorithm: TD(0)
   1. Update $v(S_t)$ toward estimated return $R_{t+1} + \gamma v(S_{t+1})$

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

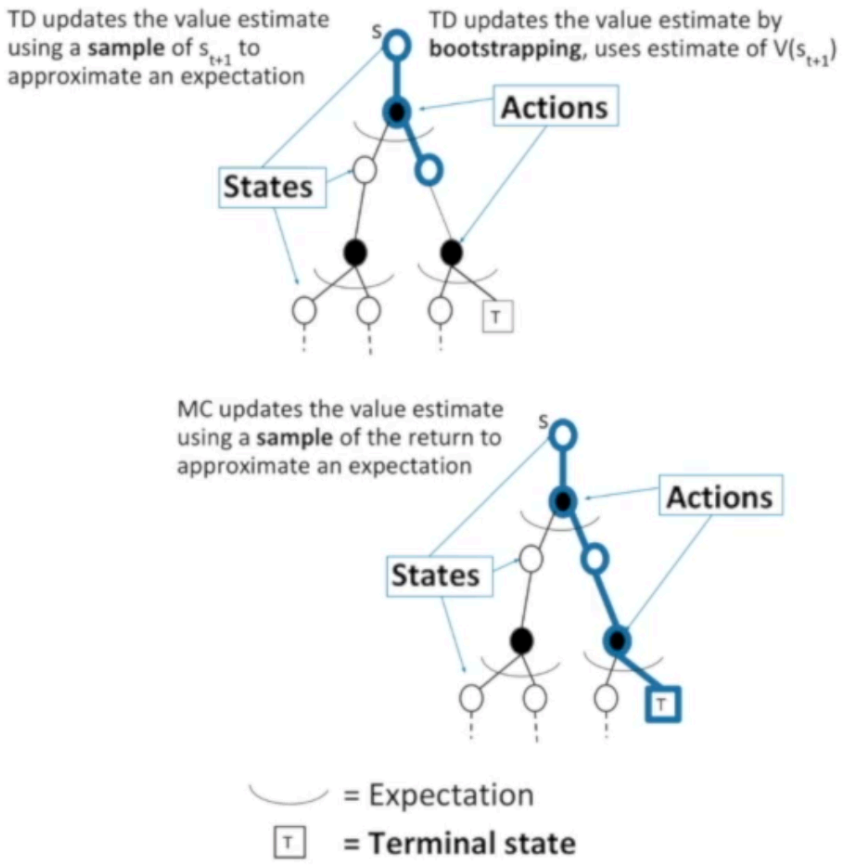
3. $R_{t+1} + \gamma v(S_{t+1})$ is called TD target
4. $\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ is called the TD error
5. Comparison: Incremental Monte-Carlo
   1. Update $v(S_t)$ toward actual return G_t given an episode i

$$v(S_t) \leftarrow v(S_t) + \alpha(G_{i,t} - v(S_t))$$

与MC相比，TD不需要跑完一个episode就可以更新。

Advantages of TD over MC

Advantages of TD over MC



Comparison of TD and MC

Comparison of TD and MC

1. TD can learn online after every step
2. MC must wait until end of episode before return is known

3. TD can learn from incomplete sequences
4. MC can only learn from complete sequences

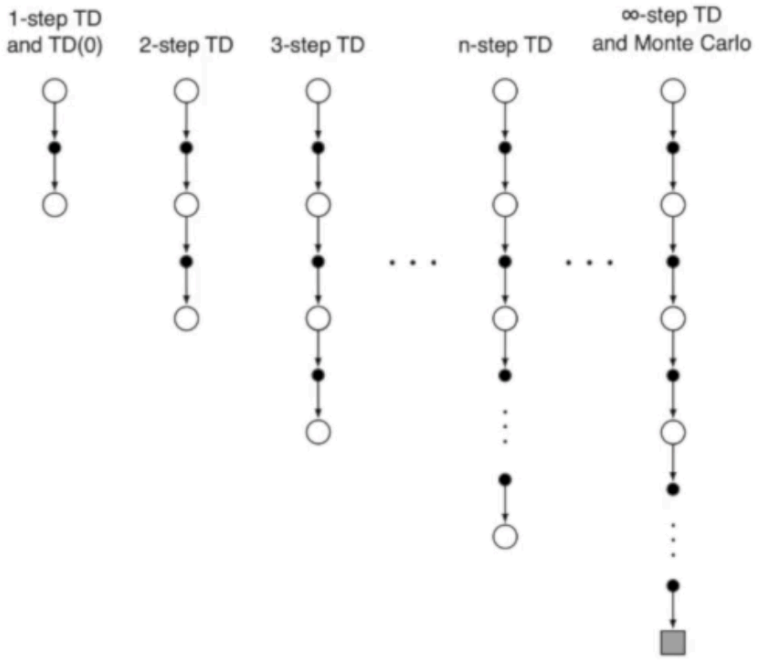
5. TD works in continuing (non-terminating) environments
6. MC only works for episodic (terminating) environments

7. TD exploits Markov property, more efficient in Markov environments
8. MC does not exploit Markov property, more effective in non-Markov environments

n-step TD

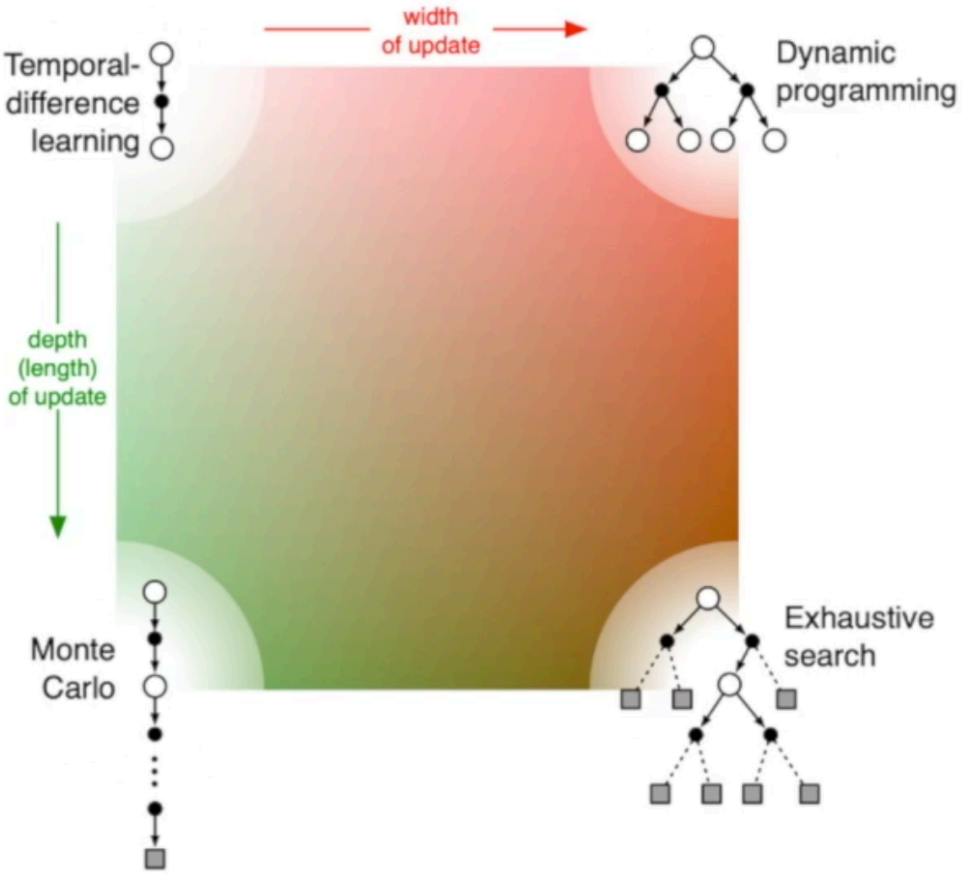
n-step TD

1. *n*-step TD methods that generalize both one-step TD and MC.
2. We can shift from one to the other smoothly as needed to meet the demands of a particular task.



各种算法的总结

Unified View of Reinforcement Learning



Monte Carlo with epsilon greedy

Monte Carlo with ϵ-Greedy Exploration

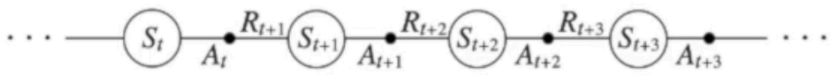
Algorithm 1

Initialize $Q(S, A) = 0, N(S, A) = 0, \epsilon = 1, k = 1$
$\pi_k = \epsilon$-greedy(Q)
loop
  Sample k-th episode $(S_1, A_1, R_2, ..., S_T) \sim \pi_k$
  for each state S_t and action A_t in the episode **do**
    $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
  end for
  $k \leftarrow k + 1, \epsilon \leftarrow 1/k$
  $\pi_k = \epsilon$-greedy(Q)
end loop

Sarsa

Sarsa: On-Policy TD Control

1. An episode consists of an alternating sequence of states and state–action pairs:



2. ϵ-greedy policy for one step, then bootstrap the action value function:

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

3. The update is done after every transition from a nonterminal state S_t
4. TD target $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

n-step Sarsa

n-step Sarsa

1. Consider the following *n*-step Q-returns for $n = 1, 2, \infty$

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

2. Thus the n-step Q-return is defined as

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

3. *n*-step Sarsa updates Q(s,a) towards the n-step Q-return:

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

on-policy vs off-policy

On-policy Learning vs. Off-policy Learning

1. On-policy learning: Learn about policy π from the experience collected from π
   1. Behave non-optimally in order to explore all actions, then reduce the exploration. e.g., ϵ-greedy
2. Another important approach is **off-policy learning** which essentially uses **two different polices**:
   1. the one which is being learned about and becomes the optimal policy
   2. the other one which is more exploratory and is used to generate trajectories
3. Off-policy learning: Learn about policy π from the experience sampled from another policy μ
   1. π: target policy
   2. μ: behavior policy

off-policy learning

Off-policy Learning

1. Following behaviour policy $\mu(a|s)$ to collect data

$$S_1, A_1, R_2, ..., S_T \sim \mu$$

Update π using $S_1, A_1, R_2, ..., S_T$

2. It leads to many benefits:
   1. Learn about optimal policy while following exploratory policy
   2. Learn from observing humans or other agents
   3. Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$

Comparison of Sarsa and Q-learning

Comparison of Sarsa and Q-Learning

1. Sarsa: On-Policy TD control

   Choose action A_t from S_t using policy derived from Q with ϵ-greedy
   Take action A_t, observe R_{t+1} and S_{t+1}
   Choose action A_{t+1} from S_{t+1} using policy derived from Q with ϵ-greedy
   $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

2. Q-Learning: Off-Policy TD control

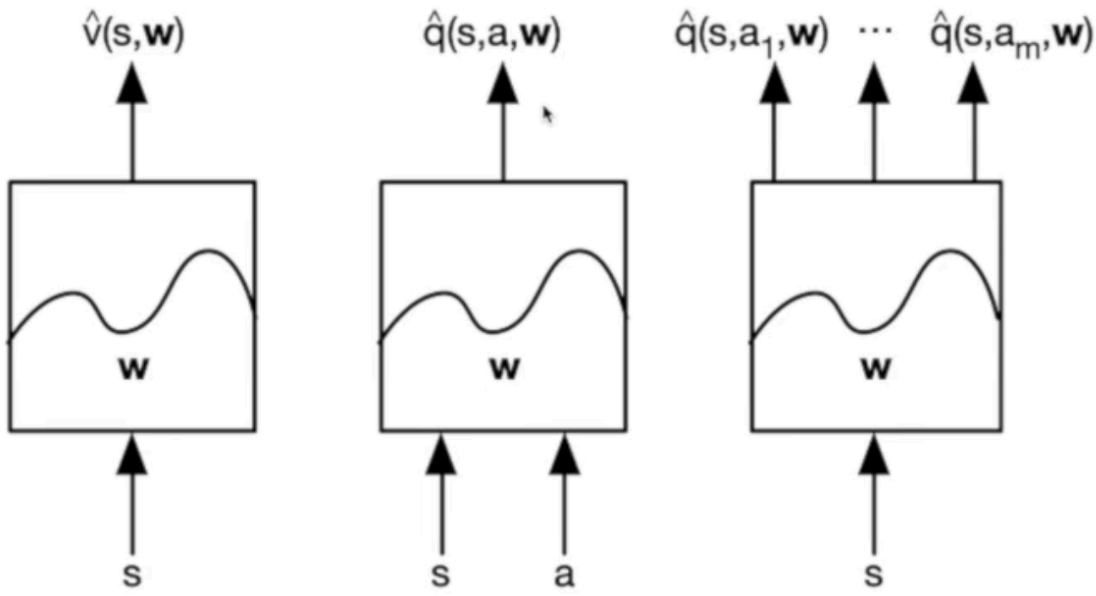
   Choose action A_t from S_t using policy derived from Q with ϵ-greedy
   Take action A_t, observe R_{t+1} and S_{t+1}
   Then 'imagine' A_{t+1} as $\arg\max Q(S_{t+1}, a')$ in the update target
   $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Lecture 4 价值函数近似

Types of Value function approximation

Types of value function approximation

Several function designs:



Linear Value Function Approximation

Linear Value Function Approximation

1. Represent value function by a linear combination of features

$$\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w} = \sum_{j=1}^{n} x_j(s) w_j$$

2. The objective function is quadratic in parameter $\mathbf{w}$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v^\pi(s) - \mathbf{x}(s)^T \mathbf{w})^2 \right]$$

3. Thus the update rule is as simple as

$$\Delta \mathbf{w} = \alpha (v^\pi(s) - \hat{v}(s, \mathbf{w})) \mathbf{x}(s)$$

Update = *StepSize* × *PredictionError* × *FeatureValue*

4. Stochastic gradient descent converges to global optimum. Because in the linear case, there is only one optimum, thus local optimum is automatically converge to or near the global optimum.

Incremental VFA Prediction Algorithms

Incremental VFA Prediction Algorithms

1. We assumed that true value function $v^{\pi}(s)$ given by supervisor/oracle

$$\Delta \mathbf{w} = \alpha\Big(v^{\pi}(s) - \hat{v}(s, \mathbf{w})\Big)\nabla_{\mathbf{w}}\hat{v}(s_t, \mathbf{w})$$

2. But in RL there is no supervisor, only rewards
3. In practice, we substitute the target for $v^{\pi}(s)$
   1. For MC, the target is the actual return G_t

$$\Delta \mathbf{w} = \alpha\Big(G_t - \hat{v}(s_t, \mathbf{w})\Big)\nabla_{\mathbf{w}}\hat{v}(s_t, \mathbf{w})$$

   2. For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{v}(s_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha\Big(R_{t+1} + \gamma\hat{v}(s_{t+1}, \mathbf{w}) - \hat{v}(s_t, \mathbf{w})\Big)\nabla_{\mathbf{w}}\hat{v}(s_t, \mathbf{w})$$

Monte Carlo Prediction with VFA

Monte-Carlo Prediction with VFA

1. Return G_t is an unbiased, but noisy sample of true value $v^{\pi}(s_t)$
2. Why unbiased? $\mathbb{E}[G_t] = v^{\pi}(s_t)$
3. So we have the training data that can be used for supervised learning in VFA:

$$< S_1, G_1 >, < S_2, G_2 >, ..., < s_t, G_T >$$

4. Using linear Monte-Carlo policy evaluation

$$\begin{aligned}\Delta \mathbf{w} &= \alpha\Big(G_t - \hat{v}(s_t, \mathbf{w})\Big)\nabla_{\mathbf{w}}\hat{v}(s_t, \mathbf{w}) \\ &= \alpha\Big(G_t - \hat{v}(s_t, \mathbf{w})\Big)\mathbf{x}(s_t)\end{aligned}$$

5. Monte-Carlo prediction converges, in both linear and non-linear value function approximation.

TD Prediction with VFA

TD Prediction with VFA

1. TD target $R_{t+1} + \gamma \hat{v}(s_{t+1}, \mathbf{w})$ is a biased sample of true value $v^{\pi}(s_t)$
2. Why biased? It is drawn from our previous estimate, rather than the true value: $\mathbb{E}[R_{t+1} + \gamma \hat{v}(s_{t+1}, \mathbf{w})] \neq v^{\pi}(s_t)$
3. We have the training data used for supervised learning in VFA:

$$< S_1, R_2 + \gamma \hat{v}(s_2, \mathbf{w}) >, < S_2, R_3 + \gamma \hat{v}(s_3, \mathbf{w}) >, ..., < S_{T-1}, R_T >$$

4. Using linear TD(0), the stochastic gradient descend update is

$$\begin{aligned} \Delta \mathbf{w} &= \alpha \Big(R + \gamma \hat{v}(s', \mathbf{w}) - \hat{v}(s, \mathbf{w}) \Big) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) \\ &= \alpha \Big(R + \gamma \hat{v}(s', \mathbf{w}) - \hat{v}(s, \mathbf{w}) \Big) \mathbf{x}(s) \end{aligned}$$

   This is also called as semi-gradient, as we ignore the effect of changing the weight vector $\mathbf{w}$ on the target
5. Linear TD(0) converges(close) to global optimum

Incremental Control Algorithm

Incremental Control Algorithm

Same to the prediction, there is no oracle for the true value $q^{\pi}(s, a)$, so we substitute a target

1. For MC, the target is return G_t

$$\Delta \mathbf{w} = \alpha\big(G_t - \hat{q}(s_t, a_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\hat{q}(s_t, a_t, \mathbf{w})$$

2. For Sarsa, the target is TD target $R_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha\big(R_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\hat{q}(s_t, a_t, \mathbf{w})$$

3. For Q-learning, the target is TD target $R_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha\big(R_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\hat{q}(s_t, a_t, \mathbf{w})$$

Convergence of VFA

Convergence of Control Methods with VFA

1. For Sarsa,

$$\Delta \mathbf{w} = \alpha\big(R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w})\big)\nabla_{\mathbf{w}} \hat{q}(s_t, a_t, \mathbf{w})$$

2. For Q-learning,

$$\Delta \mathbf{w} = \alpha\big(R_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w})\big)\nabla_{\mathbf{w}} \hat{q}(s_t, a_t, \mathbf{w})$$

3. TD with VFA doesn't follow the gradient of any objective function
4. The updates involve doing an approximate Bellman backup followed by fitting the underlying value function
5. That is why TD can diverge when off-policy or using non-linear function approximation
6. Challenge for off-policy control: behavior policy and target policy are not identical, thus value function approximation can diverge

死亡三角

The Deadly Triad for the Danger of Instability and Divergence

Potential problematic issues:

1. **Function approximation**: A scalable way of generalizing from a state space much larger than the memory and computational resources
2. **Bootstrapping**: Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods)
3. **Off-policy training**: training on a distribution of transitions other than that produced by the target policy
4. See details at Textbook Chapter 11.3

Convergence Summary

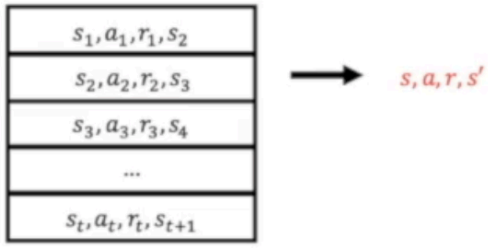
Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	$\surd$	$(\surd)$	X
Sarsa	$\surd$	$(\surd)$	X
Q-Learning	$\surd$	X	X

$(\surd)$ moves around the near-optimal value function

DQN

DQNs: Experience Replay

1. To reduce the correlations among samples, store transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{D}$



2. To perform experience replay, repeat the following
   1. sample an experience tuple from the dataset: $(s, a, r, s') \sim \mathcal{D}$
   2. compute the target value for the sampled tuple: $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w})$
   3. use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha\Big(r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})\Big)\nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

DQNs: Fixed Targets

1. To help improve stability, fix the target weights used in the target calculation for multiple updates
2. Let a different set of parameter $\mathbf{w}^-$ be the set of weights used in the target, and $\mathbf{w}$ be the weights that are being updated
3. To perform experience replay with fixed target, repeat the following
   1. sample an experience tuple from the dataset: $(s, a, r, s') \sim \mathcal{D}$
   2. compute the target value for the sampled tuple: $r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-)$
   3. use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha\Big(r + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w})\Big)\nabla_{\mathbf{w}} \hat{Q}(s, a, \mathbf{w})$$

Value-based RL versus Policy-based RL

Value-based RL versus Policy-based RL

1. Deterministic policy is generated directly from the value function using greedy $a_t = \arg\max_a Q(a, s_t)$
2. Instead we can parameterize the policy function as $\pi_\theta(a|s)$ where θ is the learnable policy parameter and the output is a probability over the action set

Advantages of Policy-based RL

1. Advantages:
   1. better convergence properties: we are guaranteed to converge on a local optimum (worst case) or global optimum (best case)
   2. Policy gradient is more effective in high-dimensional action space
   3. Policy gradient can learn stochastic policies, while value function can't
2. Disadvantages:
   1. typically converges to a local optimum
   2. evaluating a policy has high variance

两种策略函数

1. Deterministic: given a state, the policy returns a certain action to take
2. Stochastic: given a state, the policy returns a probability distribution of the actions (e.g., 40% chance to turn left, 60% chance to turn right) or a certain Gaussian distribution for continuous action

优化目标

Objective of Optimizing Policy

1. Objective: Given a policy approximator $\pi_\theta(s, a)$ with parameter θ, find the best θ
2. How do we measure the quality of a policy π_θ?
3. In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

4. In continuing environments
   1. we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

   2. or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R(s, a)$$

where d^{π_θ} is stationary distribution of Markov chain for π_θ

1. Policy-based RL is an optimization problem that find θ that maximizes $J(\theta)$
2. If $J(\theta)$ is differentiable, we can use gradient-based methods:
   1. gradient ascend
   2. conjugate gradient
   3. quasi-newton
3. If $J(\theta)$ is non-differentiable or hard to compute the derivative, some derivative-free black-box optimization methods:
   1. Cross-entropy method (CEM)
   2. Hill climbing
   3. Evolution algorithm

黑箱优化方法之CEM

Derivative-free Method: Cross-Entropy Method

1. $\theta^* = \arg\max J(\theta)$
2. Treat $J(\theta)$ as a black box score function (not differentiable)

Algorithm 1 CEM for black-box function optimization

for iter $i = 1$ to N **do**
  $\mathcal{C} = \{\}$
  for parameter set $e = 1$ to N **do**
    sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$
    execute roll-outs under $\theta^{(e)}$ to evaluate $J(\theta^{(e)})$
    store $(\theta^e, J(\theta^{(e)}))$ in $\mathcal{C}$
  end for
  $\mu^{(i+1)} = \arg\max_\mu \sum_{k \in \hat{\mathcal{C}}} \log P_\mu(\theta^{(k)})$
  where $\hat{\mathcal{C}}$ are the top 10% of $\mathcal{C}$ ranked by $J(\theta^{(e)})$
end for

3. Example of CEM for a simple RL problem:`https://github.com/cuhkrlcourse/RLexample/blob/master/my_learning_agent.py`

黑箱优化方法之近似梯度

Approximate Gradients by Finite Difference

1. To evaluate policy gradient of $\pi_\theta(s, a)$
2. For each dimension $k \in [1, n]$
   1. estimate *kth* partial derivative of objective function by perturbing θ by a small amount ϵ in *kth* dimension

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

   where u_k is unit vector with 1 in *kth* component, 0 else where
3. uses n evaluations to compute policy graident in total n dimensions
4. though noisy and inefficient, but works for arbitrary policies, even if policy is not differentiable.

梯度的新形式

Computing the Policy Gradient Analytically

1. Assume policy π_θ is differentiable whenever it is no-zero
2. and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$
3. Likelihood ratios exploit the following tricks

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

4. The score function is $\nabla_\theta \log \pi_\theta(s, a)$

Policy例子

Policy Example: Softmax Policy

1. Simple policy model: weight actions using linear combination of features $\phi(s, a)^T \theta$
2. Probability of action is proportional to the exponetiated weight

$$\pi_\theta(s, a) = \frac{\exp^{\phi(s,a)^T \theta}}{\sum_{a'} \exp^{\phi(s,a')^T \theta}}$$

3. The score function is

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

Policy Example: Gaussian Policy

1. In continuous action spaces, a Gaussian policy can be naturally defined
2. Mean is a linear combination of state features $\mu(s) = \phi(s)^T\theta$
3. Variance may be fixed σ^2 or can also be parameterized
4. Policy is Gaussian, the continuous $a \sim \mathcal{N}(\mu(s), \sigma^2)$
5. The score function is

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

Policy Gradient

Policy Gradient for One-Step MDPs

1. Consider a simple class of one-step MDPs
   1. Starting in state $s \sim d(s)$
   2. Terminating after one time-step with reward $r = R(s, a)$
2. Use likelihood ratios to compute the policy gradient

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

3. The gradient is as

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

Policy Gradient for Multi-step MDPs

1. Denote a state-action trajectory from one episode as $\tau = (s_0, a_0, r_1, ...s_{T-1}, a_{T-1}, r_T, s_T) \sim (\pi_\theta, P(s_{t+1}|s_t, a_t))$
2. Denote $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ as the sum of rewards over a trajectory τ
3. The policy objective is

$$J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T} R(s_t, a_t)\Big] = \sum_{\tau} P(\tau;\theta)R(\tau)$$

   where $P(\tau;\theta) = \mu(s_0)\prod_{t=0}^{T-1} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$ denotes the probability over trajectories when executing the policy π_θ
4. Then our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_{\tau} P(\tau;\theta)R(\tau)$$