Lecture 4： Model-free Control & Value Function Approximation

22nd Mar. 2022

Model-free Control

- Model-Free Reinforcement Learning
  - Model-free prediction
  - Estimate the value function of an unknown MDP

- This lecture:
  - Model-free control
    - Monte-Carlo control
    - Temporal Difference (TD) control
    - Off-Policy Learning
  - Optimise the value function of an unknown MDP
  - Solve large RL problems

Uses of Model-Free Control

Some example problems that can be modelled as MDPs

- Elevator
- Parallel Parking
- Ship Steering
- Bioreactor
- Helicopter
- Aeroplane Logistics
- Robocup Soccer
- Quake
- Portfolio management
- Protein Folding
- Robot walking
- Game of Go

For most of these problems, either:

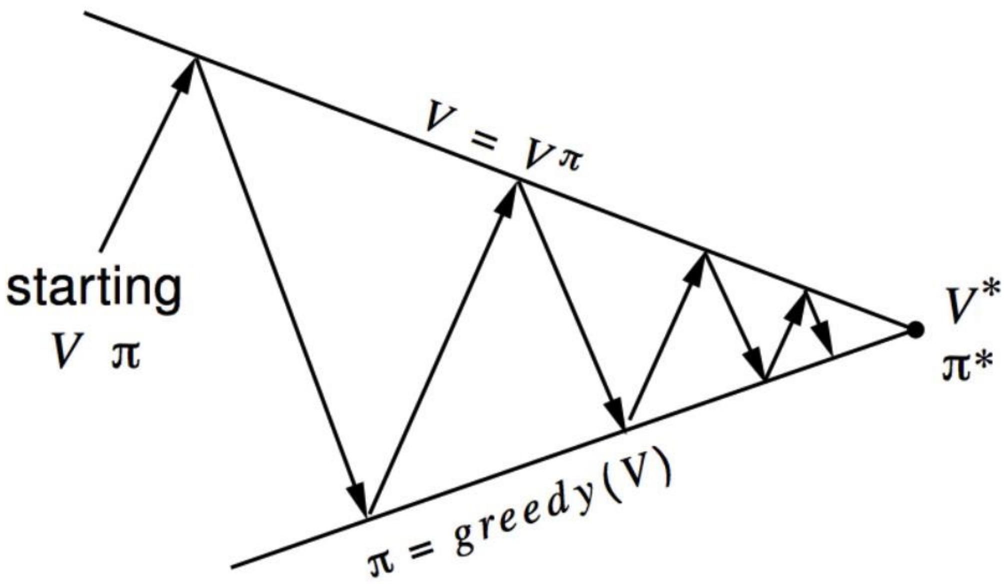
- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

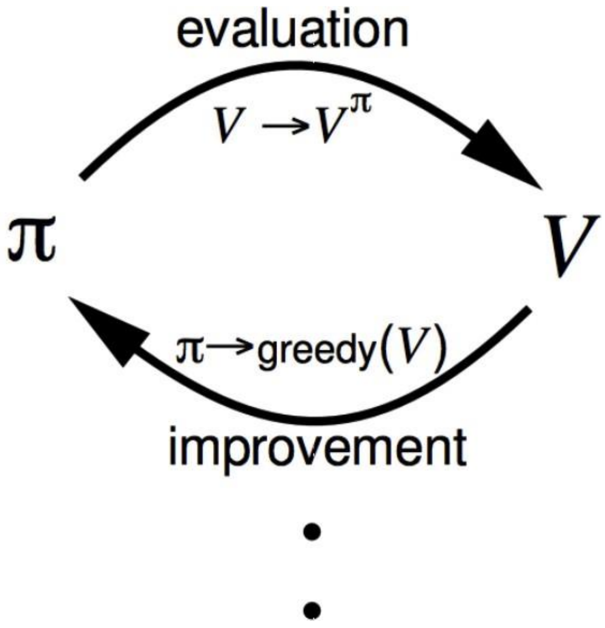
Model-free control can solve these problems

Policy Iteration

- Iteration through the two steps
  - Evaluate the policy π (computing v given current π)
  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$





Policy Iteration for a Known MDP

- Compute the state-action value of a policy π:

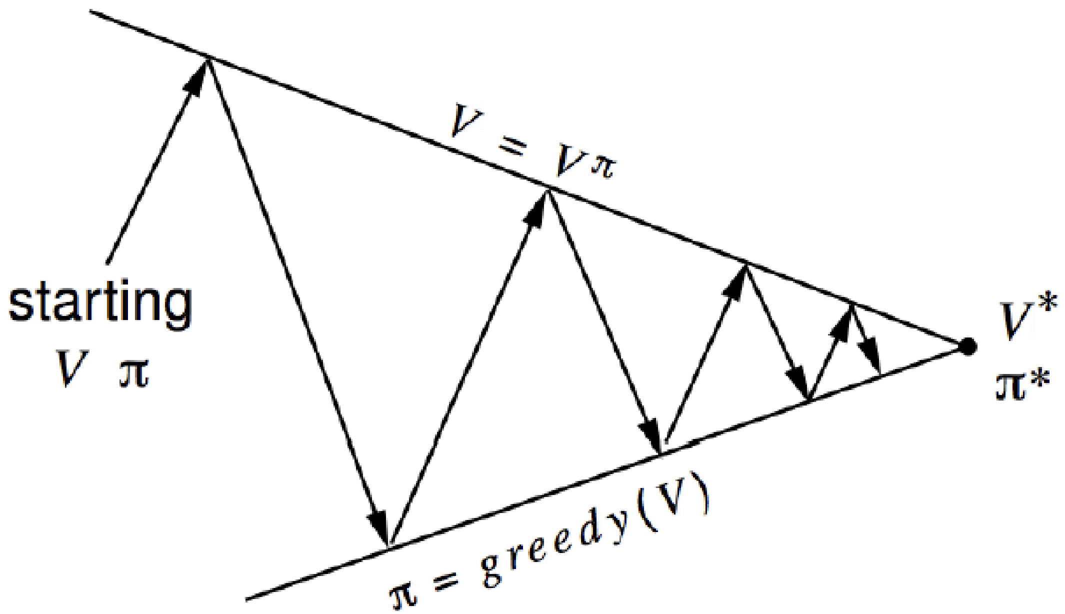
$$q_{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{\pi_i}(s')$$

- Compute new policy π_{i+1} for all $s \in S$ following

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

- Problem： what to do if there is neither $R(s,a)$ nor $P(s'|s,a)$ known/available

General PI With Monte-Carlo Evaluation



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

Using Action-Value Function

- Greedy policy improvement over $V(s)$ requires model of MDP

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

- Greedy policy improvement over $Q(s, a)$ is model-free

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

Monte Carlo with $\epsilon - Greedy$ Exploration

- $\epsilon - greedy$ Exploration: Ensuring continual exploration
  - All actions are tried with non-zero probability
  - With probability $1 - \epsilon$ choose the greedy action
  - With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/|\mathcal{A}| + 1 - \epsilon & \text{if } a^* = \arg\max_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$$

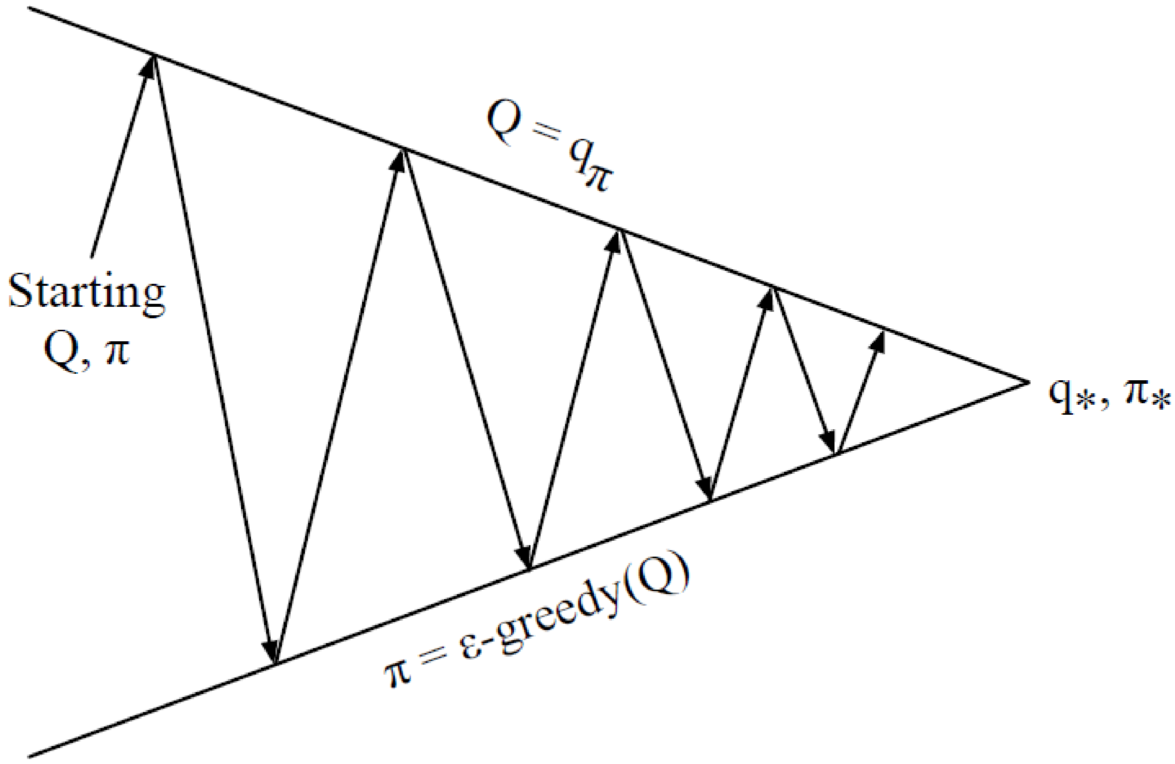
Monte Carlo with $\epsilon - Greedy$ Exploration

- Policy improvement theorem: For any policy π, the $\epsilon - greedy$ policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a) \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_a q_\pi(s, a) \\
&\geq \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}|}}{1 - \epsilon} q_\pi(s, a) \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s)
\end{aligned}
$$

Therefore, $v_{\pi'}(s) \geq v_\pi(s)$ from the policy improvement theorem

Monte-Carlo Policy Iteration



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

Policy improvement ϵ-greedy policy improvement

GLIE

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- 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,

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

- For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE Monte-Carlo Control

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

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

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

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

MC VS. TD for Prediction and Control

- Temporal-difference(TD) learning has several advantages over Monte-Carlo(MC)
  - Lower variance
  - Online
  - Incomplete sequences

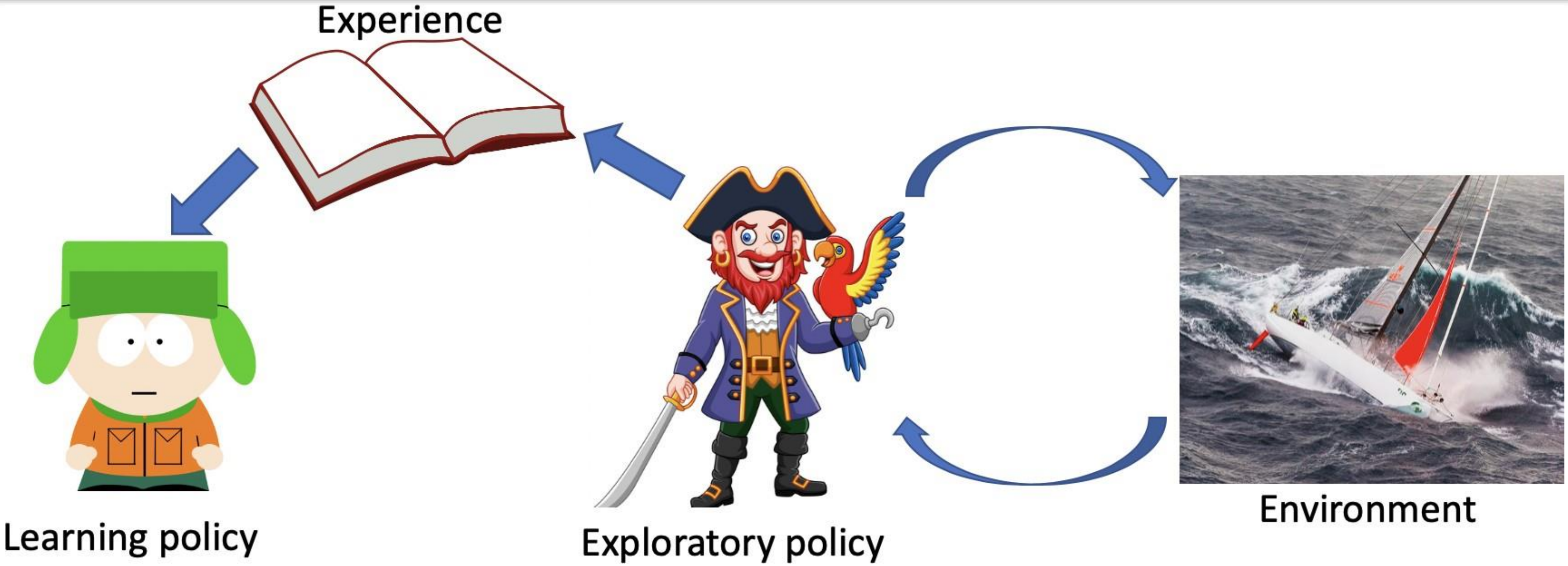
- So we can use TD instead of MC in our control loop
  - Apply TD to Q(S, A)
  - Use $\epsilon - greedy$ policy improvement
  - Update every time-step rather than at the end of one episode

On-policy learning and Off-policy learning

- **On-policy learning**
  - Learn on the job
  - Learn about policy π from experience sampled from π

- **Off-policy learning**
  - Look over someone's shoulder
  - Learn about policy π from experience sampled from μ

Off-policy learning



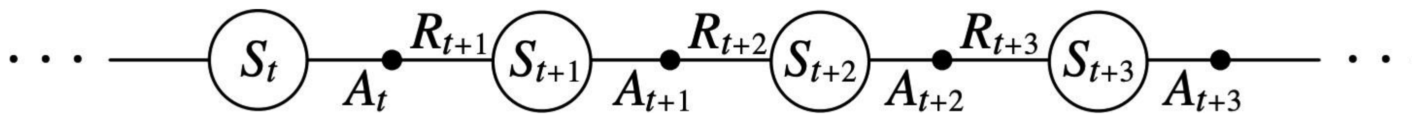
- Following behavior policy $\mu(\mathrm{a}|\mathrm{s})$ to collect data

$$S_1, A_1, R_2, ..., S_T \sim \mu$$
$$\text{Update } \pi \text{ using } S_1, A_1, R_2, ..., S_T$$

- Benefits：
  - Learn about optimal policy while following exploratory policy
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$

Sarsa: On-policy TD Control

- An episode consists of an alternating sequence of states and state-action pairs:



- $\epsilon - Greedy$ policy for one step, then bootstrap the action value function:

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

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

N-step Sarsa

- Consider the following *n-step* Q-returns for n=1,2,∞

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

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

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

- N-step Sarsa updates Q(s, a) towards the n-step Q-return:

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

Off-policy control with Q learning

- We allow both behavior and target policies to improve

- The target police π is greedy on Q(s, a)

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

- The behavior policy μ could be totally random, but we let it improve following $\epsilon - greedy$ on Q(s, a)

- Thus Q-learning target

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') =& R_{t+1} + \gamma Q(S_{t+1}, \arg\max Q(S_{t+1}, a')) \\ =& R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \end{aligned}$$

- Thus the Q-learning update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Comparison of Sarsa and Q-learning

□ Sarsa: On-Policy TD control

Choose action A_t from S_t using policy derived from Q with $\epsilon - greedy$
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 $\epsilon - greedy$

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

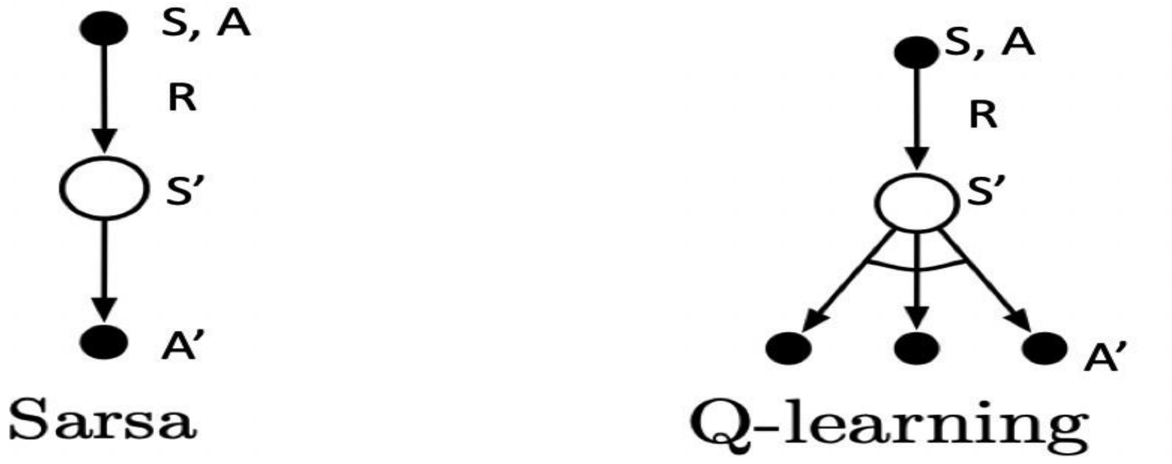
□ Q-learning: Off-Policy TD control

Choose action A_t from S_t using policy derived from Q with $\epsilon - greedy$
Take action A_t, observe R_{t+1} and S_{t+1}
Then 'imagine' A_{t+1} as argmax Q(S_{t+1}, a') in the update target

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

Comparison of Sarsa and Q-learning

- Backup diagram for Sarsa and Q-learning



- In Sarsa, A and A' are sampled from the same policy so it is on-policy
- In Q-learning, A and A′ are from different policies, with A being more exploratory and A′ determined directly by the max operator

Comparison of Sarsa and Q-learning

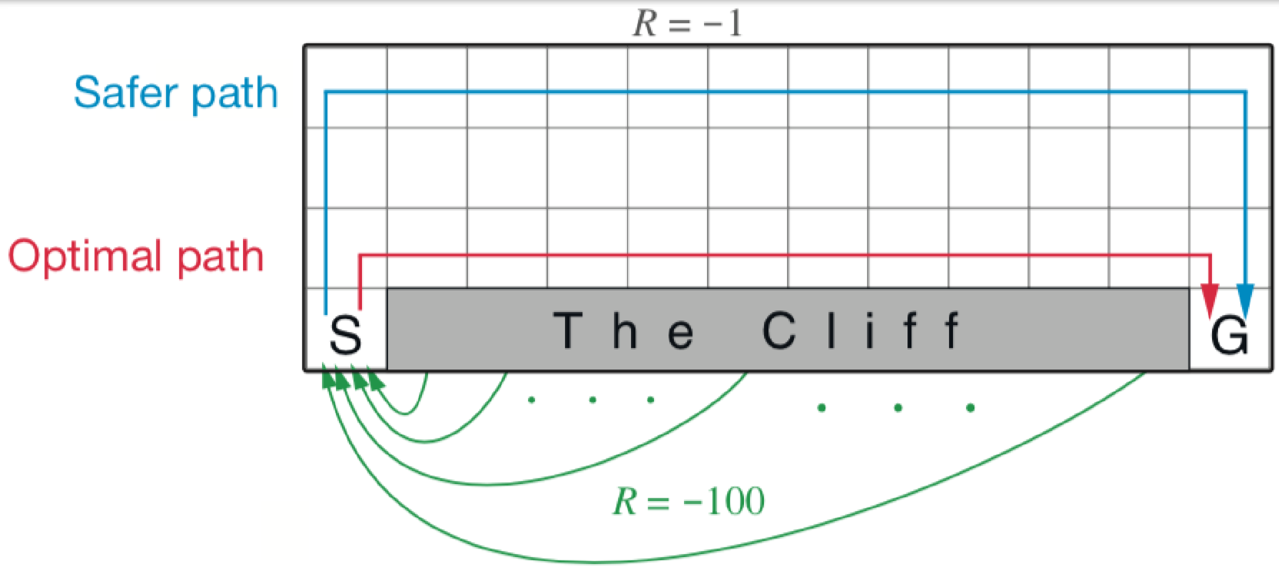
- Sarsa

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

- Q learning

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

Example on Cliff Walk



0	0	0	0	R	R	R	R	R	R	R	R
R	R	R	R	R	0	0	0	0	0	0	R
R	0	0	0	0	0	0	0	0	0	0	R
R	*	*	*	*	*	*	*	*	*	*	G

Result of Sarsa

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
R	R	R	R	R	R	R	R	R	R	R	R
R	*	*	*	*	*	*	*	*	*	*	G

Result of Q-Learning

On-line performance of Q-learning is worse than that of Sarsa

Summary of DP and TD

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

where $x \leftarrow^{\alpha} y$ is defined as $x \leftarrow x + \alpha(y - x)$

Importance Sampling

- Estimate the expectation of a function

$$E_{x \sim P}[f(x)] = \int f(x)P(x)dx \approx \frac{1}{n}\sum_i f(x_i)$$

- But sometimes it is difficult to sample x from P(x), then we can sample x from another distribution Q(x), then correct the weight

$$\begin{aligned}
\mathbb{E}_{x \sim P}[f(x)] &= \int P(x)f(x)dx \\
&= \int Q(x)\frac{P(x)}{Q(x)}f(x)dx \\
&= \mathbb{E}_{x \sim Q}\left[\frac{P(x)}{Q(x)}f(x)\right] \approx \frac{1}{n}\sum_i \frac{P(x_i)}{Q(x_i)}f(x_i)
\end{aligned}$$

Importance Sampling for Off-Policy RL

- Estimate the expectation of a return using trajectories sampled from another policy (behavior policy)

$$
\begin{aligned}
\mathbb{E}_{T\sim\pi}[g(T)] &= \int P(T)g(T)dT \\
&= \int Q(T)\frac{P(T)}{Q(T)}g(T)dT \\
&= \mathbb{E}_{T\sim\mu}\left[\frac{P(T)}{Q(T)}g(T)\right] \\
&\approx \frac{1}{n}\sum_i \frac{P(T_i)}{Q(T_i)}g(T_i)
\end{aligned}
$$

Importance Sampling for Off-Policy MC

- Generate episode from behavior policy μ and compute the generated return G_t

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

- Weight return G_t according to similarity between policies
  - Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \dots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value towards correct return

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{\pi/\mu} - V(S_t) \right)$$

Importance Sampling for Off-Policy TD

- Use TD targets generated from μ to evaluate π
- Weight TD target $R + \lambda V(S')$ by importance sampling
- Only need a single importance sampling correction

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

- Policies only need to be similar over a single step

Importance Sampling on Q-learning

- Off-policy TD

$$V(S_t) \leftarrow V(S_t) + \alpha\Big(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1} + \lambda V(S_{t+1})) - V(S_t)\Big)$$

- Why don't use importance sampling on Q-learning?
- Short answer: because Q-learning does not make expected value estimates over the policy distribution.
- Remember bellman optimality backup from value iteration

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q(s',a')$$

  - Q-learning can be considered as sample-based version of value iteration, except instead of using the expected value over the transition dynamics, we use the sample collected from the environment

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

  - Q-learning is over the transition distribution, not over policy distribution thus no need to correct different policy distributions

Large-Scale Reinforcement Learning

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space

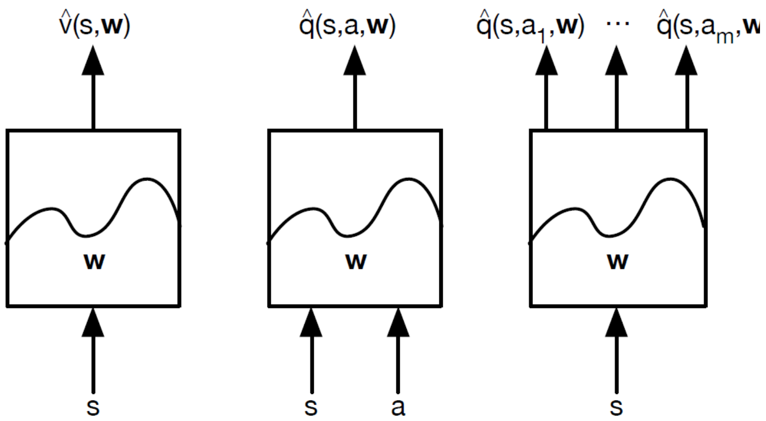
Value Function Approximation

- So far we have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - *Generalise* from seen states to unseen states
  - *Update* parameter $\mathbf{w}$ using MC or TD learning

Types of Value Function Approximation



There are many function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

Gradient Descent

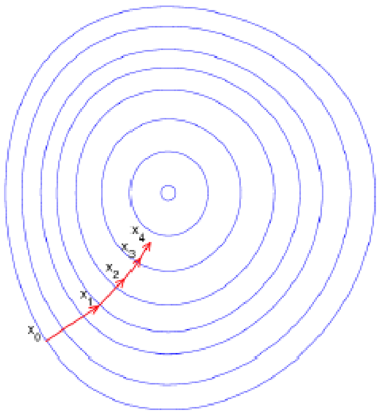
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be

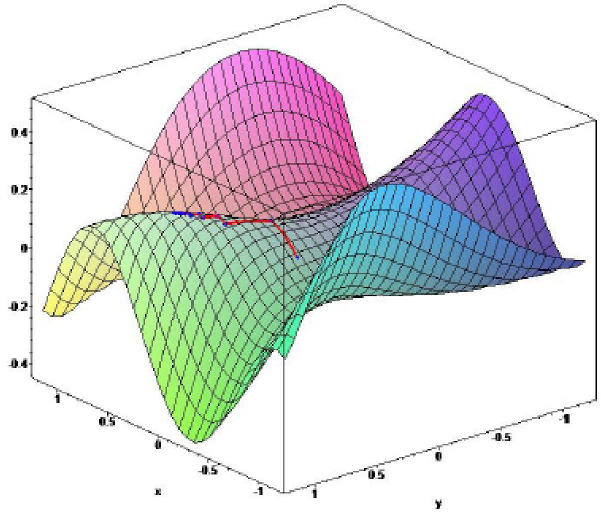
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$
- Adjust $\mathbf{w}$ in direction of -ve gradient

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

where α is a step-size parameter





Value Function Approx. By Stochastic Gradient Descent

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

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

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent *samples* the gradient

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

- Expected update is equal to full gradient update

Feature Vectors

- Represent state by a *feature vector*

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

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

Linear Value Function Approximation

- Represent value function by a linear combination of features

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

- Objective function is quadratic in parameters $\mathbf{w}$

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

- Stochastic gradient descent converges on *global* optimum
- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$
$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

Update = *step-size* × *prediction error* × *feature value*

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

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

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

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

  - For TD(λ), the target is the λ-return G_t^λ

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

Monte-Carlo with Value Function Approximation

- Return G_t is an unbiased, noisy sample of true value $v_\pi(S_t)$
- Can therefore apply supervised learning to "training data":

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, ..., \langle S_T, G_T \rangle$$

- For example, using *linear Monte-Carlo policy evaluation*

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

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

TD Learning with Value Function Approximation

- The TD-target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ is a *biased* sample of true value $v_\pi(S_t)$
- Can still apply supervised learning to "training data":

$$\langle S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w}) \rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, \mathbf{w}) \rangle, ..., \langle S_{T-1}, R_T \rangle$$

- For example, using *linear TD(0)*

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

- Linear TD(0) converges (close) to global optimum

TD(λ) with Value Function Approximation

- The λ-return G_t^{λ} is also a biased sample of true value $v_{\pi}(s)$
- Can again apply supervised learning to "training data":

$$\left\langle S_1, G_1^{\lambda} \right\rangle, \left\langle S_2, G_2^{\lambda} \right\rangle, ..., \left\langle S_{T-1}, G_{T-1}^{\lambda} \right\rangle$$

- Forward view linear TD(λ)

$$\begin{aligned} \Delta \mathbf{w} &= \alpha({\color{red}G_t^{\lambda}} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w}) \\ &= \alpha({\color{red}G_t^{\lambda}} - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t) \end{aligned}$$

- Backward view linear TD(λ)

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

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimise mean-squared error between approximate action-value fn $\hat{q}(S, A, \mathbf{w})$ and true action-value fn $q_\pi(S, A)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

Linear Action-Value Function Approximation

- Represent state *and* action by a *feature vector*

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

- Represent action-value fn by linear combination of features

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

- Stochastic gradient descent update

$$\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$$

$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A)$$

Incremental Control Algorithms

- Like prediction, we must substitute a *target* for $q_\pi(S, A)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

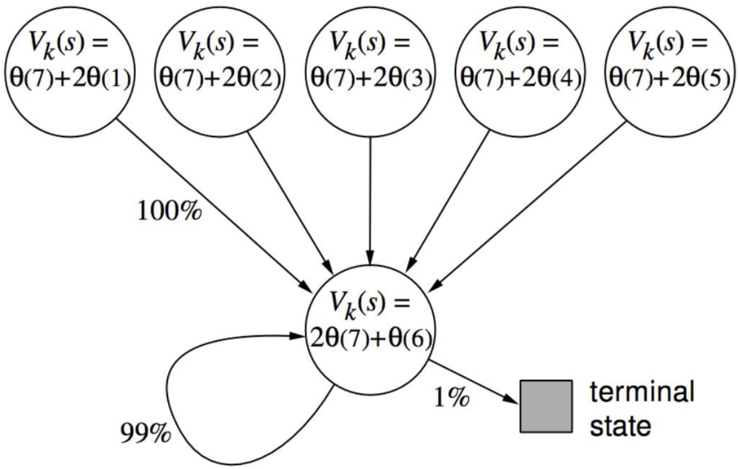
  - For forward-view TD(λ), target is the action-value λ-return

$$\Delta \mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

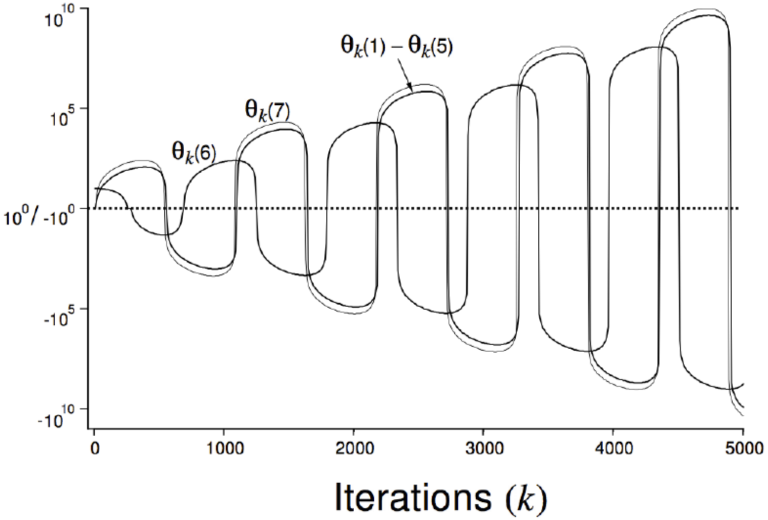
  - For backward-view TD(λ), equivalent update is

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

Convergence



Baird's Counterexample



Parameter Divergence in Baird's Counterexample

Convergence of Prediction and Control Algorithms

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

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

(✓) = chatters around near-optimal value function

Batch Reinforcement Learning

- Gradient descent is simple and appealing
- But it is *not* sample efficient
- Batch methods seek to find the best fitting value function
- Given the agent's experience ("training data")

Least Squares Prediction

- Given value function approximation $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And *experience* $\mathcal{D}$ consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Which parameters $\mathbf{w}$ give the *best fitting* value fn $\hat{v}(s, \mathbf{w})$?
- Least squares algorithms find parameter vector $\mathbf{w}$ minimising sum-squared error between $\hat{v}(s_t, \mathbf{w})$ and target values v_t^π,

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

Stochastic Gradient Descent with Experience Replay

Given experience consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

Repeat:

1. Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

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

Converges to least squares solution

$$\mathbf{w}^\pi = \underset{\mathbf{w}}{\operatorname{argmin}}\, LS(\mathbf{w})$$

Experience Replay in Deep Q-Networks (DQN)

DQN uses experience replay and fixed Q-targets

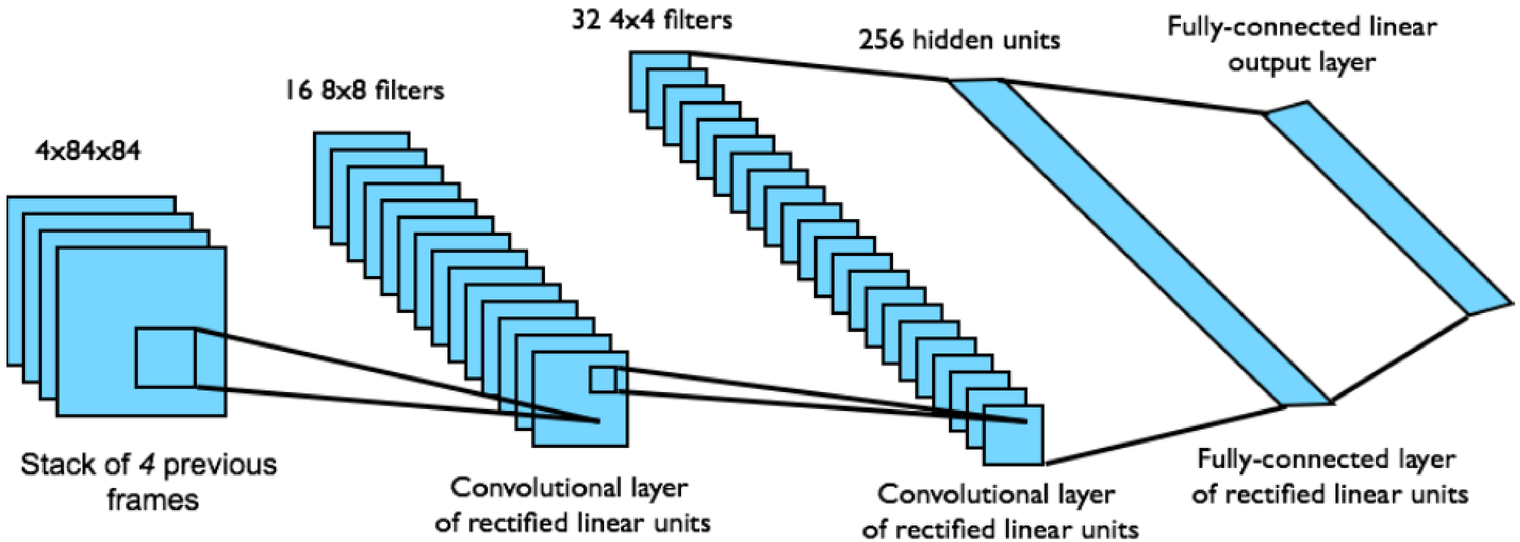
- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

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

- Using variant of stochastic gradient descent

DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

Linear Least Squares Prediction

- Experience replay finds least squares solution
- But it may take many iterations
- Using *linear* value function approximation $\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^\top \mathbf{w}$
- We can solve the least squares solution directly
  - At minimum of $LS(\mathbf{w})$, the expected update must be zero

$$
\begin{aligned}
\mathbb{E}_{\mathcal{D}}\left[\Delta \mathbf{w}\right] &= 0 \\
\alpha \sum_{t=1}^{T} \mathbf{x}(s_t)(v_t^\pi - \mathbf{x}(s_t)^\top \mathbf{w}) &= 0 \\
\sum_{t=1}^{T} \mathbf{x}(s_t) v_t^\pi &= \sum_{t=1}^{T} \mathbf{x}(s_t) \mathbf{x}(s_t)^\top \mathbf{w} \\
\mathbf{w} &= \left(\sum_{t=1}^{T} \mathbf{x}(s_t) \mathbf{x}(s_t)^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^\pi
\end{aligned}
$$

  - For N features, direct solution time is $O(N^3)$
  - Incremental solution time is $O(N^2)$ using Shermann-Morrison

Linear Least Squares Prediction Algorithms

- We do not know true values v_t^{π}
- In practice, our "training data" must use noisy or biased samples of v_t^{π}

  LSMC Least Squares Monte-Carlo uses return
  $v_t^{\pi} \approx G_t$

  LSTD Least Squares Temporal-Difference uses TD target
  $v_t^{\pi} \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

  LSTD(λ) Least Squares TD(λ) uses λ-return
  $v_t^{\pi} \approx G_t^{\lambda}$

- In each case solve directly for fixed point of MC / TD / TD(λ)

Linear Least Squares Prediction Algorithms

LSMC

$$0 = \sum_{t=1}^{T} \alpha (G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t)\mathbf{x}(S_t)^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) G_t$$

LSTD

$$0 = \sum_{t=1}^{T} \alpha (R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$

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

LSTD(λ)

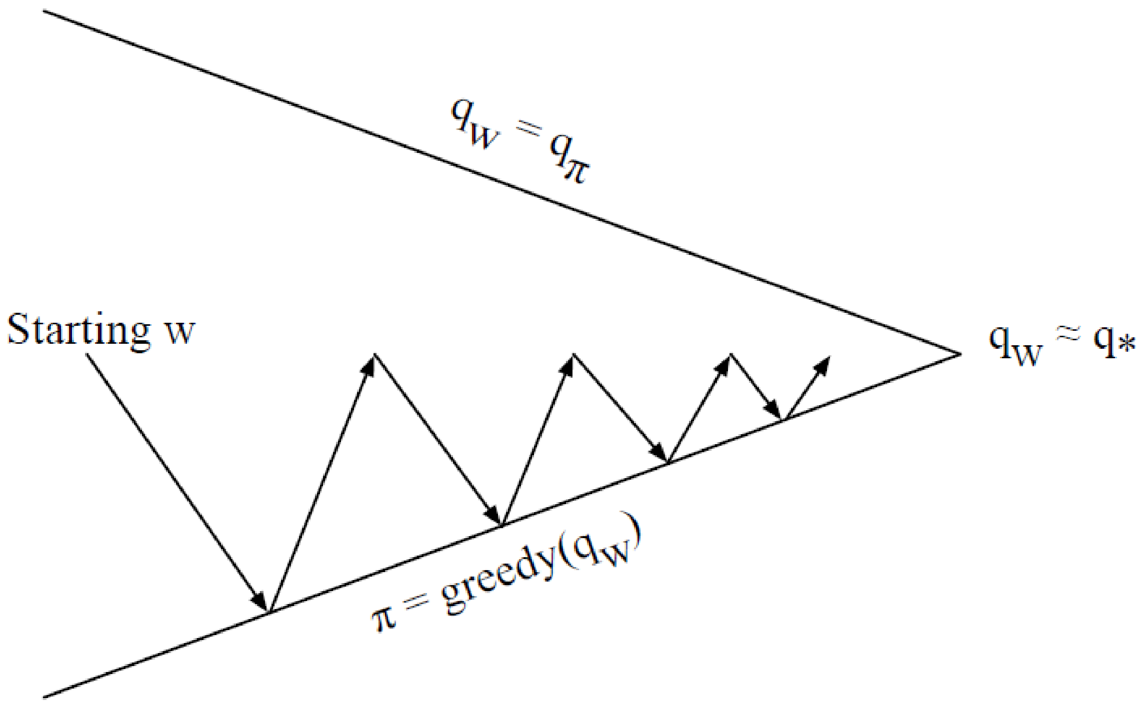
$$0 = \sum_{t=1}^{T} \alpha \delta_t E_t$$

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

Convergence of Linear Least Squares Prediction Algorithms

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

Least Squares Policy Iteration



Policy evaluation Policy evaluation by least squares Q-learning

Policy improvement Greedy policy improvement

Least Squares Action-Value Function Approximation

- Approximate action-value function $q_\pi(s, a)$
- using linear combination of features $\mathbf{x}(s, a)$

$$\hat{q}(s, a, \mathbf{w}) = \mathbf{x}(s, a)^\top \mathbf{w} \approx q_\pi(s, a)$$

- Minimise least squares error between $\hat{q}(s, a, \mathbf{w})$ and $q_\pi(s, a)$
- from experience generated using policy π
- consisting of $\langle (state, action), value \rangle$ pairs

$$\mathcal{D} = \{\langle (s_1, a_1), v_1^\pi \rangle, \langle (s_2, a_2), v_2^\pi \rangle, ..., \langle (s_T, a_T), v_T^\pi \rangle\}$$

Least Squares Control

- For policy evaluation, we want to efficiently use all experience
- For control, we also want to improve the policy
- This experience is generated from many policies
- So to evaluate $q_\pi(S, A)$ we must learn off-policy
- We use the same idea as Q-learning:
  - Use experience generated by old policy
    $S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{old}$
  - Consider alternative successor action $A' = \pi_{new}(S_{t+1})$
  - Update $\hat{q}(S_t, A_t, \mathbf{w})$ towards value of alternative action
    $R_{t+1} + \gamma \hat{q}(S_{t+1}, A', \mathbf{w}))$

Least Squares Q-Learning

- Consider the following linear Q-learning update

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

- LSTDQ algorithm: solve for total update = zero

$$0 = \sum_{t=1}^{T} \alpha (R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \mathbf{x}(S_t, A_t)$$
$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t, A_t)(\mathbf{x}(S_t, A_t) - \gamma \mathbf{x}(S_{t+1}, \pi(S_{t+1})))^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t, A_t) R_{t+1}$$

Least Squares Policy Iteration Algorithm

- The following pseudocode uses LSTDQ for policy evaluation
- It repeatedly re-evaluates experience $\mathcal{D}$ with different policies

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

Conclusion

- learn two Model-free control, $\epsilon - greedy$ exploration, Sarsa, Q-learning, on-policy, off-policy.
- Be able to implement MC and TD, including prediction and control.
- Know why and when to use the importance sampling.
- Incremental Methods for Value Function Approximation
- Batch Methods for Value Function Approximation

- 作业1：独立完成，提交截止日期4月10日