Temporal Difference Learning

Reinforcement Learning

September 21, 2021

MSU

Reminder

Reminder: Value functions

State value function (V-function): $V(s) = \mathbb{E}_T\left[\sum_{t \geq 0} {}^{t}r_t \mid s_0 = s\right]$

State-action value function (Q-function): $Q(s, a) = \mathbb{E}_T\left[\sum_{t \geq 0} {}^{t}r_t \mid s_0 = s, a_0 = a\right]$

Reminder: Value functions

State value function (V-function): $V(s) = \mathbb{E}_T\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s\right]$

State-action value function (Q-function): $Q(s, a) = \mathbb{E}_T\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a\right]$

Optimal V-function: $V^*(s) = \max_\pi V^\pi(s)$

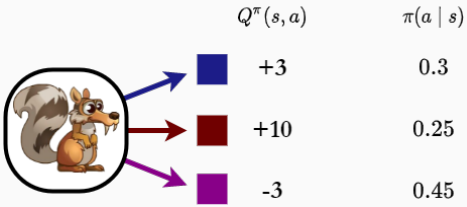
Optimal Q-function: $Q^*(s, a) = \max_\pi Q^\pi(s, a)$

Reminder: Policy Improvement

Policy Improvement:

greedy policy with respect to Q is not worse than π:

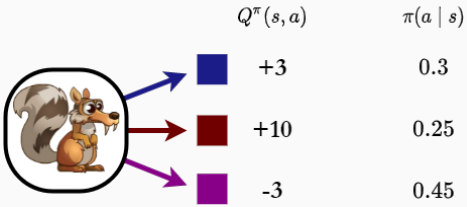
$$\pi'(s) = \arg\max_a Q^\pi(s, a)$$



Reminder: Policy Improvement

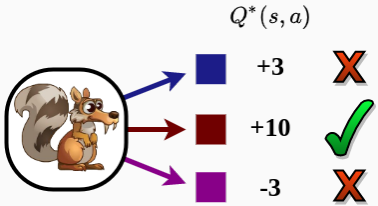
Policy Improvement:
greedy policy with respect to Q is not worse than :

$$\text{^psq} \quad \underset{a}{\text{argmax}}\, Q\ \text{ps;aq} \quad \text{ù ñ ^ ©}$$



Bellman optimality principle:
greedy policy with respect to optimal Q-function is optimal:

$$\text{psq} \quad \underset{a}{\text{argmax}}\, Q\ \text{ps;aq}$$



Bellman (expectation) equations: $\forall\ s, a$:

$$Q(s,a) = r(s,a) + \mathbb{E}_{s' \sim p(s'|s,a)} V(s')$$

$$V(s) = \mathbb{E}_{a \sim p(a|s)} Q(s,a)$$

Reminder: Bellman Equations

Bellman (expectation) equations: $\forall\, s, a$:

$$Q(s,a) = r(s,a) + \mathbb{E}_{s' \sim p(s'|s,a)} V(s')$$

$$V(s) = \mathbb{E}_{a \sim p(a|s)} Q(s,a)$$

Bellman (optimality) equations: $\forall s, a$:

$$Q(s,a) = r(s,a) + \mathbb{E}_{s' \sim p(s'|s,a)} V(s')$$

$$V(s) = \max_a Q(s,a)$$

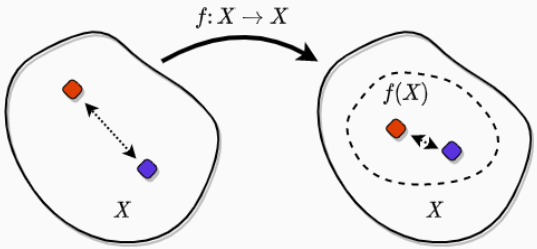
Reminder: Point iteration

Contraction Mapping

$f(x)\colon X \to X$ is a contraction mapping if:

$$\forall x_1, x_2 : \quad d(f(x_1), f(x_2)) \leq \quad d(x_1, x_2),$$

where — some metric, 1.



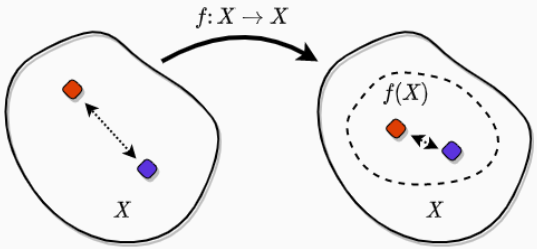
Reminder: Point iteration

Contraction Mapping

$f(x): X \to X$ is a contraction mapping if:

$$\forall x_1, x_2: \quad \rho(f(x_1), f(x_2)) \leq \alpha \, \rho(x_1, x_2),$$

where ρ — some metric, $\alpha < 1$.



If f is a contraction mapping:

- equation $x = f(x)$ has unique solution;
- we can find it using point iteration method

$$x_{k+1} = f(x_k),$$

starting from any x_0.

Reminder: Policy Iteration

Setup: $|S| ! 8; |A| ! 8$
Assumption: model is known

Reminder: Policy Iteration

Setup: $|S| \; ! \; 8; |A| \; ! \; 8$
Assumption: model is known

Policy Iteration

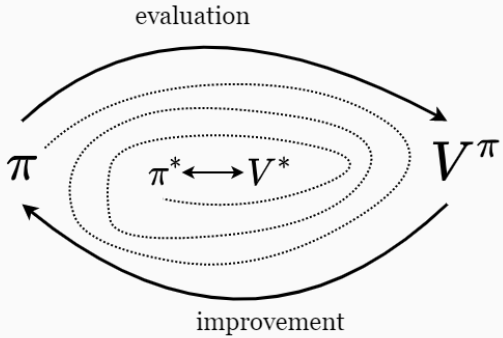
- Policy Evaluation:
  compute V^{k} for current policy $_{k}$;
- Policy Improvement:
  $_{k \;\; 1}psq \; Đ \; \underset{a}{\text{argmax}} \, Q \; ^{k}ps; aq;$
- repeat;

Reminder: Policy Iteration

Setup: $|S| < \infty$; $|A| < \infty$
Assumption: model is known

Policy Iteration

- Policy Evaluation:
  compute V^{π_k} for current policy π_k;
- Policy Improvement:
  $\pi_{k+1}(s) \doteq \arg\max_a Q^{\pi_k}(s,a)$;
- repeat;



Reminder: Generalized Policy Iteration

Generalized Policy Iteration

Initialize V; arbitrary, repeat:

Reminder: Generalized Policy Iteration

Generalized Policy Iteration

Initialize V; arbitrary, repeat:

- Policy Evaluation: move V towards V *(at least a little)*

  e.g. perform K iterations of point iteration method for Bellman equation

Reminder: Generalized Policy Iteration

Generalized Policy Iteration

Initialize V; arbitrary, repeat:

- Policy Evaluation: move V towards V *(at least a little)*
  - e.g. perform K iterations of point iteration method for Bellman equation
- Policy Improvement: improve using V *(at least a little)*
  - e.g. greedy policy improvement

Reminder: Generalized Policy Iteration

Generalized Policy Iteration

Initialize V; arbitrary, repeat:

- Policy Evaluation: move V towards V *(at least a little)*
  - e.g. perform K iterations of point iteration method for Bellman equation
- Policy Improvement: improve using V *(at least a little)*
  - e.g. greedy policy improvement

This always works!!!

Reminder: Generalized Policy Iteration

Generalized Policy Iteration

Initialize V; arbitrary, repeat:

- Policy Evaluation: move V towards V *(at least a little)*
  e.g. perform K iterations of point iteration method for Bellman equation
- Policy Improvement: improve using V *(at least a little)*
  e.g. greedy policy improvement

This always works!!!

Interesting case: K 1 and greedy policy improvement

Reminder: Value Iteration

Setup: $|S| < \infty$; $|A| < \infty$
Assumption: model is known

Value Iteration

- solve Bellman optimality equation
  (e. g. for V):

$$V_{k+1}(s) \leftarrow \max_a r(s,a) + \mathbb{E}_{s'} V_k(s')$$

- $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ is optimal;

Reminder: Value Iteration

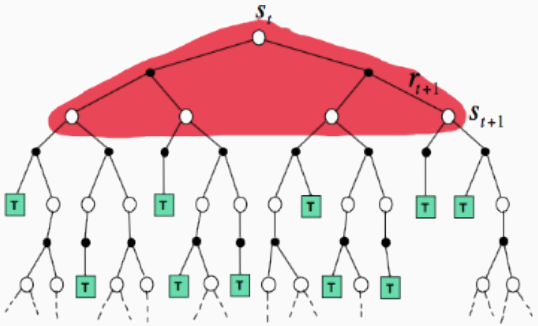
Setup: $|S| < \infty$; $|A| < \infty$
Assumption: model is known

Value Iteration

- solve Bellman optimality equation
  (e. g. for V):

$$V_{k+1}(s) \leftarrow \max_a r(s,a) + \gamma \mathbb{E}_{s'} V_k(s')$$

- $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ is optimal;



Model-free learning

Trial and Error learning

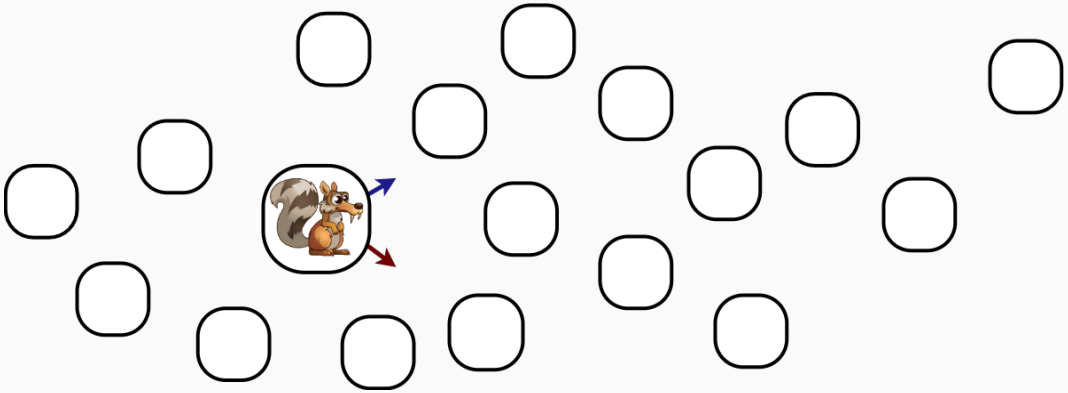
Setup: $|S| \; ! \; 8 ; |A| \; ! \; 8$

~~Assumption: model is known~~

Trial and Error learning

Setup: $|S| \; ! \; 8; |A| \; ! \; 8$

~~Assumption: model is known~~



Idea 1: Model-based RL

Learn model through experience

Use VI / PI with learned model to find optimal policy

Idea 1: Model-based RL

Learn model through experience
Use VI / PI with learned model to find optimal policy

What is easier to create:

a) universe

b) brain

Idea 1: Model-based RL

Learn model through experience
Use VI / PI with learned model to find optimal policy

What is easier to create:

a) universe

$S \quad A$ Ñ PpSq

X (almost) supervised learning

b) brain

Idea 1: Model-based RL

Learn model through experience
Use VI / PI with learned model to find optimal policy

What is easier to create:

a) universe

S A Ñ PpSq

X (almost) supervised learning

b) brain

X S Ñ A

dataset is not provided :(

Idea 1: Model-based RL

Learn model through experience
Use VI / PI with learned model to find optimal policy

What is easier to create:

a) universe

$S \quad A$ Ñ $P(S')$

X (almost) supervised learning

b) brain

X S Ñ A

dataset is not provided :(

?!?

Idea 2: Model-free RL

Do not try to learn the model

Map states and actions to reward directly

Idea 2: Model-free RL

Do not try to learn the model
Map states and actions to reward directly

Where do we use the model?

Policy Iteration

- evaluate policy
  e. g. by solving Bellman equation:

$$V^{k}(s) \leftarrow \mathbb{E}_{a}\, r(s,a) \quad \mathbb{E}_{s'} V^{k}(s')$$

- $k+1(s) \leftarrow \arg\max_{a} Q^{k}(s,a)$;
- repeat;

Value Iteration

÷ solve Bellman optimality equation
(e. g. for V):

$$V_{k+1}(s) \leftarrow \max_{a} r(s,a) \quad \mathbb{E}_{s'} V_{k}(s')$$

- $(s) \leftarrow \arg\max_{a} Q(s,a)$ is optimal;

Idea 2: Model-free RL

Do not try to learn the model
Map states and actions to reward directly

Where do we use the model?

Policy Iteration

- evaluate policy
  e. g. by solving Bellman equation:

$$V^{k}(s) \leftarrow \mathbb{E}_{a}\, r(s,a) + \mathbb{E}_{s'} V^{k}(s')$$

- $\pi_{k+1}(s) \leftarrow \arg\max_{a} Q^{k}(s,a)$;
- repeat;

Value Iteration

÷ solve Bellman optimality equation
(e. g. for V):

$$V_{k+1}(s) \leftarrow \max_{a} r(s,a) + \mathbb{E}_{s'} V_{k}(s')$$

- $\pi(s) \leftarrow \arg\max_{a} Q(s,a)$ is optimal;

Switching to Q-function

Policy Iteration

- evaluate policy:

$$Q^{k}(s,a) \quad r(s,a) \quad \mathbb{E}_{s'}\mathbb{E}_{a'}Q^{k}(s',a')$$

- $\pi_{k+1}(s) \leftarrow \operatorname{argmax}_a Q^{k}(s,a)$;
- repeat;

Value Iteration

÷ solve Bellman optimality equation:

$$Q_{k+1}(s,a) \leftarrow r(s,a) \quad \mathbb{E}_{s'} \max_{a'} Q_k(s',a')$$

- $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ is optimal;

Switching to Q-function

Policy Iteration

- evaluate policy:

$$Q^{k}(s,a) \quad r(s,a) \quad \mathbb{E}_{s'}\mathbb{E}_{a'}Q^{k}(s',a')$$

- $\pi_{k+1}(s) = \arg\max_{a} Q^{k}(s,a)$;
- repeat;

Value Iteration

÷ solve Bellman optimality equation:

$$Q_{k+1}(s,a) = r(s,a) \quad \mathbb{E}_{s'}\max_{a'} Q_k(s',a')$$

- $\pi(s) = \arg\max_{a} Q(s,a)$ is optimal;

$$x \quad \mathbb{E}_{s'} f(s',x); \quad x - ?$$

Switching to Q-function

Policy Iteration

- evaluate policy:

$$Q^{k}(s,a) \quad r(s,a) \quad E_{s'} E_{a'} Q^{k}(s',a')$$

- $\pi_{k+1}(s) \leftarrow \operatorname{argmax}_a Q^{k}(s,a)$;
- repeat;

Value Iteration

÷ solve Bellman optimality equation:

$$Q_{k+1}(s,a) \leftarrow r(s,a) \quad E_{s'} \max_{a'} Q_k(s',a')$$

- $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ is optimal;

$$x \quad E_{s'} f(s',x); \quad x - ?$$

Ok, not clear how to work with this...

Can we derive *some* model-free PI / VI?

Monte Carlo Backup

$$Q\ ps; aq : \quad E_T \quad |s_0 \ \ s;a_0 \ \ aRpT\ q$$

How to solve this: x $E_{pp"q} f\ p"q?$

Monte Carlo Backup

$$Q\ ps;aq: \quad E_{T \quad |s_0 \ s;a_0 \ a} RpT\ q$$

How to solve this: $x \quad E_{pp"q} f\ p"q?$

Play a lot of games using to collect data for Monte Carlo estimations

Monte Carlo Backup

$$Q(s,a): \quad \mathbb{E}_{T \mid s_0 = s, a_0 = a}[R(T)]$$

How to solve this: $\mathbb{E}_{p(p'')}[f(p'')]$?

Play a lot of games using to collect data for Monte Carlo estimations

Monte Carlo estimation:

$$Q(s,a) \approx \frac{1}{N} \sum_{i=0}^{N} R(T_i)$$

where $T_i \mid s_0 = s, a_0 = a$

Monte Carlo Algorithm

Monte Carlo Algorithm

Initialize policy $\pi(s)$ arbitrarily.

for $k = 0; 1; 2 \ldots$

- play several games using π;
- estimate $Q(s; a)$ using Monte Carlo;
- $\pi(s) \leftarrow \operatorname{argmax}_a Q(s; a)$

! requires infinite samples for each pair $s; a$;

Monte Carlo Algorithm

Monte Carlo Algorithm

Initialize policy $\pi(s)$ arbitrarily.

for $k = 0; 1; 2 \ldots$

- play several games using π;
- estimate $Q(s; a)$ using Monte Carlo;
- $\pi(s) \leftarrow \operatorname{argmax}_a Q(s; a)$

! requires infinite samples for each pair $s; a$;

still needs full episodes;

still does not utilize MDP structure;

wastes information about state connections;

high variance of collected samples;

Monte Carlo Algorithm

Monte Carlo Algorithm

Initialize policy $\pi(s)$ and $Q(s,a)$ arbitrarily.

for $k = 0, 1, 2, \ldots$

- play several games using π;
- update $Q(s,a)$ with new samples;
- $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$

! requires infinite samples for each pair s, a;

still needs full episodes;

still does not utilize MDP structure;

wastes information about state connections;

high variance of collected samples;

Reuse samples from previous policy
(for example, with smaller weight)

Closer look at Monte Carlo

Suppose we want to compute online mean m of i. i. d. samples $x_1; x_2 : : :$. On step k:

$$m_k : \quad \frac{1}{k} \sum_{i \quad 1}^{k} x_i$$

Closer look at Monte Carlo

Suppose we want to compute online mean m of i. i. d. samples $x_1, x_2, \ldots$. On step k:

$$m_k := \frac{1}{k} \sum_{i=1}^{k} x_i = \frac{k-1}{k} m_{k-1} + \frac{1}{k} x_k$$

Closer look at Monte Carlo

Suppose we want to compute online mean m_k of i. i. d. samples $x_1; x_2 : : :$. On step k:

$$m_k := \frac{1}{k} \sum_{i=1}^{k} x_i = \frac{k-1}{k} m_{k-1} + \frac{1}{k} x_k = m_{k-1} + \left(\frac{1}{k} \right) \left(x_k - m_{k-1} \right)$$

Closer look at Monte Carlo

Suppose we want to compute online mean of i. i. d. samples $x_1, x_2, \ldots$. On step k:

$$m_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{k-1}{k} m_{k-1} + \frac{1}{k} x_k = m_{k-1} + \frac{1}{k}(x_k - m_{k-1})$$

exponential smoothing

Closer look at Monte Carlo

Suppose we want to compute online mean of i. i. d. samples $x_1, x_2, \ldots$. On step k:

$$m_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{k-1}{k} m_{k-1} + \frac{1}{k} x_k = m_{k-1} + \frac{1}{k}(x_k - m_{k-1})$$

$$m_k = (1-\alpha) m_{k-1} + \alpha x_k = m_{k-1} - \alpha (m_{k-1} - x_k)$$

exponential smoothing — stoch. gradient descent

Closer look at Monte Carlo

Suppose we want to compute online mean of i. i. d. samples $x_1, x_2, \ldots$. On step k:

$$m_k = \frac{1}{k} \sum_{i=1}^{k} x_i = \frac{k-1}{k} m_{k-1} + \frac{1}{k} x_k = m_{k-1} - \frac{1}{k}(m_{k-1} - x_k)$$

$$m_k = (1-\alpha) m_{k-1} + \alpha x_k = m_{k-1} - \alpha (m_{k-1} - x_k)$$

exponential smoothing

stoch. gradient descent

Closer look at Monte Carlo

Suppose we want to compute online mean of i. i. d. samples $x_1, x_2, \ldots$. On step k:

$$m_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{k-1}{k} m_{k-1} + \frac{1}{k} x_k = m_{k-1} - \frac{1}{k}(m_{k-1} - x_k)$$

$$m_k = m_{k-1} - \alpha_k (m_{k-1} - x_k)$$

exponential smoothing — stoch. gradient descent

Convergence:

$m_k \to \mathbb{E}x$ if learning rate satisfies Robbins Monro conditions:

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

Temporal Di erence Idea

Solving equations

Equation	Method	Update
$x = f(x)$	Point Iteration	$x_{k+1} := f(x_k)$
$x = E_{s^{-1}}[f(s^{-1})]$	Exponential smoothing	$x_{k+1} := x_k + \alpha_k p(f(s^{-1}) - x_k)$ $s^{-1} \sim p(s^{-1})$
$x = E_{s^{-1}}[f(s^{-1}, x)]$	Stochastic approximation	

Solving equations

Equation	Method	Update
$x \quad f(x)$	Point Iteration	$x_{k+1} : f(x_k)$
$x \quad E_{s^{-1}}[f(s^{-1})]$	Exponential smoothing	$x_{k+1} : x_k + \frac{1}{p(s^{-1})}[f(s^{-1}) - x_k]$
$x \quad E_{s^{-1}}[f(s^{-1}, x)]$	Stochastic approximation	$x_{k+1} : x_k + \frac{1}{p(s^{-1})}[f(s^{-1}, x_k) - x_k]$

Temporal Di erence: intuition 1

View 1: solving Bellman expectation equation using Robbins-Monro scheme:

$$Q(s,a) \leftarrow \underbrace{E_{s'}[r(s,a)}_{f(s',x)} + \gamma E_{a'} Q(s',a')]$$

Temporal Difference: intuition 1

View 1: solving Bellman expectation equation using Robbins-Monro scheme:

$$Q(s,a) = \underbrace{\mathbb{E}_{s'}[r + \gamma \mathbb{E}_{a'}[Q(s',a')]]}_{f(s',a';x)} - Q(s',a') + Q(s',a')$$

Temporal Difference: intuition 1

View 1: solving Bellman expectation equation using Robbins-Monro scheme:

$$Q(s,a) \leftarrow \underbrace{\mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s',a')]}_{f(s',x)} = \underbrace{\mathbb{E}_{s'}\mathbb{E}_{a'}[r + \gamma Q(s',a')]}_{f(s',a',x)}$$

To update value for (s,a) we need only transition (s,a,r,s',a'):

$$y = r + \gamma Q(s',a')$$

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot y$$

Requirements for Temporal Di erence

For transition (s, a, r, s', a'):

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \Big[\underbrace{r + Q_k(s',a')}_{\text{Bellman target}} - Q_k(s,a) \Big]$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{temporal di erence}}$$

Requirements for Temporal Di erence

For transition(s, a, r, s^1, a^1):

$$Q_{k-1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \Big(\underbrace{r + Q_k(s^1, a^1)}_{\text{Bellman target}} - Q_k(s,a) \Big)$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{temporal di erence}}$$

(s, a) considered xed .

$s^1 \sim p(s^1 \mid s, a)$ random variable from interaction experience.

Requirements for Temporal Di erence

For transition(s, a, r, s^1, a^1):

$$Q_{k-1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \Big[\underbrace{r + Q_k(s^1,a^1)}_{\text{Bellman target}} - Q_k(s,a) \Big]$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{temporal di erence}}$$

(s, a) considered xed .

$s^1 \sim p(s^1 \mid s, a)$ random variable from interaction experience.

$a^1 \sim p(a^1 \mid s^1)$ random variable from algorithm.

Temporal Difference: intuition 2

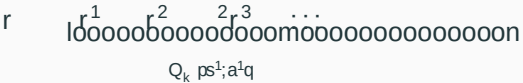
View 2: one-step bootstrapping: approximating future rewards using current approximation.

$$r \quad r^1 \quad r^2 \quad r^3 \quad \ldots$$

$$\underbrace{\qquad\qquad}_{Q_k(s^1, a^1)}$$

Temporal Di erence: intuition 2

View 2: one-step bootstrapping:
approximating future rewards using current approximation.



our own predictions used as targets!

Temporal Di erence vs Monte Carlo

$$Q_{k\ \ 1}ps; aq \text{ Ð } Q_k ps; aq \quad {}_k p\,yps; aq \quad Q_k ps; aqq$$

Which is better?

Temporal Di erence

$$yps; aq : \quad r \quad Q_k ps^{1}; a^{1}q$$

Monte Carlo

$$yps; aq : \quad r \quad r^{1} \quad {}^{2}r^{2} \quad : : :$$

Temporal Difference vs Monte Carlo

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k\big(y(s,a) - Q_k(s,a)\big)$$

Which is better?

Temporal Difference

$y(s,a)$: $r + Q_k(s',a')$

- ✗ updates after each step
- slow reward propagation

Monte Carlo

$y(s,a)$: $r + r' + \gamma^2 r'' + \dots$

- updates at the end of the episode
- ✗ fast reward propagation

Temporal Difference vs Monte Carlo

$$Q_{k-1}(s,a) \leftarrow Q_k(s,a) + \alpha_k(s,a)\left[y(s,a) - Q_k(s,a)\right]$$

Which is better?

Temporal Difference

$y(s,a) = r + Q_k(s', a')$

- ✗ updates after each step
- slow reward propagation
- ✗ low variance
- introduces bias

Monte Carlo

$y(s,a) = r + r' + \gamma^2 r'' + \ldots$

- updates at the end of the episode
- ✗ fast reward propagation
- high variance
- ✗ unbiased

Temporal Difference vs Monte Carlo

$$Q_{k-1}(s,a) \leftarrow Q_k(s,a) + \alpha_k(s,a)\left[y(s,a) - Q_k(s,a)\right]$$

Which is better?

Temporal Difference

$y(s,a) = r + Q_k(s', a')$

- ✗ updates after each step
- slow reward propagation
- ✗ low variance
- introduces bias

Monte Carlo

$y(s,a) = r + r' + \gamma^2 r'' + \ldots$

- updates at the end of the episode
- ✗ fast reward propagation
- high variance
- ✗ unbiased

Best estimation is somewhere in between (see TD( )).

Types of backups

Q-learning

Value Iteration and Policy Iteration connection

What if we improve our policy after every step?

$$\pi_{k}(s) \leftarrow \arg\max_{a} Q_{k}(s, a)$$

Value Iteration and Policy Iteration connection

What if we improve our policy after every step?

$$\pi_k(s) = \arg\max_a Q_k(s, a)$$

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left[r + Q_k(s', a') - Q_k(s, a) \right]$$

Value Iteration and Policy Iteration connection

What if we improve our policy after every step?

$$\pi_k(s) = \arg\max_a Q_k(s, a)$$

$$Q_{k-1}(s, a) = Q_k(s, a) \quad \gamma \quad r \quad Q_k(s', a') \quad Q_k(s, a)$$

$$Q_k(s, a) \quad \gamma \quad r \quad Q_k(s', \pi_k(s')) \quad Q_k(s, a)$$

Value Iteration and Policy Iteration connection

What if we improve our policy after every step?

$$\pi_k(s) \leftarrow \arg\max_a Q_k(s,a)$$

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\left(r + Q_k(s',a') - Q_k(s,a)\right)$$

$$Q_k(s,a) + \alpha\left(r + Q_k(s', \pi_k(s')) - Q_k(s,a)\right)$$

$$Q_k(s,a) + \alpha\left(r + \max_{a'} Q_k(s',a') - Q_k(s,a)\right)$$

Value Iteration and Policy Iteration connection

What if we improve our policy after every step?

$$\pi_k(s) \leftarrow \arg\max_a Q_k(s,a)$$

$$Q_{k-1}(s,a) \leftarrow Q_k(s,a)$$
$$\leftarrow r + Q_k(s^1, a^1) - Q_k(s,a)$$
$$\leftarrow r + Q_k(s^1, \pi_k(s^1)) - Q_k(s,a)$$
$$\leftarrow r + Q_k(s^1, \arg\max_{a^1} Q_k(s^1, a^1)) - Q_k(s,a)$$
$$\leftarrow r + \max_{a^1} Q_k(s^1, a^1) - Q_k(s,a)$$

Value Iteration and Policy Iteration connection

What if we improve our policy after every step?

$$\pi_k(s) \leftarrow \arg\max_a Q_k(s, a)$$

$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha \left[r + Q_k(s', a') - Q_k(s, a) \right]$$

$$Q_k(s, a) + \alpha \left[r + Q_k(s', \pi_k(s')) - Q_k(s, a) \right]$$

$$Q_k(s, a) + \alpha \left[r + Q_k(s', \arg\max_{a'} Q_k(s', a')) - Q_k(s, a) \right]$$

$$Q_k(s, a) + \alpha \left[r + \max_{a'} Q_k(s', a') - Q_k(s, a) \right]$$

Value Iteration and Policy Iteration connection

What if we improve our policy after every step?

$$\pi_k(s) = \arg\max_a Q_k(s, a)$$

$$Q_{k-1}(s, a) = Q_k(s, a) - \alpha \left(Q_k(s, a) - \left[r + \gamma Q_k(s', a') \right] \right)$$

$$Q_k(s, a) - \alpha \left(Q_k(s, a) - \left[r + \gamma Q_k(s', \pi_k(s')) \right] \right)$$

$$Q_k(s, a) - \alpha \left(Q_k(s, a) - \left[r + \gamma Q_k(s', \arg\max_{a'} Q_k(s', a')) \right] \right)$$

$$Q_k(s, a) - \alpha \left(Q_k(s, a) - \left[r + \gamma \max_{a'} Q_k(s', a') \right] \right)$$

View 1: Policy Iteration with policy improvement after each policy evaluation update;
View 2: solving Bellman optimality equation (model-free Value Iteration);

Convergence of temporal difference

Convergence of Q-learning updates

Let $Q_0(s, a)$ be initialized arbitrary, and for every s, a the following update is used:

$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha_k(s, a)\left[r(s, a) + \max_{a'} Q_k(s'_{k,s,a}, a') - Q_k(s, a)\right]$$

Convergence of temporal di erence

Convergence of Q-learning updates

Let $Q_0(s, a)$ be initialized arbitrary, and for every s, a the following update is used:

$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha_k(s, a)\left[r(s, a) + \max_{a'} Q_k(s'_{k;s;a}, a') - Q_k(s, a)\right]$$

Then, if:

$\hat{} \; s'_{k;s;a} \sim p(s' \mid s, a)$

Convergence of temporal di erence

Convergence of Q-learning updates

Let $Q_0(s,a)$ be initialized arbitrary, and for every s, a the following update is used:

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k(s,a)\left[r(s,a) + \max_{a'} Q_k(s'_{k;s,a}, a') - Q_k(s,a)\right]$$

Then, if:

- $s'_{k;s,a} \sim p(s' \mid s,a)$,
- with probability 1 for every s, a learning rate $\alpha_k(s,a) \in [0,1)$ satisfies Robbins-Monro conditions:

$$\sum_{k=0}^{\infty} \alpha_k(s,a) = \infty \qquad \sum_{k=0}^{\infty} \alpha_k^2(s,a) < \infty$$

Convergence of temporal di erence

Convergence of Q-learning updates

Let $Q_0(s,a)$ be initialized arbitrary, and for every s,a the following update is used:

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k(s,a)\left[r(s,a) + \max_{a'} Q_k(s'_{k,s,a},a') - Q_k(s,a)\right]$$

Then, if:

- $s'_{k,s,a} \sim p(s'|s,a)$,
- with probability 1 for every s,a learning rate $\alpha_k(s,a) \in [0,1)$ satis es Robbins-Monro conditions:

$$\sum_{k=0}^{\infty} \alpha_k(s,a) = \infty \qquad \sum_{k=0}^{\infty} \alpha_k^2(s,a) < \infty$$

then $Q_k(s,a) \xrightarrow[k\to\infty]{} Q^*(s,a)$

Exploration vs Exploitation

Analogy 1

Global optimization

Exploration vs Exploitation

Analogy 1

Global optimization

Exploration:
random search

Exploration vs Exploitation

Analogy 1

Global optimization

Exploration:
random search

Exploitation:
local optimization

Exploration vs Exploitation

Analogy 1
Global optimization

Exploration:
random search

Exploitation:
local optimization

Analogy 2
Multi-armed bandits
(regret minimization)

Exploration:
try something new

Exploitation:
try your favourite

Exploration vs Exploitation

Analogy 1
Global optimization

Exploration:
random search

Exploitation:
local optimization

Analogy 2
Multi-armed bandits
(regret minimization)

Exploration:
try something new

Exploitation:
try your favourite

Analogy 3
General MDP

Exploration:
explore the environment

Exploitation:
solve the task

"-greedy exploration

Do something random sometimes

Let's call policy"-greedy with respect to some Q-function, if it behaves:

randomly with probability "

greedy with probability 1 "

"-greedy exploration

Do something random sometimes

Let's call policy "-greedy with respect to some Q-function, if it behaves:

randomly with probability "

greedy with probability 1 "

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|} & \text{when } a = \arg\max_a Q(s,a) \\ \frac{\varepsilon}{|A|} & \text{otherwise} \end{cases}$$

"-greedy exploration

Do something random sometimes

Let's call policy "-greedy with respect to some Q-function, if it behaves:

randomly with probability "

greedy with probability 1 "

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|} & \text{when } a = \arg\max_a Q(s,a) \\ \frac{\varepsilon}{|A|} & \text{otherwise} \end{cases}$$

Decaying exploration:

$\varepsilon_k \to 0, \quad \varepsilon_k \neq 0$

Persistent exploration:

$\varepsilon_k = \text{const} > 0$

Q-learning

Q-learning (online)

Initialize $Q(s, a)$ arbitrarily;

observe s_0;
for $k = 0, 1, 2, \ldots$
- take action a_k " ε-greedy$(Q(s_k, a))$
- observe r_k, s_{k+1};

Q-learning

Q-learning (online)

Initialize $Q(s, a)$ arbitrarily;

observe s_0;

for $k = 0, 1, 2, \ldots$

- take action a_k ε-greedy$(Q(s_k, a))$
- observe r_k, s_{k+1};
- $y \leftarrow r_k + \max_{a_{k+1}} Q(s_{k+1}, a_{k+1})$;

Q-learning

Q-learning (online)

Initialize $Q(s, a)$ arbitrarily;

observe s_0;
for $k = 0, 1, 2, \ldots$
- take action a_k " ε-greedy$(Q(s_k, a))$
- observe r_k, s_{k+1};
- $y \leftarrow r_k + \max_{a_{k+1}} Q(s_{k+1}, a_{k+1})$;
- $Q(s_k, a_k) \leftarrow (1 - \alpha_k) Q(s_k, a_k) + \alpha_k y$

Q-learning

Q-learning (online)

Initialize $Q(s,a)$ arbitrarily;

observe s_0;
for $k = 0, 1, 2, \ldots$
- take action a_k " ε-greedy$(Q(s_k, a))$
- observe r_k, s_{k+1};
- $y = r_k + \max_{a_{k+1}} Q(s_{k+1}, a_{k+1})$;
- $Q(s_k, a_k) \leftarrow (1 - \alpha_k) Q(s_k, a_k) + \alpha_k y$

Q-learning (with replay buffer)

Initialize $Q(s,a)$ arbitrarily, $D = H$;

Q-learning

Q-learning (online)

Initialize $Q(s, a)$ arbitrarily;

observe s_0;

for $k = 0, 1, 2, \ldots$:

- take action a_k ε-greedy$(Q(s_k, a))$
- observe r_k, s_{k+1};
- $y \leftarrow r_k + \max_{a_{k+1}} Q(s_{k+1}, a_{k+1})$;
- $Q(s_k, a_k) \leftarrow (1 - \alpha_k) Q(s_k, a_k) + \alpha_k y$

Q-learning (with replay buffer)

Initialize $Q(s, a)$ arbitrarily, $D \leftarrow \{\}$;

observe s_0;

for $k = 0, 1, 2, \ldots$:

- take action a_k ε-greedy$(Q(s_k, a))$
- observe r_k, s_{k+1};
- store (s_k, a_k, r_k, s_{k+1}) in D;

Q-learning

Q-learning (online)

Initialize $Q(s, a)$ arbitrarily;

observe s_0;
for $k = 0, 1, 2, \ldots$:
- take action a_k ε-greedy$(Q(s_k, a))$
- observe r_k, s_{k+1};
- $y \leftarrow r_k + \max_{a_{k+1}} Q(s_{k+1}, a_{k+1})$;
- $Q(s_k, a_k) \leftarrow (1 - \alpha_k) Q(s_k, a_k) + \alpha_k y$

Q-learning (with replay buffer)

Initialize $Q(s, a)$ arbitrarily, $D \leftarrow \emptyset$;

observe s_0;
for $k = 0, 1, 2, \ldots$:
- take action a_k ε-greedy$(Q(s_k, a))$
- observe r_k, s_{k+1};
- store (s_k, a_k, r_k, s_{k+1}) in D;
- sample (s, a, r, s') from D;
- $y \leftarrow r + \max_{a'} Q(s', a')$;
- $Q(s, a) \leftarrow (1 - \alpha_k) Q(s, a) + \alpha_k y$

Behavior and target policies

The policy interacting with environment (collecting data) is called behavior policy;

Behavior and target policies

The policy interacting with environment (collecting data) is called **behavior policy**;

The trained policy that will be outputted by algorithm is called **target policy**;

Behavior and target policies

The policy interacting with environment (collecting data) is called **behavior policy**;

The trained policy that will be outputted by algorithm is called **target policy**;

If we learn Q using samples from , we are **on-policy**. If , we are **off-policy**.

Behavior and target policies

The policy interacting with environment (collecting data) is called **behavior policy**;

The trained policy that will be outputted by algorithm is called **target policy**;

If we learn Q using samples from , we are **on-policy**. If , we are **o -policy** .

Q-learning

o -policy

(arbitrary can be used)

Vanilla version: : " -greedy Q $s; a$ q

Monte Carlo

on-policy

(is required)

Experience replay is useless!

Q-learning is o -policy

Data source	Behavior policy	Convergence
Online experience	In nite visitation	Q
Expert data	-	Q
Experience replay	In nite visitation	Q

SARSA

Expectation

Safe RL

Expectation

Reality

Expectation

Reality

Consider the family of ε-soft policies:

$$\pi(a \mid s) \geq \frac{\varepsilon}{|A|}$$

Family of ε-soft policies: what changes

Policy Evaluation: nothing changes. Q is the same.

Family of ε-soft policies: what changes

Policy Evaluation: nothing changes Q is the same.

Optimal value function:

$$Q_{\varepsilon\text{-greedy}}(s,a) = \max_{P \in \varepsilon\text{-soft}} Q_P(s,a)$$

Family of ε-soft policies: what changes

Policy Evaluation: nothing changes Q is the same.

Optimal value function:

$$Q_{\varepsilon\text{-greedy}}(s,a) = \max_{P \in \varepsilon\text{-soft}} Q(s,a)$$

Optimal policy:

$$\pi(a \mid s) = \varepsilon\text{-greedy}(Q_{\varepsilon\text{-greedy}}(s,a))$$

Family of "-soft policies : what changes

Policy Evaluation : nothing changes$Q$ is the same.

Optimal value function:

$$Q_{\text{" -greedy}}(s, a) : \max_{\text{P" -soft}} Q(s, a)$$

Optimal policy:

$$(a \mid s) \quad \text{" -greedy}(Q_{\text{" -greedy}}(s, a))$$

"-soft Policy Improvement

In the family of "-soft policies "-greedy is policy improvement:

$$\sim : \quad \text{" -greedy}(Q(s, a)) \quad \tilde{n} \quad \sim ©$$

Q-learning vs SARSA

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \left[r(s,a) + Q_k(s', a') - Q_k(s,a) \right]$$

a' must be taken from target policy

Q-learning vs SARSA

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \left[r(s,a) + Q_k(s',a') - Q_k(s,a) \right]$$

a' must be taken from target policy

	Q-learning	SARSA
Target policy:	$\arg\max_a Q_k(s,a)$	ε-greedy$(Q_k(s,a))$
Behavior policy:	ε-greedy$(Q_k(s,a))$	ε-greedy$(Q_k(s,a))$

Q-learning vs SARSA

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \left[r(s,a) + Q_k(s',a') - Q_k(s,a) \right]$$

a' must be taken from target policy

	Q-learning	SARSA
Target policy:	$\arg\max_a Q_k(s,a)$	ε-greedy$(Q_k(s,a))$
Behavior policy:	ε-greedy$(Q_k(s,a))$	ε-greedy$(Q_k(s,a))$

Q-learning vs SARSA

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \left[r(s,a) + Q_k(s',a') - Q_k(s,a) \right]$$

a' must be taken from target policy

	Q-learning	SARSA
Target policy:	$\arg\max_a Q_k(s,a)$	ε-greedy$(Q_k(s,a))$
Behavior policy:	ε-greedy$(Q_k(s,a))$	ε-greedy$(Q_k(s,a))$
Convergence:	$\pi^*(s) = \arg\max_a Q^*(s,a)$	ε-greedy$(Q^*(s,a))$

SARSA

SARSA

Initialize $Q(s, a)$ arbitrarily.

observe s_0, select a_0 randomly;
for $k = 0, 1, 2, \ldots$:

- take action a_k, observe r_k, s_{k+1};
- sample a_{k+1} " ε-greedy w.r.t. $Q(s_{k+1}, a)$
- $y \leftarrow r_k + Q(s_{k+1}, a_{k+1})$;
- $Q(s_k, a_k) \leftarrow (1 - \alpha_k) Q(s_k, a_k) + \alpha_k y$

- on-policy by construction

SARSA

SARSA

Initialize $Q(s,a)$ arbitrarily.

observe s_0, select a_0 randomly;
for $k = 0, 1, 2, \ldots$

- take action a_k, observe r_k, s_{k+1};
- sample a_{k+1} " ε-greedy w.r.t. $Q(s_{k+1}, a)$
- $y \leftarrow r_k + Q(s_{k+1}, a_{k+1})$;
- $Q(s_k, a_k) \leftarrow (1 - \alpha_k) Q(s_k, a_k) + \alpha_k y$

- on-policy by construction
- same as Q-learning If $\varepsilon = 0$
- converges to Q if $\varepsilon_k \to 0$

SARSA

SARSA

Initialize $Q(s, a)$ arbitrarily.

observe s_0, select a_0 randomly;
for $k = 0, 1, 2, \ldots$

- take action a_k, observe r_k, s_{k+1};
- sample a_{k+1} " ε-greedy $(Q(s_{k+1}, a))$
- $y \leftarrow r_k + Q(s_{k+1}, a_{k+1})$;
- $Q(s_k, a_k) \leftarrow (1 - \alpha_k) Q(s_k, a_k) + \alpha_k y$

- on-policy by construction
- same as Q-learning If $\alpha = 0$
- converges to Q^* if $\varepsilon_k \to 0$

Expected-SARSA:

$$y \leftarrow r_k + \mathbb{E}_{a_{k+1}} Q(s_{k+1}, a_{k+1})$$

SARSA is on-policy

For transition(s, a, r, s', a'):

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \left[r + Q_k(s',a') - Q_k(s,a) \right]$$

Data source	Target policy	Convergence
Online experience	Decaying exploration Persistent exploration	Q Q^{π} ε-greedy
Expert data (a' taken from buffer)	-	

SARSA is on-policy

For transition(s, a, r, s', a'):

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \left[r + Q_k(s', a') - Q_k(s,a) \right]$$

Data source	Target policy	Convergence
Online experience	Decaying exploration	Q^*
	Persistent exploration	Q^{ε} ε-greedy
Expert data (a' taken from buffer)	-	Q^{expert}
Experience replay (a' taken from buffer)		

SARSA is on-policy

For transition(s, a, r, s', a'):

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \left[r + Q_k(s', a') - Q_k(s,a) \right]$$

Data source	Target policy	Convergence
Online experience	Decaying exploration	Q^*
	Persistent exploration	Q^{π} ε-greedy
Expert data (a' taken from buffer)	-	Q^{expert}
Experience replay (a' taken from buffer)	(arbitrary)	divergence

O -policy SARSA (not really sarsa now)

Can we learn Q for given o -policy?

O -policy SARSA (not really sarsa now)

Can we learn Q for given o -policy?

For transition$ps; a; r; s^1q$ generatea^1 $pa^1 | s^1q$:

$Q_{k \ 1}ps; aq Ð Q_k ps; aq \ _k pr \ Q_k ps^1; a^1q \ Q_k ps; aqq$

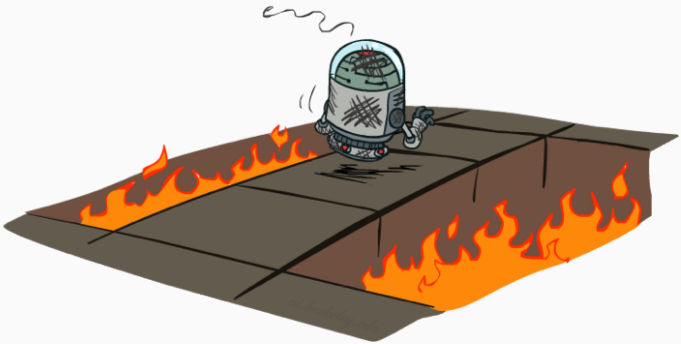
O -policy SARSA (not really sarsa now)

Can we learn Q for given o -policy?

For transition$(s; a; r; s^1)$ generatea^1 $p(a^1 | s^1)$:

$$Q_{k\ 1}(s; a) \leftarrow Q_k(s; a) \quad {}_k p r \quad Q_k(s^1; a^1) \quad Q_k(s; a)$$

Data source	Target policy	Convergence
(any, but a^1 generated online)	Decaying exploration	Q
	Persistent exploration	$Q_{"}$ -greedy



Literature

- Sutton, Barto — Reinforcement Learning, an Introduction, ch. 5-6;
- Watkins, Dayan — Technical Note, Q-learning;

Pictures from Berkeley CS 188 | Introduction to Artificial Intelligence;