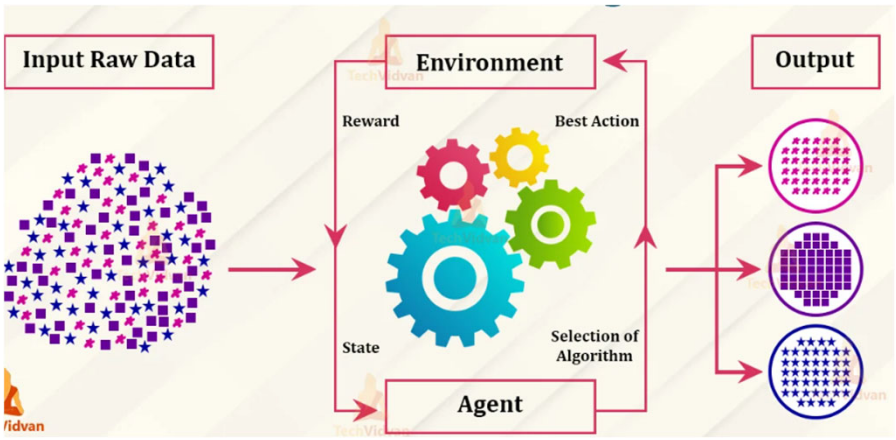
Soft computing

K.N. Toosi University of Technology

Soft Computing

یادگیری تقویتی(Reinforced Learning)

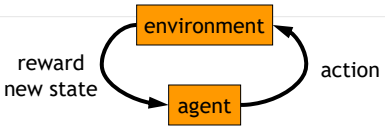


Soft Computing 2

Reinforcement Learning

انواع یادگیری تقویتی

روش	توضیح	مثال کاربردی
مبتنی بر مدل **(Model-Based)**	عامل یک مدل از محیط می‌سازد و پیش‌بینی می‌کند (ربات نقشه محیط را یاد میگیرد سپس مسیر بهینه را برنامه ریزی می کند).	بهینه‌سازی انفجارهای معدن بهینه‌سازی طراحی شیب‌ها
بدون مدل **(Model-Free)**	عامل مستقیماً و بدون مدل‌سازی با محیط تعامل دارد(ربات فقط از تجربه حرکت و گرفتن جایزه یا تنبیه، یاد می‌گیرد کدام مسیر بهتر است).	برنامه ریزی حمل و نقل مثل تردد تراک‌ها در معادن
Q-Learning	از یک جدول (Q-Table) برای ذخیره ارزش هر عمل در هر حالت استفاده می‌کند.	مدیریت انرژی، ترافیک هوشمند بهینه‌سازی تزریق آب در مخازن نفت
Deep RL	ترکیب RL با شبکه‌های عصبی عمیق (مثل DQN:Deep Q-Network)	بهینه‌سازی شکست هیدرولیکی در مخازن شیل خودروهای خودران (Tesla)



فهرست مطالب

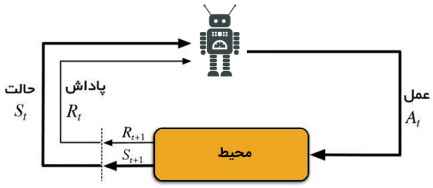
- تعریف یادگیری تقویتی و مثال
- فرایند تصمیم گیری مارکوف Markov Decision Processes
- روشهای حل
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning
- Q-learning
- SARSA

Reinforcement Learning

more general than supervised/unsupervised learning
learn from interaction w/ environment to achieve a goal

مفاهیم

- ✓ **عامل (Agent)** : موجودی که تصمیم‌گیری می‌کند (مثلاً یک ربات یا برنامه کامپیوتری).
- ✓ **محیط (Environment)** : دنیایی که عامل در آن عمل می‌کند (مثلاً یک بازی ویدیویی یا سیستم رانندگی خودکار).



- ✓ **حالت (State)** : وضعیت فعلی محیط (مثلاً موقعیت ربات)
- ✓ **عمل یا اقدام (Action)** : تصمیمی که عامل می‌گیرد (مثلاً حرکت به چپ یا راست).
- ✓ **پاداش (Reward)** : بازخورد عددی که محیط پس از هر عمل به عامل می‌دهد (مثلاً +۱ برای برد و −۱ برای باخت).
- ✓ **سیاست (Policy)** : استراتژی عامل برای انتخاب عمل‌ها در هر حالت (مثلاً اگر در حالت X هستی، عمل Y را انجام بده).

Reinforcement Learning

مفاهیم

- ✓ **اقدام (Action)** : شامل تمامی واکنش‌های محتملی است که عامل تصمیم‌گیرنده ممکن است در مواجهه با وضعیت ایجاد شده، از خود نشان دهد.
- ✓ اقدام‌ها با توجه به متناهی بودن و یا نامتناهی بودنشان به دو نوع اپیزودیک و یا مستر (غیر اپیزودیک) تقسیم‌بندی می‌شوند.
- ✓ اقدام اپیزودیک: دارای یک نقطه آغاز و یا پایان است. یک بازی را در نظر بگیرید که در آن بازی‌کننده کامپیوتری ممکن است بمیرد و یا به پایان مرحله برسد. در چنین حالتی می‌توان یک لیستی از وضعیت‌ها، اقدامات و یا پاداش‌ها داشت.
- ✓ اقدام مستمر یا غیر اپیوزدیک: نقطه پایانی برای اقدامات وجود ندارد. مثل تجارت سهام تا زمانی که عامل توسط نیروی خارجی (مثل کاربر انسانی) متوقف نشود، همواره به یادگیری و انجام اقدام مناسب به تعامل با محیط می‌پردازد.

ماتریس Q یا (Q-Table) در یادگیری تقویتی به ما می‌گوید که در یک **حالت (state)** مشخص، انتخاب هر **عمل (action)** چه پاداشی در بلندمدت (با در نظر گرفتن پاداش‌های آینده) دارد.

یادگیری تقویتی

در یادگیری تقویتی

عامل با انجام اقدامات مختلف و دریافت پاداش یا جریمه از محیط، سعی می‌کند یک سیاست بهینه پیدا کند که حداکثر پاداش تجمعی را در طول زمان به دست آورد.

مثال در بازی شطرنج:

پاداش: +۱ برای برد، −۱ برای باخت، ۰ برای تساوی.

حالت: موقعیت مهره‌ها روی صفحه.

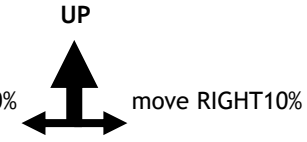
عمل: حرکت یک مهره.

عامل با میلیون‌ها بار بازی کردن، یاد می‌گیرد که چه حرکاتی شانس برد را افزایش می‌دهند.

حرکت یک ربات در اتاق

			+1
			-1
START			

Actions: UP, DOWN, LEFT, RIGHT

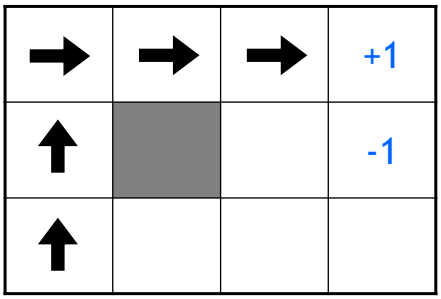


- reward +1 at [4,3], -1 at [4,2]
- reward -0.1 for each step

- what's the strategy to achieve max reward?
- what if the actions were deterministic?

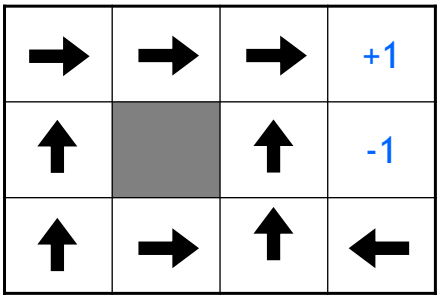
حرکت یک ربات در اتاق

Is this a solution?



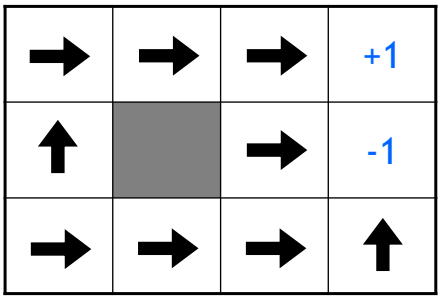
- only if actions deterministic
  - not in this case (actions are stochastic)
- solution/policy
  - mapping from each state to an action

Optimal policy

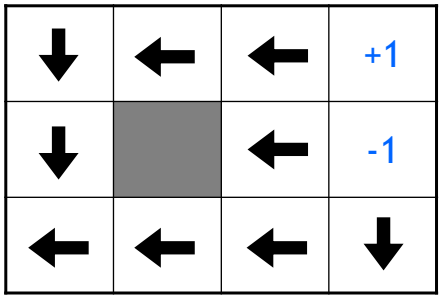


Reward for each step: -0.1

تاثیر پاداش



Reward for each step: -2



Reward for each step: +0.1

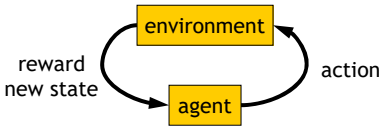
فهرست مطالب

- تعریف یادگیری تقویتی و مثال
- فرایند تصمیم گیری مارکوف Markov Decision Processes
- روشهای حل
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning
  - Q-learning
  - SARSA

Markov Decision Process (MDP)

فرآیندهای تصمیم‌گیری مارکوف : مدل تصمیم‌گیری متوالی در محیط‌های تصادفی است که پایه‌ی نظری بسیاری از الگوریتم‌های یادگیری تقویتی را تشکیل می‌دهد.

- **S (States)**: **مجموعه حالت‌ها** که درمحیط ممکن است (مثلاً موقعیت ربات در یک نقشه)
- **A (Actions)**: **مجموعه اقدامات** که قابل انجام توسط عامل است (مثلاً حرکت به چپ/راست)
- **P (Transition Probability)**: **تابع انتقال** که احتمال گذار بین حالتها است
  $P(s'|s,a) =$ احتمال رفتن به حالت s' اگر در حالت s عمل a انجام شود
- **R (Reward Function)**: **تابع پاداش**
  $R(s,a,s') =$ پاداش دریافت‌شده پس از گذار از s به s' با عمل a



خاصیت مارکوف (فرض مارکوف): وضعیت آینده فقط به حالت فعلی و عمل فعلی بستگی دارد، نه به تاریخچه گذشته

$$P(s_{t+1} \mid s_t, a_t) = P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots)$$

Markov Decision Process (MDP)

مؤلفه‌های اضافی: ضریب تخفیف، استراتژی

γ (Discount Factor): ضریب تخفیف (بین ۰ تا ۱) برای ارزش‌دهی به پاداش‌های آینده

π (Policy): استراتژی عامل برای انتخاب عمل در هر حالت

$\pi(a|s) =$ احتمال انتخاب عمل a در حالت s

هدف MDP: یافتن استراتژی است

یافتن استراتژی یا سیاست بهینه (π^*)که مقدار بازده تجمعی مورد انتظار را حداکثر کند

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) \mid s_t = s\right]$$

تابع ارزش (تخمین تمام پاداشهای آتی) در یک حالت s

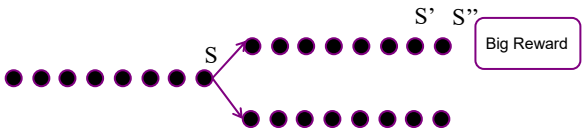
حل MDP: روش‌های مختلف از جمله **برنامه‌ریزی پویا**

الگوریتم ارزش‌تکرار (Value Iteration):

$$V_{k+1}(s) = \max_a \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V_k(s')]$$

تاخیر در پاداش

تاخیر در پاداش یعنی اقدام فعلی عامل ممکن است هم‌اکنون پاداش نداشته باشد یا حتی منفی باشد، ولی در آینده منجر به پاداش مطلوب شود (مثل قربانی کردن مهره در شطرنج).



عامل باید یاد بگیرد که:
فقط به پاداش فوری نگاه نکند، بلکه ارزش تجمعی آینده را نیز در نظر بگیرد.

راه حل:

- ✓ استفاده از تخفیف پاداش‌ها (پارامتر γ)
- ✓ استفاده از روش‌های مبتنی بر ارزش (Value-Based) مثل Q-Learning
- ✓ روش‌های سیاست‌محور (Policy Gradient) با حافظه بلندمدت

سیاست قطعی و آماری

$\pi(s)$ سیاست قطعی: اگر گرسنه‌ای (s)، همیشه غذا بخور(a)
$\pi(a|s)$ سیاست آماری: اگر گرسنه‌ای (s)، با احتمال ۰.۷ غذا بخور، با احتمال ۰.۳ بخواب

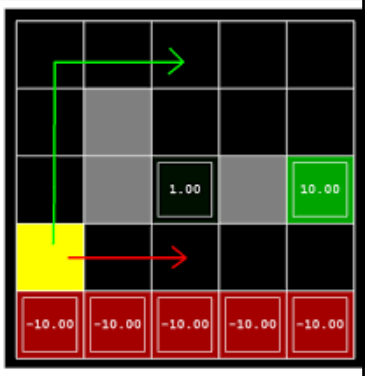
سیاست قطعی: برای محیط ساده و کاملاً قابل پیش‌بینی نظیر الگوریتم DDPG (یادگیری تقویتی برای محیط‌های پیوسته که ترکیبی از یادگیری عمیق و استراتژی قطعی است)

سیاست آماری: برای محیط تصادفی یا پیچیده (مثلاً با پاداش تأخیری یا نویز)

- ✓ حتی در الگوریتم‌های قطعی مثل DDPG در مرحله آموزش معمولاً نویز تصادفی اضافه می‌شود تا اکتشاف بیشتر انجام شود.

MDP مثال

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) \mid s_t = s\right]$$



four preference scenarios:
(a) Prefer the close exit (+1), risking the cliff (-10)
(b) Prefer the close exit (+1), but avoiding the cliff (-10)
(c) Prefer the distant exit (+10), risking the cliff (-10)
(d) Prefer the distant exit (+10), avoiding the cliff (-10)

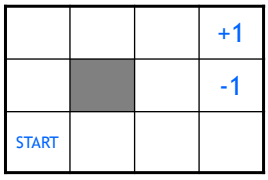
four parameter settings:

(1) γ=0.1,	noise = 0.5	(a)
(2) γ=0.1,	noise = 0	(b)
(3) γ=0.99,	noise = 0.5	(c)
(4) γ= 0.99,	noise = 0	(d)

Soft Computing 17

Markov Decision Process (MDP)

- set of states S, set of actions A, initial state S_0
- transition model P(s,a,s')
  - P([1,1], up, [1,2]) = 0.8
- reward function r(s)
  - r([4,3]) = +1
- goal: maximize cumulative reward in the long run



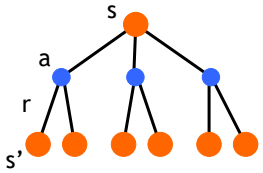
- policy: mapping from S to A
  - π(s) or π(s,a) (deterministic vs. stochastic)
- reinforcement learning
  - transitions and rewards usually not available
  - how to change the policy based on experience
  - how to explore the environment

محاسبه پاداش

- additive rewards — پاداش تجمعی
  - $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  - infinite value for continuing tasks
- discounted rewards — پاداش تجمعی تخفیف یافته
  - $V(s_0, s_1, \ldots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$
  - value bounded if rewards bounded

Value functions تابع ارزش

- state value function: $V^{\pi}(s)$ — تابع ارزش حالت
  - expected return when starting in *s* and following π
- state-action value function: $Q^{\pi}(s,a)$ — تابع ارزش حالت – اقدام
  - expected return when starting in *s*, performing *a*, and following π
- useful for finding the optimal policy
  - can estimate from experience
  - pick the best action using $Q^{\pi}(s,a)$



- Bellman equation — فرمول بلمن

$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^{\pi}(s') \right] = \sum_a \pi(s,a) Q^{\pi}(s,a)$$

تابع ارزش حالت- اقدام رابر اساس پاداش فعلی و ارزش آینده محاسبه می کند

تابع ارزش بهینه

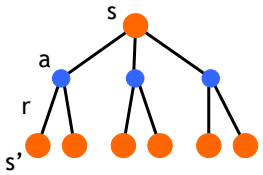
- there's a set of *optimal* policies
  - V^{π} defines partial ordering on policies
  - they share the same optimal value function

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- Bellman optimality equation

$$V^*(s) = \max_{a} \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^*(s') \right]$$

  - solve for V*(s)
  - easy to extract the optimal policy



- having Q*(s,a) makes it even simpler

$$\pi^*(s) = \arg\max_{a} Q^*(s,a)$$

سیاست بهینه عملی را انتخاب می‌کند که بیشترین مقدار تابع ارزش حالت-اقدام بهینه را داشته باشد

این فرمول پایه بسیاری از الگوریتم‌های یادگیری تقویتی مانند Q-learning و DQN است

فهرست مطالب

- تعریف یادگیری تقویتی و مثال
- فرایند تصمیم گیری مارکوف Markov Decision Processes
- روشهای حل
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning
  - Q-learning
  - SARSA

برنامه ریزی پویا

DP: Dynamic Programming

برنامه ریزی پویا (DP) الگوریتم‌هایی که از معادلات بازگشتی (مانند معادلات بلمن) استفاده می‌کنند تا تابع ارزش و سیاست بهینه را پیدا کند.

- ✓ تابع ارزش بهینه $V^*(s)$ یا تابع ارزش حالت-عمل بهینه $Q^*(s,a)$
- ✓ سیاست بهینه $\pi^*(s)$

پیش‌نیازهای استفاده از برنامه‌ریزی پویا(DP)

1. **مشخص بودن کامل مدل**: یعنی دانستن تابع انتقال $P(s'|s,a)$ و تابع پاداش $R(s,a)$
2. **فضای حالت و اقدام محدود**: الگوریتم‌های DP معمولاً به حافظه و زمان زیادی نیاز دارند.

Policy evaluation/improvement

- policy evaluation: π -> V^π
  - با نوشتن معادلات بلمن برای یک محیط با n حالت n معادله بدست می‌آید که با حل آنها توابع ارزش برای هر حالت تعیین می شود
    روش تکرار برای حل معادلات استفاده می شود.

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{k'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

  - start with an arbitrary value function V_0, iterate until V_k converges

- policy improvement: V^π -> π'
  - برای یک محیط با n خالت و m اقدام n*m معادله بدست می‌آید که با حل آنها توابع ارزش برای هرحالت- اقدام تعیین می شود
    .

$$\pi'(s) = \arg\max_a Q^\pi(s,a)$$
$$= \arg\max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^\pi(s') \right]$$

  - π' either strictly better than π, or π' is optimal (if $\pi = \pi'$)

روش مونت کارلو

MC Methods: Monte Carlo methods

روش مونت کارلو : بر پایه‌ی اجرای اپیزودهای کامل و سپس استفاده از **میانگین پاداش‌های مشاهده شده** برای تخمین ارزش حالت‌ها یا جفت‌های حالت-اقدام است.

مراحل

1.اجرای سیاست مشخص π در محیط به‌صورت چندین اپیزود.

2. برای هر حالت s، پاداش تجمعی آینده G_t را محاسبه می شود:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$$

3. میانگین پاداش‌های تجمعی حالت s را در هر بار دیدن آن، محاسبه کنید:

$$V(s) = \text{average of all } G_t \text{ when } s \text{ is visited}$$

4.در نهایت، V(s) به صورت میانگین تجربی تخمین زده می‌شود.

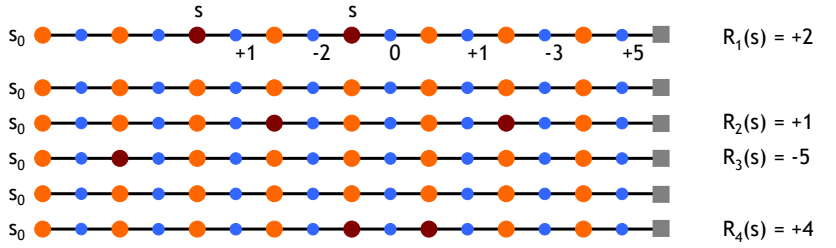
مزایا:
- ✓ ساده و شهودی
- ✓ بدون نیاز به مدل محیط
- ✓ قابل استفاده برای تخمین مستقیم سیاست

معایب:
- ➢ فقط در محیط‌های اپیزودیک قابل استفاده است
- ➢ همگرایی کندتر نسبت به روش‌های مبتنی بر بوت‌استرپ
- ➢ ناپایدار بودن در حالت تغییرات زیاد

روش مونت کارلو- ارزیابی سیاست

- want to estimate $V^{\pi}(s)$
  = expected return starting from s and following π
  - estimate as average of observed returns in state s

- first-visit MC
  - average returns following the first visit to state s



$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$

یادگیری اختلاف زمانی

TD Learning: Temporal Difference Learning

در این روش به جای اینکه صبر کنیم تا اپیزود تمام شود، در همان لحظه‌ای گرفتن پاداش، تخمین به‌روزرسانی می‌شود.

- combines ideas from MC and DP
  - like MC: learn directly from experience (don't need a model)
  - like DP: learn from values of successors
  - works for continuous tasks, usually faster than MC
- constant-alpha MC:
  - have to wait until the end of episode to update

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

target

- simplest TD
  - update after every step, based on the successor

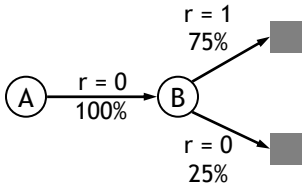
$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

MC vs. TD

- observed the following 8 episodes:

A - 0, B - 0	B - 1	B - 1	B - 1
B - 1	B - 1	B - 1	B - 0

- MC and TD agree on V(B) = 3/4
- MC: V(A) = 0
  - converges to values that minimize the error on training data
- TD: V(A) = 3/4
  - converges to ML estimate of the Markov process



فهرست مطالب

- تعریف یادگیری تقویتی و مثال
- فرایند تصمیم گیری مارکوف Markov Decision Processes
- روشهای حل
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal-Difference learning
  - Q-learning
  - SARSA

Q-learning

یادگیری Q : الگوریتم یادگیری تقویتی بدون سیاست (off-policy) که به عامل اجازه می‌دهد بدون نیاز به مدل محیط یاد بگیرد که در هر وضعیت چه عملی را انتخاب کند تا در بلندمدت پاداش بیشتری دریافت کند.

مقدار تابع Q را برای هر زوج (state, action) تخمین می‌زند و به‌روز می‌کند. تابع Q به عامل کمک می‌کند بهترین تصمیم را بگیرد.

- Q-learning: off-policy
  - use any policy to estimate Q

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

  - Q directly approximates Q* (Bellman optimality equation)
  - independent of the policy being followed
  - only requirement: keep updating each (s,a) pair

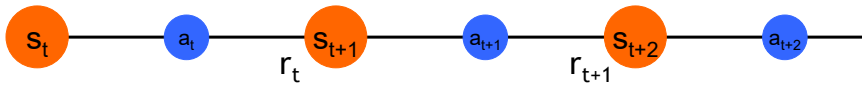
مزایا:
- ✓ بدون نیاز به مدل محیط
- ✓ الگوریتم پایه برای یادگیری عمیق
- ✓ ساده و قابل پیاده‌سازی آسان

معایب:
- نیاز به حافظه زیاد برای حالات بزرگ
- کند بودن در محیط‌های پیوسته یا بزرگ
- گاهی ناپایدار در محیط‌های پیچیده

SARSA: State,Action, Reward, State, Action

SARSAیک الگوریتم یادگیری **بر پایه سیاست (on-policy)** است که از تجربه‌ی واقعی در طول حرکت عامل (Agent) استفاده می‌کند، و تخمین تابع ارزش **Q(s,a)** را با استفاده از مشاهدات به‌روزرسانی می‌کند (یادگیری اختلاف زمانی).

- need Q(s,a), not just V(s)



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- control
  - start with a random policy
  - update Q and π after each step
  - again, need ε-soft policies

مزایا:
- ✓ یادگیری مبتنی بر تجربه‌ی واقعی و هم‌راستا با رفتار عامل
- ✓ مناسب برای محیط‌های نویزی و پیچیده
- ✓ ساده و قابل پیاده‌سازی آسان

معایب:
- در مقایسه با Q-Learningممکن است به جواب بهینه‌ی کمتری برسد (زیرا از سیاست جاری استفاده می‌کند، نه بهترین سیاست ممکن)

Deep Reinforcement Learning

یادگیری تقویتی عمیق (Deep Reinforcement Learning)یعنی استفاده از شبکه‌های عصبی عمیق به عنوان تابع تقریبی در الگوریتم‌های یادگیری تقویتی برای حل مسائل پیچیده که فضای حالت آن‌ها بسیار بزرگ یا پیوسته است.

در یادگیری تقویتی کلاسیک، اگر فضای حالت بزرگ باشد، نگه داشتن Q-table غیرممکن می‌شود. در اینجا، یادگیری عمیق کمک می‌کند تا:
- به‌جای ذخیره Qدر جدول، یک **شبکه عصبی** مقدار Q(s, a)را تقریب بزند.
- با این روش، می‌توان مسائل با فضای حالت بسیار زیاد مانند بازی‌ها، رباتیک و رانندگی خودکار را حل کرد.

الگوریتم	توضیح
DQN (Deep Q-Network)	از شبکه عصبی برای تخمین Qاستفاده می‌کند
DDPG(Deep Deterministic Policy Gradient)	مناسب برای اعمال پیوسته (Continuous Actions)
A3C(Asynchronous Advantage Actor-Critic) / **A2C**(Advantage Actor-Critic)	الگوریتم‌های Actor-Criticبرای بهبود پایداری
PPO (Proximal Policy Optimization)	الگوریتم پایدار و قدرتمند برای یادگیری سیاست

مثال

مثال: تغییرات آب‌وهوا به روش فرایند تصمیم مارکوف بین سه حالت **"آفتابی"**، **"ابری"** و **"بارانی"**

```
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict

# 1. تعریف حالت‌ها و ماتریس انتقال
states = ["Sunny", "Claudy", "Rainy"]
transition_matrix = np.array([
    [0.7, 0.2, 0.1],  # از آفتابی به [آفتابی, ابری, بارانی]
... # از ابری به [۰.۳, ۰.۴, ۰.۳]
... # از بارانی به [۰.۲, ۰.۳, ۰.۵]
])

# ۲. تابع تولید دنباله مارکوف
def generate_markov_sequence(initial_state, steps=100):
    current_state = initial_state
    sequence = [current_state]
    state_indices = {s: i for i, s in enumerate(states)}

    for _ in range(steps):
        prob = transition_matrix[state_indices[current_state]]
        current_state = np.random.choice(states, p=prob)
        sequence.append(current_state)

    return sequence
```

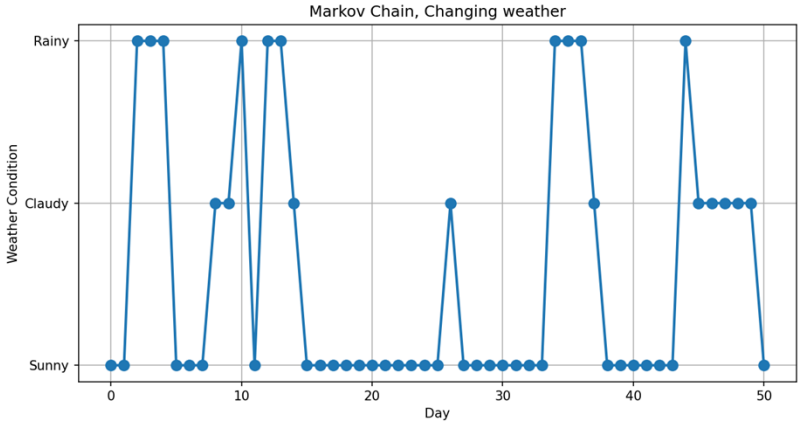
14 RL.py

مثال

```
# ۳. تولید دنباله و ذخیره نتایج
np.random.seed(42)  # برای تکرارپذیری
sequence = generate_markov_sequence("Sunny", 50)

# 4. تبدیل حالت‌ها به اعداد برای رسم نمودار
state_to_num = {"Sunny": 0, "Claudy": 1, "Rainy": 2}
numeric_sequence = [state_to_num[s] for s in sequence]

# 5. رسم نمودار
plt.figure(figsize=(10, 5))
plt.plot(numeric_sequence, 'o-', markersize=8, linewidth=2)
plt.yticks([0, 1, 2], states)
plt.xlabel("Day")
plt.ylabel("Weather Condition")
plt.title("Markov Chain, Changing weather")
plt.grid(True)
plt.show()
```

یک نمودار خطی تولید می‌شود که محور **x** نشان‌دهنده روزها و
محور **y** وضعیت آب‌وهوا را نمایش می‌دهد

نتیجه مثال



Soft Computing 35

مثال:پیش بینی قیمت طلا در شش ماه آینده

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime, timedelta

# 1. Generate synthetic historical data ending at $3360 today
np.random.seed(42)
days_history = 365  # 1 year of historical data
dates = pd.date_range(end=datetime.today(), periods=days_history, freq='D')

# Generate random walk with $3360 as last price
random_walk = np.cumsum(np.random.normal(0, 5, days_history))
gold_prices = random_walk - random_walk[-1] + 3360  # Force last price to be $3360

# 2. Define states based on daily % changes
def get_state(price_change):
    if price_change < -0.5:
        return "Sharp Drop"
    elif -0.5 <= price_change < 0:
        return "Mild Drop"
    elif 0 <= price_change < 0.5:
        return "Mild Rise"
    else:
        return "Sharp Rise"

states = ["Sharp Drop", "Mild Drop", "Mild Rise", "Sharp Rise"]

# 3. Calculate transition matrix
price_changes = np.diff(gold_prices) / gold_prices[:-1] * 100
state_sequence = [get_state(change) for change in price_changes]
```

مثال:پیش بینی قیمت طلا در شش ماه آینده

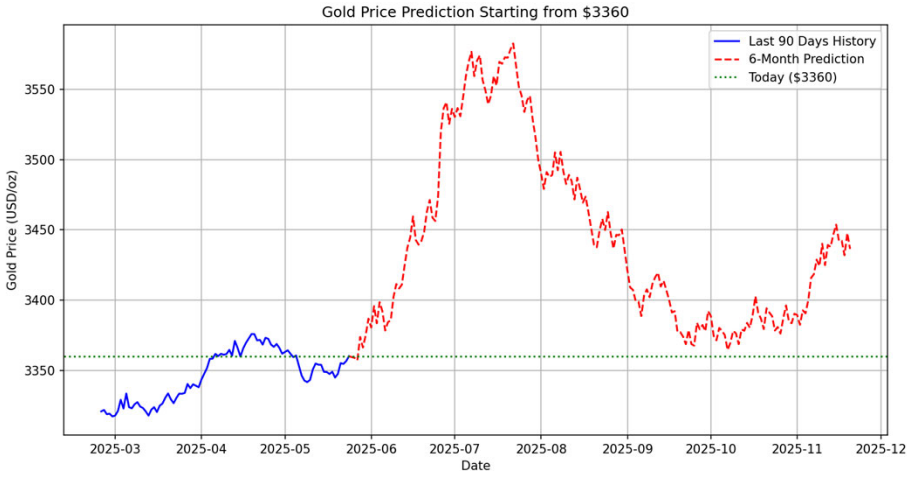
```
# Initialize and populate transition matrix
transition_matrix = np.zeros((4, 4))
state_indices = {s: i for i, s in enumerate(states)}
for i in range(len(state_sequence)-1):
    current = state_indices[state_sequence[i]]
    next_ = state_indices[state_sequence[i+1]]
    transition_matrix[current, next_] += 1
# Normalize rows to get probabilities
transition_matrix /= transition_matrix.sum(axis=1, keepdims=True)
# 4. Prediction function
def predict_future(last_price, transition_matrix, days=180):
    current_state = get_state(0)  # Start from no-change state
    predicted_prices = [last_price]
    for _ in range(days):
        # Get next state probabilities
        probs = transition_matrix[state_indices[current_state]]
        next_state = np.random.choice(states, p=probs)

        # Generate price change based on state
        if next_state == "Sharp Drop":
            change = np.random.uniform(-2.0, -0.5)
        elif next_state == "Mild Drop":
            change = np.random.uniform(-0.5, 0)
        elif next_state == "Mild Rise":
            change = np.random.uniform(0, 0.5)
        else:  # Sharp Rise
            change = np.random.uniform(0.5, 2.0)
        new_price = predicted_prices[-1] * (1 + change/100)
        predicted_prices.append(new_price)
        current_state = next_state

    return predicted_prices
```

14 RL Gold 6 month.py

مثال:پیش بینی قیمت طلا در شش ماه آینده

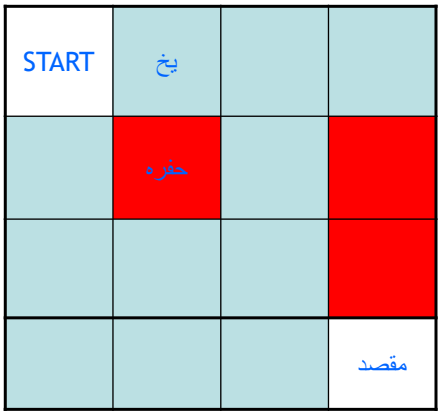
```
# 5. Run prediction starting from $3360
predicted_prices = predict_future(3360, transition_matrix)

# 6. Plot results
future_dates = pd.date_range(start=dates[-1], periods=181, freq='D')

plt.figure(figsize=(12, 6))
plt.plot(dates[-90:], gold_prices[-90:], label='Last 90 Days
History', color='blue')
plt.plot(future_dates, predicted_prices, label='6-Month Prediction',
color='red', linestyle='--')
plt.axhline(y=3360, color='green', linestyle=':', label='Today
($3360)')
plt.xlabel('Date')
plt.ylabel('Gold Price (USD/oz)')
plt.title('Gold Price Prediction Starting from $3360')
plt.legend()
plt.grid(True)
plt.show()

# Print key metrics
print(f"Predicted price after 6 months: ${predicted_prices[-1]:.2f}")
print(f"Maximum predicted price: ${max(predicted_prices):.2f}")
print(f"Minimum predicted price: ${min(predicted_prices):.2f}")
```

پیش بینی قیمت طلا در شش ماه آینده



مثال SARSA

محیط FrozenLake یک ماتریس ۴×۴ از یخ و حفره است. هدف رسیدن به خانه هدف بدون افتادن در حفره است.



نصب کتابخانه‌های مورد نیاز

```
pip install gym matplotlib numpy
```

استفاده از محیط **FrozenLake** که یکی از محیط‌های کلاسیک برای آموزش الگوریتم‌های RL است

مثال SARSA

```
import numpy as np
import matplotlib.pyplot as plt
import gymnasium as gym  # Using the newer gymnasium package

class SafeTimeLimit(gym.Wrapper):
    """Custom time limit wrapper to avoid import issues"""
    def __init__(self, env, max_episode_steps=None):
        super().__init__(env)
        self._max_episode_steps = max_episode_steps
        self._elapsed_steps = 0

    def step(self, action):
        obs, reward, terminated, truncated, info =
self.env.step(action)
        self._elapsed_steps += 1

        if self._elapsed_steps >= self._max_episode_steps:
            truncated = True

        return obs, reward, terminated, truncated, info

    def reset(self, **kwargs):
        self._elapsed_steps = 0
        return self.env.reset(**kwargs)
```

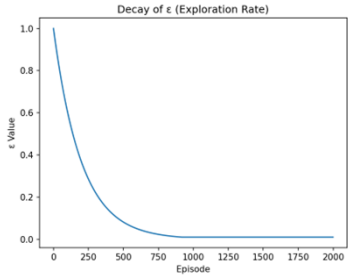
14 RL SARSA.py

مثال SARSA

```
# Initialize environment with custom time limitenv = gym.make('FrozenLake-v1',
is_slippery=True,
render_mode='rgb_array')
env = SafeTimeLimit(env, max_episode_steps=100)  # Using our custom wrapper

# Q-learning parameters
state_size = env.observation_space.n
action_size = env.action_space.n
Q = np.zeros((state_size, action_size))

# Hyperparameters
alpha = 0.8
gamma = 0.95
epsilon = 1.0
epsilon_min = 0.01
epsilon_decay = 0.995
episodes = 2000
```

نرخ یادگیری: چقدر از اطلاعات جدید را در مقایسه با مقدار قدیمی بپذیریم – مقدار بیشتر یادگیری سریعتر

انتخاب تصادفی اولیه برای اکتشاف
حداقل مقدار برای اکتشاف
کاهش تدریجی در پایان هر اپیزود



سیاست ε-Greedy : ترکیبی هوشمندانه از اکتشاف و بهره‌برداری:

- با احتمال ε یک عمل تصادفی انتخاب می‌شود (**اکتشاف**)
- با احتمال ۱-ε عمل بهینه انتخاب می‌شود (**بهره‌برداری**)

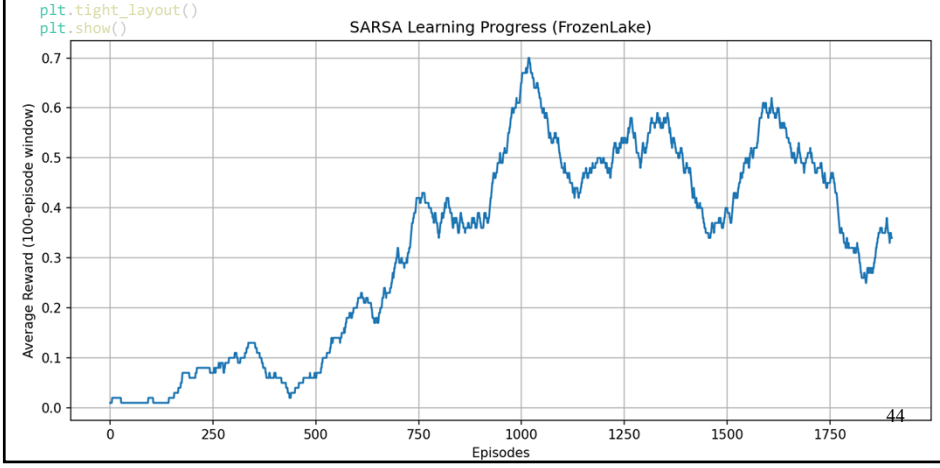
SARSA مثال

```
# Training loop
rewards = []
for episode in range(episodes):
    state, _ = env.reset()
    total_reward = 0

    # Initial action selection
    action = env.action_space.sample() if np.random.rand() < epsilon else np.argmax(Q[state])
    while True:
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated

        # Next action selection
        next_action = (env.action_space.sample() if np.random.rand() < epsilon
                       else np.argmax(Q[next_state]))
        # SARSA update
        Q[state, action] += alpha * (reward + gamma * Q[next_state, next_action] - Q[state, action])

        state, action = next_state, next_action
        total_reward += reward
        if done:
            break
    # Update exploration rate
    epsilon = max(epsilon_min, epsilon * epsilon_decay)
    rewards.append(total_reward)
```

Soft Computing 43

SARSA مثال

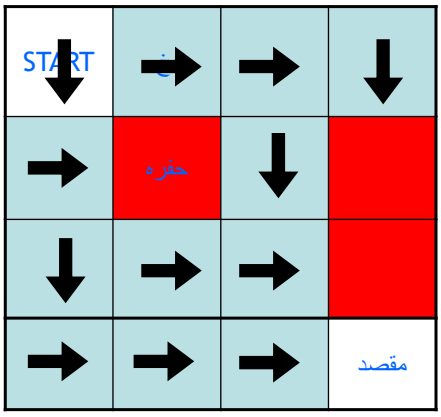
```
# Plot results
window_size = 100
moving_avg = np.convolve(rewards, np.ones(window_size)/window_size, mode='valid')

plt.figure(figsize=(10, 5))
plt.plot(moving_avg)
plt.title("SARSA Learning Progress (FrozenLake)")
plt.xlabel("Episodes")
plt.ylabel(f"Average Reward ({window_size}-episode window)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



44

مثال SARSA

مسیر بهینه یادگیری شده توسط SARSA



1. به دلیل لغزنده بودن محیط (is_slippery=True)، حرکات همیشه دقیقاً در جهت انتخاب شده انجام نمی‌شوند
2. مسیر نشان داده شده، مسیر بهینه‌ای است که عامل پس از یادگیری انتخاب می کند
3. احتمال موفقیت نهایی معمولاً بین ۶۰-۸۰٪ خواهد بود

Soft Computing 45

تمرین برنامه نویسی

تمرین : بهینه‌سازی برنامه‌ریزی تعمیر و نگهداری یک سازه بتنی با استفاده از الگوریتم‌های یادگیری تقویتی Q-learning، SARSA، Monte Carlo

- سازه در هر دوره‌ی زمانی می‌تواند در یک سطح خرابی خاص (state) باشد:
  - سالم،
  - نیمه‌خراب،
  - شدیداً خراب
- در هر وضعیت، تصمیم(action) می‌تواند یکی از گزینه‌های زیر باشد:
  - هیچ کاری انجام نده
  - تعمیر جزئی
  - تعمیر اساسی
- هدف: کاهش مجموع هزینه‌ها در طول زمان (شامل هزینه تعمیر + ریسک خرابی شدید).
- پاداش بر اساس هزینه‌ها و ریسک‌ها تعریف می‌شود.
- با استفاده از الگوریتم‌های یادگیری تقویتی، سیاست بهینه‌ای برای تصمیم‌گیری در هر وضعیت به‌دست می‌آید.
- **خروجی مورد انتظار**
  - ماتریس Qبرای هر الگوریتم
  - سیاست بهینه (جدولی یا نموداری)
  - نمودار مقایسه هزینه تجمعی بین روش‌ها

Soft Computing 46