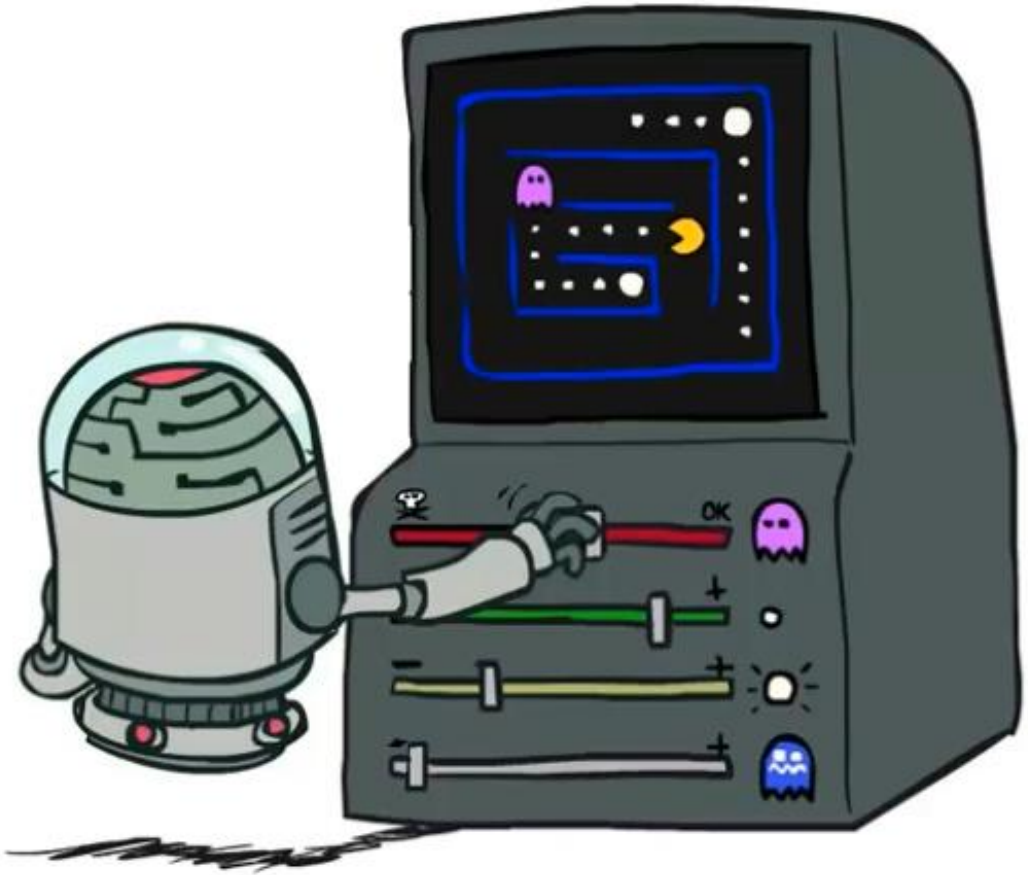
یادگیری تقویتی

سید ناصر رضوی n.razavi@tabrizu.ac.ir

۱۳۹۵

یادگیری تقویتی



یادگیری تقویتی: یادآوری

- هنوز یک فرآیند تصمیم مارکوف داریم:
  - یک مجموعه از حالت‌ها $s \in S$
  - یک مجموعه از اعمال $a \in A$
  - یک مدل $T(s,a,s')$
  - یک تابع پاداش $R(s,a,s')$
- هنوز به دنبال یک سیاست $\pi(s)$ هستیم.
- تفاوت. توابع T و R ناشناخته هستند.
  - برای یادگیری باید عمل‌های مختلف و حالت‌های نتیجه شده را آزمایش کنیم.
- ایده‌ی اصلی. محاسبه‌ی میانگین روی T با استفاده از نمونه‌ها.

تا کنون: MDP و RL

MDP شناخته شده: راه‌حل آفلاین

هدف	روش
محاسبه‌ی V^*، Q^*، π^*	الگوریتم تکرار مقدار / سیاست
ارزیابی سیاست ثابت π	ارزیابی سیاست

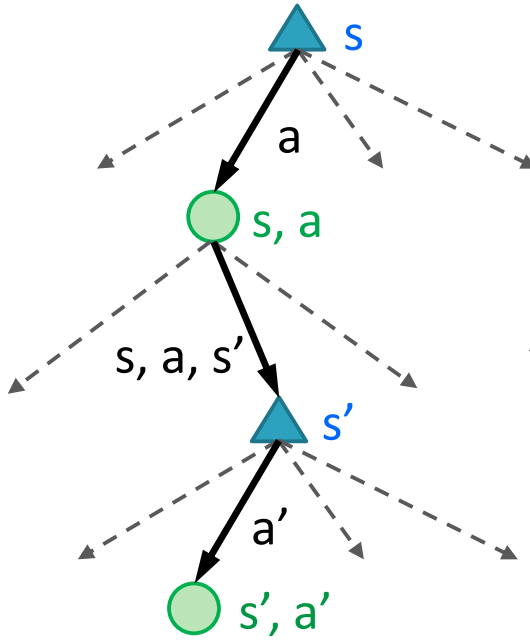
MDP ناشناخته: مبتنی بر مدل

هدف	روش
محاسبه‌ی V^*، Q^*، π^*	الگوریتم VI/PI روی مدل تقریبی
ارزیابی سیاست ثابت π	ارزیابی سیاست روی مدل تقریبی

MDP ناشناخته: مستقل از مدل

هدف	روش
محاسبه‌ی V^*، Q^*، π^*	یادگیری Q
ارزیابی سیاست ثابت π	یادگیری مقدار

تکرار مقدار Q



- تکرار مقدار. محاسبه‌ی ارزش حالت‌ها به صورت تکرار شونده
  - با بردار $V_0(s) = 0$ شروع کن (که می‌دانیم درست است).
  - در هر تکرار، با داشتن بردار $V_k(s)$، بردار $V_{k+1}(s)$ را محاسبه کن.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

- اما مقادیر Q مفیدتر هستند، پس آنها را محاسبه کن.
  - با $Q_0(s,a) = 0$ شروع کن (که می‌دانیم درست است).
  - در هر تکرار، با داشتن بردار $Q_k(s,a)$ بردار $Q_{k+1}(s,a)$ را محاسبه کن.

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

الگوریتم یادگیری Q

- یادگیری Q. الگوریتم تکرار مقدار Q مبتنی بر نمونه‌برداری

$$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

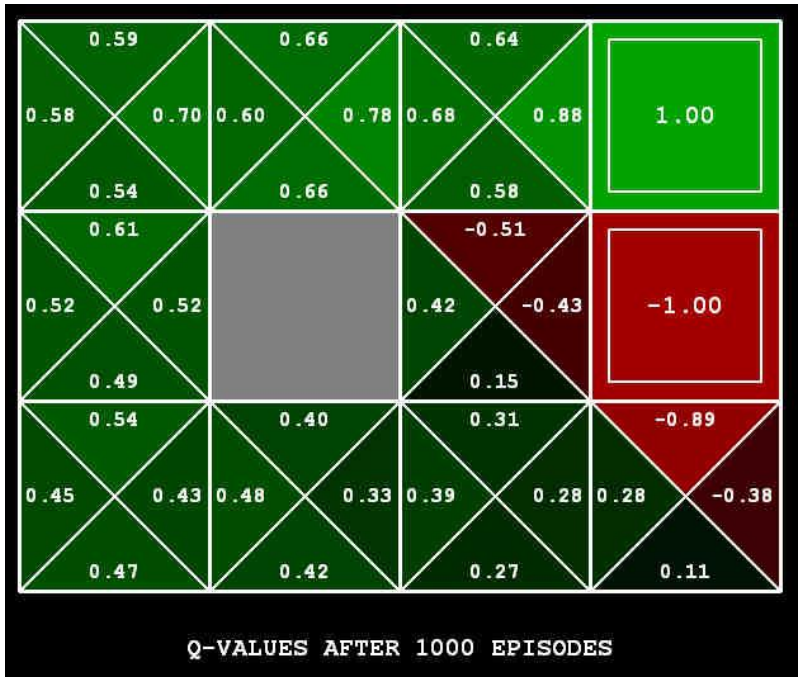
← اما T و R ناشناخته هستند!

- یادگیری مقادیر Q(s, a)
  - دریافت نمونه (s, a, s', r)
  - در نظر گرفتن تخمین قبلی: Q(s, a)
  - در نظر گرفتن تخمین مربوط به نمونه‌ی جدید:

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

  - به روز رسانی تخمین: [میانگین‌گیری]

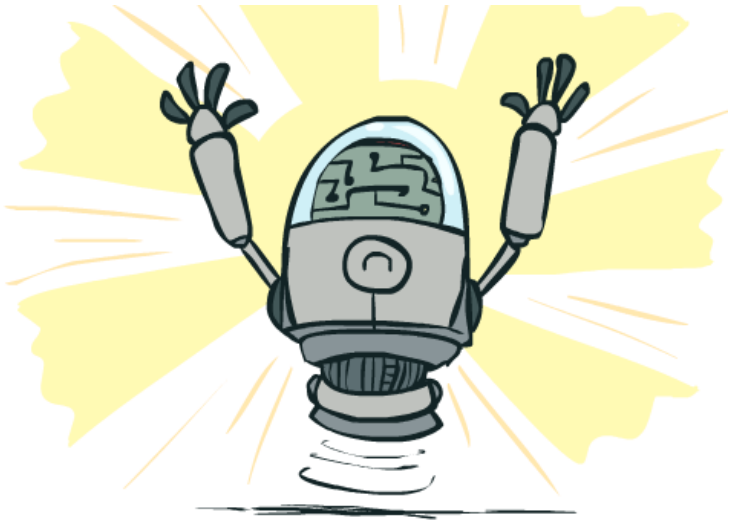
$$Q(s,a) = (1-\alpha)Q(s,a) + (\alpha)[sample]$$




```
% python gridworld.py -a q -k 1000
```

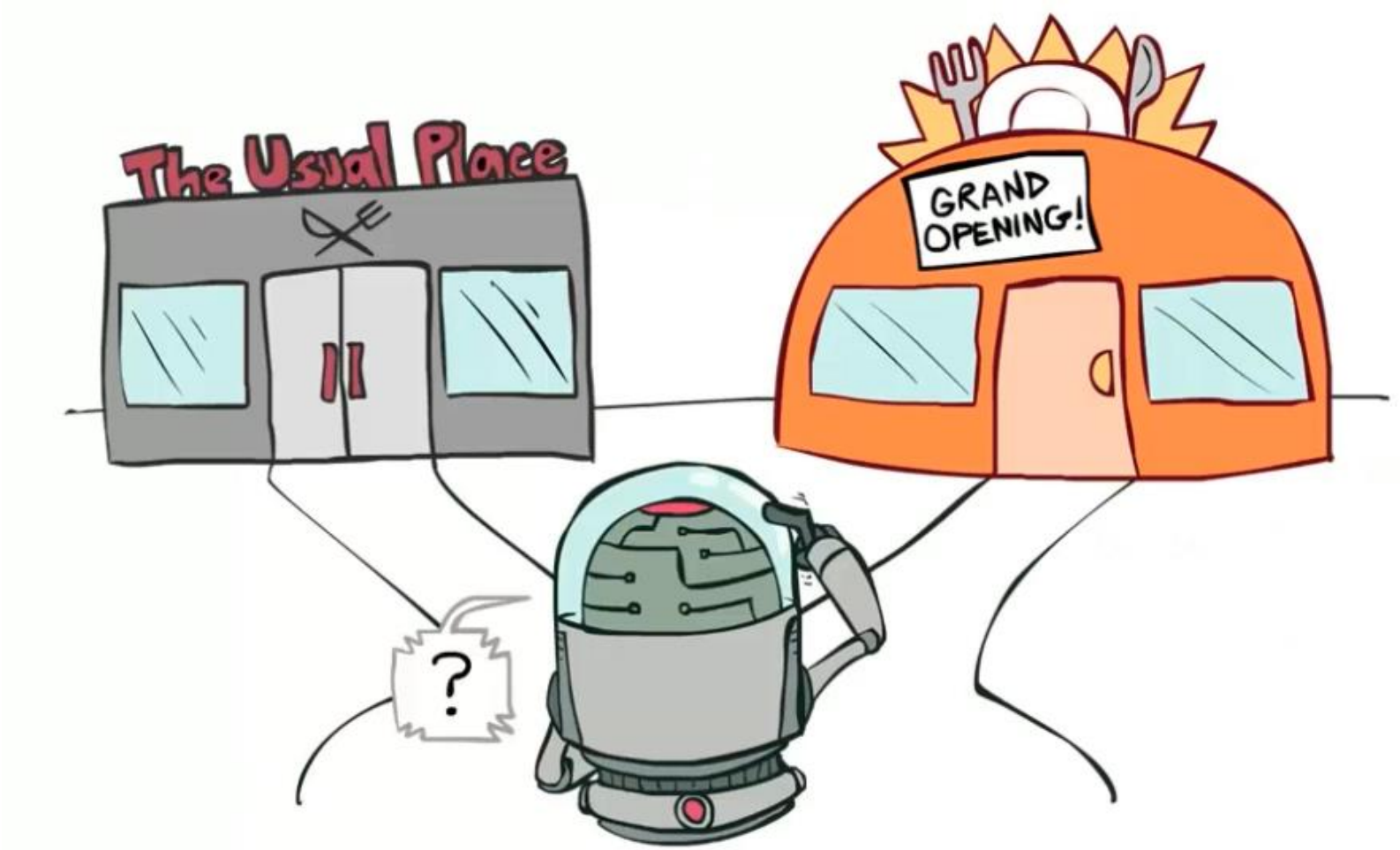
ویژگی‌های الگوریتم یادگیری Q

- همگرایی. الگوریتم یادگیری Q در سیاست بهینه همگرا می‌شود.
  - حتی اگر عامل بهینه عمل نکند!!!

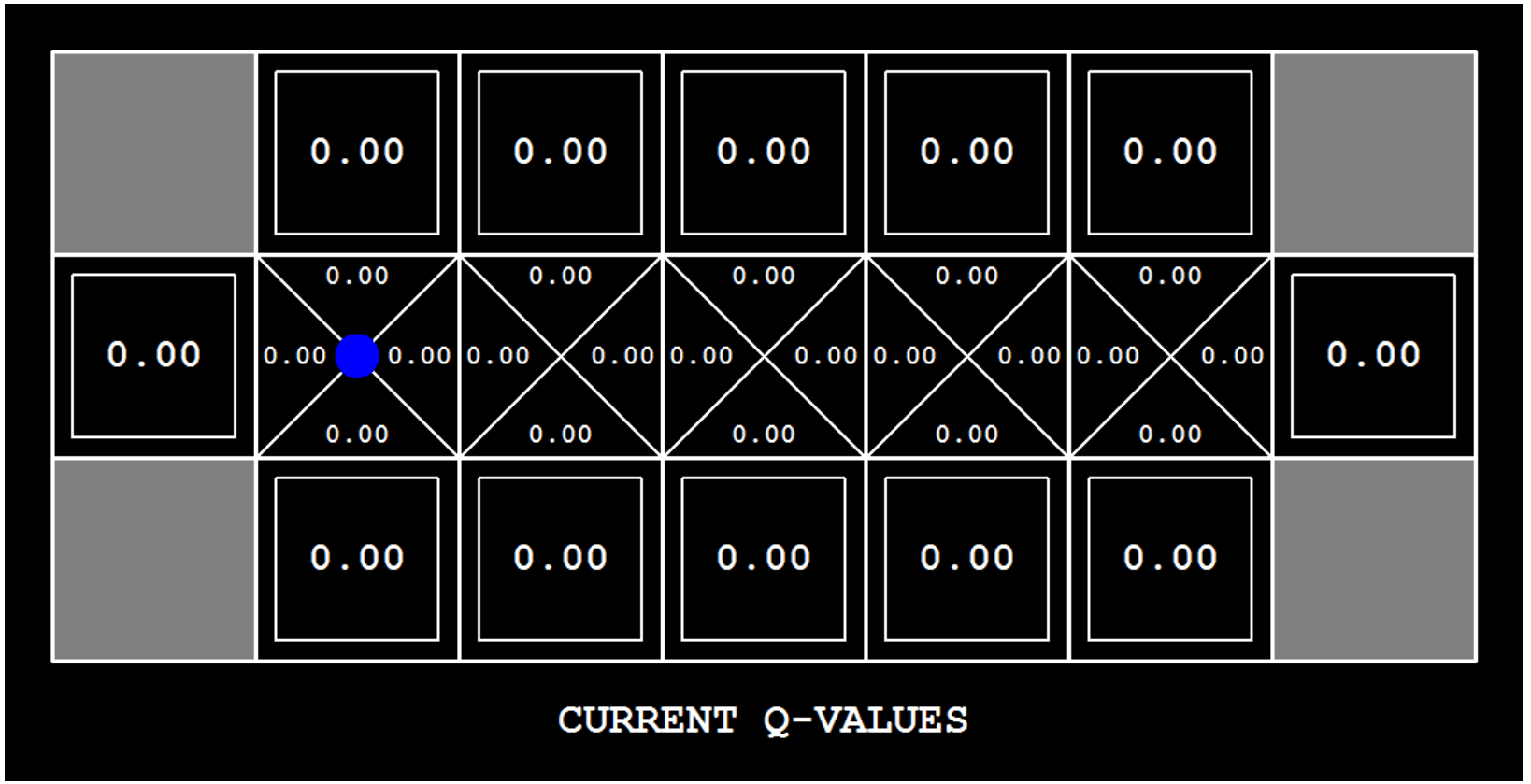


- هشدارها.
  - عامل باید به اندازه‌ی کافی محیط را کاوش کند.
  - نرخ یادگیری باید در نهایت به اندازه‌ی کافی کوچک شود.
  - ... اما مقدار آن نباید خیلی سریع کاهش داده شود.
  - به طور مبنایی، در حد، چگونگی انتخاب عمل به وسیله‌ی عامل اهمیت ندارد!

کاوش در برابر بهره‌برداری



اجرای نمایشی: کاوش در برابر بهره‌برداری




```
% python gridworld.py -a q -g BridgeGrid -k 100 -m
```

چگونه کاوش کنیم؟

- روش‌های مختلف برای کاوش محیط.
  - ساده‌ترین روش: برخی مواقع تصادفی عمل کن $[\varepsilon - greedy]$
    - قبل از انجام هر حرکت، شیر یا خط کن.
    - با احتمال کوچک ε، تصادفی عمل کن.
    - با احتمال بزرگ $1 - \varepsilon$، سیاست فعلی را دنبال کن.
- مشکلات تصادفی عمل کردن.
  - عامل سرانجام تمامی محیط را کاوش می‌کند، اما پس از مدتی با این که عمل درست را یاد گرفته است، باز هم تصادفی عمل می‌کند.
  - یک راه‌حل: کاهش مقدار ε در طول زمان
  - یک راه‌حل دیگر: توابع کاوش

توابع کاوش

- چه زمانی و چگونه باید کاوش کنیم؟
  - اعمال تصادفی: کاوش تمام نواحی به صورت برابر
  - یک ایده‌ی بهتر: کاوش بیشتر در ناحیه‌هایی که بد بودن آنها (هنوز) ثابت نشده است. اما به محض این که فهمیدی یک ناحیه بد است، دست از جستجو در آن ناحیه بردار.
- تابع کاوش.
  - ورودی: تعداد دفعات رویت یک حالت (n) و یک تخمین از سودمندی آن حالت (u)
  - خروجی: یک تخمین خوش‌بینانه از سودمندی حالت مانند: $f(u,n) = u + k/n$

$$Q(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

قاعده‌ی به روزرسانی معمولی

$$Q(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} f\big(Q(s',a'), N(s',a')\big)$$

قاعده‌ی به روزرسانی اصلاح شده

اجرای نمایشی: روبات خزنده

خلاصه

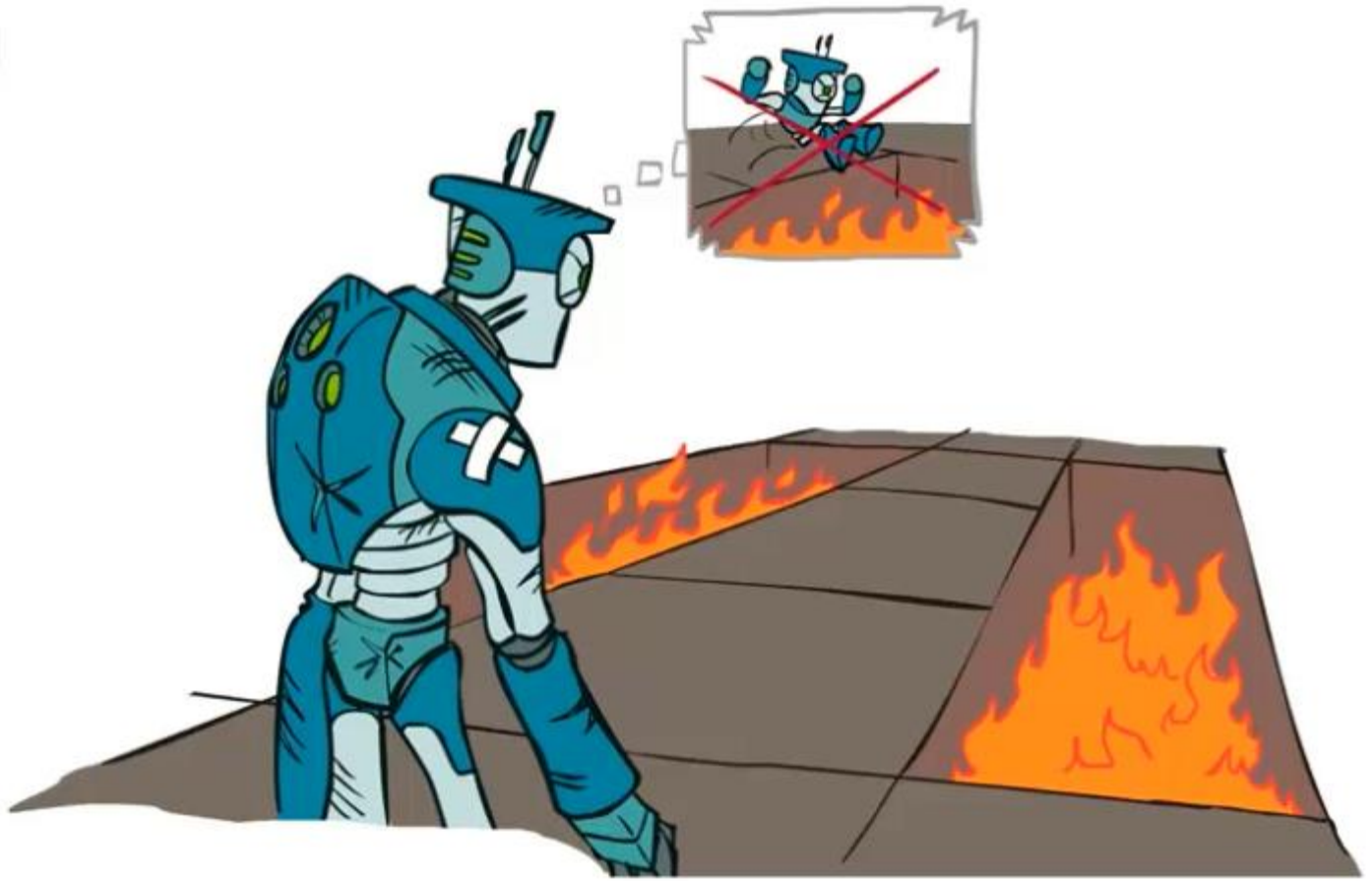
کمیت‌های قابل محاسبه:

- اگر MDP شناخته شده باشد:
  - محاسبه‌ی دقیق V^*، Q^* و π^*
  - ارزیابی سیاست ثابت π
- اگر MDP ناشناخته باشد:
  - می‌توانیم آن را تخمین زده و سپس حل کنیم.
  - می‌توانیم V را برای سیاست ثابت π تخمین بزنیم.
  - می‌توانیم $Q^*(s, a)$ را برای سیاست بهینه تخمین بزنیم.

روش‌های محاسبه:

- راه‌حل آفلاین:
  - الگوریتم تکرار مقدار
  - الگوریتم تکرار سیاست
- یادگیری تقویتی:
  - یادگیری تقویتی مبتنی بر مدل
  - مستقل از مدل: یادگیری مقدار
  - مستقل از مدل: یادگیری کیو.

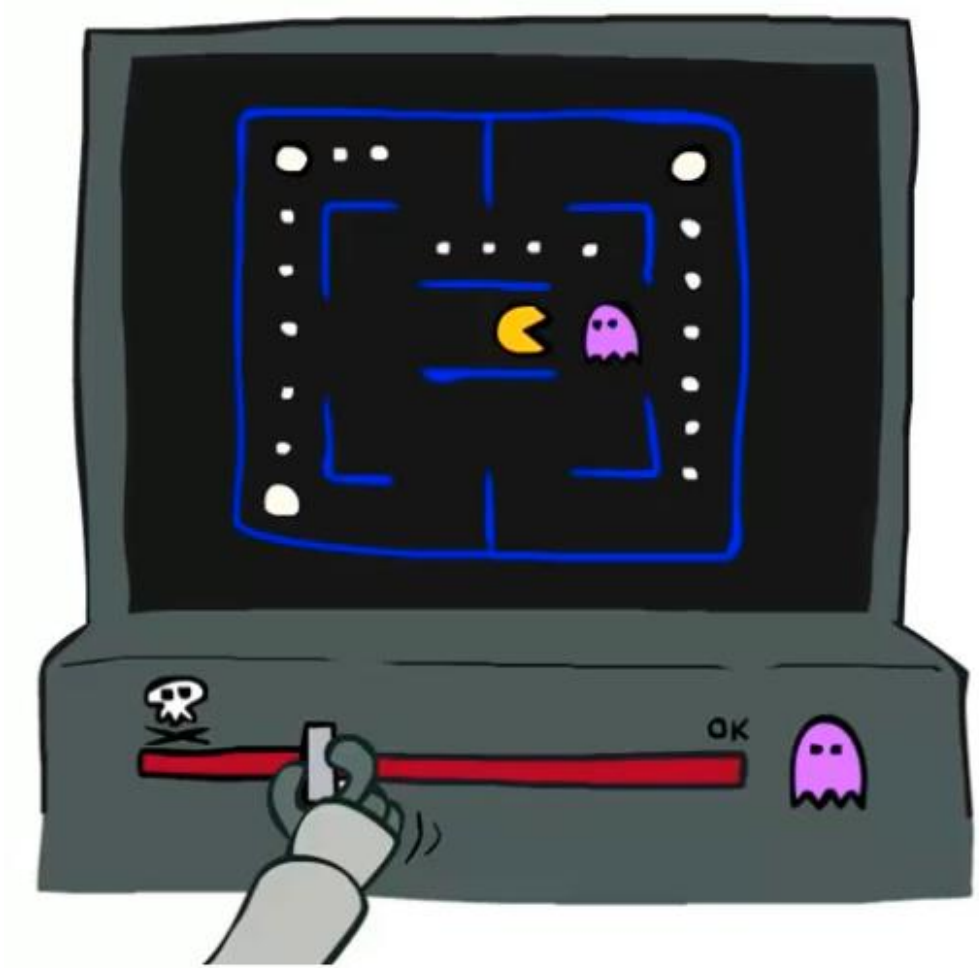
حسرت!



حسرت!

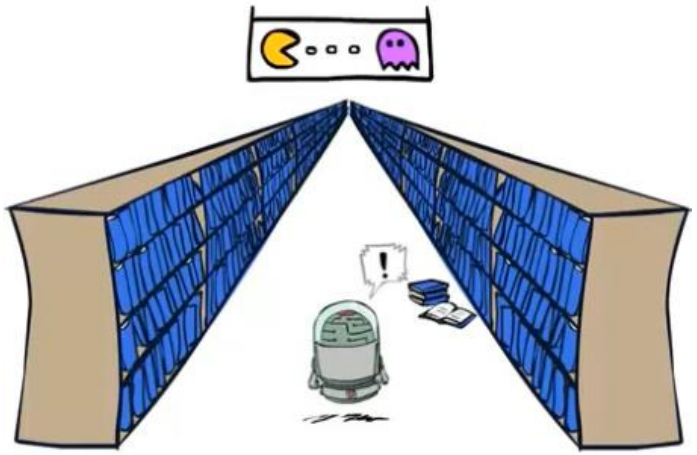
- حتی در صورت یاد گرفتن سیاست بهینه، ممکن است عامل در طول یادگیری اشتباه کند.
- معیار حسرت بیانگر مجموع هزینه‌ی اشتباهات در طول فرایند یادگیری است:
  - یعنی، اختلاف میان پاداش مورد انتظار به دست آمده در هر مرحله و پاداش بهینه‌ی مورد انتظار.
- کمینه‌سازی معیار حسرت فراتر از یادگیری بهینه بودن است:
  - یادگیری بهینه برای بهینه بودن!
  - یعنی، یادگیری سیاست بهینه با کمترین میزان اشتباهات!
- مثال. کاوش تصادفی و تابع کاوش هر دو منجر به یادگیری سیاست بهینه می‌شوند، اما کاوش تصادفی در مجموع دارای میزان اشتباهات (حسرت) بیشتری است.

یادگیری Q تقریبی



تعمیم میان حالت‌ها

- الگوریتم یادگیری Q یک جدول از تمامی مقادیر Q نگهداری می‌کند.
  - در موقعیت‌های واقعی، امکان یادگیری در مورد همه‌ی حالت‌ها به صورت جداگانه وجود ندارد!
    - به دلیل تعداد بسیار زیاد حالت‌ها، نمی‌توان همه‌ی حالت‌ها را در حین آموزش رویت نمود.
    - به دلیل تعداد بسیار زیاد حالت‌ها، نمی‌توان همه‌ی جدول را در حافظه ذخیره نمود.



- در عوض می‌توانیم تعمیم دهیم.
  - کسب تجربه در مورد یک زیرمجموعه‌ی کوچک از حالت‌ها
  - تعمیم این تجربه به حالت‌های جدید مشابه
  - قابلیت تعمیم یکی از مفاهیم اصلی در یادگیری ماشین است.

مثال: پکمن

فرض کنید با تجاربی که در محیط کسب می‌کنیم، دریابیم که این حالت، حالت خوبی نیست.

با این وجود در الگوریتم پایه‌ای یادگیری Q، هنوز هیچ چیزی در مورد این حالت نمی‌دانیم.

همین طور در مورد این حالت!

اجرای نمایشی: در حین یادگیری ...

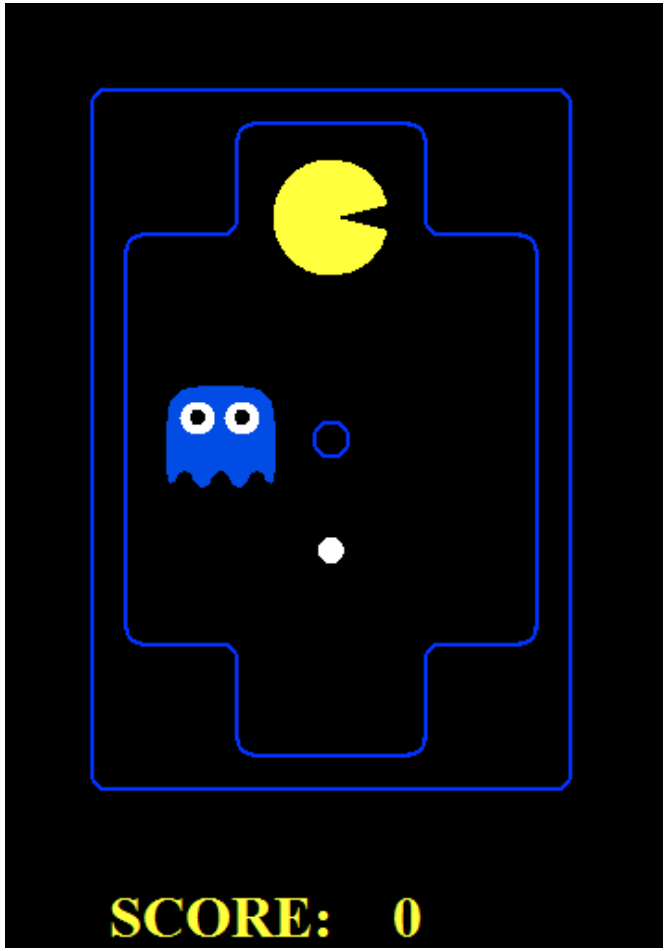

```
pacman.py -p PacmanQAgent -n 10 -l tinyTest
```

اجرای نمایشی: پس از یادگیری ...

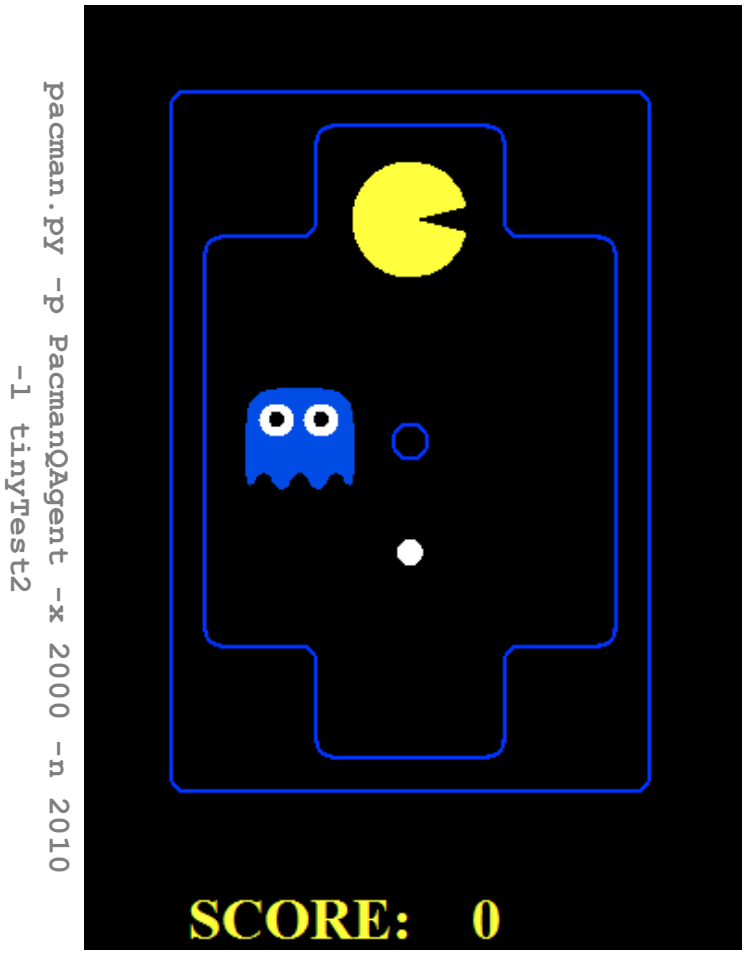

```
pacman.py -p PacmanQAgent -x 2000 -n 2010 -l
tinyTest
```

اجرای نمایشی: در حین یادگیری ...

pacman.py -p PacmanQAgent -n 30 -l tinyTest2



اجرای نمایشی: پس از یادگیری ...



نمایش مبتنی بر ویژگی

- راه‌حل. توصیف حالت‌ها به صورت یک بردار از ویژگی‌ها!
  - هر ویژگی یک تابع از حالت‌ها به اعداد حقیقی است و بیانگر خصوصیات مهم آن حالت است.
- ویژگی‌های مثالی.
  - فاصله تا نزدیک‌ترین روح
  - فاصله تا نزدیک‌ترین غذا
  - تعداد ارواح
  - آیا پکمن در یک تونل است؟ (صفر-یک)
  - و ...
- به همین ترتیب، حالت‌های q را نیز می‌توان به صورت برداری از ویژگی‌ها نمایش داد.
  - مثلاً آیا این عمل پکمن را به غذا نزدیک‌تر می‌کند؟

توابع مقدار خطی

- به وسیله نمایش مبتنی بر ویژگی، می‌توان برای هر حالت با استفاده از تعدادی وزن یک تابع q نوشت:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)$$

- مزیت. تجربه‌ی عامل در چند عدد (مقادیر پارامترهای وزن) خلاصه می‌شود.
- ایراد. ممکن است حالت‌ها ویژگی‌های مشترک داشته باشند، اما ارزش آنها بسیار متفاوت باشد.

الگوریتم یادگیری تقریبی Q

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)$$

- یادگیری با استفاده از توابع خطی q.
  - مشاهده‌ی یک تجربه‌ی جدید: [تغییر حالت] (s, a, s', r)
  - محاسبه‌ی تفاوت: $difference = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$
  - اصلاح تابع q با اصلاح وزن‌ها:

مقادیر دقیق Q: $Q(s,a) \leftarrow Q(s,a) + \alpha[difference]$

مقادیر تقریبی Q: $w_i \leftarrow w_i + \alpha \cdot [difference] \cdot f_i(s,a)$

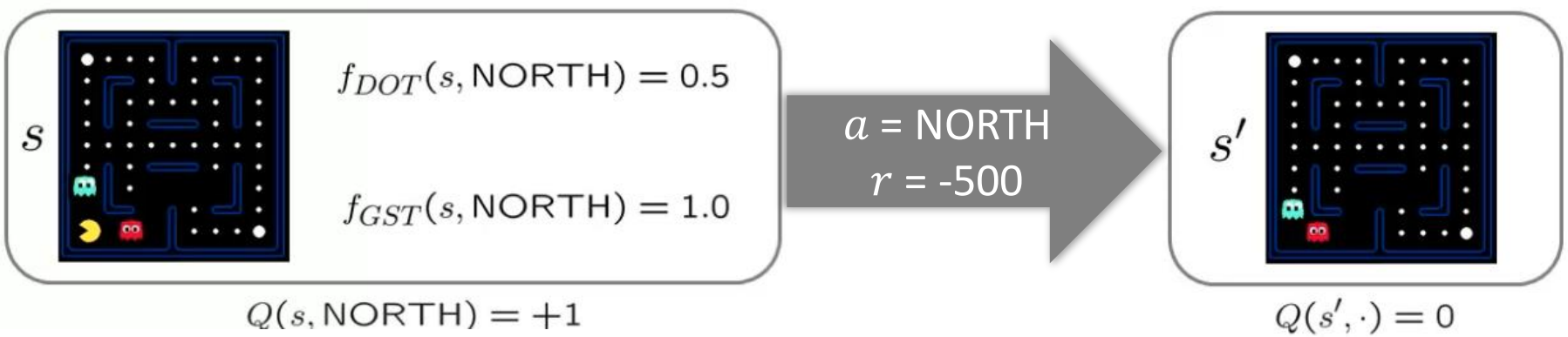
- به صورت شهودی:
  - تنظیم وزن مربوط به ویژگی‌های فعال.
  - یعنی، اگر به طور ناگهانی اتفاق بدی بیفتد، وزن مربوط به آن ویژگی کاهش داده می‌شود و این عمل باعث می‌شود تمام حالت‌های مشابه که این ویژگی را دارند، از نظر عامل کم ارزش‌تر شوند.

مثال: پکمن

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

عکس فاصله تا نزدیک‌ترین غذا

عکس فاصله تا نزدیک‌ترین روح

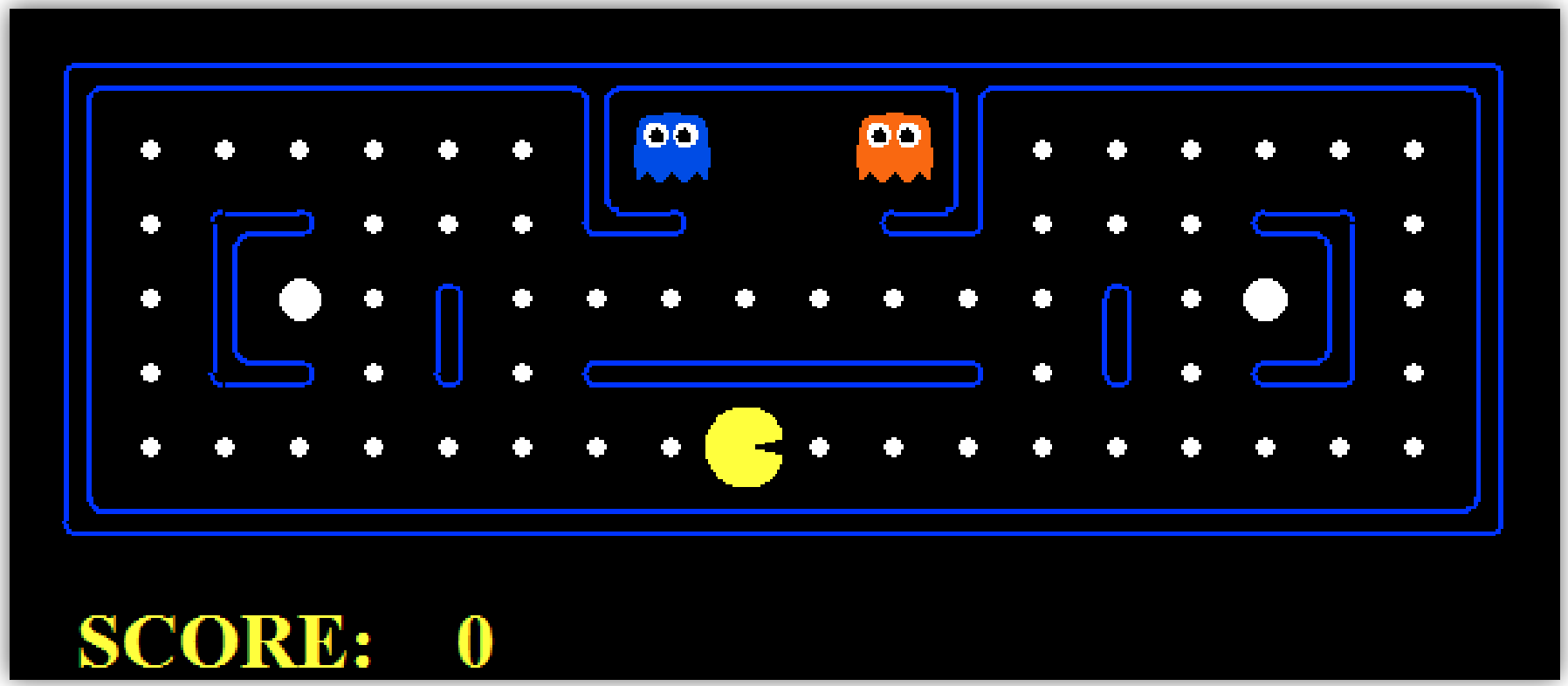


$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

$$difference = -501 \Rightarrow \begin{aligned} w_{DOT} &= 4.0 + \alpha[-501](0.5) \\ w_{GST} &= -1.0 + \alpha[-501](1.0) \end{aligned}$$

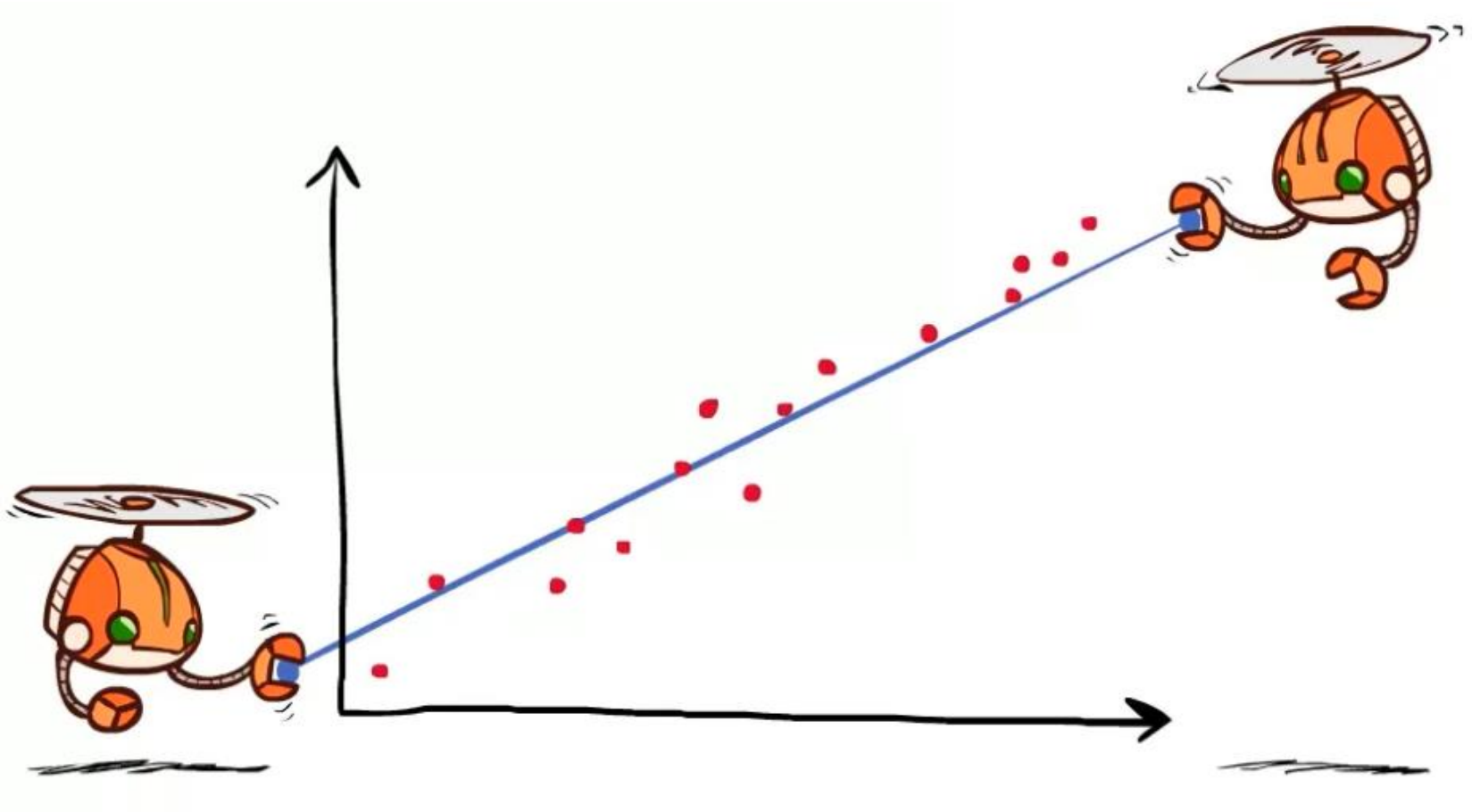
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

اجرای نمایشی

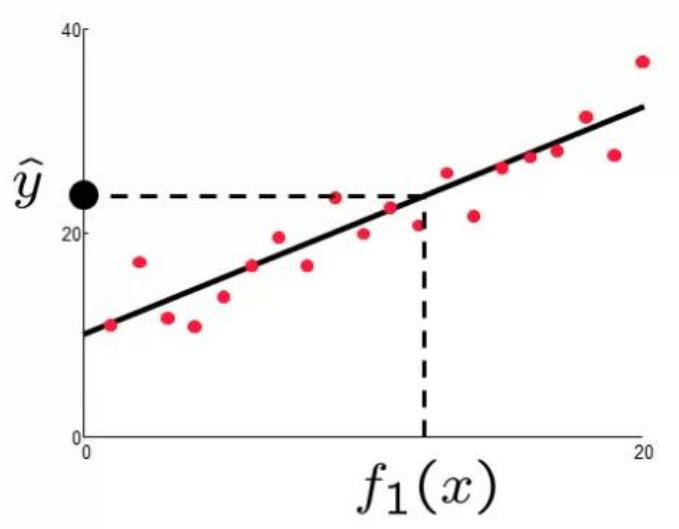



```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -n 10 -l smallClassic
```

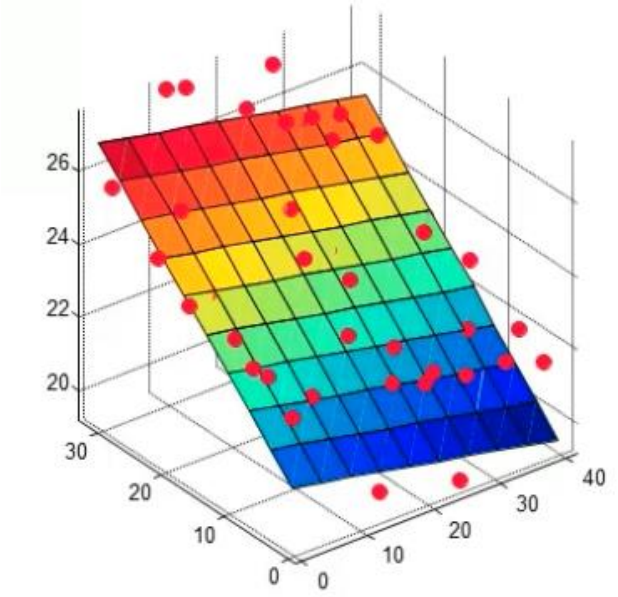
کمینه‌سازی خطا



تقریب خطی: رگرسیون



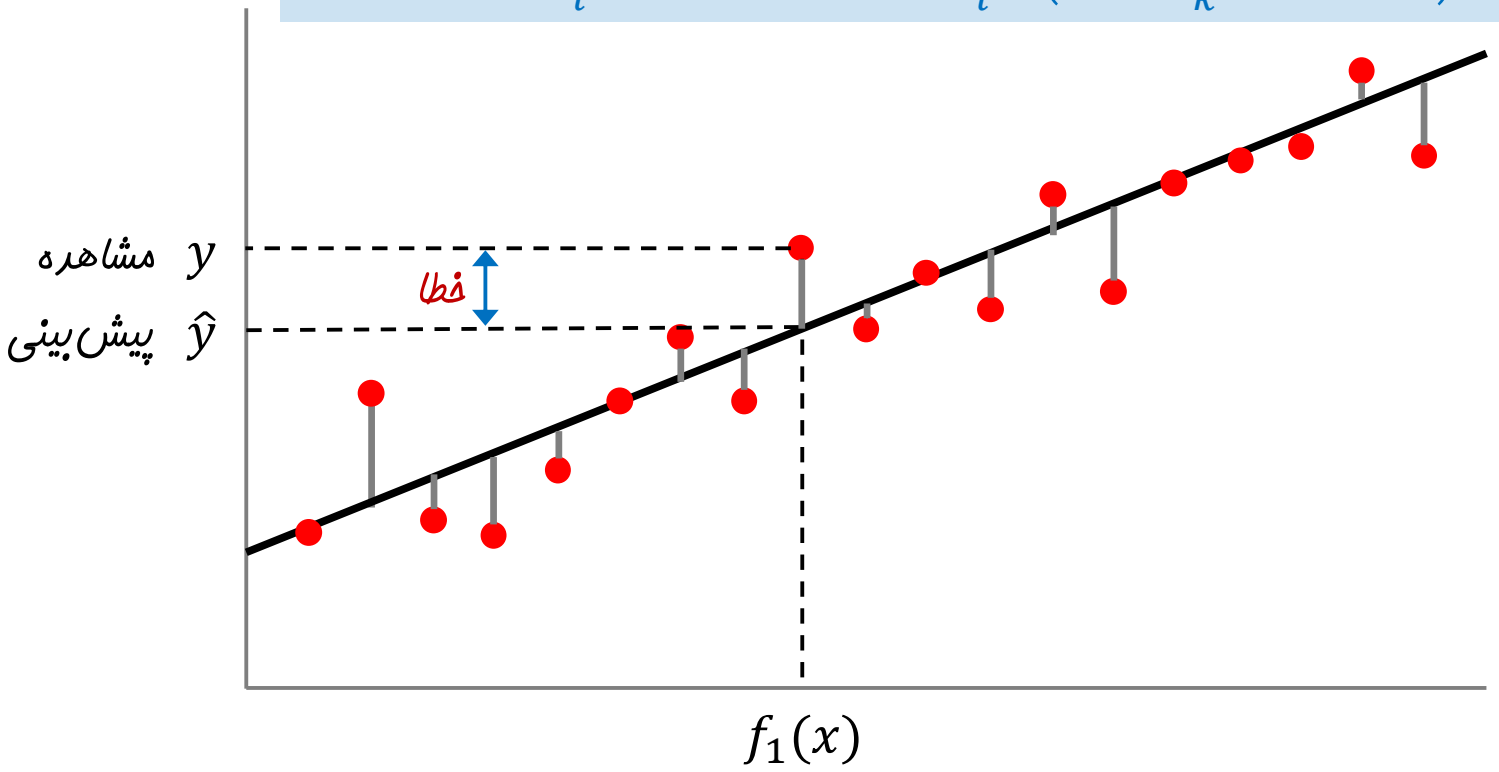
$$\hat{y} = w_0 + w_1 f_1(x)$$



$$\hat{y} = w_0 + w_1 f_1(x) + w_2 f_2(x)$$

بهینه‌سازی: کمینه‌سازی خطا

$$error = \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_i \left(y_i - \sum_k w_k f_k(x) \right)^2$$



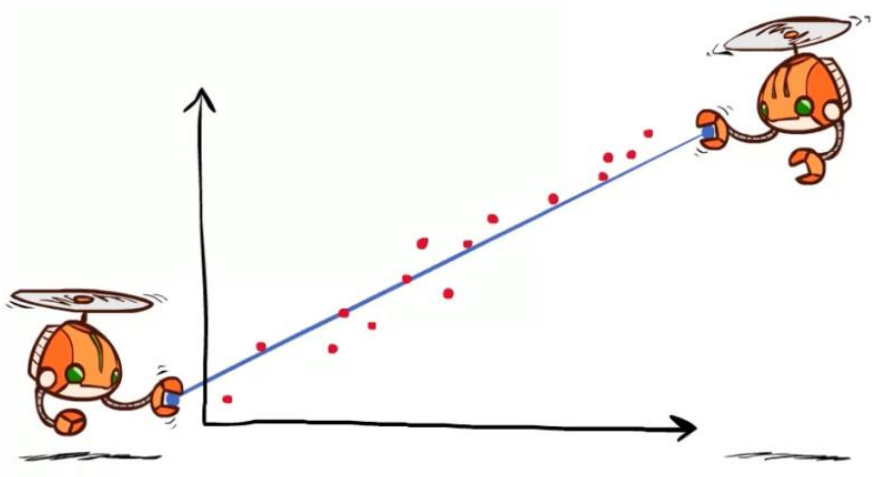
کمینه‌سازی خطا

- کمینه‌سازی خطا. فرض کنید تنها یک نمونه x، با بردار ویژگی $f(x)$، مقدار هدف y و وزنهای w داشته باشیم.

$$error(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial error(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

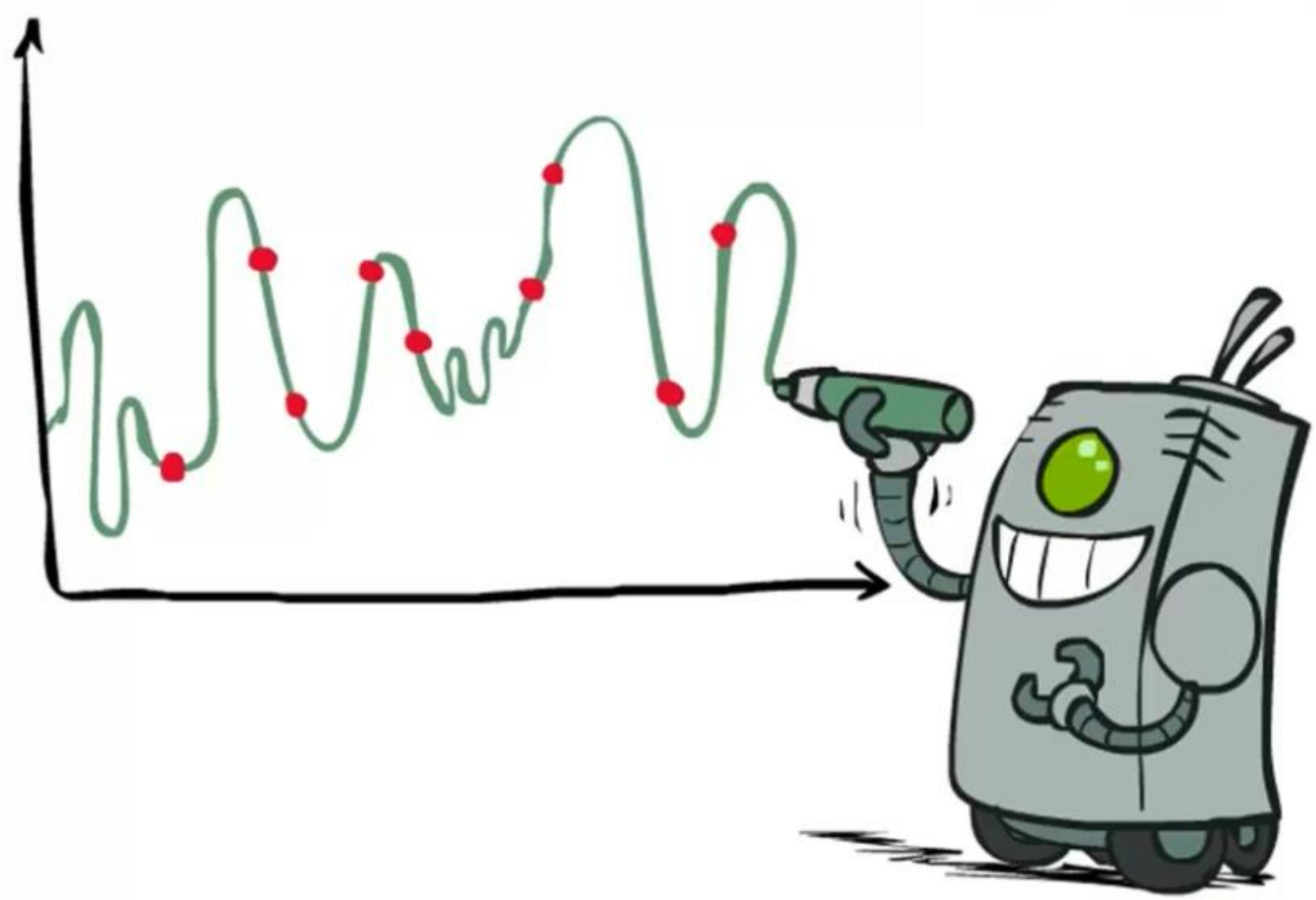


- قاعده به روزرسانی وزن‌ها در الگوریتم تقریبی یادگیری q.

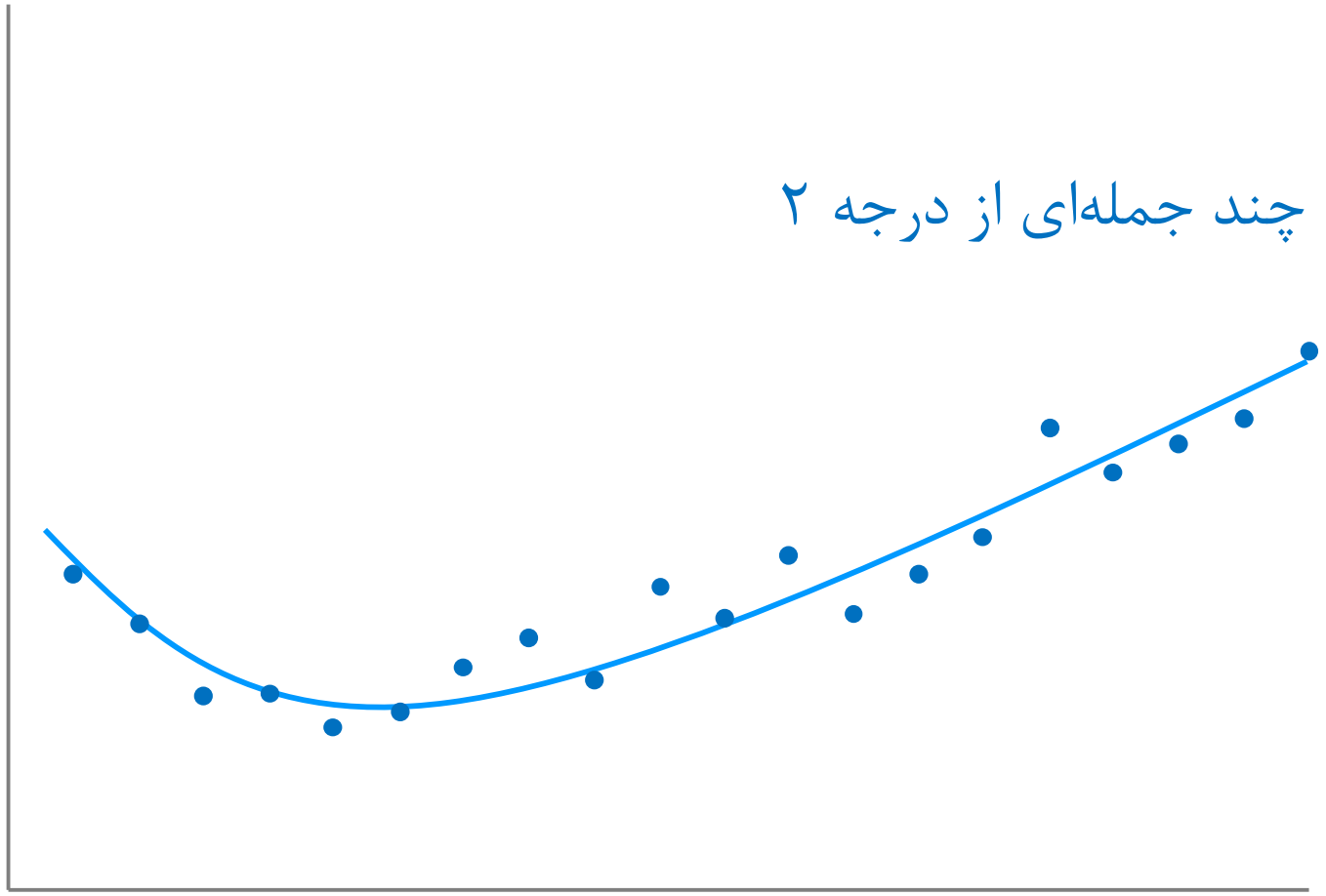
$$w_m \leftarrow w_m + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right) f_m(x)$$

مشاهده — پیش‌بینی

بیش‌برازش: چرا محدود کردن ظرفیت مفید است؟

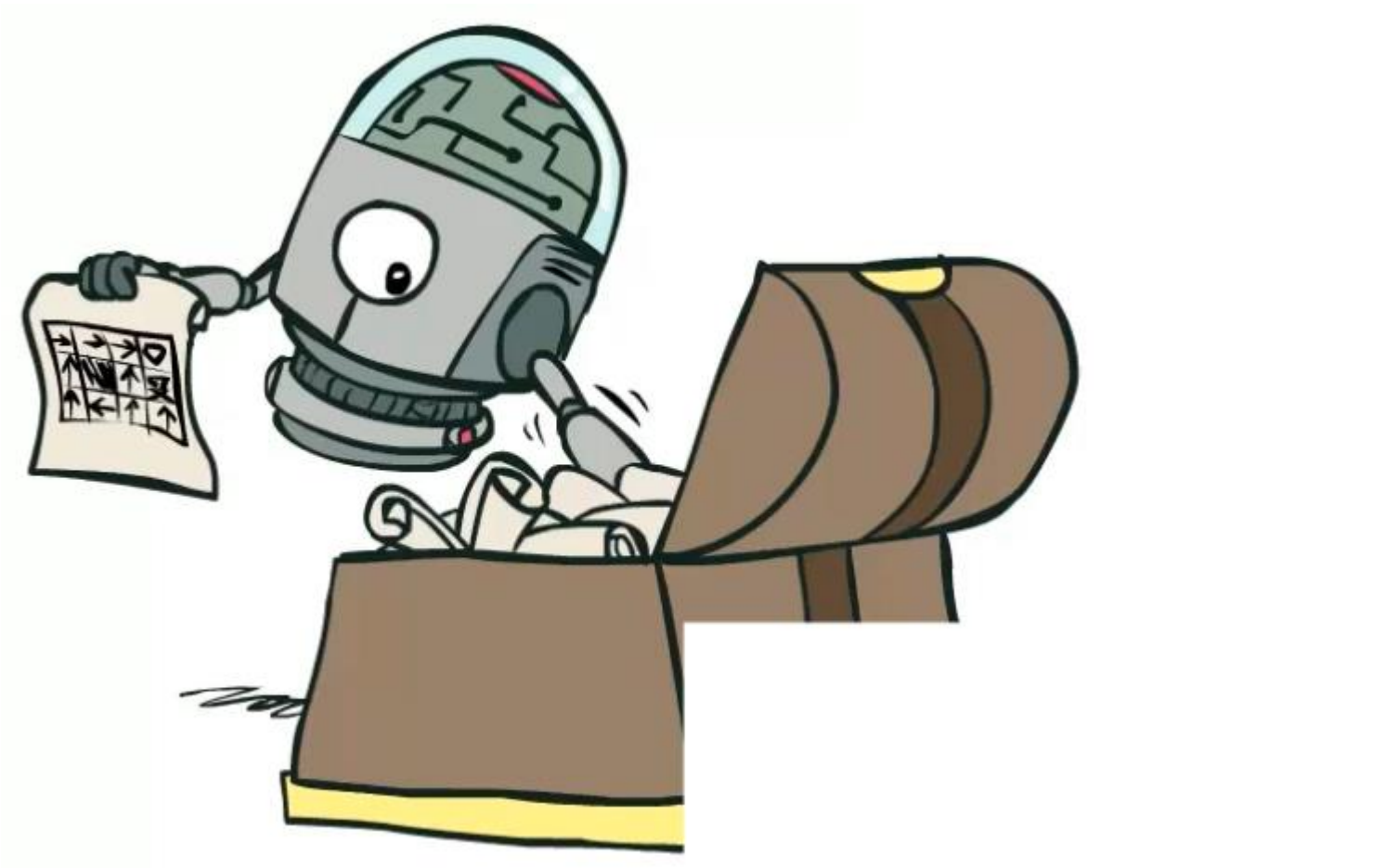


بیش‌برازش: چرا محدود کردن ظرفیت مفید است؟



بیش‌برازش: چرا محدود کردن ظرفیت مفید است؟

جستجوی سیاست



جستجوی سیاست

- مشکل. در اغلب موارد سیاست‌های مبتنی بر ویژگی که در عمل به خوبی کار می‌کنند (برنده شدن در بازی)، آنهایی نیستند که مقادیر Q یا V را به خوبی تخمین می‌زنند.
  - اولویت اصلی در الگوریتم یادگیری Q: تخمین دقیق مقادیر Q (مدل‌سازی)
  - اولویت اصلی در انتخاب عمل: به دست آوردن ترتیب درست برای مقادیر Q (پیش‌بینی)

- راه‌حل. یادگیری سیاست‌هایی که پاداش را بیشنه می‌سازند، نه یادگیری مقادیری که سیاست‌ها را پیش‌بینی می‌کنند!

- جستجوی سیاست. با یک راه‌حل خوب (مثلاً راه‌حل به دست آمده از یادگیری Q) شروع کن و سپس با انجام تپه‌نوردی بر روی وزن ویژگی‌ها، آن را بهبود ببخش.

جستجوی سیاست

- ساده‌ترین روش جستجوی سیاست.
  - با یک تابع Q اولیه شروع کن.
  - مقادیر وزنها را کم و زیاد کن و ببین آیا سیاست جدید نسبت به قبل بهتر شده یا خیر.

- مشکلات.
  - چگونه می‌توان تشخیص داد یک سیاست نسبت به قبل بهتر شده است؟
    - نیاز به اجرای اپیزودهای آموزشی بسیار زیاد!
    - اگر تعداد ویژگی‌ها زیاد باشد، این روش عملی نیست.

- روش‌های بهتر. بهره‌برداری از ساختار پیش‌بینی، نمونه‌برداری هوشمندانه، تغییر چند پارامتر به طور همزمان و ...

جستجوی سیاست

نتیجه‌گیری

- پایان بخش اول: جستجو و برنامه‌ریزی!
- استفاده از هوش مصنوعی برای:
  - مسائل جستجو
  - مسائل ارضای محدودیت
  - بازی‌ها
  - مسائل تصمیم‌گیری مارکوف
  - یادگیری تقویتی
- بخش دوم: عدم قطعیت و یادگیری!