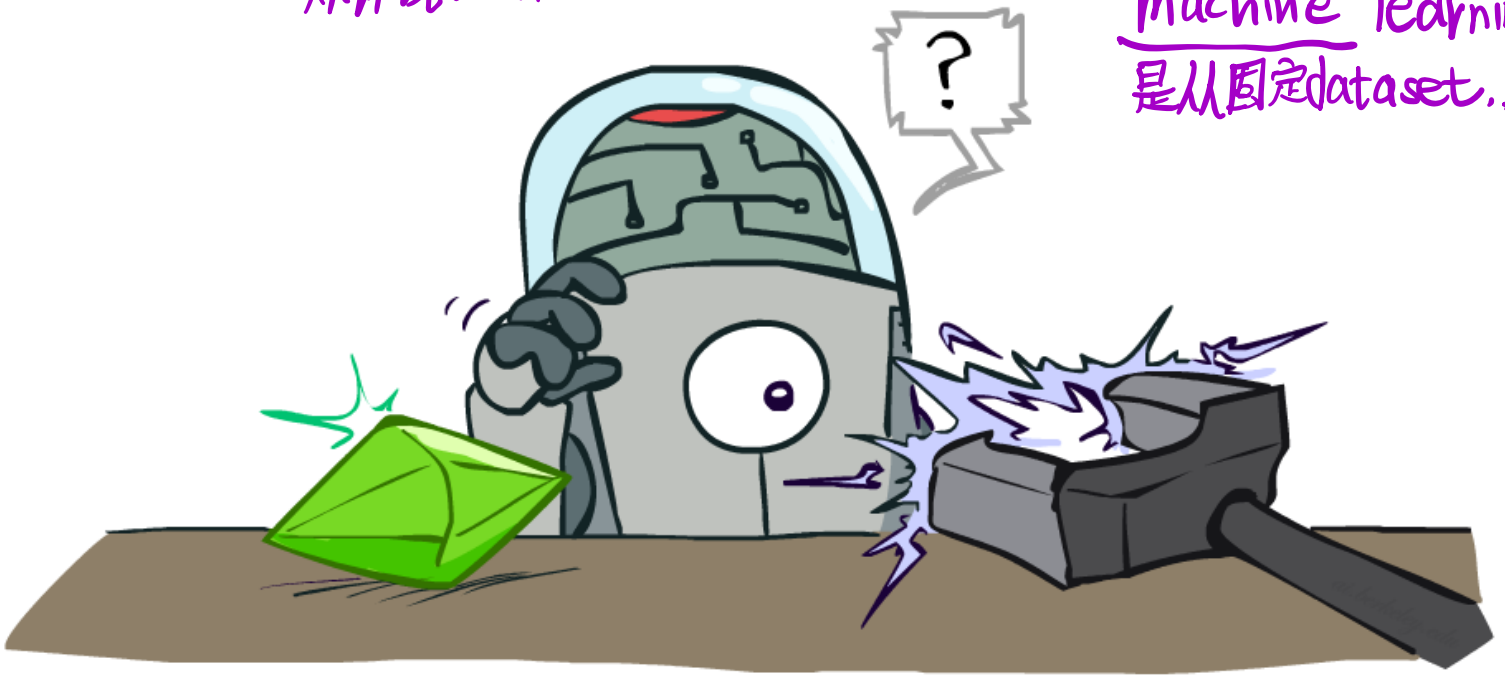
本身比较innocent

# Reinforcement Learning

从环境里面learn

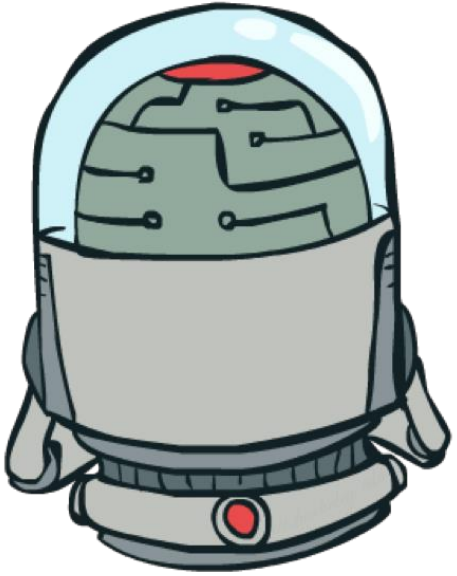
machine learning
是从固定dataset，固定分布去拟合




AIMA Chapter 21

[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley]

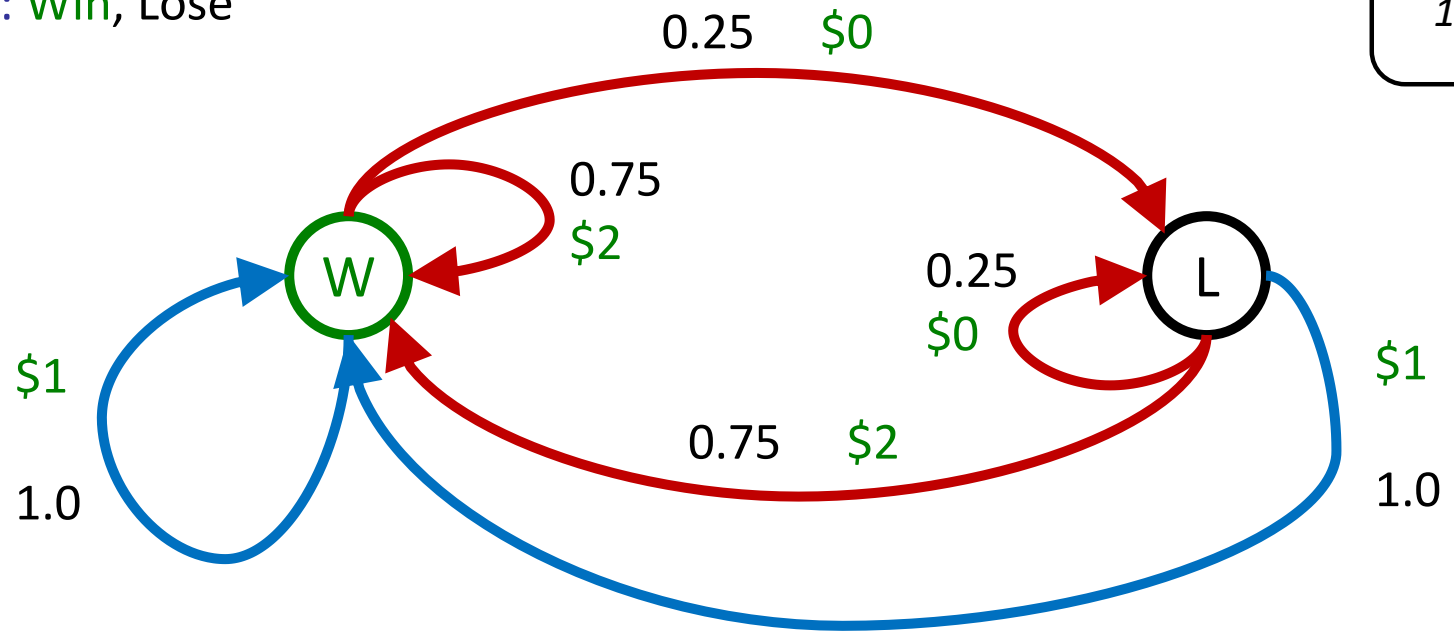
# Double Bandits

# Double-Bandit MDP

- Actions: *Blue, Red*
- States: Win, Lose

*No discount*
*100 time steps*



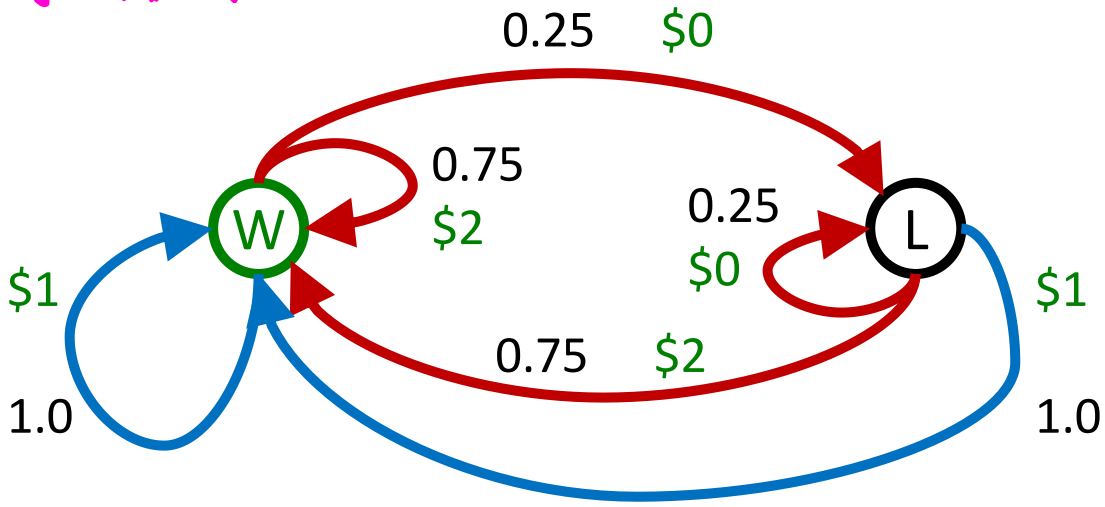
# Offline Planning

- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

↗可以纸上谈兵
通过计算可以确认所有数据
必须知晓MDP
可以不玩游戏

*No discount*
*100 time steps*

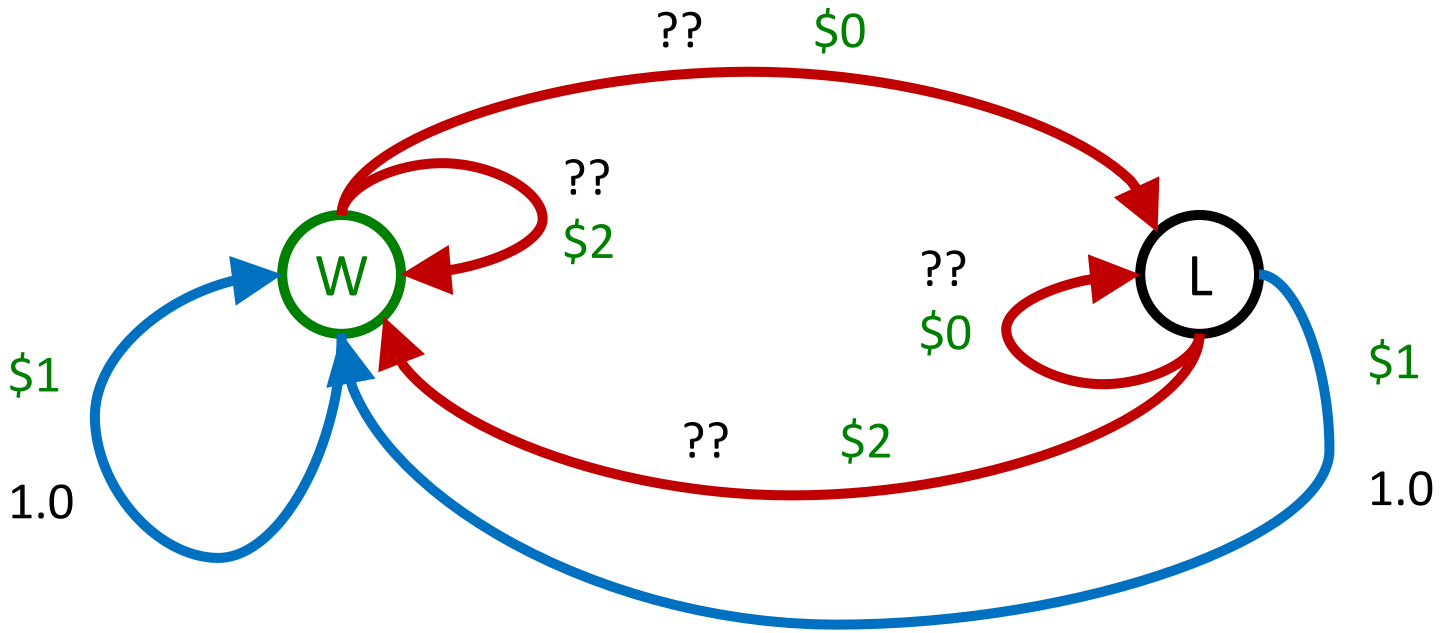
| | Value |
|---|---|
| Play Red | 150 |
| Play Blue | 100 |



# Online Planning

- Rules changed! Red's win chance is different.

online: 要实际进行游戏才能知道怎么更好地适应



# Let’s Play!



$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

# What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning 存在MDP,但不能通过计算得到
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out 真实去玩才能得到


- Important ideas in reinforcement learning that came up 太多:成本过高; 太少,已知信息太少.
  - Exploration: you have to try unknown actions to get information 用真实的未知行动去获取信息.
  - Exploitation: eventually, you have to use what you know 利用已知的信息
  - Regret: even if you learn intelligently, you make mistakes 即使学习了也会犯错
  - Sampling: because of chance, you have to try things repeatedly 由于偶然性,会重复尝试.
  - Difficulty: learning can be much harder than solving a known MDP 是一个有挑战性工作.

# Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states s ∈ S
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy π(s)




目标仍然是去获得最优policy

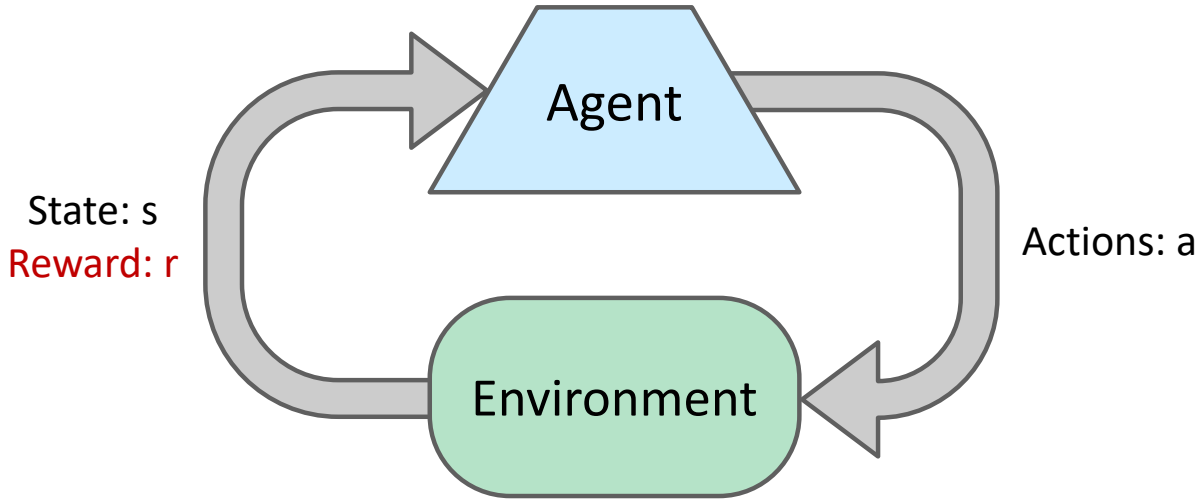
- New twist: don't know T or R

大部分条件下可能没有即时reward.

  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

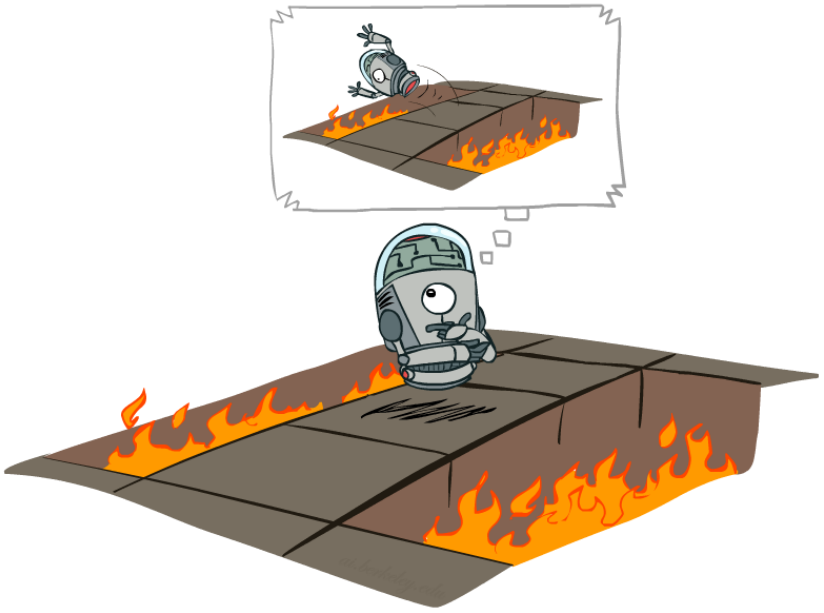
对于State好坏和action未知

# Reinforcement Learning




- Basic idea: 做出操作，观察(状态转移，奖励)
  - Take actions and observe outcomes (new states, rewards)
  - Learning is based on observed samples of outcomes 学习是基于观察样本
  - Must (learn to) act so as to maximize expected rewards
    尽量学会 act 如何最大期望奖励.

# Offline (MDPs) vs. Online (RL)

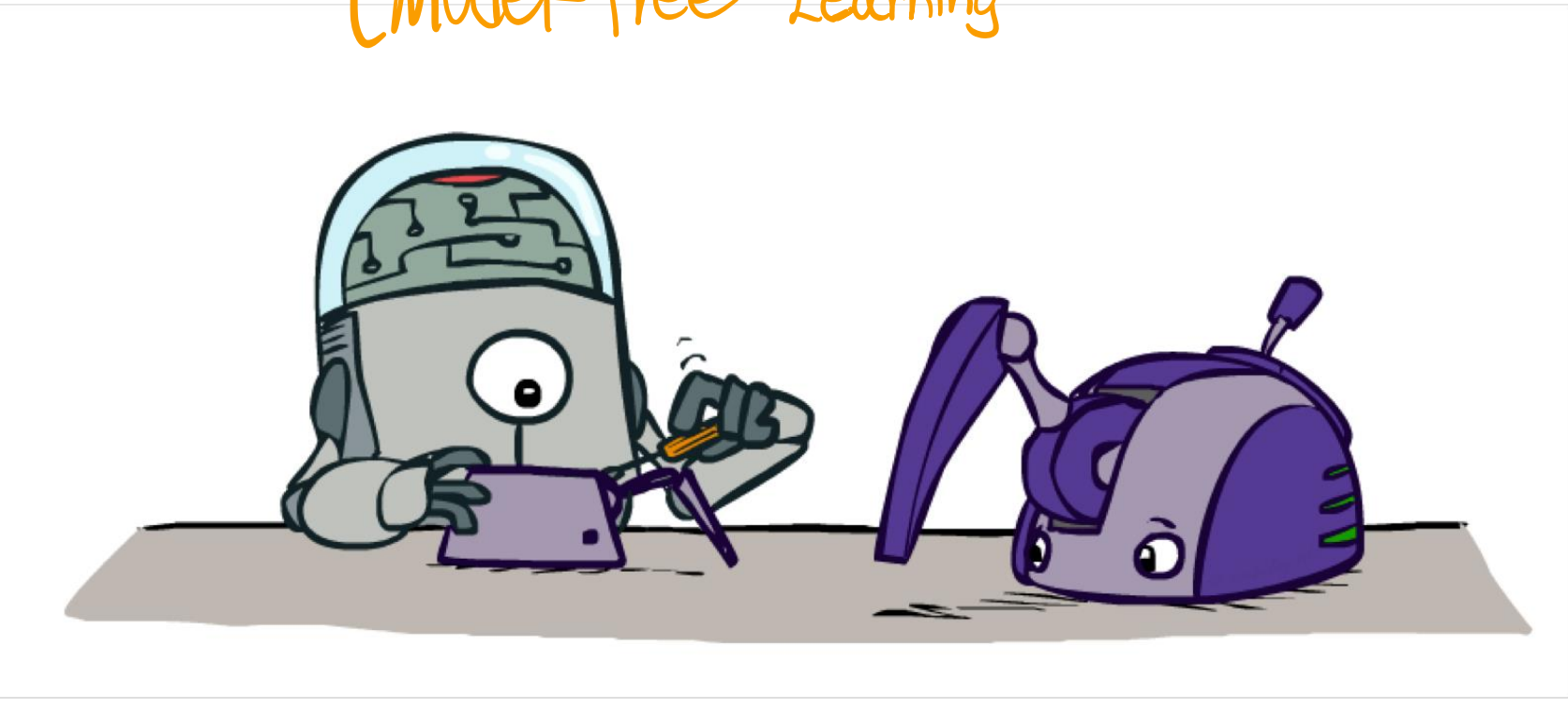


Offline Solution

Online Learning

# Model-Based Learning

Model-free Learning



# Model-Based Learning

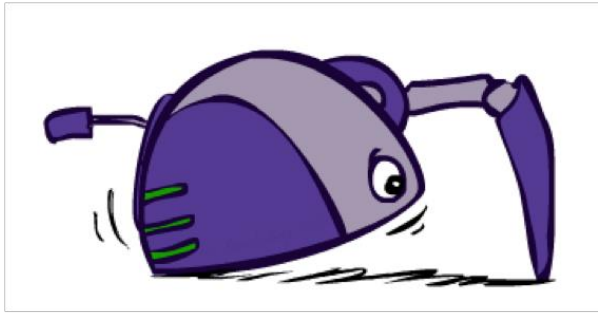
在 reinforce learning 里面，model 特指 MDP

其他的叫 policy.

- Model-Based Idea: 学习一个近似的模型
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model was correct

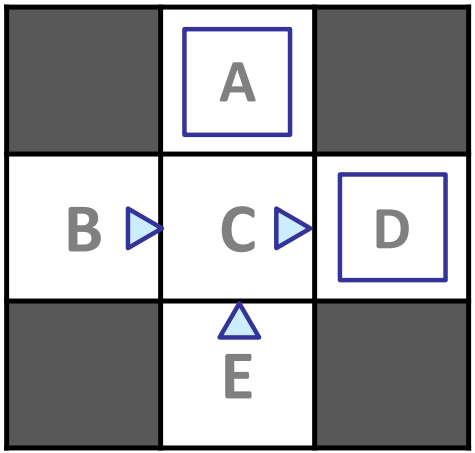
    根据计算的MDP来求值.
- Step 1: Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Normalize to give an estimate of $\hat{T}(s, a, s')$
  - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')

  建模就是将这两个未知参变已知
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before



# Example: Model-Based Learning

Episode: 从开始到结束是一个episode.




*Assume:* $\gamma = 1$

## Observed Episodes (Training)

**Episode 1**

B, east, C, -1
C, east, D, -1
D, exit, x, +10

**Episode 2**

B, east, C, -1
C, east, D, -1
D, exit, x, +10

**Episode 3**

E, north, C, -1
C, east, D, -1
D, exit, x, +10

**Episode 4**

E, north, C, -1
C, east, A, -1
A, exit, x, -10

## Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

# Model-Based vs. Model-Free

Goal: Compute expected age of ShanghaiTech students

**Known P(A)**

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [$a_1$, $a_2$, … $a_N$]

**Unknown P(A): "Model Based"**

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

算出每种概

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Why does this work? Because eventually you learn the right model.

**Unknown P(A): "Model Free"**

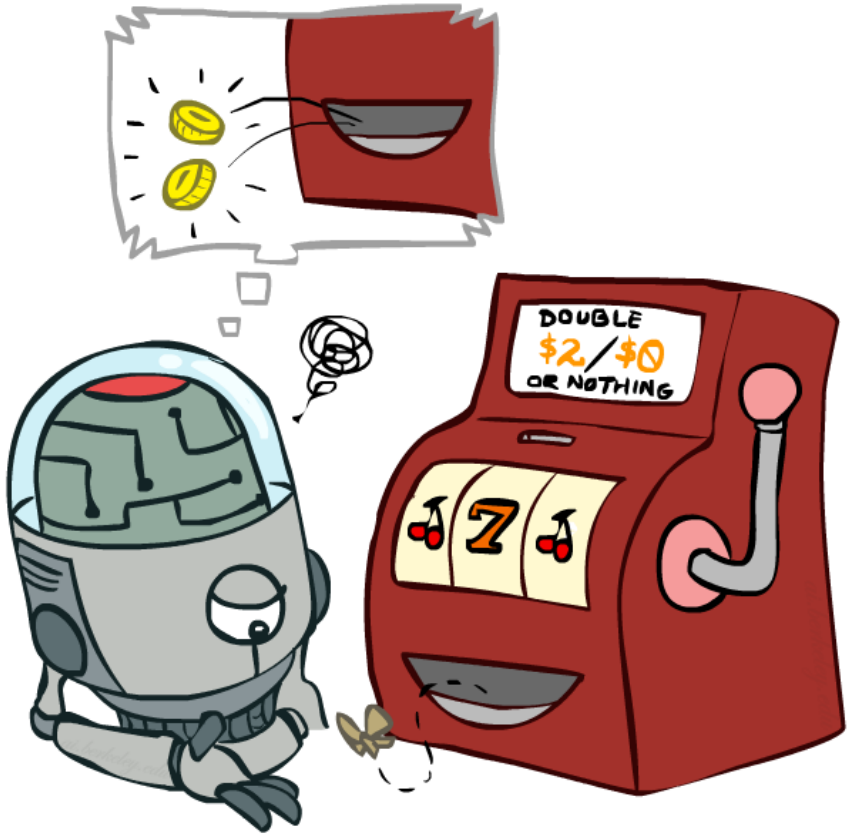
不用建模

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

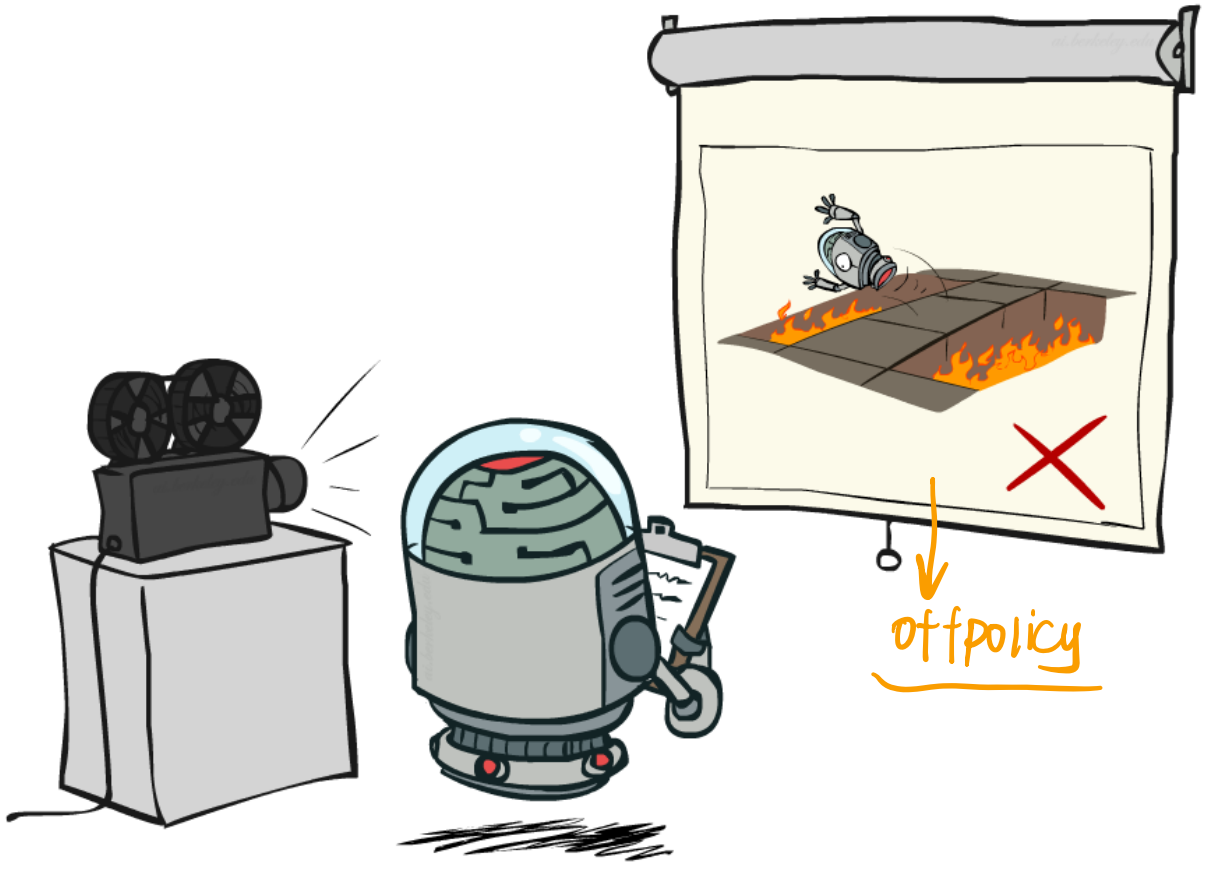
Why does this work? Because samples appear with the right frequencies.

model有问题的话，结论会有问题.

# Model-Free Learning

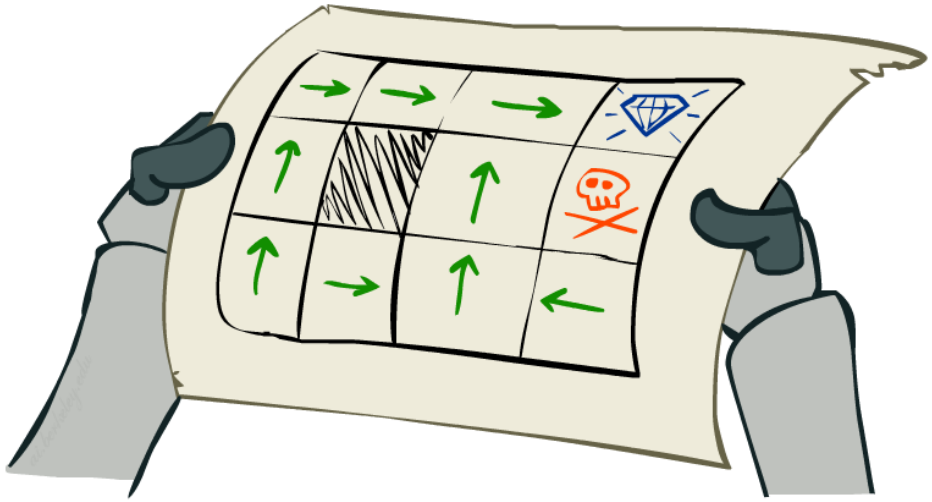


# Passive Reinforcement Learning



# Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: learn the state values



- In this case:
  - No choice about what actions to take 不用选择action
  - Just execute the policy and learn from experience 根据action来学习
  - This is NOT offline planning! You actually take actions in the world. 在真实环境在做action

# Direct Evaluation

- Goal: Compute values for each state under $\pi$

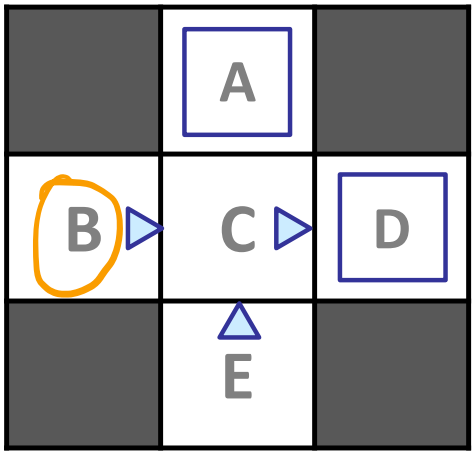
  计算每一个State在π策略下的Value

- Idea: Average together observed sample values
  - Act according to $\pi$ (固定策略) 计算平均的样本value
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples 计录reward

  计算平均的样本value

# Example: Direct Evaluation

## Input Policy $\pi$




*Assume:* $\gamma = 1$

## Observed Episodes (Training)

从B出发，reward:8

**Episode 1**

B, east, C, -1
C, east, D, -1
D, exit, x, +10

**Episode 2**

B, east, C, -1
C, east, D, -1
D, exit, x, +10

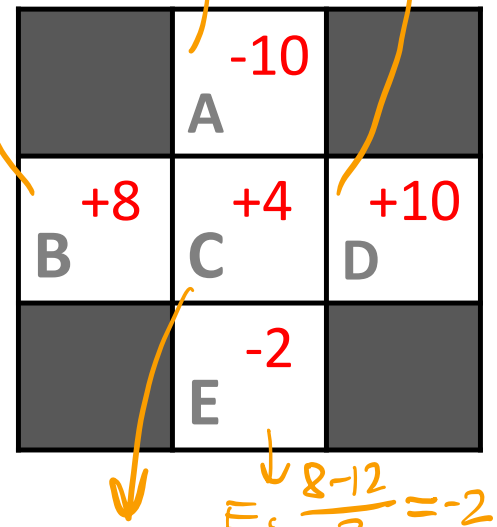
**Episode 3**

E, north, C, -1
C, east, D, -1
D, exit, x, +10

**Episode 4**

E, north, C, -1
C, east, A, -1
A, exit, x, -10

## Output Values




E: $\frac{8-12}{2} = -2$

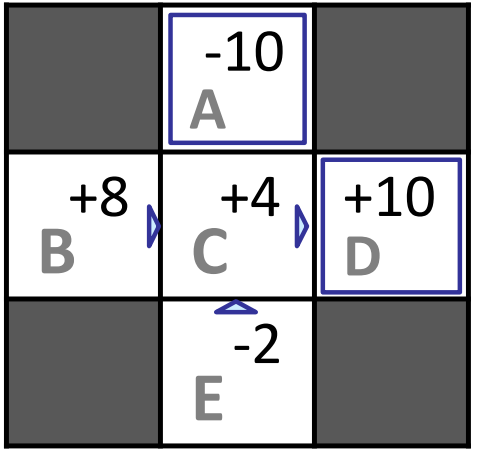
C出发: $\frac{9+9+9-11}{4} = +4$

# Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions

- What bad about it?
  - It wastes information about state connections 浪费信息
  - Each state must be learned separately 每一个stage都要单独训练.
  - So, it takes a long time to learn

Output Values



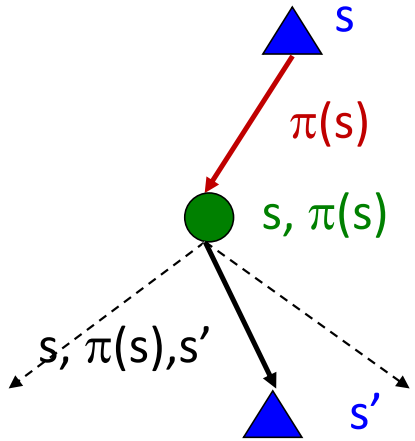

*B goes to C, so we may use Bellman equation*

# Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:

$$V_0^{\pi}(s) = 0$$

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




  - This approach fully exploits the connections between the states
  - Unfortunately, we need T and R to do it!

- Key question: how can we do this update to V without knowing T and R?
  - In other words, how do we take a weighted average without knowing the weights?

我们如何避免知晓T,R而update V呢.

# Sample-Based Policy Evaluation?

- We want to compute these averages:

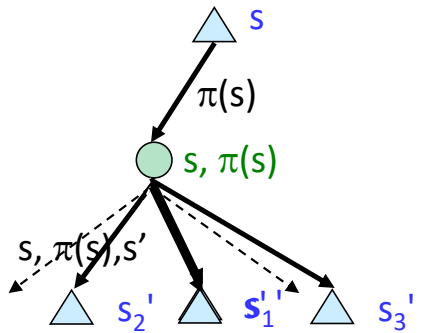
$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

问题：在实际操作中，我们从S开始进行了sample，那么就很难返回到S(开始)的时候，所以我们可能永不到N作平均.

- Idea: Take samples of outcomes s' (by doing the action!) and average

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2)$$
$$\cdots$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n)$$

?




$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i sample_i$$

可以避免转移

$$= \frac{\sum r(sample)}{\#\ of\ (sample)}$$

$\sum p(sample) r(sample)$

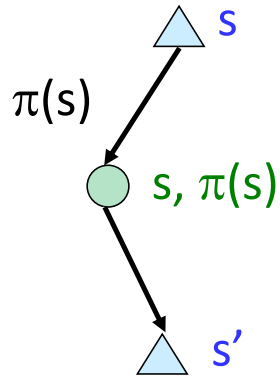
*But we can't rewind time to get sample after sample from state s!*

# Temporal Difference Learning

- Big idea: learn immediately from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)

在每一个sample之后就更新,而不是等到N个样本之后再取均值

- Temporal difference learning of values
  - (Policy still fixed, still doing evaluation!)
  - Move the value towards the sample




Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$ $\alpha \in (0,1)$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

# Exponential Moving Average

- Exponential moving average traditional: $\frac{1}{N}\Sigma x_n$
  - The running interpolation update: $\bar{x}_n = (1-\alpha)\cdot\bar{x}_{n-1} + \alpha\cdot x_n$

$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2\cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

  - Makes recent samples more important 使最近的权重加大
  - Forgets about the past (distant past values were wrong anyway) 忘记过去

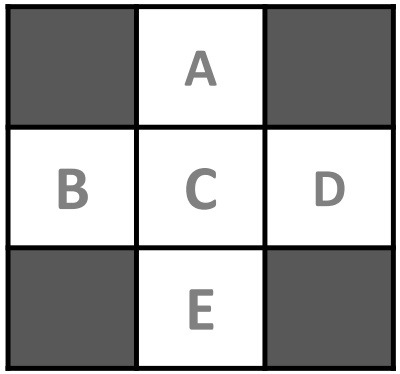
- Decreasing learning rate (alpha) can give converging averages

降低学习率($\alpha$)可以使平均值收敛.

# Example: Temporal Difference Learning

时序差分学习.

States




*Assume:* $\gamma = 1$, $\alpha = 1/2$

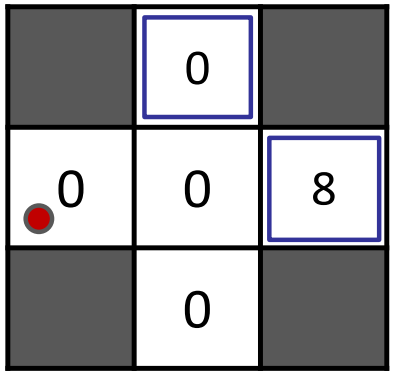
⇓

通常接近1.

Observed Transitions

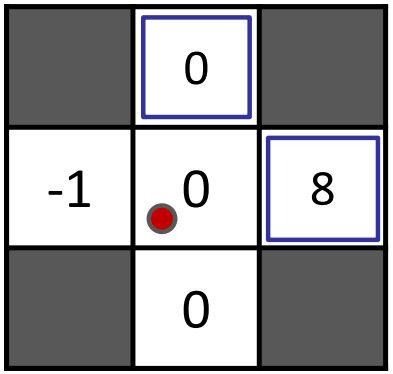
sample → update

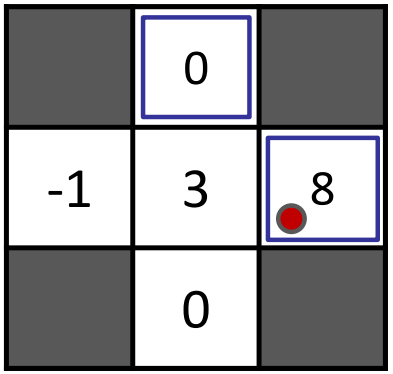
B, east, C, -2




sample → update

C, east, D, -2




$V^e(B) = 0.5V^e(B) + 0.5(-2) = -1$ (sample)

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

$V^e(C) = 0.5V^e(C) + 0.5(-2+8) = -3$

# Limitations of TD Value Learning

TD: $V^\pi(s)$ 只是一个数字，但实际上我们需要 $\pi(s)$，但 $Q(s,a)$ 包含 $a$

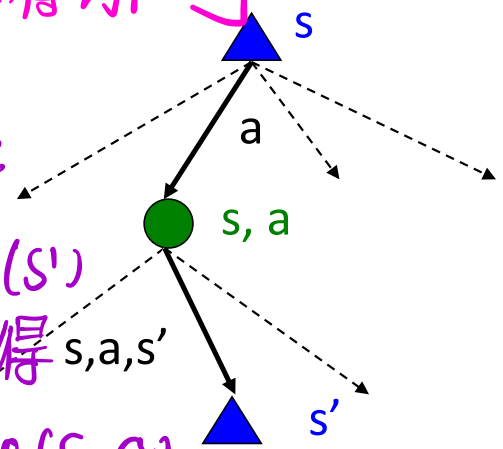
- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages 类似 Bellman by sample 平均
- However, if we want to turn values into a (new) policy…

$$\pi(s) = \arg\max_a Q(s,a)$$ 所以我们可以学习 $Q$ 来得到 policy

$$Q(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V(s') \right]$$

*Unknown!*

因为 $T, R$ 未知，所以无法直接从 $V(s')$ 推 $Q(s,a)$ 进而得 $\arg\max_a Q(s,a)$




- Idea: learn Q-values, not values
- Makes action selection model-free too!

⇒ 改进：直接使用 $Q$ 迭代

# Q-Learning

- Q-value iteration

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

- Q-Learning: learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

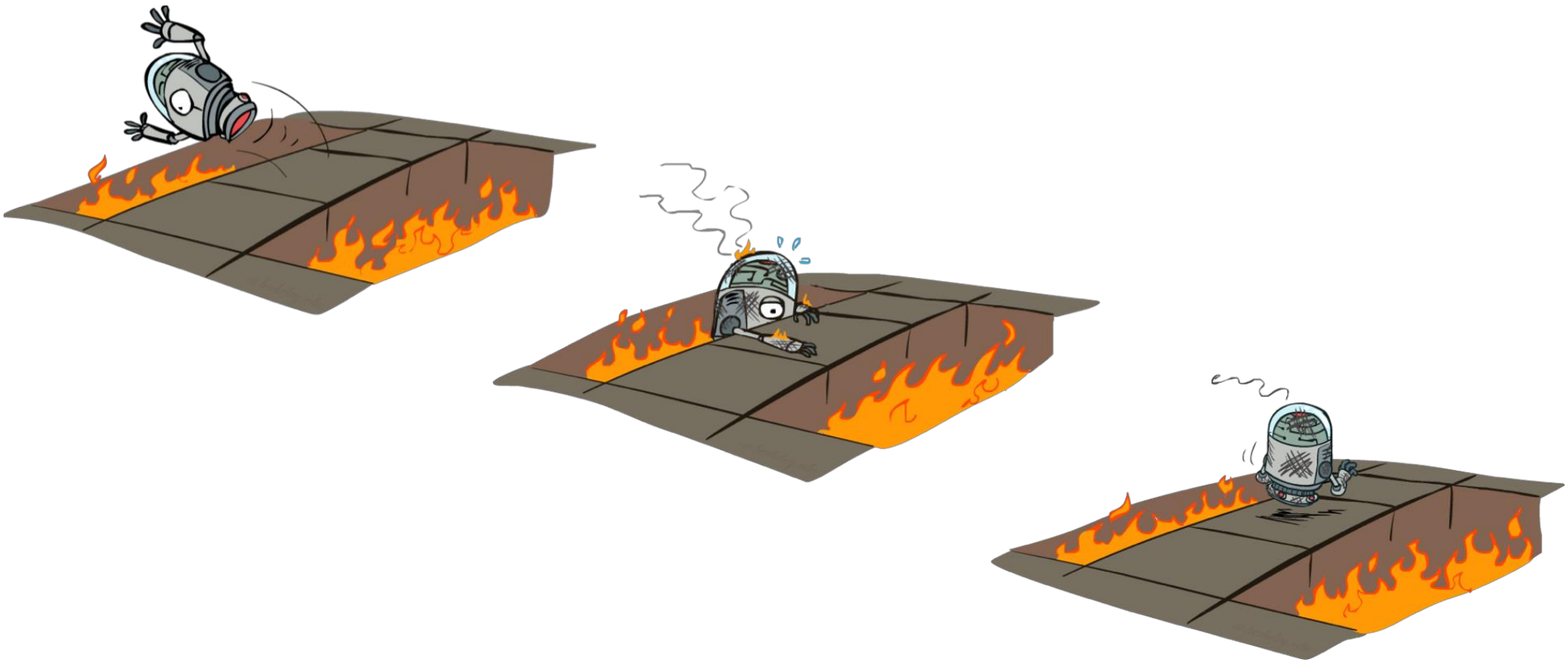
$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

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

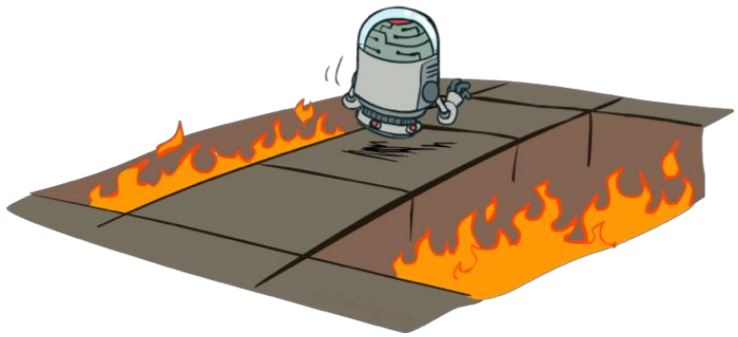
⇑以上都是给定了policy(α)

# Active Reinforcement Learning



# Active Reinforcement Learning

- Full reinforcement learning
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values



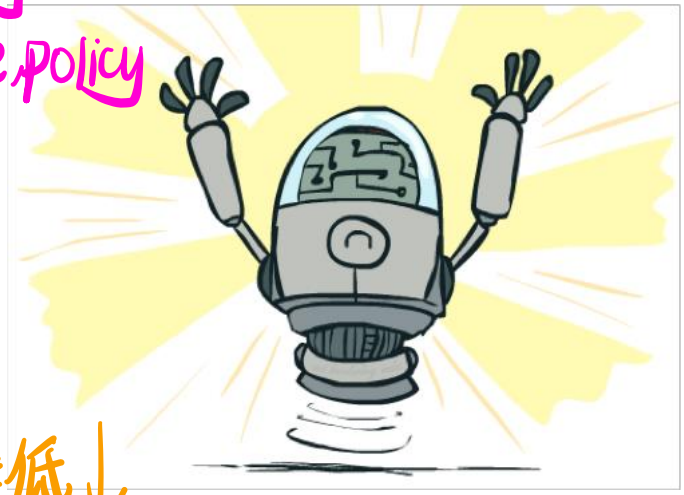
- Q-learning:
  - Learner makes choices (according to current values / policy, and also explore…)
    - Fundamental tradeoff: exploration vs. exploitation 探索与利用的权衡
  - This is NOT offline planning! You actually take actions in the world and find out what happens… online planning 在实际

# Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!

  尽管不是最优policy选择，但是会收敛到最优policy

- This is called off-policy learning 故称作offline policy



- Caveats:
  - You have to explore enough 探索多
  - You have to eventually make the learning rate small enough 学习率很低
  - … but not decrease it too quickly

学习率逐步降低↓

前期 → 快速吸收

后期 → 保持稳定.

# The Story So Far: MDPs and RL

已知MDP

## Known MDP: Offline Solution

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, π* | Value / policy iteration VI/PI |
| Evaluate a fixed policy π | Policy evaluation PE |

## Unknown MDP: Model-Based

写估计参数

估计MDP

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, π* | VI/PI on approx. MDP |
| Evaluate a fixed policy π | PE on approx. MDP |

## Unknown MDP: Model-Free

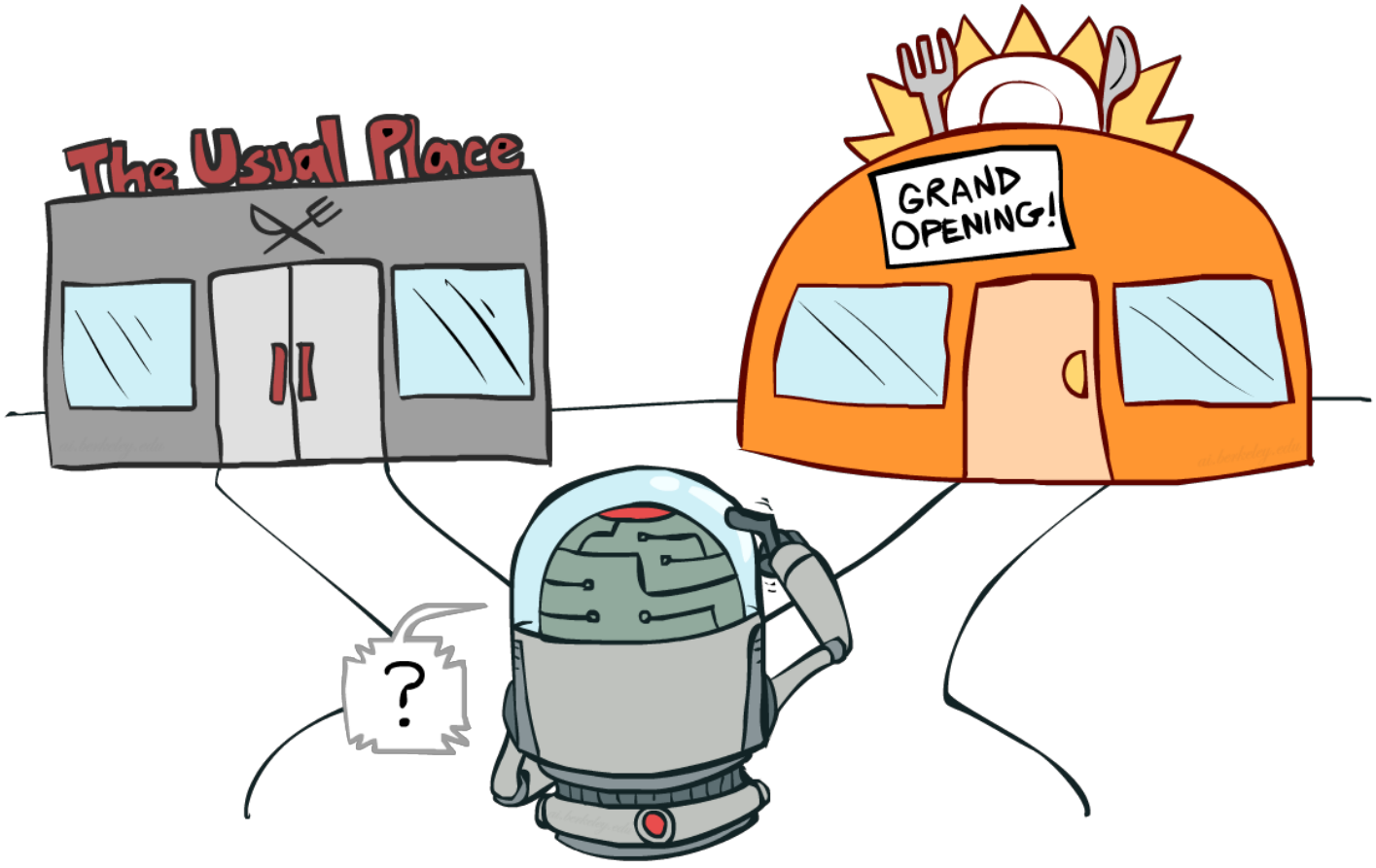
Q-learning

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, π* | Q-learning |
| Evaluate a fixed policy π | TD Value Learning |

TD 不做取 Max

temporary difference

# Exploration vs. Exploitation



# How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy

ε: 随机

1- ε: act current policy

# How to Explore?

- Several schemes for forcing exploration
  - Problems with random actions? 随机可能导致探索不全
    - You do eventually explore the space, but keep thrashing around once learning is done
  - One solution: lower $\varepsilon$ over time ①降低ε
  - Another solution: exploration functions ②探索函数

# Exploration Functions

- When to explore?
  - Explore states that haven't been sufficiently explored
  - Eventually stop exploring

鼓励探索更多的地方


- Idea: select actions based on modified Q-value
  - Exploration function: takes a Q-value estimate u and a visit count n, and returns an optimistic utility, e.g.

$$f(u, n) = u + k/n$$

k: 奖励; n: 访问的次数

我如果访问了很多次,那么这个地方reward降低

  - Q-Update

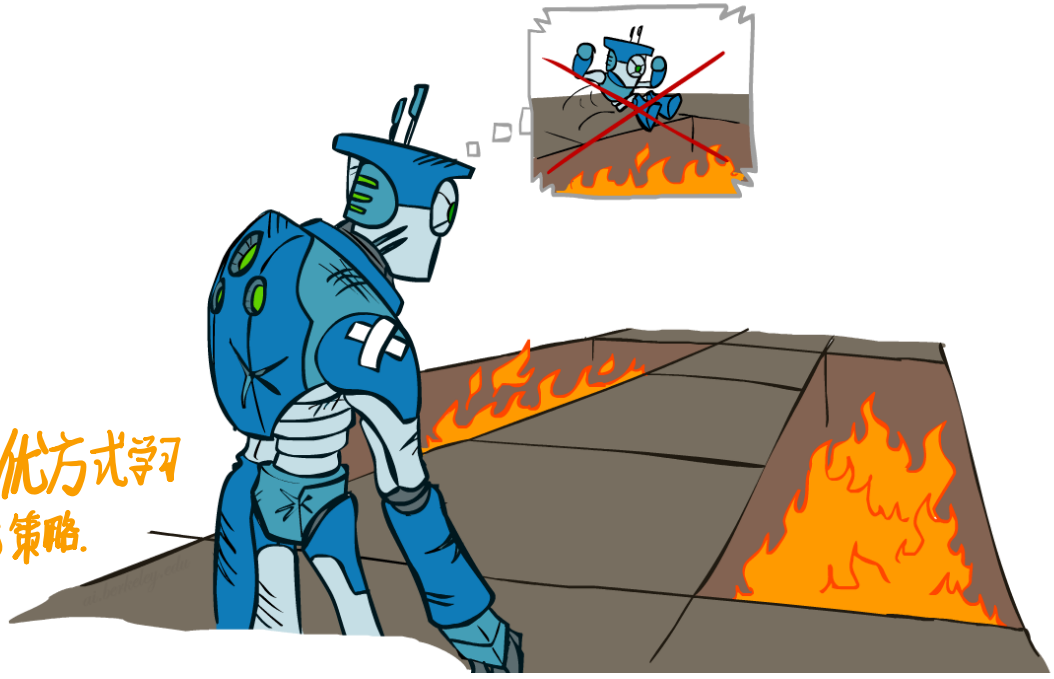
Regular Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

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

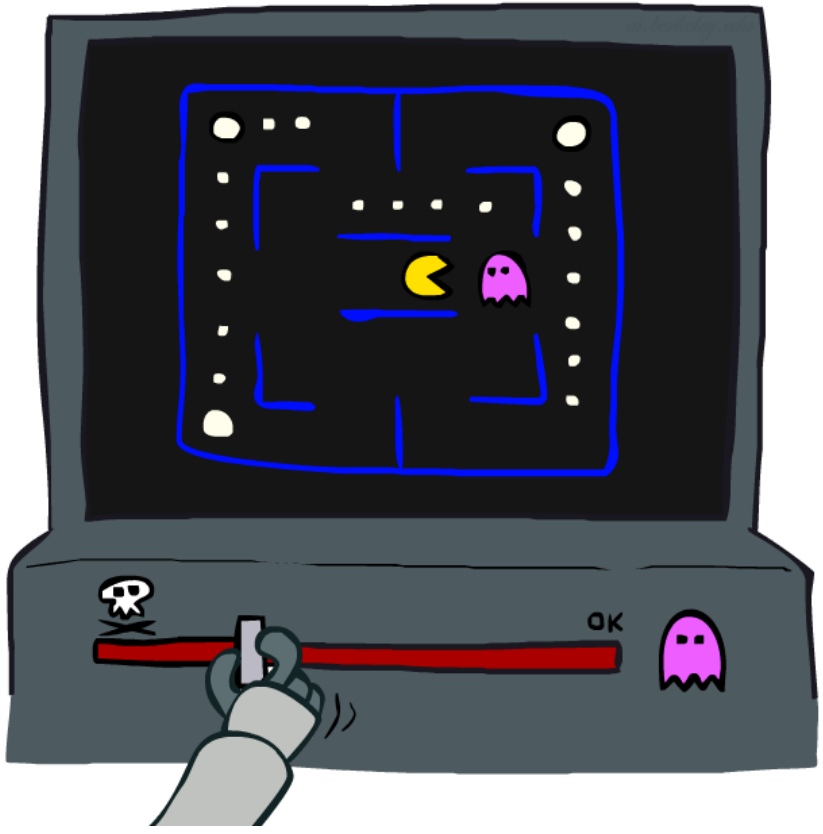
This propagates the "bonus" back to states that lead to under-explored states

# Regret

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards 期望与最优之间差异
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal 以最优方式学习 减少difference 最优策略.
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret eg. 随机探索带来很多次优选择.

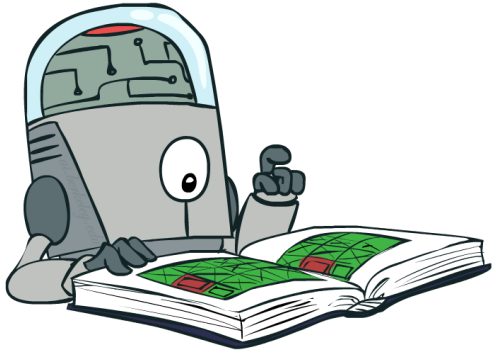


# Approximate Q-Learning

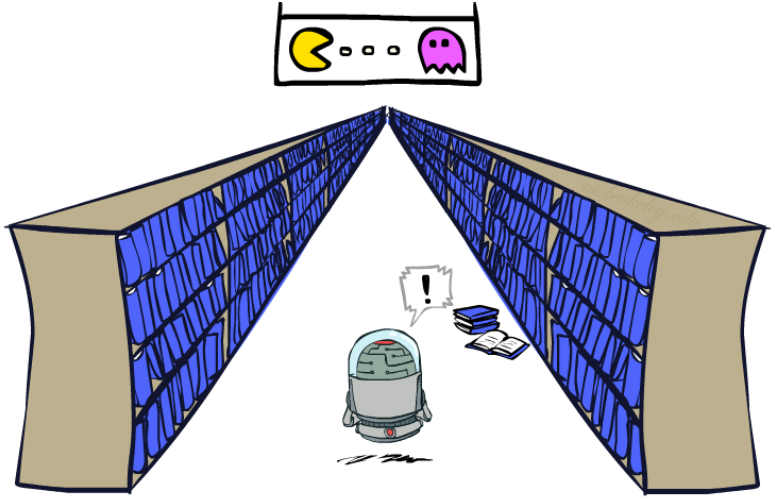


# Generalizing Across States

- Basic Q-Learning keeps a table of all q-values

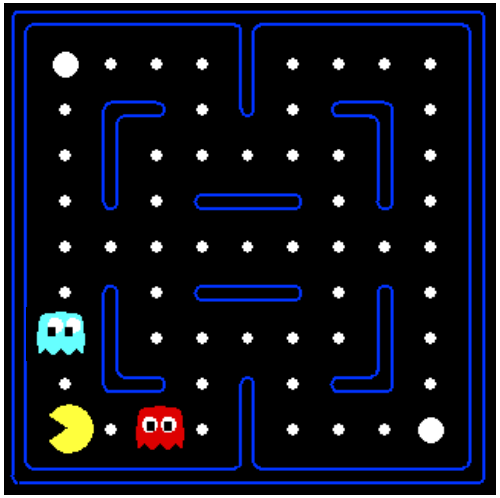


- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

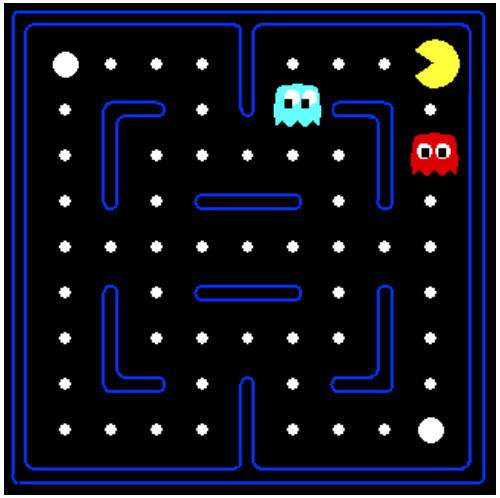


# Example: Pacman

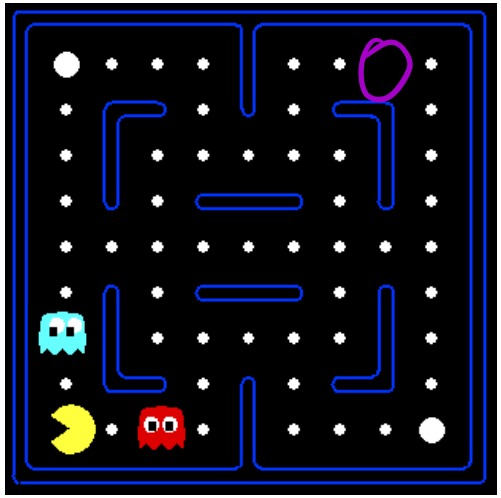
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



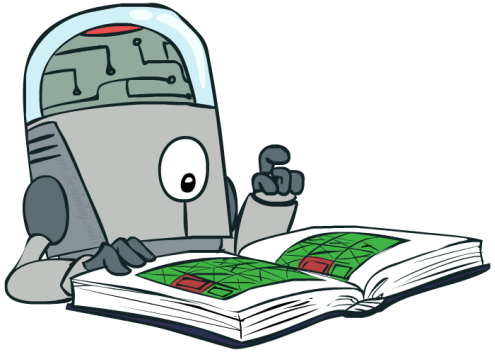
Or even this one!



理论上这几种情况应该共享

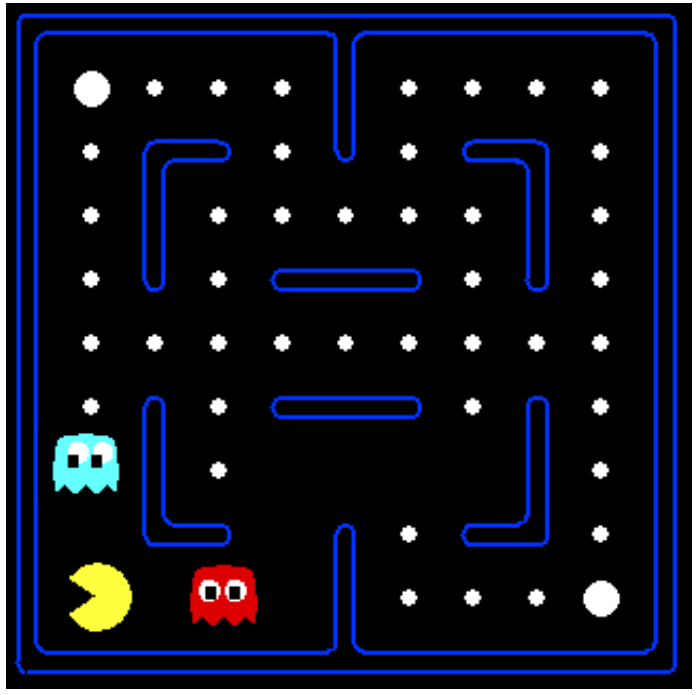
# Generalizing Across States

- We want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it again later



# Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)$^2$
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



# Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

feature 1 → $f_1(s)$ ... feature n → $f_n(s)$

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

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

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

# Approximate Q-Learning

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

- Q-learning with linear Q-functions:

Q:为什么叫它偏导啊?

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$

Exact Q's

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$

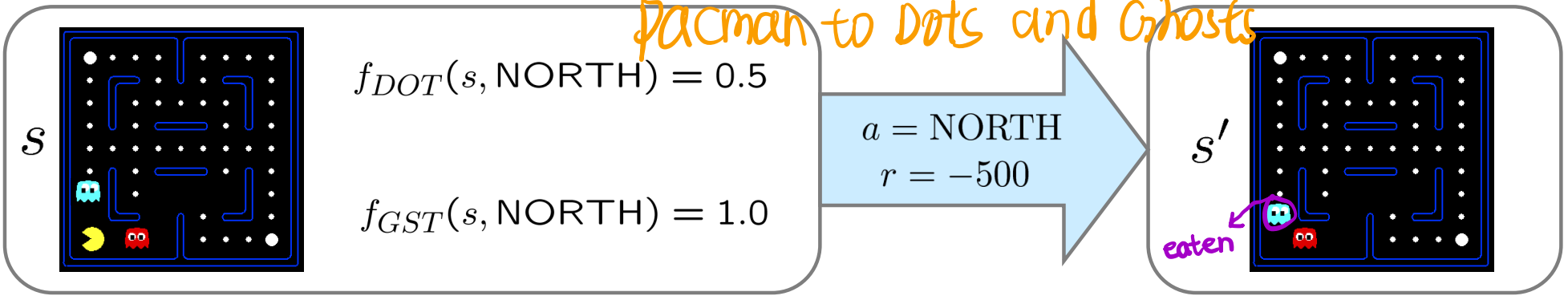
Approximate Q's
(based on online least squares)

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

# Example: Q-Pacman

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

Pacman to Dots and Ghosts




$Q(s, \text{NORTH}) = +1$

$Q(s', \cdot) = 0$

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

difference $= -501$

$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

If $\alpha = 0.004$: $Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$

# More Powerful Functions

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

自己定义的

Polynomial: $Q(s,a) = w_{11} f_1(s,a) + w_{12} f_1(s,a)^2 + w_{13} f_1(s,a)^3 + \ldots$

Neural network: $Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$

深度神经网络

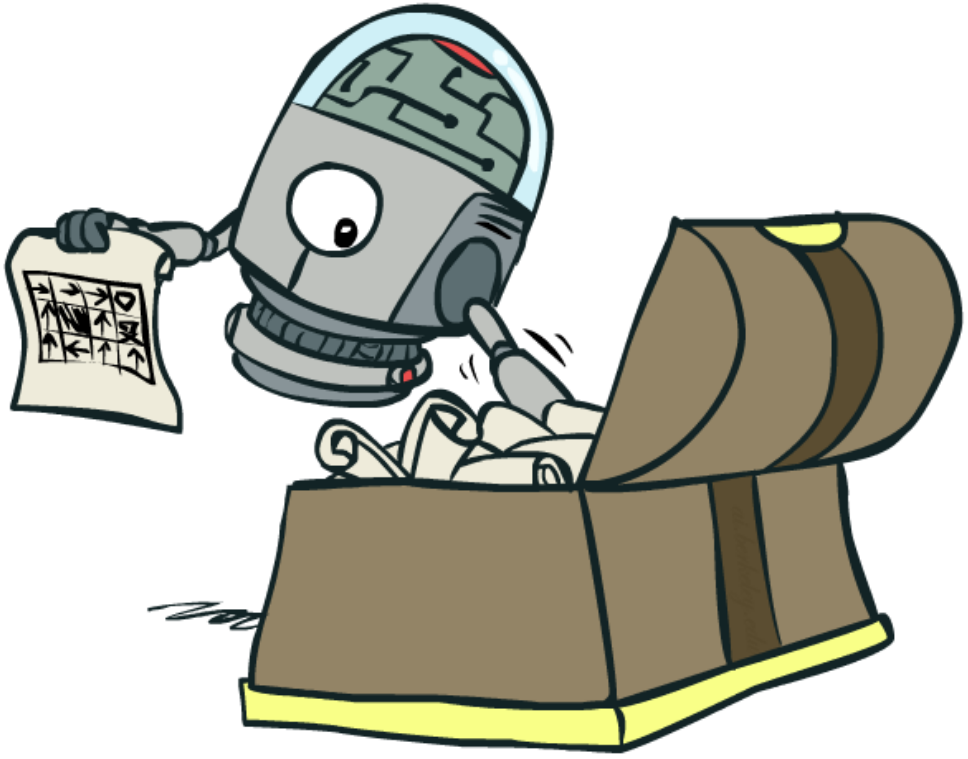
learn these too. 不手工定义，也通过learn

$$w_m \leftarrow w_m + \alpha \left[ r + \gamma \max_a Q(s',a') - Q(s,a) \right] \frac{dQ}{dw_m}(s,a)$$

$= f_m(s,a)$ in linear case

$\frac{dQ}{dw_n} = f_n(s,a)$ 在linear条件下

# Policy Search



# Policy Search

- Q-learning's priority: get Q-values close 应该是学习准确的Q值
- Observation: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from project 1b were probably horrible estimates of future rewards, but they still produced good decisions
  - The real priority: get ordering of Q-values right (action prediction)
- Idea: learn policies that maximize rewards, not the values that predict them
- Policy search: start with an OK solution (e.g., approximate Q-learning), then fine-tune feature weights to find a better policy

# Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Change each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

# Policy Search

- Q-learning's priority: get Q-values close
- Observation: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from PA 1b were probably horrible estimates of future rewards, but they still produced good decisions
  - The real priority: get ordering of Q-values right (action prediction)

- Idea: learn policies that maximize rewards, not the values that predict them
- Policy search: start with an OK solution (e.g., approximate Q-learning), then fine-tune feature weights to find a better policy

# Summary

- Reinforcement learning
  - MDP without knowing T and R
- Model-based learning
- Model-free learning
  - Policy evaluation: TD Learning
  - Computing q-values/policy: Q-Learning
- Exploration vs. Exploitation
  - Random exploration, exploration function
- Feature-based representation of states
- Policy Search