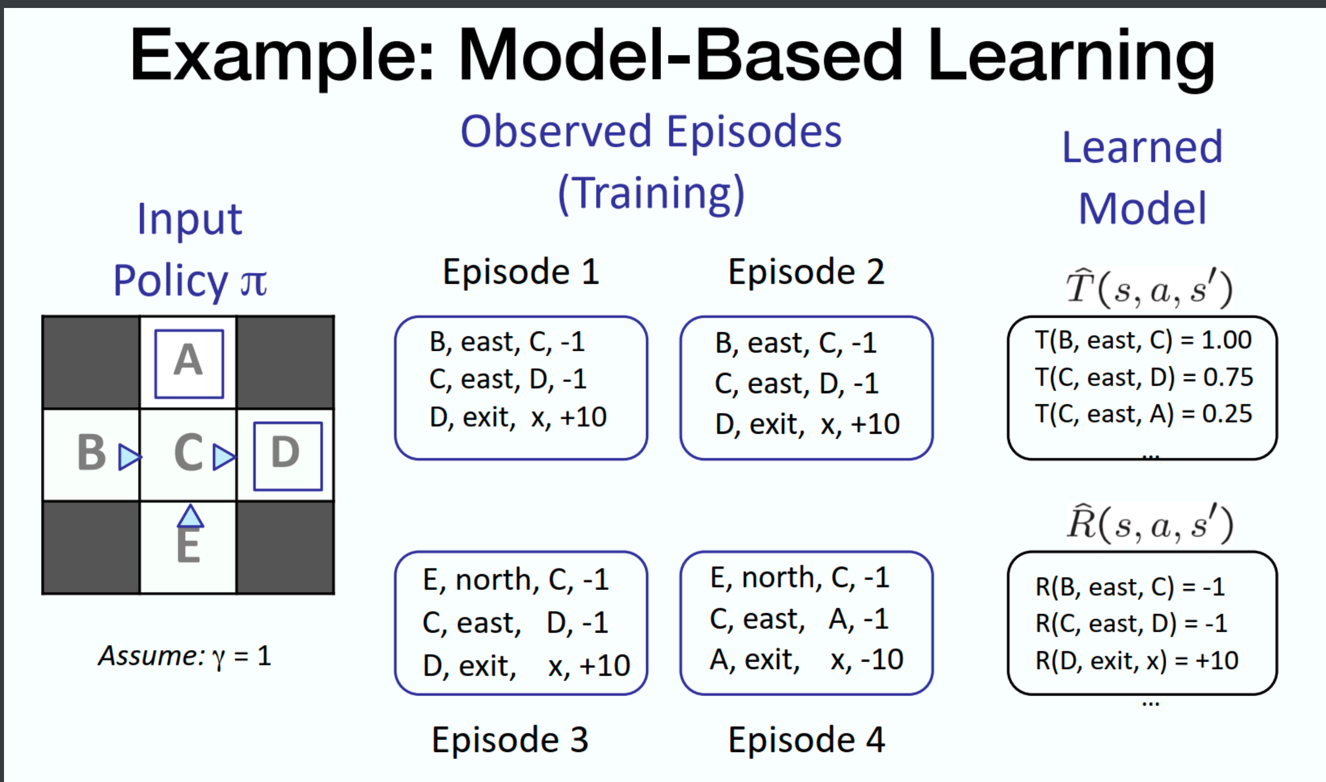
Reinforcement learning (1)

model-based

- 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)
- New twist: don’t know T or R

- idea: learn empirical MDP model -> solve the learned MDP
- example



- model-based是先学一个完整model（T和R），model-free是绕过model

---

model-free

passive reinforcement learning (value learning)

- task: policy evaluation

  given a fixed policy (still don't know T/R), learn the state values

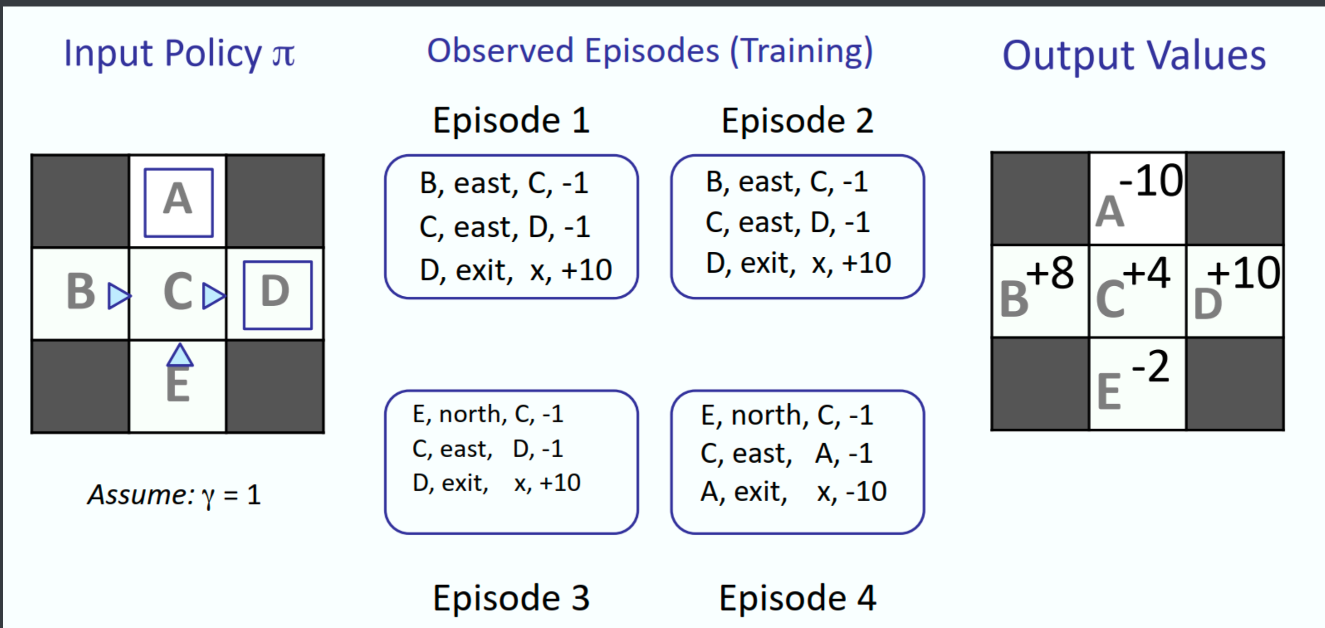
- learner just executes the policy and learns from experience!

  this is NOT offline planning!

  offline planning 是已知T和R，用value iteration或policy iteration的方式提前规划出最优行动

- direct evaluation

  average observed sample values



  easy to perform, but wastes information about state connections

- temporal difference learning

  learn from every experience:

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

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

  makes recent samples more important; alpha is learning rate

  decreasing learning rate can given converging averages

  problem: not able to turn values into a better policy

  the idea would be, to learn Q-values rather than values (you choose the actions!)

active reinforcement learning (q-learning)

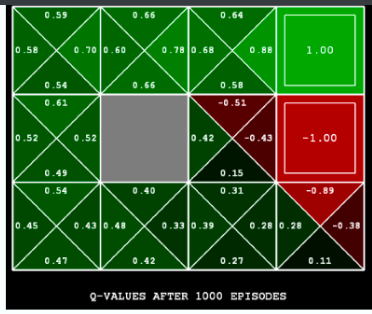
- full reinforcement learning: don't know transitions / rewards, choose the actions to learn the optimal policy / values
- recap: q-value iteration

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

- 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:

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

q-learning converges to optimal policy, even the actions are suboptimal;

but, explore enough, eventually the learning rate should be small enough

summary of the algorithms

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

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

Unknown MDP: Model-Free

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

---

exploration vs. exploitation

- random actions

  - 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

  <u>problem:</u>

  keep thrashing around once learning is done

  （掌握了最优policy之后依然有epsilon的概率随机选择动作）

  <u>solution 1:</u> lower epsilon over time

  <u>solution 2:</u> exploration functions

- exploration functions

  <u>basic idea:</u>

  introduce a visit count, explore not-enough-visited areas with higher priority

  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

  when n is small, f would be larger

  when n is large, f would be approaching the q value

  <u>prioritized q-learning update function:</u>

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

  <u>bonus propagation:</u>

  if s' is an unknown state, then Q(s', a') would be large, which would be propagated to Q(s, a)

- regret

  衡量学习过程中的错误代价 (we wanna optimally learn to be optimal)

  random exploration has higher regret than exploration functions

---

approximate q-learning

in realistic situations we cannot possibly learn about every single state

-> generalization is important

- feature-based representations

  a vector,

  e.g. in pacman features might be distance to closest ghost / dot etc.

- linear value functions

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

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

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

- approximate Q-learning

  - Q-learning with linear Q-functions:

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

  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

- reasoning: minimizing error

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

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

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

  Approximate q update explained:

  $$w_m \leftarrow w_m + \alpha \left[\underbrace{r + \gamma \max_a Q(s', a')}_{\text{“target”}} - \underbrace{Q(s,a)}_{\text{“prediction”}}\right] f_m(s,a)$$

target is sample. prediction is current q-value