17 MAKING COMPLEX DECISIONS

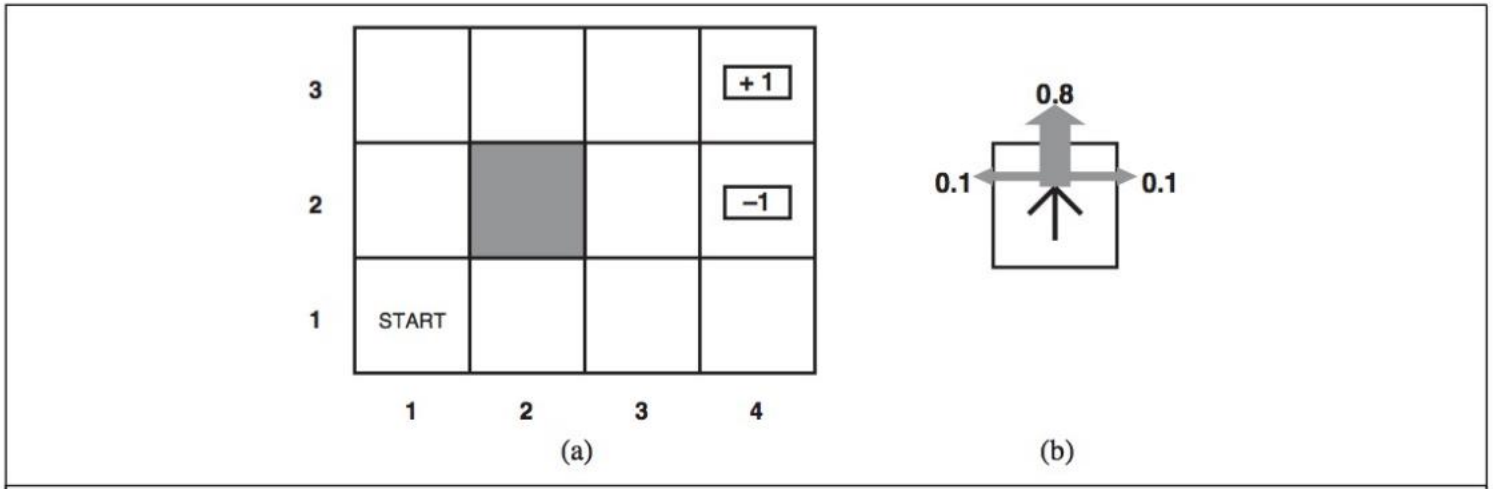
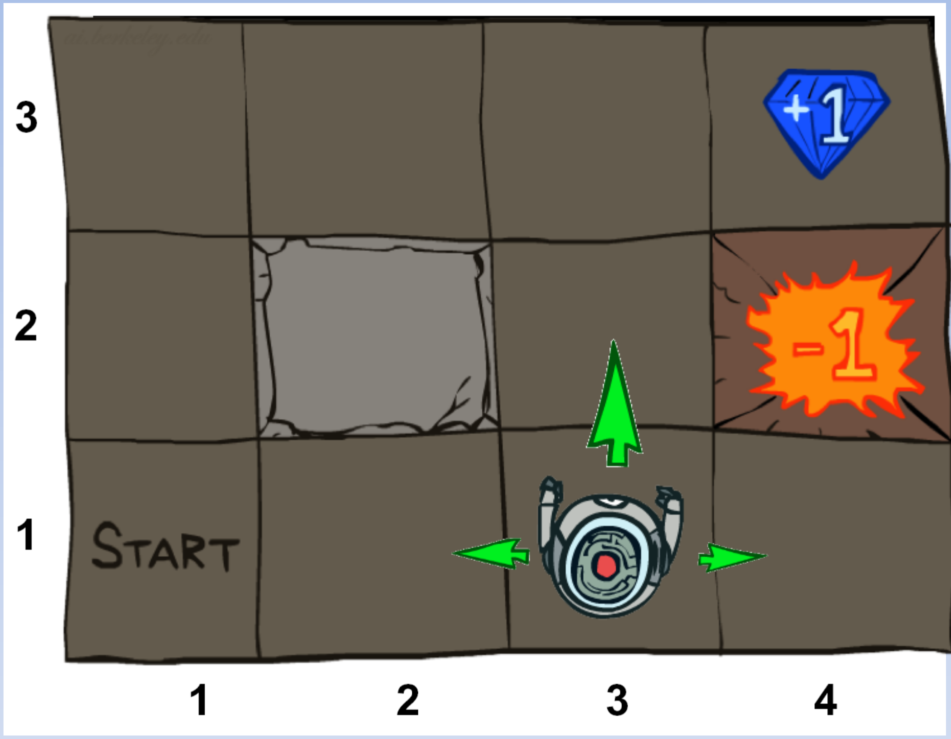


Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

Grid World

- An MDP is defined by:
    - A set of states s ∈ S
    - A set of actions a ∈ A
    - A transition function T(s, a, s')
        - Probability that a from s leads to s', i.e., P(s' | s, a)
        - Also called the model or the dynamics
    - A reward function R(s, a, s')
        - Sometimes just R(s) or R(s')
    - A start state
    - Maybe a terminal state



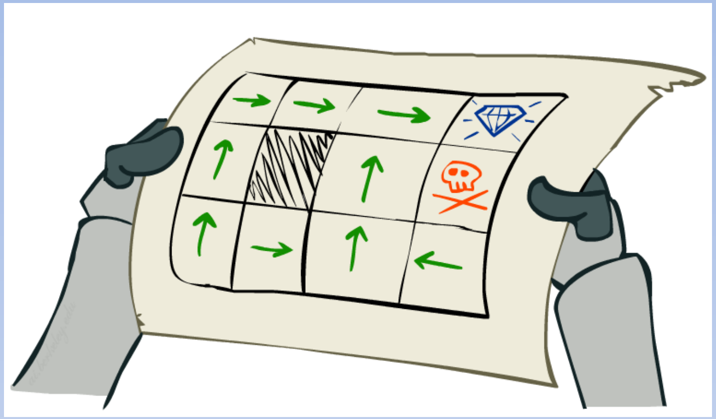
Policies

- With MDP recognize that we never know for sure when we might end up in any particular state.
- Need to know what to do not just from the start state, but from any state!!!
  - Sounds costly.
  - Need generalization (later).

Policies

- For MDPs, we want an optimal policy
  - $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

- Expectimax didn't compute entire policies
  - It computed the value for a single state only



Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

Rewards

- R(s) or R(s,a,s’)
- Expectiminimax only worried about terminal states.
- How should we evaluate the sequence of rewards??

Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

Worth Next Step

γ^2

Worth In Two Steps

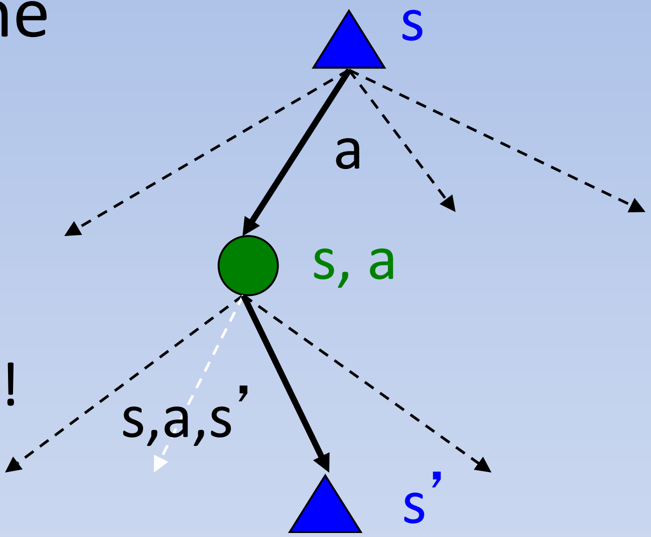
Solving MDP's

- Problem Review: Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
    - sometimes R(s)

- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

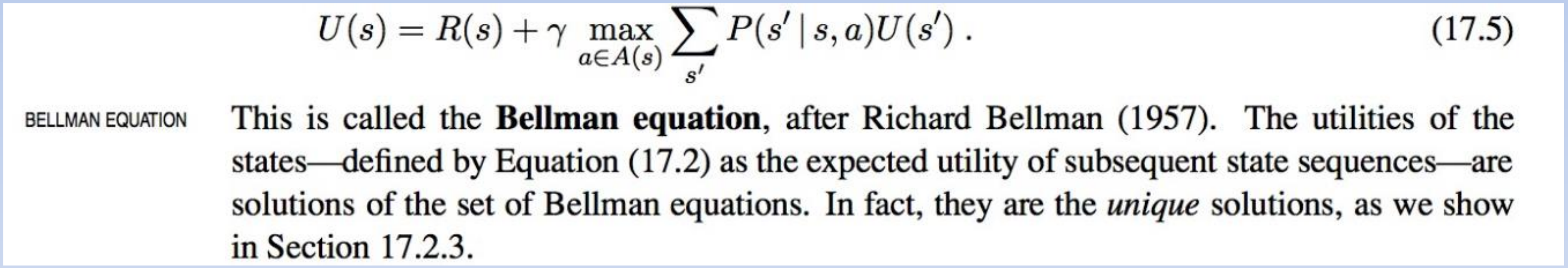
Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!
- Can we define the utility/value in a closed form (recursively).



Bellman Equation

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$



$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s,a)U(s') \,. \tag{17.5}$$

BELLMAN EQUATION

This is called the **Bellman equation**, after Richard Bellman (1957). The utilities of the states—defined by Equation (17.2) as the expected utility of subsequent state sequences—are solutions of the set of Bellman equations. In fact, they are the *unique* solutions, as we show in Section 17.2.3.

Closer Look

$$\sum_{s'} P(s'|s,a)U(s')$$

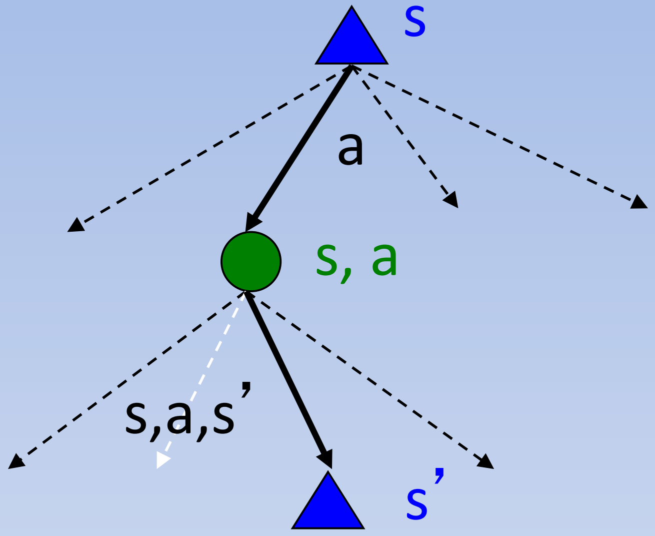
- For every possible state (s’) weight its utility U(s’) by the probability of moving to that state when taking action a in state s.
- The sum is the Expected Utility of action a in state s.
- We want to take the action with the highest expected utility.

Bellman Equation

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

- The expected utility of the best action is then discounted by γ and added to the utility of the current state.
- Total utility is sum of:
  - Utility of current state.
  - Discounted expected utility of the best action for this state.

Bellman Equation



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

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

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

- Alternative with R(s, a, s')
- T(s, a, s') = P(s' | s, a)
- Rewards are given when action is taken.

Grid World

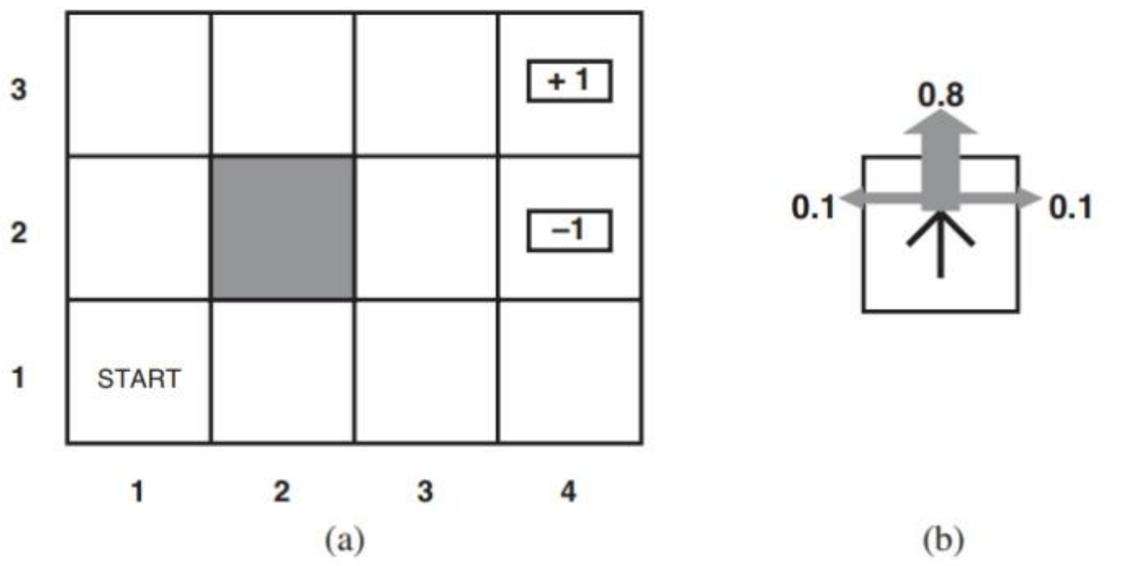
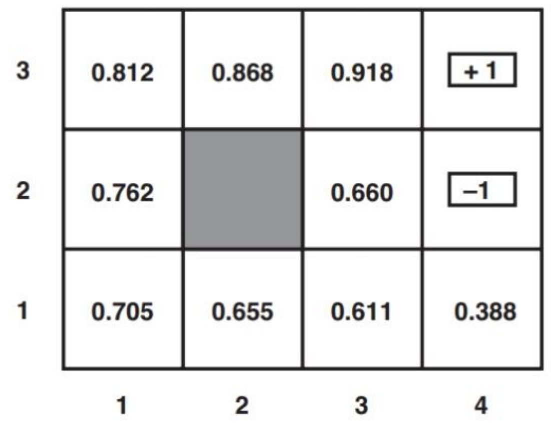


Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

Example



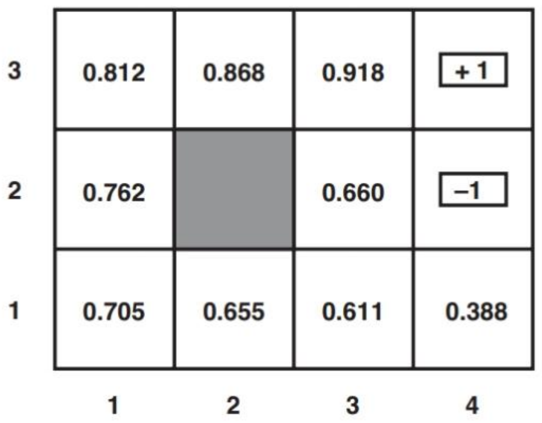
3	0.812	0.868	0.918	$\boxed{+1}$
2	0.762	▨	0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388
	1	2	3	4

Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

- Trying action up in state (1,1)
- P((1,2)|(1,1), up) = 0.8
- P((2,1)|(1,1), up) = 0.1
- P((1,1)|(1,1), up) = 0.1 , accidently going left

Q((1,1), up) = $0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1)$,

Example



3	0.812	0.868	0.918	$\boxed{+1}$
2	0.762	▨	0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388
	1	2	3	4

Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

$$U(1,1) = -0.04 + \gamma \max[$$

(up) $0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1),$

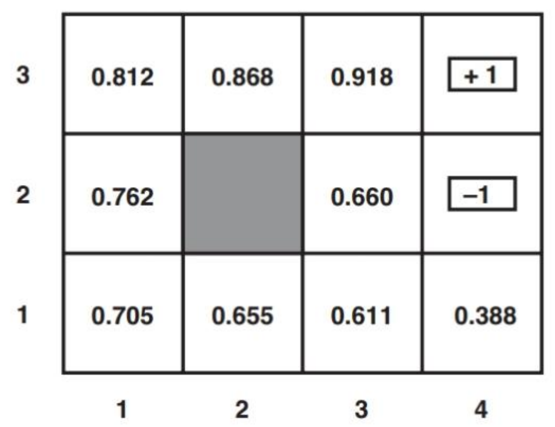
(left) $0.9U(1,1) + 0.1U(1,2),$

(right) $0.9U(1,1) + 0.1U(2,1),$

(down) $0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)$

$]$

Example



3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

$$U(1,1) = -0.04 + \gamma \max[$$

(up)0.7456,

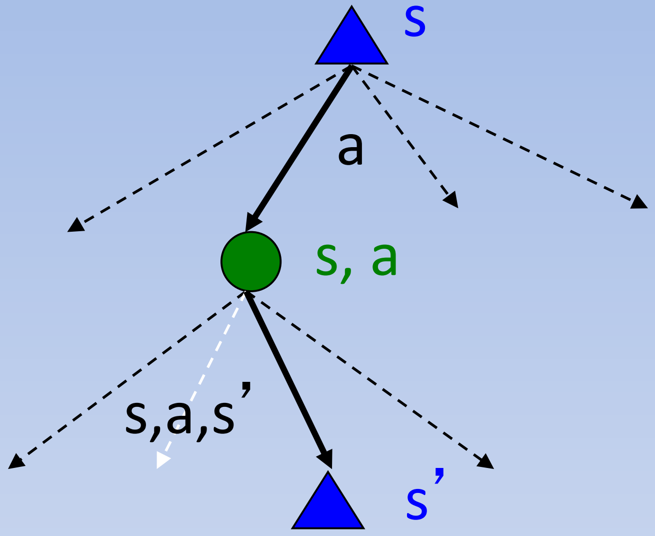
(left) $0.7107,$

(right)0.7

(down)0.6707

$$]$$

Bellman Equation



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

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

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

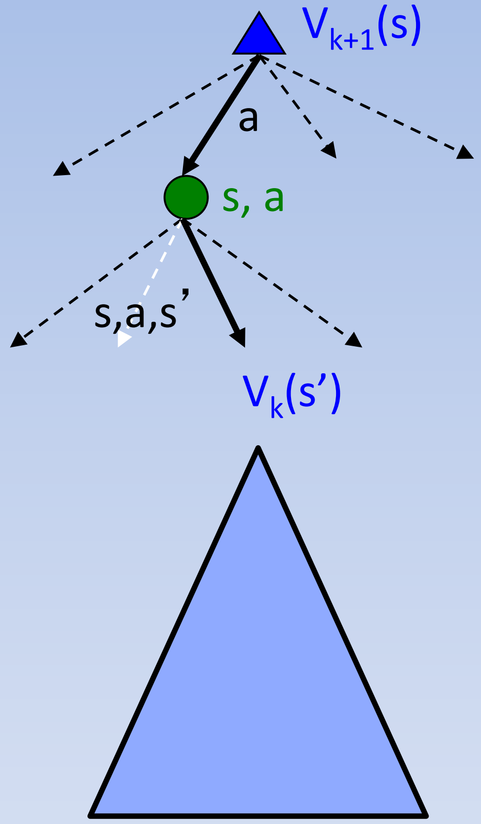
- Alternative with R(s, a, s')
- T(s, a, s') = P(s' | s, a)
- Rewards are given when action is taken.

Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

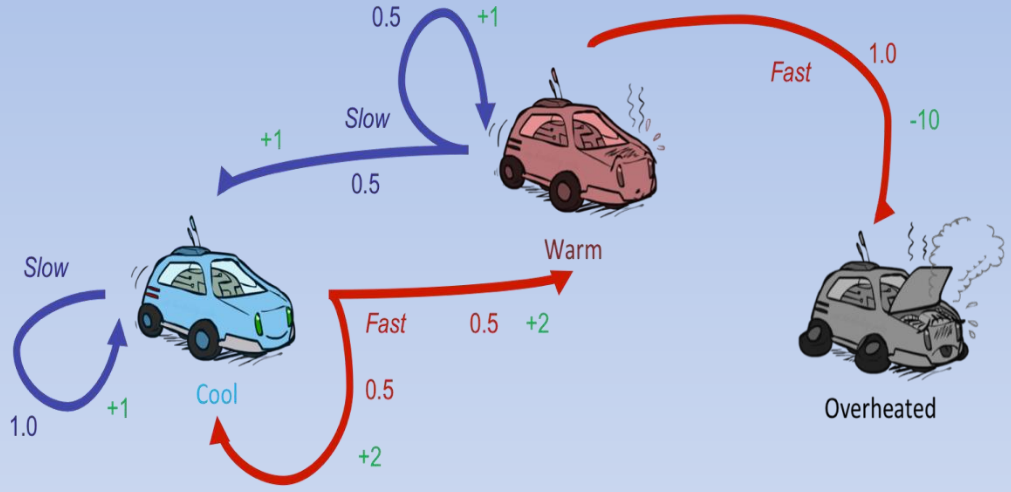
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do



Example: Value Iteration

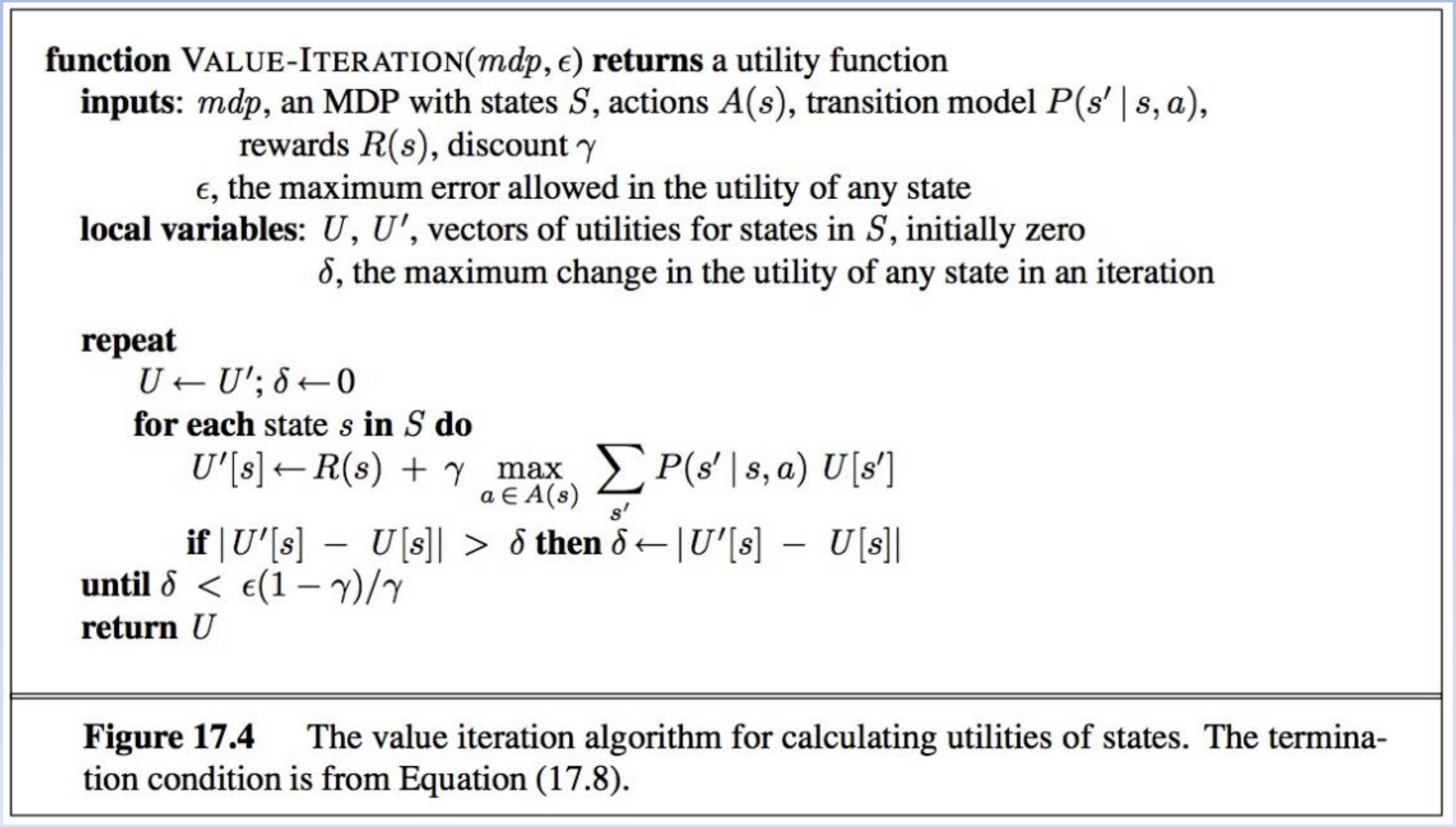
V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

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

Value Iteration w/ Textbook R(s)



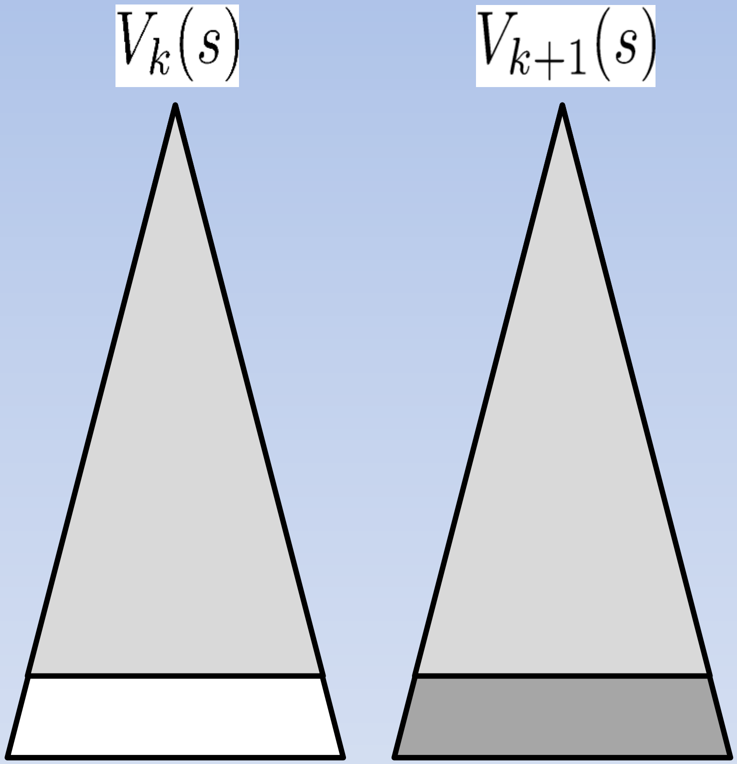
```
function VALUE-ITERATION($mdp, \epsilon$) returns a utility function
  inputs: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$,
              rewards $R(s)$, discount $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U$, $U'$, vectors of utilities for states in $S$, initially zero
                   $\delta$, the maximum change in the utility of any state in an iteration

  repeat
      $U \leftarrow U'; \delta \leftarrow 0$
      for each state $s$ in $S$ do
          $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
          if $|U'[s] - U[s]| > \delta$ then $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return $U$
```

Figure 17.4 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

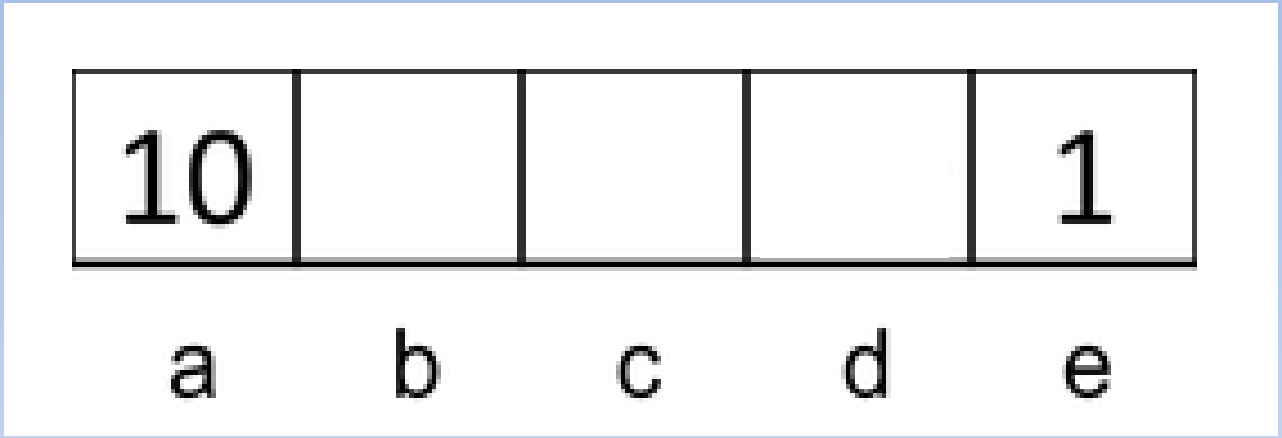
Convergence*

- How do we know the V_k vectors are going to converge?

- Case 1: If the tree has maximum depth M, then V_M holds the actual untruncated values
- Case 2: If the discount is less than 1
  - Sketch: For any state V_k and V_{k+1} can be viewed as depth k+1 expectimax results in nearly identical search trees
  - The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
  - That last layer is at best all R_{MAX}
  - It is at worst R_{MIN}
  - But everything is discounted by γ^k that far out
  - So V_k and V_{k+1} are at most $\gamma^k \max|R|$ different
  - So as k increases, the values converge

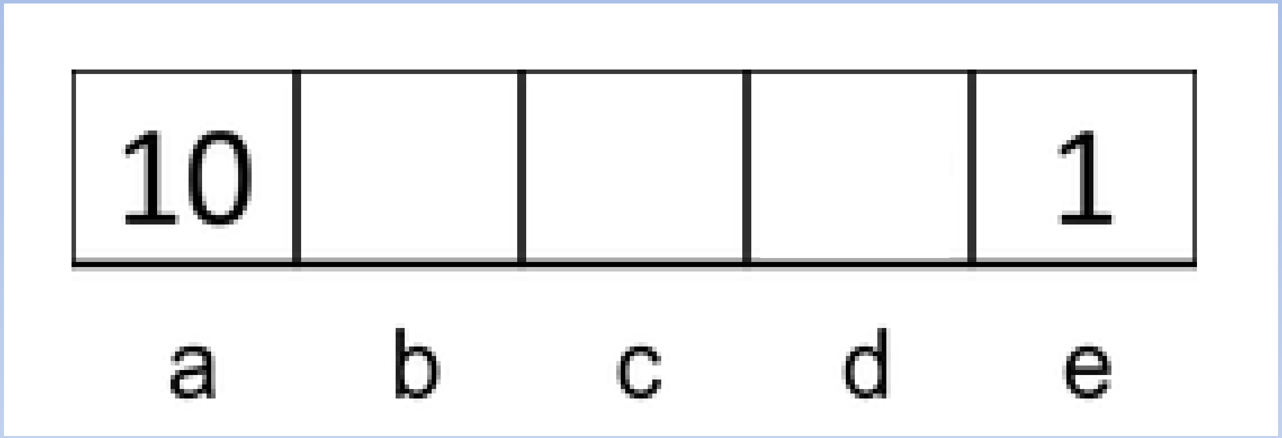


Discounting + Stochastic Element



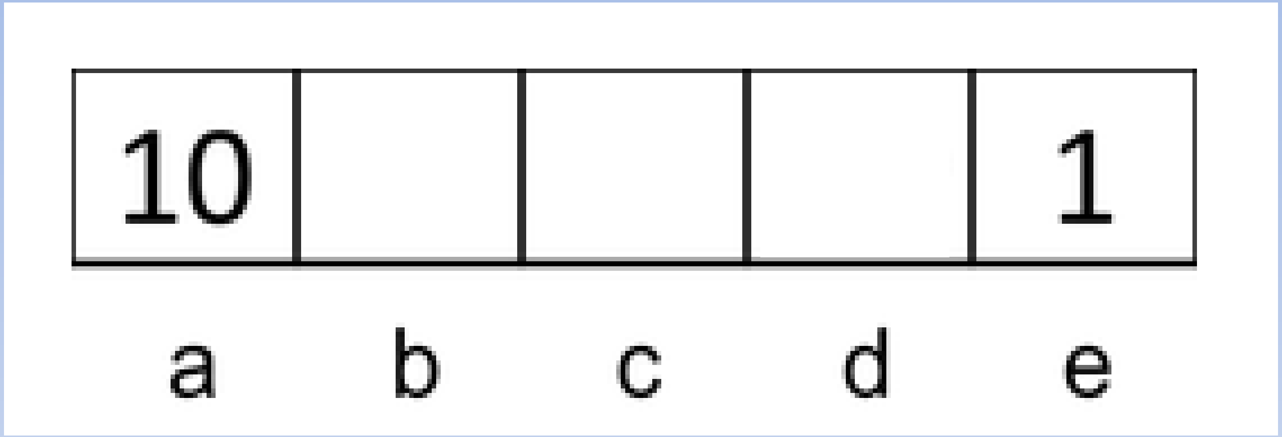
- Our World:
    - 5 State = {a, b, c, d, e}
    - 3 Actions = {Left, Right, Exit}
- **Transitions are no longer deterministic**
- Left and Right are successful 80% of the time.
    - When not success, the agent stays in place.
- Exit is successful 100% of the time
- State = {a, b, c, d, e}
- $\gamma = 0.2$
- Calculate V*(state) = V_{∞}(state)

Discounting + Stochastic Element



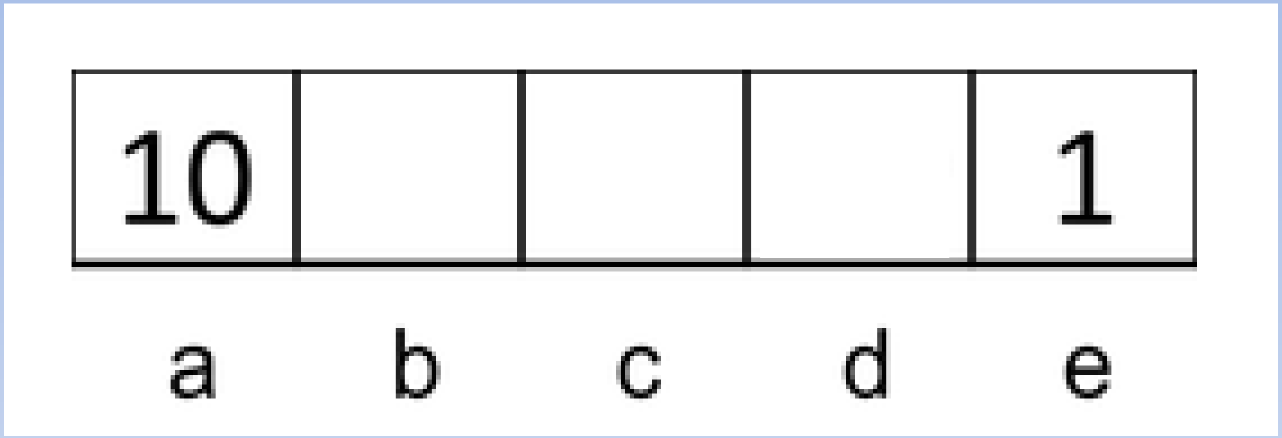
- V*(a) =

Discounting + Stochastic Element



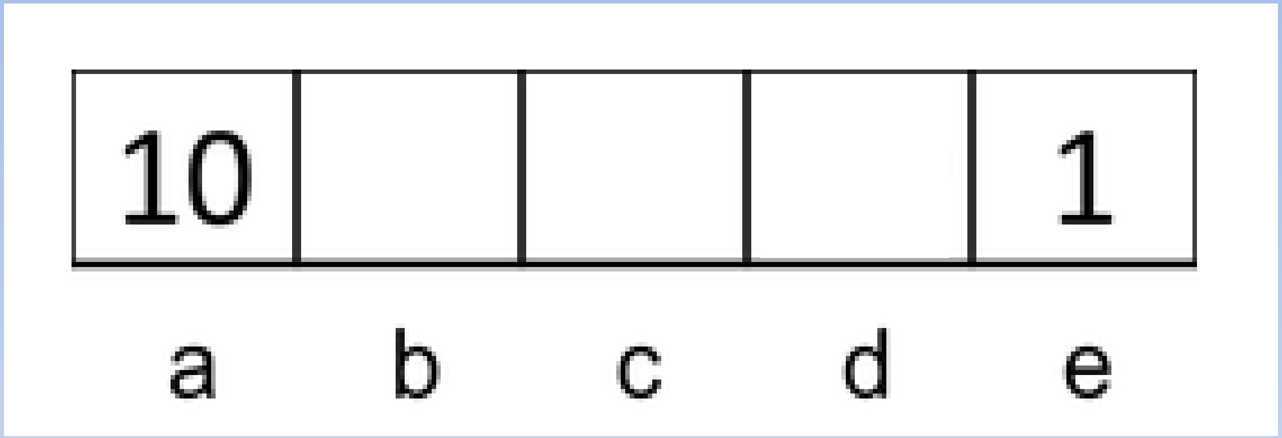
- $V_0(a) = 0$
- $V_0(b) = 0$
- $V_0(c) = 0$
- $V_0(d) = 0$
- $V_0(e) = 0$

Discounting + Stochastic Element



- $V_{\infty}(a) = 10$
- $V_{\infty}(e) = 1$

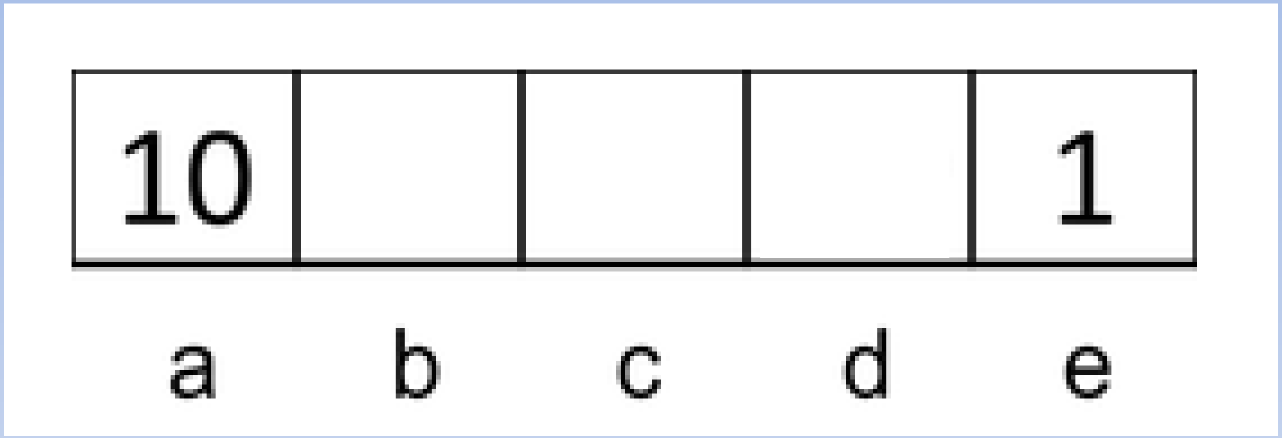
Discounting + Stochastic Element



$$U_{k+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_k(s')$$

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

Discounting
+ Stochastic Element



- $V_\infty(a) = 10$
- $V_\infty(b) = 2$
- $V_\infty(c) = 0.4$
- $V_\infty(d) = 0.2$
- $V_\infty(e) = 1$

- The following MDP world consists of 5 states and 3 actions:

(1, 1) Actions: down, right	(1, 2) Action: Exit = -10
(2, 1) Actions: down, right	(2, 2) Action: Exit = -10
(3, 1) Action: Exit = 10	

- When taking action down, it is successful with probability 0.6, otherwise you go right.
- When taking action right, it is successful with probability 0.6, otherwise you go down.
- When taking action Exit, it is successful with probability 1.0.
- The only reward is when taking action Exit, and there is no discounting.

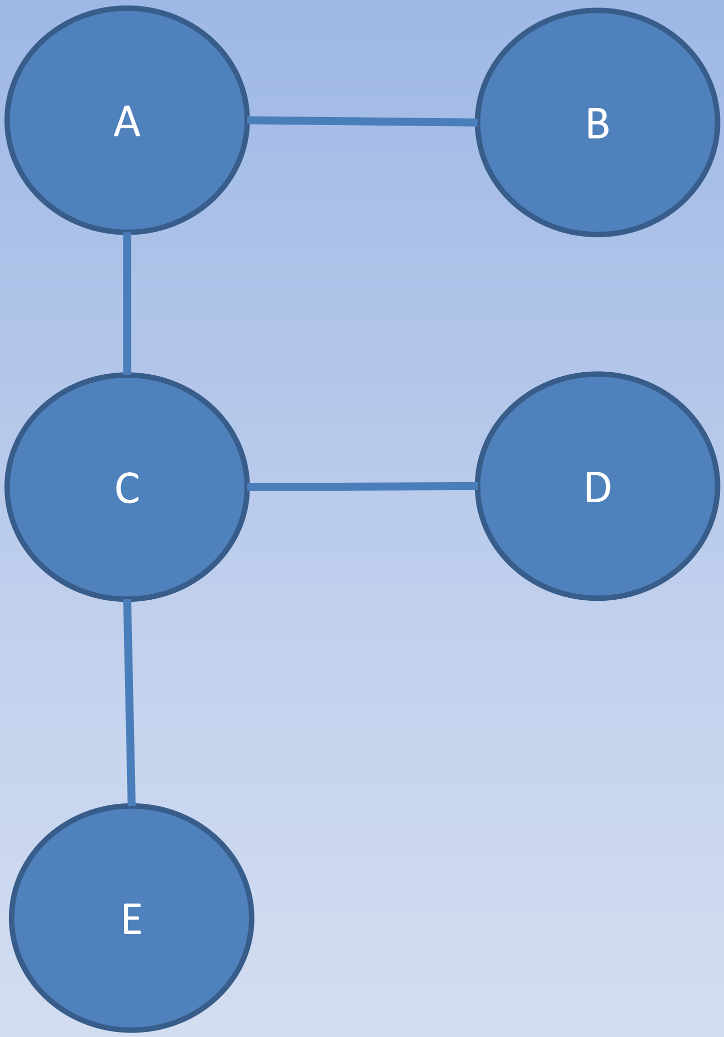
- Calculate the value of states using Value Iteration algorithm showing work for time step K=0, 1, 2

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

$V_0(1, 1)=$ $V_1(1, 1)=$ $V_2(1, 1)=$	$V_0(1, 2)=$ $V_1(1, 2)=$ $V_2(1, 2)=$
$V_0(2, 1)=$ $V_1(2, 1)=$ $V_2(2, 1)=$	$V_0(2, 2)=$ $V_1(2, 2)=$ $V_2(2, 2)=$
$V_0(3, 1)=$ $V_1(3, 1)=$ $V_2(3, 1)=$	

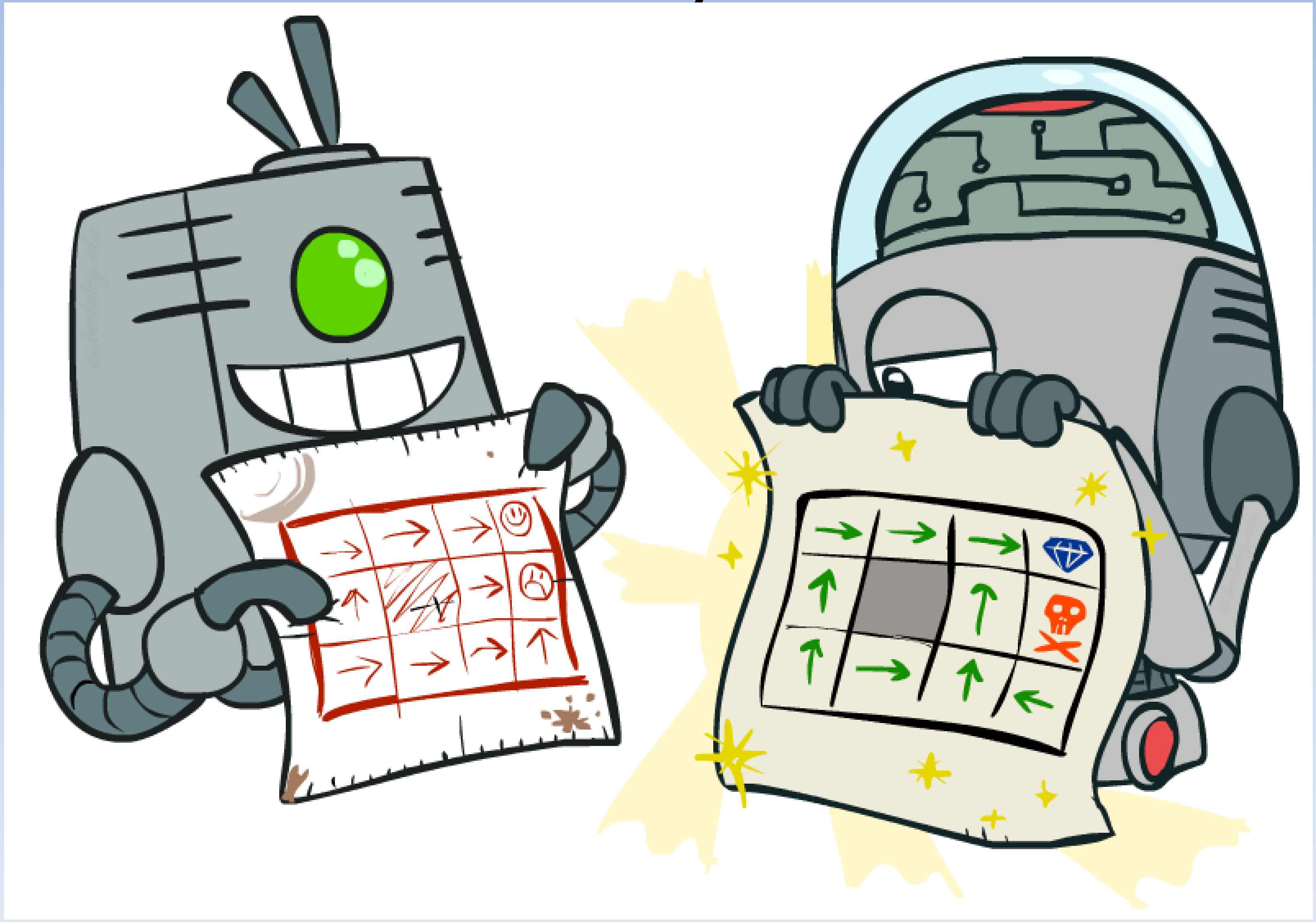
Graph

- R(B)=-1
- R(D)= -10
- R(E) = 100
- R(A) = 0
- R(C) = 0

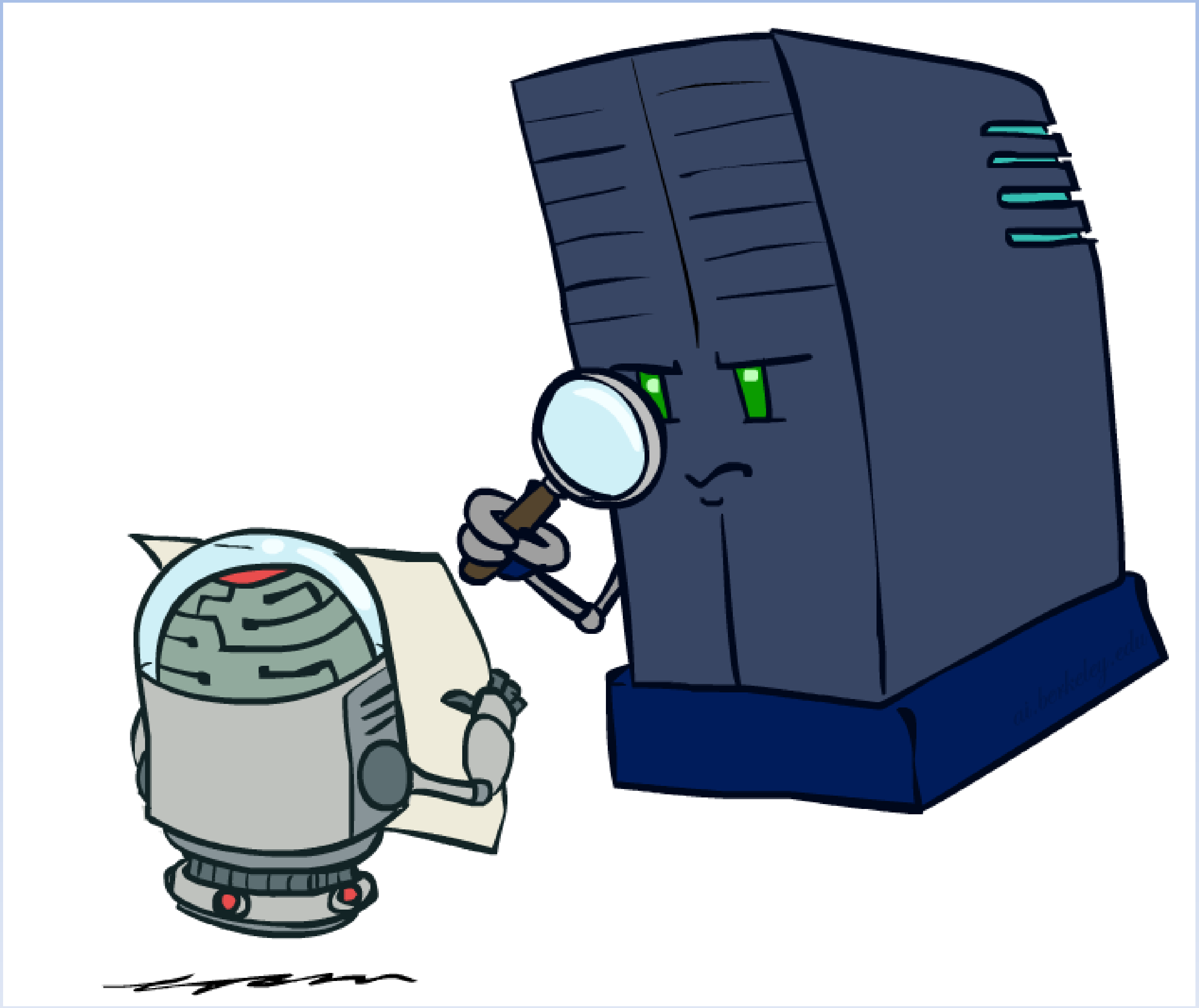


- $U_{k+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_k(s')$

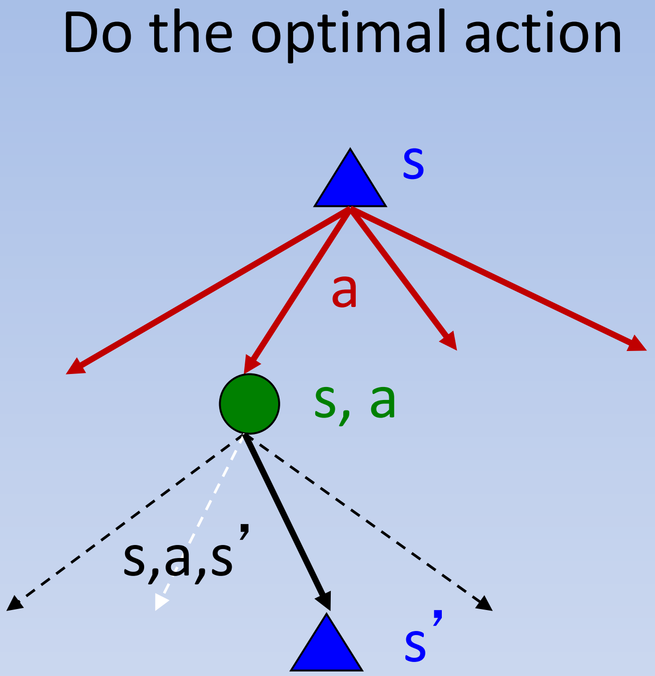
Policy Methods:
Which Policy is Better?

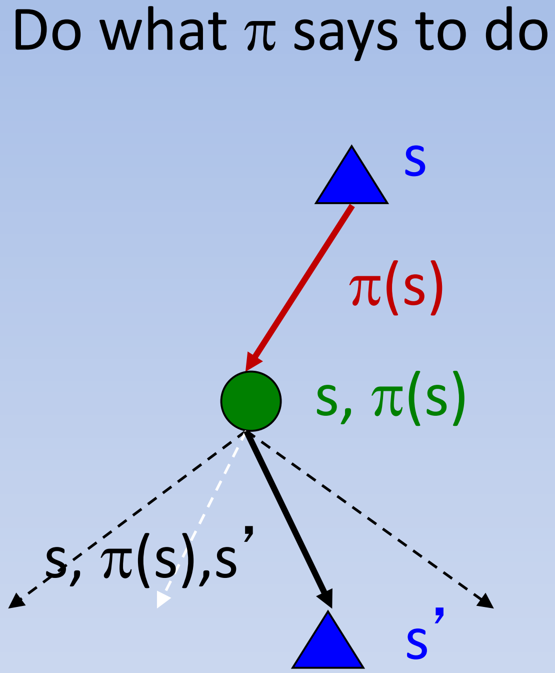


Policy Evaluation



Fixed Policies





- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

Utilities for a Fixed Policy

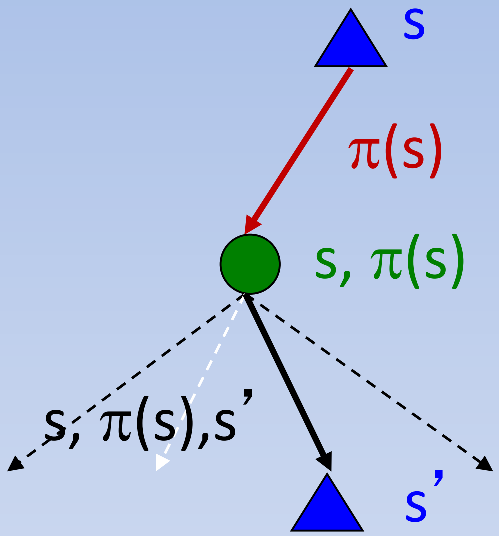
- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- Define the utility of a state s, under a fixed policy π:

  $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

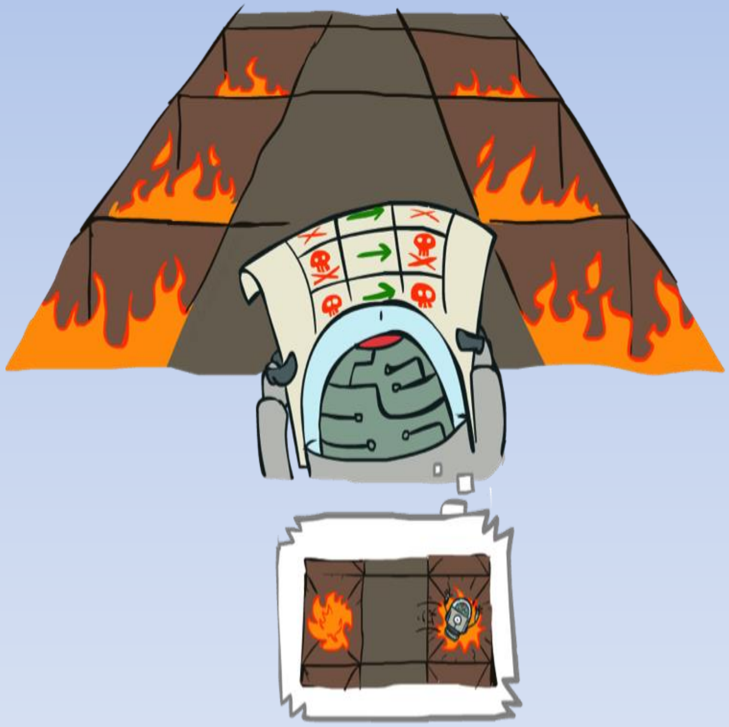
- Recursive relation (one-step look-ahead / Bellman equation):

$$V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

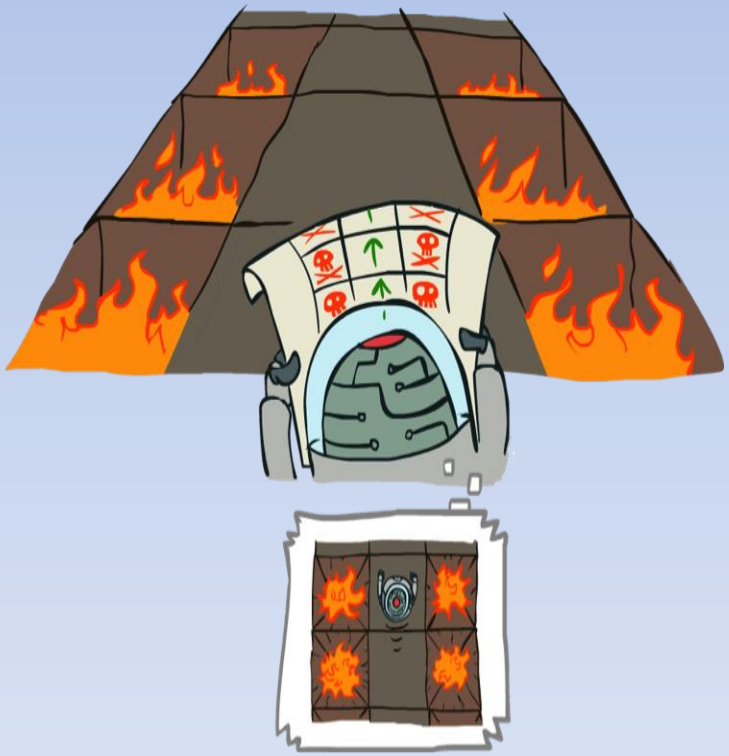


Example: Policy Evaluation

Always Go Right

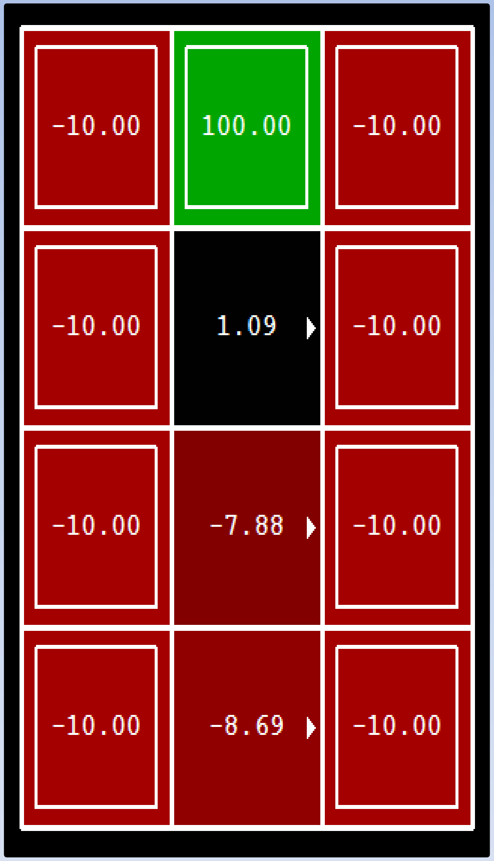


Always Go Forward

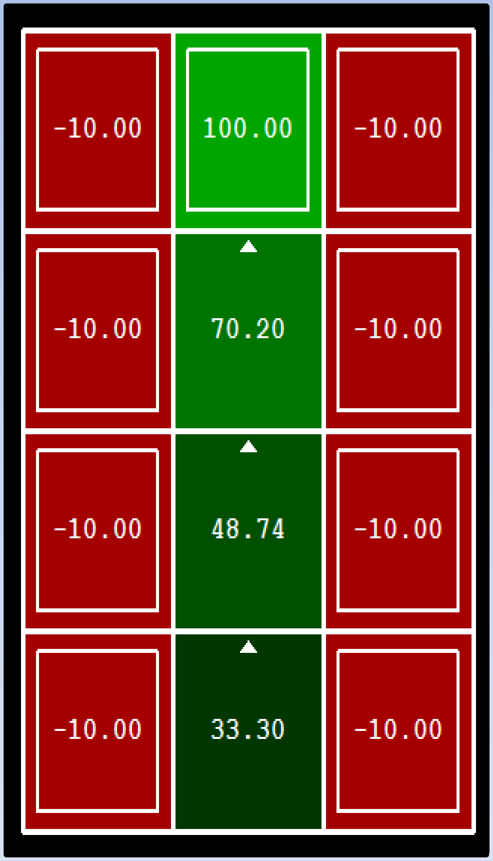


Example: Policy Evaluation

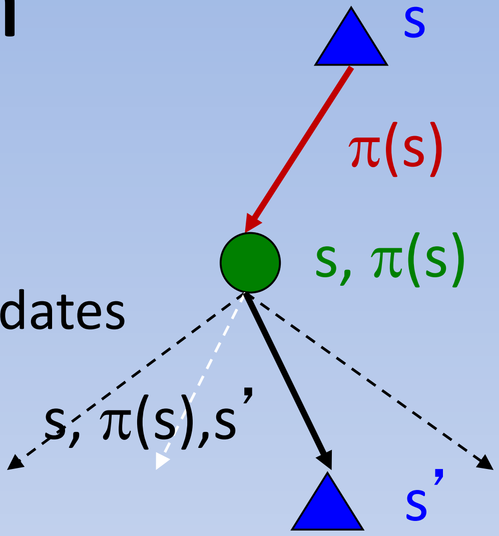
Always Go Right



Always Go Forward



Policy Evaluation



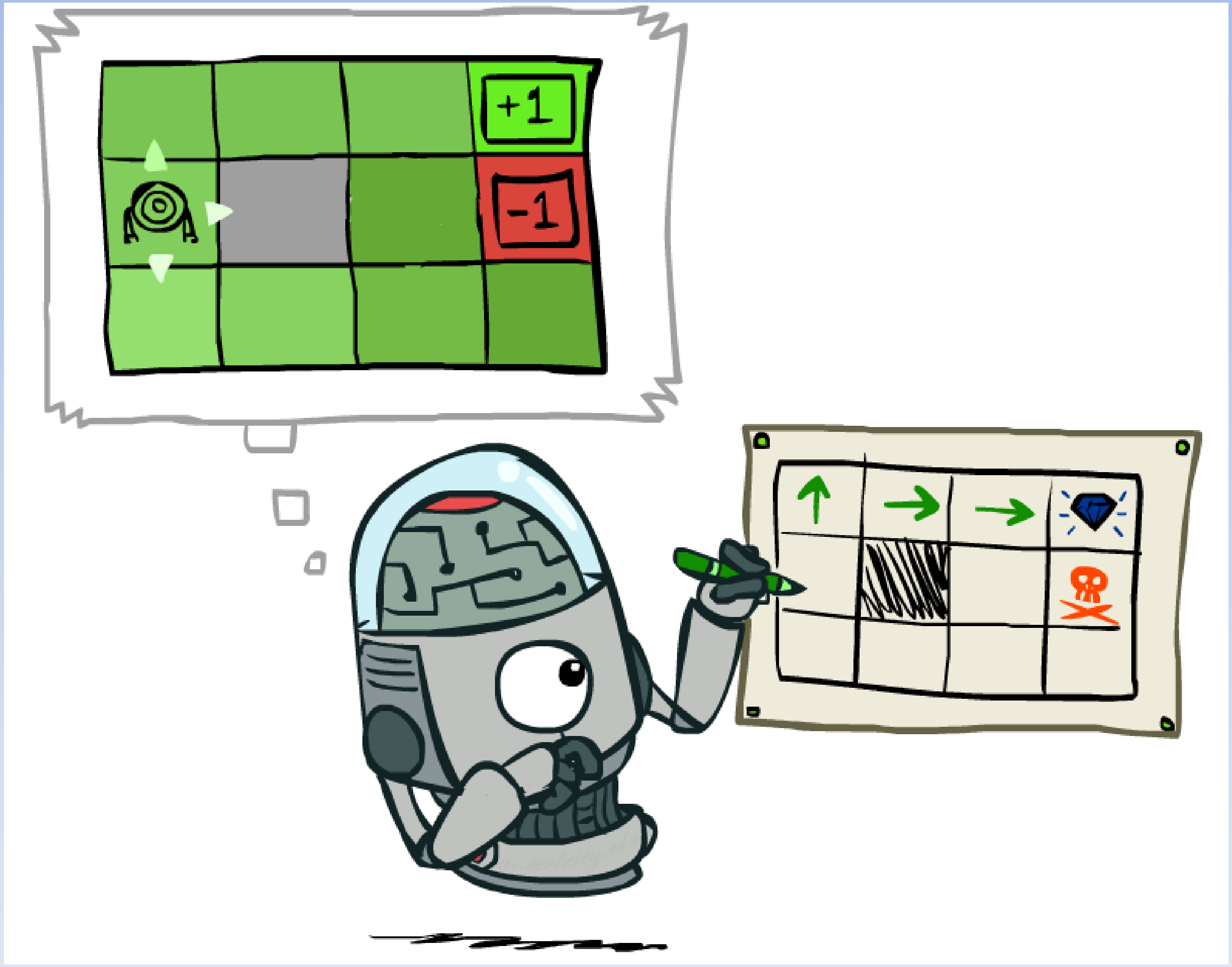
- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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

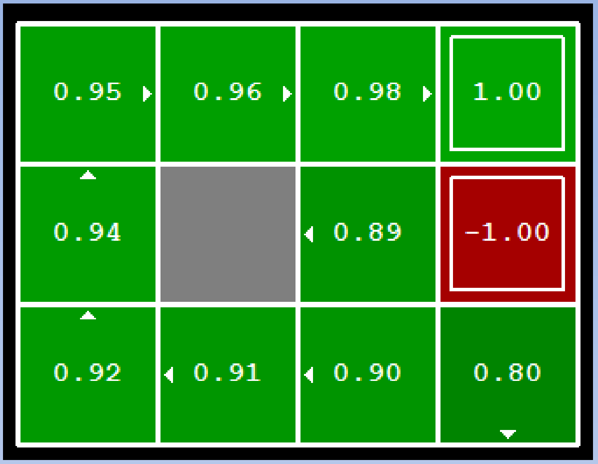
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$

- Efficiency: O(S^2) per iteration
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

Policy Extraction



Computing Actions from Values

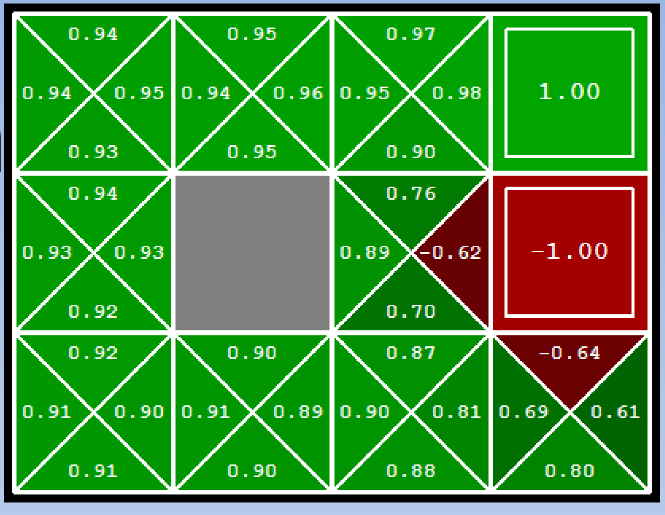


- Let's imagine we have the optimal values V*(s)
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values



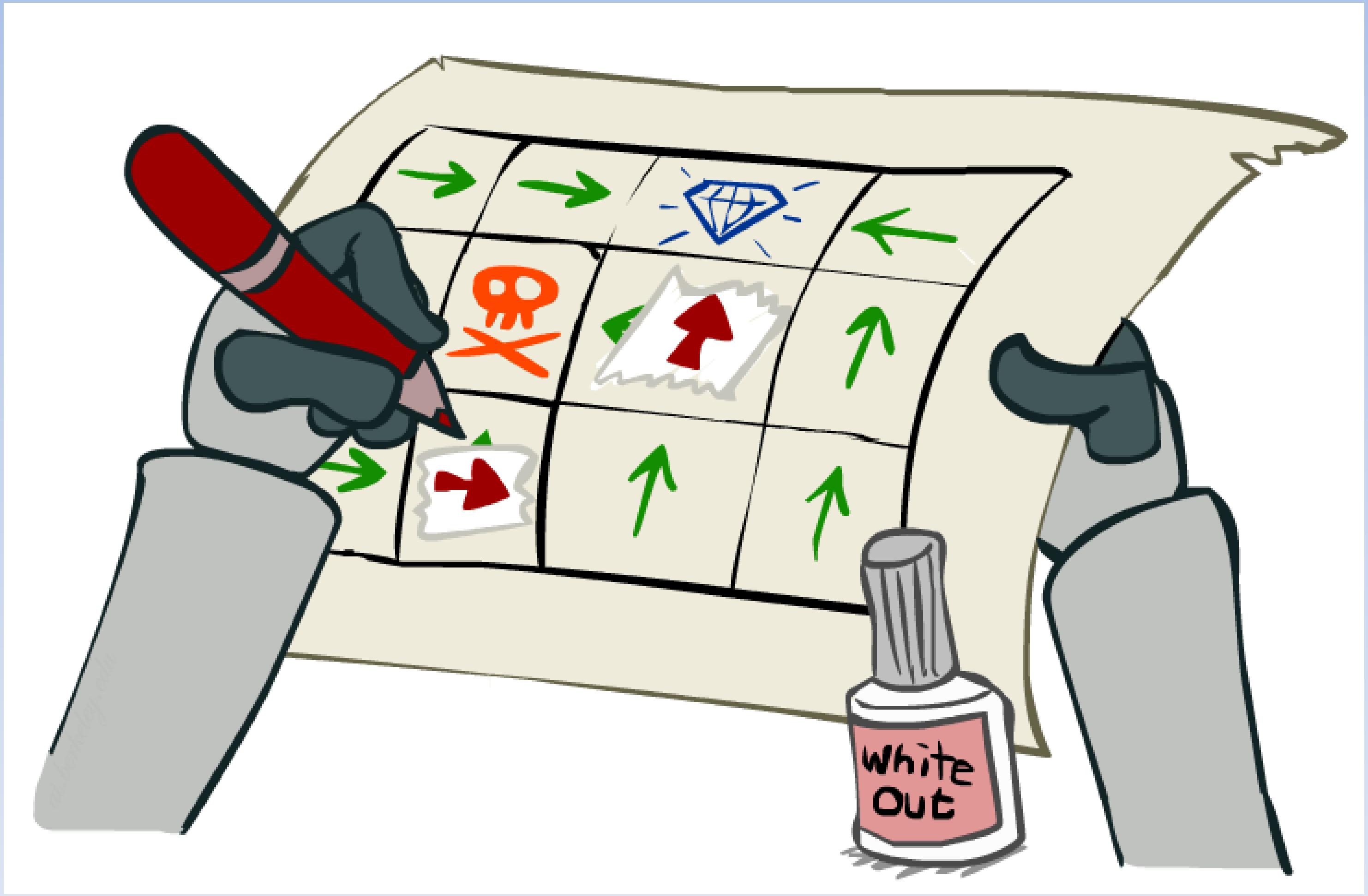
- Let's imagine we have the optimal q-values:

- How should we act?
  - Completely trivial to decide!

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

- Important lesson: actions are easier to select from q-values than values!

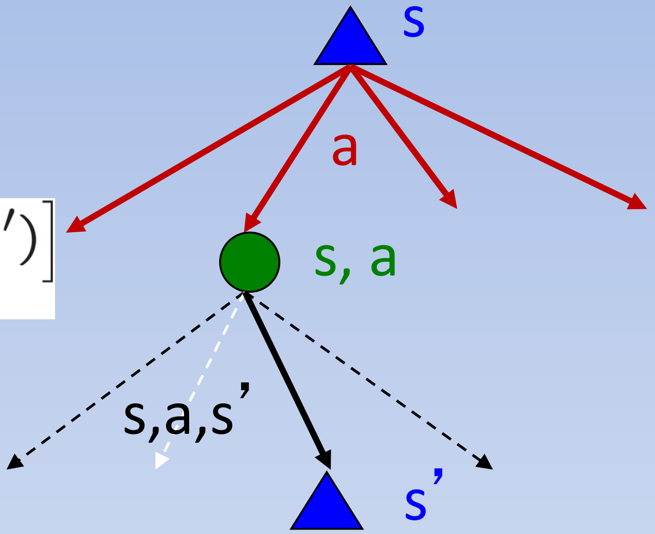
Policy Iteration



Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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



- Problem 1: It's slow – O(S^2A) per iteration

- Problem 2: The "max" at each state rarely changes

- Problem 3: The policy often converges long before the values

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Iteration

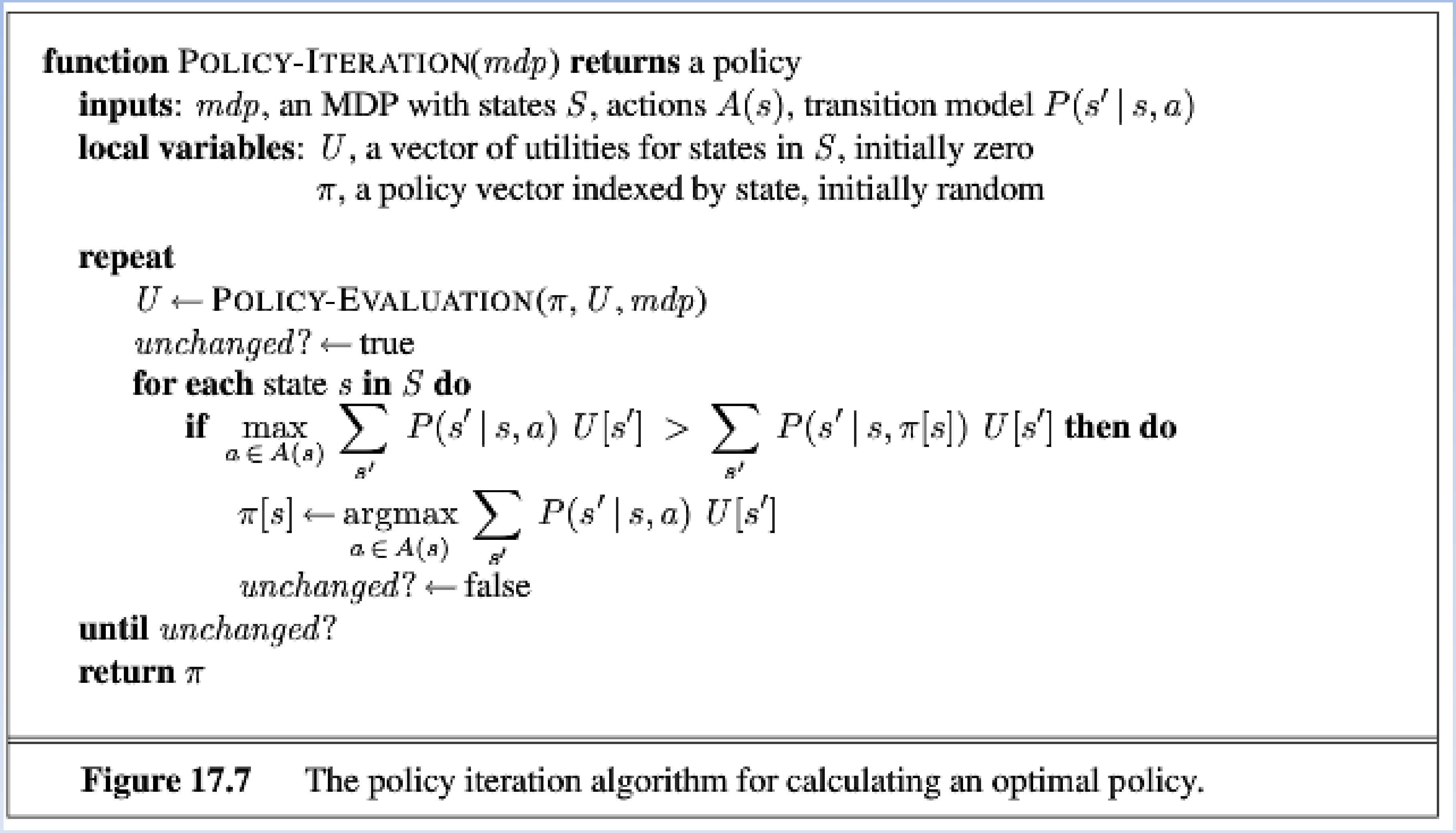
- Evaluation: For fixed current policy π, find values with policy evaluation:
    - Iterate until values converge:

$$V^{\pi_i}_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V^{\pi_i}_k(s')\right]$$

- Improvement: For fixed values, get a better policy using policy extraction
    - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Policy Iteration w/ Textbook



```
function POLICY-ITERATION(mdp) returns a policy
  inputs: mdp, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: $U$, a vector of utilities for states in $S$, initially zero
                   $\pi$, a policy vector indexed by state, initially random

  repeat
     $U \leftarrow$ POLICY-EVALUATION($\pi, U, mdp$)
     unchanged? $\leftarrow$ true
     for each state $s$ in $S$ do
        if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ then do
           $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
           unchanged? $\leftarrow$ false
  until unchanged?
  return $\pi$
```

Figure 17.7 The policy iteration algorithm for calculating an optimal policy.

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)
- Both are dynamic programs for solving MDPs

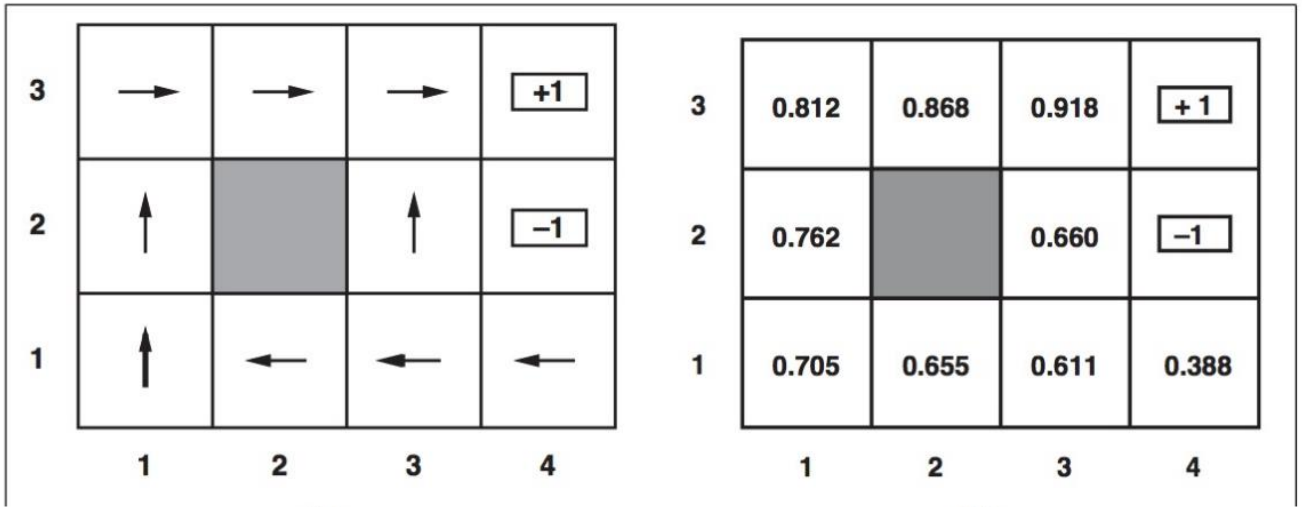
Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)

- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

21 REINFORCEMENT LEARNING

In which we examine how an agent can learn from success and failure, from reward and punishment.



$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (2,1)_{\textbf{-.04}} \rightsquigarrow (3,1)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (4,2)_{\textbf{-1}} \, .$$

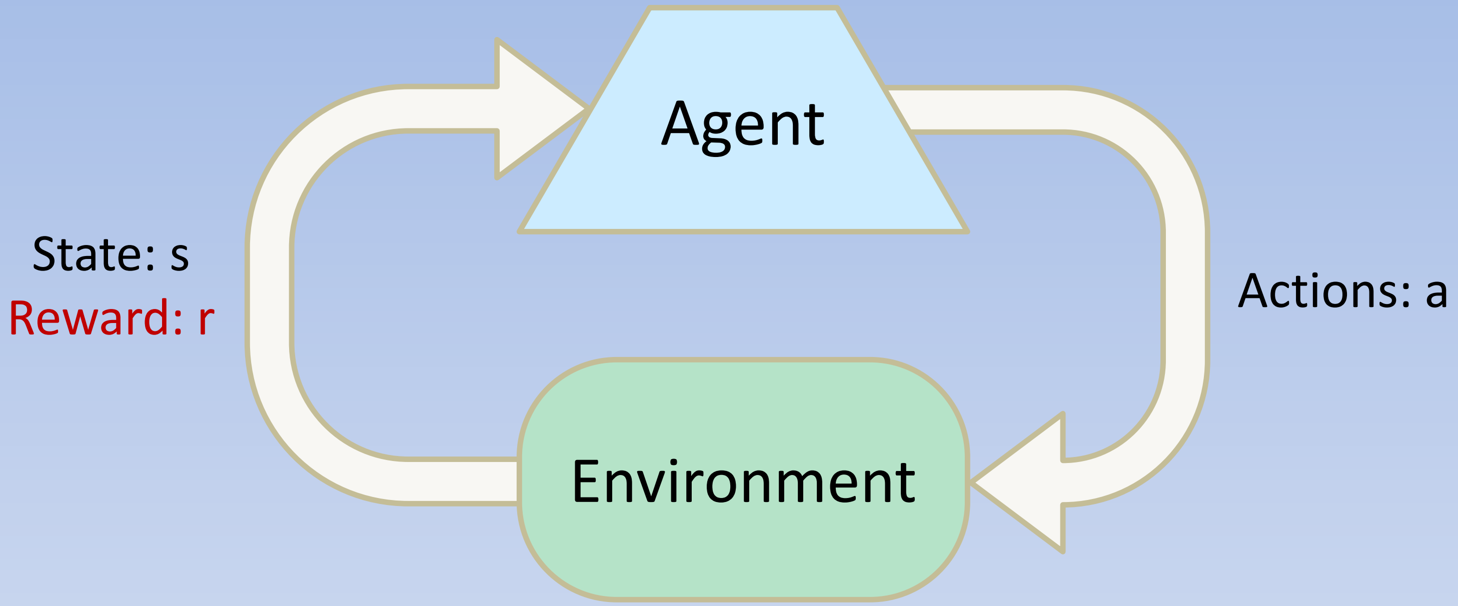
Rewards/Reinforcements

- Now we learn where Reinforcements arise.
- Agent must know when something good or something bad has happened!
  - Rewards
    - + 20
    - - 200
  - Reinforcements
    - Good Job!
    - More work is needed.

Rewards/Reinforcements

- Reinforcements come in the end with games like chess.
- Other environments get feedback more frequently
  - Ping-Pong
  - Learning to crawl
- Reinforcements are a percept, but hardwired differently.
  - Animals perceive hunger/pain differently than vision/auditory percepts.
- Reinforcement Learning studied by psychologists for more than 60 years.

Reinforcement Learning

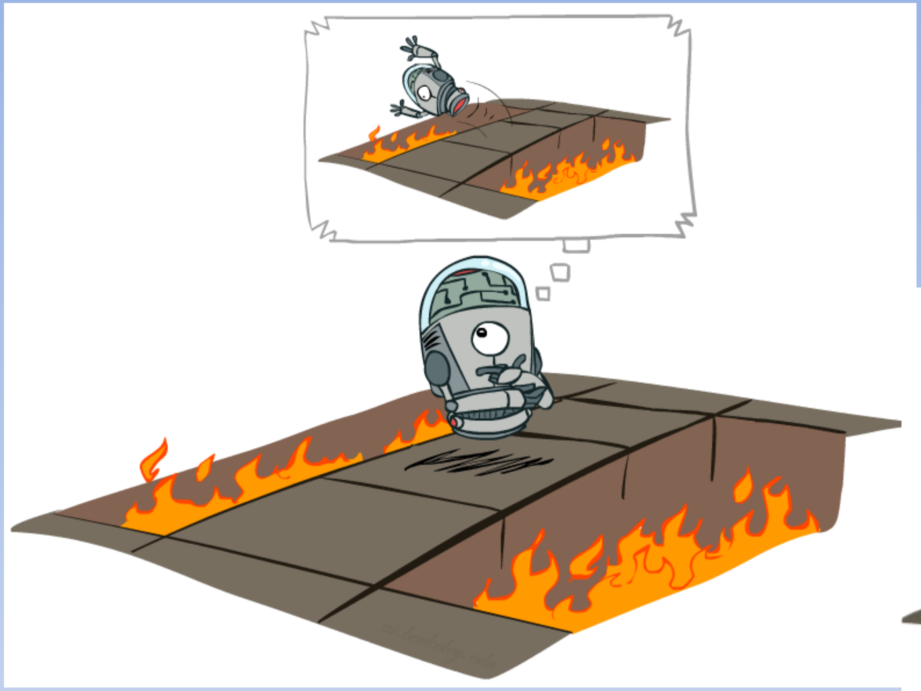


- Receive feedback in the form of rewards
- Agent’s utility is defined by the reward function
- Must (learn to) act so as to maximize expected rewards
- All learning is based on observed samples of outcomes!

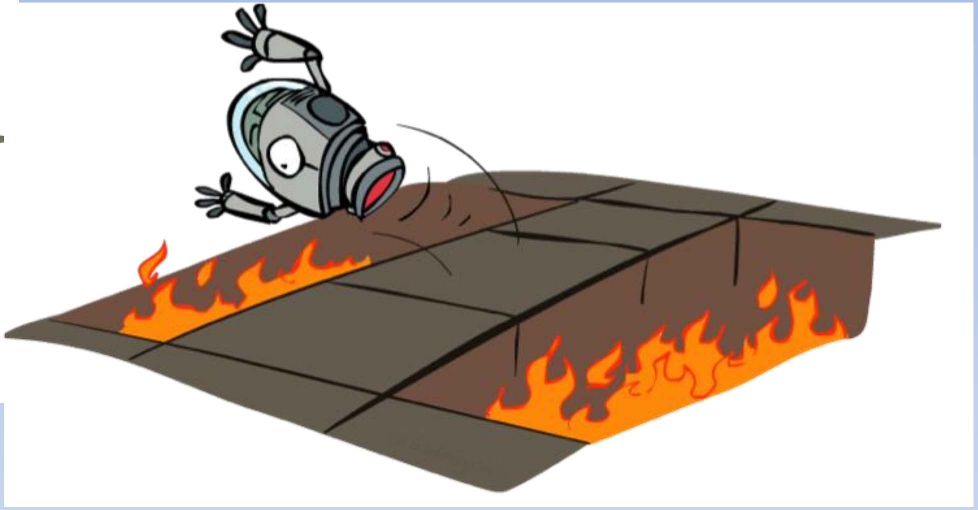
Model

- Assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s') -- P(s'|s,a)
  - A reward function R(s,a,s') – Sometimes R(s)
- Still looking for a policy $\pi(s)$

- Assume now: we don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Learning requires actually trying actions and states out



Offline Solution



Online Learning

- When MDP Known – Solution found offline!
- Without MDP (T, R), need to take actions to learn!

Reinforcement Learning

- What should we learn?
- First thought, what do we not know?
  - What are our unknowns?
- Unknowns:
  - $T(s, a, s')$ or $P(s'|s,a)$
  - $R(s, a, s')$ or $R(s)$
- Let’s Learn T and R!

Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- 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’)

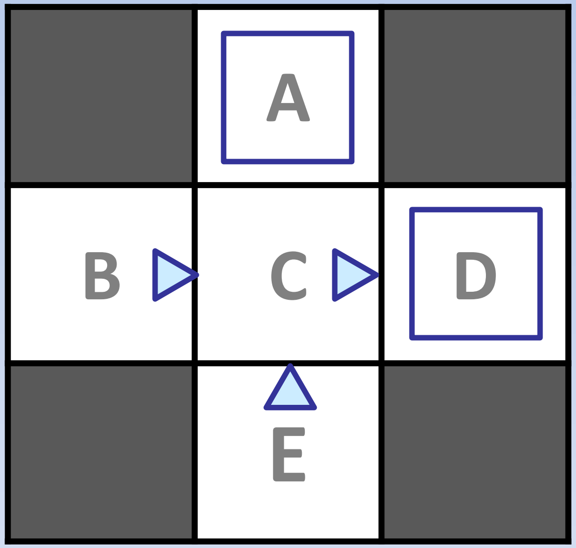
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Passive Learning Agent

- State-based Representation
- Fully observable environment
- Fixed policy π
- In state s agent always executes action $\pi(s)$

Example

Input Policy π



Assume: $\gamma = 1$

- Let's put the agent in environment and see what happens!

Agent Acting

Observed Episodes (Training)

Episode 1

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

- $\hat{R}(s, a, s')$
  - (B, east, C) = -1
  - (C, east, D) = -1
  - (D, exit, x) = +10
- $\hat{T}(s, a, s')$
  - (B, east, C) = 1.0/1.0
  - (C, east, D) = 1.0/1.0
  - (D, exit, x) = 1.0/1.0

Agent Acting

Observed Episodes (Training)

Episode 1	Episode 2
B, east, C, -1	B, east, C, -1
C, east, D, -1	C, east, D, -1
D, exit, x, +10	D, exit, x, +10

- $\hat{R}(s, a, s')$
  - (B, east, C) = -1
  - (C, east, D) = -1
  - (D, exit, x) = +10
- $\hat{T}(s, a, s')$
  - (B, east, C) = 2.0/2.0
  - (C, east, D) = 2.0/2.0
  - (D, exit, x) = 2.0/2.0

Agent Acting

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

- $\hat{R}(s, a, s')$
  - (B, east, C) = -1
  - (C, east, D) = -1
  - (D, exit, x) = +10
  - (E, north, C) = -1
- $\hat{T}(s, a, s')$
  - (B, east, C) = 2.0/2.0
  - (C, east, D) = 3.0/3.0
  - (D, exit, x) = 3.0/3.0
  - (E, north, C) = 1.0/1.0

Agent Acting

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

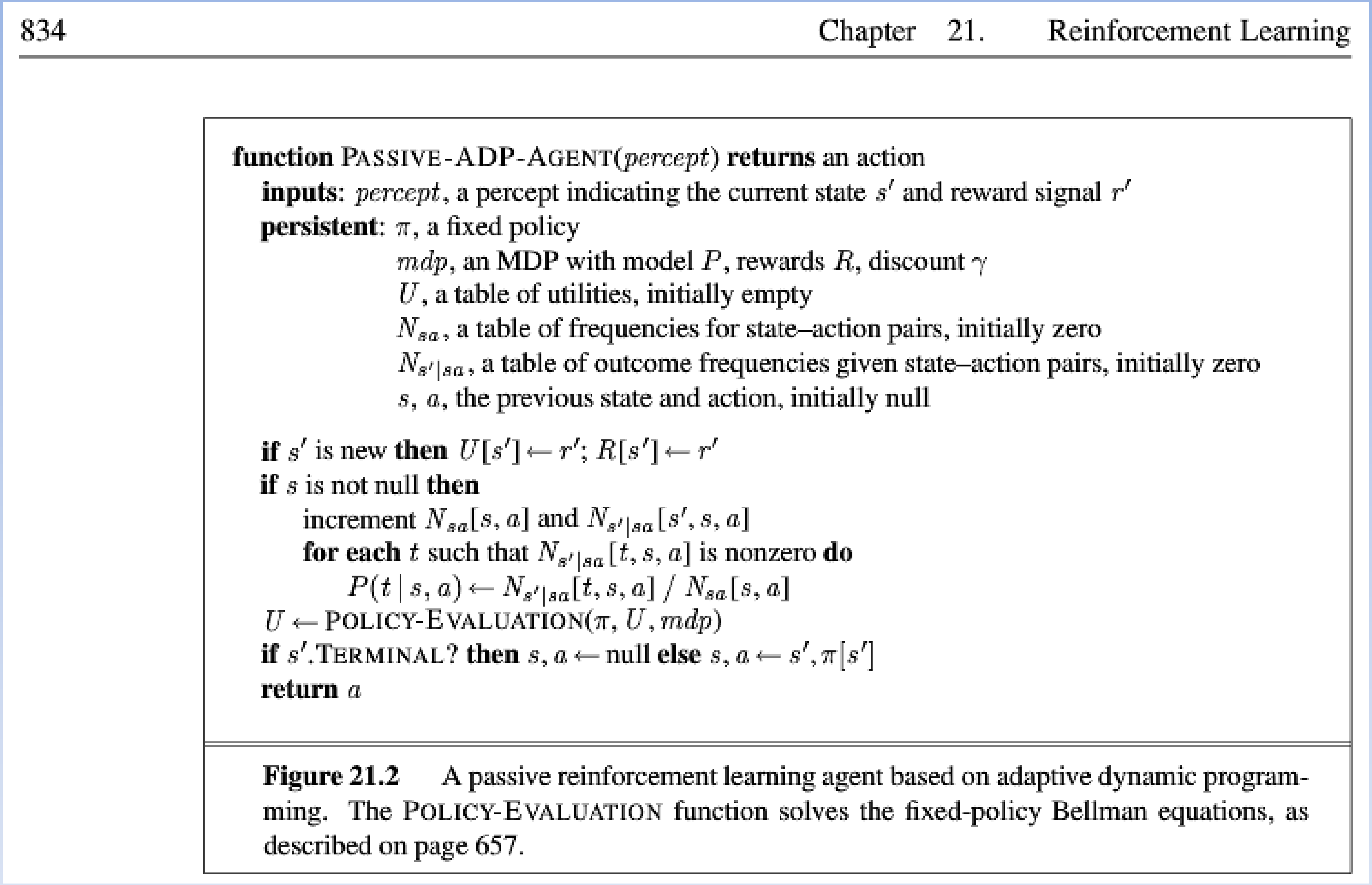
Episode 4

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

- $\hat{R}(s,a,s')$
  - (A, exit, x) = -10
  - (B, east, C) = -1
  - (C, east, D) = -1
  - **(C, east, A) = -1.0**
  - (D, exit, x) = +10
  - (E, north, C) = -1
- $\hat{T}(s,a,s')$
  - (A, exit, x) = 1.0/1.0
  - (B, east, C) = 2.0/2.0
  - **(C, east, D) = 3.0/4.0**
  - **(C, east, A) = 1.0/4.0**
  - (D, exit, x) = 3.0/3.0
  - (E, north, C) = 2.0/2.0

Now Know T, R

- $\hat{R}(s, a, s')$
  - (A, exit, x) = -10
  - (B, east, C) = -1
  - (C, east, D) = -1
  - (C, east, A) = -1.0
  - (D, exit, x) = +10
  - (E, north, C) = -1
- $\hat{T}(s, a, s')$
  - (A, exit, x) = 1.0/1.0 = 1.0
  - (B, east, C) = 2.0/2.0 = 1.0
  - (C, east, D) = 3.0/4.0 = 0.75
  - (C, east, A) = 1.0/4.0 = 0.25
  - (D, exit, x) = 3.0/3.0 = 1.0
  - (E, north, C) = 2.0/2.0 = 1.0
- We can use Policy Evaluation to Calculate utility of states.



834 Chapter 21. Reinforcement Learning

```
function PASSIVE-ADP-AGENT($percept$) returns an action
   inputs: $percept$, a percept indicating the current state $s'$ and reward signal $r'$
   persistent: $\pi$, a fixed policy
               $mdp$, an MDP with model $P$, rewards $R$, discount $\gamma$
               $U$, a table of utilities, initially empty
               $N_{sa}$, a table of frequencies for state–action pairs, initially zero
               $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
               $s$, $a$, the previous state and action, initially null

   if $s'$ is new then $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
   if $s$ is not null then
      increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
      for each $t$ such that $N_{s'|sa}[t, s, a]$ is nonzero do
         $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \,/\, N_{sa}[s, a]$
   $U \leftarrow$ POLICY-EVALUATION($\pi$, $U$, $mdp$)
   if $s'$.TERMINAL? then $s, a \leftarrow$ null else $s, a \leftarrow s', \pi[s']$
   return $a$
```

Figure 21.2 A passive reinforcement learning agent based on adaptive dynamic programming. The POLICY-EVALUATION function solves the fixed-policy Bellman equations, as described on page 657.

(1, 1) Actions: down, right	(1, 2) Action: Exit
(2, 1) Actions: down, right	(2, 2) Action: Exit
(3, 1) Action: Exit	

This time you do not know the transition model or reward function. You observe the following episodes under policy π:

Episode 1: [(1,1), down, 0, (2,1)), ((2,1), down, 0, (3,1)), ((3,1), Exit, 10, x))]

Episode 2: [(1,1), down, 0, (1,2)), ((1,2), Exit, -10, x)]

Episode 3: [(1,1), down, 0, (2,1)), ((2,1), down, 0, (2,2)), ((2,2), Exit, -10, x))]

Episode 4: [(1,1), down, 0, (1,2)), ((1,2), Exit, -10, x)]

Transitions are defined as (State, Action, Reward, New State).

Using our model based learning approach, what is P((1,2) | (1,1), Down)?

Can we do better?

- Effort expended learning model, while model is not our goal.
- Frequently better to learn values closest to what‘s needed.
- Still, what then should we learn??

Example: Expected Age

Goal: Compute expected age of cs166 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"

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

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

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

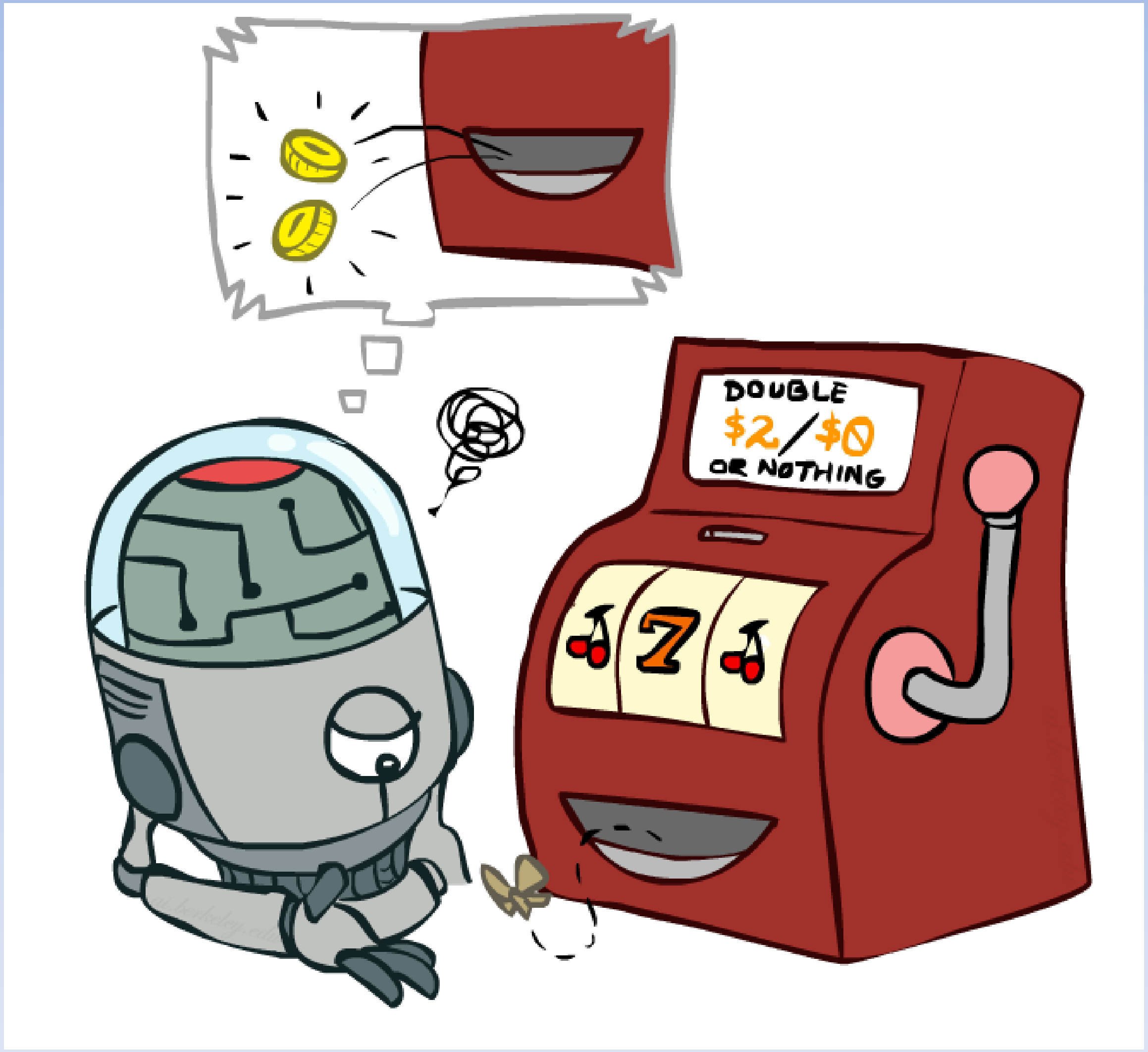
Unknown P(A): "Model Free"

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

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

Model-Free Learning

- Learn the utility of states directly!
- Why worry about MDP!
    - Sounds good.
    - Don't worry, we'll get back to MDP.



Passive Reinforcement Learning

- Passive Reinforcement Learning:
  - State-based Representation
  - Fully observable environment
  - Fixed policy π
    - In state s agent always executes action $\pi(s)$
- What should we learn?
  - First thought, what do we not know?
  - What are our unknowns?
    - Unknowns:
      - T(s, a, s') or P(s'|s,a)
      - R(s, a, s') or R(s)
  - This time.. Let's learn utility of states directly.
- Passive Reinforcement Learning:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

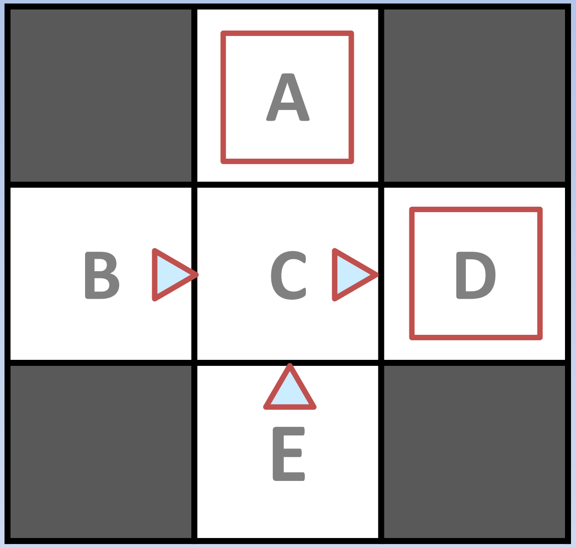
Direct-Evaluation

- Goal: Compute values for each state under π

- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

Example: Direct Evaluation

Input Policy π



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

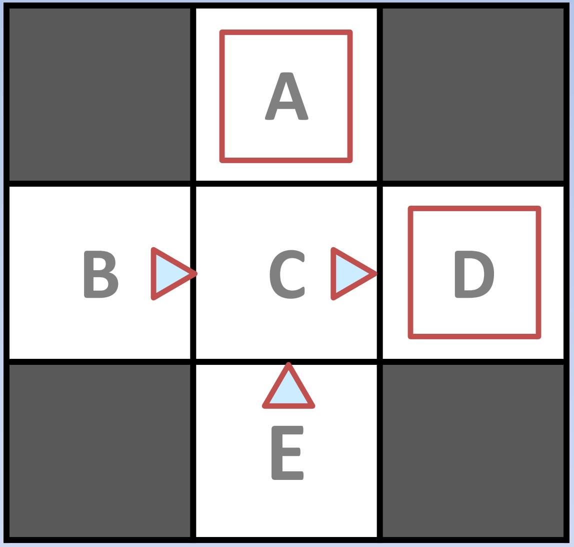
Episode 4

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

- Utility of E equals Average of
  - -1+(-1)+10, -1+(-1)-10

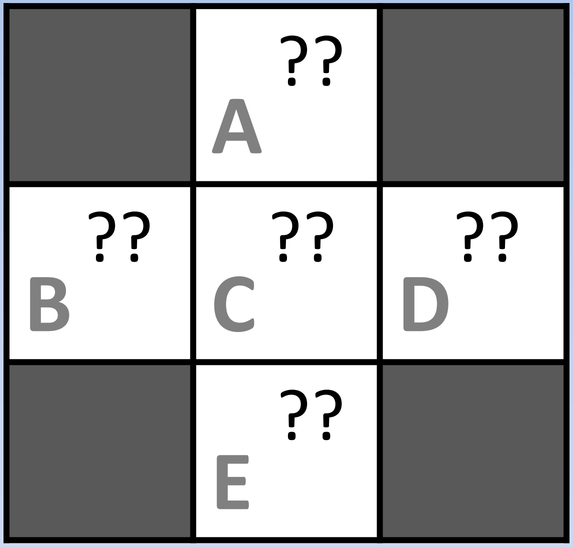
Example: Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

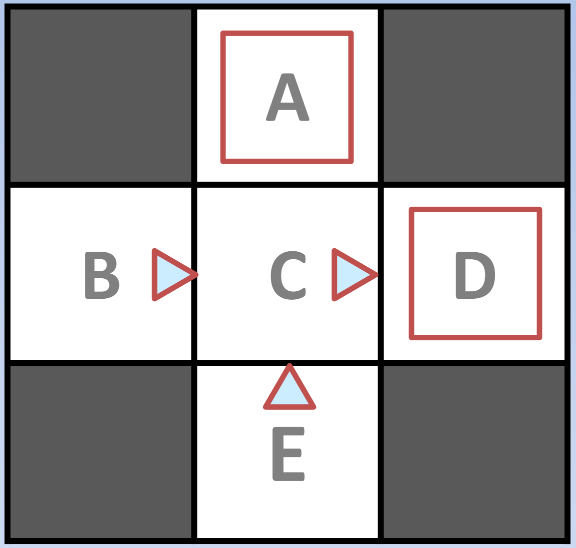
Output Values



- Episodes 1, 2, 3, 4

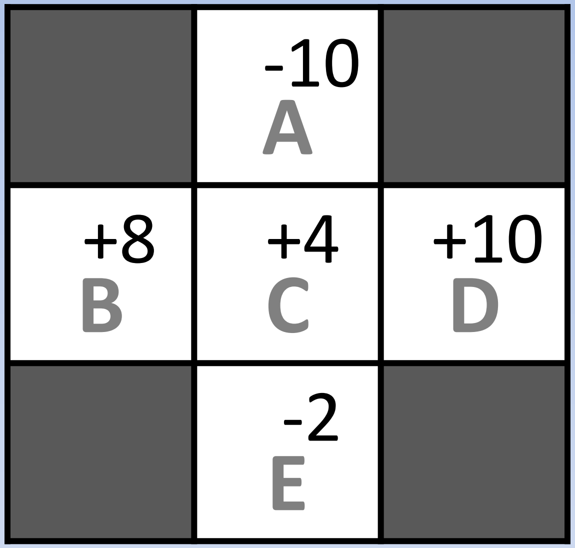
Example: Direct Evaluation

Input Policy π



Assume: γ = 1

Output Values



- Episodes 1, 2, 3, 4

Evaluating Direct Evaluation

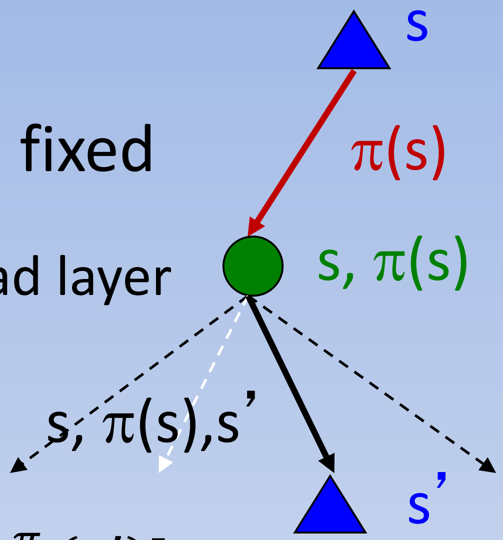
- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions

- What bad about it?
  - It wastes information about state connections
  - **Each state must be learned separately**
  - So, it takes a long time to learn
- Need more MDP!
  - Bellman Equations!!!

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V

$$V_0^{\pi} = 0$$

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



  - This approach fully exploited 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 to we take a weighted average without knowing the weights?**

Model-Free

$$V_{k+1}{}^{\pi}(s) =$$

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

- E(X) = $\sum_{x \in X} P(x)[x]$
- E(X) = $\frac{1}{n}\sum_i x_i$
  - Drop P(x) with samples
- Need samples of:
  - $R(s,\pi(s),s') + \gamma V_k{}^{\pi}(s')$
  - Available from episodes

Sample-Based Policy Evaluation?

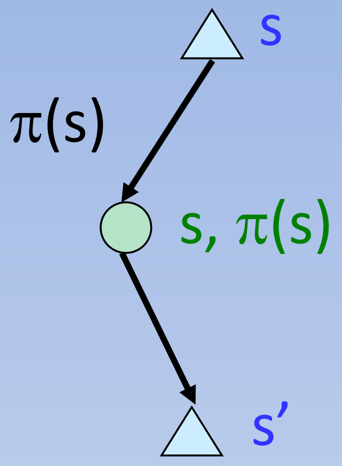
$V_{k+1}{}^{\pi}(s) =$

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

- Idea: Take samples of outcomes s' (by doing the action!) and average
  - sample$_1$=$R(s,\pi(s),s') + \gamma V_k{}^{\pi}(s')$
  - sample$_2$=$R(s,\pi(s),s') + \gamma V_k{}^{\pi}(s')$
  - *** sample$_n$= $R(s,\pi(s),s') + \gamma V_k{}^{\pi}(s')$

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

Running Average



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

- Still not convenient.
- Reformulate as running average:

sample=$R(s, \pi(s), s') + \gamma V_k{}^{\pi}(s')$

$$\boldsymbol{V_{k+1}{}^{\pi}(s) \leftarrow (1-\alpha)V_k{}^{\pi}(s) + \alpha * sample}$$

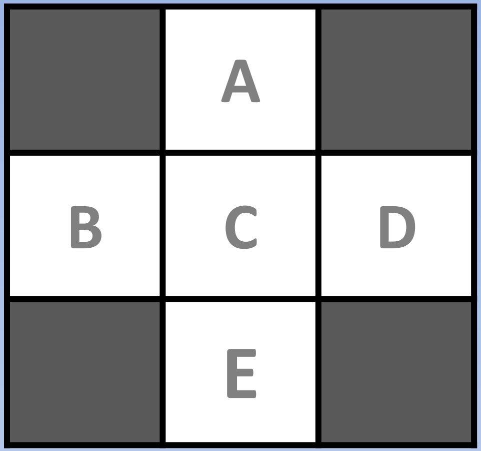
$$V_{k+1}{}^{\pi}(s) \leftarrow V_k{}^{\pi}(s) + \alpha(sample - V_k{}^{\pi}(s))$$

Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s’, r)
  - Likely outcomes s’ will contribute updates more often

- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

Example: Temporal Difference Learning

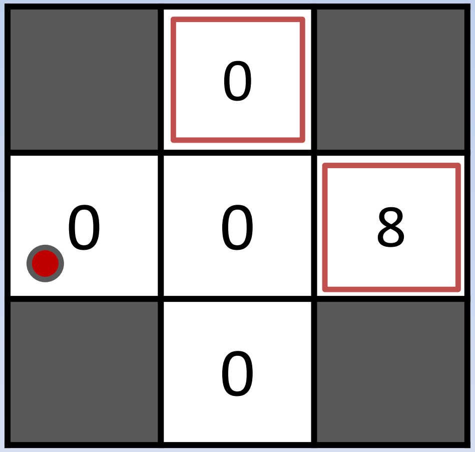


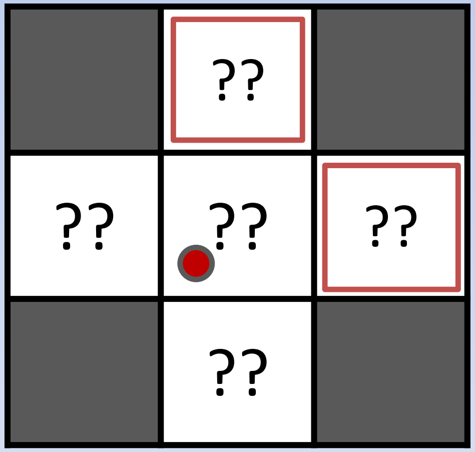
Assume: $\gamma = 1$, $\alpha = 1/2$

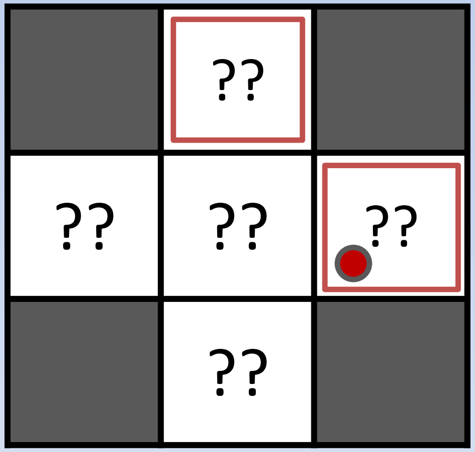
Observed Transitions

B, east, C, -2

C, east, D, -2

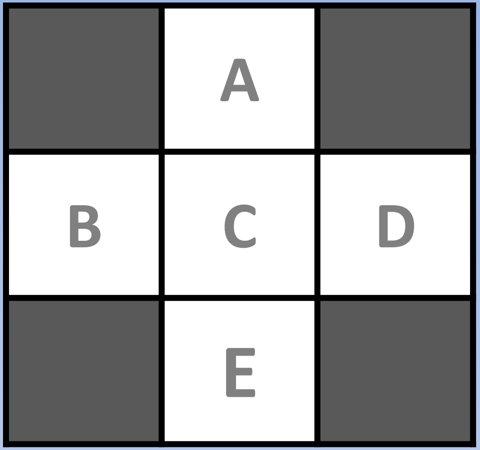






$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

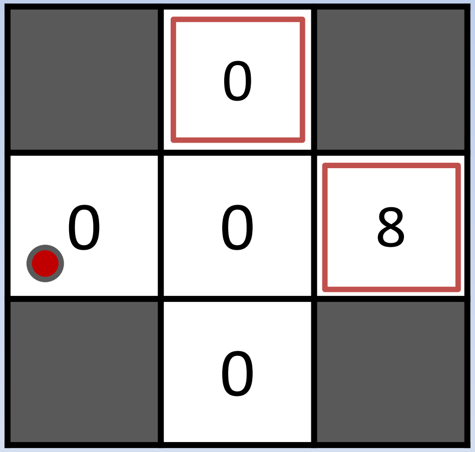
Example: Temporal Difference Learning

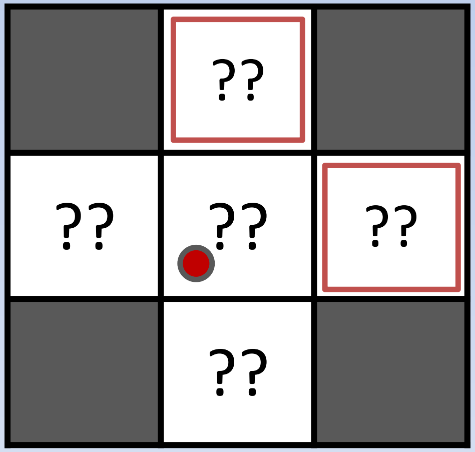


Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

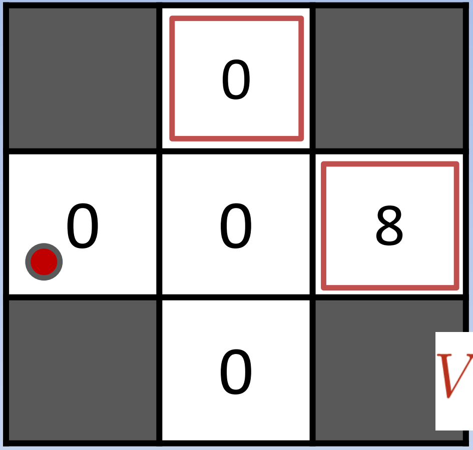
B, east, C, -2



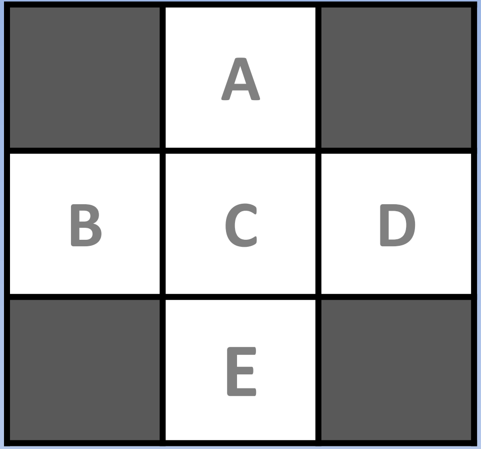


$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

Example: Temporal Difference Learning



B, east, C, -2

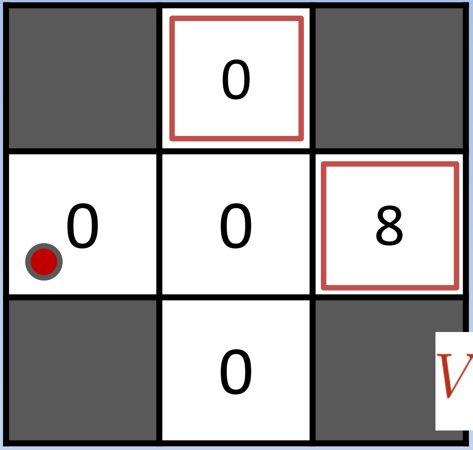


Assume: $\gamma = 1$, $\alpha = 1/2$

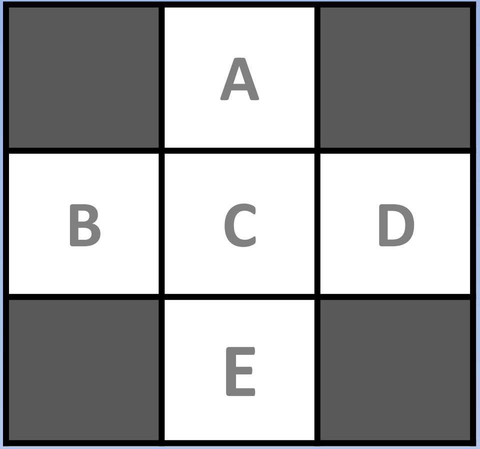
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

- State is: B
- Action is: east
- New state s' is: C
- Update $V_k^{\pi}(B)$
- Sample = $R(B, east, C) + \gamma V_k^{\pi}(C)$
  - -2 + 1*0 = -2
- Update = ½*($V_k^{\pi}(B)$=0) + ½*(Sample=-2) = -1

Example: Temporal Difference Learning



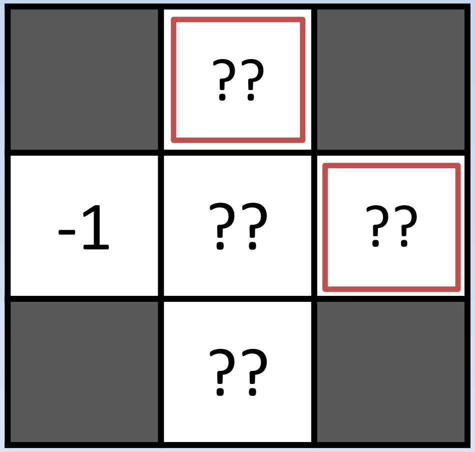
B, east, C, -2



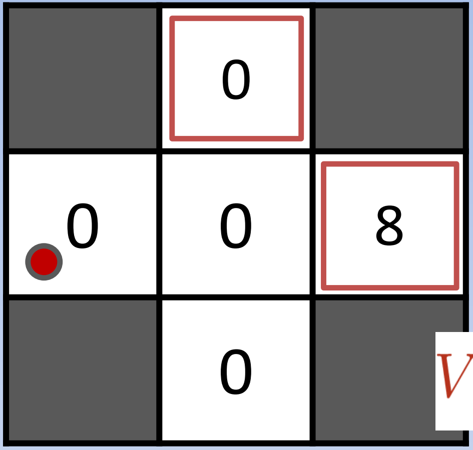
Assume: γ = 1, α = 1/2

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

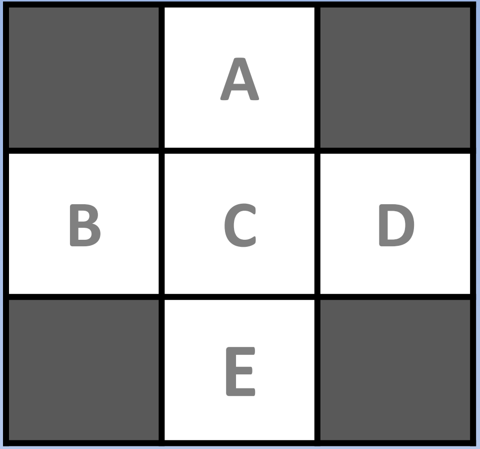
- State is: B
- Action is: east
- New state s' is: C
- Update $V_k^\pi(B)$
- Sample = $R(B, east, C) + \gamma V_k^\pi(C)$
  - -2 + 1*0 = -2
- Update = ½*($V_k^\pi(B)$=0) + ½*(Sample=-2) = -1



Example: Temporal Difference Learning



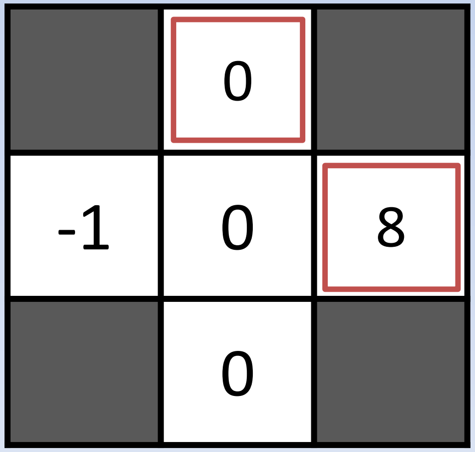
B, east, C, -2



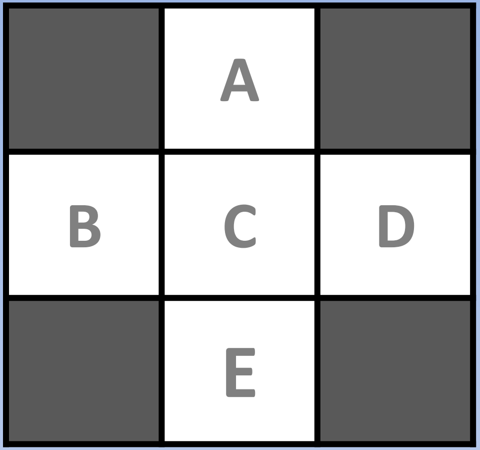
Assume: γ = 1, α = 1/2

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

- State is: B
- Action is: east
- New state s' is: C
- Update $V_k^{\pi}(B)$
- Sample = $R(B, east, C) + \gamma V_k^{\pi}(C)$
  - -2 + 1*0 = -2
- Update = ½*($V_k^{\pi}(B)$=0) + ½*(Sample=-2) = -1



Example: Temporal Difference Learning

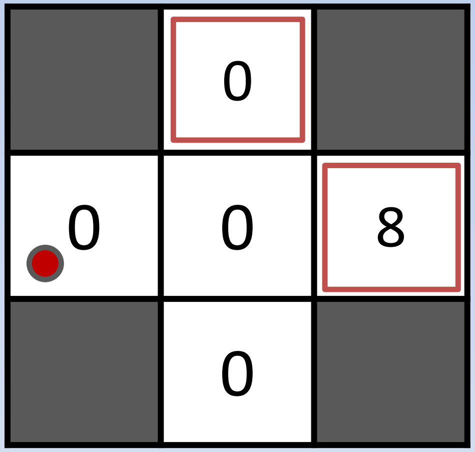


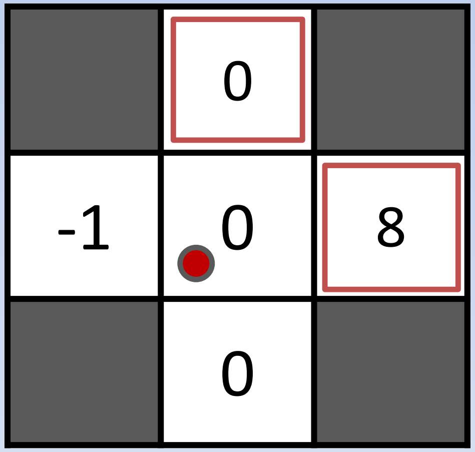
Assume: $\gamma = 1$, $\alpha = 1/2$

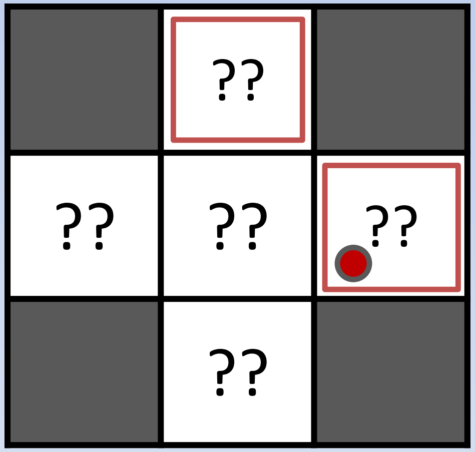
Observed Transitions

B, east, C, -2

C, east, D, -2

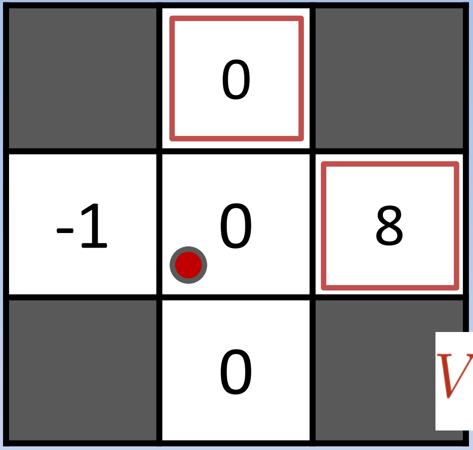




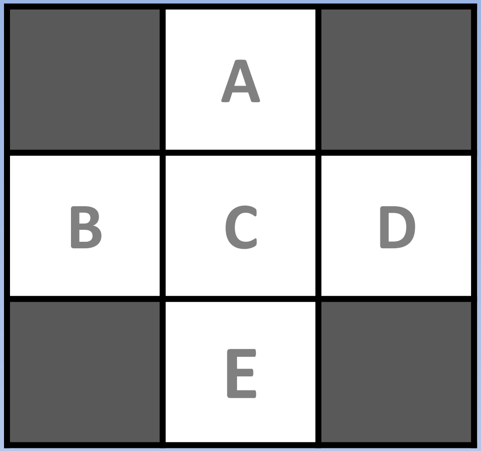


$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

Example: Temporal Difference Learning



C, east, D, -2

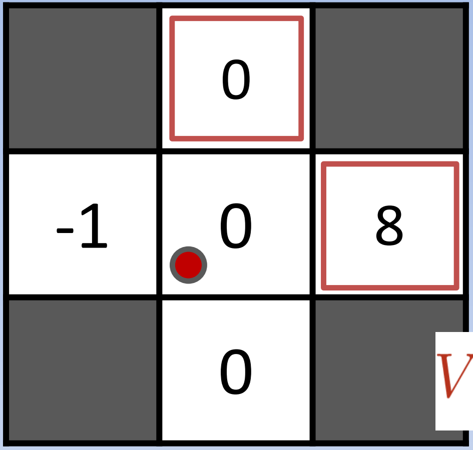


Assume: γ = 1, α = 1/2

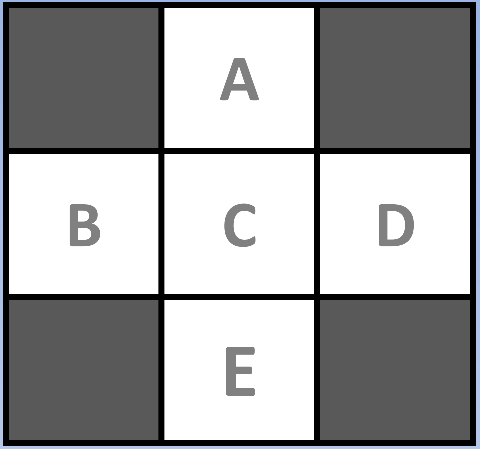
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

- State is: C
- Action is: east
- New state s' is: D
- Update $V_k^{\pi}(C)$
- Sample = $R(C, east, D) + \gamma V_k^{\pi}(C)$
  - -2 + 1*8 = 6
- Update = ½*($V_k^{\pi}(C)$=0) + ½*(Sample=6) = 3

Example: Temporal Difference Learning



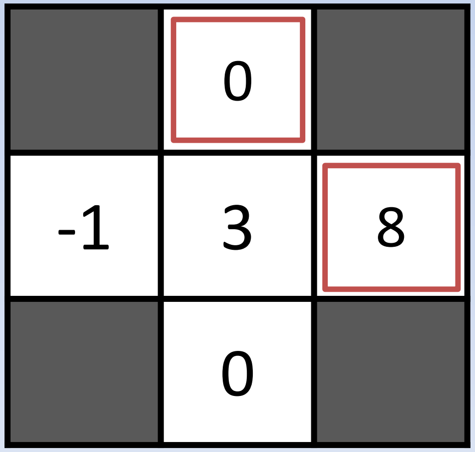
C, east, D, -2



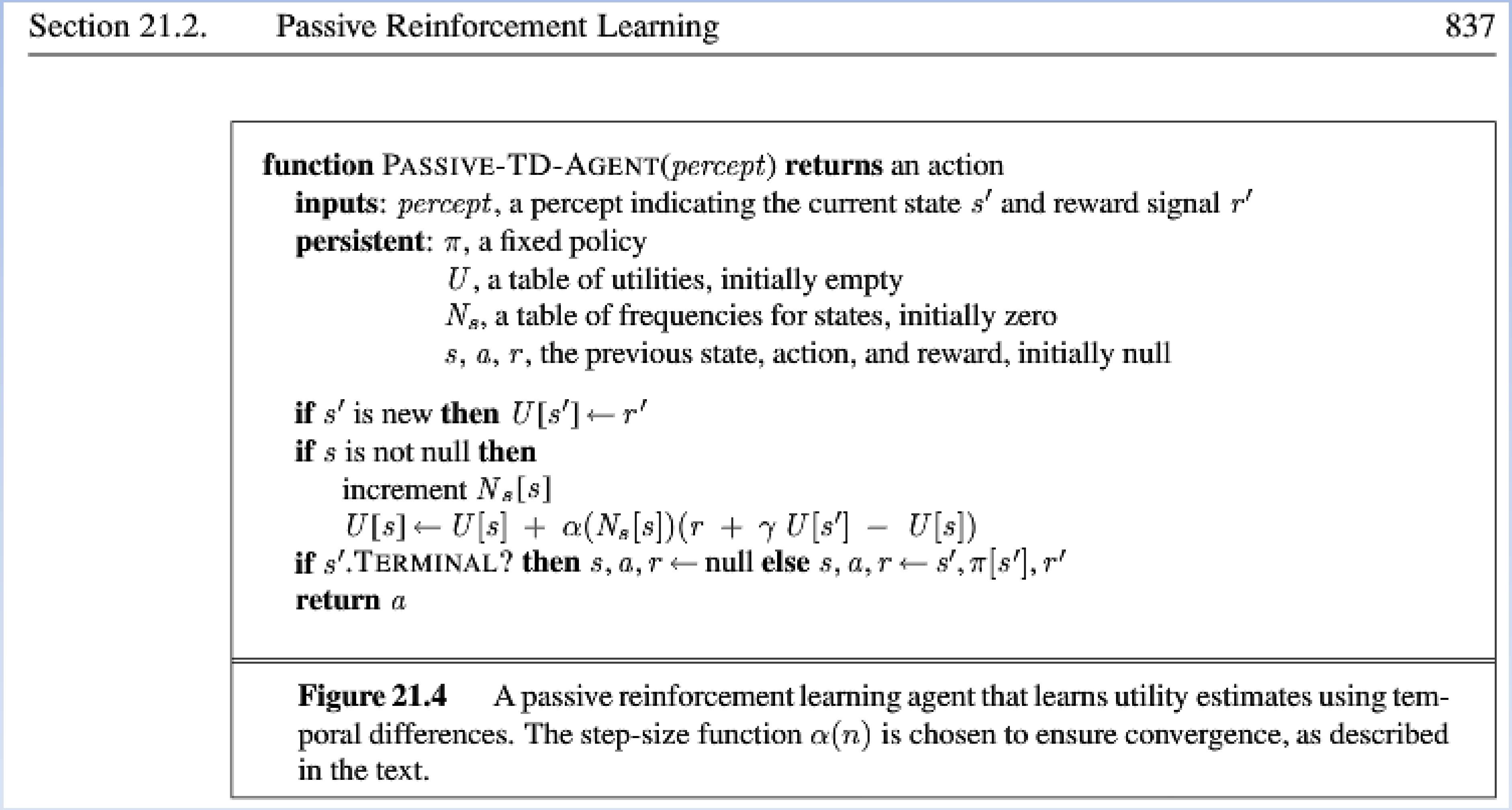
Assume: $\gamma = 1$, $\alpha = 1/2$

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

- State is: C
- Action is: east
- New state s' is: D
- Update $V_k^{\pi}(C)$
- Sample = $R(C, east, D) + \gamma V_k^{\pi}(C)$
  - -2 + 1*8 = 6
- Update = ½*($V_k^{\pi}(C)$=0) + ½*(Sample=6) = 3



Temporal Difference Learning



Section 21.2. Passive Reinforcement Learning 837

```
function PASSIVE-TD-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
  persistent: $\pi$, a fixed policy
              $U$, a table of utilities, initially empty
              $N_s$, a table of frequencies for states, initially zero
              $s$, $a$, $r$, the previous state, action, and reward, initially null

  if $s'$ is new then $U[s'] \leftarrow r'$
  if $s$ is not null then
     increment $N_s[s]$
     $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
  if $s'$.TERMINAL? then $s, a, r \leftarrow$ null else $s, a, r \leftarrow s', \pi[s'], r'$
  return $a$
```

Figure 21.4 A passive reinforcement learning agent that learns utility estimates using temporal differences. The step-size function $\alpha(n)$ is chosen to ensure convergence, as described in the text.

(1, 1) Actions: down, right	(1, 2) Action: Exit
(2, 1) Actions: down, right	(2, 2) Action: Exit
(3, 1) Action: Exit	

This time you do not know the transition model or reward function. You observe the following episodes under policy π:

Episode 1: [(1,1), down, 0, (2,1)), ((2,1), down, 0, (3,1)), ((3,1), Exit, 10, x))]

Episode 2: [(1,1), down, 0, (1,2)), ((1,2), Exit, -10, x)]

Episode 3: [(1,1), down, 0, (2,1)), ((2,1), down, 0, (2,2)), ((2,2), Exit, -10, x))]

Episode 4: [(1,1), down, 0, (1,2)), ((1,2), Exit, -10, x)]
Transitions are defined as (State, Action, Reward, New State).
Assume α=0.5, and γ=1.0, and initially all state are assumed to have a value of 0.

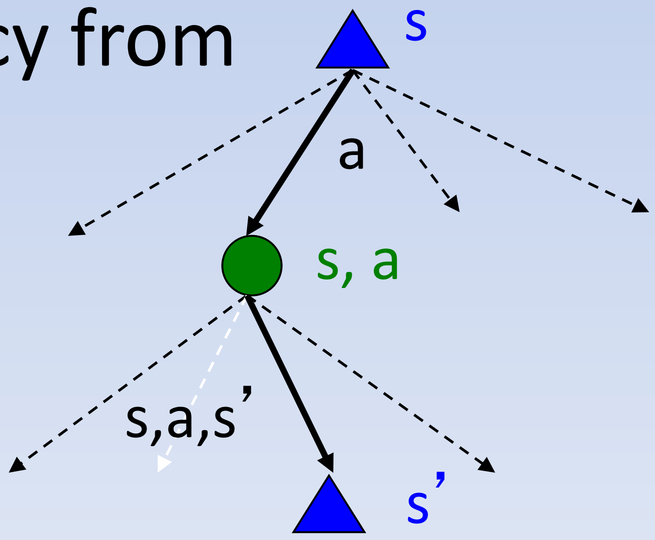
Using Temporal-Difference Learning, after seeing all of the transitions of episode 1, what would the value estimate be for state (3,1).

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
  - YaY!
- How do we generate new policy from values???

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

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



Problem

- Our error was not being focused on goal
  - Policy
- Learning needs to support goal!
- What should we learn to support goal??

Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - 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
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with V0(s) = 0, which we know is right
  - Given Vk, calculate the depth k+1 values for all states:

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

- Rather than the value of state, focus on Q-States
  - Q-values are more useful
  - Start with Q_0(s,a) = 0, which we know is right
  - Given Q_k, calculate the depth k+1 q-values for all q-states:

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

Now w/ Model-Free Learning Q-Learning

- Q-Learning: sample-based Q-value iteration

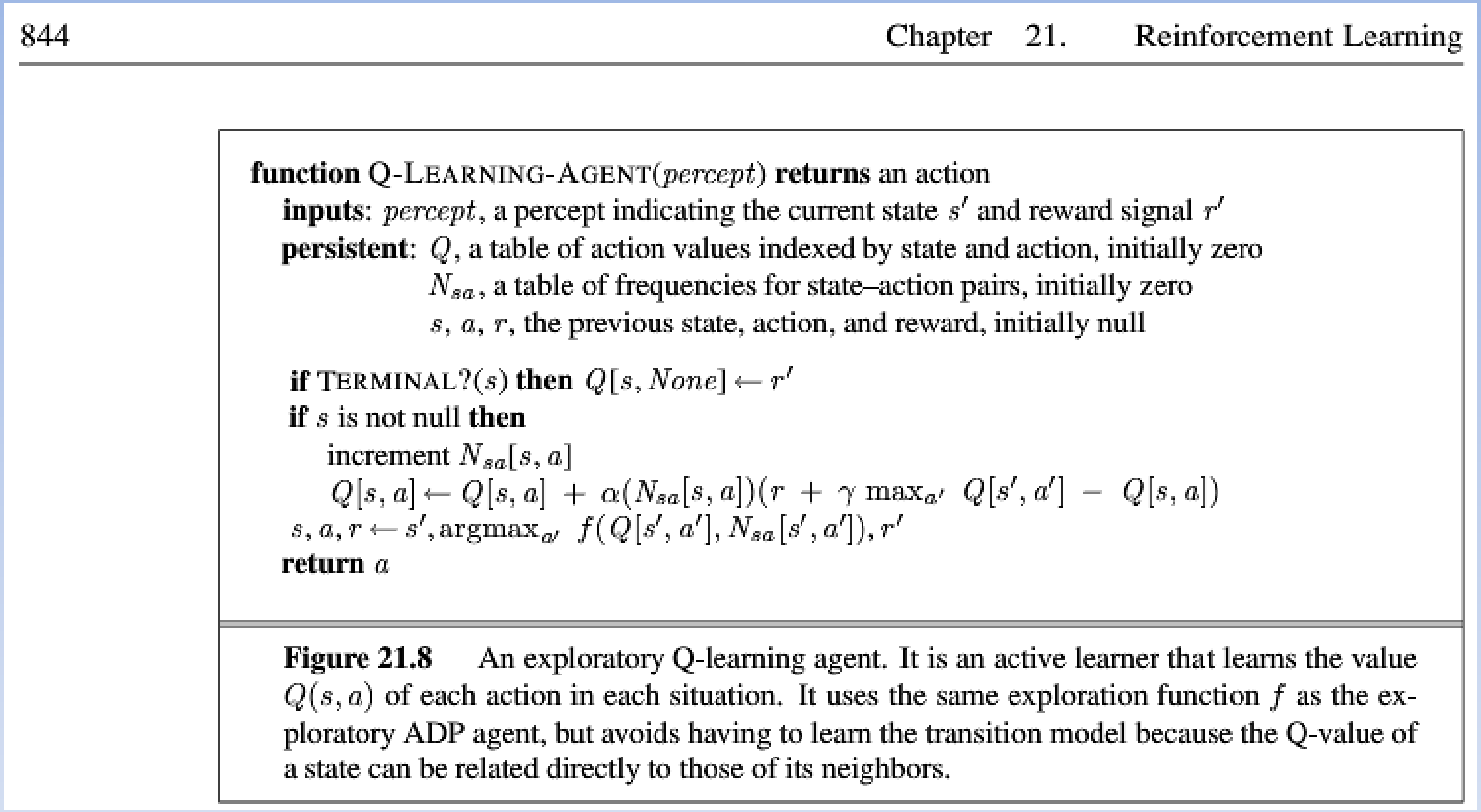
$$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_a Q_k(s,a)]$$

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

$$\text{sample} = R(s,a,s') + \gamma \max_a Q_k(s,a)$$

- Incorporate the new estimate into a running average:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha(sample)$$
$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha(sample - Q_k(s,a))$$



```
function Q-LEARNING-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
  persistent: $Q$, a table of action values indexed by state and action, initially zero
              $N_{sa}$, a table of frequencies for state–action pairs, initially zero
              $s$, $a$, $r$, the previous state, action, and reward, initially null

  if TERMINAL?($s$) then $Q[s, None] \leftarrow r'$
  if $s$ is not null then
      increment $N_{sa}[s, a]$
      $Q[s,a] \leftarrow Q[s,a] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s',a'], N_{sa}[s',a']), r'$
  return $a$
```

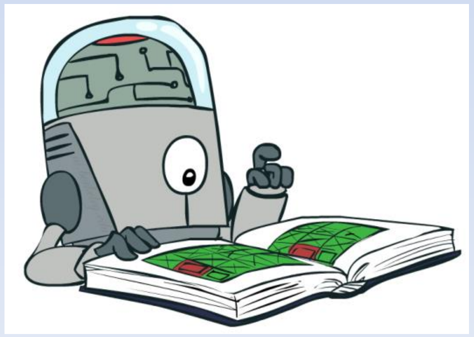
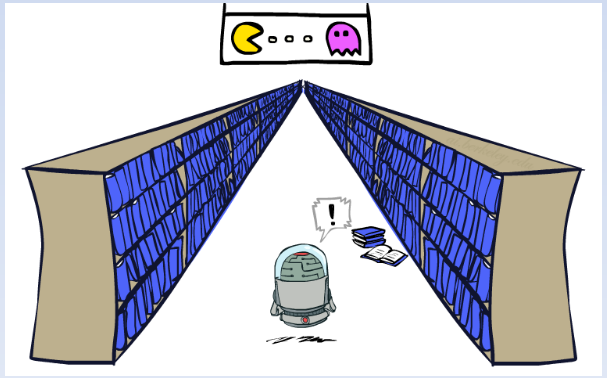
Figure 21.8 An exploratory Q-learning agent. It is an active learner that learns the value $Q(s, a)$ of each action in each situation. It uses the same exploration function f as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
    - You have to explore enough
    - You have to eventually make the learning rate small enough
    - … but not decrease it too quickly
    - Basically, in the limit, it doesn't matter how you select actions (!)

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
- Instead, 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 over and over again

Linear Value Functions

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

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

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

- 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:

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

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

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

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

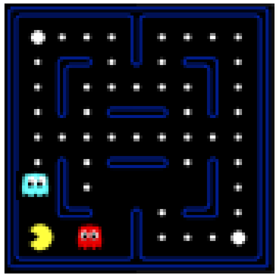
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

Example: Q-Pacman

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

s

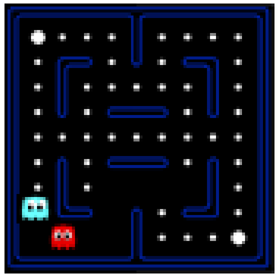


$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

$a = \text{NORTH}$
$r = -500$

s'



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

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

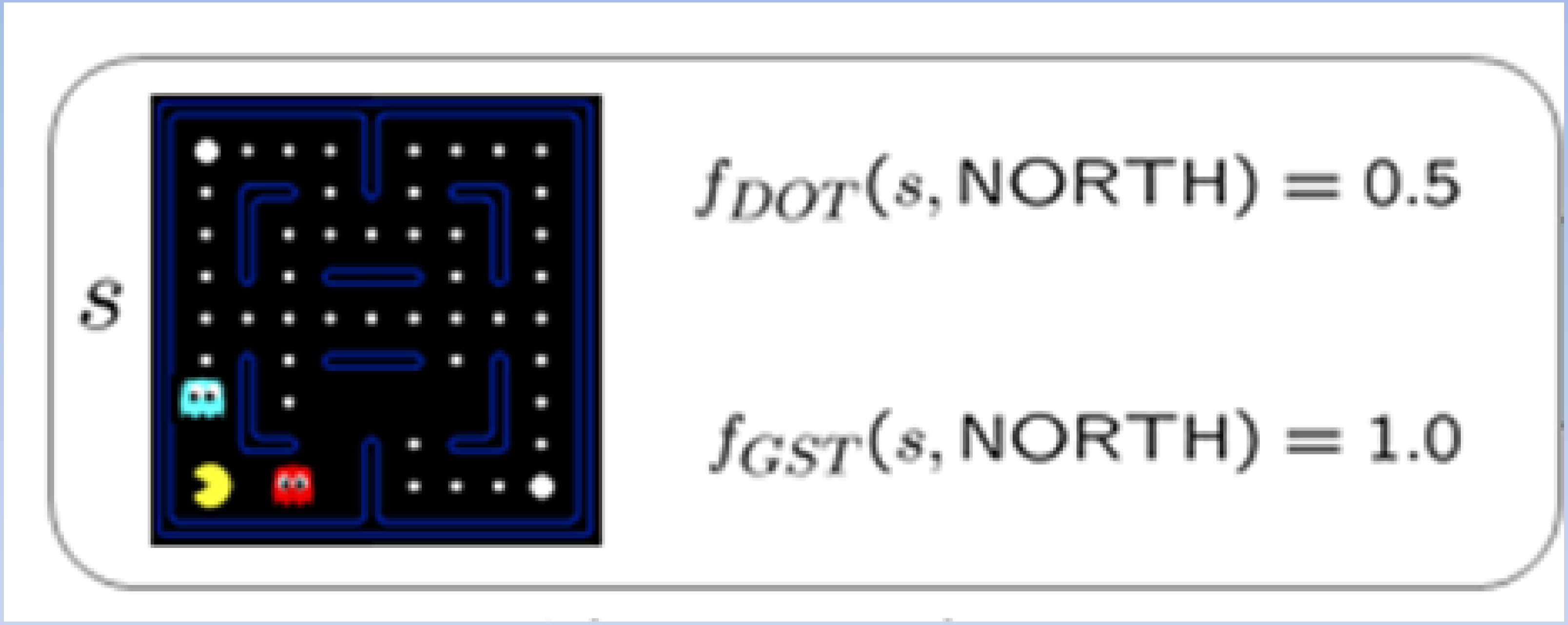
$Q(s', \cdot) = 0$

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

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

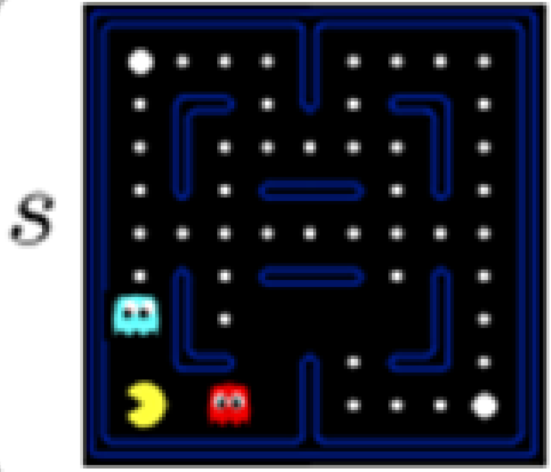
[Demo: approximate Q-learning pacman (L11D10)]

Current State: S


s
$f_{DOT}(s, \text{NORTH}) = 0.5$
$f_{GST}(s, \text{NORTH}) = 1.0$

- f_{DOT}(s, a) = 1/(distance to nearest dot after executing action a in state s)
- f_{GST}(s,a) = (Distance to nearest ghost after executing action a in state s)
- Might need different weights for each action in other situations.

Current State: S

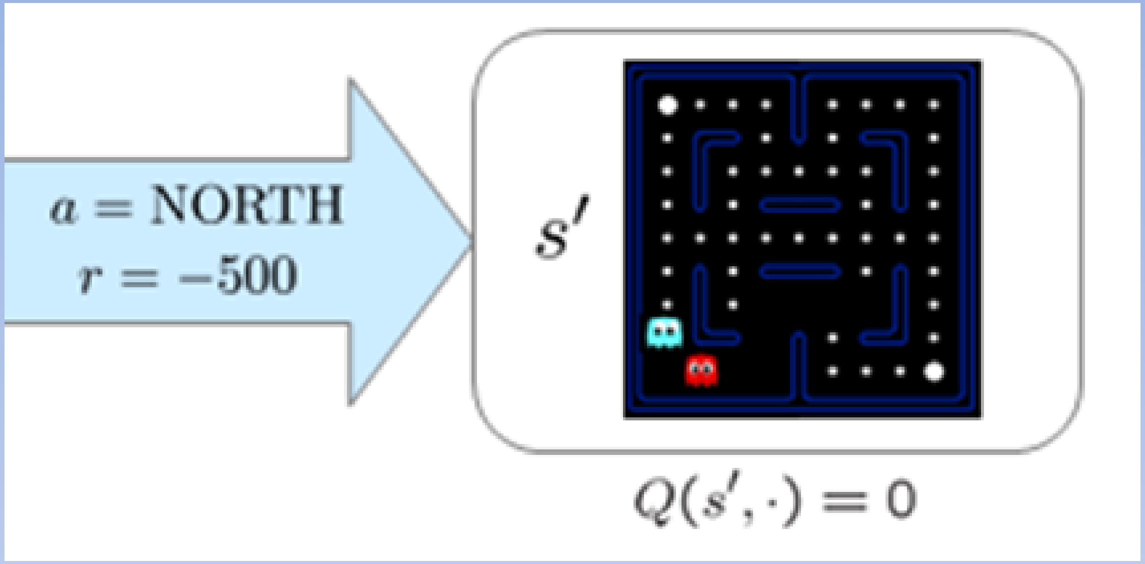


$$f_{DOT}(s, \text{NORTH}) = 0.5$$

$$f_{GST}(s, \text{NORTH}) = 1.0$$

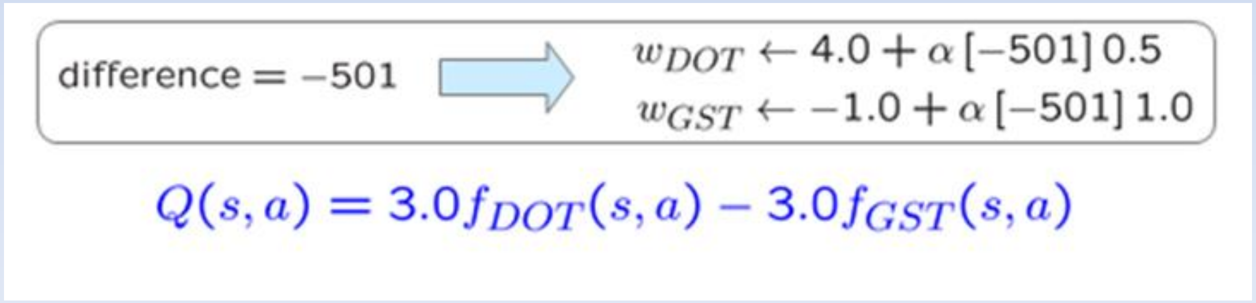
$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

- G(s, NORTH) = 4*0.5 – 1*1.0 = 1.0



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

- sample = -500 + Q(s’, .) = -500 + 0 = -500
- G(s, NORTH) = 4.0*0.5 – 1.0*1.0 = 1.0
- difference = -501



$$\text{difference} = -501 \Longrightarrow \begin{array}{l} w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5 \\ w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0 \end{array}$$

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

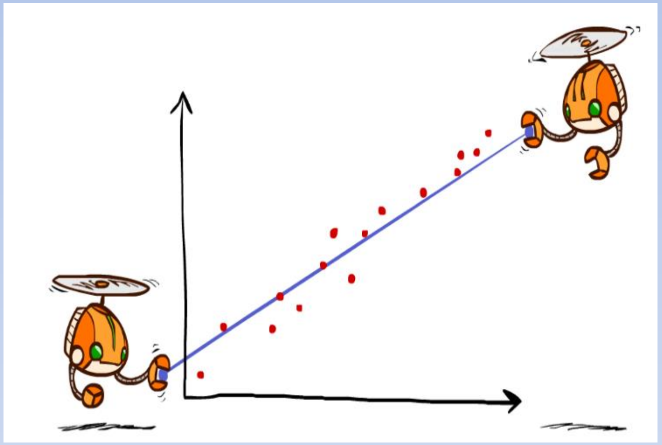
Minimizing Error*

Imagine we had only one point x, with features f(x), target value y, and weights w:

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

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

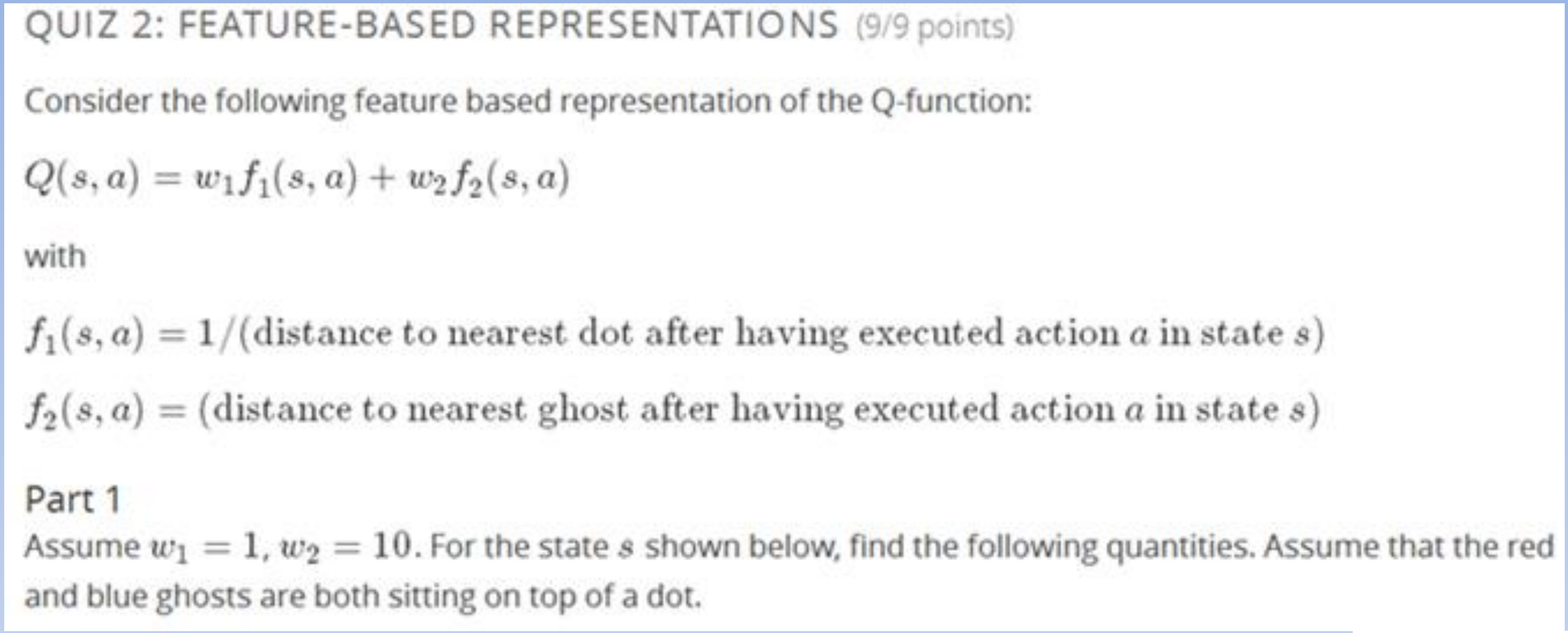
$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

“target” “prediction”



QUIZ 2: FEATURE-BASED REPRESENTATIONS (9/9 points)

Consider the following feature based representation of the Q-function:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a)$$

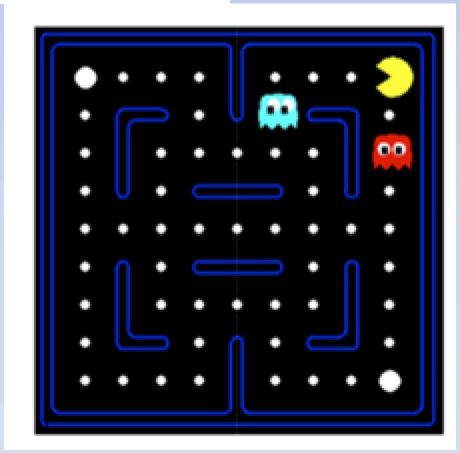
with

$$f_1(s,a) = 1/(\text{distance to nearest dot after having executed action } a \text{ in state } s)$$

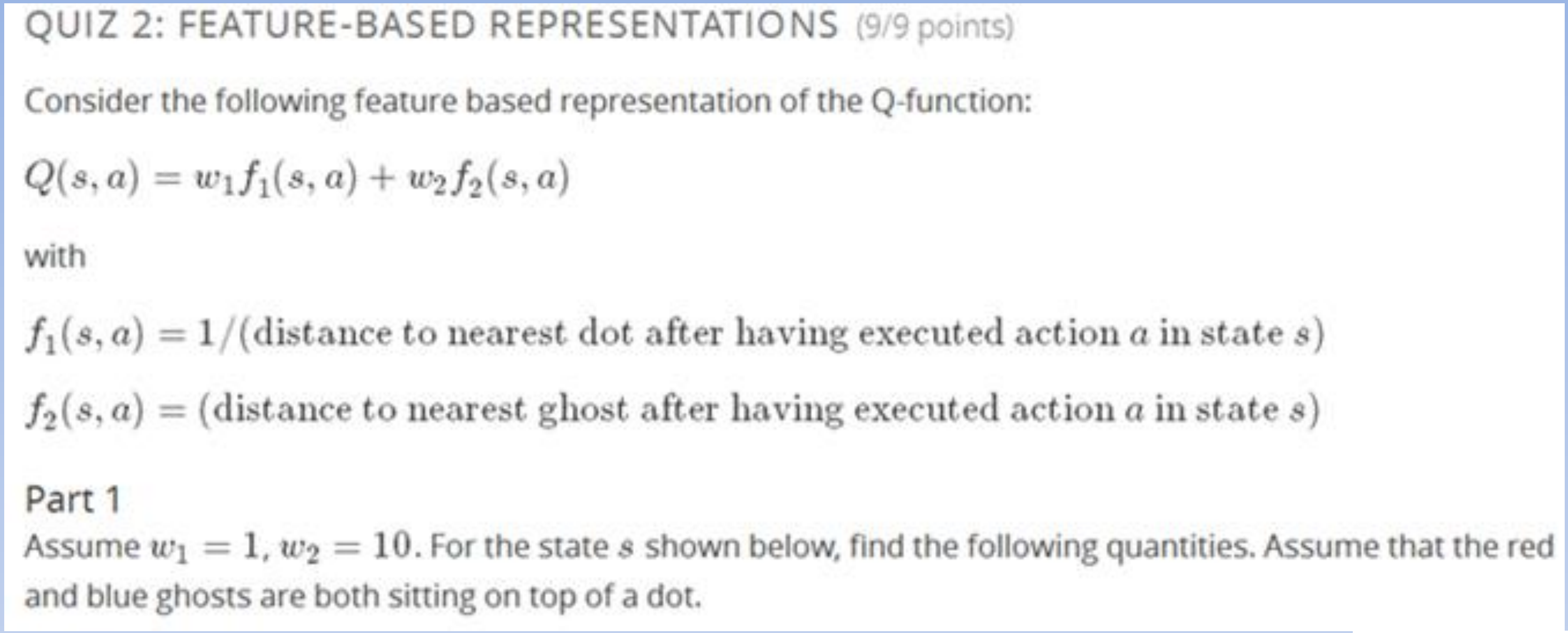
$$f_2(s,a) = (\text{distance to nearest ghost after having executed action } a \text{ in state } s)$$

Part 1

Assume $w_1 = 1$, $w_2 = 10$. For the state s shown below, find the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.



- Q(s, West) = ?
- Q(s, South) = ?
- Based on this approximate Q-function, which action would be chosen?



QUIZ 2: FEATURE-BASED REPRESENTATIONS (9/9 points)

Consider the following feature based representation of the Q-function:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a)$$

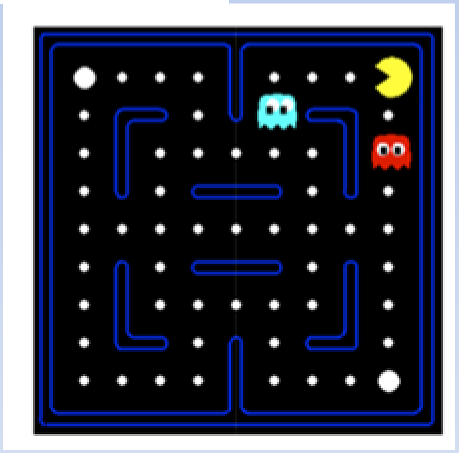
with

$$f_1(s,a) = 1/(\text{distance to nearest dot after having executed action } a \text{ in state } s)$$

$$f_2(s,a) = (\text{distance to nearest ghost after having executed action } a \text{ in state } s)$$

Part 1

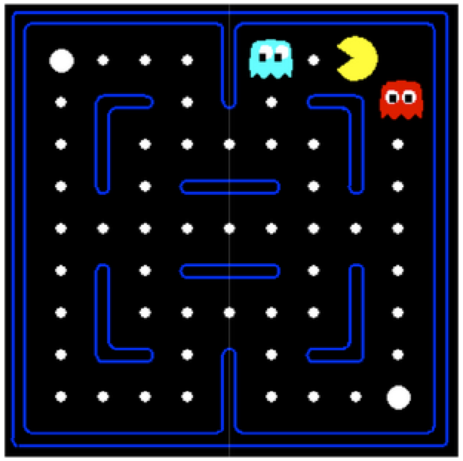
Assume $w_1 = 1$, $w_2 = 10$. For the state s shown below, find the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.



- Q(s,West) =
  - 1*(1/1)+10*(3 moves away)
- Q(s,South) =
  - 1*(1/1)+10*(1 move away)
- Based on this approximate Q-function, which action would be chosen?

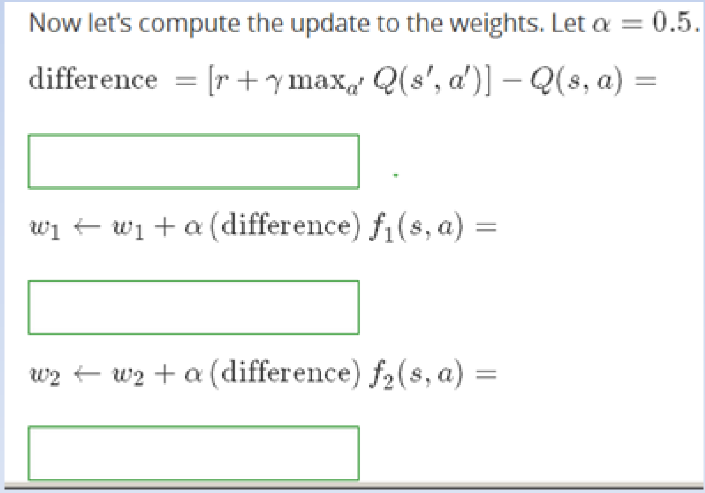
Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

- Q-Values:
  - Q(s', West) = ?
  - Q(s', East) = ?
- Now compute update with $\alpha = 0.5$:
  - Sample =
  - Difference =
  - w_1=
  - w_2=



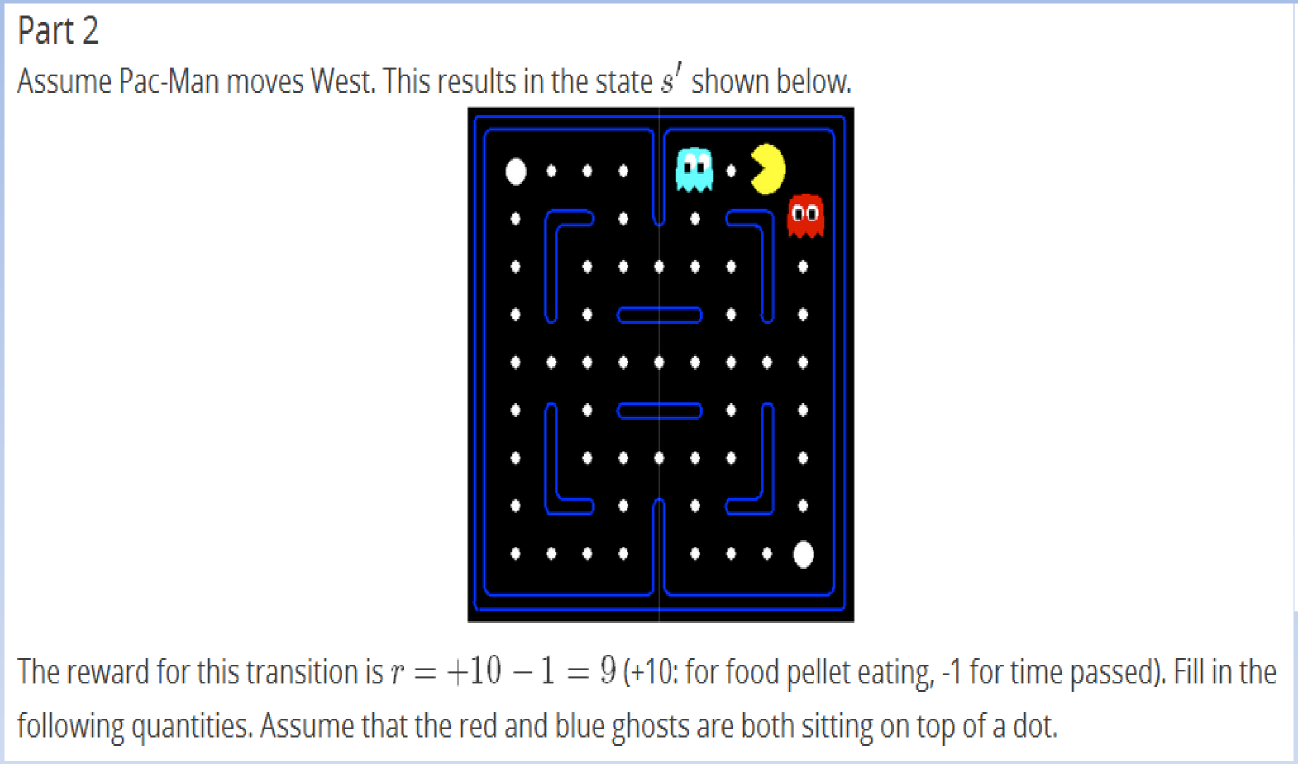
Now let's compute the update to the weights. Let $\alpha = 0.5$.

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

$w_1 \leftarrow w_1 + \alpha\,(\text{difference})\, f_1(s, a) =$

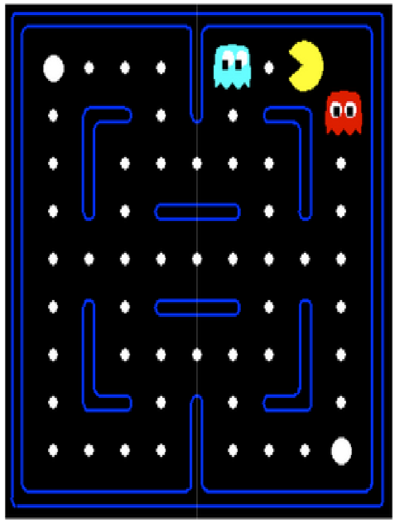
$w_2 \leftarrow w_2 + \alpha\,(\text{difference})\, f_2(s, a) =$

Question

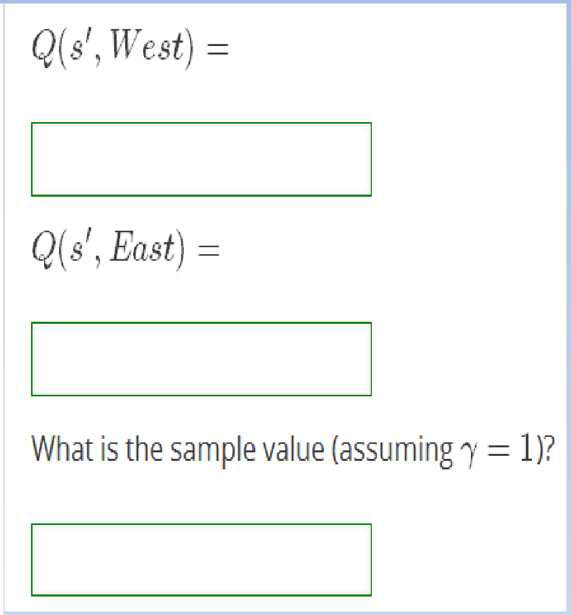


Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.



$Q(s', West) =$

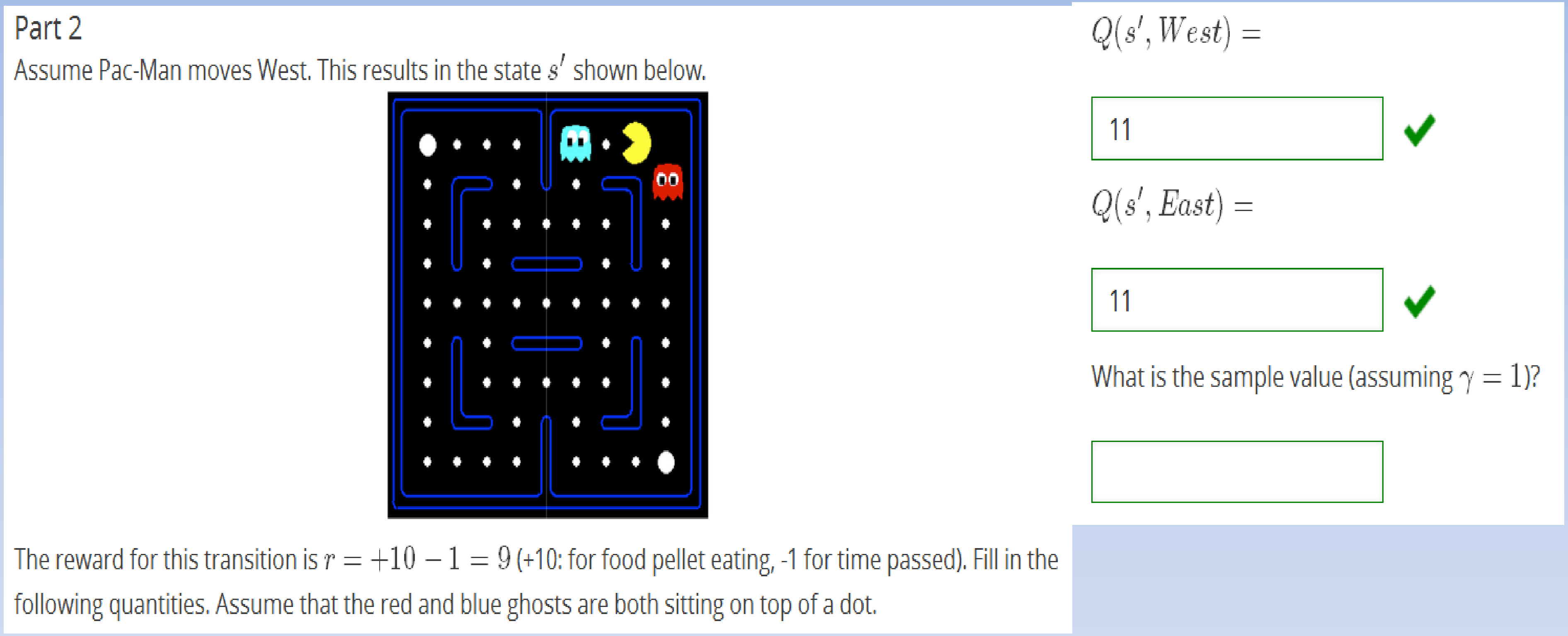
$Q(s', East) =$

What is the sample value (assuming $\gamma = 1$)?

Q(s',West) = 1*(1/1)+10*(1 move away)

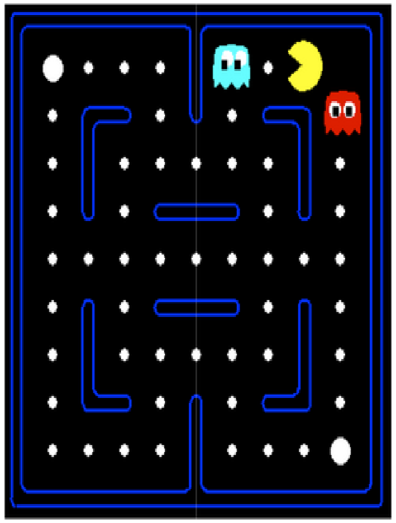
Q(s',East) = 1*(1/1)+10*(1 move away)

Question



Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$Q(s', West) =$

11 ✓

$Q(s', East) =$

11 ✓

What is the sample value (assuming $\gamma = 1$)?

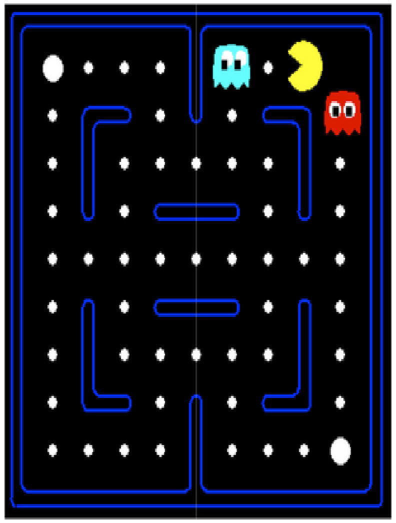
Q(s',West) = 1*(1/1)+10*(1 move away)

Q(s',East) = 1*(1/1)+10*(1 move away)

Question

Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

$Q(s', West) =$

11 ✓

$Q(s', East) =$

11 ✓

What is the sample value (assuming $\gamma = 1$)?

Q(s',West) = 1*(1/1)+10*(1 move away)

Q(s',East) = 1*(1/1)+10*(1 move away)

Sample = 9 (Reward) + 1 (γ) * 11 ($\max_{a'} Q(s',a')$)=20

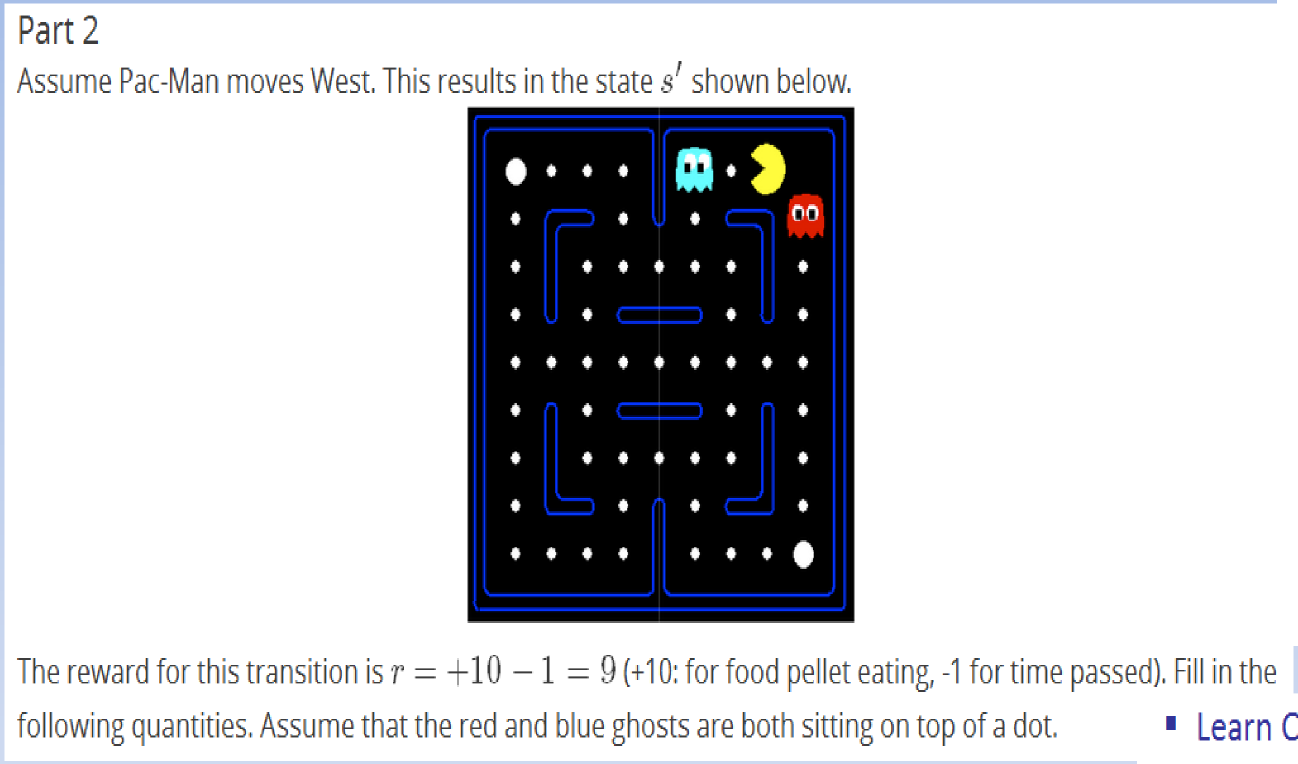
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

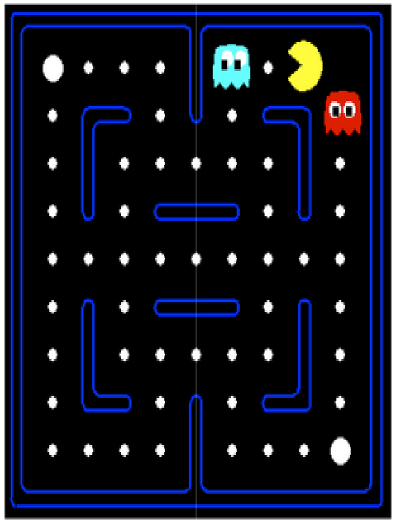
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

Question

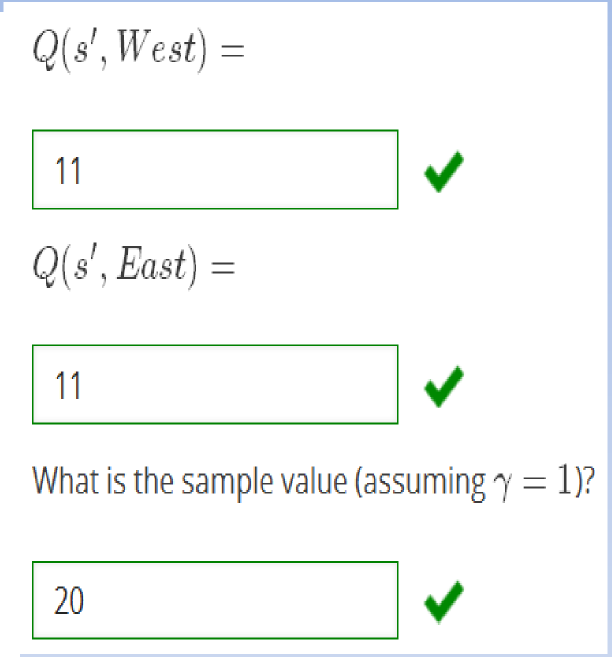


Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.



$Q(s', West) =$

11 ✓

$Q(s', East) =$

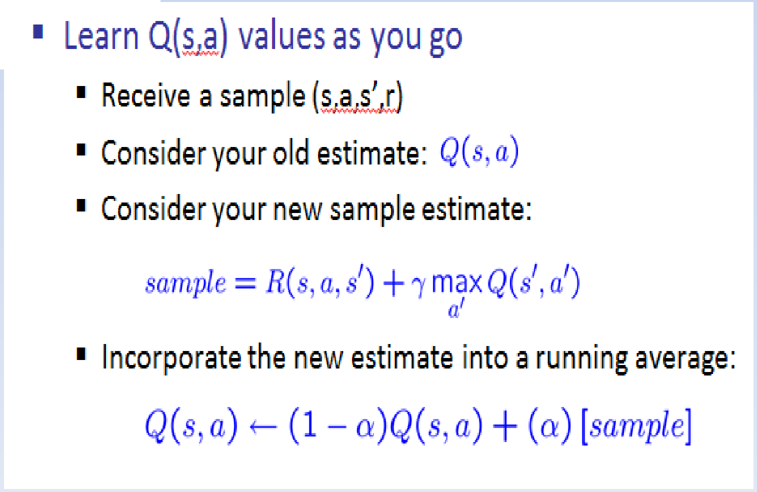
11 ✓

What is the sample value (assuming $\gamma = 1$)?

20 ✓

Q(s',West) = 1*(1/1)+10*(1 move away)
Q(s',East) = 1*(1/1)+10*(1 move away)

Sample = 9 (Reward) + 1 (γ) * 11 ($\max_{a'} Q(s',a')$)=20



- 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:

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

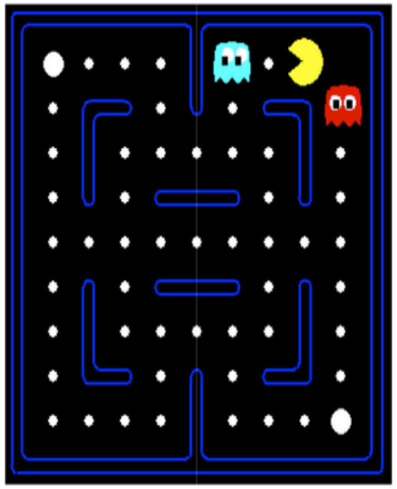
  - Incorporate the new estimate into a running average:

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

Question

Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in th
following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

Part 3

Now let's compute the update to the weights. Let $\alpha = 0.5$.

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

$w_1 \leftarrow w_1 + \alpha\,(\text{difference})\, f_1(s, a) =$

$w_2 \leftarrow w_2 + \alpha\,(\text{difference})\, f_2(s, a) =$

Q(s’,West) = 1*(1/1)+10*(1 move away)

Q(s’,East) = 1*(1/1)+10*(1 move away)

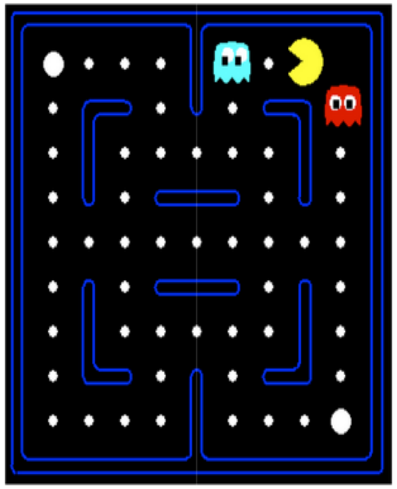
Sample = 9 (Reward) + 1 (γ) * 11 ($\max_{a'}$Q(s’,a’))=20

Difference = 20 (Sample) – 31 (Q(s,west))=-11

Question

Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

Part 3

Now let's compute the update to the weights. Let $\alpha = 0.5$.

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

-11 ✓

$w_1 \leftarrow w_1 + \alpha\,(\text{difference})\, f_1(s, a) =$

$w_2 \leftarrow w_2 + \alpha\,(\text{difference})\, f_2(s, a) =$

Q(s',West) = 1*(1/1)+10*(1 move away)

Q(s',East) = 1*(1/1)+10*(1 move away)

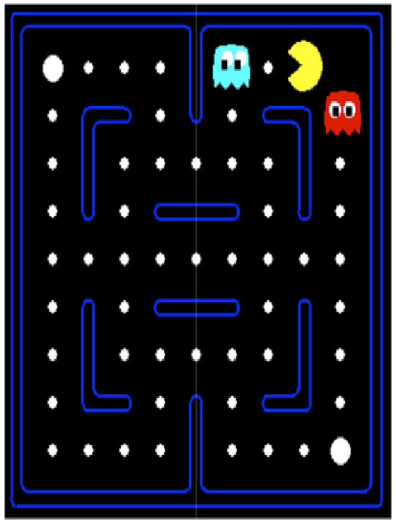
Sample = 9 (Reward) + 1 (γ) * 11 ($\max_{a'}$Q(s',a'))=20

Difference = 20 (Sample) – 31 (Q(s,west))=-11

Question

Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

Part 3

Now let's compute the update to the weights. Let $\alpha = 0.5$.

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

-11 ✓

$w_1 \leftarrow w_1 + \alpha\,(\text{difference})\, f_1(s, a) =$

$w_2 \leftarrow w_2 + \alpha\,(\text{difference})\, f_2(s, a) =$

Q(s’,West) = 1*(1/1)+10*(1 move away)

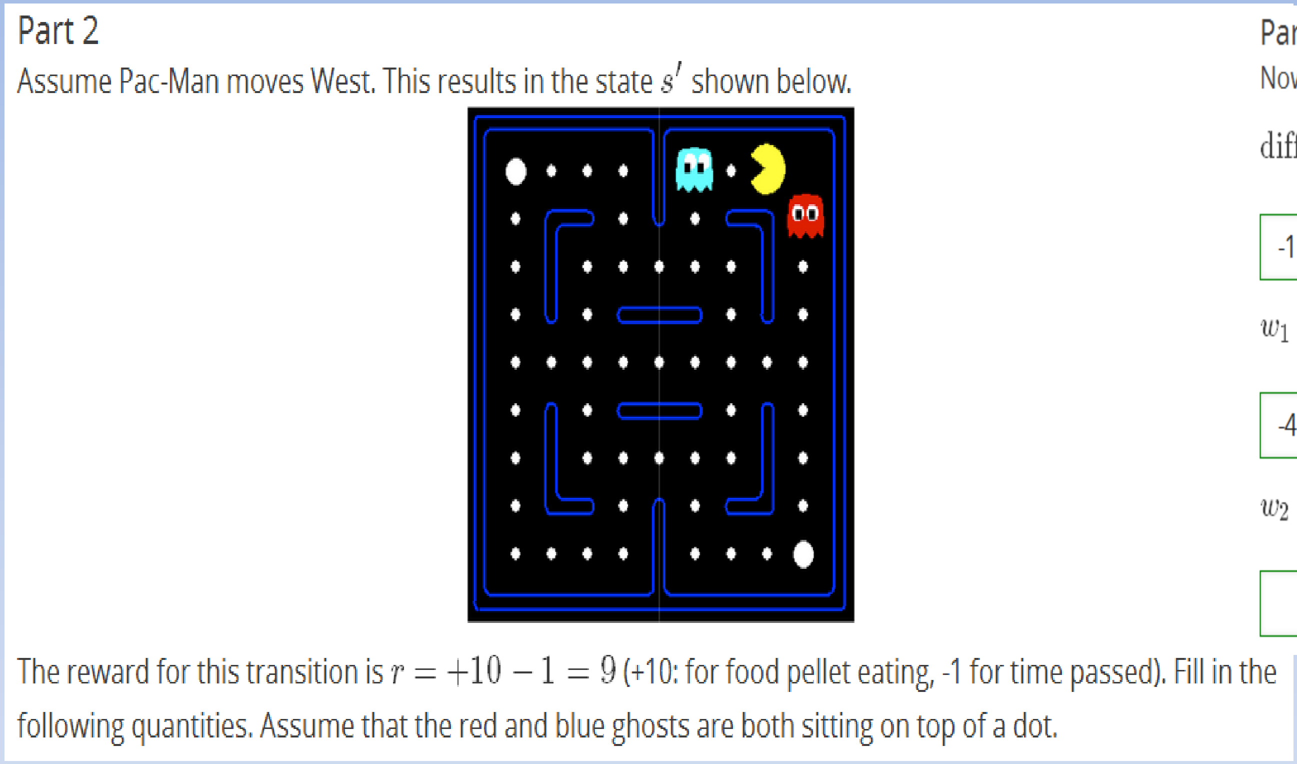
Q(s’,East) = 1*(1/1)+10*(1 move away)

Sample = 9 (Reward) + 1 (γ) * 11 ($\max_{a'}$Q(s’,a’))=20

Difference = 20 (Sample) – 31 (Q(s,west))=-11

$w_1 \leftarrow$ 1 (w_1) + 0.5 (α) * 1 * (-11 diff) * (f_1(s,west)) = 1- 5.5 = -4.5

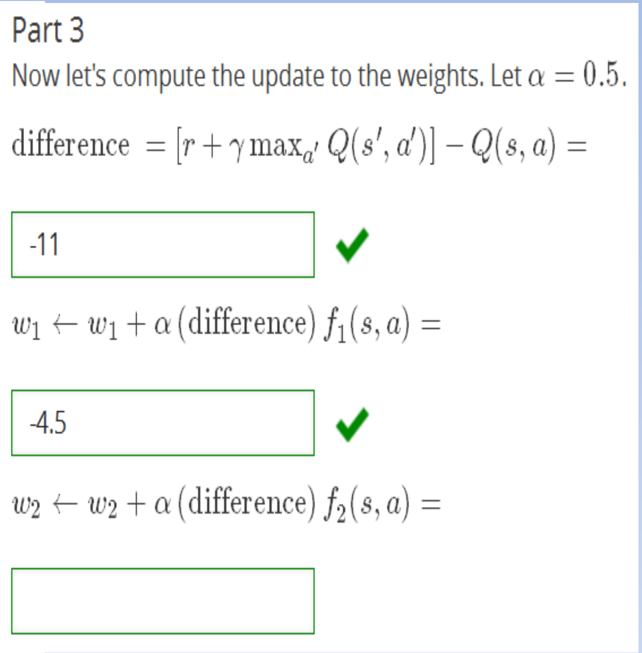
Question



Part 2

Assume Pac-Man moves West. This results in the state s' shown below.

The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.



Part 3

Now let's compute the update to the weights. Let $\alpha = 0.5$.

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

-11 ✓

$w_1 \leftarrow w_1 + \alpha\,(\text{difference})\, f_1(s, a) =$

-4.5 ✓

$w_2 \leftarrow w_2 + \alpha\,(\text{difference})\, f_2(s, a) =$

Q(s',West) = 1*(1/1)+10*(1 move away)

Q(s',East) = 1*(1/1)+10*(1 move away)

Sample = 9 (Reward) + 1 (γ) * 11 ($\max_{a'}$Q(s',a'))=20

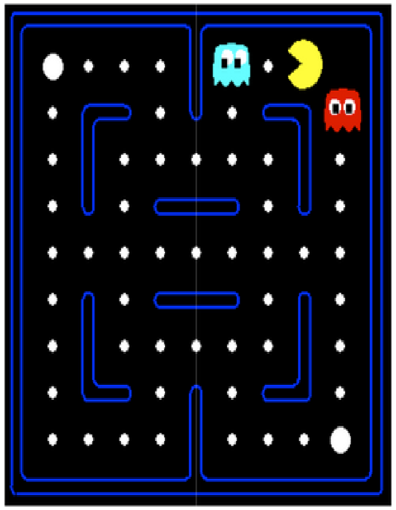
Difference = 20 (Sample) – 31 (Q(s,west))=-11

$w_1 \leftarrow$ 1 (w_1) + 0.5 (α) * 1 * (-11 diff) * (f_1(s,west)) = 1- 5.5 = -4.5

Question

Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in the following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

Part 3

Now let's compute the update to the weights. Let $\alpha = 0.5$.

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

-11 ✓

$w_1 \leftarrow w_1 + \alpha\,(\text{difference})\, f_1(s, a) =$

-4.5 ✓

$w_2 \leftarrow w_2 + \alpha\,(\text{difference})\, f_2(s, a) =$

Sample = 9 (Reward) + 1 (γ) * 11 ($\max_{a'}Q(s',a')$)=20

Difference = 20 (Sample) – 31 (Q(s,west))=-11

$w_1 \leftarrow$ 1 (w_1) + 0.5 (α) * 1 * (-11 diff) * (1 f_1(s,west)) = 1- 5.5 = -4.5

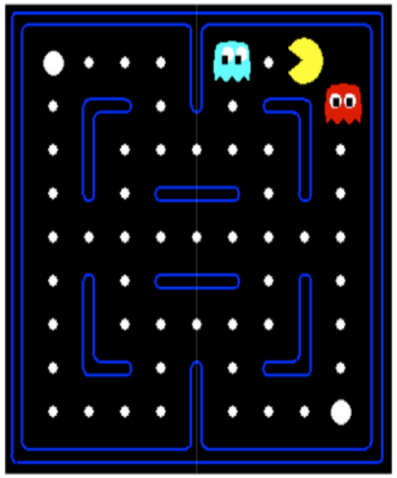
$w_2 \leftarrow$ 10 (w_2) + 0.5 (α) * 1 * (-11 diff) * (3 f_2(s,west)) =

10 + 0.5 * (-11*3) = 10 + 0.5*(-33)= 10-16.5 = -6.5

Question

Part 2

Assume Pac-Man moves West. This results in the state s' shown below.



The reward for this transition is $r = +10 - 1 = 9$ (+10: for food pellet eating, -1 for time passed). Fill in following quantities. Assume that the red and blue ghosts are both sitting on top of a dot.

Part 3

Now let's compute the update to the weights. Let $\alpha = 0.5$.

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

-11 ✓

$w_1 \leftarrow w_1 + \alpha\,(\text{difference})\, f_1(s, a) =$

-4.5 ✓

$w_2 \leftarrow w_2 + \alpha\,(\text{difference})\, f_2(s, a) =$

-6.5 ✓

Sample = 9 (Reward) + 1 (γ) * 11 ($\max_{a'}Q(s',a')$)=20

Difference = 20 (Sample) – 31 (Q(s,west))=-11

$w_1 \leftarrow$ 1 (w_1) + 0.5 (α) * 1 * (-11 diff) * (1 f_1(s,west)) = 1- 5.5 = -4.5

$w_2 \leftarrow$ 10 (w_2) + 0.5 (α) * 1 * (-11 diff) * (3 f_2(s,west)) =

10 + 0.5 * (-11*3) = 10 + 0.5*(-33)= 10-16.5 = -6.5

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
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Exploration Functions

- When to explore?
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring
- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g.

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

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

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

  - Note: this propagates the "bonus" back to states that lead to unknown states as well!

Reinforcement Learning

Model-Based Learning

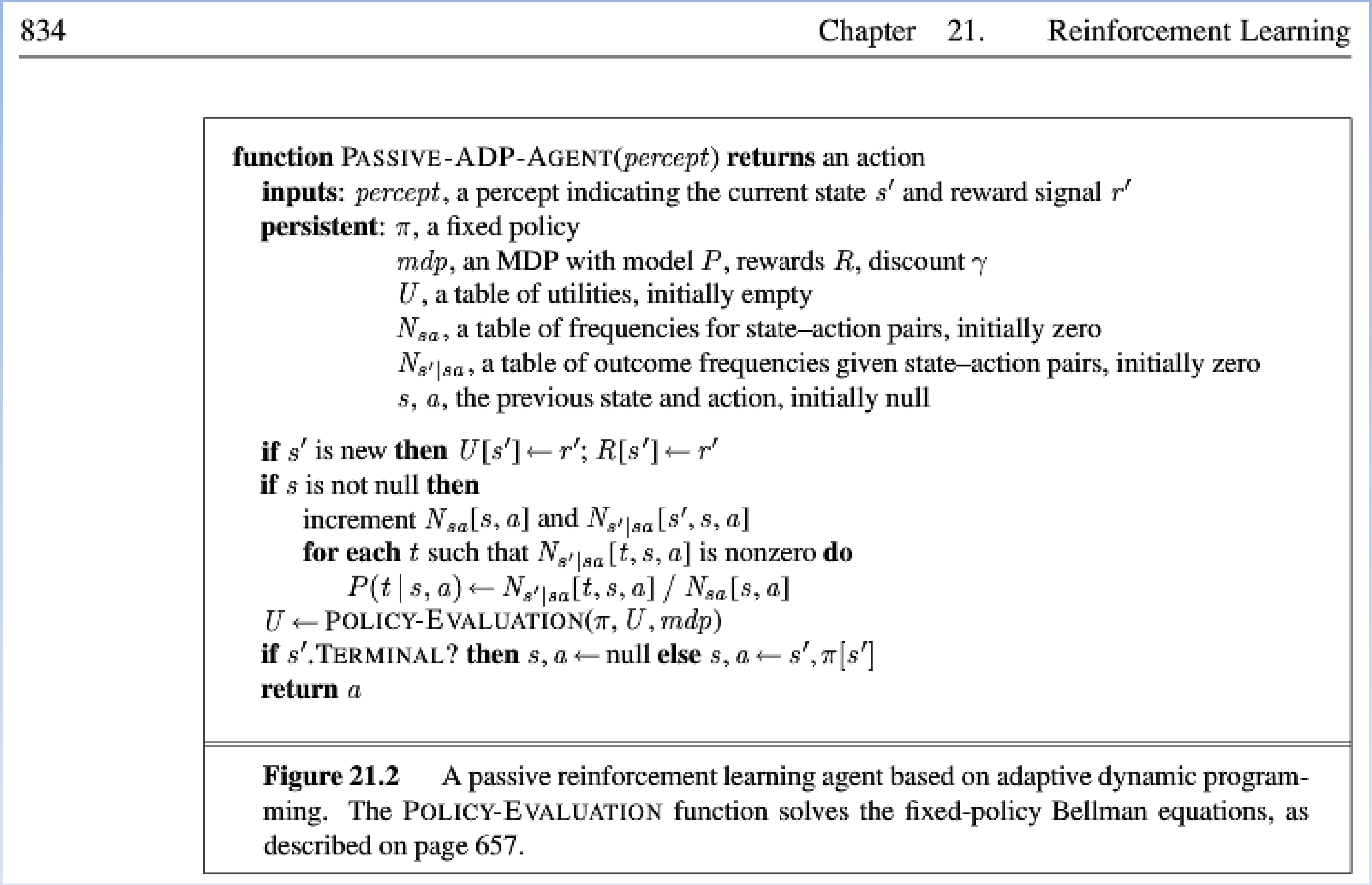
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- 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’)

- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Passive Learning Agent

- State-based Representation
- Fully observable environment
- Fixed policy π
- In state s agent always executes action $\pi(s)$



```
function PASSIVE-ADP-AGENT($percept$) returns an action
  inputs: $percept$, a percept indicating the current state $s'$ and reward signal $r'$
  persistent: $\pi$, a fixed policy
              $mdp$, an MDP with model $P$, rewards $R$, discount $\gamma$
              $U$, a table of utilities, initially empty
              $N_{sa}$, a table of frequencies for state–action pairs, initially zero
              $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
              $s$, $a$, the previous state and action, initially null

  if $s'$ is new then $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
  if $s$ is not null then
     increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
     for each $t$ such that $N_{s'|sa}[t, s, a]$ is nonzero do
         $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \;/\; N_{sa}[s, a]$
  $U \leftarrow$ POLICY-EVALUATION($\pi, U, mdp$)
  if $s'$.TERMINAL? then $s, a \leftarrow$ null else $s, a \leftarrow s', \pi[s']$
  return $a$
```

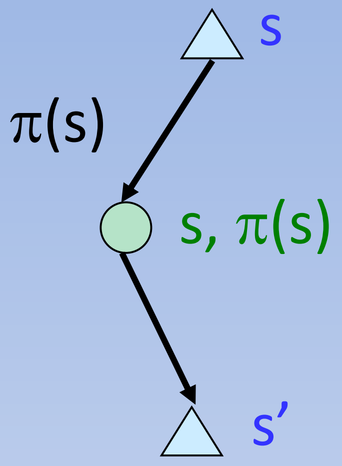
Figure 21.2 A passive reinforcement learning agent based on adaptive dynamic programming. The POLICY-EVALUATION function solves the fixed-policy Bellman equations, as described on page 657.

Direct-Evaluation

- Goal: Compute values for each state under π

- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

Temporal-Difference Learning



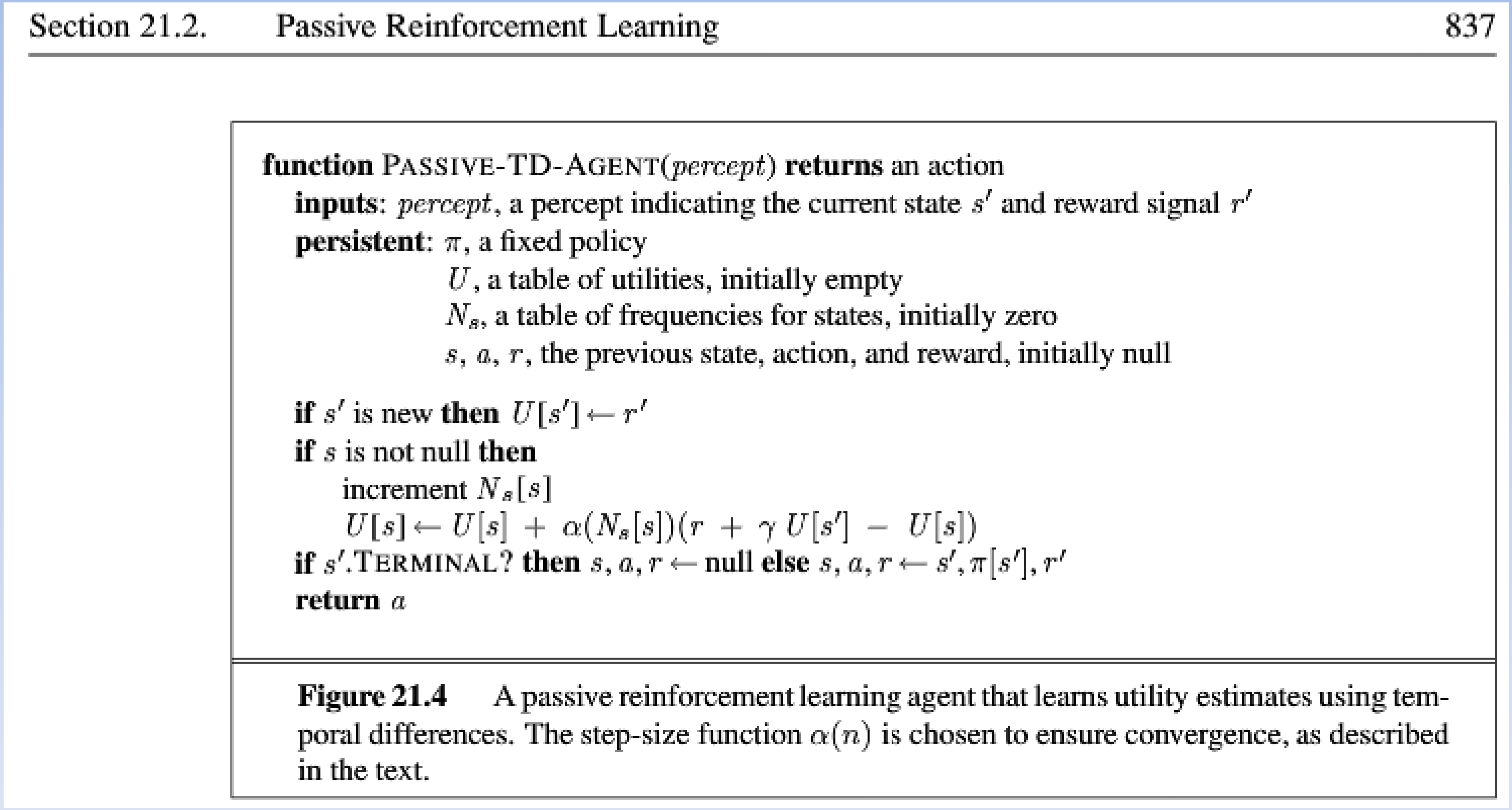
$$V_{k+1}{}^{\pi}(s) = \frac{1}{n}\sum_i sample_i$$

sample=$R(s,\pi(s),s') + \gamma V_k{}^{\pi}(s')$

$$\boldsymbol{V_{k+1}{}^{\pi}(s) \leftarrow (1-\alpha)V_k{}^{\pi}(s) + \alpha * sample}$$

$$V_{k+1}{}^{\pi}(s) \leftarrow V_k{}^{\pi}(s) + \alpha(sample - V_k{}^{\pi}(s))$$

Temporal Difference Learning



Section 21.2. Passive Reinforcement Learning 837

```
function PASSIVE-TD-AGENT(percept) returns an action
   inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
   persistent: $\pi$, a fixed policy
               $U$, a table of utilities, initially empty
               $N_s$, a table of frequencies for states, initially zero
               $s$, $a$, $r$, the previous state, action, and reward, initially null

   if $s'$ is new then $U[s'] \leftarrow r'$
   if $s$ is not null then
      increment $N_s[s]$
      $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
   if $s'$.TERMINAL? then $s, a, r \leftarrow$ null else $s, a, r \leftarrow s', \pi[s'], r'$
   return $a$
```

Figure 21.4 A passive reinforcement learning agent that learns utility estimates using temporal differences. The step-size function $\alpha(n)$ is chosen to ensure convergence, as described in the text.

Now w/ Model-Free Learning
Q-Learning

- Q-Learning: sample-based Q-value iteration

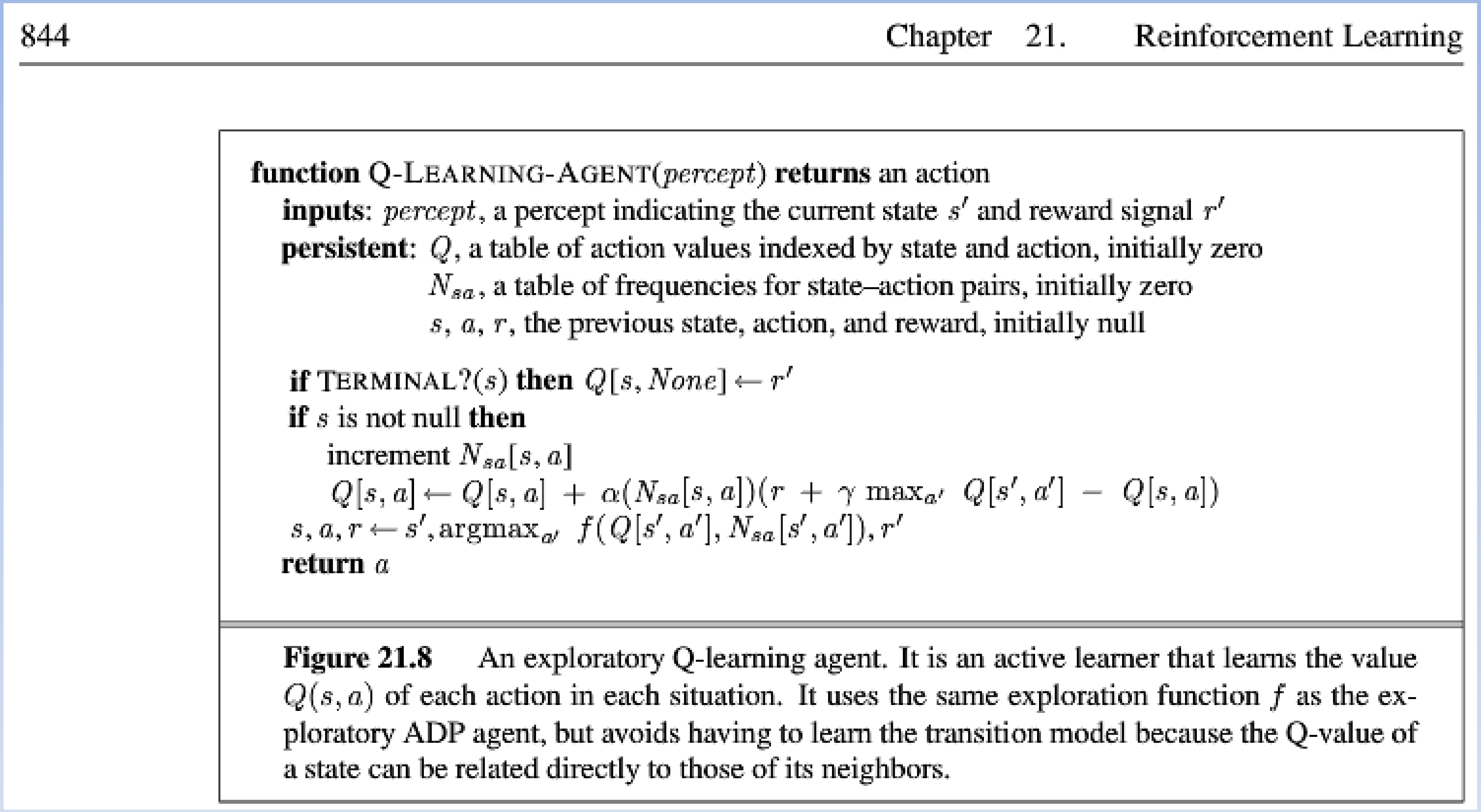
$$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a} Q_k(s,a)]$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s’,r)
  - Consider your old estimate:
  - Consider your new sample estimate:

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

- Incorporate the new estimate into a running average:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha(sample)$$
$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha(sample - Q_k(s,a))$$



```
function Q-LEARNING-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
  persistent: $Q$, a table of action values indexed by state and action, initially zero
              $N_{sa}$, a table of frequencies for state–action pairs, initially zero
              $s, a, r$, the previous state, action, and reward, initially null

  if TERMINAL?($s$) then $Q[s, None] \leftarrow r'$
  if $s$ is not null then
      increment $N_{sa}[s, a]$
      $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \mathrm{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return $a$
```

Figure 21.8 An exploratory Q-learning agent. It is an active learner that learns the value $Q(s, a)$ of each action in each situation. It uses the same exploration function f as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.

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:

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

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

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

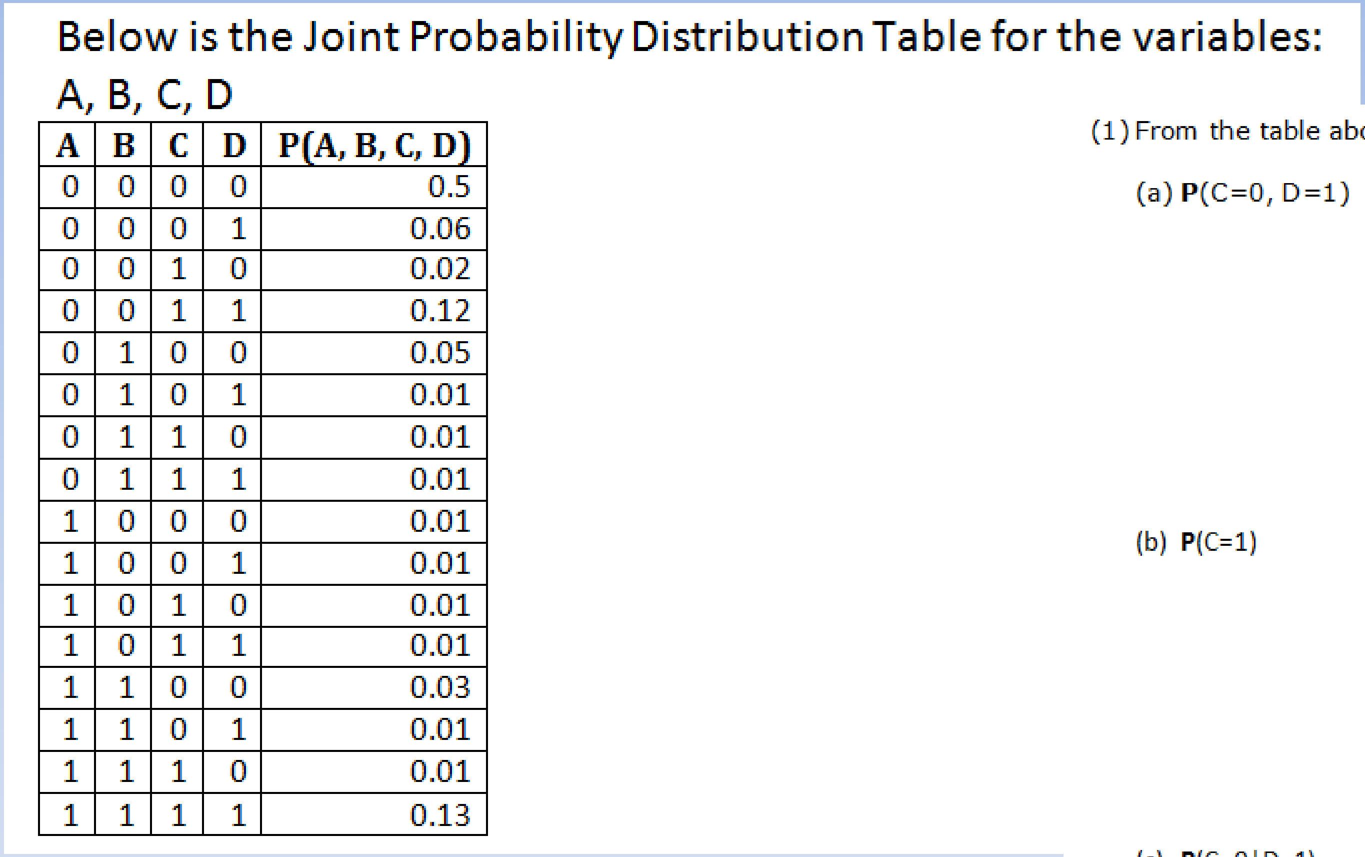
$$w_i \leftarrow w_i + \alpha\, [\text{difference}]\, f_i(s,a)$$ Approximate Q's

- 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
- Formal justification: online least squares

Probabilistic Reasoning

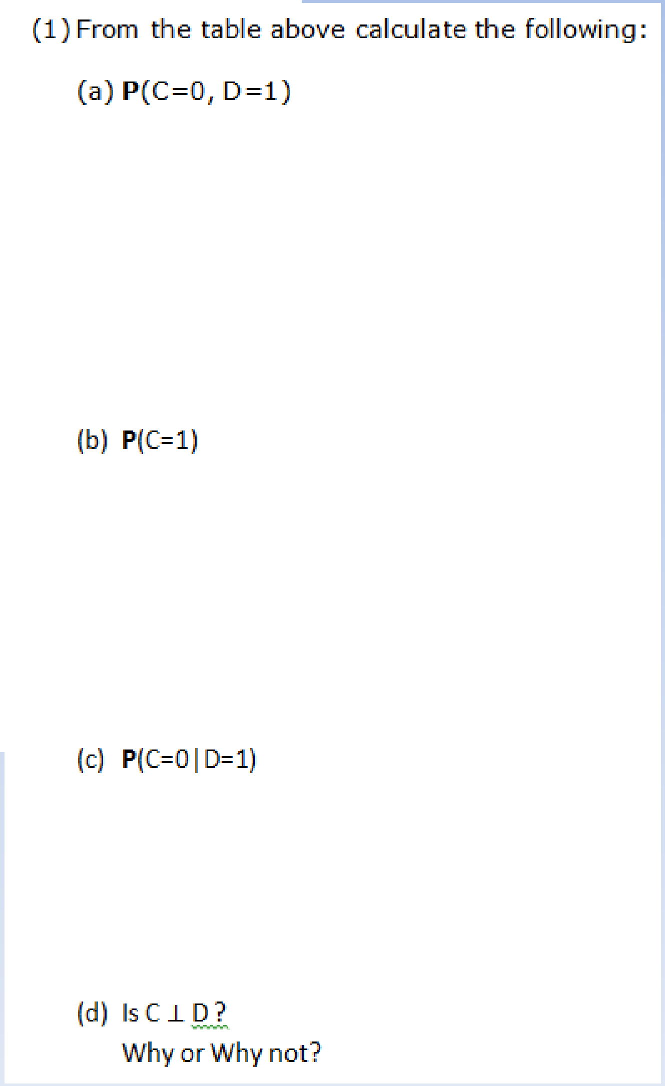
- Probability Theory
  - Marginal Probability
  - Conditional Probability

Question


Below is the Joint Probability Distribution Table for the variables:

A, B, C, D

A	B	C	D	P(A, B, C, D)
0	0	0	0	0.5
0	0	0	1	0.06
0	0	1	0	0.02
0	0	1	1	0.12
0	1	0	0	0.05
0	1	0	1	0.01
0	1	1	0	0.01
0	1	1	1	0.01
1	0	0	0	0.01
1	0	0	1	0.01
1	0	1	0	0.01
1	0	1	1	0.01
1	1	0	0	0.03
1	1	0	1	0.01
1	1	1	0	0.01
1	1	1	1	0.13


(1) From the table above calculate the following:

(a) **P**(C=0, D=1)

(b) **P**(C=1)

(c) **P**(C=0|D=1)

(d) Is $C \perp D$?

Why or Why not?

Question:
P(C=0, D=1)

A	B	C	D	P(A,B,C,D)
0	0	0	0	0.5
0	0	0	1	0.06
0	0	1	0	0.02
0	0	1	1	0.12
0	1	0	0	0.05
0	1	0	1	0.01
0	1	1	0	0.01
0	1	1	1	0.01
1	0	0	0	0.01
1	0	0	1	0.01
1	0	1	0	0.01
1	0	1	1	0.01
1	1	0	0	0.03
1	1	0	1	0.01
1	1	1	0	0.01
1	1	1	1	0.13

Question:

P(C=0, D=1)

A	B	C	D	P(A,B,C,D)
0	0	0	1	0.06
0	1	0	1	0.01
1	0	0	1	0.01
1	1	0	1	0.01
				=0.09

Question:
P(C=1)

A	B	C	D	P(A,B,C,D)
0	0	0	0	0.5
0	0	0	1	0.06
0	0	1	0	0.02
0	0	1	1	0.12
0	1	0	0	0.05
0	1	0	1	0.01
0	1	1	0	0.01
0	1	1	1	0.01
1	0	0	0	0.01
1	0	0	1	0.01
1	0	1	0	0.01
1	0	1	1	0.01
1	1	0	0	0.03
1	1	0	1	0.01
1	1	1	0	0.01
1	1	1	1	0.13

Question: **P**(C=1)

A	B	C	D	P(A,B,C,D)
0	0	1	0	0.02
0	0	1	1	0.12
0	1	1	0	0.01
0	1	1	1	0.01
1	0	1	0	0.01
1	0	1	1	0.01
1	1	1	0	0.01
1	1	1	1	0.13
				=0.32

Question:
P(C=0 | D=1)

A	B	C	D	P(A,B,C,D)
0	0	0	0	0.5
0	0	0	1	0.06
0	0	1	0	0.02
0	0	1	1	0.12
0	1	0	0	0.05
0	1	0	1	0.01
0	1	1	0	0.01
0	1	1	1	0.01
1	0	0	0	0.01
1	0	0	1	0.01
1	0	1	0	0.01
1	0	1	1	0.01
1	1	0	0	0.03
1	1	0	1	0.01
1	1	1	0	0.01
1	1	1	1	0.13

Question:
$\mathbf{P}(C=0 \mid D=1)=P(C=0,D=1)/P(D=1)$

A	B	C	D	P(A,B,C,D)
0	0	0	0	0.5
0	0	0	1	0.06
0	0	1	0	0.02
0	0	1	1	0.12
0	1	0	0	0.05
0	1	0	1	0.01
0	1	1	0	0.01
0	1	1	1	0.01
1	0	0	0	0.01
1	0	0	1	0.01
1	0	1	0	0.01
1	0	1	1	0.01
1	1	0	0	0.03
1	1	0	1	0.01
1	1	1	0	0.01
1	1	1	1	0.13

Question: $\mathbf{P}(C=0 \mid D=1)=P(C=0,D=1)/P(D=1)$

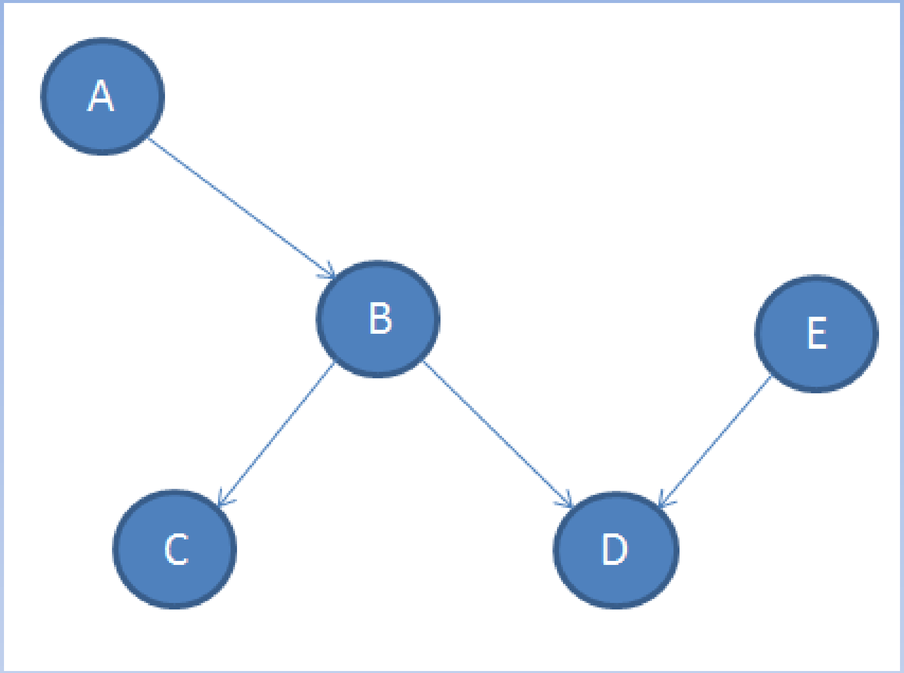
A	B	C	D	P(A,B,C,D)
0	0	0	0	0.5
0	0	0	1	0.06
0	0	1	0	0.02
0	0	1	1	0.12
0	1	0	0	0.05
0	1	0	1	0.01
0	1	1	0	0.01
0	1	1	1	0.01
1	0	0	0	0.01
1	0	0	1	0.01
1	0	1	0	0.01
1	0	1	1	0.01
1	1	0	0	0.03
1	1	0	1	0.01
1	1	1	0	0.01
1	1	1	1	0.13

$\mathbf{P}(C=0 \mid D=1)$
$=P(C=0,D=1)/P(D=1)$
$=0.09/P(D=1)$

Question: **P**(C=0 | D=1)=P(C=0,D=1)/P(D=1)

A	B	C	D	P(A,B,C,D)
0	0	0	1	0.06
0	0	1	1	0.12
0	1	0	1	0.01
0	1	1	1	0.01
1	0	0	1	0.01
1	0	1	1	0.01
1	1	0	1	0.01
1	1	1	1	0.13
				=0.36

P(C=0 | D=1)
=P(C=0,D=1)/P(D=1)
=0.09/P(D=1)
=0.09/0.36
=1/4=0.25



Question:

- Given the Bayes Net above:
  - What would be the set of Conditional Probability Distributions needed to represent the full joint probability distribution over A, B, C, D, E?
    - P(A), P(B|A), P(C|B), P(D|B, E), P(E)
  - Show the formula for computing the full joint probability distribution over all the variables with the Conditional Probability Distributions.
    - P(A,B,C,D,E) = P(A) * P(B|A) * P(C|B) * P(D|B, E) * P(E)

(1) Given the Probability Distribution Table Below:

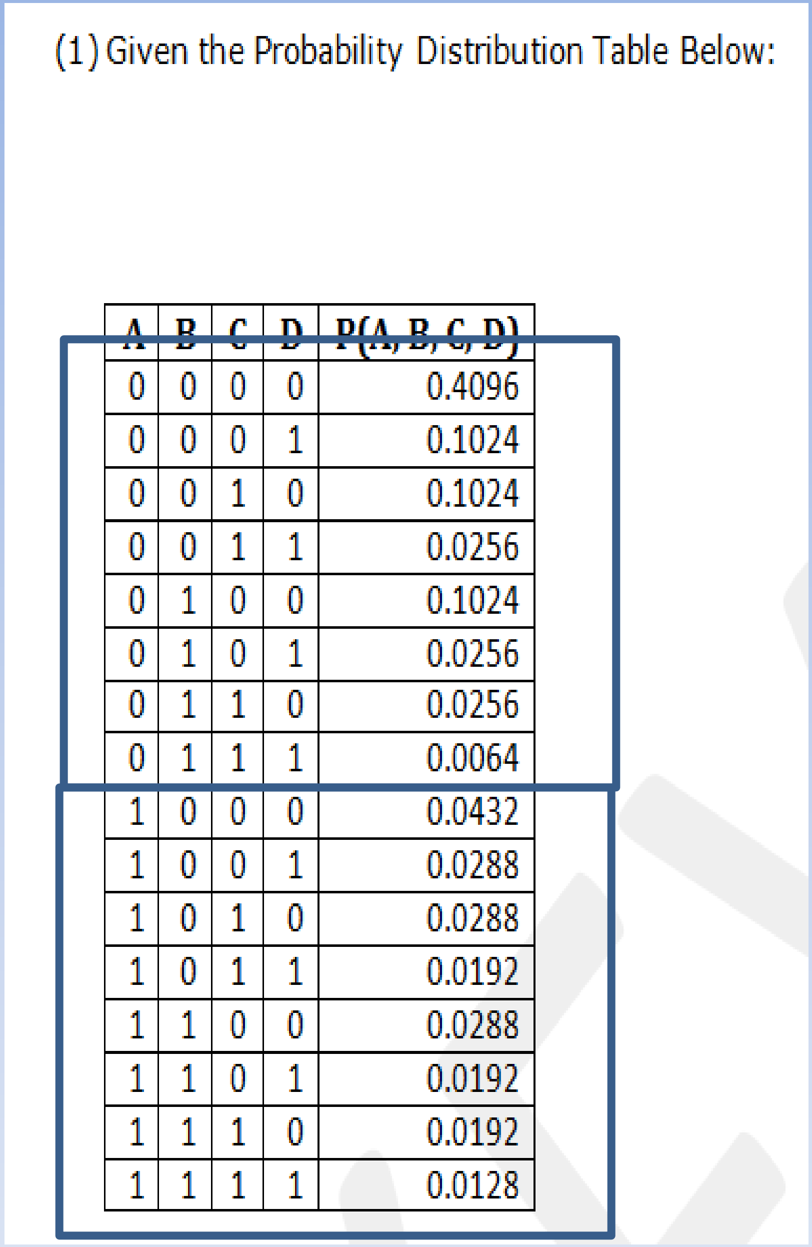
A	B	C	D	P(A, B, C, D)
0	0	0	0	0.4096
0	0	0	1	0.1024
0	0	1	0	0.1024
0	0	1	1	0.0256
0	1	0	0	0.1024
0	1	0	1	0.0256
0	1	1	0	0.0256
0	1	1	1	0.0064
1	0	0	0	0.0432
1	0	0	1	0.0288
1	0	1	0	0.0288
1	0	1	1	0.0192
1	1	0	0	0.0288
1	1	0	1	0.0192
1	1	1	0	0.0192
1	1	1	1	0.0128

Calculate the following using the next page to show work:

(a) **P**(A)

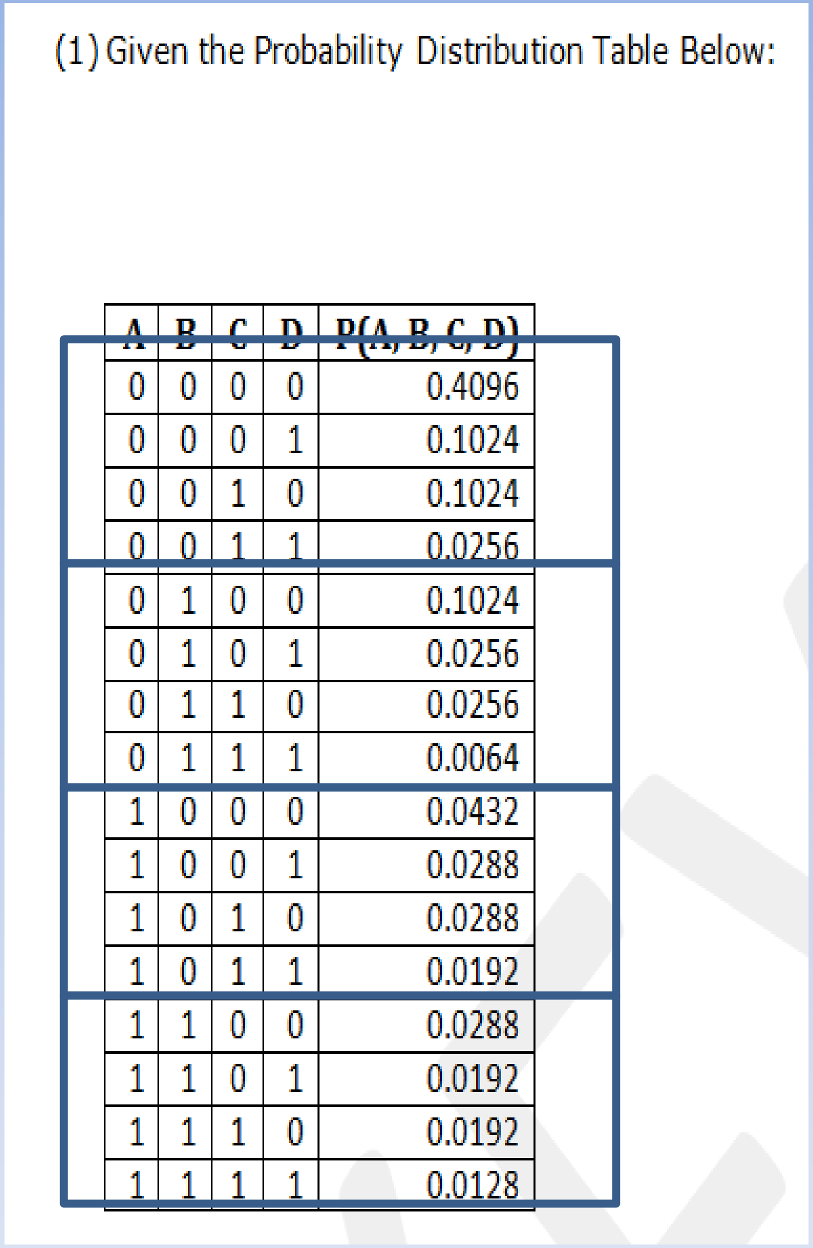
(b) **P**(B|A)

(c) **P**(C|A)


(1) Given the Probability Distribution Table Below:

A	B	C	D	P(A, B, C, D)
0	0	0	0	0.4096
0	0	0	1	0.1024
0	0	1	0	0.1024
0	0	1	1	0.0256
0	1	0	0	0.1024
0	1	0	1	0.0256
0	1	1	0	0.0256
0	1	1	1	0.0064
1	0	0	0	0.0432
1	0	0	1	0.0288
1	0	1	0	0.0288
1	0	1	1	0.0192
1	1	0	0	0.0288
1	1	0	1	0.0192
1	1	1	0	0.0192
1	1	1	1	0.0128

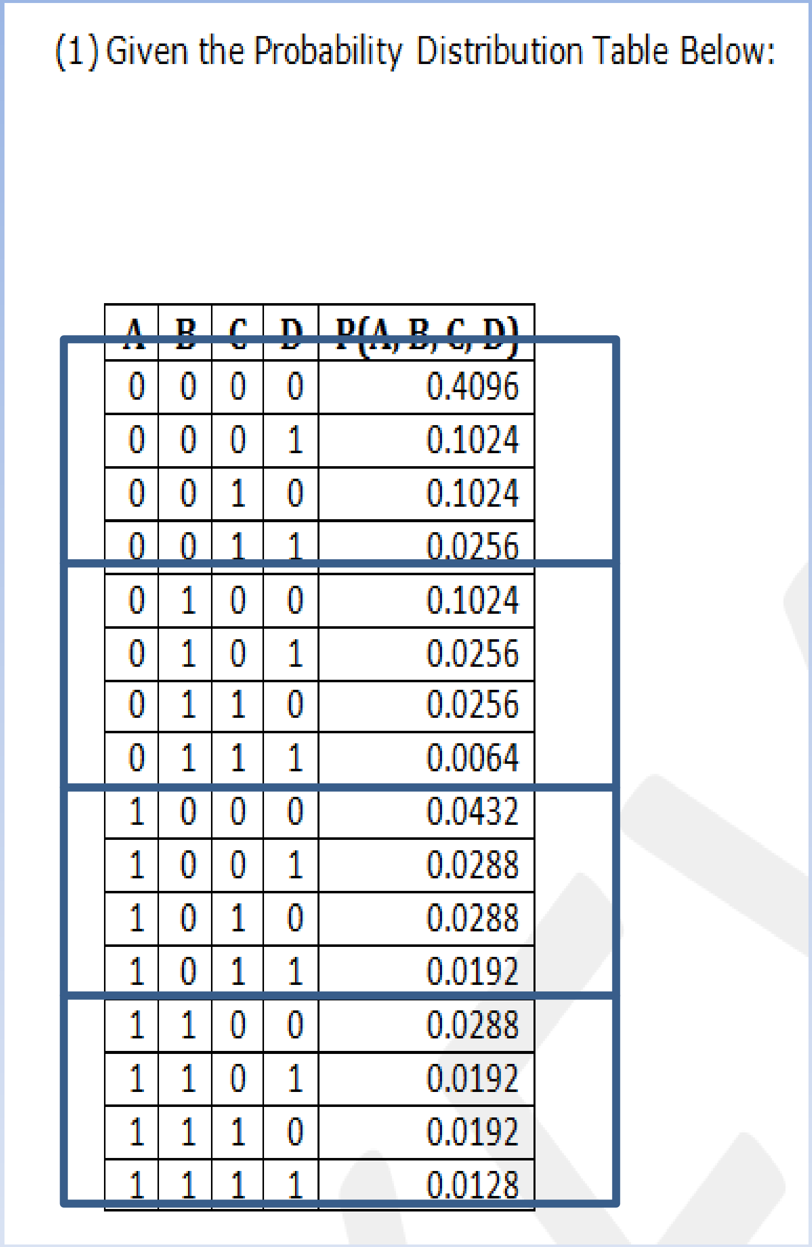
- $P(A=0) = 0.8$
- $P(A=1) = 0.2$



(1) Given the Probability Distribution Table Below:

A	B	C	D	P(A, B, C, D)
0	0	0	0	0.4096
0	0	0	1	0.1024
0	0	1	0	0.1024
0	0	1	1	0.0256
0	1	0	0	0.1024
0	1	0	1	0.0256
0	1	1	0	0.0256
0	1	1	1	0.0064
1	0	0	0	0.0432
1	0	0	1	0.0288
1	0	1	0	0.0288
1	0	1	1	0.0192
1	1	0	0	0.0288
1	1	0	1	0.0192
1	1	1	0	0.0192
1	1	1	1	0.0128

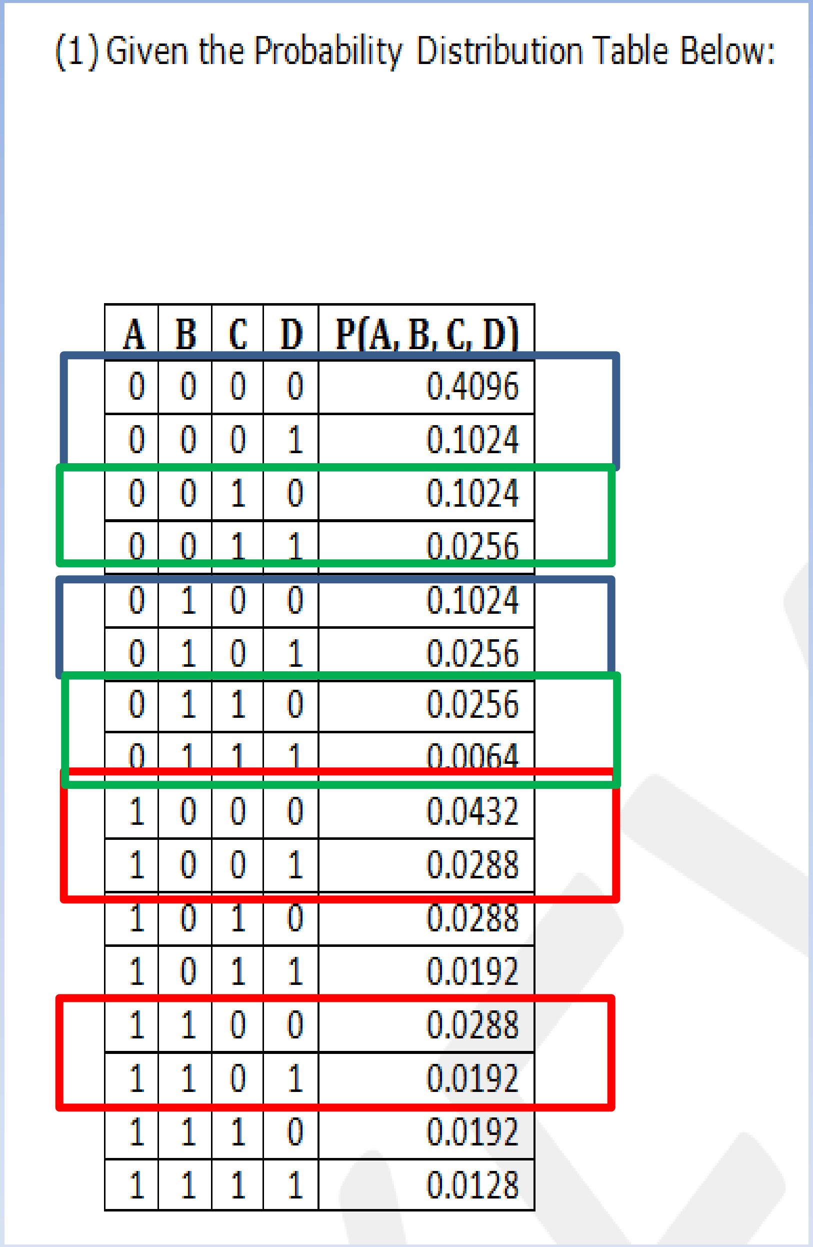
- P(A=0, B=0) = 0.64
- P(A=0, B=1) = 0.16
- P(A=1, B=0) = 0.12
- P(A=1, B=1) = 0.08



(1) Given the Probability Distribution Table Below:

A	B	C	D	P(A, B, C, D)
0	0	0	0	0.4096
0	0	0	1	0.1024
0	0	1	0	0.1024
0	0	1	1	0.0256
0	1	0	0	0.1024
0	1	0	1	0.0256
0	1	1	0	0.0256
0	1	1	1	0.0064
1	0	0	0	0.0432
1	0	0	1	0.0288
1	0	1	0	0.0288
1	0	1	1	0.0192
1	1	0	0	0.0288
1	1	0	1	0.0192
1	1	1	0	0.0192
1	1	1	1	0.0128

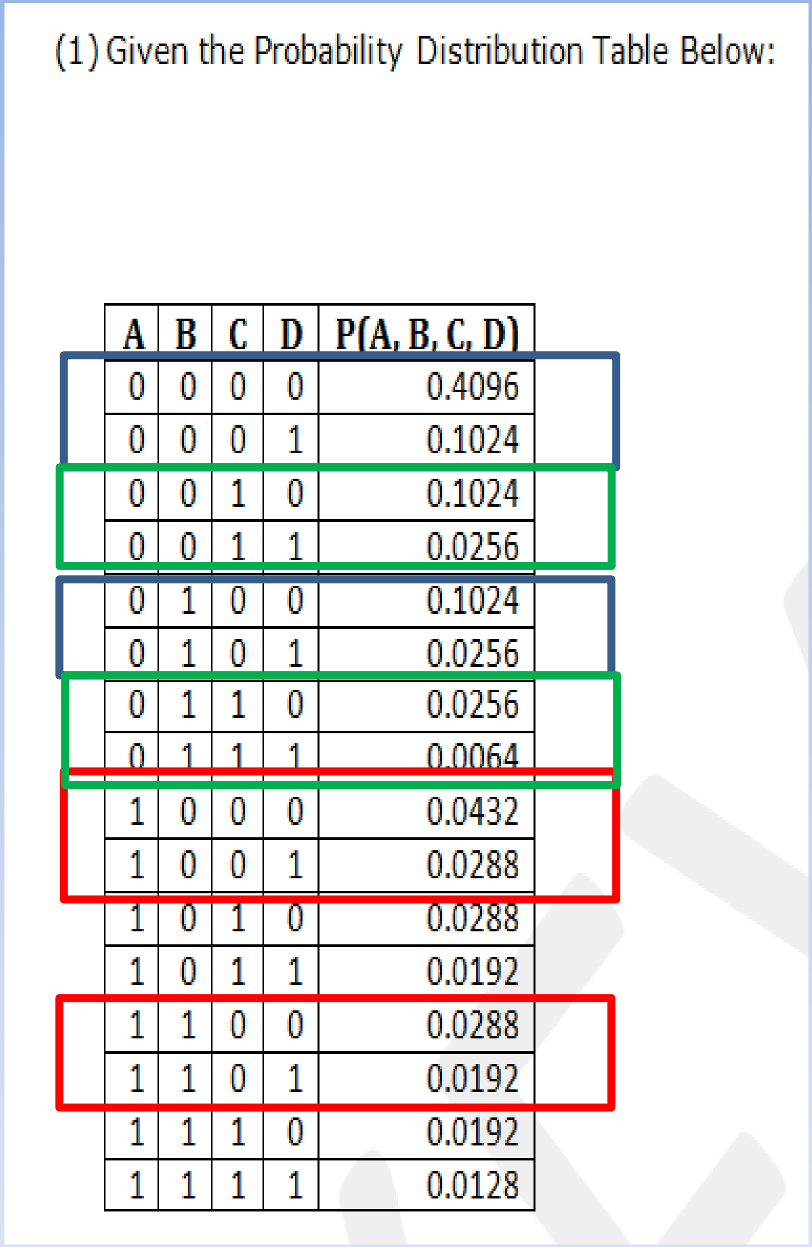
- $P(B=0|A=0) = 0.64/0.8=0.8$
- $P(B=1|A=0) = 0.16/0.8=0.2$
- $P(B=0|A=1) = 0.12/0.2=0.6$
- $P(B=1|A=1) = 0.08/0.2=0.4$



(1) Given the Probability Distribution Table Below:

A	B	C	D	P(A, B, C, D)
0	0	0	0	0.4096
0	0	0	1	0.1024
0	0	1	0	0.1024
0	0	1	1	0.0256
0	1	0	0	0.1024
0	1	0	1	0.0256
0	1	1	0	0.0256
0	1	1	1	0.0064
1	0	0	0	0.0432
1	0	0	1	0.0288
1	0	1	0	0.0288
1	0	1	1	0.0192
1	1	0	0	0.0288
1	1	0	1	0.0192
1	1	1	0	0.0192
1	1	1	1	0.0128

- P(A=0, C=0) = 0.64
- P(A=0, C=1) = 0.16
- P(A=1, C=0) = 0.12
- P(A=1, C=1) = 0.08



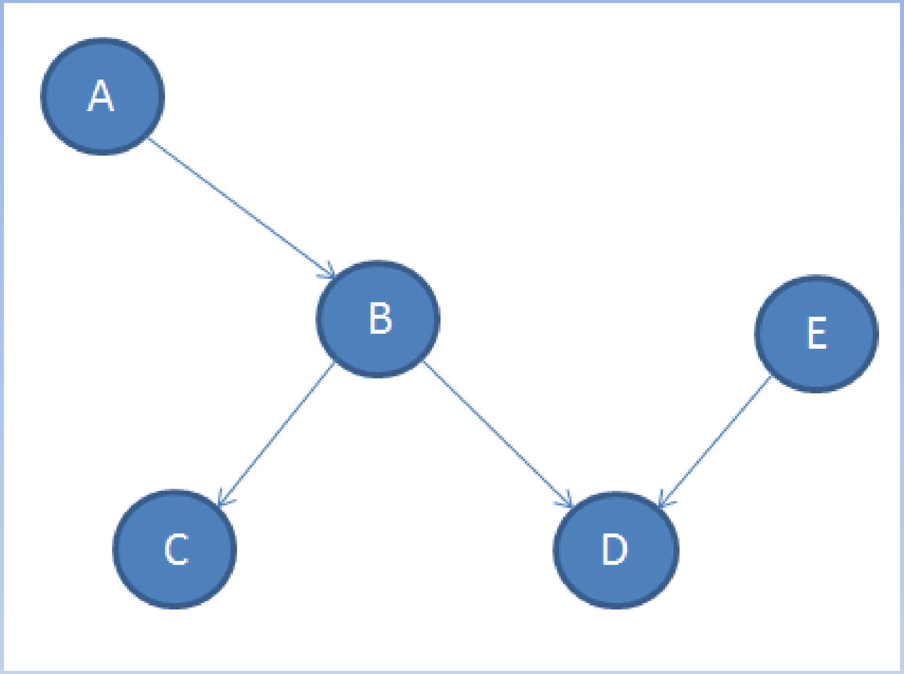
(1) Given the Probability Distribution Table Below:

A	B	C	D	P(A, B, C, D)
0	0	0	0	0.4096
0	0	0	1	0.1024
0	0	1	0	0.1024
0	0	1	1	0.0256
0	1	0	0	0.1024
0	1	0	1	0.0256
0	1	1	0	0.0256
0	1	1	1	0.0064
1	0	0	0	0.0432
1	0	0	1	0.0288
1	0	1	0	0.0288
1	0	1	1	0.0192
1	1	0	0	0.0288
1	1	0	1	0.0192
1	1	1	0	0.0192
1	1	1	1	0.0128

- $P(C=0|A=0) = 0.64/0.8=0.8$
- $P(C=1|A=0) = 0.16/0.8=0.2$
- $P(C=0|A=1) = 0.12/0.2=0.6$
- $P(C=1|A=1) = 0.08/0.2=0.4$

Probabilistic Reasoning

- Bayesian Networks
  - Chain Rule for Bayesian Networks
  - Conditional Independencies



Question:

- Given the Bayes Net above:
  - What would be the set of Conditional Probability Distributions needed to represent the full joint probability distribution over A, B, C, D, E?
  - Show the formula for computing the full joint probability distribution over all the variables with the Conditional Probability Distributions.