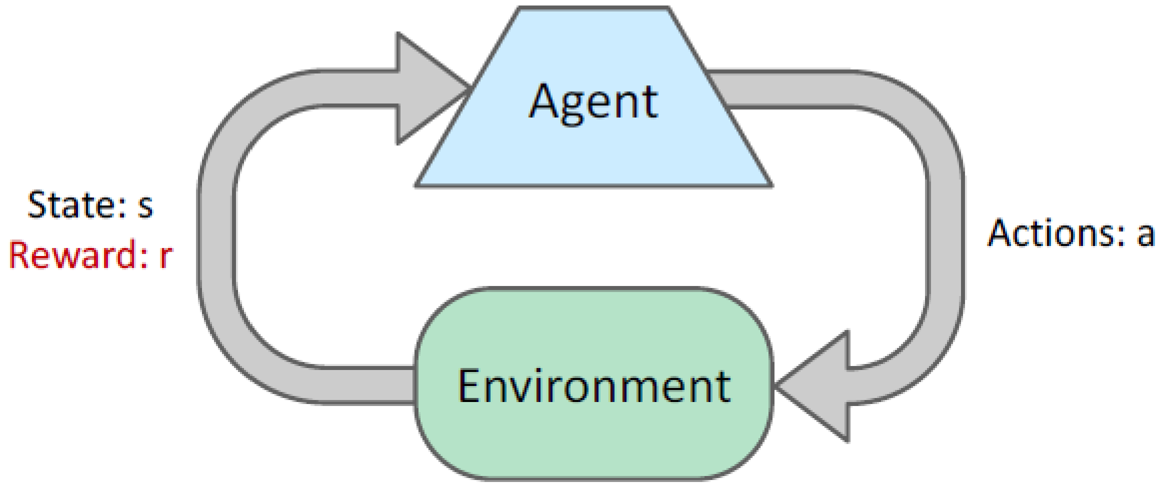
Reinforcement Learning

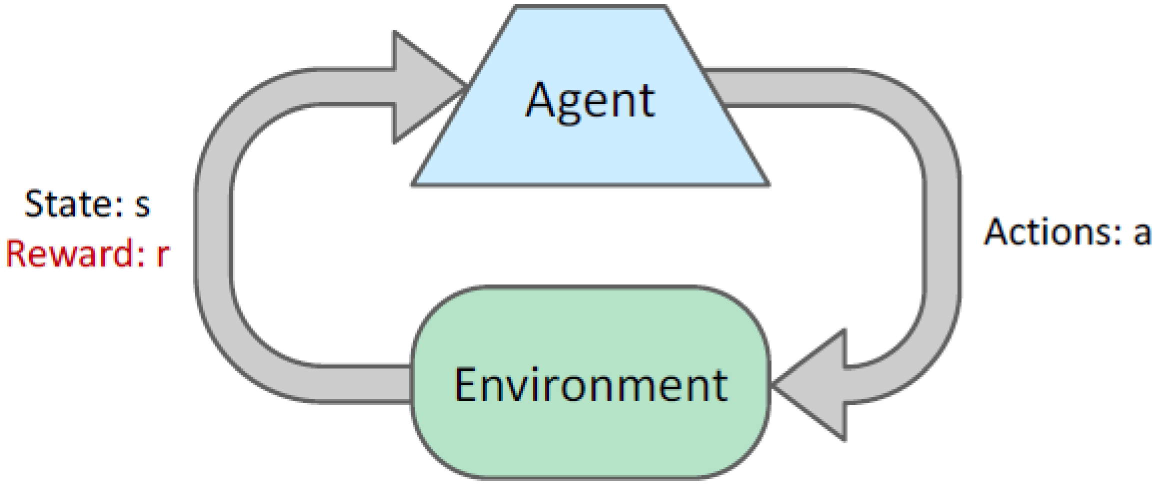
Reinforcement Learning

- Basic idea:
  - 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!.



Reinforcement Learning

- The agent needs to know that something good has happened and that something bad has happened as a result of its action.
- This kind of feedback is called a **reward**, or **reinforcement**.



Reinforcement Learning

- Assumes 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)
  - π(s) is the action recommended by the policy π for state s.
  - An optimal policy is a policy that yields the highest expected utility.
  - The expected utility of an action given the evidence, EU(a|e), is just the average utility value of the outcomes, weighted by the probability that the outcome occurs
- But we don't know T or R
  - i.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn
- The **task of reinforcement learning** is to use observed rewards to learn an optimal (or nearly optimal) policy for the environment.

Reinforcement Learning: Model-Based Learning

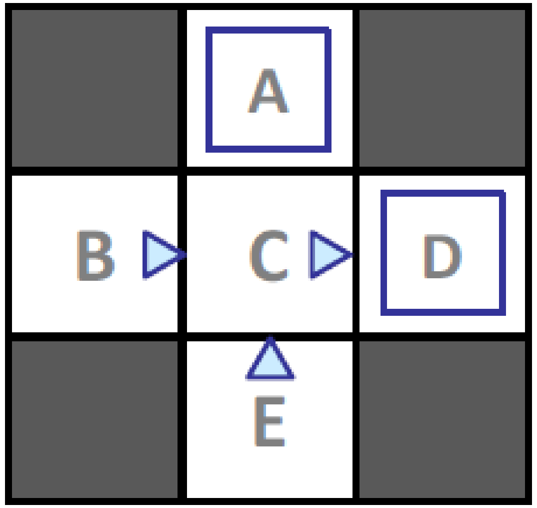
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- 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

Reinforcement Learning:
Model-Based Learning: Example

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

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
...

Passive and Active Learning

- A **passive learner** simply watches the world going by, and tries to learn the utility of being in various states.
  - In passive learning, the agent's policy is fixed and the task is to learn the utilities of states (or state–action pairs).
  - This could also involve learning a model of the environment.

- An **active learner** must also act using the learned information, and can use its problem generator to suggest explorations of unknown portions of the environment.
  - In active learning, the agent must also learn what to do.
  - The principal issue is **exploration**: an agent must experience as much as possible of its environment in order to learn how to behave in it.

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:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience

Passive Reinforcement Learning

- **Passive Reinforcement Learning:**
  - Agent's policy π is fixed;
  - Learn state utility values U(s) without knowing the transition model P(s'|s,a) or the reward function R(s)
  - In policy iteration, we have learned how to evaluate a policy, i.e., compute U(s) given P(s'|s,a) and R(s)

- Two basic approaches:
  - **Model-based:** Build a model of R(s), P(s'|s,a) then evaluate policy
  - **Model-free**: Directly evaluate without building a model

Model-Based Passive Reinforcement Learning

Problem Formulation:

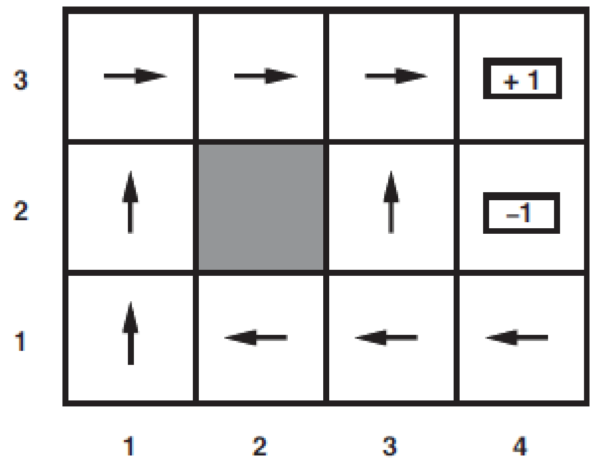
- We are given a policy, but we don’t know the details of the environment
- Follow policy π, perform many trials/experiments to get sample sequences
- Estimate MDP model parameters R(s) and P(s’|s,a) given observed transitions and rewards
- If finite set of states and actions, can just count and average counts
- Use estimated MDP to evaluate policy

Trail:

- The agent executes a set of trials in the environment using its policy π.
- In each trial, the agent starts in the starting state and experiences a sequence of state transitions until it reaches one of the terminal states.
- Its percepts supply both the **current state** and the **reward** received in that state.

Model-Based Passive Reinforcement Learning: Example

- Start at s=(1,1) action a=up, based on π
  - reward = -0.04; end up at s' = (1,2)
- s=(1,2) action a=up, based on π
  - reward = -0.04; end up at s' = (1,2)
- s=(1,2) action a=up, based on π
  - reward = -0.04; end up at s' = (1,3)
- s=(1,3) action a=right, based on π
  - reward = -0.04; end up at s' = (2,3)
- s=(2,3) action a=right, based on π
  - reward = -0.04; end up at s' = (3,3)
- s=(3,3) action a=right, based on π
  - reward = -0.04; end up at s' = (4,3)
- s = (4, 2); no action available
  - reward = 1.00; terminate



Estimate P(s'|s,a): P((1,3)|(1,2), up) = 1/2 = 0.5

Estimate R((1,2)) = -0.04

Model-Based Passive Reinforcement Learning: Example

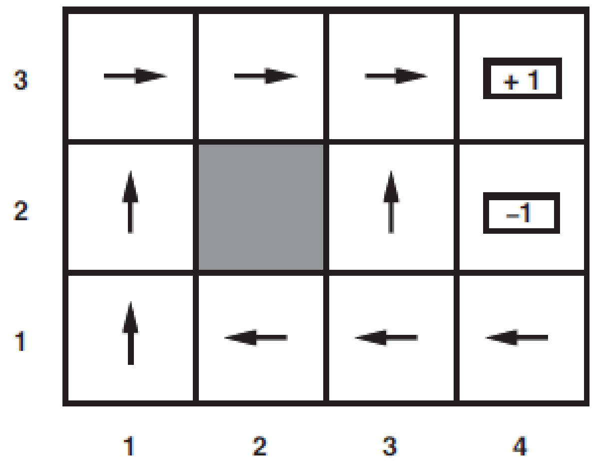
- We can run more trails:

Trail 1: (1,1)→(1,2)→(1,2)→(1,3)→(2,3)→(3,3)→(4,3)

Trail 2: (1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(4,3)

Trail 3: (1,1)→(1,2)→(1,3)→(2,3)→(2,3)→(3,3)→(4,3)

Trail 4: (1,1)→(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(4,3)



Estimate P(s'|s,a): P((1,3)|(1,2), up) = 4/5 = 0.80

Estimate R((1,2)) = -0.04

Model-Based Passive Reinforcement Learning

- Empirical estimate of transition probability P:

$$P(s'|s,a) = \frac{\#(s,a,s')}{\#(s,a)}$$

- Empirical estimate of rewards R:

$$R(s) = \frac{\sum_s R(s)}{\#(s)}$$

- Given estimates of P and R, we can do MDP policy evaluation:

$$U(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi(s))U(s')$$

Model-Based Passive Reinforcement Learning

Advantage:

- Makes good use of data you have

Disadvantage:

- Require building the actual MDP model, which can be intractable if state space is too large

Model-Free Passive Reinforcement Learning

Strategy:

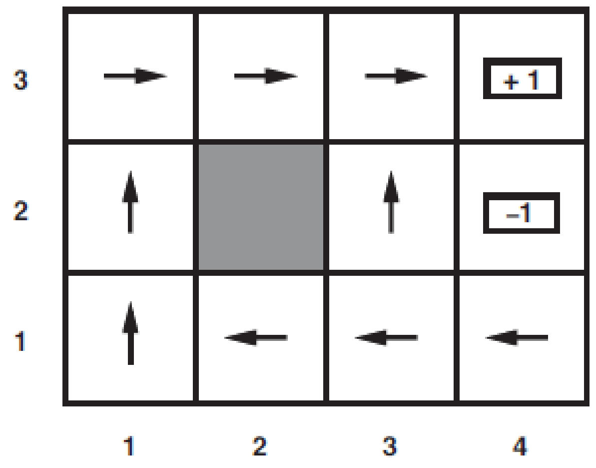
- evaluate policy directly, without first estimating P and R

Direct utility estimation:

- Calculate expected total reward from that state onward.
- When a trial hits a state, view it as a sample of the total reward from that state onward.

Model-Free Passive Reinforcement Learning
Direct Utility Estimation: Example

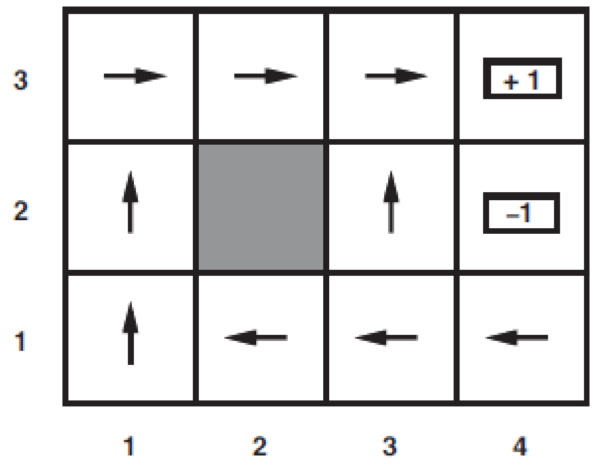
- Start at s=(1,1) action a=up, based on π
  - reward = -0.04; end up at s' = (1,2)
- s=(1,2) action a=up, based on π
  - reward = -0.04; end up at s' = (1,2)
- s=(1,2) action a=up, based on π
  - reward = -0.04; end up at s' = (1,3)
- s=(1,3) action a=right, based on π
  - reward = -0.04; end up at s' = (2,3)
- s=(2,3) action a=right, based on π
  - reward = -0.04; end up at s' = (3,3)
- s=(3,3) action a=right, based on π
  - reward = -0.04; end up at s' = (4,3)
- s = (4, 2); no action available
  - reward = 1.00; terminate



Estimate $U^{\pi}(1,2) = (0.84+0.80)/2 = 0.82$

Model-Free Passive Reinforcement Learning
Direct Utility Estimation: Example

- We can run more trails:



Trail 1: (1,1)→(1,2)→(1,2)→(1,3)→(2,3)→(3,3)→(4,3)

Trail 2: (1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(4,3)

Trail 3: (1,1)→(1,2)→(1,3)→(2,3)→(2,3)→(3,3)→(4,3)

Trail 4: (1,1)→(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(4,3)

Estimate $U^{\pi}(1,2) = (0.84+0.80+0.84+0.80+0.84)/5 = 0.824$

Model-Free Passive Reinforcement Learning
Direct Utility Estimation

Advantage:

- estimates utility of policy without having to calculate P and S

Disadvantage:

- Need to wait until you reach terminal state.
- Estimates U(s) and U(s') separately, ignoring their relationship:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P\big(s' \,\big|\, s, \pi(s)\big) U^{\pi}(s')$$

- Converges very slowly

Model-Free Passive Reinforcement Learning Temporal-Difference (TD) Learning

- Key idea: Do not wait until the trial terminates, update after each state transition using a running average

- More likely outcomes will contribute to the update more often

- Does not need a transition model, only experience

Model-Free Passive Reinforcement Learning
Temporal-Difference (TD) Learning

- Updates performed using exponential moving average:

$$U_{i+1}(s) \leftarrow (1-\alpha)U_i + \alpha(R(s) + \gamma U_i(s'))$$

- Rearranging, we get:

$$U_{i+1}(s) \leftarrow U_i(s) + \alpha(R(s) + \gamma U_i(s') - U_i(s))$$

- Without subscripts, we get the general update:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

Model-Free Passive Reinforcement Learning
Temporal-Difference (TD) Learning Equation

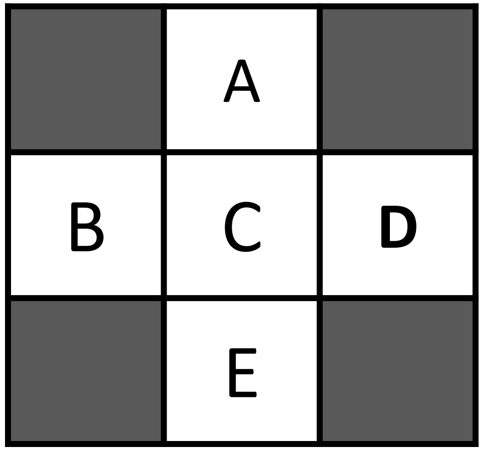
$$\underbrace{U^\pi(s)}_{\text{estimated utility of } s} \leftarrow U^\pi(s) + \overset{\substack{\text{learning}\\\text{rate}\\\downarrow}}{\alpha}\left(R(s) + \overset{\substack{\text{discount}\\\text{rate}\\\downarrow}}{\gamma}\underbrace{U^\pi(s')}_{\text{estimated utility of } s'} - U^\pi(s)\right)$$

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha\underbrace{(\overbrace{R(s) + \gamma U^\pi(s')}^{\text{what we observed}} - \overbrace{U^\pi(s)}^{\text{what we predicted}})}_{\text{error}}$$

$$U^\pi(s) \leftarrow U^\pi(s) + \underbrace{\alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))}_{\text{update}}$$

Example: Temporal Difference Learning

States

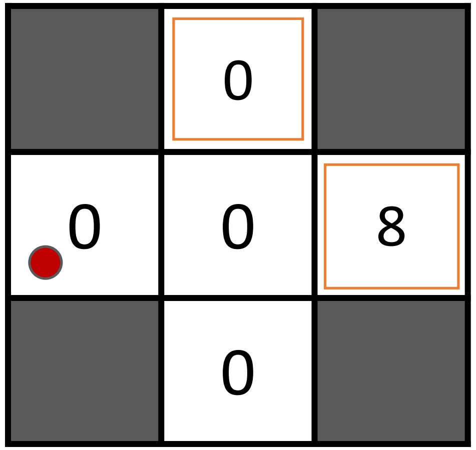


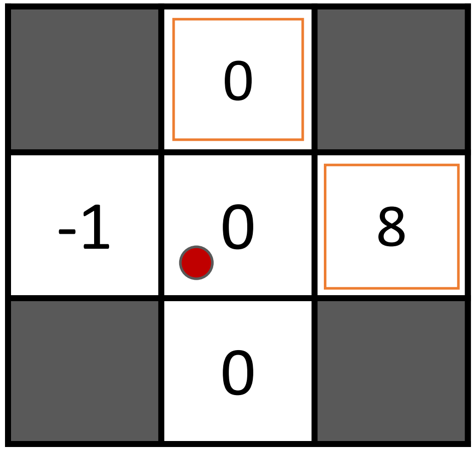
Assume: $\gamma = 1$, $\alpha = 1/2$

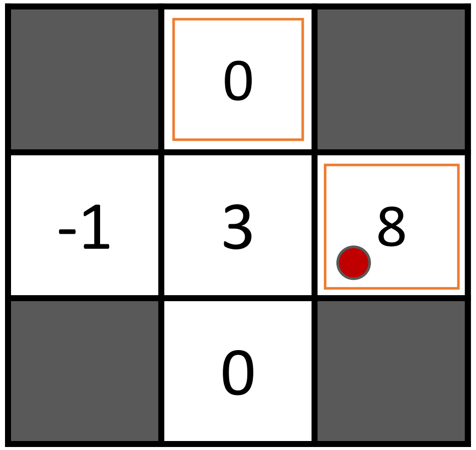
Observed Transitions

B, east, C, -2

C, east, D, -2







$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \underbrace{\alpha\left(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)\right)}_{\text{update}}$$

Active Reinforcement Learning

- Passive agents follow a fixed policy, estimate expected utilities

- Active agents need to decide on what actions to perform to maximize expected utility

- Passive agents face a prediction problem, while active ones face a control problem

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
  - You actually take actions in the world and find out what happens…

Active Reinforcement Learning
Action-Utility Function

- TD-learning learns the utility of states $U^{\pi}(s)$ for one action

- Without a policy, we need to learn about all the actions,

  Q(s, a): The expected value of taking action **a** in state **s**

- We can use Q values instead of U values

$$U(s) = \max_a Q(s, a)$$
$$\pi(s) = \operatorname*{argmax}_a Q(s, a)$$

Active Reinforcement Learning
Q-Values

- Recall Bellman Equation given policy π :

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) U^{\pi}(s')$$

- For optimal policy π*:

$$U^{*}(s) = R(s) + \gamma \max_{\alpha \in A(s)} \left[\sum P(s' | s, a) U^{*}(s') \right]$$

- Q-values are similar to value function, but defined on state-action pair rather than just states.

- $Q^{\pi}(s, a)$ is the expected total reward from state s onward if taking action a in state s, following policy π after

Active Reinforcement Learning
Q-Values

- We can express the Q-value of a given state-action pair in terms of the Q-value of its neighbors.

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^\pi(s')$$

$$Q^\pi(s, a) = R(s) + \gamma \sum_{s'} P(s' \mid s, a) U^\pi(s')$$

$$U^\pi(s) = Q^\pi(s, \pi(s))$$

$$Q^\pi(s, a) = R(s) + \gamma \sum_{s'} P(s' \mid s, a) Q^\pi(s', \pi(s'))$$

Active Reinforcement Learning
Optimal Q-Values

- When using optimal policy π *, we will take the action that leads to maximum total utility at each state

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s, a)$$

- Thus:

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

$$\begin{aligned} Q^*(s, a) &= R(s) + \gamma \sum_{s'} P(s' \mid s, a) U^*(s') \\ &= R(s) + \gamma \sum_{s'} P(s' \mid s, a) \max_a Q^*(s', a') \end{aligned}$$

Active Reinforcement Learning
Q-Learning

- Similar to Value Iteration, TD Learning
- Use information at s' to update the estimated Q-value at (s,a) through update:

$$\hat{Q}^*(s,a) \leftarrow (1-\alpha)\hat{Q}^*(s,a) + \alpha(r + \gamma \max_{a'} \hat{Q}^*(s',a'))$$

- Given estimated value of Q*(s,a), we can derive an estimate of the optimal policy

$$\hat{\pi}^*(s) = \operatorname{argmax}_{a} \hat{Q}^*(s,a)$$

Q-Learning

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

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

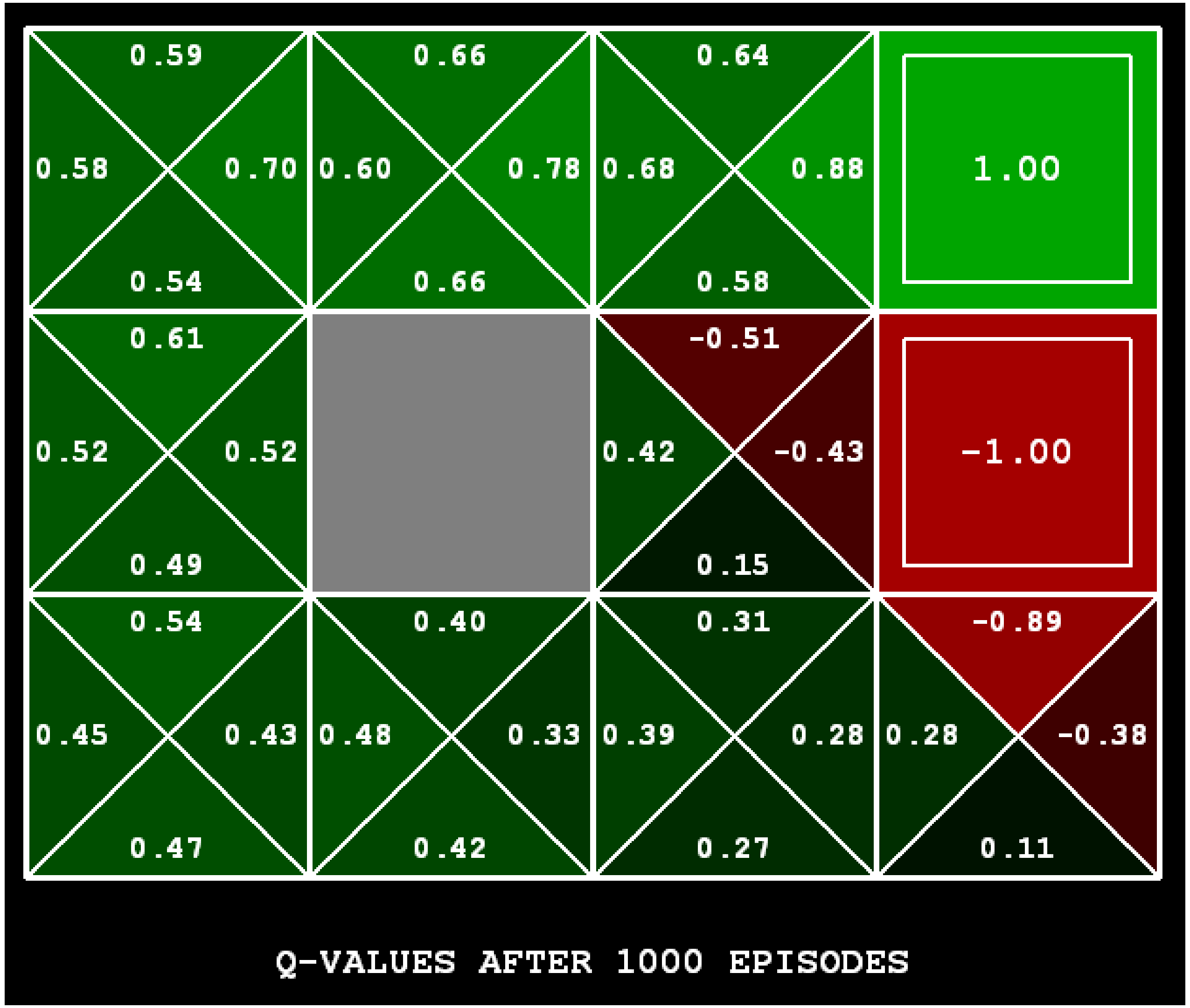
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

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

Q-Learning


0.59
0.58
0.70
0.54
0.66
0.60
0.78
0.66
0.64
0.68
0.88
0.58
1.00
0.61
0.52
0.52
0.49
-0.51
0.42
-0.43
0.15
-1.00
0.54
0.45
0.43
0.47
0.40
0.48
0.33
0.42
0.31
0.39
0.28
0.27
-0.89
0.28
-0.38
0.11
Q-VALUES AFTER 1000 EPISODES

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 (!)

Approximate Q-Learning

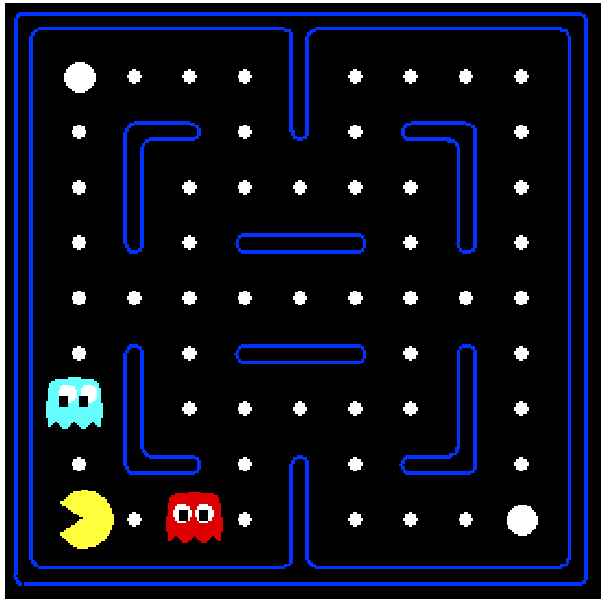
- Basic Q-Learning keeps a table of all q-values

- 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

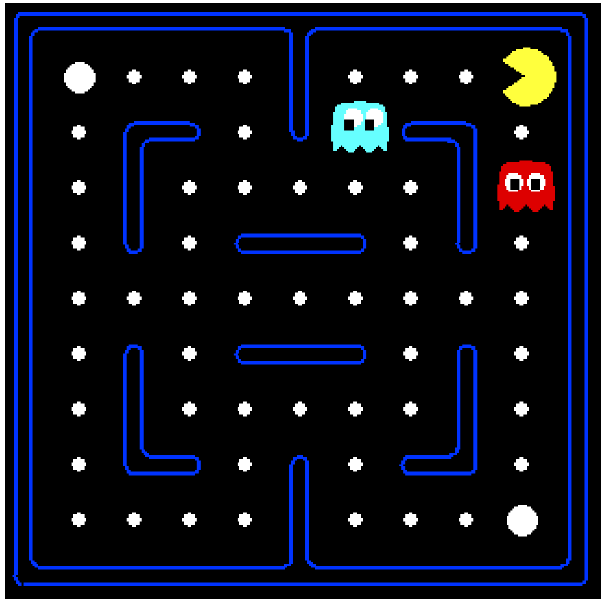
- Instead, we want to generalize:
    - Learn about some small number of training states from experience
    - Generalize that experience to new, similar situations
    - This is a fundamental idea in machine learning, and we’ll see it over and over again

Approximate Q-Learning - Example: Pacman

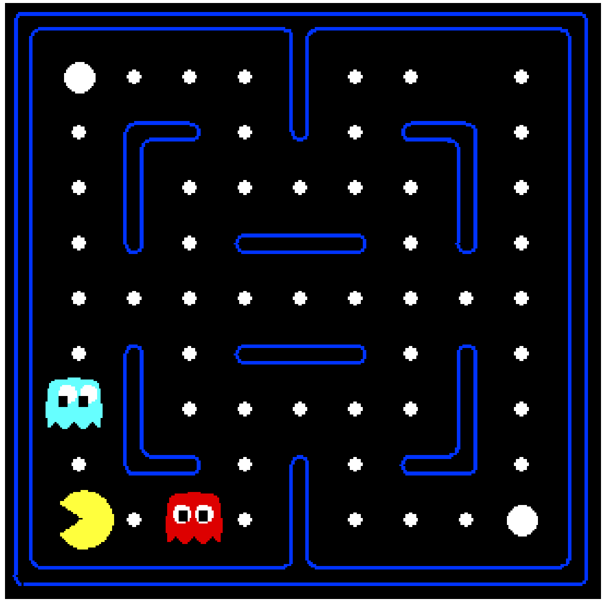
Let's say we discover through experience that this state is bad:



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

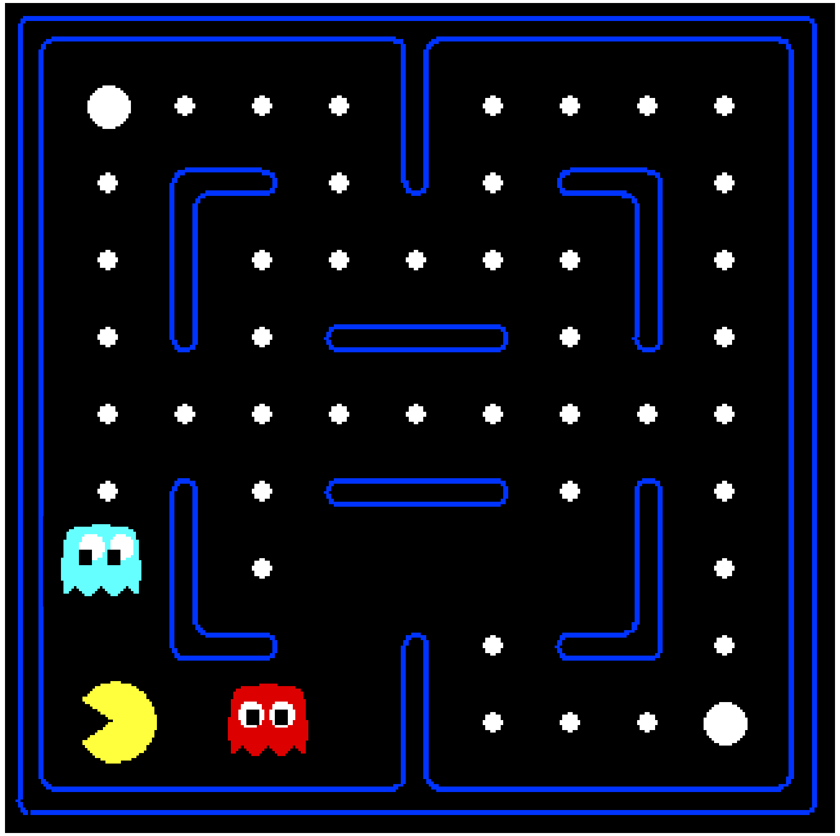


Or even this one!



Approximate Q-Learning - 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)



Approximate Q-Learning - 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