COL333/671: Introduction to AI

Semester I, 2021

Reinforcement Learning

Rohan Paul

Outline

- Last Class
  - Markov Decision Processes
- This Class
  - Reinforcement Learning
- Reference Material
  - Please follow the notes as the primary reference on this topic. Supplementary reading on topics covered in class from AIMA Ch 21 sections 21.1 – 21.4.

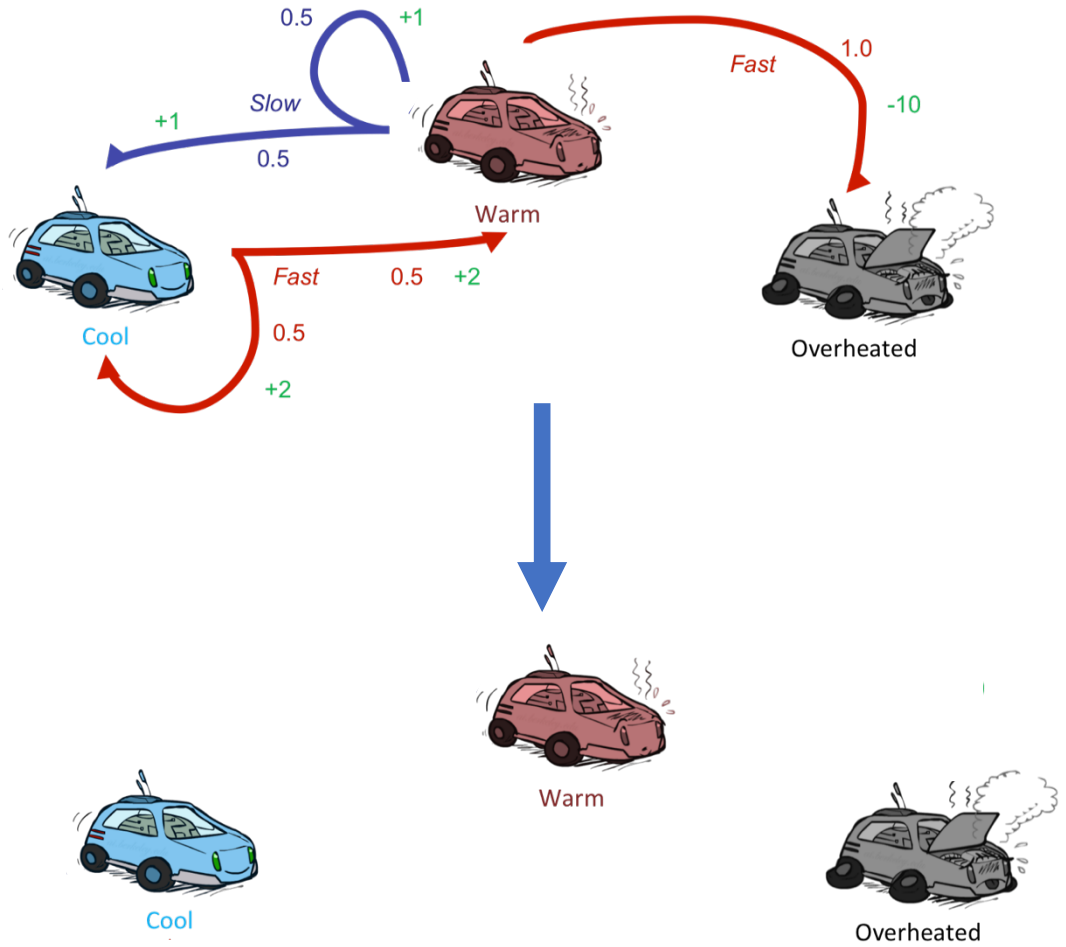
Acknowledgement

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Parag, Emma Brunskill, Alexander Amini, Dan Klein, Anca Dragan, Nicholas Roy and others.

Reinforcement Learning Setup

- Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Goal is to determine a policy $\pi(s)$

- Don't know T or R
  - We don't know which states are good or what the actions do
  - Must actually try actions and states out to learn
  - Goal is still to act optimally.



Reinforcement Learning Setup

- Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Goal is to determine a policy $\pi(s)$

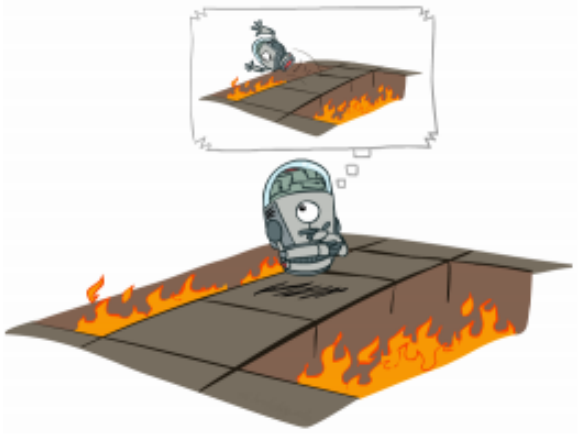
- But we don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

In the Car MDP (discussed previously) we now **do not** know the transition model and the rewards. How should the agent act in this setup to maximize expected future rewards.

How should the agent act in this setup so as to maximize expected future rewards. Also, think of the setup as acting in an **"unknown" MDP**

Another View: Offline (MDP) vs Online (RL)

- Offline (MDP)
  - We are given an MDP
  - We use policy or value iteration to learn a policy. This is computed offline.
  - At runtime the agent only executes the computed policy



Offline Solution

The agent solves the MDP and computes a policy. Now it simply acts with it.

- Online (RL)
  - We do not have full knowledge of the MDP
  - We must interact with the world to learn which states are good and which actions eventually lead to good rewards.

Online Learning

The agent interacts with the world and learns that one of the states is bad.

Reinforcement Learning

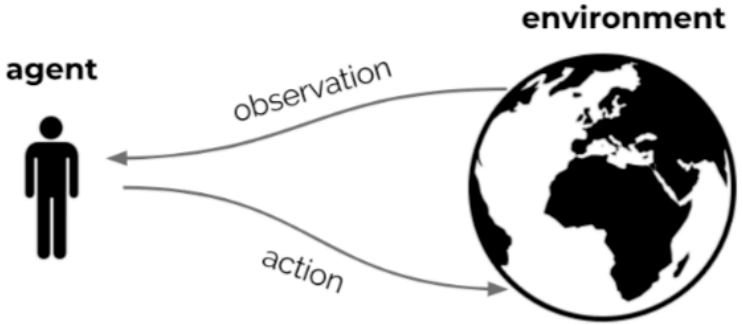
- **Given**
  - Agent, states, actions, immediate rewards and environment.
- **Not given**
  - Transition function (the agent cannot predict which state will it land in once it takes an action)
  - Does not know the reward function. Does not know what reward it will get in a state.
- **Agent's Task**
  - A policy that maximizes expected rewards (the objective has not changed)

Reinforcement Learning (RL)

- People and animals learn by interacting with our environment
  - It is active rather than passive.
  - Interactions are often sequential — future interactions can depend on earlier ones

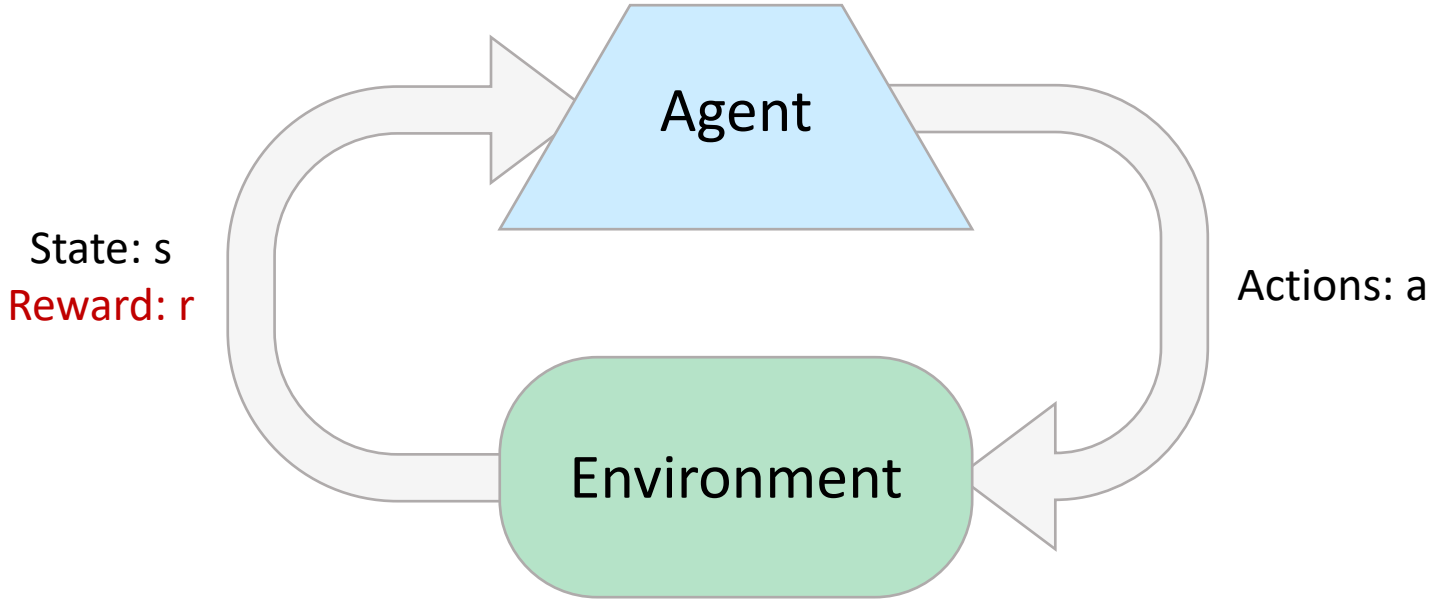
- Reward Hypothesis
  - Any goal can be formalized as the outcome of maximizing a cumulative reward.
  - We can learn without examples of optimal behaviour Instead, we optimise some reward signal

Biological motivation



Goal: optimise sum of rewards, through repeated interaction

Reinforcement Learning: Setup



Key characteristic of reinforcement learning

- **Only evaluative feedback present.**
- **The agent takes and action and is provided feedback (reward).**
- **The agent is not told which action it should take in a state.**

- 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!

Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$x \rightarrow y$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:


This thing is like the other thing.

Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

Apple example:

Eat this thing because it will keep you alive.

Figure: Alexander Amini

Examples of RL

▶ Fly a helicopter	→ **Reward**: air time, inverse distance, ...
▶ Manage an investment portfolio	→ **Reward**: gains, gains minus risk, ...
▶ Control a power station	→ **Reward**: efficiency, ...
▶ Make a robot walk	→ **Reward**: distance, speed, ...
▶ Play video or board games	→ **Reward**: win, maximise score, ...

Examples: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

Examples: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

Examples: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

Examples: Game Play

Google Deep Mind

Examples: Healthcare Domain

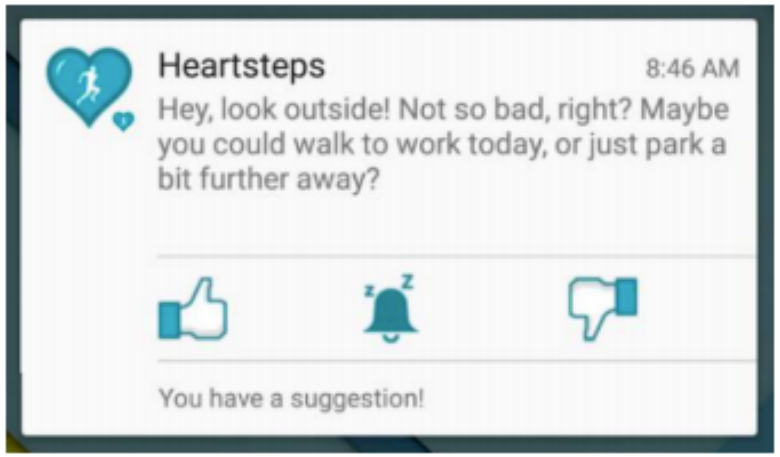
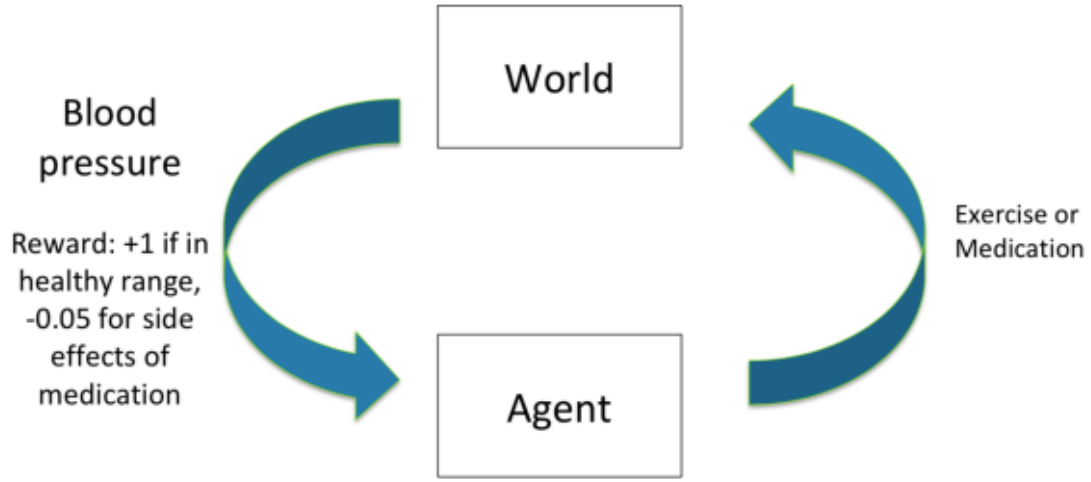


Figure: Personalized HeartSteps: A Reinforcement Learning Algorithm for Optimizing Physical Activity. Liao, Greenewald, Klasnja, Murphy 2019 arxiv



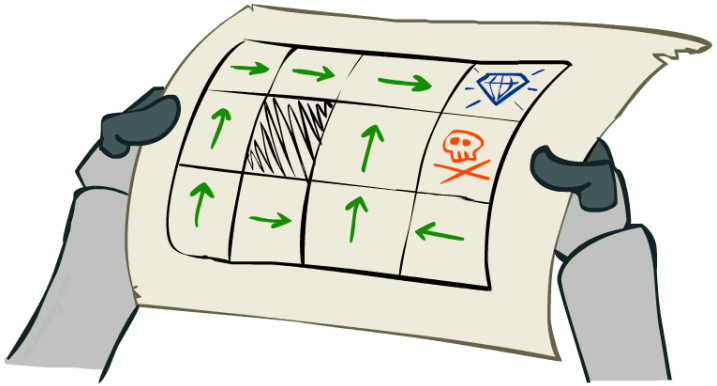
Blood pressure control

Reinforcement Learning: Approaches

- **Different learning agents**
  - **Utility-based agent**
    - Learn a value/utility function on states and use it to select actions that maximize the expected outcome utility.
  - **Q-learning**
    - Learn action-utility function (or Q-function) giving expected utility of taking a given action in a given state.
  - **Reflex agent.**
    - Directly learn a mapping from states to action (policy).
- **Approaches**
  - **Passive Learning**
    - The agent's policy is fixed. the task is to learn the utilities of states (or state-action pairs); this could involve learning a model of the environment.
    - It cannot select the actions during training.
  - **Active Learning**
    - The agent can select actions, it must also learn what actions to take.
    - Issue of exploration: an agent must experience as much as possible of the environment in order to learn how to behave in it.

Passive Reinforcement Learning

- Setup
  - Input: a fixed policy $\pi(s)$
  - Don't know the transitions T(s,a,s')
  - Don't know the rewards R(s,a,s')
  - The agent executes a set of trials or episodes using its policy $\pi(s)$
  - In each episode or trial it starts in a state and experiences a sequence of state transitions and rewards till it reaches a terminal state.
  - Goal: learn (estimate) the state values $V^{\pi}(s)$



The learner is provided with a policy (cannot change that), using the policy it executes trials or episodes in the world. The goal is to determine the value of states.

Model-Based Reinforcement Learning

- Model-Based Idea
  - Learn an approximate model (R(), T()) based on experiences
  - Compute the value function using the learned model (as if it were correct).

- Step 1: Learn an 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')

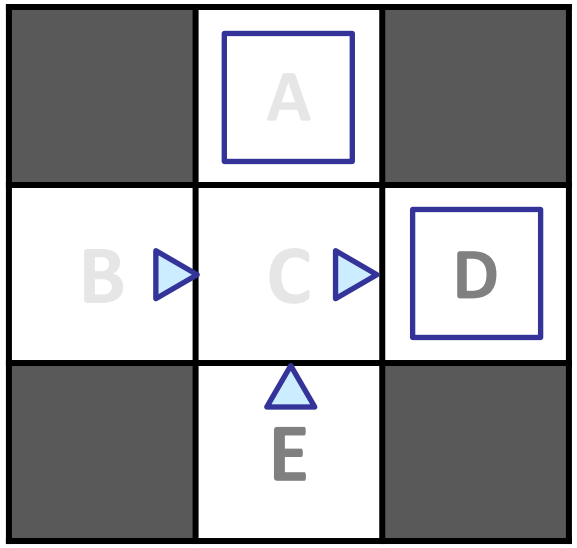
- Step 2: Solve the learned MDP
  - For example, use value iteration, to obtain the final policy.
  - Plug in the estimated T and R in the following equation:

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

*Note: going through an intermediate stage of learning the model. In contrast, "**model-free**" approaches do not learn the intermediate model.*

Example: Model-Based Learning

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
…

Toy example: Model-Based vs Model-Free

Goal: Compute expected age of COL333/671 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"

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

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

Eventually, we learn the right model which gives the right estimates.

Unknown P(A): "Model Free"

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

Bypass the model construction. Averaging works because the samples appear with the right frequencies.

Model-Based vs. Model Free RL

Model-based RL:

- Explore environment and learn model, T=P(**s**'|**s**,**a**) and R(**s**,**a**), (almost) everywhere. Use model to plan a policy (solving the MDP)
- Suitable when the state-space is manageable

Model-free RL:

- Do not learn a model; learn value function or policy directly
- Suitable when the state space is large

Next, we look at a model free approach to RL

Direct Evaluation of the Value Function $V^{\pi}(s)$

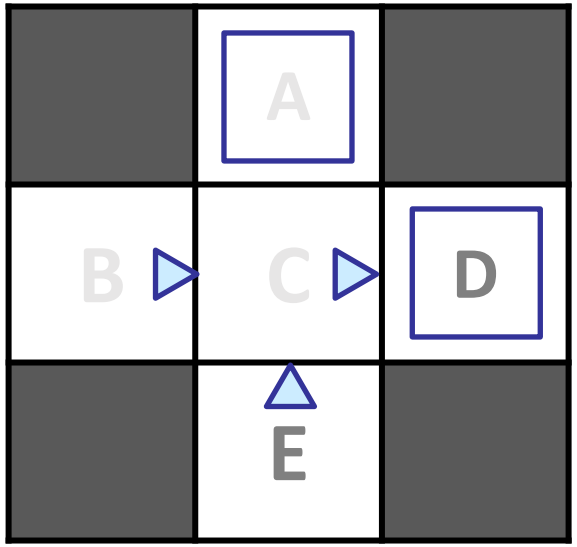
- Goal:
  - Compute values for each state under the policy π
  - Note: still doing passive RL (the agent does not decide the actions)

- Idea: Average together observed sample values
  - Act in the environment according to the given policy π
  - For a visited state, compute the sum of discounted rewards that were obtained
  - The above is one sample. Continue to get more samples
  - Average all the samples to get the estimate of the value/utility of the state

- A model free approach

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

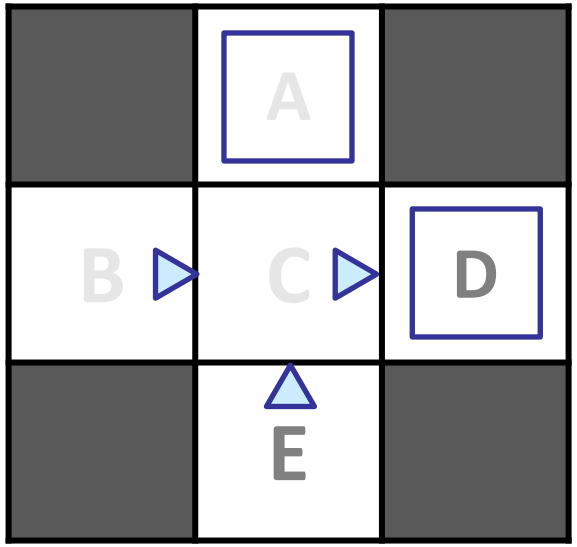
Episode 4

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

Value/utility of state c
$V^{\pi}(C) = ((9 + 9 + 9 + (-11))/4)$
$= 4$

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

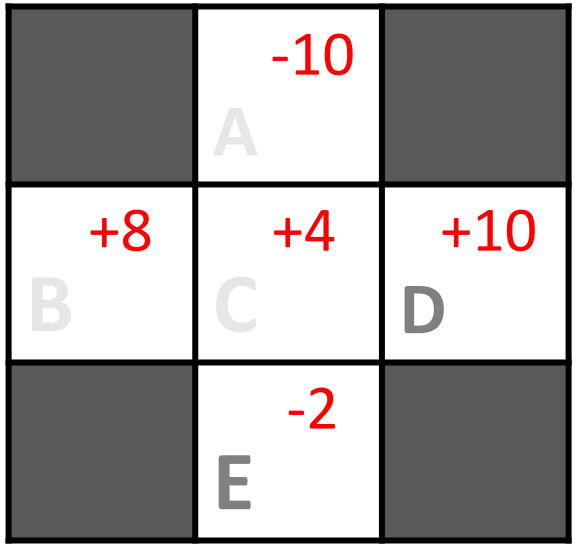
Episode 3

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

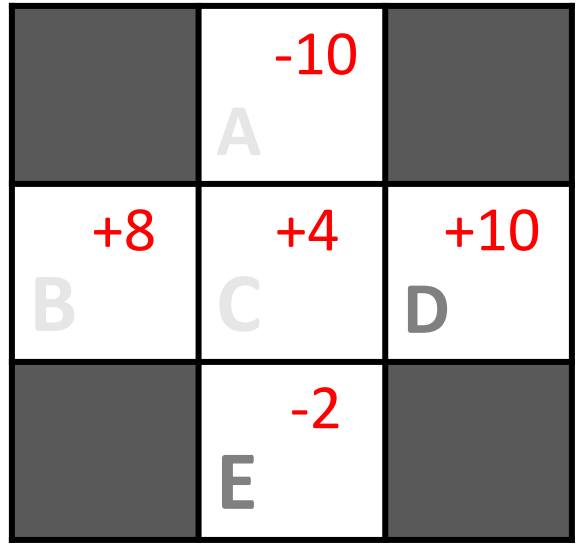
Output Values



- **Problem: we have lost the connection between states.**
- We go through state C, 4 times. We use only 2 estimates for E.

Direct Evaluation: Pros and Cons

- Pros
  - Does not estimate a model. It does not require any knowledge of T(), R().
  - With large number of runs it computes the correct average values, using just sample transitions
- Cons
  - Each state must be learned separately. Loses the state connection information.
  - Bellman characterization is not getting used here.
  - Hence, takes a long time to learn.



- **Problem: we have lost the connection between states.**
- If B and E both go to C under this policy, how can their values be different?

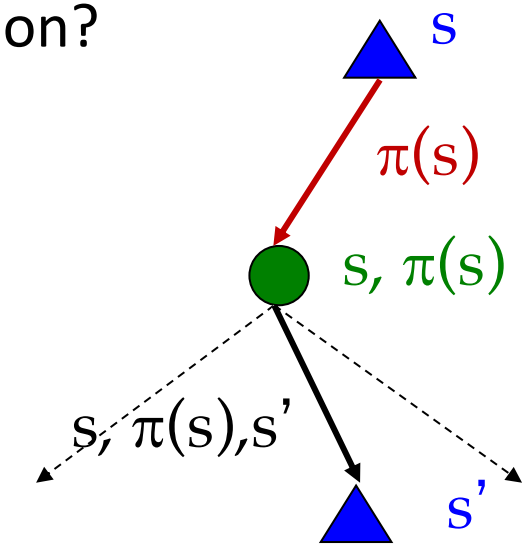
How can the Bellman equation be incorporated in this learning? Let us look at two ways.

1. Policy Evaluation

- We want to estimate the value V of states. Can we use Policy Evaluation?
- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V

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

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



  - This approach fully exploited the connections between the states
  - But this equation requires T and R.

- 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?

Sample-Based Policy Evaluation

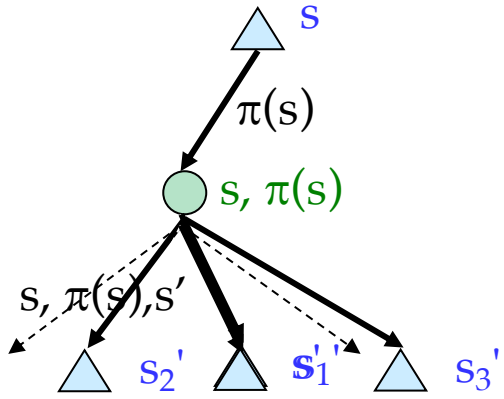
- We want to improve our estimate of V by computing these averages:

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

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

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

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

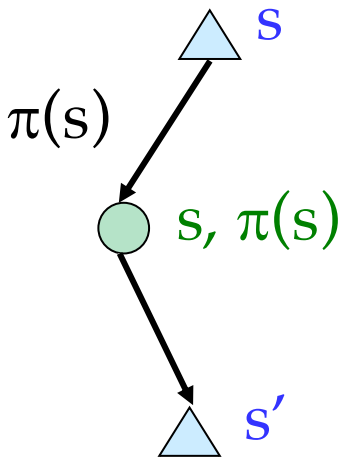


Assumes that we can rewind and start at state s again

2. Temporal Difference (TD) Learning

- Idea: learn from *every* experience (*no resetting* required)
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute to updates more often

- Temporal difference learning of values
  - Policy still fixed
  - Move values toward the value of the successor that is encountered
  - Keep a running average
  - Don't need to store all the experience to build models.



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

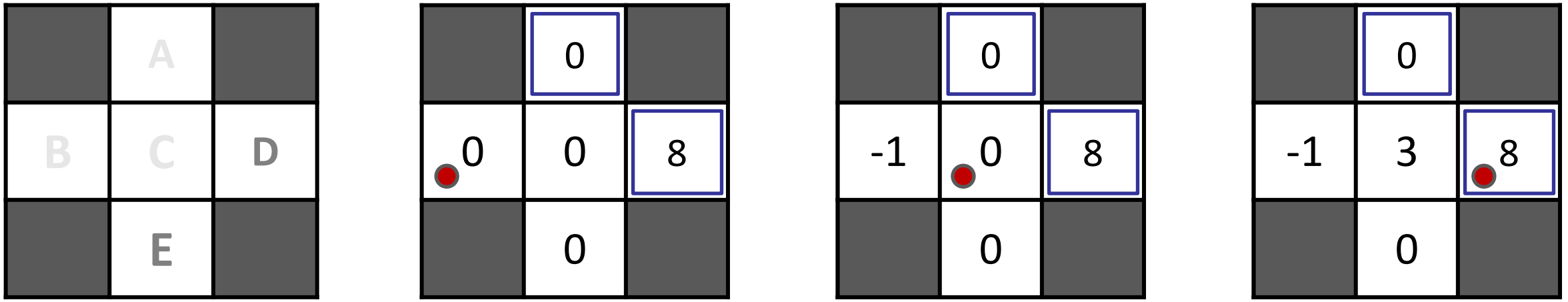
Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$ — Modify the old value

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$ — Sign and magnitude of the difference.

Temporal Difference Learning: Example

States

Observed Transitions



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

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

Recap from MDP Lecture: Optimal Utility and Optimal Policy

- **The value (utility) of a state s:**

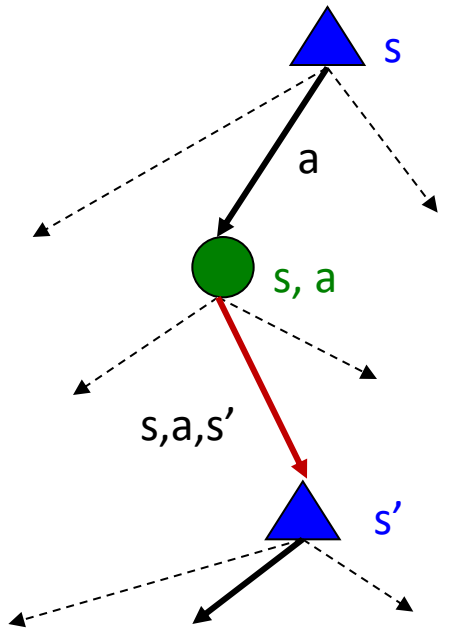
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- **The optimal policy:**

  $\pi^*(s)$ = optimal action from state s



s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

Recap from MDP Lecture: Value of States

- **Values of states are related to each other.**
- **Fundamental Operation**
  - Compute the expectimax value of the state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards

- **Recursive definition of value:**

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

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

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

TD Value Learning: Problems

- TD Value Learning
  - Model-free approach to perform policy evaluation
  - Incorporates Bellman updates with running sample averages
- Output of TD Value Learning
  - TD Value learning outputs the value function
- *How to turn the learned values into a new policy?*
  - Can use the following relationships:

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

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

- Problem: we don't have T and R
- Next: Can we learn the Q-values and not values? Then select actions for the new policy using the relationships above.

From Value Iteration to Q-Value Iteration

- **Value Iteration**
  - Start with $V_0(s) = 0$
  - Given V_k, calculate the V_{k+1} for all states as:

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

- **Q-Value Iteration**
  - Start with $Q_0(s,a) = 0$
  - Given Q_k, calculate the depth Q_{k+1} q-values for all q-states:

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

Q-Learning

- Sample-based Q-value iteration

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

- Estimate Q(s,a) values as:
  - Receive a sample (s,a,s',r)
  - Consider your old estimate:
  - Consider your new sample estimate
  - Incorporate the new estimate into a running average:

$$Q(s,a)$$

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

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

Q-Learning: Procedure

- **Forall s, a**
  - **Initialize Q(s, a) = 0**
- **Repeat Forever**

  Where are you? s

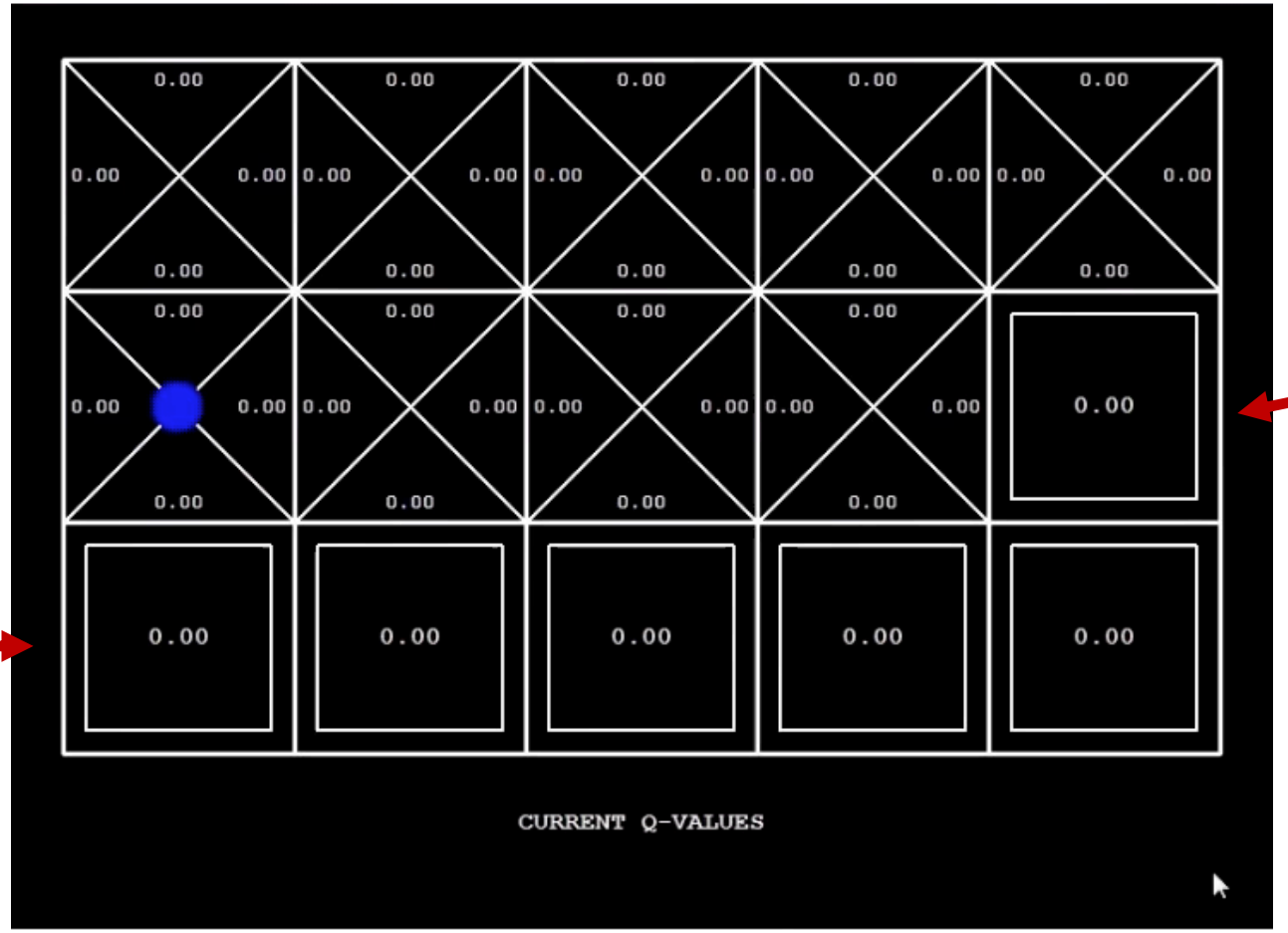
  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Q-Learning



Exit with +10 reward

Cliff at the bottom, negative reward here.

The agent does not know the rewards a-priori. Learns the effect of the east action over time. Only the actions we do are updated. Occasionally falls in the cliff and gets the negative reward. Note that the max of the Q values is propagated (green values) to other states as it is approximating the optimal Q value.

Q-Learning: Properties

- Off-policy learning
  - Q-learning converges to optimal policy -- even if the agent is acting *sub-optimally.*

- Some technical conditions:
  - Exploration is enough
  - In the limit does not matter how the actions are selected.

SARSA

- State-Action-Reward-State-Action (SARSA)
  - Update using (s, a, r, s', a')
- SARSA Update equation

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

- Note
  - SARSA waits until an action is taken and backs up the Q-value for that action. Learns the Q-function from actual transitions.
  - On-policy algorithm
  - More realistic if the policy is partly controlled by other agents. Learns from more realistic values.

Active Reinforcement Learning

- **Q-learning so far**
  - Q-learning allows action selection because we are learning the Q(s,a) function.
  - This means there is a policy that can be derived from the Q function learned.
  - Should the agent follow this policy exactly or should it explore at times?
- **Active RL**
  - The agent can select actions
  - Actions play two roles
    - A means to collect reward (exploitation)
    - **Help in acquiring a model of the environment (exploration)**
- **Exploration vs. Exploitation trade-off**
    - Act according to the current optimal (based on Q-Values)
    - Pick a different action to explore.
    - Example: a new tea stall opens in IIT (should you try the new one or stick to the old one? Goal is to maximize tea utility over time

Exploration Strategy

- **How to force exploration?**
- **An ε-greedy approach**
  - Every time step: either pick a random action or act on the current policy
  - With (small) probability **ε**, pick a random action
  - With (large) probability **1-ε**, act based on the current policy (based on the current Q –values in the table that the agent is updating)
- **What is the problem with ε-greedy?**
  - It takes a long time to explore.
  - Exploration is *not directed* towards states of which we have less information.

Exploration Functions

- How to **direct exploration** towards state-action pairs that are less explored?

- **Exploration function**
  - Trades off *exploitation* (preference for high values of u) vs. *exploration* (preference for actions that have not been tried out as yet).
  - f(u,n): Increasing in u and decreasing in n.

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

- **What is the exploration function trying to achieve?**
  - The exploration function provides an optimistic estimate of the best possible value attainable in any state.
  - Makes the agent think that there is high reward propagated from states that are explored less.

Exploratory Q-Learning

- **In Q-learning**
  - Explicitly encode the value of exploration in the Q-function
- **Exploratory Q-Learning**

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

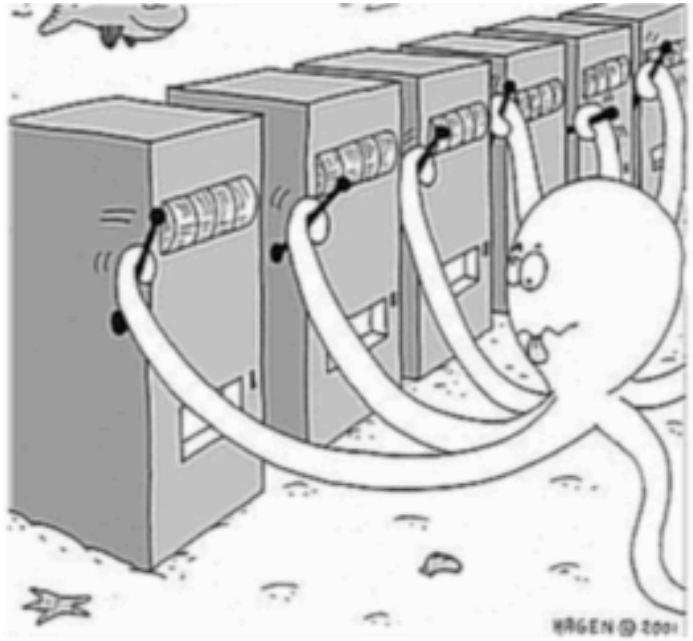
Modified update

- **Effects**
  - The lower the N(s', a') is the higher is the exploration bonus.
  - The exploration bonus makes those states favorable which lead to unknown states (propagation).
  - Will have a cascading effect when an exploration action is there.

Multi-arm Bandits

Multi-armed bandits are equivalent to a one state MDP. Goal is to learn a policy that picks actions such that in the expected rewards are maximized.

- Multi-armed bandit is a tuple of $(\mathcal{A}, \mathcal{R})$
- $\mathcal{A}$: known set of m actions (arms)
- $\mathcal{R}^a(r) = \mathbb{P}[r \mid a]$ is an unknown probability distribution over rewards
- At each step t the agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
- Goal: Maximize cumulative reward $\sum_{\tau=1}^{t} r_\tau$



Source: Emma Brunskill (CS232 Course) Lecture 10

Toy Example: *Treatment planning*

- Consider deciding how to best treat patients with broken toes
- Imagine have 3 possible options:
  - Do Surgery
  - Perform buddy taping the broken toe with another toe,
  - Do Nothing
- Outcome measure / reward is binary variable: whether the toe has healed (reward +1) or not healed (reward 0) after 6 weeks, as assessed by x-ray

- We can model this problem as a multi-arm bandit problem with 3 arms
- Imagine true (unknown) Bernoulli reward parameters for each arm (actions) are
  - surgery: $Q(a^1) = \theta_1 = .95$
  - buddy taping: $Q(a^2) = \theta_2 = .9$
  - doing nothing: $Q(a^3) = \theta_3 = .1$

Source: Emma Brunskill (CS232 Course) Lecture 10

Toy Example: *Treatment Planning*

- We consider algorithms that estimate $\hat{Q}_t(a) \approx Q(a) = \mathbb{E}[R(a)]$
- Estimate the value of each action by Monte-Carlo evaluation

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{t=1}^{T} r_t \mathbb{1}(a_t = a)$$

Avg. the rewards for each action.

- The **greedy** algorithm selects the action with highest value

$$a_t^* = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a)$$

- The ϵ-**greedy** algorithm proceeds as follows:
  - With probability $1 - \epsilon$ select $a_t = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a)$
  - With probability ϵ select a random action

Exploration vs. Exploitation trade off!
If the reward variance is high then the epsilon-greedy approach works better than simply greedy.

Source: Emma Brunskill (CS232 Course)

Upper Confidence Bound

- Greedy actions are those that look best at the present, but some of the other actions may be better.
- Epsilon-greedy forces the non-greedy actions to be tried.
  - Indiscriminately, with no preference for those actions that are nearly greedy or particularly uncertain.
- It would be better to select among the non-greedy actions according to their potential for being optimal.
  - Take into account how close their estimates are to being maximal and the uncertainties in their estimates.

Upper Confidence Bound

- Upper Confidence Bound (UCB) for action selection
  - Square-root term is a measure of uncertainty or variance in the estimate of the action's value.
    - When a is selected then $N_t(a)$ is incremented. The uncertainty reduces as denominator increases.
    - Each time when a is not selected then t increases but $N_t(a)$ does not. The uncertainty estimate increases (numerator).
  - Natural logarithm
    - Increases get smaller over time but are unbounded; all actions will eventually be selected.
    - Actions with lower value estimates will be selected with lower frequency.

$$a_t = \arg\max_{a \in \mathcal{A}} [\hat{Q}(a) + \sqrt{\frac{2 \log t}{N_t(a)}}]$$

Upper Confidence Bound

- True (unknown) parameters for each arm (action) are
  - surgery: $Q(a^1) = \theta_1 = .95$
  - buddy taping: $Q(a^2) = \theta_2 = .9$
  - doing nothing: $Q(a^3) = \theta_3 = .1$
- UCB1 (Auer, Cesa-Bianchi, Fischer 2002)
  1. Sample each arm once
     - Take action a^1 ($r \sim$Bernoulli(0.95)), get $+1$, $\hat{Q}(a^1) = 1$
     - Take action a^2 ($r \sim$Bernoulli(0.90)), get $+1$, $\hat{Q}(a^2) = 1$
     - Take action a^3 ($r \sim$Bernoulli(0.1)), get 0, $\hat{Q}(a^3) = 0$
  2. Set $t = 3$, Compute upper confidence bound on each action

$$UCB(a) = \hat{Q}(a) + \sqrt{\frac{2 \log t}{N_t(a)}}$$

  3. $t = 3$, Select action $a_t = \arg\max_a UCB(a)$,
  4. Observe reward 1
  5. Compute upper confidence bound on each action

Source: Emma Brunskill (CS232 Course)

Problem of Generalization

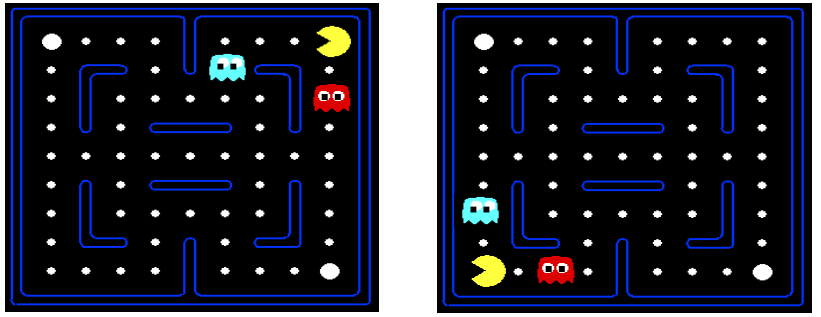
- Problem when the state space is very large
  - Visiting all states is not possible at training time.
  - Memory issue: cannot fit the q-table in memory.

- Need for Generalization
  - Learn from *small number* of training states encountered in training
  - Generalize that experience to *new*, *similar situations* that not encountered before

- Feature-based Representations
  - Features or properties are functions from states to real numbers (often 0/1) that capture important properties of the state.
  - Example features that can be computed from the state
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts

State space is raw pixels, the state space size is large, cannot experience all states.

$$\left(256^{100\times200}\right)^{3}$$



The states are similar. But for Q-learning it will be a different entry. It does not know that in essence these states are the same.

Linear Value Functions

- Using a feature representation, a q function (or value function) can be expressed for any state using a set of weights:

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

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

- Now the goal of Q-learning is to estimate these weights from experience. Once the weights are learned the resulting Q-values will hopefully be close to the true Q values.

- Main consequence:
  - A new state that shares features with a previously seen state will have the same Q-values.
- Disadvantage
  - If there are less features, actually different states (good and bad states) may start looking the same.

Approximate Q-learning

- Q-learning with linear Q-functions: $Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$

transition $= (s, a, r, s')$

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

Exact Q function updates

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

"target" "prediction"

Approximate Q function updates

- Interpretation:
  - Adjust weights of active features.
  - If the difference is positive (what we get is higher than previous) and the feature value is 1 then the weight increases and the q value increases. No change if the feature value is 0.