GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

E016350 - Artificial Intelligence

Lecture 18

Decisions & Action

Reinforcement learning

Aleksandra Pizurica

Ghent University

Fall 2024

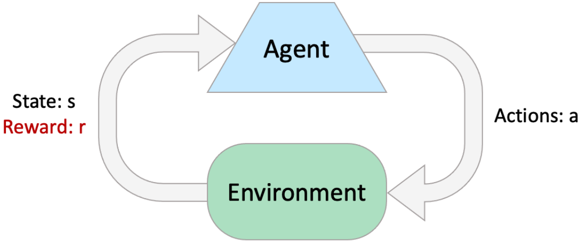
Overview

- Reinforcement Learning (RL) as MDP
- Passive RL
- Active RL including exploration
- Generalization
- Policy search

[R&N], Chapter22

These slides are based on: S. Russel and P. Norvig: *Artificial Intelligence: A Modern Approach*, (Fourth Ed.), http://aima.cs.berkeley.edu/, the corresponding slides of S. Russel and the slides of D. Klein and P. Abbeel (course Introduction to Artificial Intelligence), http://ai.berkeley.edu/

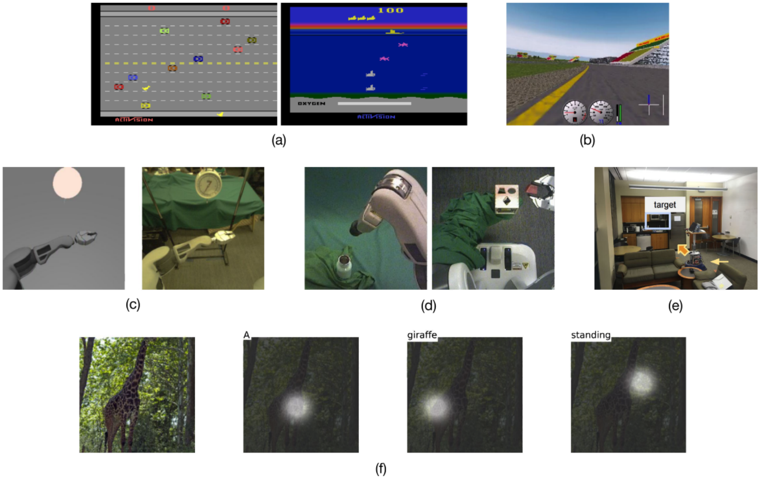
Reinforcement Learning



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!

Example applications

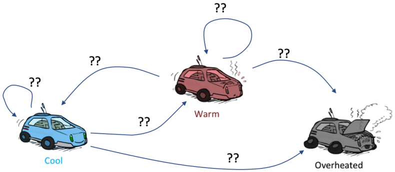


(a) Game playing; (b) racing simulators; (c) transferring knowledge from simulators to the real world; (d) robotic tasks (assembling pieces, picking up, precision tasks...) (e) navigation in various spaces; (f) learning where to look in order to recognize/interpret a captured scene.

K. Arulkumaran et al: Deep Reinforcement Learning: A brief Survey, IEEE Signal Processing Magazine, 2017.

Reinforcement Learning – Problem formulation

- Still assume a Markov decision process (MDP)
  - A set of states $\mathcal{S}$ (with an initial state s_0)
  - A set $\mathcal{A}$ of actions in each state: $Actions(s)$
  - A transition model $T(s,a,s') = P(s'|s,a)$ and
  - A reward function $R(s,a,s')$
- Still looking for a policy $\pi(s)$
- New twist: 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
- MDP computes an optimal policy
- RL learns an optimal policy



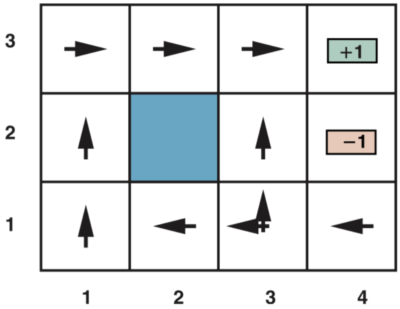
A categorization of RL algorithms

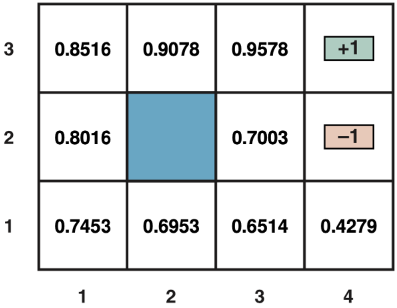
- Passive vs Active
  - Passive: Agent executes a fixed policy (**is told what to do**) and evaluates it
  - Active: Agent **decides what to do** and updates policy as it learns
- Model-based vs Model-free
  - Model-based: Learn transition and reward model, use it to get optimal policy
  - Model free: Derive optimal policy without learning the model

A categorization of RL algorithms, cont'd

- Model-based reinforcement learning
  - Use a transition model of the environment to help interpret the reward signals and to make decisions about how to act
  - The model can be
    - initially unknown (the agent learns it by observing the effects of its actions)
    - known (e.g., knowing the rules of chess without knowing how to make good moves)
- Model-free reinforcement learning
  - Action-utility learning
    - Q-learning
  - Policy search
    - reflex-agent

Passive Reinforcement Learning



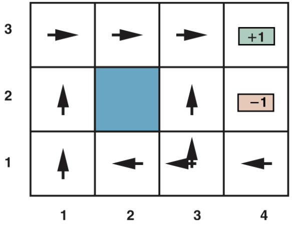


Goal: evaluate how good a fixed policy $\pi(s)$ is

$\rightarrow$ need to learn the expected utility $U^{\pi}(s)$ for each state s

- passive learning agent
- similar task to policy eval. in MDPs but $P(s'|s,a)$ and $R(s,a,s')$ unknown

Passive Reinforcement Learning



Agent executes a sequence of trials:

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

The goal is to learn

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(S_t, \pi(S_t), S_{t+1})\right]$$

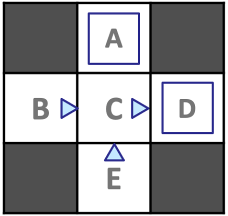
S_t is a RV denoting the state reached at t executing π, starting from $S_0 = s$

Direct Evaluation

- Idea:
  - The utility of a state = the expected **reward-to-go**
  - Each trial provides a sample of this quantity
- Practically:
  - 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
- This is called direct evaluation or direct utility estimation

Direct Evaluation: Example

Input Policy π

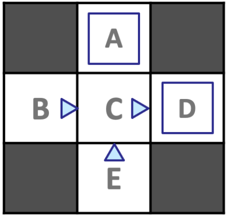


Assume: $\gamma = 1$

Output Values

Direct Evaluation: Example

Input Policy π



Assume: γ = 1

Observed Episodes (Training)

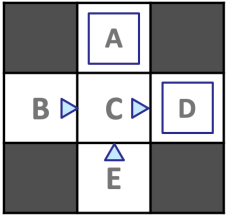
Episode 1

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

Output Values

Direct Evaluation: Example

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

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

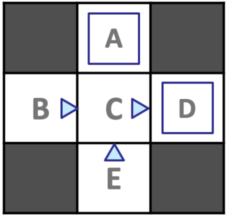
Episode 2

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

Output Values

Direct Evaluation: 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

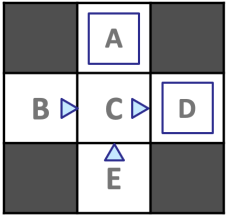
Episode 3

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

Output Values

Direct Evaluation: 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

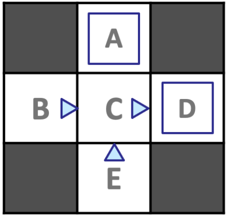
Episode 4

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

Output Values

Direct Evaluation: 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

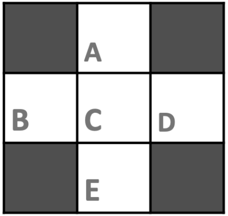
Episode 3

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

Episode 4

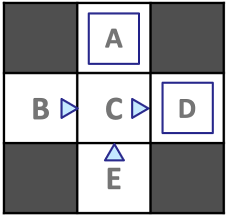
E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Direct Evaluation: 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

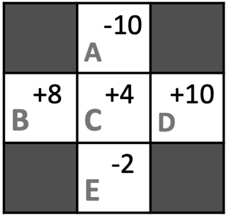
Episode 3

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

Episode 4

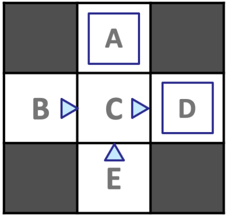
E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Direct Evaluation: 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

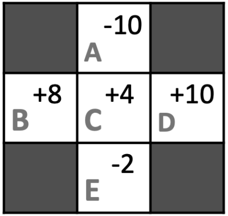
Episode 3

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

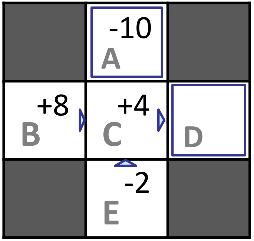
Output Values



Reduced to a standard supervised learning problem with (*state*, *reward-to-go*) pairs

Problems with Direct Evaluation

- What is good about direct evaluation?
  - Easy to understand
  - Doesn't require any knowledge of T, R
  - Eventually computes the correct average values, using just sample transitions
- What is bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn



If B and E both go to C under this policy, how can their values be different?

Adaptive Dynamic Programming (ADP)

- A smarter method: make use of Bellman equations to get $U^\pi(s)$:

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

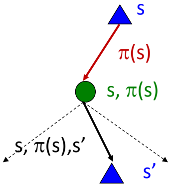
- Need to learn $P(s'|s,\pi(s))$ and $R(s,\pi(s),s')$ from trials
- Plug-in learnt transition and reward in the Bellman equations
- Solving for $U^\pi(s)$:
  - Solving a system of n linear equations
    (Note: Bellman equations are linear when the policy is fixed!)
  - Alternatively: modified policy iteration using a simplified value iteration process
    (to update the utility estimates after each change to the learned model)
- Inefficient if state space is large
  (e.g., in Backgammon need to solve $\approx 10^{20}$ equations in 10^{20} unknowns)
- ADP is a standard baseline for other RL methods

Temporal Difference (TD) Learning

- Does not require the agent to learn the transition model
- Best of both worlds
  - Only update states that are directly affected
  - Approximately satisfy the Bellman equations

Digression: Bellman updates without knowing/learning T and R?

- Simplified Bellman updates calculate U for a fixed policy:
  - Each round, replace U with a one-step-look-ahead layer over U
    $U_0^\pi(s) = 0$
    $U_{i+1}^\pi(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma U_i^\pi(s')]$
  - This approach fully exploited the connections between the states
    Unfortunately, we need T and R to do it!



- **Key question**: how to do this update without knowing T and R and without learning them?
  - I.e., how to take a weighted average without knowing the weights?

Digression: Sample-Based Policy Evaluation?

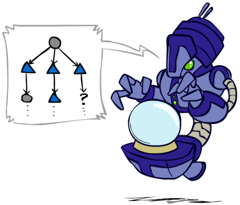
- We want to improve our estimate of U by computing these averages:

$$U^{\pi}_{i+1}(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma U^{\pi}_i(s')]$$

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

$$\begin{aligned}
sample_1 &= R(s,\pi(s),s'_1) + \gamma U^{\pi}_i(s'_1) \\
sample_2 &= R(s,\pi(s),s'_2) + \gamma U^{\pi}_i(s'_2) \\
\dots \\
sample_n &= R(s,\pi(s),s'_n) + \gamma U^{\pi}_i(s'_n)
\end{aligned}$$

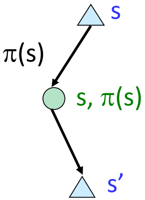
$$U^{\pi}_{i+1} \leftarrow \frac{1}{n}\sum_i sample_i$$



Credit: P. Abbeel & D. Klein

Temporal Difference Learning

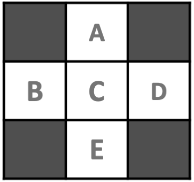
- Big idea: learn from every experience!
  - Update $U(s)$ each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average



$$
\begin{aligned}
sample &= R(s, \pi(s), s') + \gamma U^{\pi}(s') \\
U^{\pi}(s) &\leftarrow (1-\alpha)U^{\pi}(s) + \alpha \cdot sample \\
U^{\pi}(s) &\leftarrow U^{\pi}(s) + \alpha(sample - U^{\pi}(s))
\end{aligned}
$$

Temporal Difference Learning: Example

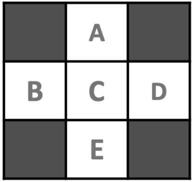
States



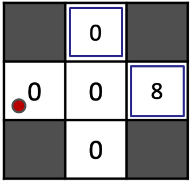
Assume: $\gamma = 1$, $\alpha = 1/2$

Temporal Difference Learning: Example

States

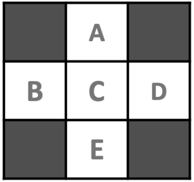


Assume: $\gamma = 1$, $\alpha = 1/2$



Temporal Difference Learning: Example

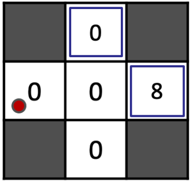
States



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

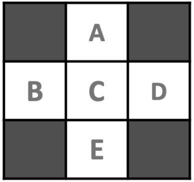
Observed Transitions

B, east, C, -2



Temporal Difference Learning: Example

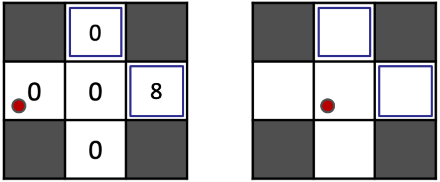
States



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

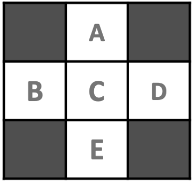
Observed Transitions

B, east, C, -2



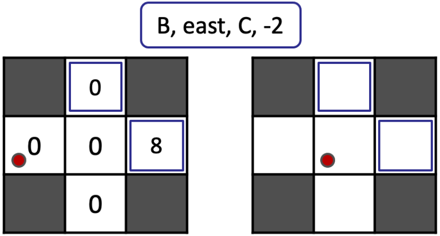
Temporal Difference Learning: Example

States



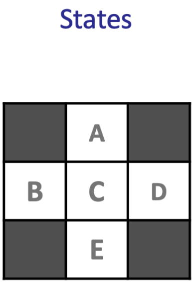
Assume: γ = 1, α = 1/2

Observed Transitions

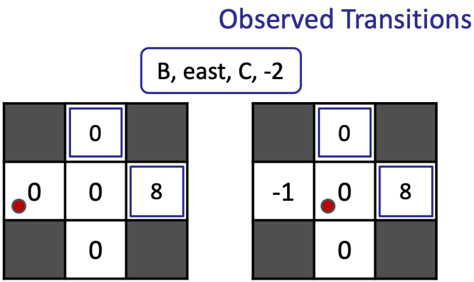


$$U^\pi(s) \leftarrow (1-\alpha)U^\pi(s) + \alpha\Big[R(s,\pi(s),s') + \gamma U^\pi(s')\Big]$$

Temporal Difference Learning: Example

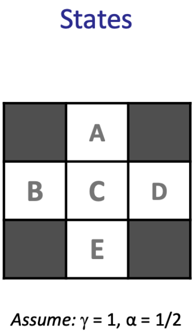


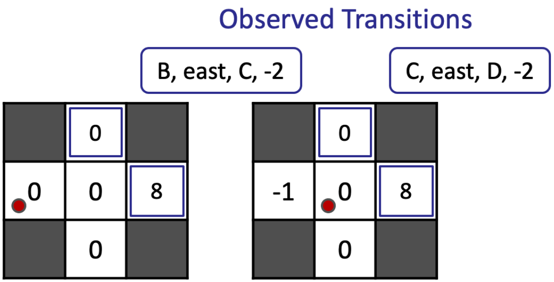
Assume: $\gamma = 1$, $\alpha = 1/2$



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

Temporal Difference Learning: Example

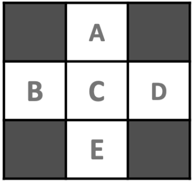




$$U^{\pi}(s) \leftarrow (1-\alpha)U^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma U^{\pi}(s')\right]$$

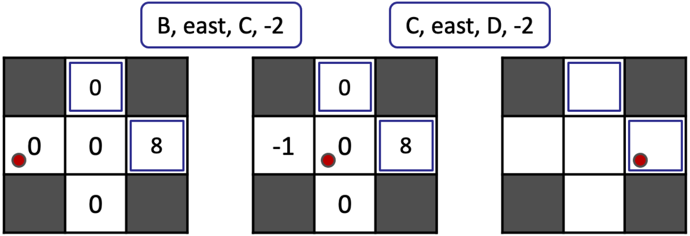
Temporal Difference Learning: Example

States



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

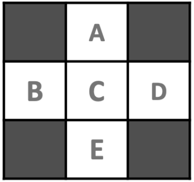
Observed Transitions



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

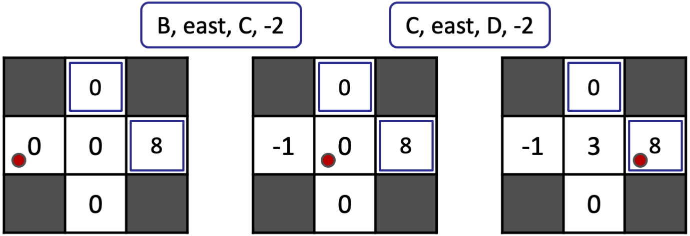
Temporal Difference Learning: Example

States



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

Observed Transitions



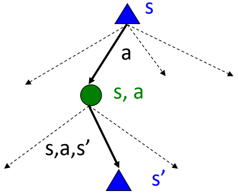
$$U^{\pi}(s) \leftarrow (1-\alpha)U^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma U^{\pi}(s')\right]$$

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, it won't work well!
  - Greedy agent (applies simply the learned model)
  - Optimal actions in the learned and real environments may differ
- Idea: learn an action-utility function $Q(s,a)$

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

$$Q(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$



- Makes action selection model-free too!

Active Reinforcement Learning

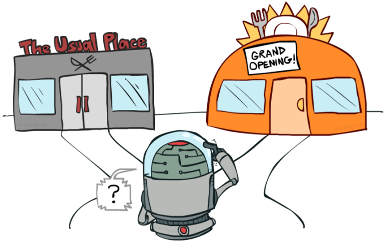
- Agent updates policy as it learns
- Goal: learn the optimal policy
- Needs to learn a complete transition model with outcome probabilities of *all* actions (not just for fixed policy)
- Learning using the passive ADP agent
  - Estimate the model $P(s'|a,s)$, $R(s,a,s')$ from observations
  - The utilities it needs to learn obey the Bellman equation

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

  - Solve using value iteration or policy iteration
- Agent has "optimal" action
- Simply execute the"optimal" action. But should it?

Exploration vs. Exploitation

- The passive approach gives a greedy agent
- Exactly executes the recipe for solving MDPs
- Rarely converges to optimal utility and policy
  - Learned model different from true environment
- What to do?



Source: Berkeley CS188

- Trade-off
  - **Exploitation**: Maximize rewards using current estimates
    - Agent stops learning and starts executing policy
  - **Exploration**: Maximize long term rewards
    - Agent keeps learning by trying out new things

Exploration Function

- Suppose we are using value iteration in an ADP agent
- Alter Bellman equations using optimistic utilities $U^+(s)$

$$U^+(s) = \max_a f\left(\sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma U^+(s')], N(s,a)\right)$$

$N(s,a)$ – the number of times a has been tried in s

$f(u,n)$ – the **exploration function** (trade-off between greed and curiosity)

- should increase in expected utility u
- should decrease with number of tries n

- A simple definition

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

R^+ – an optimistic estimate of the best possible reward obtainablein any state

N_e – a fixed parameter

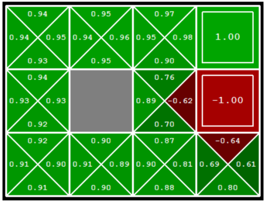
Q-Learning

- Exploration function gives an active ADP agent
- A corresponding TD agent can be constructed
  - Surprisingly, the TD update can remain the same
  - Converges to the optimal policy as active ADP
  - Slower than ADP in practice
- Q-learning learns an action-value function $Q(s,a)$:
  - $Q(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$
  - Utility values $U(s) = \max_{a} Q(s,a)$
- A model-free TD method
  - No model for learning or action selection

Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $U_0(s) = 0$
  - Given U_i calculate the depth $i+1$ values for all states:
    $U_{i+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_i(s')]$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$
  - Given Q_i calculate the depth $i+1$ values for all q-states:
    $Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')]$



Q-Learning

- We would like to do Q-value updates to each Q-state

  $$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')]$$

  - But can't compute this update without knowing T, R
- Instead, compute average as we go
  - Receive a sample transition (s,a,r,s')
  - This sample suggests
    $$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$
- But we want to average over results from (s,a) (Why?)
- So keep a running average

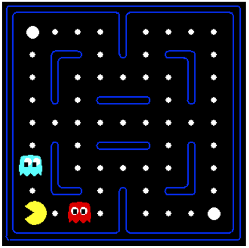
  $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Generalizing across states

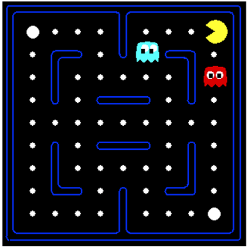
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning

Example: Pacman

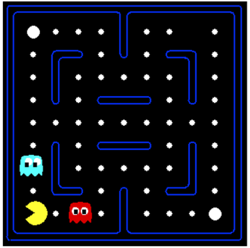
Let's say we discover through experience that this state is bad:



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



Or even this one!

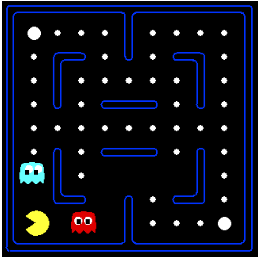


Introduction to Artificial Intelligence of P. Abbeel and D. Klein http://ai.berkeley.edu/

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
    - ★ $1/(\text{dist to dot})^2$
    - ★ Is Pacman in a tunnel? (0/1)

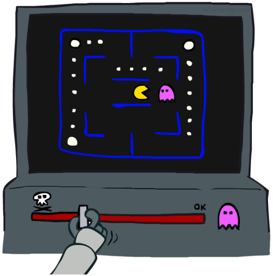
    etc.
  - A q-state (s, a) can also be described with features (e.g. action moves closer to food)



Approximate Q-Learning

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

- Q-learning with linear Q-functions:
  - $transition = (s, a, r, s')$
  - $difference = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$
  - Exact Q's:
    $Q(s,a) \leftarrow Q(s,a) + \alpha[difference]$
  - Approximate Q-learning:
    $w_i \leftarrow w_i + \alpha[difference] f_i(s,a)$
- 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

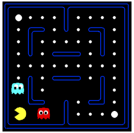


Source: Berkeley CS188

Example: Q-Pacman

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

s

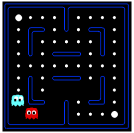


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

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

s'



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

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

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

difference $= -501$

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

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

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

Introduction to Artificial Intelligence of P. Abbeel and D. Klein http://ai.berkeley.edu/

Policy search

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

Summary

- RL is necessary for agents in unknown environments
- Passive Learning: Evaluate a given policy
  - Direct utility estimate by supervised learning
  - ADP learns a model and solves linear system
  - TD only updates estimates to match successor state
- Active Learning: Learn an optimal policy
  - DP using proper exploration function
  - Q-learning using model-free TD approach
- Policy search
  - Simple updates of feature weights in approx Q-learning
  - Better methods (exploit structure, sample wisely, change multiple parameters...)