Introduction to Artificial Intelligence
COMP 3501 / COMP 4704-4
Lecture 15: Reinforcement Learning

Prof. Nathan Sturtevant

Today

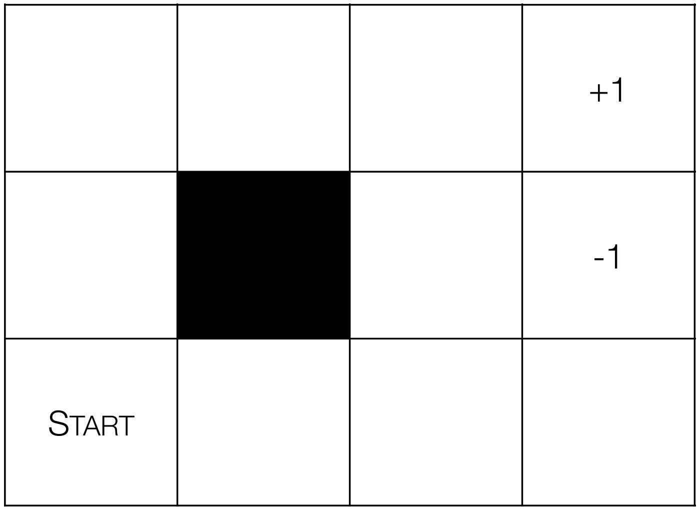
- Demo from last time
- Making complex decisions (17.1, 17.2, 17.3)
  - Background for Reinforcement Learning
- Reinforcement Learning (Ch 21)

Limitations

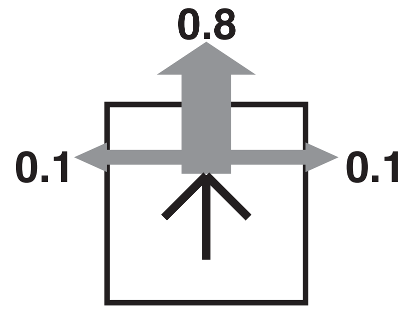
- Previous approaches learn a function or a classifier
  - How would you learn to move in an environment?
  - A* works on deterministic domains
  - Need more advanced approaches for more complex domains

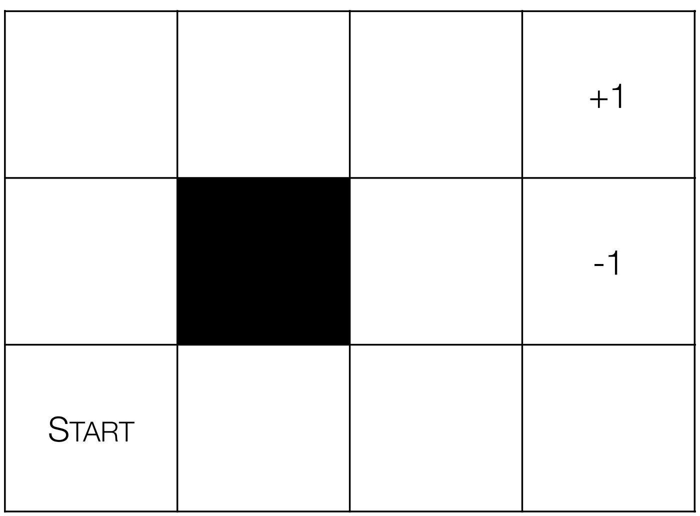
What happens if the world is more stochastic?

- Assume a 4x3 grid world
  - The agent has 2 goal states
  - Assume the world is fully observable
  - Actions: Left, Right, Up, Down



What if actions are nondeterministic?





What is the chance of reaching a goal?

Transition model

- Previously, actions were deterministic
  - Now, actions have probabilities:
  - P(*s*' | *s, a*)
    - Probability of ending in state *s*' given that we take action *a* in state *s*

Markov

- An environment is Markov (Ch 15) if:
  - The current state only depends on a finite number of previous states
  - 1-Markov: on requires history of one state
  - Also implies optimal policy doesn't rely on history

Utility

- The utility function depends on the full history
- Reward for each step in the world (-0.04)
  - Terminal states have reward -1/1
- Utility is cost of path until a goal state is reached
  - Negative reward at each step encourages short paths

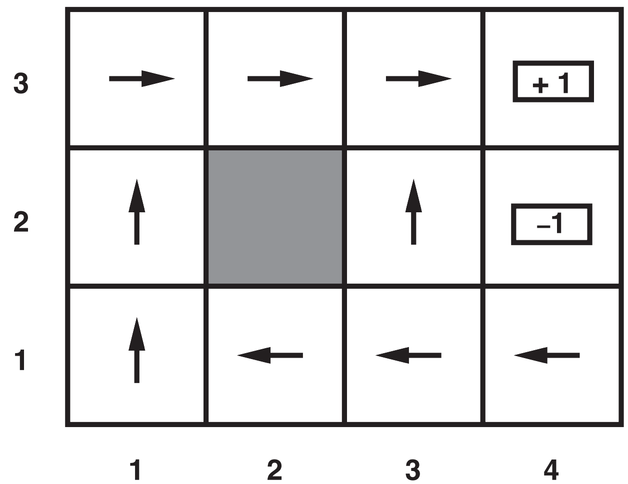
Markov Decision Process (MDP)

- Markov Decision Process
  - Initial state s_0
  - A set of states
  - A set of actions for each state ACTIONS(s)
  - A transition model P(*s*' | *s, a*)
  - A reward function R(*s*)

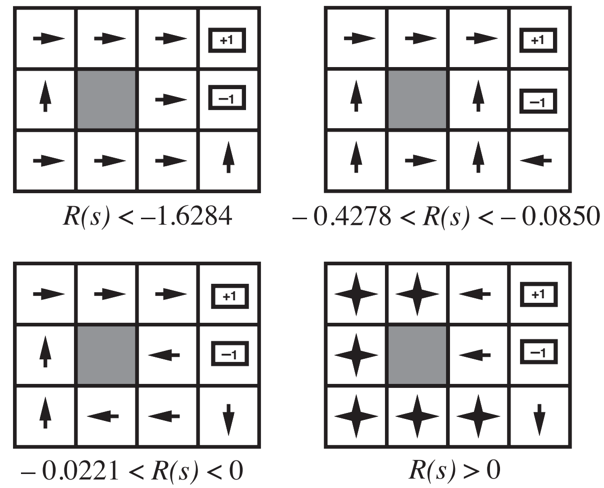
Solving a MDP

- What does a solution to a MDP look like?
  - Cannot be a fixed set of actions
  - Need a general policy for each state, π
  - Policy in a state is π(s)
- Each policy has an expected utility
  - Average reward over executions of policy
- The optimal policy, π*, has maximum utility

Optimal Policy



Other policies



Reward models

- Additive rewards
  - $R(s_0) + R(s_1) + R(s_2) \ldots$
- Discounted rewards
  - $R(s_0) + \gamma \cdot R(s_1) + \gamma^2 \cdot R(s_2) \ldots$
  - $0 \leq \gamma \leq 1$ is a discount factor
  - $\gamma = 1$ is equivalent to additive rewards

Discounted rewards

- Discounted rewards are needed if there are infinite sequences
  - Can bound the total utility:

$$\sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma}$$

Expected utility of a policy

- We can now formally define the utility of a policy
  - Let the initial state be s_0
  - Let S_t be the state reached at time *t* when following policy π

$$U^\pi(s_0) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

- Optimal policy π*: $\pi^*_{s_0} = \underset{\pi}{\text{argmax}}\, U_\pi(s)$
- But, policy independent of s_0

Testing for optimal policies

- The Bellman equation defines when a policy is optimal

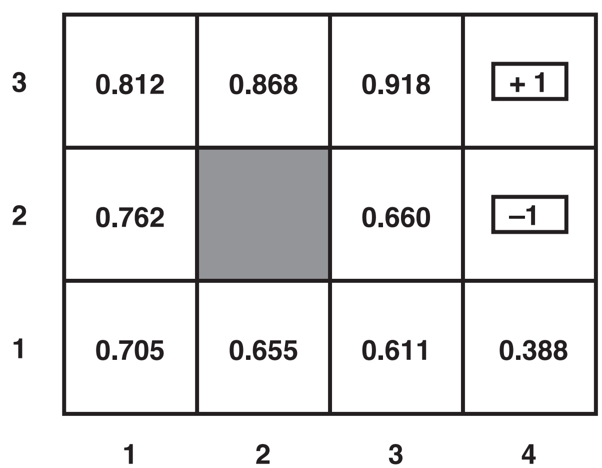
$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

- But, how do we find the optimal policy?
  - Initialize utilities to 0, then iterative update:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U_i(s')$$

  - Called *value iteration* (Often use V(s), not U(s))

Value Iteration example - is [1,1] optimal?



Value Iteration example

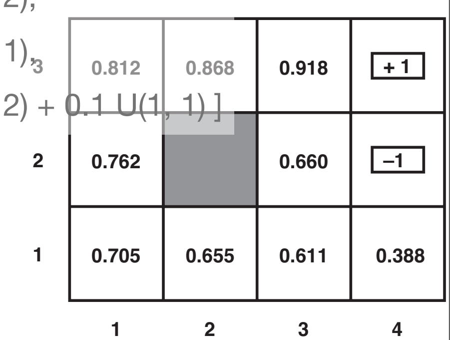
- U(1, 1) = -0.04 + γ max

  [0.8 U(1, 2) + 0.1 U(2, 1) + 0.1 U(1, 1),

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

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

  0.8 U(2, 1) + 0.1 U(1, 2) + 0.1 U(1, 1)]



Policy iteration

- Policy is much coarser than value function
- Policy iteration involves:
  - Evaluation: Given policy π_i, find U^{π_i}
  - Improvement: Compute π_{i+1} based on U_i

$$\pi^*_{s_0} = \operatorname*{argmax}_{\pi} U_\pi(s)$$

Performing policy iteration

- Policy can be "solved" as a linear equation
- Policy can be incrementally updated
  - Replace action with policy, π

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

- Now we are ready to tackle Reinforcement Learning!

Reinforcement Learning

- What if we don't have any source of training examples?
  - Can we learn directly from experiences in the world?
    - Must receive feedback for good/bad experiences
    - Called rewards or reinforcement
  - Assume that reward input is known
    - eg don't have to figure out that a particular sensory input corresponds to reward

Reinforcement Learning

- Previously we assumed a complete model of the environment and reward function
  - Can we really give up this assumption?

Two types of reinforcement learning

- Value-based (utility)
  - Learn the value of states to select best
- Q-learning
  - Learn the value of actions in a state

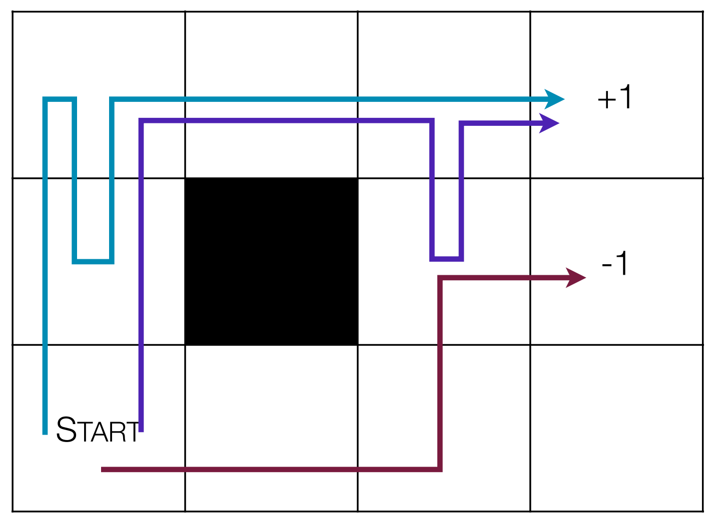
Passive learning

- Assume a observable agent & a fixed policy, π
- How can we learn the value of π? [$U^\pi(s)$]
  - Similar to policy iteration
  - Unknown transition model: P(s' | s, a)
- As before, by definition:

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

Direct utility estimation

- Each trial of the agent provides a sample of the utility
  - Run a trial
  - For each state, update the utility according to the average utility seen so far on all trials
- What is the drawback of this approach?
  - Is there information that can improve it?

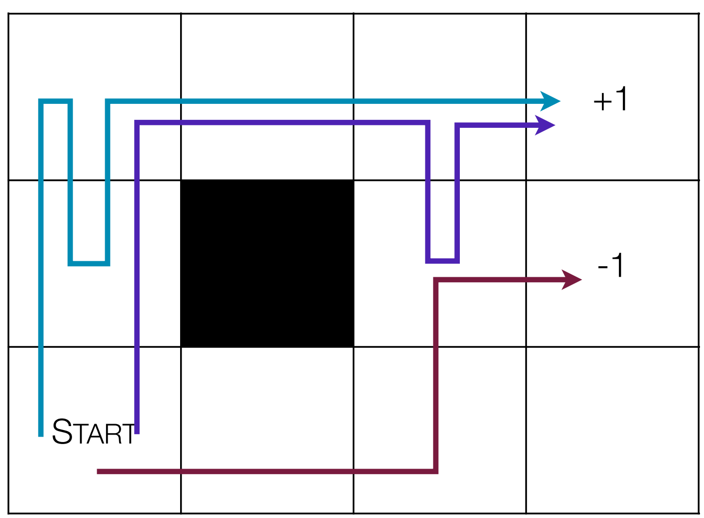


Direct utility estimation

- This approach ignores that the values of states are correlated
- If we knew the transition probabilities, we could use the bellman equation to easily solve the problem
- Also called Monte-Carlo Policy Estimation

Modified Policy Iteration

- Update the utility of each state with the Bellman equation
  - Estimate the probabilities given the history
- Called Adaptive Dynamic Programming if we solve the MDP directly instead of sampling it



Temporal Difference Learning

- Version 1 (from book)
  - $U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$
- Note that the update only considers the next state
  - But, when running over many trials, the frequency of next states will approach the true distribution
- Compare with ADP which estimates prob. directly

Active Reinforcement Learning

- Now, consider learning the policy *while* we learn the value of states

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

- What if we always act according to these utilities?
  - May not converge
  - Need exploration (eg soft-max)
- Note that we also need to learn the model of P(…)

Q learning

- Q-learning learns Q(s, a) instead of utilities [V(s)/U(s)]
  - Q(s, a) is the value of taking action *a* in state *s*
  - $U(s) = \max_a Q(s,a)$
- Q-learners do not need a model of the world
  - They directly learn actions

Q-learning

- Equivalence of bellman equation for Q(s, a):

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

- Can convert into a TD update, which doesn't require P()

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$$

Generalization

- These approaches require that we represent every state in the state space
- Many problems far too large to fit into memory
  - Simple policies exist in a lower-dimensional space
- Generate features of the state space & learn from features as if they were the states themselves

Temporal Difference Learning in practice

- [Departing slightly from book here]
- TD learning learns directly from exploring the world
  - Often described as TD(λ)
- Different views of TD(λ):
  - Are we exploring the world and dynamically learning?
  - Do we have traces of exploration in the world from which we are learning?
- Focus on the second case

Eligibility Traces

- An eligibility trace is a sample of the plays that were made in a game from the beginning to the end
- Training can occur on eligibility traces
  - Have an associated payoff
  - For our purposes, payoff is only at the end
  - Models a game
    - Not difficult to extend to payoffs at every state

Monte-Carlo

- Play a game with the current value function
  - Often use a soft-max (small probability of a random move) instead of a pure maximization
  - Gives some chance of exploration and reaching every state in the game
- At the end of the game, take note of the score
  - Train all the states in the history of moves to predict the final score of the game
- This won't work if it is hard/impossible to reach the end of the game

Monte-Carlo

- Given a eligibility trace s_1, a_1, s_2, a_2, … a_{n-1}, s_n
- Followed by a reward r.
- Train function approximator with:
  - output($f(s_i)$) ← r
  - f is the features associated with state i
  - Note that this is supervised learning

Dynamic Programming

- Given a eligibility trace s_1, a_1, s_2, a_2, … a_{n-1}, s_n
- Followed by a reward r.
- Train function approximator with:
  - output(f(s_i)) ← output(f(s_{i+1}))
  - where first training is output(f(s_n)) ← r
  - (from i = n to i = 1)

TD(λ)

- Combination of the two approaches
  - Given a eligibility trace s_1, a_1, s_2, a_2, … a_{n-1}, s_n
  - Followed by a reward r.
  - Train function approximator with:
    - $output(f(s_n)) \leftarrow r$
    - $output(f(s_{n-1})) \leftarrow (1-\lambda)output(f(s_n)) + \lambda r$
  - In general over *i* steps:

$$R(i) = (1-\lambda)output(f(s_i)) + \lambda R(i+1)$$

$$R(n) = r \qquad output(f(s_i)) \leftarrow R(i)$$