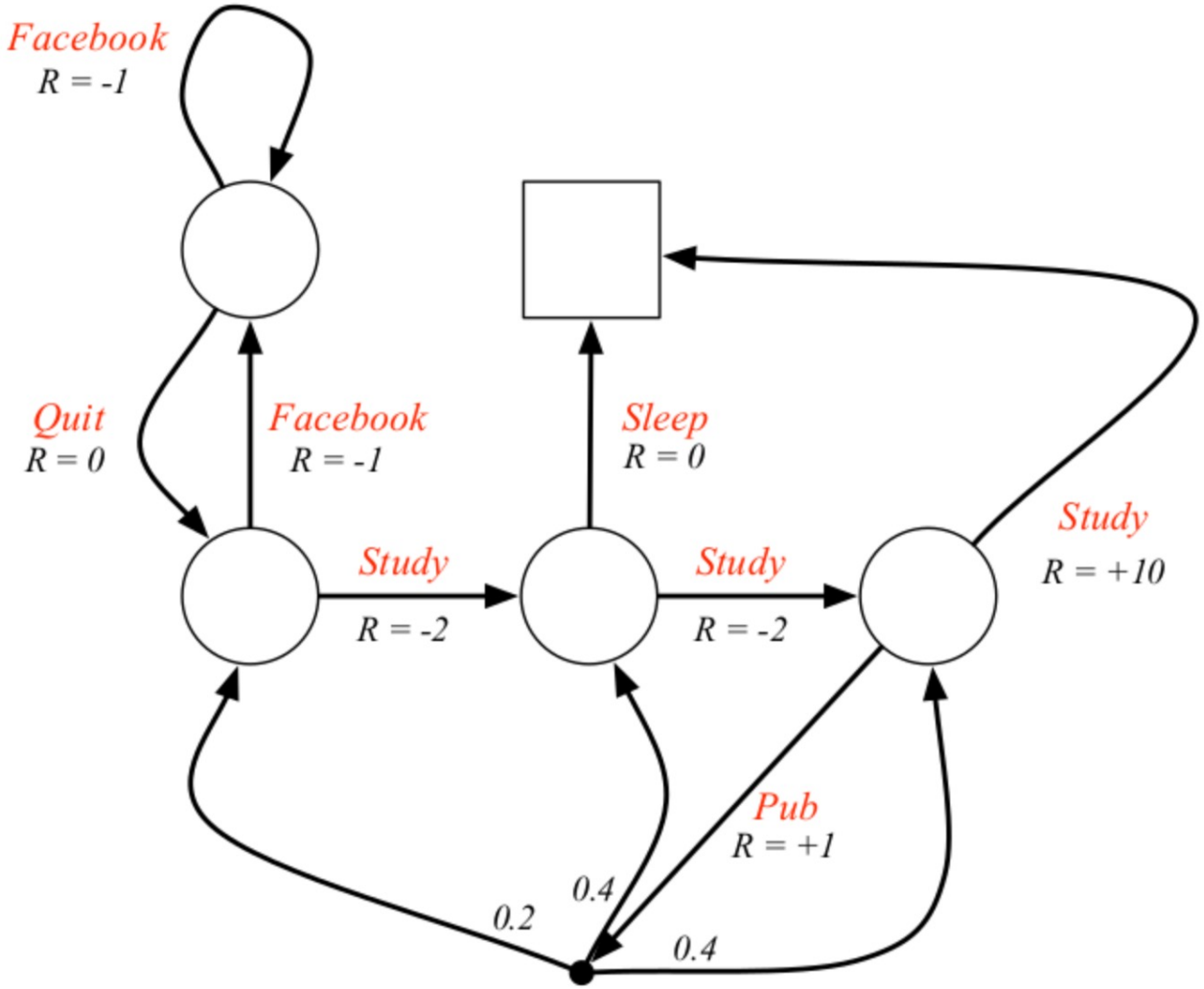
CSCI 3202: Intro to Artificial Intelligence

Lecture 4: Policy iteration, Q-learning

Rhonda Hoenigman

Department of Computer Science



Student MDP with actions

Value Iteration

```
def value_iteration(mdp, tolerance):

    # initilize utility for all states

    # iterate:

        # make a copy of current utility, to be modified

        # initialize maximum change to 0

        # for each state s:

            # for each available action, what next states
            # are possible, and their probabilities?

            # calculate the maximum expected utility

            # new utility of s = reward(s) +
            #                    discounted max expected utility

            # update maximum change in utilities, if needed

        # if maximum change in utility from one iteration to the
        # next is less than some tolerance, break!

    return # final utility
```

max utility and association

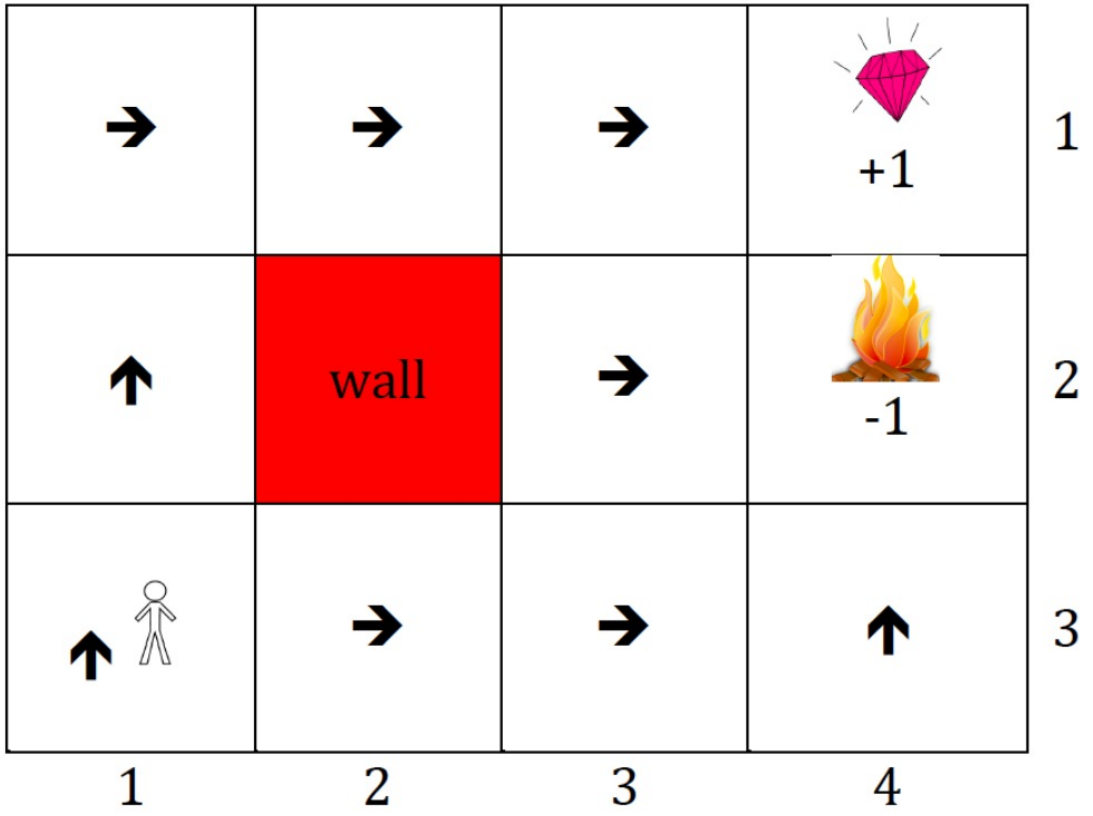
Loop until change in utility value < threshold

Policy Iteration

Value iteration: Changes to U(s) may not result in policy change.

- Faster: Iterate through policies instead of utilities. fewer iterations than value iteration
- Terminate algorithm when no policy changes

actions in all states are unchanged.



Policy Iteration

Policy Iteration: Iterate between policy evaluation and policy improvement to calculate π

Two steps:

Policy Evaluation: given policy π_i

$U_i = U_i^{\pi}$ - the utility of each state if policy π_i were executed

- only one action, not all possible actions in state

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

Policy improvement:

Calculate the new policy π_{i+1} using π_i and U_i

Policy Iteration - code from github repo on canvas

- **Policy evaluation**: Given a policy π_i, calculate $U_i = U(\pi_i)$, the utility of each state if π_i were to be executed.
- **Policy improvement**: Calculate a new policy π_{i+1} using one-step look-ahead based on the utility values calculated.

```
def expected_utility(a, s, U, mdp):
    """The expected utility of doing a in state s, according to the MDP and U."""
    return sum([p * U[s1] for (p, s1) in mdp.T(s, a)])
```

```
def policy_iteration(mdp):
    """Solve an MDP by policy iteration [Figure 17.7]"""
    U = {s: 0 for s in mdp.states}
    pi = {s: random.choice(mdp.actions(s)) for s in mdp.states}
    while True:
        U = policy_evaluation(pi, U, mdp)
        unchanged = True
        for s in mdp.states:
            a = argmax(mdp.actions(s), key=lambda a: expected_utility(a, s, U, mdp))
            if a != pi[s]:
                pi[s] = a
                unchanged = False
        if unchanged:
            return pi
```

eval

initialize U, π - random actions

calculate U for current action for all states

policy improvement

flag

find action w/ max utility

if max action is not current

update

prepare to loop again

exit if no changes

Policy Iteration

Policy improvement:

For each state, can we pick a better action?

$$\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U(s') > \sum_{s'} P(s' \mid s, \pi(s))\, U(s')$$

newly calculated

existing policy

If so, set π(s) = action that maximizes this expected utility

Active Reinforcement Learning

Passive Learning:

- Agent given a fixed policy π
- Learn how good the policy is by learning $U^{\pi}(s)$ (the utility of each state under the policy)

Active Learning:

value & policy iteration

- Agent starts with policy π
- Learn how good policy is
- Adapt it and improve it

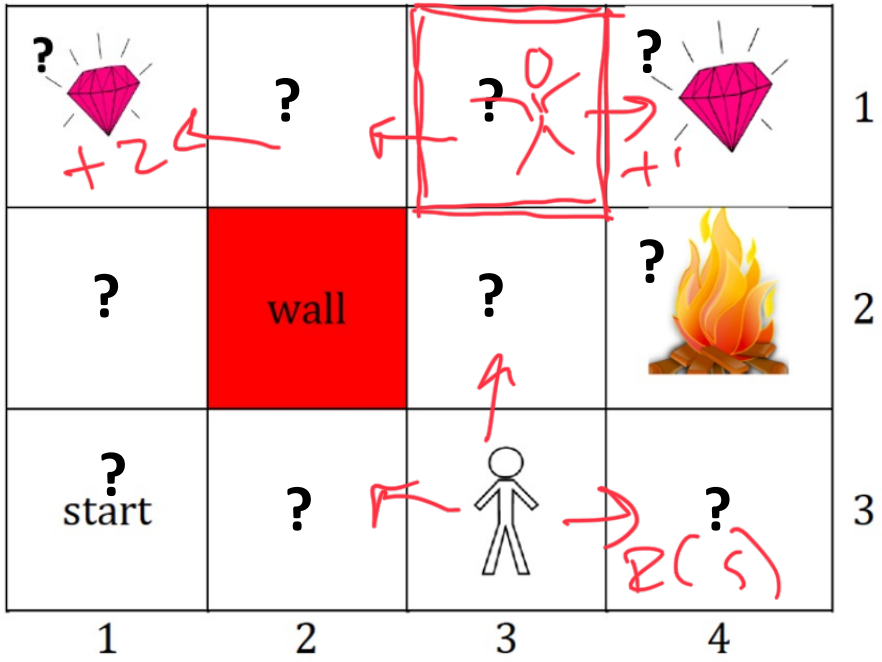
- Temporal difference
- Q-Learning

} active learning more complex problems

Active Reinforcement Learning

What if…

- Agent didn't know model? (Rewards, transition matrix, discounting)
- Multiple rewards of different values?



model

Model free - agent doesn't know transitions, etc

Agent in (1,3) would go right and miss larger reward to the left in value or policy iteration

Reinforcement Learning

- Behavior reinforced on individual steps
- Reinforced at end

ping-pong
chess
} how behavior reinforced

- Can learn model
- Can learn utility of decisions without the model

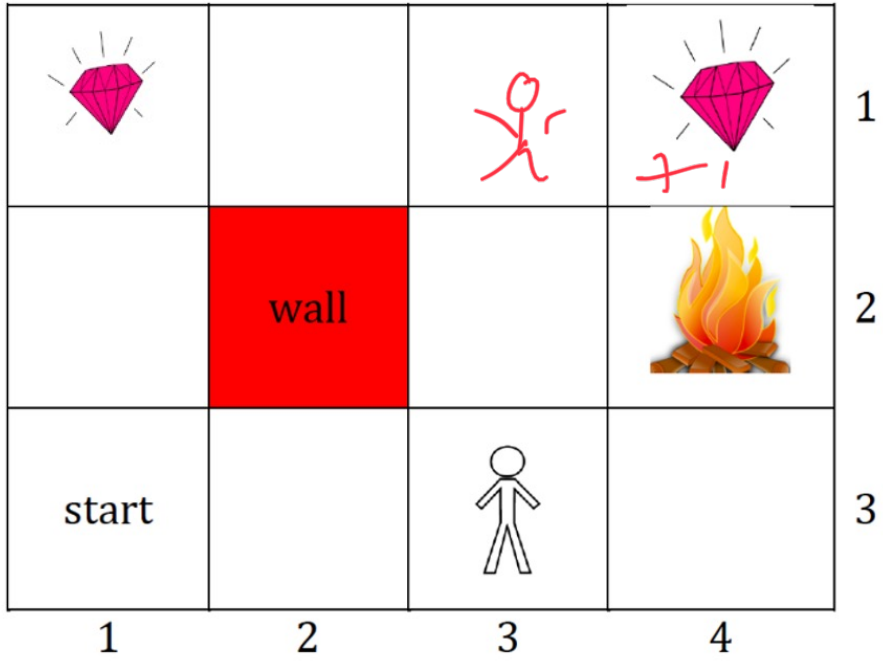
e.g. - transitions - execute trials over and over and measure where agent ends up
value of state-action pair w/o knowing why

- Temporal difference
- Q-Learning

} model free reinforcement learning

Reinforcement Learning – Exploration vs. Exploitation

Balance



Exploitation: agent found a good policy, sticks to it.
Greedy

Exploration: agent should be encouraged to keep looking for better policies

- Can obtain better rewards in the future by learning the true model

choose an action that doesn't have highest reward value

Randomness

Exploration versus Exploitation

Example:

1 2 3

- Start with initial policy (e.g. pull random lever)
  - ➢ exploration
- But as we learn which actions led to reward (pull 3 does better), we want to exploit that knowledge
- One training episode = One pull
  - Call $Q[k]$ the expected reward for pulling lever k

Exploration versus Exploitation

Example: N-armed bandit

One training episode = One pull

Call $Q[k]$ the expected reward for pulling lever k

Suppose our first 10 training episodes are:

Lever #	Result
1	L
3	L
2	W
1	L
3	W
3	L
2	L
2	L
1	W
2	W

1 - 1/3 W

2 - 1/2 W

3 - 1/3 W

pull lever 2

Question: What would we do if we were trying to exploit?

Exploration versus Exploitation

What would we do if we were trying to exploit?

- Bet heavily on lever 2

But that greedy strategy neglects the fact that our next 10 training episodes might turn out like this:

Lever #	Result
3	W
2	L
2	W
3	W
1	L
2	L
3	W
1	L
1	W
2	L

1 – 1/3 W
2 – 1/4 W
3 – 3/3 W

Pull lever 3

Question: Now what would we do if we were trying to exploit?

Exploration versus Exploitation

"Greedy in the limit of infinite exploration"

- The idea: If we have explored enough, then it's time to be greedy.

Early on - explore

Greedy after "enough" trials

ϵ – greedy agent

ϵ is a parameter of the algorithm

- Keep track of estimate of expected payout, Q
- **Exploration**: Pick a random action with probability ϵ
- **Exploitation**: Pick a current best action with probability $1 - \epsilon$

Exploration function

ϵ – greedy is a way to force exploration

How could we incentivize it in the problem formulation? Need to force agent to choose lower-valued solution

Instead of updating based on estimates of utility, we could use an exploration function, which formalizes the trade-off between curiosity and greed:

R+ is overly positive reward for state action pair

u = utility

# of times state-action pair observed

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

R^+ = optimistic (over-) estimate of best possible reward

N_e = some minimum number of times we want to explore each state-action pair

Q–learning – model free

Instead of learning a model and utilities of individual states, let's do what we really care about at the core of this learning problem:

- Estimate the value of doing action a in state s

We do this using Q-functions:

$Q(s, a)$ = the expected value of doing action a in state s

estimates for each action in a state, all actions, not just max action

Q-learning

Temporal Difference: difference in utilities between successive states. Use knowledge of surrounding state utilities to generate correct utility estimates

Temporal Difference Q-learning:

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha\,[R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a)]$$

$Q_i(s,a)$: current value

α: learning parameter, function of # of iterations

$R(s)$: reward for state

γ: discount factor

$\max_{a'} Q_i(s',a')$: value for max action in the state

$Q_i(s,a)$: existing value for state-action

adjustment

Book:

$$\alpha = \frac{60}{59+n}$$

Q_learning

Temporal Difference Q-learning:

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha\,[R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a)]$$

Q - Learning

Init policy, Q, rewards

for each trial

choose random starting state

for each step

if terminal state

calc. cumulative rewards, exit

else

pick an action using ϵ greedy

count state-action pair

calc. Q[s,a] for action selected

update policy for action at Q[s,a]

calc. cumulative rewards using discount step, R(s)

update Q, counter, cumulative rewards for trial

Activity

1. **In one or two sentences, describe the meaning of P(s'|s, a)?**

Probability of being in state s' given you're in state s and do action a.

2. **What is the discount factor? Why is it < 1?**

Rewards decrease at time increases

3. **What is the difference between value iteration and policy iteration?**

Value iteration keeps track of utilities and terminates when update < threshold. Policy iteration also stores policie and terminates when no Policy changes

4. **Which parameter in the Q-Learning algorithm controls exploration?**

ϵ greedy parameter

Policy Iteration

Example: Given the following grid, find the optimal policy of each state using the policy iteration algorithm. The PolicyEvaluation() function sets U for all states using the current policy, U, and the MDP. Use policy iteration algorithm on slides

- The terminal states are a and e, and those states have the rewards shown. Let $\gamma = 0.9$
- The actions are move left and move right.

$$U'(s) = R(s) + \gamma \max\left[P(s'|s,a)\,U(s)\right]$$

- $P(s'|a,s) = 0.80$ success and 0.20 that the agent stays in the same state.

$\pi = \rightarrow$, $U(s) = \phi$

- $R(s) = -0.04$

After 1 iteration

→	→	→		
10	-.04	-.04	-.04	**1**
a	b	c	d	e

Policy Iteration

Example: (continued)

2nd iteration

$v(b) = -.04 + \gamma \begin{cases} \leftarrow .8 \times 10 + .2 \times -.04 = 7.99 \\ \rightarrow .8 \times -.04 + .2 \times -.04 = -.04 \end{cases}$

$= -.04 + .9 \times 7.99 = 7.15$

$v(c) = -.04 + \gamma \begin{cases} \leftarrow .8 \times -.04 + .2 \times -.04 = -.076 \\ \rightarrow .8 \times -.04 + .2 \times -.04 \end{cases}$ policy unchanged

$v(d) = -.04 + \gamma \begin{cases} \leftarrow .8 \times -.04 + .2 \times -.04 \\ \rightarrow .8 \times 1 + .2 \times -.04 \end{cases}$

$= -.04 + .9 \times (.8 \times 1 + .2 \times -.04) = .672$ policy unchanged

$\leftarrow \quad \rightarrow \quad \rightarrow$

10	7.15	-.076	.672	**1**
a	b	c	d	e

Policy Iteration

– This is the third iteration. If you do another iteration you'll find that the policy for state d does change to move left.

Example: (continued)

next iteration

$$U(b) = -.04 + \gamma \max \begin{cases} \leftarrow .8 \times 10 + .2 \times 7.15 \\ \rightarrow .8 \times -.076 + .2 \times 7.15 \end{cases}$$

policy unchanged

$$= -.04 + .9(.8 \times 10 + .2 \times 7.15) = 8.44$$

$$U(c) = -.04 + .9 \max \begin{cases} \leftarrow .8 \times 7.15 + .2 \times -.076 \\ \rightarrow .8 \times .672 + .2 \times -.76 \end{cases}$$

policy updated to left

$$= -.04 + .9(.8 \times 7.15 + .2 \times -.076) = 5.09$$

$$U(d) = -.04 + .9 \max \begin{cases} \leftarrow .8 \times -.076 + .2 \times .672 \\ \rightarrow .8 \times 1 + .2 \times .672 \end{cases}$$

policy unchanged

$$= -.04 + .9(.8 \times 1 + .2 \times .672) = .8$$

10	← 8.44	← 5.09	→ .8	1
a	b	c	d	e