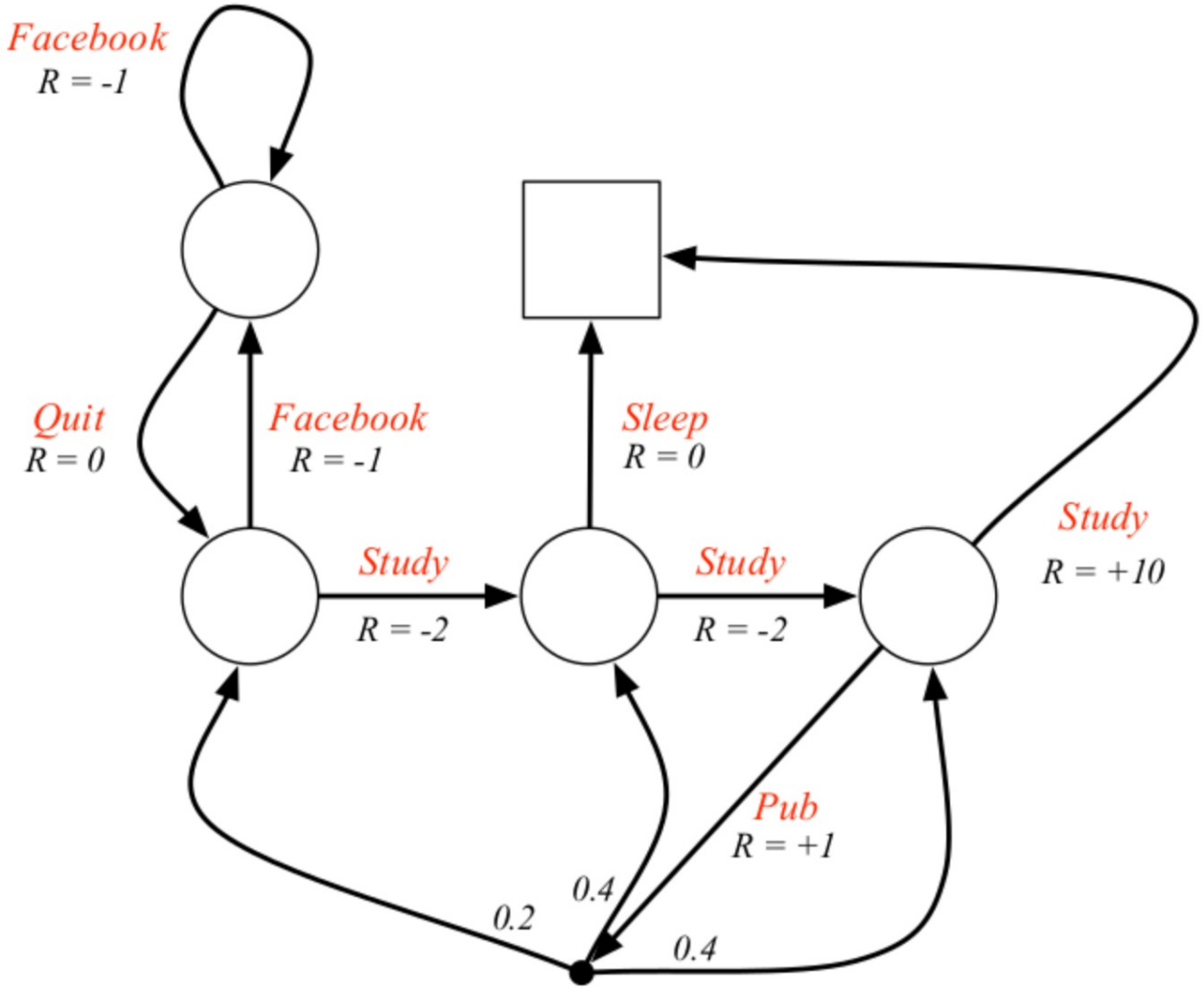
CSCI 3202: Intro to Artificial Intelligence

Lecture 4: Policy iteration, Q-learning

Rhonda Hoenigman

Department of Computer Science



Student MDP with actions

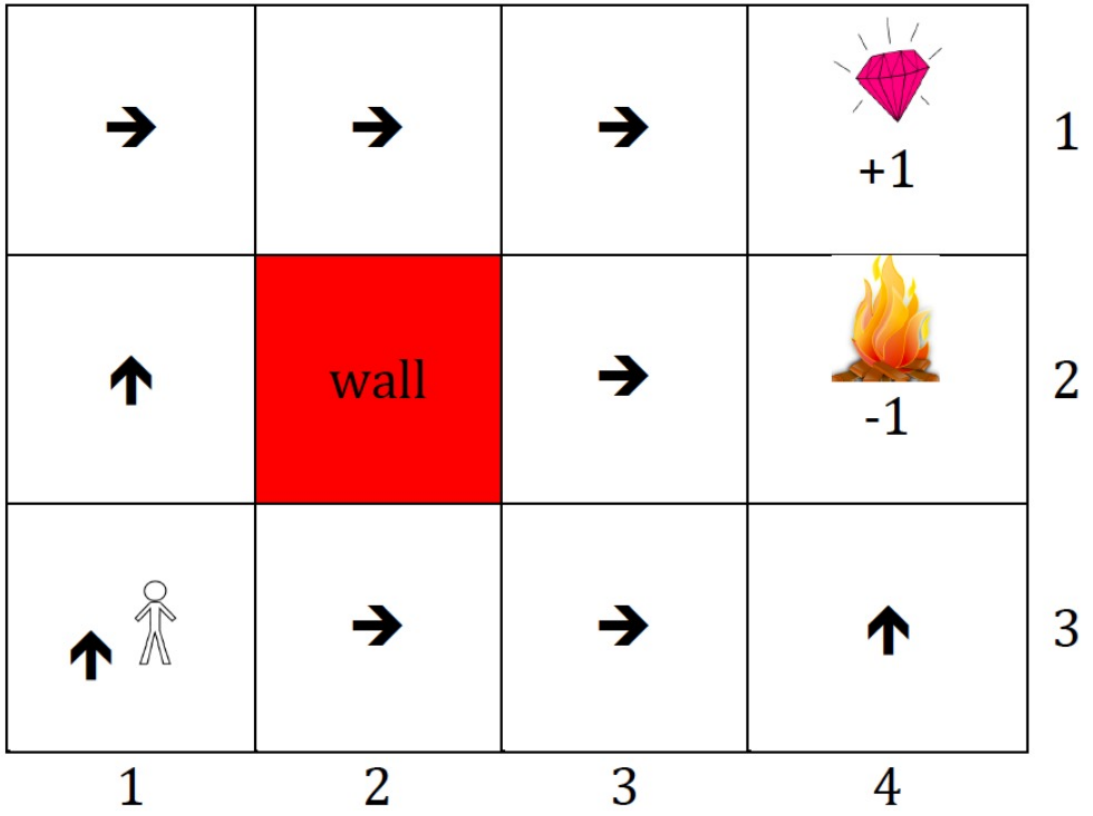
Value Iteration

```
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
```

Policy Iteration

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

- Faster: Iterate through policies instead of utilities.
- Terminate algorithm when no policy changes



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

- **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
```

Policy Iteration

Policy improvement:

For each state, can we pick a better action?

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

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:

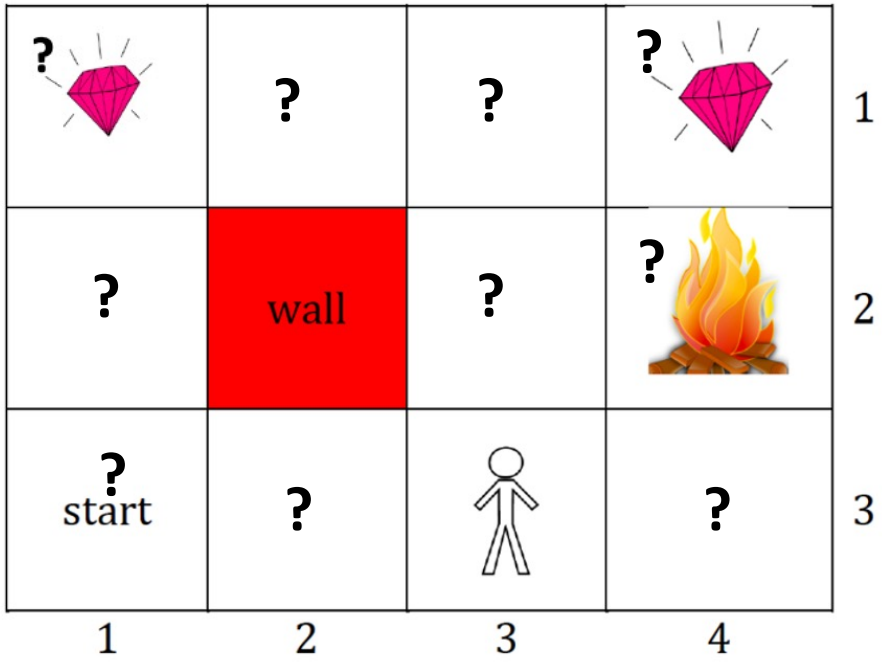
- Agent starts with policy π
- Learn how good policy is
- Adapt it and improve it

- Temporal difference
- Q-Learning

Active Reinforcement Learning

What if…

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

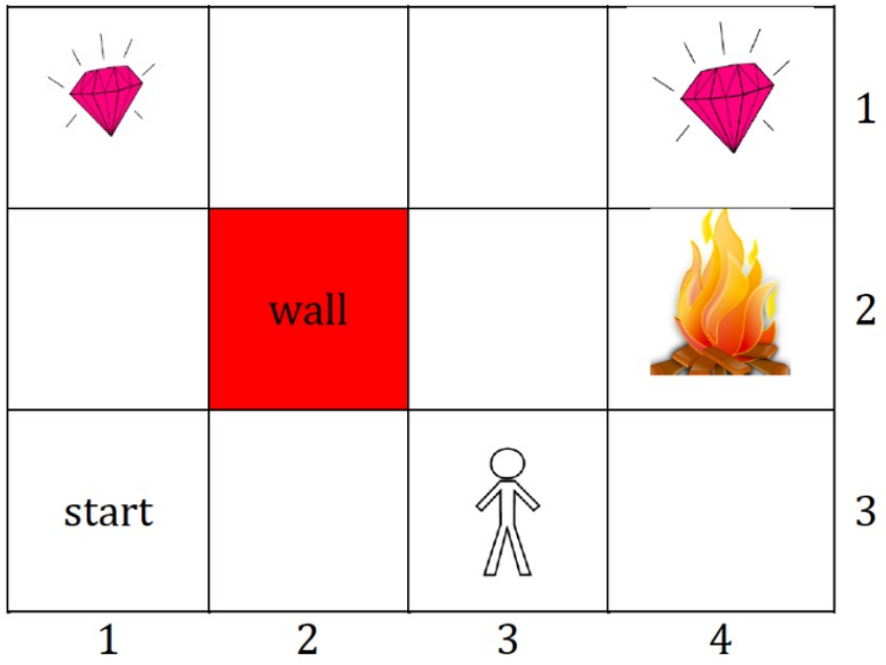


Reinforcement Learning

- Behavior reinforced on individual steps
- Reinforced at end
- Can learn model
- Can learn utility of decisions without the model

- Temporal difference
- Q-Learning

Reinforcement Learning – Exploration vs. Exploitation



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

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

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

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.

ϵ − greedy agent

- 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

$\epsilon -$ greedy is a way to force exploration

How could we incentivize it in the problem formulation?

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

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

$R^+ =$ 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

Q–learning

Temporal Difference:

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

Temporal Difference Q-learning:

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

Activity

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

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

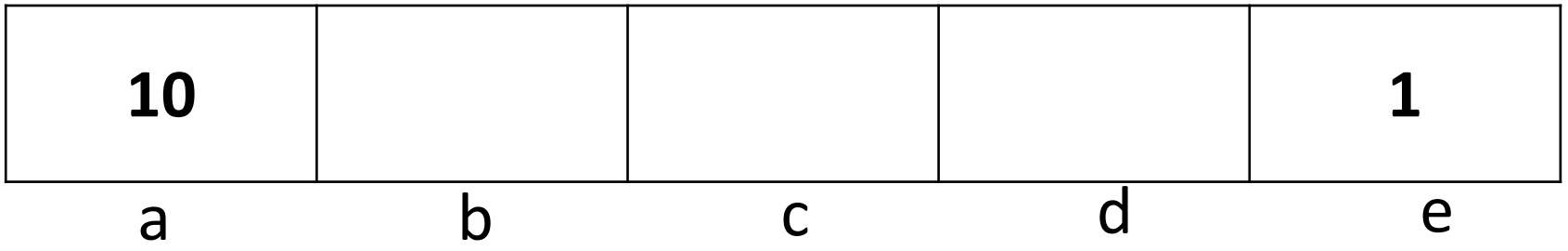
3. What is the difference between value iteration and policy iteration?

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

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.

- 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.
- $P(s'|a, s) = 0.80$ success and 0.20 that the agent stays in the same state.
- $R(s) = -0.04$



Policy Iteration

Example: (continued)

Policy Iteration

Example: (continued)