# Reinforcement learning

CS434

# Reinforcement Learning in a nutshell

Imagine playing a new game whose rules you don’t know; after a hundred or so moves, your opponent announces, “You lose”.

-Russell and Norvig

*Introduction to Artificial Intelligence*

# Reinforcement Learning

- Agent placed in an environment and must learn to behave optimally in it
- Assume that the world behaves like an MDP (satisfy the Markov Property):
  - Agent can act but does not know the transition model
  - Agent observes its current state, its reward but doesn't know the reward function
- Goal: learn an optimal policy

# Factors that Make RL Difficult

- Actions have non-deterministic effects
  - which are initially unknown and must be learned
- Rewards / punishments can be infrequent
  - Often at the end of long sequences of actions
  - How do we determine what action(s) were really responsible for reward or punishment? (credit assignment problem)
  - World is large and complex

# Importance of Credit Assignment

# Passive vs. Active learning

- Passive learning
  - The agent acts based on a fixed policy $\pi$ and tries to learn how good the policy is by observing the world go by
  - Analogous to policy evaluation in policy iteration
- Active learning
  - The agent attempts to find an optimal (or at least good) policy by exploring different actions in the world
  - Analogous to solving the underlying MDP

# Model-Based vs. Model-Free RL

- *Model based approach to RL:*
  - learn the MDP model (T and R), or an approximation of it
  - use it to find the optimal policy
- *Model free approach to RL:*
  - derive the optimal policy without explicitly learning the model

We will consider both types of approaches

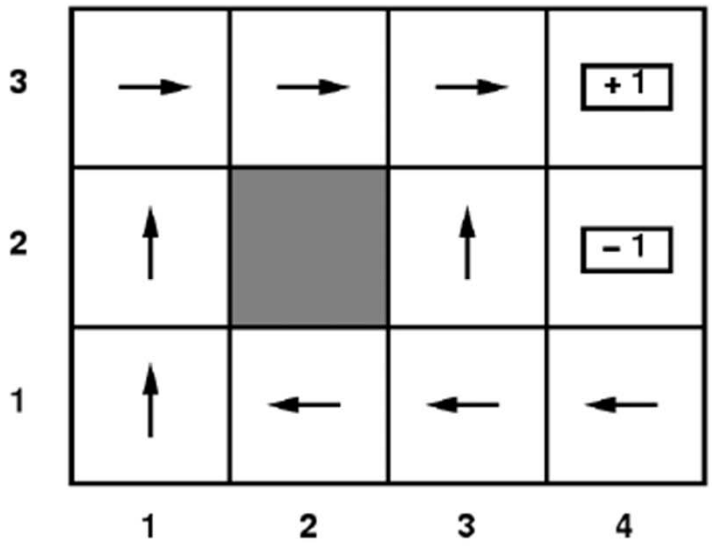
# Passive Reinforcement Learning

- Suppose agent's policy π is fixed
- It wants to learn how good that policy is in the world i.e. it wants to learn $U_\pi(s)$
- This is just like the policy evaluation part of policy iteration
- The big difference: the agent doesn't know the transition model or the reward function (but it gets to observe the reward in each state it is in)

# Passive RL

- Suppose we are given a policy
- Want to determine how good it is

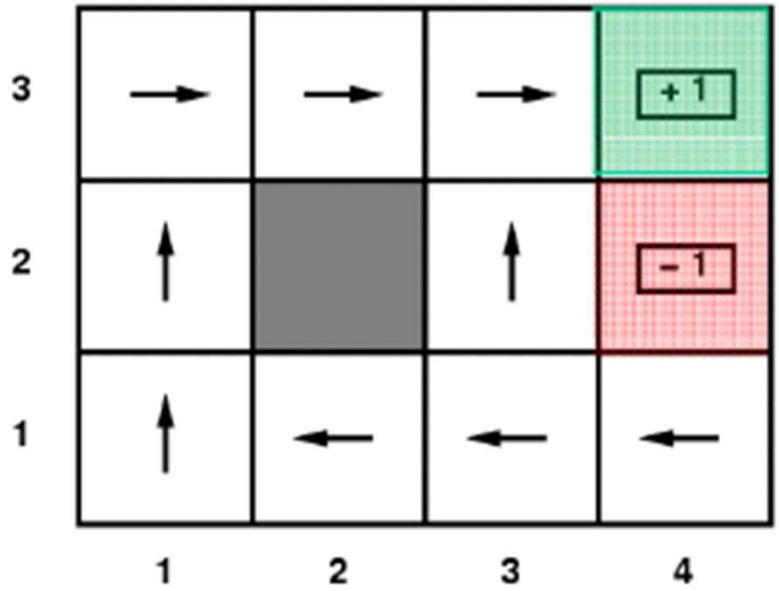
Given $\pi$:




Need to learn $U_\pi(S)$:

| | | | | |
|---|---|---|---|---|
| 3 | 0.812 | 0.868 | 0.918 | +1 |
| 2 | 0.762 | | 0.660 | -1 |
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |
| | 1 | 2 | 3 | 4 |

# Passive RL




- Given policy $\pi$,
  - estimate $U_{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!
- Simply follow the policy for many epochs
- Epochs: training sequences

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

# Adaptive Dynamic Programming (A Model based approach)

- Basically it learns the transition model **T** and the reward function **R** from the training sequences
- Based on the learned MDP (**T** and **R**) we can perform policy evaluation (which is part of policy iteration previously taught)

# Adaptive Dynamic Programming

- Recall that policy evaluation in policy iteration involves solving the utility for each state if policy $\pi_i$ is followed.
- This leads to the equations:

$$U_{\pi_i}(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_{\pi_i}(s')$$

- The equations above are linear, so they can be solved with linear algebra in time $O(n^3)$ where n is the number of states

# Adaptive Dynamic Programming

- Make use of policy evaluation to learn the utilities of states
- In order to use the policy evaluation eqn:

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_\pi(s')$$

the agent needs to learn the transition model T(s,a,s') and the reward function R(s)

***How do we learn these models?***

# Adaptive Dynamic Programming

- Learning the reward function R(s):

  Easy because it's deterministic. Whenever you see a new state, store the observed reward value as R(s)

- Learning the transition model T(s,a,s'):

  Keep track of how often you get to state s' given that you are in state s and do action a.

  – eg. if you are in s = (1,3) and you execute Right three times and you end up in s'=(2,3) twice, then T(s,Right,s') = 2/3.

# ADP Algorithm

**function** PASSIVE-ADP-AGENT(*percept*) **returns** an action

**inputs**: *percept*, a percept indicating the current state *s'* and reward signal *r'*

**static**: $\pi$, a fixed policy

*mdp*, an MDP with model T, rewards R, discount γ

*U*, a table of utilities, initially empty

$N_{sa}$ a table of frequencies for state-action pairs, initially zero

$N_{sas'}$, a table of frequencies for state-action-state triples, initially zero

*s*, *a* the previous state and action, initially null

**if** *s'* is new **then do** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$

**if** *s* is not null, **then do**

increment $N_{sa}[s,a]$ and $N_{sas'}[s,a,s']$

for each *t* such that $N_{sas'}[s,a,t]$ is nonzero do

$T[s,a,t] \leftarrow N_{sas'}[s,a,t] \,/\, N_{sa}[s,a]$

$U \leftarrow$ POLICY-EVALUATION($\pi$, *U*, *mdp*)

**if** TERMINAL?[*s'*] then *s*, *a* ← null **else** *s*, *a* ← *s'*, $\pi[s']$

**return** a

Update reward function

Update transition model

# The Problem with ADP

- Need to solve a system of simultaneous equations – costs $O(n^3)$
  - Very hard to do if you have $10^{50}$ states like in Backgammon
- Can we avoid the computational expense of full policy evaluation?

# Temporal Difference Learning

- Instead of calculating the exact utility for a state can we approximate it and possibly make it less computationally expensive?
- Yes we can! Using Temporal Difference (TD) learning

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_\pi(s')$$

- Instead of doing this sum over all successors, only adjust the utility of the state based on the successor observed in the trial.
- It does not estimate the transition model – model free

# TD Learning

Example:

- Suppose you see that $U_\pi(1,3) = 0.84$ and $U_\pi(2,3) = 0.92$ after the first trial.
- If the transition $(1,3) \rightarrow (2,3)$ happens all the time, you would expect to see (assuming $\gamma = 1$):

  $U_\pi(1,3) = R(1,3) + U_\pi(2,3)$

  $\Rightarrow U_\pi(1,3) = -0.04 + U_\pi(2,3)$

  $\Rightarrow U_\pi(1,3) = -0.04 + 0.92 = 0.88$
- Since you observe $U_\pi(1,3) = 0.84$ in the first trial, it is a little lower than 0.88, so you might want to "bump" it towards 0.88.

# Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers
  - E.g. to estimate the mean of a r.v. from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n+1}\left( x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i \right)$$

$$= \hat{X}_n + \frac{1}{n+1}\left(x_{n+1} - \hat{X}_n\right)$$

average of n+1 samples

learning rate

sample n+1

- Given a new sample x(n+1), the new mean is the old estimate (for n samples) plus the weighted difference between the new sample and old estimate

# Temporal Difference Learning (TD)

- TD update for transition from s to s':

$$U_\pi(s) = U_\pi(s) + \alpha(R(s) + \gamma U_\pi(s') - U_\pi(s))$$

learning rate

New (noisy) sample of utility based on next state

- So the update is maintaining a "mean" of the (noisy) utility samples
- If the learning rate decreases with the number of samples (e.g. 1/n) then the utility estimates will eventually converge to true values!

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_\pi(s')$$

# Temporal Difference Update

When we move from state s to s', we apply the following update rule:

$$U_\pi(s) = U_\pi(s) + \alpha(R(s) + \gamma U_\pi(s') - U_\pi(s))$$

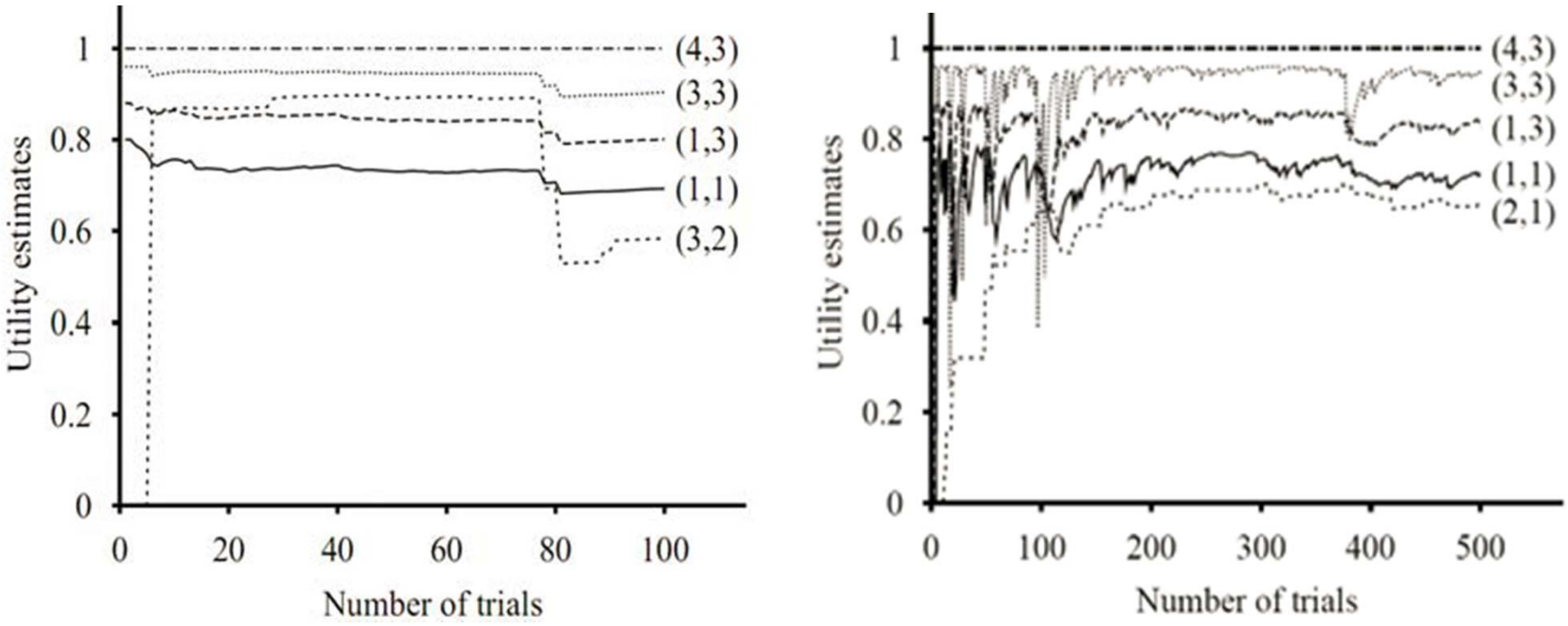
This is similar to one step of value iteration

We call this equation a "backup"

# Convergence

- Since we're using the observed successor s' instead of all the successors, what happens if the transition s $\rightarrow$ s' is very rare and there is a big jump in utilities from s to s'?
- How can $U_\pi(s)$ converge to the true equilibrium value?
- Answer: The average value of $U_\pi(s)$ will converge to the correct value
- This means we need to observe enough trials that have transitions from s to its successors
- Essentially, the effects of the TD backups will be averaged over a large number of transitions
- Rare transitions will be rare in the set of transitions observed

# ADP and TD




Learning curves for the 4x3 maze world, given the optimal policy

Which figure is ADP?

# Comparison between ADP and TD

- Advantages of ADP:
  - Converges to the true utilities faster
  - Utility estimates don't vary as much from the true utilities
- Advantages of TD:
  - Simpler, less computation per observation
  - Crude but efficient first approximation to ADP
  - Don't need to build a transition model in order to perform its updates (this is important because we can interleave computation with exploration rather than having to wait for the whole model to be built first)

# Passive learning

- Learning $U_{\pi}(s)$ does not lead to a optimal policy, why?
- the models are incomplete/inaccurate
- the agent has only tried limited actions, we cannot gain a good overall understanding of *T*
- This is why we need active learning

# Goal of active learning

- Let's first assume that we still have access to some sequence of trials performed by the agent
  - The agent is not following any specific policy
  - We can assume for now that the sequences should include a thorough exploration of the space
  - We will talk about how to get such sequences later

- The goal is to learn an optimal policy from such sequences

# Active Reinforcement Learning Agents

We will describe two types of Active Reinforcement Learning agents:

- Active ADP agent
- Q-learner (based on TD algorithm)

# Active ADP Agent (Model-based)

- Using the data from its trials, the agent learns a transition model $\hat{T}$ and a reward function $\hat{R}$
- With $\hat{T}$(s,a,s') and $\hat{R}$(s), it has an estimate of the underlying MDP
- It can compute the optimal policy by solving the Bellman equations using value iteration or policy iteration

$$U(s) = \hat{R}(s) + \gamma \max_a \sum_{s'} \hat{T}(s,a,s')U(s')$$

- If $\hat{T}$ and $\hat{R}$ are accurate estimation of the underlying MDP model, we can find the optimal policy this way

# Issues with ADP approach

- Need to maintain MDP model
- $T$ can be very large $\mathrm{O}(|S|^2 \times |A|)$
- Also, finding the optimal action requires solving the bellman equations – time consuming
- Can we avoid this large computational complexity both in terms of time and space?

# Q-learning

So far, we have focused on the utilities for states

- U(s) = utility of state s = expected maximum future rewards

An alternative is to store Q-values, which are defined as:

- Q(*a*,*s*) = utility of taking action *a* at state *s*

  = expected maximum future reward if action *a* at state *s*

- Relationship between U(s) and Q(*a*, *s*)?

$$U(s) = \max_a Q(a, s)$$

# Q-learning can be model free

- Note that after computing U(*s*), to obtain the optimal policy, we need to compute:

$$\pi(s) = \max_a \sum_{s'} T(s,a,s')U(s')$$

  - This requires T, the model of world
  - So even if we use TD learning (model free), we still need the model to get the optimal policy
- However, if you successfully estimate Q(*a*,*s*) for all *a* and *s*, we can compute the optimal policy without using the model:

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

# Q-learning

At equilibrium when the Q-values are correct, we can write the constraint equation:

$$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s')U(s')$$

Reward at state s

Expected value for action-state pair (a, s)

Expected value averaged over all possible states s' that can be reached from s after executing action a

# Q-learning

At equilibrium when the Q-values are correct, we can write the constraint equation:

$$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s')$$

Reward at state s

Best expected value for action-state pair (a, s)

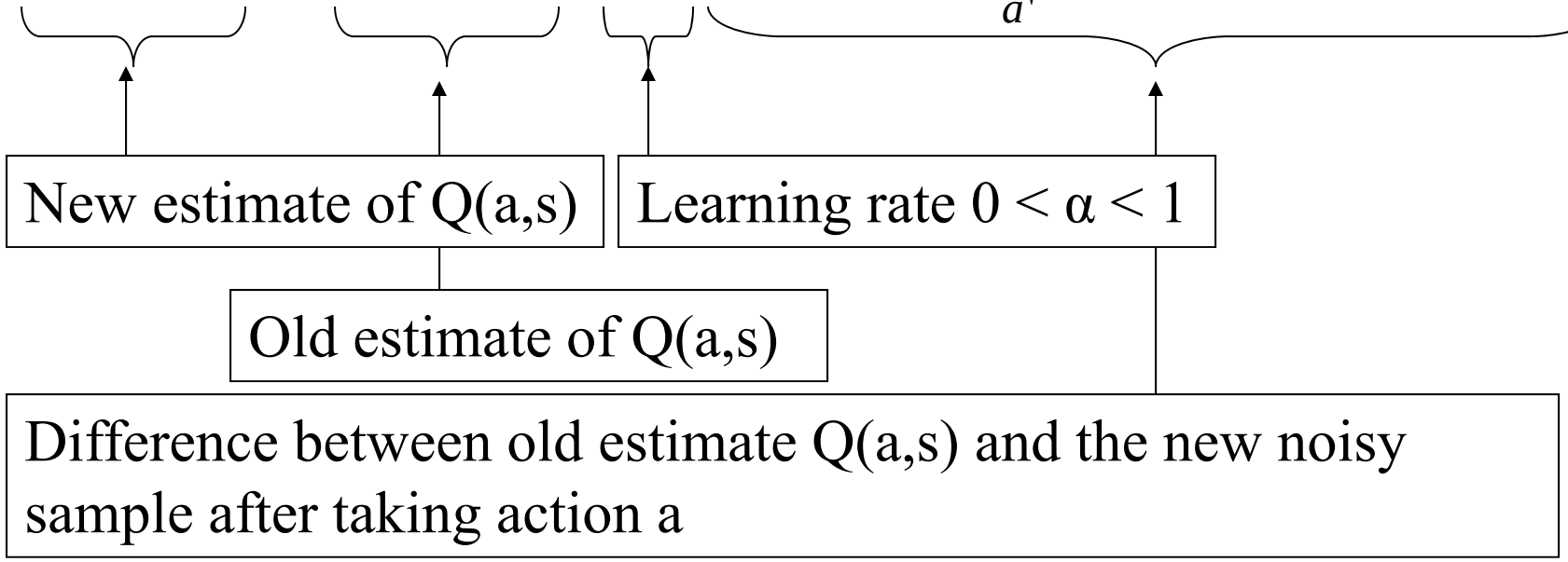
Best value averaged over all possible states s' that can be reached from s after executing action a

Best value at the next state = max over all actions in state s'

# Q-learning Without a Model

- We can use a temporal differencing approach which is model-free
- After moving from state s to state s' using action a:

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



# Q-learning: Estimating the Policy

Q-Update: After moving from state s to state s' using action a:

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

Note that T(s,a,s') does not appear anywhere!

Further, once we converge, the optimal policy can be computed without T.

This is a completely model-free learning algorithm.

# Q-learning Convergence

- Guaranteed to converge to the true Q values given enough exploration
- Very general procedure (because it's model free)
- Converges slower than ADP agent (because it is completely model free and it doesn't enforce consistency among values through the model)


0.59
0.58
0.70
0.54
0.66
0.60
0.78
0.66
0.64
0.68
0.88
0.58
1.00
0.61
0.52
0.52
0.49
-0.51
0.42
-0.43
0.15
-1.00
0.54
0.45
0.43
0.47
0.40
0.48
0.33
0.42
0.31
0.39
0.28
0.27
-0.89
0.28
-0.38
0.11
Q-VALUES AFTER 1000 EPISODES

- So far, we have assumed that all training sequences are given and they fully explore the state space and action space
- But how do we generate all the training trials?
  - We can have the agents random explore first, to collect training trials
  - Once we accumulate enough trials, we perform the learning (eith ADP, or Q-learning)
  - We then choose the optimal policy
- How much exploration do we need to do?
- What if the agent is expected to learn and perform reasonably constantly, not just at the end

# A greedy agent

- At any point, the agent has a current set of training trials, and we've got a policy that is "optimal" based on our current understanding of the world
- A greedy agent can execute the optimal policy for the learned model at each time step

# A greedy Q-learning agent

**function** Q-learning-agent(*percept*) **returns** an action

**inputs**: *percept*, a percept indicating the current state *s'* and reward signal *r'*

**static**: *Q*, a table of action values index by state and action

$N_{sa}$ a table of frequencies for state-action pairs, initially zero

*s*, *a*, *r* the previous state and action, initially null

**if** *s* is not null, **then do**

increment $N_{sa}[s,a]$

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

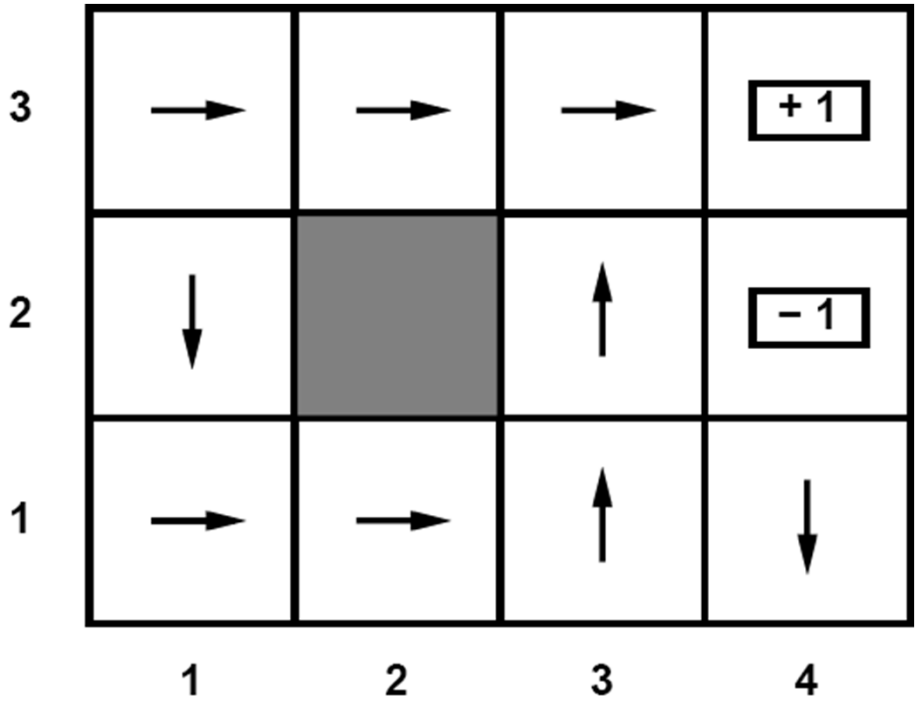
**if** TERMINAL?[*s'*] then *s*, *a* ← null

**else** *s*, *a*, *r* ← *s'*, $\arg\max_{a} Q(s',a)$ , *r'*

**return** a

Always choose the action that is deemed the best based on current Q table

# The Greedy Agent




The agent finds the lower route to get to the goal state but never finds the optimal upper route. The agent is stubborn and doesn't change so it doesn't learn the true utilities or the true optimal policy

# What happened?

- How can choosing an optimal action lead to suboptimal results?
- What we have learned (T/R, or Q) may not truly reflect the true environment
- In fact, the set of trials observed by the agent was often insufficient

**How can we address this issue?**

**We need good training experience …**

# Exploitation vs Exploration

- Actions are always taken for one of the two following purposes:
  - Exploitation: Execute the current optimal policy to get high payoff
  - Exploration: Try new sequences of (possibly random) actions to improve the agent's knowledge of the environment even though current model doesn't believe they have high payoff
- Pure exploitation: gets stuck in a rut
- Pure exploration: not much use if you don't put that knowledge into practice

# Optimal Exploration Strategy?

- What is the optimal exploration strategy?
  - Greedy?
  - Random?
  - Mixed? (Sometimes use greedy sometimes use random)
- It turns out that the optimal exploration strategy has been studied in-depth in the N-armed bandit problem

# N-armed Bandits

- We have N slot machines, each can yield $1 with some probability (different for each machine)

- What order should we try the machines?
  - Stay with the machine with the highest observed probability so far?
  - Random?
  - Something else?
- Bottom line:
  - It's not obvious
  - In fact, an exact solution is usually intractable

# GLIE

- Fortunately it is possible to come up with a **reasonable** exploration method that eventually leads to optimal behavior by the agent
- Any such exploration method needs to be Greedy in the Limit of Infinite Exploration (GLIE)
- Properties:
  - Must try each action in each state an unbounded number of times so that it doesn't miss any optimal actions
  - Must eventually become greedy

# Examples of GLIE schemes

- $\varepsilon$-greedy:
  - Choose optimal action with probability (1-$\varepsilon$)
  - Choose a random action with probability $\varepsilon$/(number of actions-1)
- Active ε-greedy agent
  1. Start from the original sequence of trials
  2. Compute the optimal policy under the current understanding of the world
  3. Take action use the ε-greedy exploitation-exploration strategy
  4. Update learning, go to 2

# Another approach

- Favor actions the agent has not tried very often, avoid actions believed to be of low utility (based on past experience)
- We can achieve this using an ***exploration function***

# An exploratory Q-learning agent

**function** Q-learning-agent(*percept*) **returns** an action

**inputs**: *percept*, a percept indicating the current state *s'* and reward signal *r'*

**static**: *Q*, a table of action values index by state and action

$N_{sa}$ a table of frequencies for state-action pairs, initially zero

*s*, *a*, *r* the previous state and action, initially null

**if** *s* is not null, **then do**

increment $N_{sa}[s,a]$

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

**if** TERMINAL?[*s'*] then *s*, *a* ← null

**else** *s*, *a*, *r* ← *s'*, $\arg\max_{a'} f(Q(a',s'), N_{sa}[s',a'])$, *r'*

**return** a

Exploration function:

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

# Exploration Function

Exploration function f(q,n):

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

- Trades off greedy (preference for high utilities q) against curiosity (preference for low values of n – the number of times a state-action pair has been tried)
- R+ is an optimistic estimate of the best possible reward obtainable in any state with any action
- If *a* hasn't been tried enough in s, you assume it will somehow lead to *gold* – optimistic
- $N_e$ is a limit on the number of tries for a state-action pair

# Model-based/Model-free

- Two broad categories of reinforcement learning algorithms:
  1. Model-based eg. ADP
  2. Model-free eg. TD, Q-learning
- Which is better?
  - Model-baesed approach is a knowledge-based approach (ie. model represents known aspects of the environment)
  - Book claims that as environment becomes more complex, a knowledge-based approach is better