



CS 5/7320 Artificial Intelligence

Reinforcement Learning AIMA Chapter 17+22

Slides by Michael Hahsler
with figures from the AIMA textbook.



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

Remember Chapter 16: Making Simple Decisions

For a decision that we make frequently and making it once does not affect the future decisions (**episodic environment**), we can use the **Principle of Maximum Expected Utility (MEU)**.

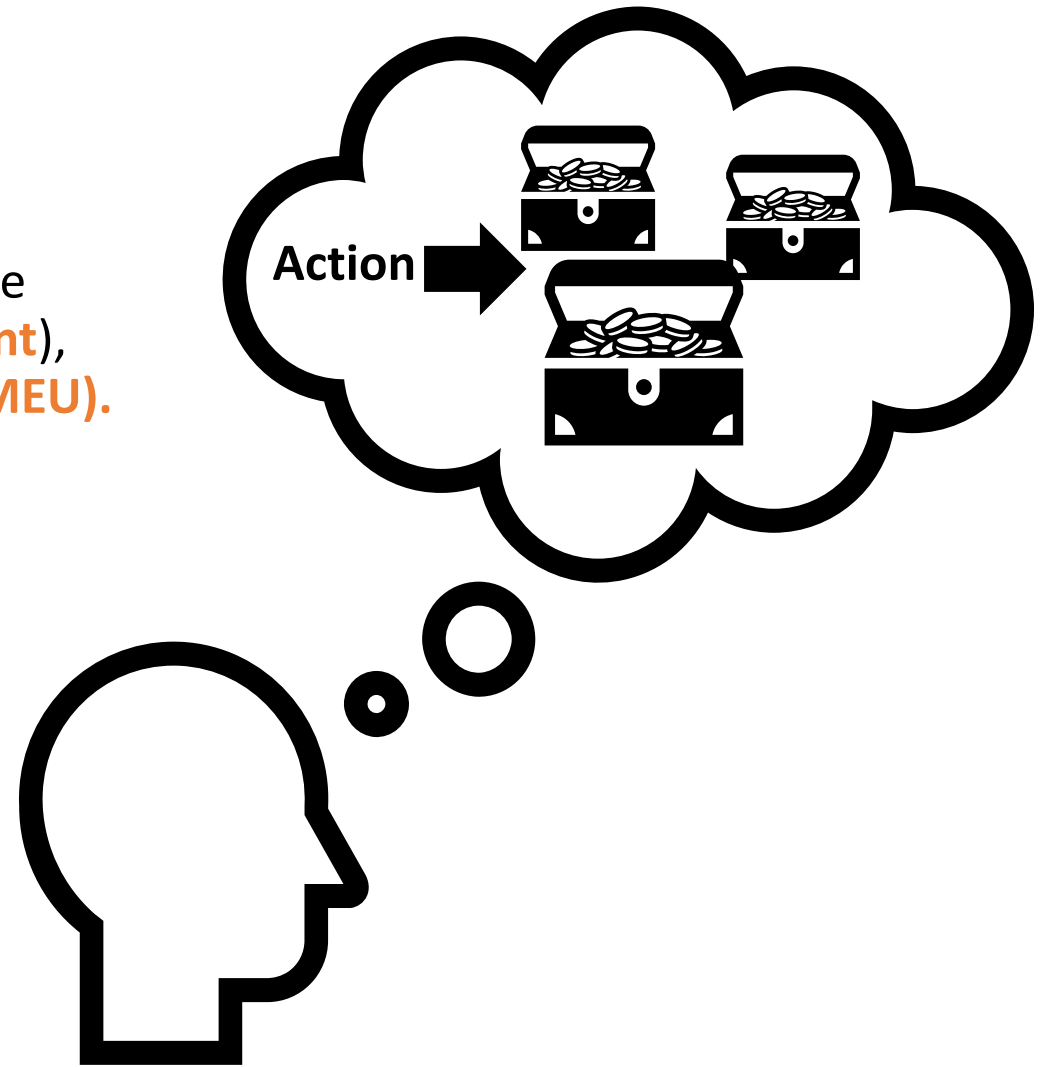
Given the expected utility of an action

$$EU(a) = \sum_{s'} P(Result(a) = s')U(s')$$

choose action that maximizes the expected utility:

$$a^* = \operatorname{argmax}_a EU(a)$$

Now we will talk about decision making in **sequential environments**.



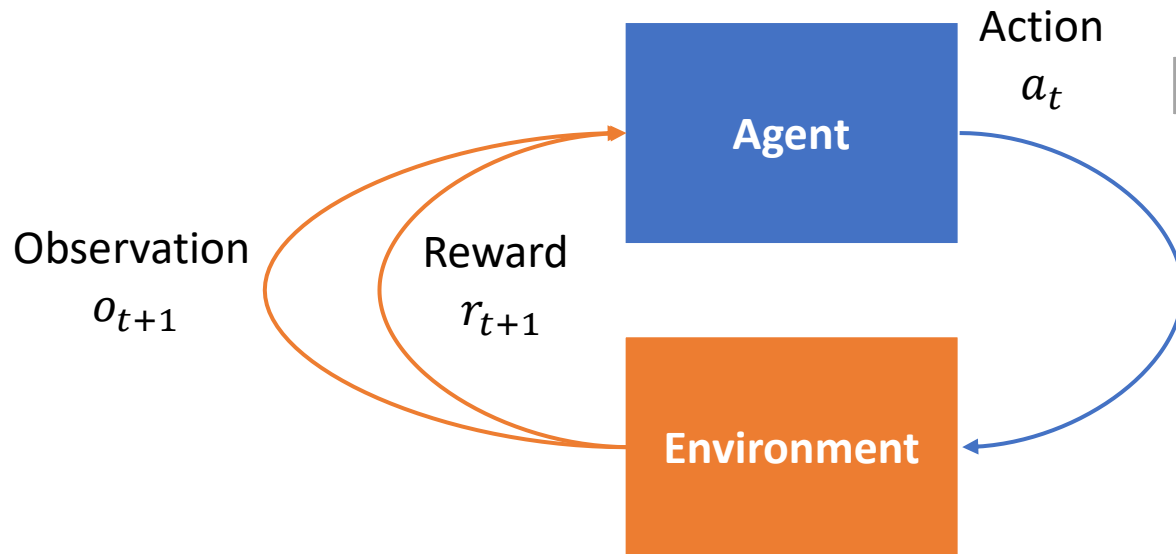


Sequential Decision Problems

AIMA Chapter 17: Making Complex Decisions

Sequential Decision Problems

- **Utility-based agent:** The agent's utility depends on a sequence of decisions spread out over time.
- Sequential decision problems incorporate utilities, uncertainty, and sensing.



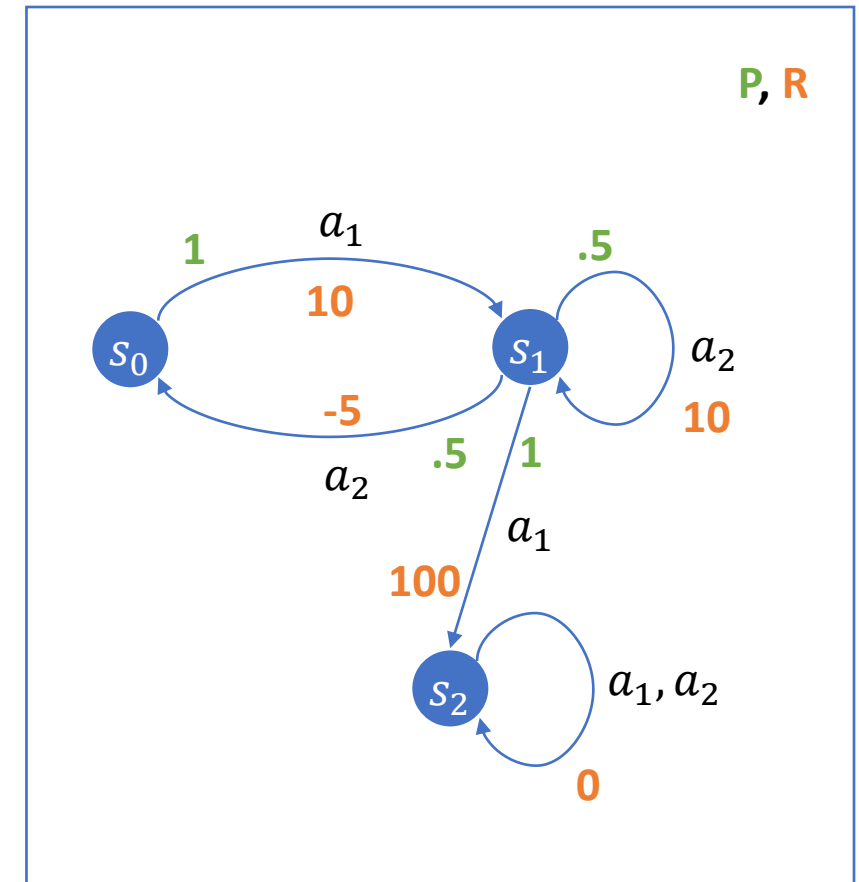
Sequence: $(o_0, r_0), a_0, (o_1, r_1), a_1, (o_2, r_2), a_2, \dots$

Goal: Observations and rewards depend on the state of the system and the agent wants to maximize expected reward over time (discounted by γ):

$$U = E \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

Definition: Markov Decision Process (MDP)

- MDPs are sequential decision problems with
 - a fully observable ($o_t = s_t$), stochastic environment;
 - a Markovian transition model: future states do not depend on past states given the current state;
 - additive rewards.
- MDPs are discrete-time stochastic control processes defined by:
 - a finite set of **states** $S = \{s_0, s_1, s_2, \dots\}$ (initial state s_0)
 - a set of available **actions** $ACTIONS(s)$ in each state s
 - a **transition model** $P(s' | s, a)$ where $a \in ACTIONS(s)$
 - a **reward function** $R(s)$ where the reward depends on the current state (often $R(s, a, s')$ is used)
- Time horizon
 - **Infinite horizon**: non-episodic task environment
 - **Finite horizon**: episodic task environment. Episode ends after a number of periods or when a terminal state is reached. Episodes contain a sequence of several actions that affect each other.



Example: 4x3 Grid World

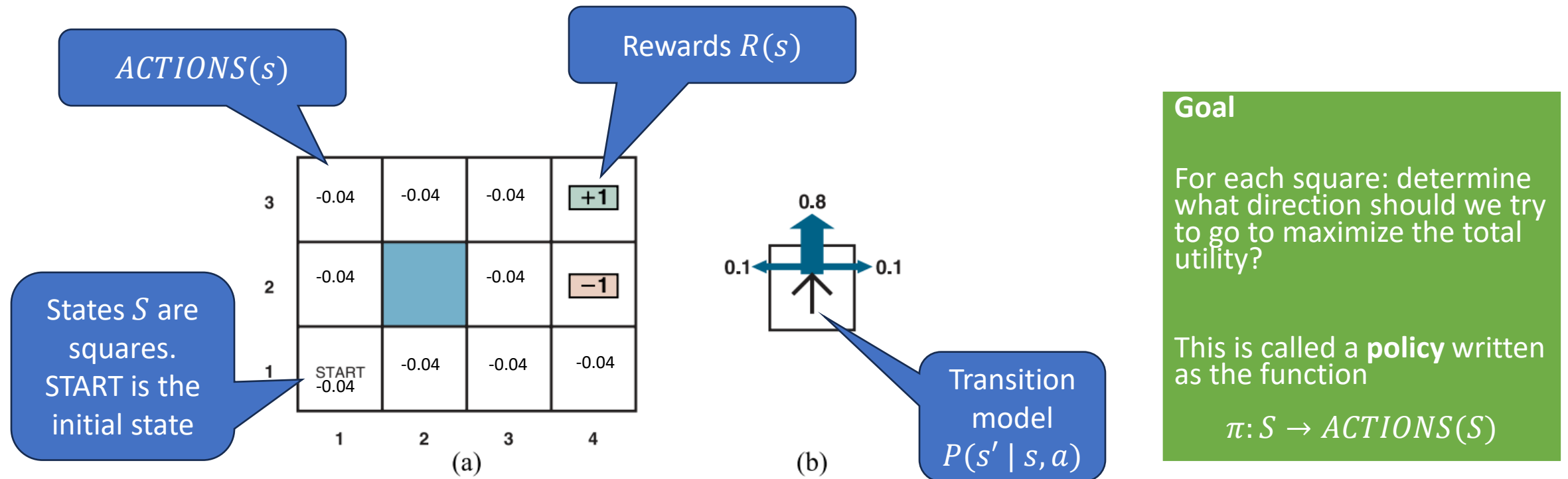


Figure 17.1 (a) A simple, stochastic 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the “intended” outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. Transitions into the two terminal states have reward +1 and -1, respectively, and all other transitions have a reward of -0.04.

Optimal Policy

- A policy $\pi = \{\pi(s_0), \pi(s_1), \dots\}$ defines for each state which action to take.
- The expected utility of being in state s under policy π can be calculated as the sum:

$$U^\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s \right]$$

γ is a discounting factor to give more weight to immediate rewards.

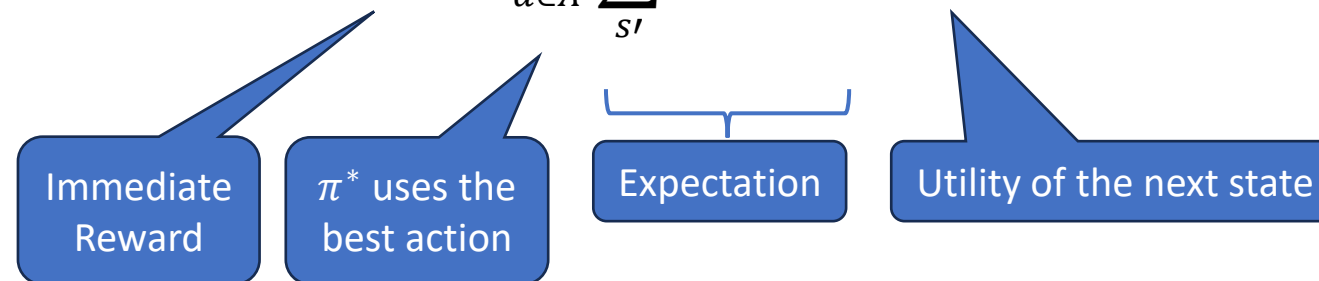
E_π is the expectation over sequences that can be created by following π .

- The goal of solving an MDP is to find an optimal policy π that maximizes the expected future utility for each state

$$\pi^*(s) = \operatorname{argmax}_{\pi} U^\pi(s) \quad \text{for all } s \in S$$

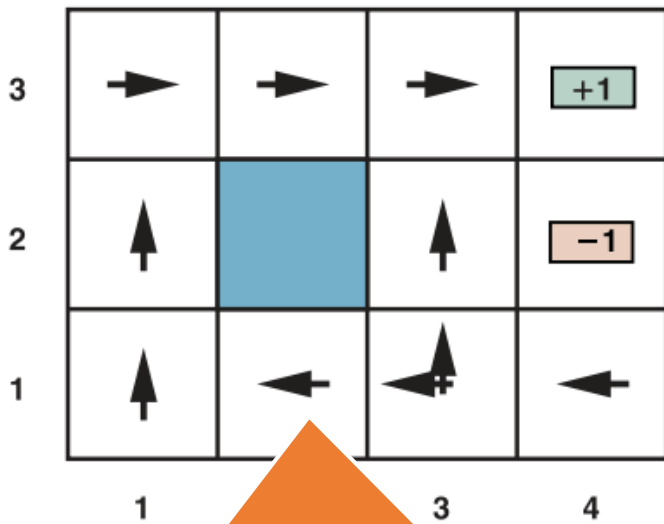
- The recursive **Bellman equation** holds for the optimal value function U (“Bellman optimality condition”):

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



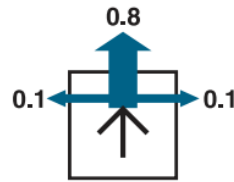
Solution: 4x3 Grid World

Optimal action in each state
(policy π^*)

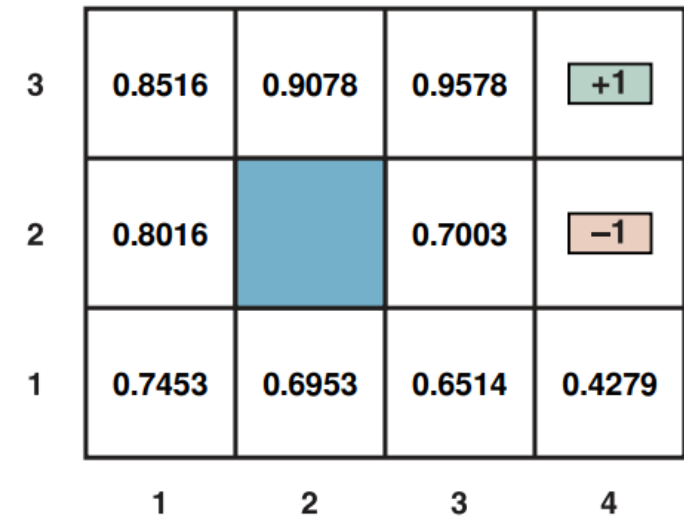


Why is it optimal to walk away from the +1 square?

Always pick the action with the highest expected utility.



Value of being in a state $U(s)$
(given that we will follow π^*)



$\gamma = 1$

How to we find the optimal value function/optimal policy?

Policy Iteration

Value Iteration

Q-Function

- $Q(s, a)$ is called the state-action value function. It gives the expected utility of action a in state s .

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

Immediate
Reward

Expected utility of the
next state

- Relationship with the state value function: $U(s) = \max_a Q(s, a)$
- The Q-function is often used for convenience in solving MDPs.

Value Iteration: Estimate the Optimal Value Function U^{π^*}

Algorithm: Start with a $U(s)$ vector of 0 for all states and then update (Bellman update) the vector iteratively until it converges to the unique optimal solution U^{π^*} .

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function

inputs: mdp , an MDP with states S , actions $A(s)$, transition model $P(s' | s, a)$,
rewards $R(s, a, s')$, discount γ

ϵ , the maximum error allowed in the utility of any state

local variables: U, U' , vectors of utilities for states in S , initially zero

δ , the maximum relative change in the utility of any state

repeat

$U \leftarrow U'; \delta \leftarrow 0$

for each state s **in** S **do**

$U'[s] \leftarrow \max_{a \in A(s)} \text{Q-VALUE}(mdp, s, a, U)$

if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$

until $\delta \leq \epsilon(1 - \gamma)/\gamma$

return U

Update with the value of
the best action in state s .

Uses a proxy for policy loss
 $\|U^{\pi} - U\|_{\infty}$ as the stopping criterion

U converges to U^{π^*}
and we can extract π^*

Policy Iteration: Learn the Optimal Policy π^*

Policy iteration tries to directly find the optimal policy by iterating policy evaluation and improvement.

function POLICY-ITERATION(mdp) **returns** a policy

inputs: mdp , an MDP with states S , actions $A(s)$, transition model $P(s' | s, a)$

local variables: U , a vector of utilities for states in S , initially zero

π , a policy vector indexed by state, initially random

repeat

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp)$

$unchanged? \leftarrow \text{true}$

for each state s **in** S **do**

$a^* \leftarrow \underset{a \in A(s)}{\text{argmax}} \text{Q-VALUE}(mdp, s, a, U)$

if $\text{Q-VALUE}(mdp, s, a^*, U) > \text{Q-VALUE}(mdp, s, \pi[s], U)$ **then**

$\pi[s] \leftarrow a^*$; $unchanged? \leftarrow \text{false}$

until $unchanged?$

return π

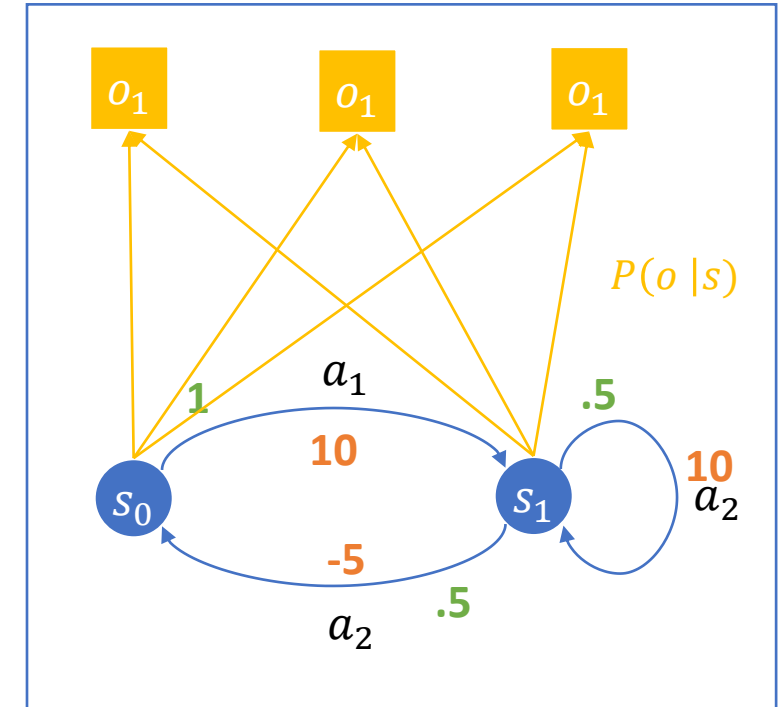
Calculate U given current policy
(either solve an LP or value iteration
with fixed policy)

Greedy policy
Improvement

π converges to π^*
(and U converges to U^{π^*})

Partially Observable Markov Decision Model (POMDP)

- If the environment is **partially observable**, then $o_t \neq s_t$ and the model is expanded by
 - a **sensor model** $P(o | s)$ for receiving observation o given being in state s .
- This makes things a lot more complicated, and we have to work with **belief states**. A belief state is a distribution over states.
Example: For a problem with three states, the belief state $b = (.2, .8, 0)$ means the agent beliefs that it is with 20% in state 1 and 80% in state 2 but not in state 3.
- An MDP that uses belief states instead of system states is called a **belief MDP**.
Issue: the probabilities in belief states are continuous, and the number of different belief states is infinite.
- The solution of a POMDP is a policy with the optimal actions for sets of belief states (i.e., ranges of belief).
- For all but tiny problems, POMDPs can only be solved **approximately** (e.g., by grid-based methods).





Reinforcement Learning

AIMA Chapter 22

Reinforcement Learning (RL)

- RL assumes that the problem can be modeled by an **MDP**.
- What if we do not know the exact transition model $P(s' \mid s, a)$?

Now we cannot solve the MDP (estimate the state utility function/policy) because we cannot predict what the future states after an action are!

- The agent needs to explore (try actions) and **use the reward signal to update its estimate of the utility of states and actions**. This is a learning process where the reward provides positive reinforcement.

Q-Learning

- Q-Learning learns the state-action value function from interactions with the environment (percepts).
- This agent function learns a table for the state-action function Q .

function Q-LEARNING-AGENT(*percept*) **returns** an action

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

persistent: Q , a table of action values indexed by state and action, initially zero

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

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

New episode
has no s .

if s is not null **then**

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

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

$s, a \leftarrow s', \operatorname{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a'])$

return a

Learning rate

Make $Q[s, a]$ a little more similar to the received reward + the best Q-value of the successor state.

f is the exploration function and decides on the next action. As N increases it can exploit good actions more.

Value Function Approximation

- $U(Q)$ needs to store and estimate one entry for each state (state/action combination).
- Issues and solutions
 - Too many entries to store → lossy compression
 - Many combinations are rarely seen → generalize to unseen entries
- **Idea:** Estimate the state value by learning an approximation function $\hat{U}(s) = g_{\theta}(s)$ based on features of s .
- 4x3 Grid World Example: Use a linear combination of state features (x, y) and learn θ from observed data.

3	0.8516	0.9078	0.9578	+1
2	0.8016		0.7003	-1
1	0.7453	0.6953	0.6514	0.4279
	1	2	3	4

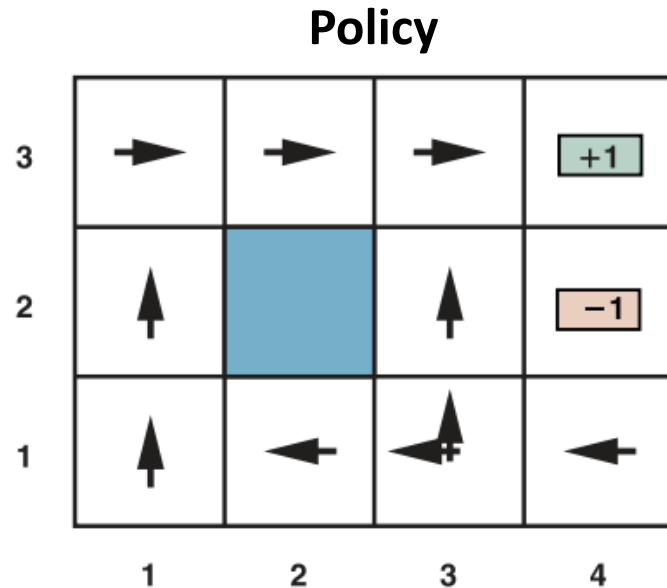
Learn θ from observed interactions with the environment to approximate $U(s)$

$$\hat{U}_{\theta}(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

Notes:

- θ can be updated iteratively after each new observed utility.
- We typically need non-linear approximators that can be incrementally updated (online learning). → Deep ANNs called Value Net

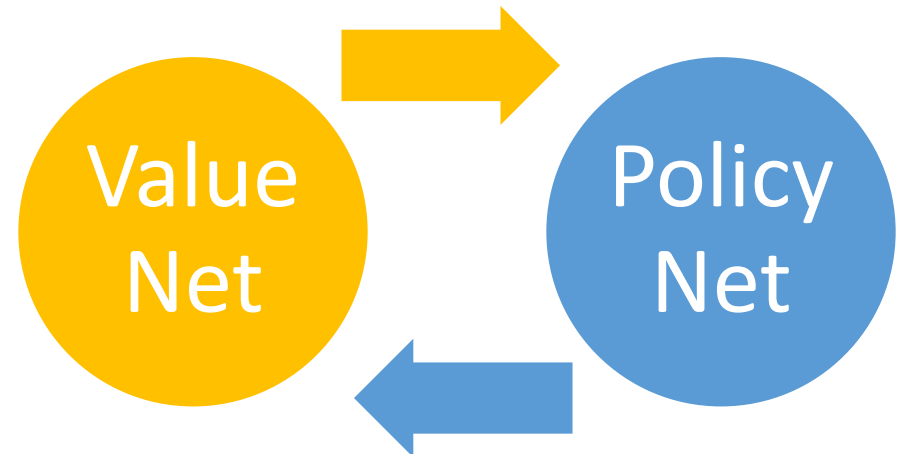
Policy Nets: Policy Approximation



- Policy tables are also large and generalization may be helpful.
- Use a neural network to learn to represent a policy as a function

$$f: S \rightarrow A.$$

- Often used together with a Value Net to iteratively improve the Value Net and then the Policy Net.





Summary

- Agents can learn the value of being in a state from **reward signals**.
- Rewards can be delayed (e.g., at the end of a game).
- Not being able to fully **observe the state** makes the problem more difficult (POMDP).
- **Unknown transition models** lead to the need of exploration by trying actions (model free methods like Q-Learning).
- All these problems are computationally very expensive and often can only be solved by **approximation**. State of the art is to use deep artificial neural networks for function approximation.