Technische Hochschule Rosenheim

Modul - Unsupervised and Reinforcement Learning (URL)

Bachelor Programme AAI

09 - Markov Decision Processes (MDP)

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Agenda

> Code can be found in
>
> - RL_Policy_Optimization.ipynb
> - or on GitHub or hosted on myBinder

On the menu for today:

- Reinforcement Learning
  - Agent, Rewards, States and Actions
  - Markov Decision Processes
  - Bellmann equation/function
  - Policy Evaluation, Improvement and Iteration

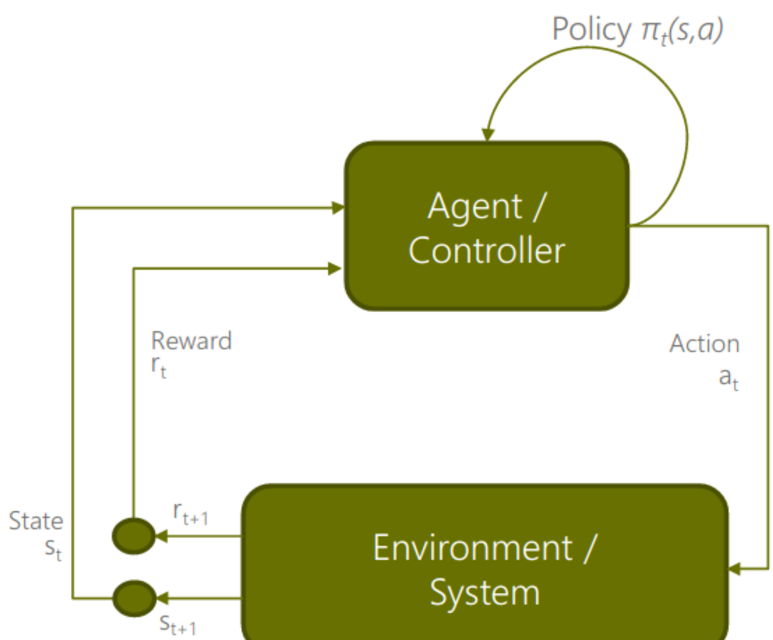
Agent and Environment

At each step the agent

- receives state observations S_t
- receives scalar reward R_t
- executes action A_t

The environment

- receives action A_t
- emits state observation S_{t+1}
- emits scalar reward R_{t+1}

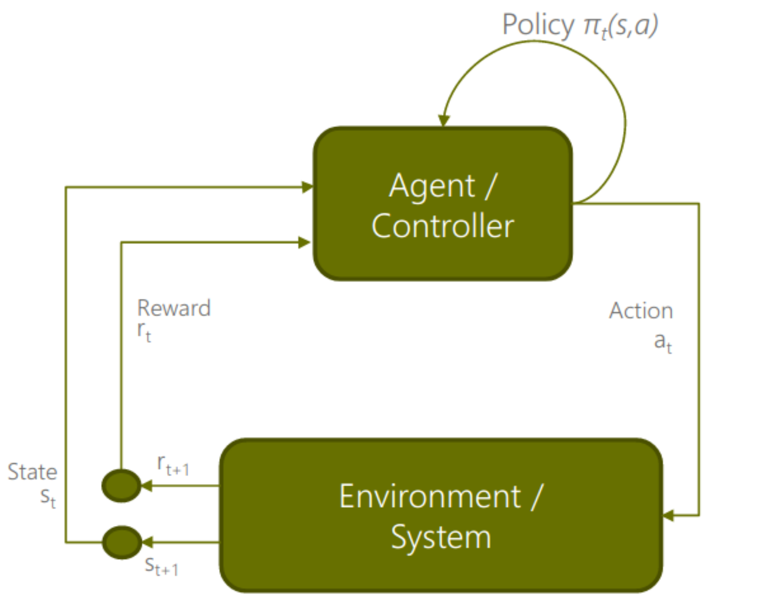


RL Agent: Policy

Telling the agent what to do:

- Deterministic $a = \pi_t(s)$
- Stochastic
  $\pi_t(s,a) = Pr(a_t = a | s_t = s)$

Given the situation at time *t* is state *s*, the policy gives the probability the agent's action will be *a*.



Goal is to find the policy(π)that maximizes rewards!

Deterministic Policy

Example: Frozenlake

```
# Make the environment based on deterministic policy (is_slippery=False)
env = gym.make('FrozenLake-v1', is_slippery=False, render_mode="ansi")
# Go right once (action = 2)
env.reset()
action = 2
(observation, reward, done, prob, a) = env.step(action)
print(env.render())
# Observation = 0: move to the right once from grid 0 to grid 1
# Prob = 1: deterministic policy, if we choose to go right, we will go right
print(observation, reward, done, prob, a)
```

```
                     (Right)
[S]FFF               S[F]FF
FHFH                 FHFH
FFFH        ==>      FFFH
HFFG                 HFFG
```

Stochastic Policy

Example: Frozenlake

```
# Make the environment based on non-deterministic policy (is_slippery=True)
env = gym.make('FrozenLake-v1', is_slippery=True, render_mode="ansi")
# Go right once (action = 2)
env.reset()
action = 2
(observation, reward, done, prob, a) = env.step(action)
print(env.render())
# Observation = 0: move to the right once from grid 0 to grid 1
# Prob = 1/3: deterministic policy, if we choose to go right, we might not go right
print(observation, reward, done, prob, a)
```

```
                       (Right)
[S]FFF                 SFFF
FHFH                   [F]HFH
FFFH          ==>      FFFH
HFFG                   HFFG
```

Markov Decision Processes

Technische Hochschule **Rosenheim**

- Typically we can frame all RL tasks as MDPs
- The key in MDPs is the **Markov Property**
  - Essentially the future depends on the present and not the past
  - More specifically, the future is independent of the past given the present
  - There's an assumption the present state encapsulates past information.

> Putting into the context of what we have covered so far: our agent can (1) **control its action based** on its current (2) **completely known state**

Two main characteristics for MDPs

- Control over state transitions
- States completely observable

Types of Markov Models

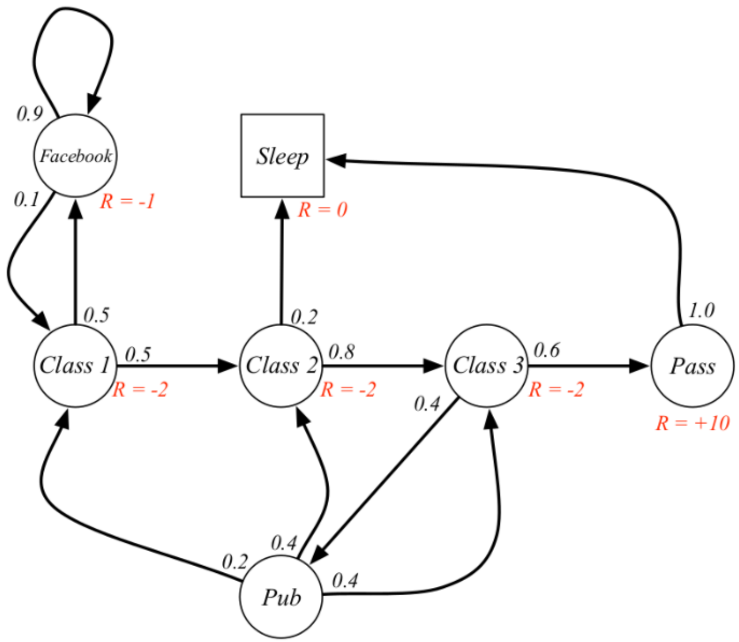
Permutations of whether there is presence of the two main characteristics would lead to different Markov models:

- Control over state transitions and completely observable states: MDPs
- Control over state transitions and partially observable states: Partially Observable MDPs (POMDPs)
- No control over state transitions and completely observable states: Markov Chain
- No control over state transitions and partially observable states: Hidden Markov Model

Markov Decision Process (MDP)

MDP is represented by five important elements:

- A set of states S the agent can actually be in
- A set of actions A that can be performed by an agent, for moving from one state to another
- A transition probability $P^{a}_{ss'}$, which is the probability of moving from one state S to another S' by performing an action a
- A reward probability $R^{a}_{ss'}$, which is the probability of a reward acquired by the agent for moving from one state S to another S' by performing some action a
- A discount factor γ, which controls the importance of immediate and future rewards



Terminology

We may or may not know our model

- **Model-based RL**: this is where we can **clearly define** our (1) transition probabilities and/or (2) reward function
    - A global minima can be attained via Dynamic Programming (DP)
- **Model-free RL**: this is where we cannot clearly define our (1) transition probabilities and/or (2) reward function
    - Most real-world problems are under this category so we will mostly place our attention on this category

Dynamic Programming

Dynamic programming is breaking down a problem into smaller sub-problems, solving each sub-problem and storing the solutions to each of these sub-problems in an array (or similar data structure) so each sub-problem is only calculated once.

- It is both a mathematical optimization method and a computer programming method.
- Richard Bellman invented DP in the 1950s.

Example:

```
def fibonacciVal(n):
    memo[0], memo[1] = 0, 1
    for i in range(2, n+1):
        memo[i] = memo[i-1] + memo[i-2]
    return memo[n]
```

Agent Acting

Technische Hochschule Rosenheim

- When the agent acts given its state under the policy (π(a|s)), the transition probability function $P^a_{ss'}$ determines the subsequent state (s')

$$P = P(s'|s, a) = P[S_{t+1} = s'|S_t = s, A_t = a]$$

- When the agent act based on its policy (π(a|s)) and transited to the new state determined by the transition probability function $P^a_{ss'}$ it gets a reward based on the reward function R^a_s as a feedback

$$R = E[R_{t+1}|S_t = s, A_t = a]$$

- Rewards are short-term, given as feedback after the agent takes an action and transits to a new state. Summing all future rewards and discounting them would lead to our return G_t

$$G = \sum \gamma^i R_{t+1+i}$$

Discount Factor

Technische Hochschule **Rosenheim**

Goal: Maximize total rewards!

- Immediate reward r at any time t_x (each step).
- If a reward r at any time t_x is defined as r_x then the cumulative future reward R_t at time t is defined as where T is the terminal time:

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

- with T= ∞, we use discounted future rewards :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where γ is the discount factor given as $0 \leq \gamma \leq 1$

- If $\gamma -> 0$ then we say the algorithm is myopic
- If $\gamma -> 1$ then the algorithm is far-sighted (it maximizes future rewards!)

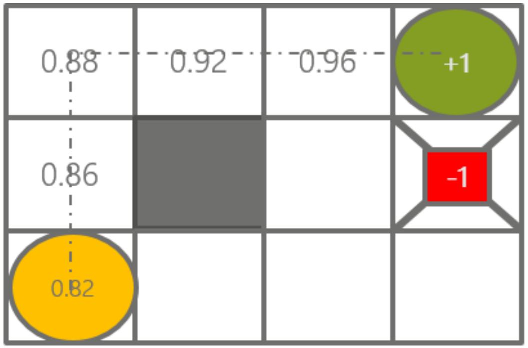
State-Value Function

- We need to calculate the accumulative reward from being in that state and follow the policy until the goal state, where the episode ends.
- We could calculate the value of 0.82 for the first given state.
- Also, we could calculate the same way we did, for each of the states that make up the path to the goal state.
- We are looking for the utility of a state under a given policy

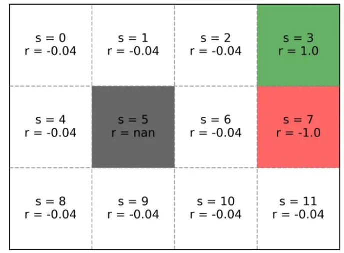
> The utility value is the summary of the reward of itself and all the future states following that policy.

$$V^{\pi}(s) = \mathbb{E}[\Sigma_{t\geq 0}\gamma^{t} r_t | s_t = s, \pi]$$

For each state s
It yields the expected return
If the agent starts in state s
And then uses the policy
To choose ist action



Utility or Value of States

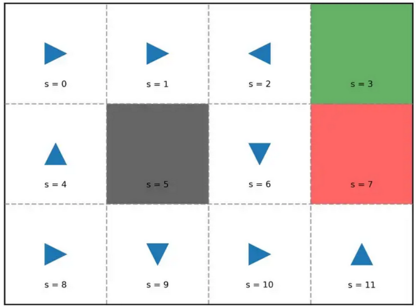


Let's consider a *non-deterministic* MDP:

- We are standing in s = 8. If we go to s = 9, we receive a reward of -0.04; if we go to s = 4, we also receive a reward of -0.04.
    - If we choose s = 9, it may lead us to the path of [9, 10, 11, 7] corresponding to rewards of [-0.04, -0.04, -0.04, -1]. Since MDP has the property of additive rewards, the total reward along this path is thus -1.12.
    - If we choose s = 4, it may lead us to the path of [4, 0, 1, 2, 3] corresponding to rewards of [-0.04, -0.04, -0.04, -0.04, +1] with a total reward of 0.84.
- Here -1.12 is the utility of state s = 9 while 0.84 is the utility of state s = 4. We usually use V to denote the *utility of states* or the *state values*.

Utility for Policy

Deterministic case



For instance, if we look at s = 6 and we follow the policy π, we have the path [6, 10, 11, 7] in front of us. Therefore, the utility of s = 6 is the sum of r(6), r(10), r(11), and r(7).

$$V(6)_{\pi=[6,10,11,7]} = r(6) + r(10) + r(11) + r(7)$$

or

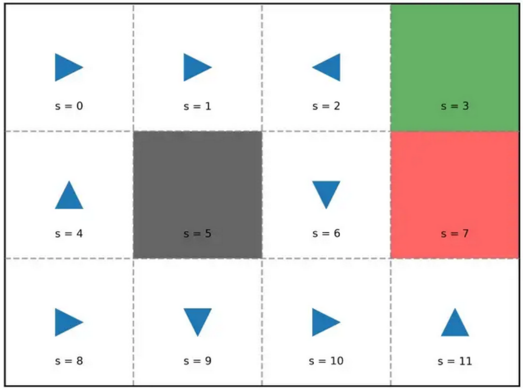
$$V(6)_{\pi=[6,10,11,7]} = r(6) + V(10)$$

Utility for Policy

Stochastic case

In a stochastic environment, even if we follow the policy, we are not guaranteed to move along the path [6, 10, 11, 7]

$$
\begin{aligned}
v(6) &= 0.8 \times \text{discounted rewards following policy}[6, 10...] \\
&+ 0.1 \times \text{discounted rewards following policy}[6, 7] \\
&+ 0.1 \times \text{discounted rewards following policy}[6, 6...] \\
&= 0.8 \times 0.8 \times \text{discounted rewards following policy}[6, 10, 11...] \\
&+ 0.8 \times 0.1 \times \text{discounted rewards following policy}[6, 10, 10...] \\
&+ 0.8 \times 0.1 \times \text{discounted rewards following policy}[6, 10, 6...] \\
&+ 0.1 \times \text{discounted rewards following policy}[6, 7] \\
&+ 0.1 \times 0.8 \times \text{discounted rewards following policy}[6, 6, 10...] \\
&+ 0.1 \times 0.1 \times \text{discounted rewards following policy}[6, 6, 7] \\
&+ 0.1 \times 0.1 \times \text{discounted rewards following policy}[6, 6, 6...]
\end{aligned}
$$



We can generalize to:

$$V^{\pi}(s) = E[\sum_{t \geq 0} \gamma^t r_t | s_t = s, \pi]$$

Utility for Policy

Stochastic case

With stochastic, the utility of a state under a certain policy can also be represented as the sum of its immediate reward and the utility of its successor state following a probability distribution:

$$V(s) = R_s + \gamma \sum_{s'} P^a_{ss'} V(s')$$

Example:

$$V(6) = r(6) + \gamma[0.8V(10) + 0.1V(6) + 0.1V(7)]$$

Finally, we will have 11 equations with 11 unknowns.

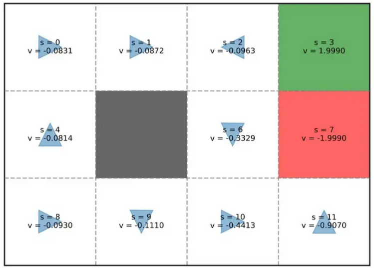
$$\begin{aligned}
v(0) &= r(0) + \gamma\,[0.8v(1) + 0.1v(0) + 0.1v(4)] \\
v(1) &= r(1) + \gamma\,[0.8v(2) + 0.1v(1) + 0.1v(1)] \\
v(2) &= r(2) + \gamma\,[0.8v(1) + 0.1v(6) + 0.1v(2)] \\
v(3) &= r(3) + \gamma\,[v(3)] \\
v(4) &= r(4) + \gamma\,[0.8v(0) + 0.1v(4) + 0.1v(4)] \\
v(6) &= r(6) + \gamma\,[0.8v(10) + 0.1v(6) + 0.1v(7)] \\
v(7) &= r(7) + \gamma\,[v(7)] \\
v(8) &= r(8) + \gamma\,[0.8v(9) + 0.1v(4) + 0.1v(8)] \\
v(9) &= r(9) + \gamma\,[0.8v(9) + 0.1v(8) + 0.1v(10)] \\
v(10) &= r(10) + \gamma\,[0.8v(11) + 0.1v(6) + 0.1v(10)] \\
v(11) &= r(11) + \gamma\,[0.8v(7) + 0.1v(10) + 0.1v(11)]
\end{aligned}$$

Policy Evaluation

Policy evaluation: for a given policy, we evaluate that policy by determining the utility of each state.

Instead of directly solving the linear equations, we can use a *dynamic programming approach* called *iterative policy evaluation*.

1. We first initialize the utility of each state as zero, then we loop through the states.
2. We repeat until the changes of utility values between consecutive sweeps are marginal.



$$
\begin{aligned}
v(0) &= r(0) + \gamma\,[0.8v(1) + 0.1v(0) + 0.1v(4)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times 0 + 0.1 \times 0] = -0.04 \\
v(1) &= r(1) + \gamma\,[0.8v(2) + 0.1v(1) + 0.1v(1)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times 0 + 0.1 \times 0] = -0.04 \\
v(2) &= r(2) + \gamma\,[0.8v(1) + 0.1v(6) + 0.1v(2)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.04) + 0.1 \times 0 + 0.1 \times 0] = -0.056 \\
v(3) &= r(3) + \gamma\,[v(3)] \\
&= 1 + 0.5 \times 0 = 1 \\
v(4) &= r(4) + \gamma\,[0.8v(0) + 0.1v(4) + 0.1v(4)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.04) + 0.1 \times 0 + 0.1 \times 0] = -0.056 \\
v(6) &= r(6) + \gamma\,[0.8v(10) + 0.1v(6) + 0.1v(7)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times 0 + 0.1 \times 0] = -0.04 \\
v(7) &= r(7) + \gamma\,[v(7)] \\
&= -1 + 0.5 \times 0 = -1 \\
v(8) &= r(8) + \gamma\,[0.8v(9) + 0.1v(4) + 0.1v(8)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times (-0.056) + 0.1 \times 0] = -0.0428 \\
v(9) &= r(9) + \gamma\,[0.8v(9) + 0.1v(8) + 0.1v(10)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times (-0.0428) + 0.1 \times 0] = -0.04214 \\
v(10) &= r(10) + \gamma\,[0.8v(11) + 0.1v(6) + 0.1v(10)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times (-0.04) + 0.1 \times 0] = -0.042 \\
v(11) &= r(11) + \gamma\,[0.8v(7) + 0.1v(10) + 0.1v(11)] \\
&= -0.04 + 0.5 \times [0.8 \times (-1) + 0.1 \times (-0.042) + 0.1 \times 0] = -0.4421
\end{aligned}
$$

Policy Evaluation

Technische Hochschule Rosenheim

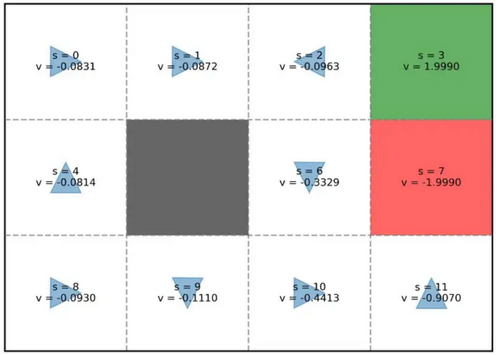
Pseudo-Code

```
input  : reward function r(s), transitional model p(s'|s, a),
         discounted factor γ, convergence threshold θ
         policy π(s), value v(s)
output: converged value v(s)
converge ← false
while converge = false do
    Δ ← 0
    for s ∈ S do
        temp ← v(s)
        v(s) ← r(s) + γ Σ_{s'} p(s'|s, a = π(s))v(s')
        Δ ← max(Δ, |temp − v(s)|)
    end
    if Δ < θ then
        converge ← true
    end
end
return v(s)
```

Policy Improvement

After we get the utility for this policy, it is time to improve the policy!

- The utility represents how good a state is.
  - When choosing actions for a state, we should prefer successor states with higher utility.
  - For instance, if we look at state s = 2, potential successor states include 1,3, and 6. Clearly s = 3 is the best choice as it has the highest utility of 1.999. Another example, if we look at state s = 8, potential successor states include 4 and 9 with utility of -0.0814 and -0.1110 respectively. As such, s=4 is preferable than s=9.



Policy Improvement

- Since our MDP is stochastic, selecting a preferable successor state does not guarantee we will reach it.
- Rather than successor states, what we should compare is actions. For state s = 6, we have four possible actions:

  1. UP : 0.8V(2) + 0.1V(6) + 0.1V(7) = -0.3093
  2. LEFT : 0.8V(6) + 0.1V(10) + 0.1V(2) = -0.3201
  3. DOWN : 0.8V(10) + 0.1V(6) + 0.1V(7) = -0.5862
  4. RIGHT: 0.8V(7) + 0.1V(2) + 0.1V(10) = -1.6530

Comparing the outcomes of these four possible actions, clearly UP is the best choice. Therefore, we should update the policy of the state s = 6 to UP.

We perform this process for each state:

$$\pi(s) = argmax[\sum_{s'} P^a_{ss'} V(s')]$$

Policy Improvement

Technische Hochschule Rosenheim

Pseudo-Code

input : transitional model $p(s'|s,a)$,
policy $\pi(s)$, value $v(s)$
output: updated policy $\pi(s)$,
binary indicating whether any change occurs

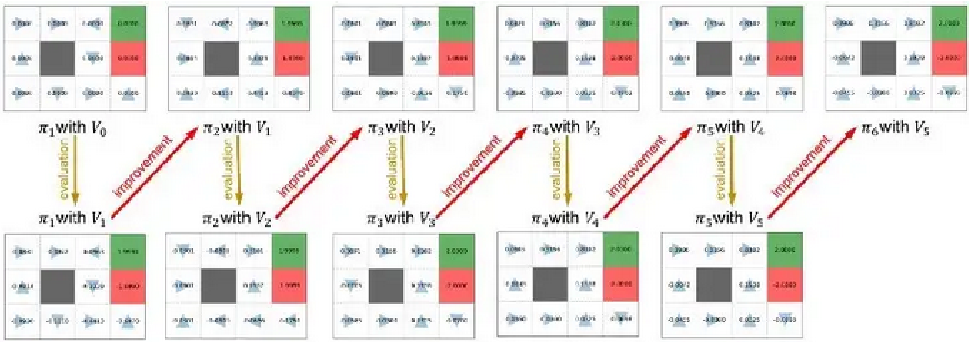
```
change ← false
for s ∈ S do
    temp ← π(s)
    π(s) ← argmax_a Σ_{s'} p(s'|s,a) v(s')
    if temp ≠ π(s) then
        change ← true
    end
end
return π(s), change
```

Policy Iteration

Technische Hochschule Rosenheim

We are not done yet!

- We do get a policy better than our initial one, but it is not necessarily the best one?
- What we need is to repeat this whole process of **Policy Evaluation** and then **Policy Improvement** again and again.
- When we perform policy evaluation again, we use for utility the results of policy iteration.



Policy Iteration

Pseudo-Code

input : reward function $r(s)$, transitional model $p(s'|s,a)$,
discounted factor γ, convergence threshold θ
output: optimal policy $\pi^*(s)$

initialize $\pi(s)$ randomly
initialize $v(s)$ with zeros
stable $\leftarrow$ false
while stable $=$ false **do**
    $v(s) \leftarrow$ `policy evaluation`$(r(s), p(s'|s,a), \gamma, \theta, \pi(s), v(s))$
    $\pi(s)$, change $\leftarrow$ `policy improvement`$(p(s'|s,a), \pi(s), v(s))$
    if change $=$ false **then**
        stable $\leftarrow$ true
    end
end
$\pi^*(s) \leftarrow \pi(s)$
return $\pi^*(s)$

State-Value Function

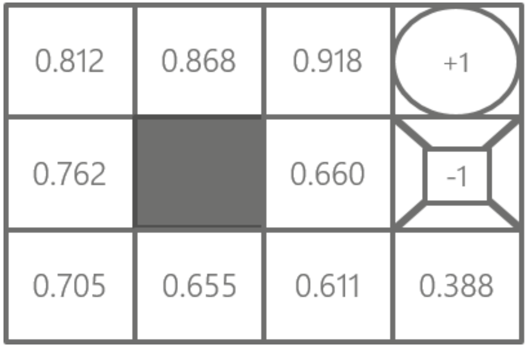
- State value (How good is it to be in state s?)

$$V^{\pi}(s) = E[\sum_{t \geq 0} \gamma^t r_t | s_t = s, \pi]$$

$$= \sum_{a} \pi(s,a)[R^a_{ss'} + \gamma \sum_{s'} P^a_{ss'} V^{\pi}(s')]$$

Expected cumulative reward from following the policies from state s

State	**Value**
State0	0.705
State5	0.762
State9	0.812
State2	0.655



Bellman Expectations Equations

- Basic: State-value function $V_\pi(s)$
  - Current state S
  - Multiple possible actions determined by stochastic policy $\pi(a|s)$
  - Each possible action is associated with a action-value function $Q_\pi(s,a)$ returning a value of that particular action
  - Multiplying the possible actions with the action-value function and summing them gives us an indication of how good it is to be in that state

$$V_\pi(s) = \sum_a \pi(a|s)Q(s,a)$$

> Loose intuitive interpretation: **state-value = sum(policy determining actions * respective action-values)**

Action-Value Function

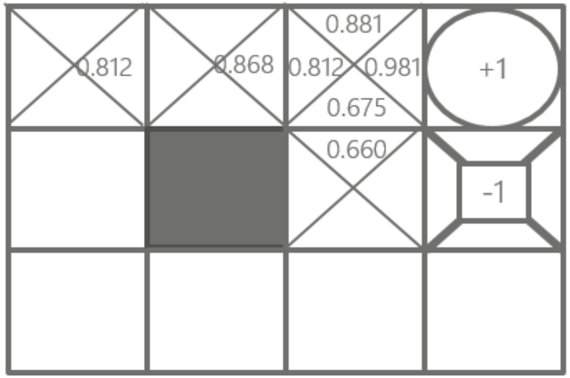
- Action value (How good is a (state, action) pair)

$$Q^{\pi}(s,a) = E[\sum_{t>0} \gamma^t r_t | s_t = s, a_t = a, \pi]$$

$$= R_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_{a'} \pi(a'|s') Q(s',a')$$

Expected cumulative reward from taking action a in state s and then following the policy.

State	Action	Value
State8	Action0 (←)	0.812
State8	Action1 (↑)	0.881
State8	Action2 (→)	0.981
State8	Action3 (↓)	0.675



Bellman Expectations Equations

Technische Hochschule **Rosenheim**

- Basic: Action-value function $Q_\pi(s, a)$
- With a list of possible multiple actions, there is a list of possible subsequent states s' associated with:
    - state value function $V_\pi(s')$
    - transition probability function $P^a_{ss'}$ determining where the agent could land in based on the action
    - reward R^a_s for taking the action

$$Q_\pi(s, a) = R^a_s + \gamma \sum_{s'} P^a_{ss'} V_\pi(s')$$

> Loose intuitive interpretation: **state-value = sumaction-value = reward + sum(transition outcomes determining states * respective state-values)**

Value Iteration

$$Q^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma \sum_{a'} Q^{\pi}(s',a')]$$

1. Initialize the random value function (random value for each state)
2. Compute Q function for all state action pairs of Q(s,a)
3. Update value function with the max value from Q(s,a)
4. Repeat steps until the change in the value function is very small

1

State	Value
A	0 -> 0.3
B	0
C	0

3 max Q(s,a) ←

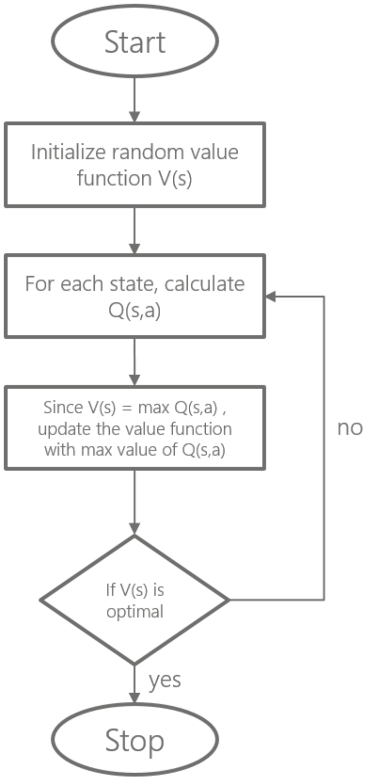
2

State	Action	Value
A	0	-0.1
A	1	0.3
B	0	

State	Action	Next State	TP	RP
A	0	A	0.1	0
A	0	B	0.4	-1.0
A	0	C	0.3	1.0
A	1	A	0.3	0
A	1	B	0.1	-2.0
A	1	C	0.5	1.0

Q(A,0)=(0.1*(0+0)) + (0.4*(-1.0+0)) + (0.3*(1.0+0))=-0.1

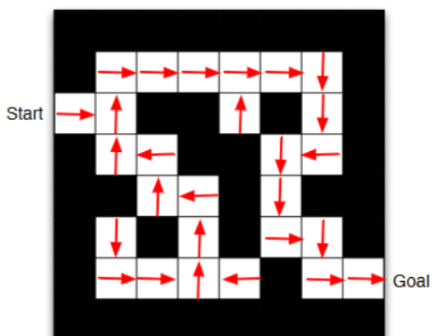
Q(A,1)=(0.3*(0+0)) + (0.1*(-2.0+0)) + (0.5*(1.0+0))=0.3



Results

Policy-based RL

Search directly for the optimal policy π^*
This is the policy achieving maximum future reward



Arrows represent policy π(s) for each state s

Value-based RL

Value is (an estimate of) the expected future reward
Value = long-term vs. Reward = immediate

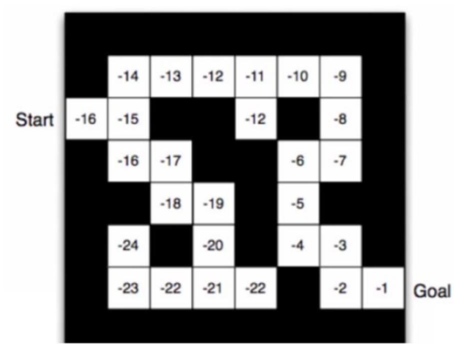
State Value Function $$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left(R^{a}_{ss'} + \gamma V^{\pi}(s')\right)$$

State-Action Value Function $$Q(s,a) = \sum_{s'} P^{a}_{ss'} \left(R^{a}_{ss'} + \gamma V^{\pi}(s')\right)$$

Estimate the optimal state-action value function Q*(S,A)

$$Q^{*}(s,a) = \max Q(s,a)$$

This is the maximum value achievable under any policy



Numbers represent value $V^{\pi}(s)$ of each state s

Comparison

- Both *value iteration* and *policy iteration* compute the same thing (all optimal values)
- In **policy iteration**:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)
- In **value iteration**:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- Both are dynamic programs for solving MDPs

Summary

Lessons learned today:

- **Model based**: Attempts to model the environment to find the best policy
- **Policy**: Evaluation, Improvement, and Optimization
- **State-value based**: Search for the optimal state-value function (goodness of action in the state)
- **Action-value based**: Search for the optimal action-value function (goodness of policy)

> Code can be found in
>
> - RL_Policy_Optimization.ipynb
> - or on GitHub or hosted on myBinder

Exercise

https://inf-git.fh-rosenheim.de/aai-url/09_uebung

- Value and Policy Iteration by hand
- Questions?