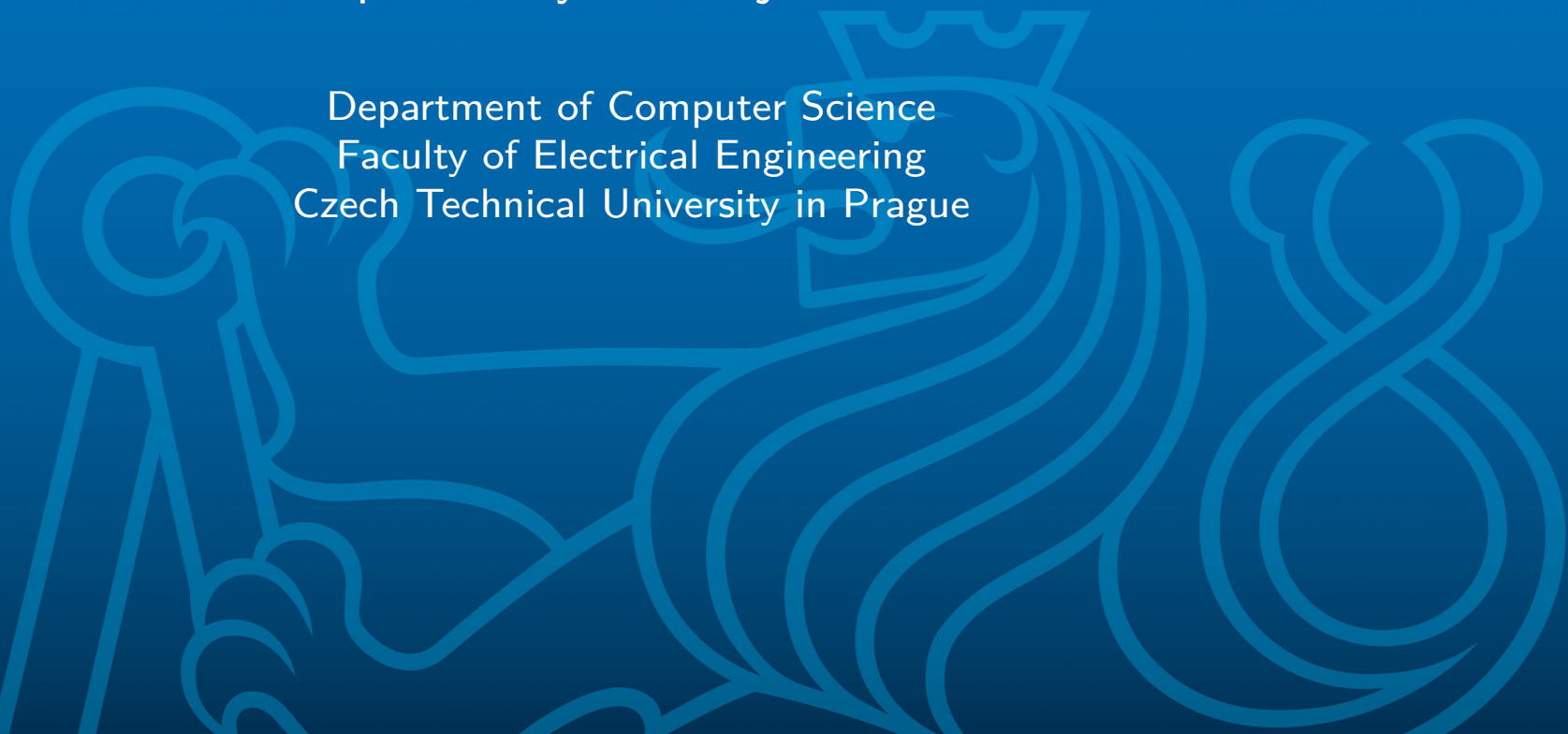
Symbolic Machine Learning

Lecture Slides

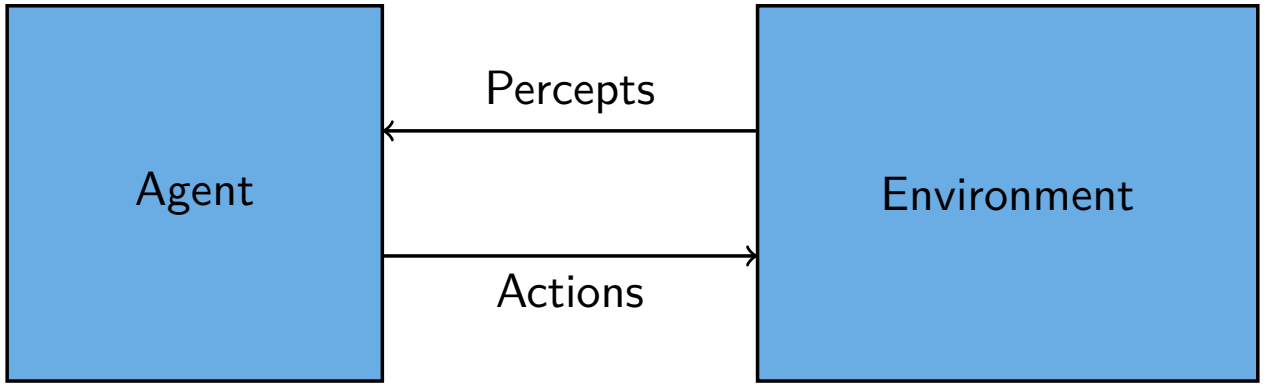
Filip Železný, Ondřej Kuželka

Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague



A General Framework

Agent Interacts with Environment



- Discrete *time*

$$k = 1, 2, \ldots$$

- *Percepts*

$$\forall k : x_k \in X$$

- *Actions*

$$\forall k : a_k \in A$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Histories, Probability, Agent's Policy

Interaction or *history* up to time m which we denote as $xa_{\leq m}$

$$xa_{\leq m} = x_1, a_1, x_2, a_2, \ldots, x_m, a_m$$

starts with a percept (arbitrary choice but stick to it) and has *probability*

$$P(xa_{\leq m}) = P(x_1)P(a_1|x_1)P(x_2|x_1, a_1)\ldots P(x_m|xa_{<m})P(a_m|x_m, xa_{<m})$$

The $P(x_1)$ and $P(x_k|.)$ factors depend on the stochastic environment while $P(a_k|.)$ factors depend on the agent. We will only assume *deterministic agents* so

$$P(a_k|xa_{<k}, x_k) = \begin{cases} 1 \text{ if } a_k = \pi(xa_{<k}, x_k) \\ 0 \quad \text{otherwise} \end{cases}$$

where $\pi(xa_{<k}, x_k)$ is the agent's *policy*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Rewards

The environment rewards the agent depending on its past actions. Formally, *rewards* $r_k \in R \subset \mathbb{R}$ are a distinguished part of percepts

$$x_k = (o_k, r_k)$$

while everything else in the percepts are *observations* $o_k \in O$. The set R of possible rewards must be *bounded*, i.e., for some $a, b \in \mathbb{R}$ and all $r \in R$, $a \leq r \leq b$.

Notes:

- Rewards, as part of percepts, generally depend on the entire history. A long sequence of 'good' actions may be needed for a high reward. Example: chess-game with the only 'win' reward at the end.
- r_1 is immaterial.

Define marginals on O and R so that

$$P(x_k|xa_{<k}) = P(o_k, r_k|xa_{<k}) = P_O(o_k|r_k, xa_{<k})P_R(r_k|xa_{<k})$$

Agent's Goal

We want the agent to maximize rewards, which can be formalized e.g. as:

- With a *finite* time horizon $m \in \mathbb{N}$ of interest, π should yield $a_1, a_2, \ldots, a_{m-1}$ to maximize expected cumulative reward

$$\mathbb{E}\left(\sum_{k=1}^{m} r_k\right) = \sum_{r_{\leq m}} P_R(r_{\leq m}|a_{<m})(r_1 + r_2 + \ldots + r_m)$$

- If horizon *infinite*, use a discount sequence δ_k such that $\sum_{i=1}^{\infty} \delta_i < \infty$ and maximize

$$\mathbb{E}\left(\sum_{k=1}^{\infty} r_k \delta_k\right) = \lim_{m\to\infty} \sum_{r_{\leq m}} P_R(r_{\leq m}|a_{<m}) \sum_{k=1}^{m} r_k \delta_k$$

  Usual choice $\delta_k = \gamma^k$ for $0 < \gamma < 1$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Markovian Environments

A *Markovian* or *state-based* environment is one for which a *state* variable $s : \mathbb{N} \to S$ and distributions P_x,P_S exist such that S has bounded size and

- $P(x_k|xa_{<k}) = P_x(x_k|s_k)$, i.e., x_k depends only on the current state. The assumption is strong because S is bounded and cannot contain a state for each possible history $xa_{<k}$ as $k \to \infty$.
- State s_k $(k > 1)$ is distributed according to $P_S(s_k|s_{k-1}, a_{k-1})$, i.e., it depends only the previous state and the action taken on it by the agent. The initial state is distributed by $P_S(s_1)$.

Note: since $P_x(x_k|s_k) = P_x((o_k, r_k)|s_k)$, both o_k and r_k depend on the current state s_k. Sometimes it is more convenient to model the reward r_k as distributed by $P_r(r_k|s_{k-1}, a_{k-1})$ instead.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Markovian Agents

A *Markovian* or *state-based* agent is one for which a *state* variable $t : \mathbb{N} \to T$ and *functions* π,$\mathcal{T}$ exist such that T has bounded size and

- $\pi(xa_{<k}, x_k) = \pi(t_k, x_k)$. Since T is bounded, some different histories will result in the same action of the agent. One can view t_k as agent's flexible (learnable) decision model while π its fixed interpreter.
- For $k > 1$, $t_k = \mathcal{T}(t_{k-1}, x_k)$, i.e., it depends only on the previous state and the current percept (t_1 is some initial state). $\mathcal{T}$ is the state update function, which will be the core of learning.

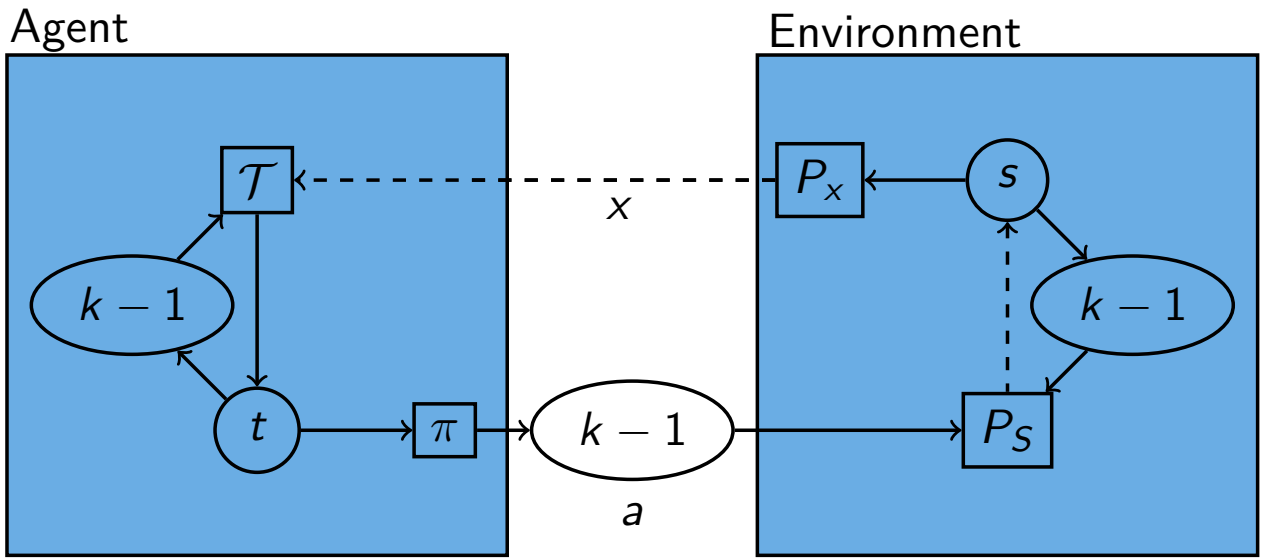
Note: since $\pi(t_k, x_k) = \pi(t_k, (o_k, r_k))$, the policy depends also on r_k. Usually, it suffices to consider dependence only on o_k by $\pi(t_k, o_k)$. And since both o_k, r_k can be stored as part of t_k by the update function, one may even use $\pi(t_k)$ as done on the next page.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Markovian Interaction Model

We now have a general structure in which to study learning:

Agent		Environment	
actions:	$a = \pi(t_k)$	percepts:	$x_k \sim P_x(x_k \mid s_k)$
state update:	$t_k = \mathcal{T}(t_{k-1}, x_k)$	state update:	$s_k \sim P_S(s_k \mid s_{k-1}, a_{k-1})$



Terminal States, Proper Policies

We can distinguish some states from S as *terminal*. If s_k is terminal then s_{k+1} is sampled independently of s_k, a_k from $P_s(s_{k+1})$, i.e. just like the initial state s_1. The interaction between a terminal (or initial) state and the next terminal state is called an *episode*.

Informally, the environment is 'restarted' after a terminal state. However, the agent is not restarted ($t_{k+1} = \mathcal{T}(t_k, x_k)$), so it can learn from one episode to another.

For a given environment, a policy π is *proper* if it is guaranteed to achieve a terminal state.

With a proper policy, we can modify the agent's goal as to maximize $\mathbb{E}\left(\sum_{k=1}^{m} r_k\right)$ where s_m is the first terminal state in the interaction (no need for a discount factor).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Reinforcement Learning

FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

Reinforcement Learning

Reinforcement learning is a collection of techniques by which the agent achieves high rewards in the state-based (Markovian) setting under two assumptions:

- Environment *fully observable*, i.e., $O = S$ and for $\forall k$: $o_k = s_k$.
- Reward is a function of current state: $\forall k : r_k = r(s_k)$.

There is no P_X anymore because percepts are here a function of states

$$x_k = (o_k, r_k) = (s_k, r(s_k))$$

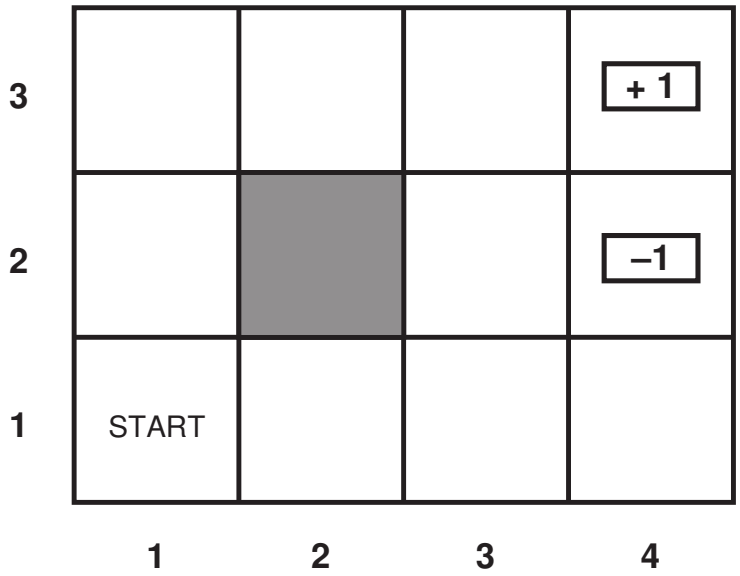
The environment is described by the update (transition) distribution P_S and the reward function r, both of which are unknown to the agent.

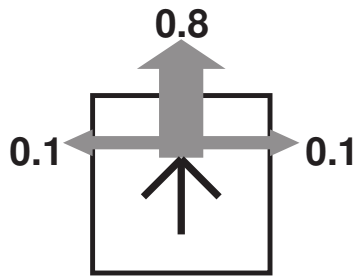
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Reinforcement Learning: Example

Agent should learn to get from START to the +1 goal in the grid world. The -1 goal should be avoided.

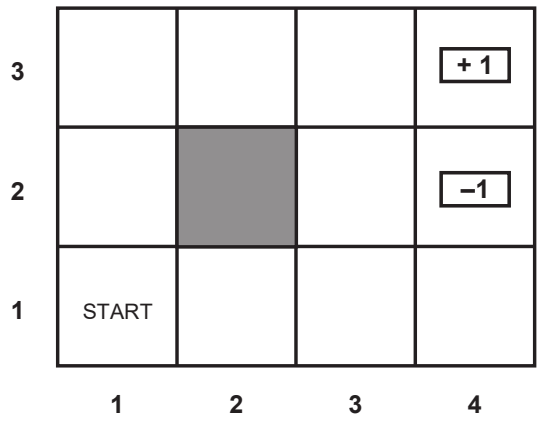
Effects of actions (moves) are uncertain.





(Images in this lecture are from the AIMA book)

Formalizing the Example in the Markovian Setting



Env. states $S = \{1, 2, 3, 4\} \times \{1, 2, 3\} \setminus \{(2, 2)\}$ correspond to agent's positions on the grid. States $(4, 3)$ and $(4, 2)$ are terminal, with respective rewards 1 and -1.

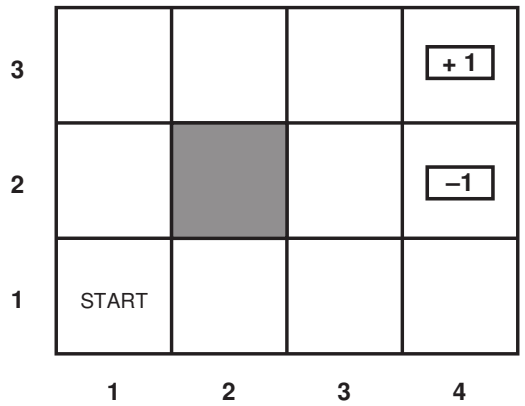
Percepts $X = S \times R$

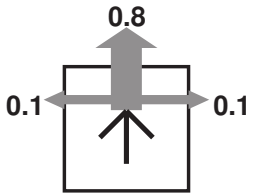
Agent states T encode possible agent's decision models. Depend on the chosen implementation of an agent (we will study that).

Actions $A = \{\text{left}, \text{right}, \text{up}, \text{down}\}$.

Formalizing the Example (cont'd)

Actions have uncertain effects on the environment state.





Prescribed by distribution $P_S(s_{k+1}|s_k, a_k)$. Here e.g.

$$P_S((3,2)|(3,1),\mathrm{up}) = 0.8$$
$$P_S((2,1)|(3,1),\mathrm{up}) = 0.1$$
$$P_S((4,1)|(3,1),\mathrm{up}) = 0.1$$

Bouncing: if outcome s_{k+1} out of grid, then $s_{k+1} := s_k$.

Agent State, Fixed Policy

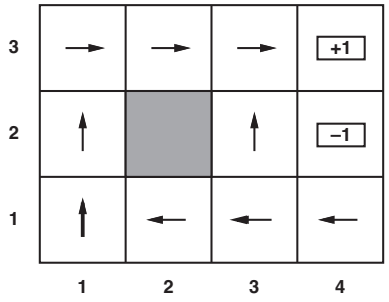
Agent state t_k contains the agent's model at time k, prescribing the action (decision) policy

$$a_k = \pi(t_k, s_k)$$

In the simplest case, t_k may be a static lookup table Π (see on right)

$$\pi(\Pi, s) = \Pi[s]$$

When the agent state does not change with k as here, we call the policy *fixed*. Of course, fixed policy means no learning. In this case we can omit the first argument, writing just $\pi(s_k)$.



s	$\Pi[s]$
$(1,1)$	up
$(1,2)$	up
$(1,3)$	right
$(2,1)$	left
$(2,3)$	right
$(3,1)$	left
$(3,2)$	up
$(3,3)$	right
$(4,1)$	left

State Utility Under a Fixed Policy

Given a fixed policy π, how good is it to be in state s_k at time k? The better, the higher the *expected utility* of the state (under that policy):

$$U^\pi(s_k) = \mathbf{E}\left[\sum_{i=0}^{\infty} \gamma^i r(s_{k+i})\right] = r(s_k) + \gamma U^\pi(s_{k+1})$$

where $0 < \gamma < 1$ is the discount factor. If π is proper, we can have $\gamma = 1$, summing only up to the first terminal state s_{k+i}.

Since s_{k+1} $(k \geq 1)$ are distributed by $P_S(s_{k+1}|s_k, a_k)$, we can write this as

$$U^\pi(s_k) = r(s_k) + \gamma \sum_{s \in S} P_S(s|s_k, \pi(s_k)) U^\pi(s)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Optimal Policy, State Utility

$$\pi^* = \arg\max_{\pi} U^{\pi}(s)$$

is called the *optimal policy*.

Which policy $\pi : S \to A$ is optimal depends on a state s by this definition. However, it can be shown that any $s \in S$ yields the same π^*.

Considering the definition of $U^{\pi}(s_k)$, π^* maps each s_k $(k > 1)$ to an action maximizing the expected utility of the next state

$$\pi^*(s_k) = \arg\max_{a \in A} \sum_{s \in S} P_S(s|s_k, a) U(s)$$

$U(s) = U^{\pi^*}(s)$ is called the *state utility* (without adjectives).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing an Optimal Policy

So *if the agent knows P_S and r*, it can decide optimally by

$$a_k = \arg\max_{a \in A} \sum_{s \in S} P_S(s|s_k, a) U(s)$$

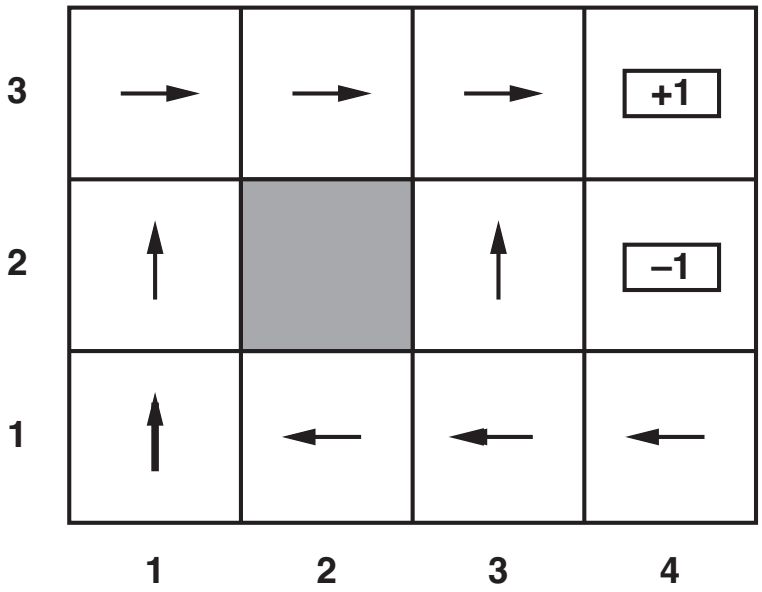
For this, it first needs to compute $U(s)$ for all $s \in S$. They are solutions of $|S|$ non-linear *Bellman* equations (one for each $s \in S$)

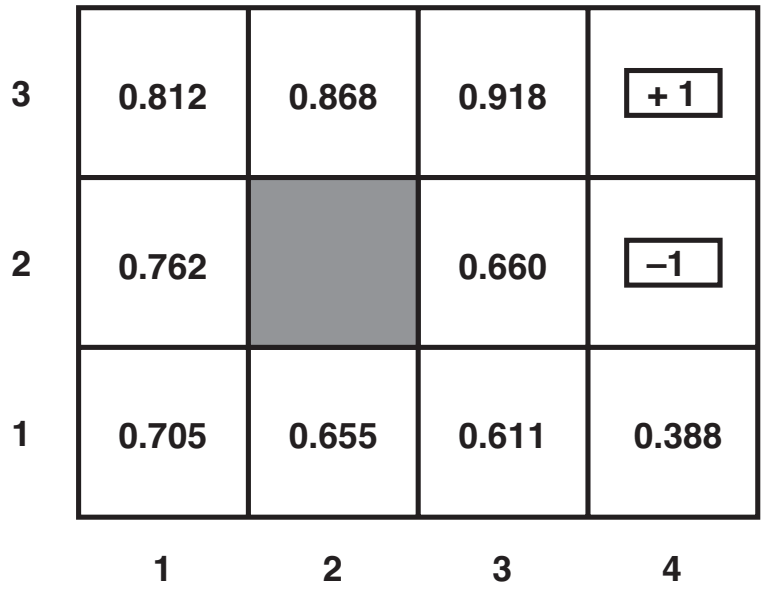
$$U(s) = r(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_S(s'|s, a) U(s')$$

These can be solved by the *value iteration* algorithm known from the theory of *Markov Decision Processes*.

(Note: P_S, r, S, A, γ define an MDP, π^* is its solution.)

Optimal Policy and State Utilities

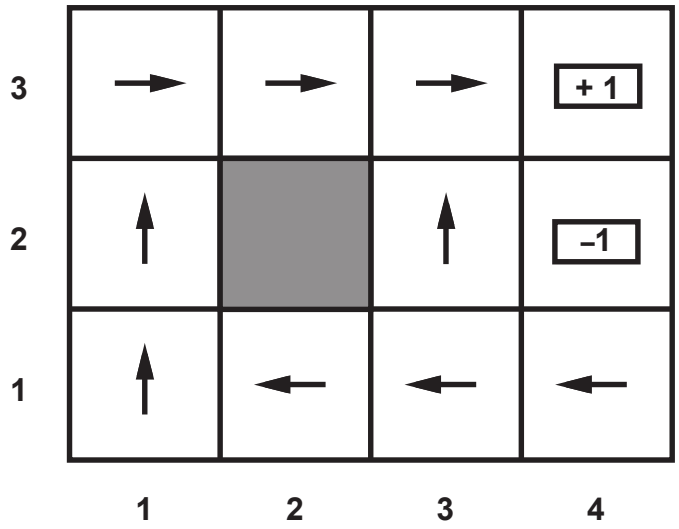




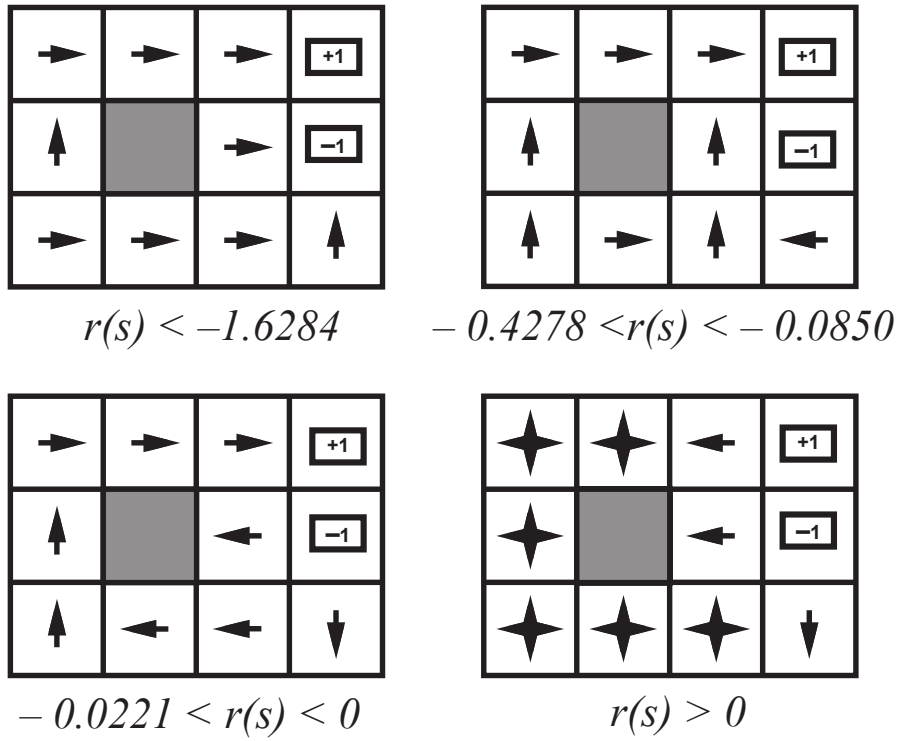
Optimal policy (left) and state utilities (right) for $\gamma = 1$ and $r(s) = -0.04$ for all non-terminal states s

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Optimal Policies Under Different Rewards



Optimal policy for $\gamma = 1$ and $r(s) = -0.04$ for non-terminal states s.



Optimal policies for $\gamma = 1$ and different ranges of $r(s)$ for non-terminal states s.

Learning an Optimal Policy

What if P_s and r are not known?

1. If the agent knows the state utilities, it can determine an optimal policy.
2. State utilities can be estimated by interacting with the environment. But for this interaction, some policy is needed.

So 1 and 2 must be somehow interleaved.

We will first look at 2. The agent will be prescribed a fixed policy. Such an agent is called *passive*.

Then we will see how to combine 1 with 2.

Passive Direct Utility Estimation Agent

- Follows a fixed proper policy π - see example on right. Policy formally extended with end actions to indicate terminal states (needed for later pseudo-codes).
- With $\gamma = 1$, agent's estimate of $U^\pi(s)$ for state s at time k is the average of all *rewards-to-go* of s until k.
- A reward-to-go of s is the sum of rewards from s till the end of the current episode. If s visited multiple times in one episode, then that episode produces multiple rewards-to-go to include in the average.

s	$\Pi[s]$
$(1,1)$	up
$(1,2)$	up
$(1,3)$	right
$(2,1)$	left
$(2,3)$	right
$(3,1)$	left
$(3,2)$	up
$(3,3)$	right
$(4,1)$	left
$(4,2)$	end
$(4,3)$	end

Example: estimate utility of state $(1,2)$ over 3 episodes in the grid:

$(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$ . . . 0.76, 0.84
$(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (3,2)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$ . . . 0.76
$(1,1)_{-.04} \rightarrow (2,1)_{-.04} \rightarrow (3,1)_{-.04} \rightarrow (3,2)_{-.04} \rightarrow (4,2)_{-1}$ . . . no occurrence

So the estimate is $(0.76 + 0.84 + 0.76)/3$.

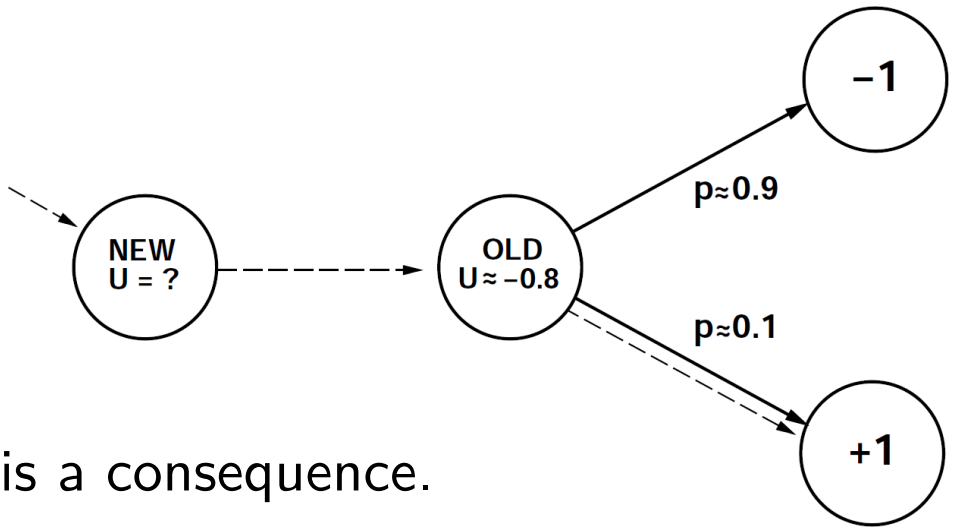
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Passive DUE Agent: Properties

The DUE Agent does not make use of the known dependence between state utilities

$$U^{\pi}(s_K) = r(s_K) + \gamma \sum_{s \in S} P_S(s|s_K, \pi(s_K)) U^{\pi}(s)$$

E.g. below the newly explored state will likely have low utility due to the neighbor state (already explored). The agent does not 'know' this.



Slow convergence is a consequence.

Passive Adaptive Dynamic Programming Agent

Instead of computing $\widehat{U}$ directly from samples, learn a model of P_S and r and compute $\widehat{U}$ from them.

For r, just collect an array $\widehat{r}[s]$ of observed rewards for observed states.

For $P_S(s'|s,a)$, collect the counts $N[s',s,a]$ of observed triples of action a taken in state s and resulting in state s'. Then estimate:

$$P_S(s'|s,a) \approx \frac{N[s',s,a]}{\sum_{s'' \in S} N[s'',s,a]}$$

The *policy evaluation* algorithm (as known from MDP's) takes $\widehat{r}$ and N, and produces $\widehat{U}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Policy Evaluation: Reminder of Pre-Requisite Material

State utilities should satisfy

$$U^{\pi}(s) = r(s) + \gamma \sum_{s' \in S} P_S(s'|s, \pi(s)) U^{\pi}(s')$$

The policy evaluation plugs in the model $\frac{N[s',s,a]}{\sum_{s'' \in S} N[s'',s,a]}$ and $\widehat{r}$ of $P_S(s'|s,a)$ and r, and calculates $\widehat{U}$ by *value iteration* for all $s \in S$ until convergence:

$$\widehat{U}[s] \leftarrow \widehat{r}[s] + \gamma \sum_{s' \in S} \frac{N[s', s, \pi(s)]}{\sum_{s'' \in S} N[s'', s, \pi(s)]} \widehat{U}[s']$$

This is a system of *linear* assignments, so instead of iterating them, the corresponding equations can be solved through matrix algebra in $\mathcal{O}(|S|^3)$ time.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Passive ADP Agent: Design

Agent's state is a tuple

$$t_k = \left\langle \Pi, s_k^{\mathrm{old}}, N_k, \widehat{r}_k, \widehat{U}_k \right\rangle$$

- Π: fixed decision array (as in the DUE agent)
- s_k^{old}: last seen state
- N_k: 3-way contingency array indexed by $[s' \in S, s \in S, a \in A]$.
- $\widehat{r}_k$: reward array indexed by $s \in S$.
- $\widehat{U}_k$: state utility estimate array indexed by $s \in S$.

The last four variables are initially ($k = 1$) filled with the none value.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Passive ADP Agent: Design (cont'd)

Update step $t_{k+1} = \mathcal{T}(t_k, x_k)$ where $t_k = \left\langle \Pi, s_k^{\text{old}}, N_k, \widehat{r}_k, \widehat{U}_k \right\rangle$ and $x_k = (r_k, s_k)$:

$s_{k+1}^{\text{old}} = s_k$	Current state is stored for use at next update.
$N_{k+1}[s_k, s_k^{\text{old}}, \Pi[s_k^{\text{old}}]]$ $= N_k[s_k, s_k^{\text{old}}, \Pi[s_k^{\text{old}}]] + 1$	Contingency array is incremented.
$\widehat{r}_{k+1}[s_k] = r_k$	Reward observed for the current state is stored.
$\widehat{U}_{k+1} =$ $\text{policy\_eval}(N_{k+1}, \widehat{r}_{k+1})$	Utilities are estimated by the policy evaluation algorithm.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE