

REINFORCEMENT LEARNING

Ivan Bratko

Faculty of Computer and Information Science

University of Ljubljana

References

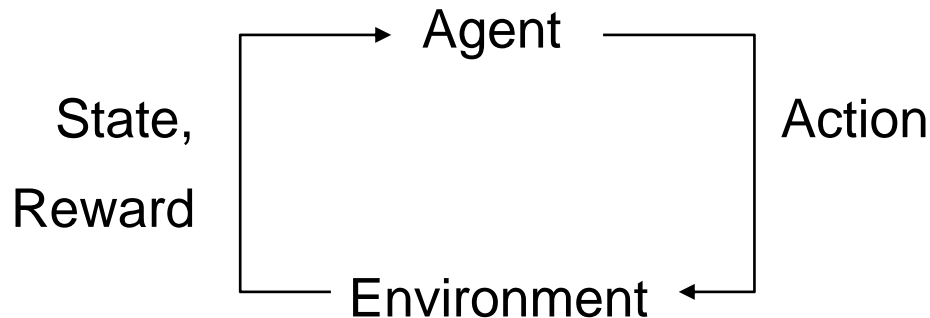
- These slides largely follow Chapter 21, Reinforcement learning, of S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach (3rd edition), 2010. Parts of Chapter 17 contain useful prerequisite knowledge for Chapter 21. In 4th edition (2021) Reinforcement learning is covered in Chapter 22.
- Classic book on RL: Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 1998; second edition 2018
- Chapter 12 (Learning to act) in D.L. Poole and A.K. Mackworth, Artificial Intelligence: Foundations of Computational Agents (2nd ed.) Cambridge University Press, 2017.
- Abbreviation: Reinforcement Learning = RL
- In Slovenian: RL = Spodbujevano učenje

RL: KEY TERMS IN SLOVENIAN

- | | |
|--------------------------|--|
| • reinforcement learning | spodbujevano učenje |
| • reward, reinforcement | nagrada (spodbuda) |
| • delayed reward | zakasnjena nagrada (zapoznela nagrada) |
| • cumulative reward | celotna nagrada, akumulirana nagrada |
| • discounted cum. reward | znižana nagrada |
| • policy | strategija |
| • utility | koristnost (korist) |

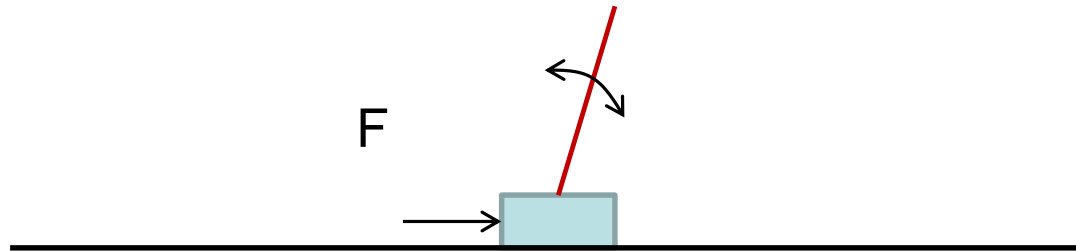
REINFORCEMENT LEARNING SETTING

- Intended for learning decision strategies of an agent, or system control (robot control), taking into account uncertainty
- Agent interacts with its environment



- Execution cycle (State, Action, NextState) repeated at discrete times t_0, t_1, t_2, \dots

A CLASSICAL EXAMPLE: POLE-CART



- $F = +1$ or -1 (bang-bang regime)
- Model of pole-cart not available
- Task 1: Find F in time to avoid pole falling
- Task 2: Find F in time to move cart from start to goal position while avoiding pole falling
- Note: Pos./neg. feedback only occurs when pole falls
- This is very late, “delayed reward”, this makes problem hard
- How is this stated formally in RL?
- Reinforcement learning is a method for this (“the” method)

INVERTED PENDULUM EQUATIONS (POLE AND CART, INVERTED PENDULUM)

$$(M + m) \ddot{x} - m\ell\ddot{\theta} \cos \theta + m\ell\dot{\theta}^2 \sin \theta = F$$

$$\ell\ddot{\theta} - g \sin \theta = \ddot{x} \cos \theta$$

OTHER FAMOUS REINF. LEARNING TASKS

- Balancing double or triple pendulum (triple pole and cart)
- Acrobot
- Learn to fly a quadcopter
- Exploring maze-like environments,
- Playing complex games, total reward (success) only known at end of game; AlphaZero is a famous example
- Marketing; actions - advertisements, rewards – sales
- Handling pandemic
- Container balancing, bike balancing

DEMOS OF POLE-CART (INVERTED PENDULUM)

- <http://www.youtube.com/watch?v=a4c7AwHFkT8>
- http://www.youtube.com/watch?v=Ci_y14y3DU4
- <http://www.youtube.com/watch?v=f6Gpehgzy4w&feature=related>
(Flying inverted pendulum)
- <http://www.youtube.com/watch?v=cyN-CRNrb3E&feature=related>
(Triple pendulum, acrobot-like)
- https://www.youtube.com/watch?v=cyN-CRNrb3E&ebc=ANyPxKocJ7Cc3YWUr6bkbqMPEI-08oUKgqNp8Nb_Um7NV_4oiCc_uENUPBO2033zR5hWy7WAPpx9
3-link inverted pendulum, Acrobot like, same as above
- <http://www.youtube.com/watch?v=Lt-KLtkDIh8>
Reinforcement learning to swing up pole on cart, Martin Riedmiller
- <https://www.youtube.com/watch?v=FeCwtvrD76I>
- <https://www.youtube.com/watch?v=sMZRnE3q72c>
Acrobot controller

Learning to fly a quadcopter

- Jemin Hwangbo, Inkyu Sa, Roland Siegwart, Marco Hutter: Control of a quadrotor with reinforcement learning

<https://www.youtube.com/watch?v=T0A9voXzhng> (4 min)

- Model predictive control of quadcopter:

https://www.youtube.com/watch?v=dL_ZFSvLXIU (2 min)

DEMOS

CRAWLING ROBOT, RESCUE ROBOT

- Crawling robot by Bevčar and Kotnik
- Flipper: human driving over obstacle
- Flipper learning to climb stairway

Spectacular success of Alpha/X programs from DeepMind which use RL

- 2016: AlphaGo (David Silver,, Demis Hassabis, Nature 2016)
AlphaGo defeated a leading Go-player Lee Sedol
(first time that a computer performed better
than best human go player)
AlphaGo learned from examples of good games played by humans
- October 2017: AlphaGo Zero (Silver, ..., Hassabis, Nature)
AlphaGo Zero vs. AlphaGo 100:0
AlphaGo Zero only learned by self-play (no access to human
knowledge about the game; only the rules of the game)

AlphaZero

- December 2017: AlphaZero (Silver, ..., Hassabis, ArXiv 2017)
In less than 24 hours of learning by self-play better than any human and any other program at chess, Go and shogi
- In 90 min of self-play, AlphaZero outperformed best human chess players
- An interesting observation from learning chess:
 - To surpass best human players, AlphaZero played 6.9 M games against itself
 - ChessBase's Mega Database 2018 contains 7.1 M best (more or less) games ever played between humans (representative of total human knowledge of chess)

TECHNIQUES USED BY AlphaZero

- Uses **Reinforcement Learning** to learn by self-play (simulated games against itself)
- Uses **Deep Neural Networks** to generalize results of RL: learn to predict values and move probabilities for positions never encountered before
- Uses **Monte Carlo Tree Search** to select the move to play

AlphaZero

- Big surprise: Alpha Zero plays very creative chess, unlike other chess playing programs
- cf. also LeelaChess, a re-implementation of Alpha Zero approach on normally accessible hardware
- see chess commentary by Anna Rudolf

ARE THERE ANY LIMITATIONS TO AlphaZero?

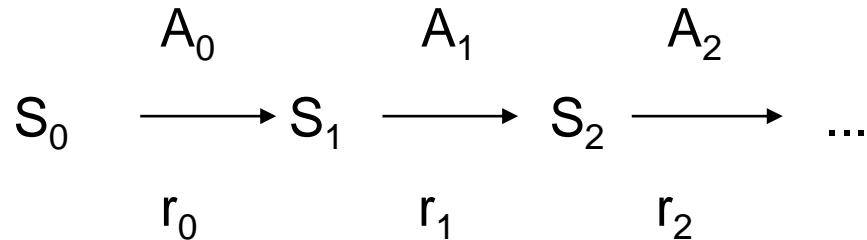
- AlphaZero – What's Missing? (Bratko, Informatica 2018)
- Limitations:
 - Relies on virtually unlimited self-play, i.e. **unlimited simulation**; in many real-world domains this is not possible (in medicine, **nature** produces new examples, not a simulator)
 - **Inability to explain** its decisions to human players (example games between AlphaZero and Stockfish)
 - “Explainable AI”

TYPICAL PROPERTIES OF RL PROBLEM

- Find best **policy**: for each state, what is the best action?
- Agent typically has **no model** of environment: agent does not know what will be effects of actions (planning not possible)
- Environment may behave **non-deterministically** (probabilistically): the same action may produce different results at different times
- Rewards are typically **delayed**; so agent may not know how it is doing during execution because reward (positive or negative) will come much later
- Agent may try to learn state transition model, etc.

IMMEDIATE AND CUMULATIVE REWARDS

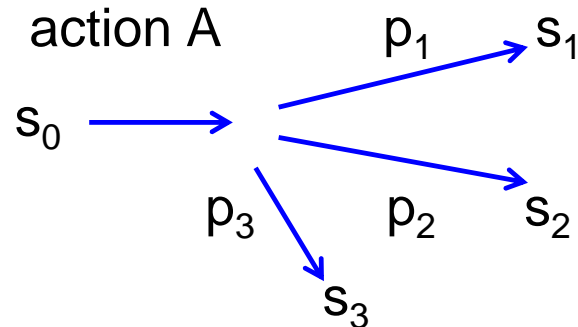
- Time sequence of states, actions and rewards (“trial”):



- r_i = immediate reward from performing action A_i in state S_i
- Cumulative reward* = $r_0 + \text{gamma} * r_1 + \text{gamma}^2 * r_2 + \dots$
- $0 < \text{gamma} \leq 1$
- This is *discounted* cumulative reward
- Why discounting? Avoid infinite rewards in infinite trials

ENVIRONMENT MAY BE NON-DETERMINISTIC

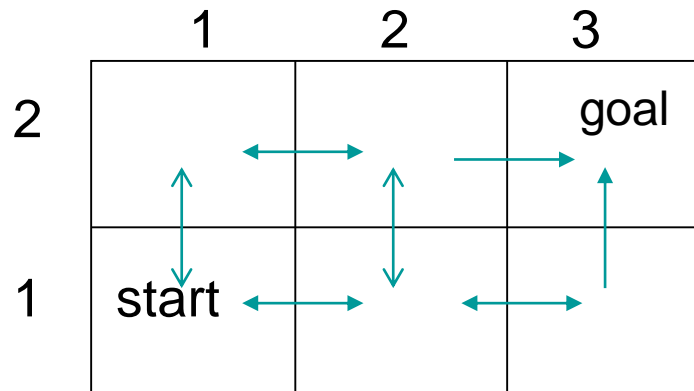
- Results of actions can be non-deterministic, characterised by probabilities



- How can pole-cart be viewed as non-deterministic?
- When continuous space ($X, V, \text{Theta}, D\text{Theta}$) is discretised into “boxes” this introduces uncertainty about where in a box the system actually is
- Therefore non-deterministic transitions to next boxes appear

DIFFICULTIES: A SIMPLE EXAMPLE

- A simple robot world consisting of a 3x2 grid. Robot can move horizontally or vertically between adjacent cells; transition into goal state gives reward 100, any other transition reward -1.



- In any state, robot may choose between actions l, r, u, d, but cannot predict their effects
- Suppose robot is at (2,1) and chooses action u, and observes next state is (1,1); another time u will produce (2,2) (non-determinism)
- Suppose robot is at (2,1): any action there will produce reward -1, so it is hard to figure out which action is best?

LEARNING TASK

- Agent performs trials (trial after trial)
- Agent only observes current state of environment and immediate reward
- Agent knows what are available actions, and can choose next action, without necessarily knowing its effects
- Uncertainty: effects of actions may be non-deterministic
- Goal of learning: Learn to choose actions that maximise cumulative reward
- Formally: Learn control policy, that is mapping
PI: States \rightarrow Actions
- PI, gamma, delta usually written as Greek letters π, γ, δ

DISTINGUISHING FEATURES OF THIS LEARNING PROBLEM

- **Delayed reward**
 - It is not possible to definitely evaluate the merit of an action on the basis of immediate reward only: which of the actions in the sequence are to be credited with causing cumulative reward?
- **Exploitation and exploration**
 - Agent searches among possible policies
 - Which exploration strategy produces most effective learning?
 - Trade-off between ***exploration*** and ***exploitation***: **dilemma** between choosing unexplored actions to gather new information (**explore**), or actions already known to produce high reward (**exploit**)

DISTINGUISHING FEATURES CTD.

- **Partially observable states**
 - Sometimes states may not be completely observed. When part of state cannot be observed, agent may have to consider recent past observations (states) when choosing action
- **Life-long learning**
 - Learn several tasks within the same environment - past experience from previous tasks may speed-up the learning of a new task (called transfer of skill between tasks; “**transfer learning**”)

REINFORCEMENT LEARNING AS MDPs

- Formulation based on Markov Decision Processes (MDPs)
 - S = set of states agent can perceive
 - A = set of actions agent can perform
 - Let agent perform action A_t in state S_t at time t . Environment responds with next state and reward:
 - state $S_{t+1} = \text{delta}(S_t, A_t)$, and
 - reward $r_t = r(S_t, A_t, S_{t+1})$
- Note: Functions r and delta are not known to the agent
- Sometimes it is convenient to use expected reward from executing action a in s :

$$R(s, a) = \sum_{s'} r(s, a, s') * P(s'|s, a)$$

where $P(s'|s, a)$ is the prob. of transition from s to s' given a

POLICY

- Policy π determines next action:

$$\pi: S \rightarrow A$$

- Given a policy π , *utility* of a state S_t is defined as *expected cumulative reward* from executing policy π from state S_t :

$$U(\pi, S_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0,1,\dots,\infty} \gamma^i r_{t+i}$$

where for all i , $A_{t+i} = \pi(S_{t+i})$ and $S_{t+i+1} = \delta(S_{t+i}, A_{t+i})$

- $U(\pi, s)$ is *discounted cumulative reward* achieved by policy π from initial state s
- For non-deterministic systems, this has to be averaged probabilistically over possible executions
- Equivalent notations: $U(\pi, s) = U^\pi(s)$

BELLMAN EQUATIONS

- Bellman equations for a fixed policy:

$$U^{PI}(s) = R(s) + \gamma * \text{SUM}_{s'} [P(s'|s, PI(s)) * U^{PI}(s')]$$

- Here, $R(s)$ is expected reward from making the action according to PI in state s :

$$R(s) = \text{SUM}_{s'} [P(s'|s, PI(s)) * r(s, PI(s), s')]$$

- Equivalent formulation:

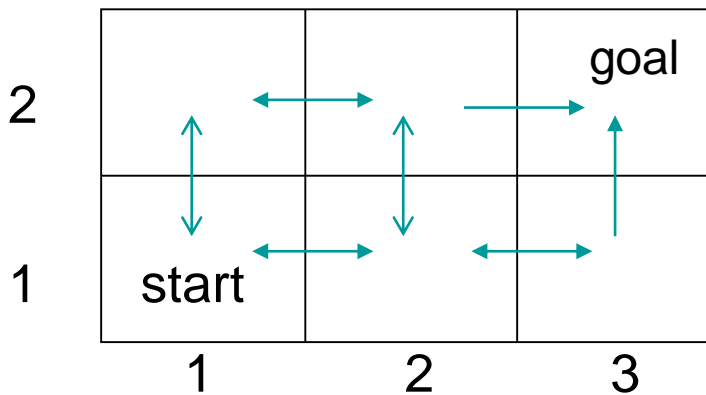
$$U^{PI}(s) = \text{SUM}[s'] P(s'|s, a) * [r(s, a, s') + \gamma * U^{PI}(s')]$$

OPTIMAL POLICY

- *Optimal policy* π^* maximises $U(\pi, s)$ for all s over π :
For all s , $\pi^*(s) = \operatorname{argmax}_{\pi} U(\pi, s)$
- We write: $U^*(s) = \max_{\pi} U(\pi, s)$
- $U^*(s)$ = highest possible expected cumulative reward achievable from s

EXAMPLE

- Consider a simple robot world consisting of a 3x2 grid. Robot can move horizontally or vertically between adjacent cells; the goal is to arrive into the topmost rightmost cell as quickly as possible.



- How can this be formulated as a reinforcement learning task? How may reward function be appropriately defined? With discount or without?

REWARDS AND BEST POLICY

- Reward can appropriately be defined for example as: $\text{Reward} = 100$ for the two possible moves into the top-right cell, and 0 for all other states and actions; γ is, say, 0.9.
- Also, it is appropriate to restrict the actions in the top-right cell just to one possible action that keeps the robot in this cell

EXERCISE

- How can the state utilities for this simple problem be calculated?
Use Bellman equations.

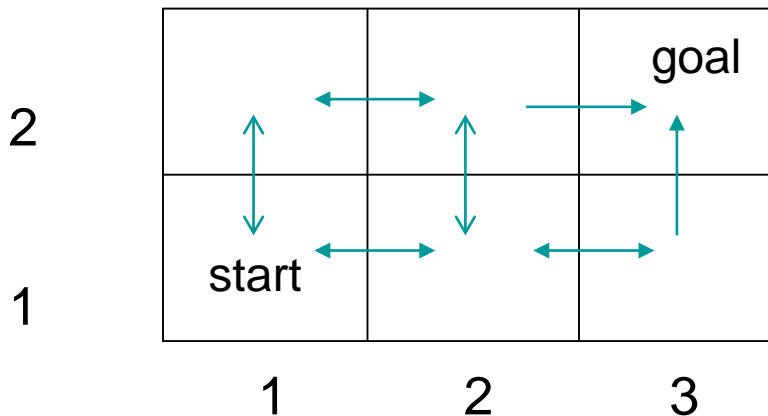
Utilities for deterministic case

- According to this definition of reward function, the best policy is always to move towards the top-right cell. This policy achieves U^* cumulative reward values of states shown below:

90	100	0
81	90	100

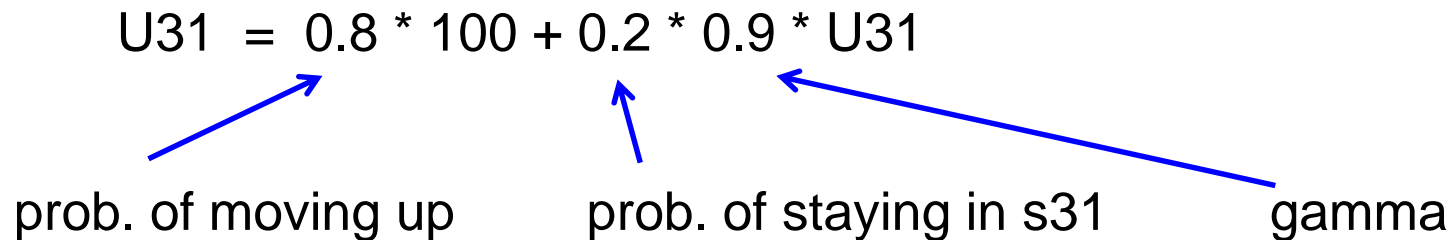
EXERCISE, CTD.

- What if the actions have non-deterministic effects? For example, due to mechanical errors, in only 80% of the cases the result of an action is as intended, but in 20% of the cases the robot fails to move (stays where it was before the action). What is the utility of state s_{31} (bottom-right cell) in this case? What are the utilities of all the states in this case?



Utility U31

- In s31, robot takes action “up”. Result is state s32 (with prob. 0.8), or s31 with prob. 0.2.
- So:

$$U_{31} = 0.8 * 100 + 0.2 * 0.9 * U_{31}$$


prob. of moving up prob. of staying in s31 gamma

- $U_{31} = 97.56$

A solution for U31, U21, U11

- Write Bellman equations for all the relevant states (1,1), (2,1), and (3,1). Write U31 as abbreviation for U(s31) etc.
- Resulting linear equations can be stated as constraints on real numbered variables, and directly solved e.g. with CLP(R) in Prolog:

?- {U31 = 0.8*100 + 0.2*0.9*U31,
U21 = 0 + 0.9*0.8*U31 + 0.9*0.2*U21,
U11 = 0.8*0 + 0.9*0.8*U21 + 0.2*0 + 0.9*0.2*U11}.

U11 = 75.21655228449967,

U21 = 85.66329565734684,

U31 = 97.5609756097561

EXERCISE

- What are the utilities of states given optimal policy if an action may produce transition into the current cell or any of the adjacent cells. The probability of any unintended transition is 0.1, and the rest is the prob. of the intended transition.
- For example, for action up in s31, we have:
 $P(s_{21} | s_{31}, \text{up}) = 0.1$
 $P(s_{31} | s_{31}, \text{up}) = 0.1$
 $P(s_{32} | s_{31}, \text{up}) = 0.8$
- $P(s_{22} | s_{21}, \text{up}) = 0.7$, etc.

EXERCISE CTD.

- Bellman equations for this case solved with CLP(R) in Prolog:
- ?- { $U_{31} = 0.8 \cdot 100 + 0.1 \cdot 0.9 \cdot U_{31} + 0.1 \cdot 0.9 \cdot U_{21}$, $U_{21} = 0.9 \cdot (0.7 \cdot U_{31} + 0.1 \cdot (U_{21} + U_{22} + U_{11}))$, $U_{11} = 0.9 \cdot (0.8 \cdot U_{21} + 0.1 \cdot U_{11} + 0.1 \cdot U_{12})$, $U_{22} = 0.7 \cdot 100 + 0.9 \cdot (0.1 \cdot U_{21} + 0.1 \cdot U_{22} + 0.1 \cdot U_{12})$, $U_{12} = 0.9 \cdot (0.8 \cdot U_{22} + 0.1 \cdot U_{12} + 0.1 \cdot U_{11})$ }.
- $U_{31} = 96.12605235204434$,
- $U_{21} = 83.05230711511501$,
- $U_{22} = 93.14703232706051$,
- $U_{11} = 73.72170648367864$,
- $U_{12} = 80.98990863627114$.

Alternative definition of rewards and discount to make agent find shortest path to goal

- $\text{Gamma} = 1$
- Immediate rewards for moving into terminal state s32 is 100. All other immediate rewards are -1
- This also makes shortest paths optimal policies