Data-driven Intelligent Systems

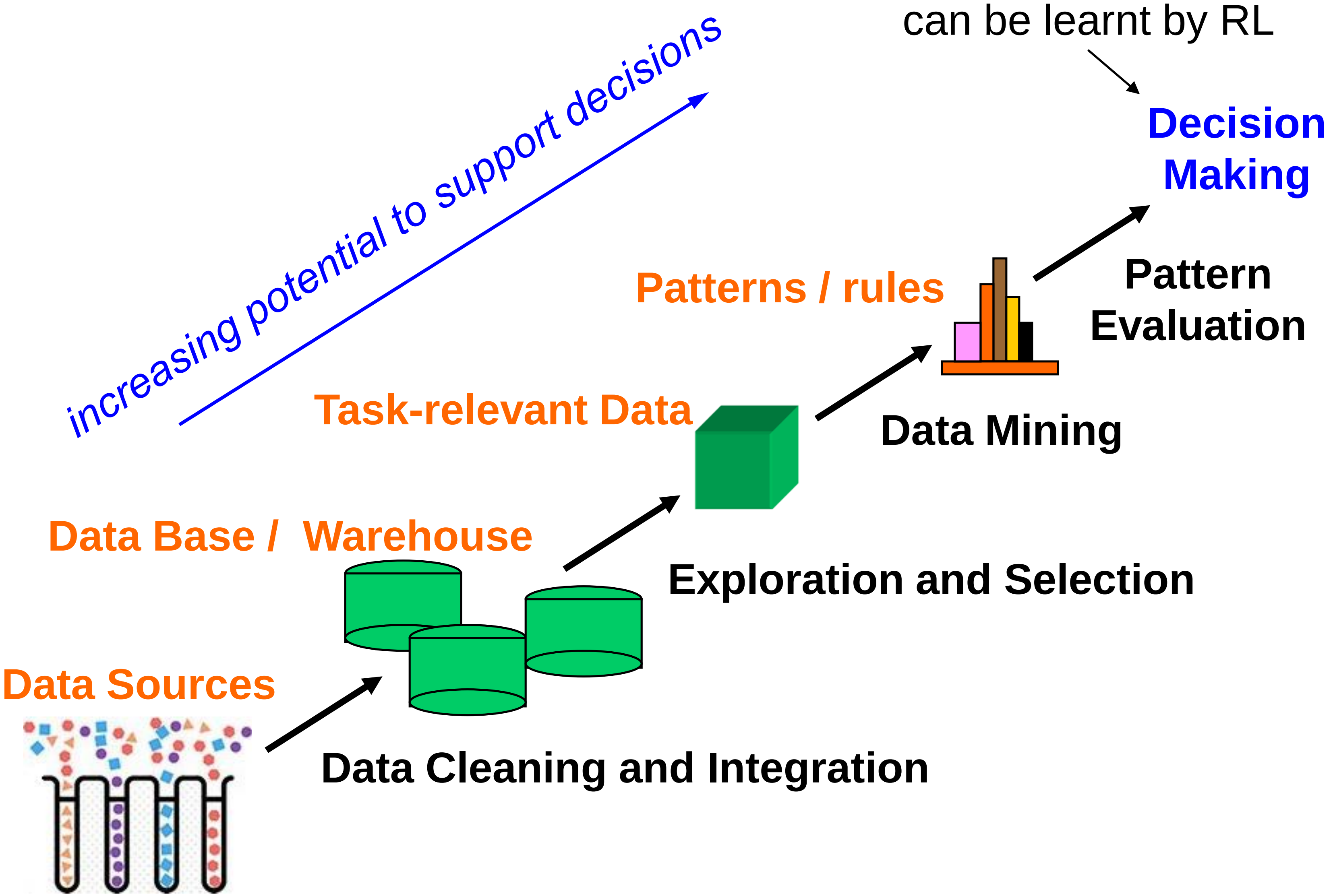
Lecture 16

Reinforcement Learning I

KNOWLEDGE
TECHNOLOGY

http://www.informatik.uni-hamburg.de/WTM/

Knowledge Discovery from Data



Outline

- Agents
- Markov Decision Process (MDP), Estimation of Return, Action Selection
- Tabular vs. Deep RL
- TD-learning, SARSA, Actor-Critic
- Example Applications

Intelligent Agents

- Agents
  - Perceive their environment via sensors
  - Act in their environment through effectors
- Rational agents
  - Do the “right“ thing that lets them “succeed“
  - A performance measure quantifies success
- Autonomous agents
  - Express behaviour, which depends on their own experience

http://www.informatik.uni-leipzig.de/~brewka/lehre1.html

Types of Agents

- Reflexive agents
  - perceive information about the world state
  - use condition-action rules to choose actions
- Agents with internal state
  - store information about previous world states
  - store information about the effect of actions
- Goal-based agents
  - have information about goal states
  - infer actions from desired goals
- Agents with some use function
  - know a measure of desirability of certain states
  - can therefore choose between different goals

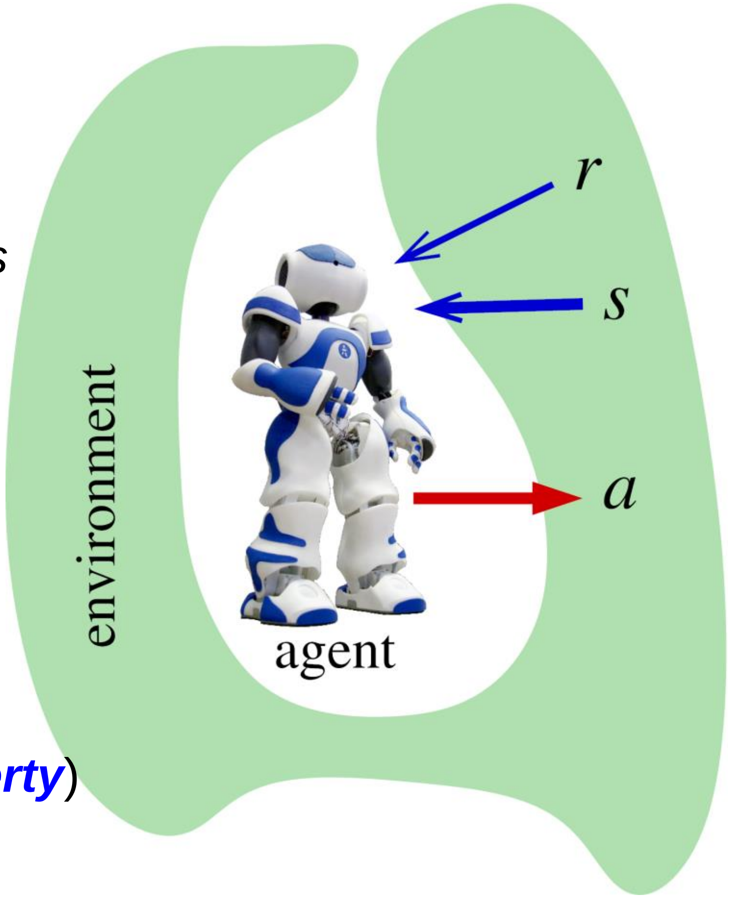
Types of Environments

- Accessible vs. inaccessible
  - Is the information, which is relevant to choose actions, accessible via sensors?
- Deterministic vs. non-deterministic
  - Is the next world state uniquely determined given the current state and the current action?
- Episodic vs. non-episodic
  - Does the quality of the actions depend on previous actions?
- Static vs. dynamic
  - Does the world state change independently of the agent‘s actions?
- Discrete vs. continuous
  - Is the number of possible states and actions limited?

Outline

- Agents
- Markov Decision Process (MDP), Estimation of Return, Action Selection
- Tabular vs. Deep RL
- TD-learning, SARSA, Actor-Critic
- Example Applications

Markov Decision Process



Agent-environment interaction:

- Loop:
  - agent percieves information about state *s*
  - agent performs action *a*
  - agent may receive reward *r*
  - state & action at next time step: *s`, a`*

Markov Decision Process (***MDP***):

- Fixed transition probabilities

  $P(s`) = P(s`|s,a)$
  - not dependent on history (***Markov property***)
- Fixed reward probability

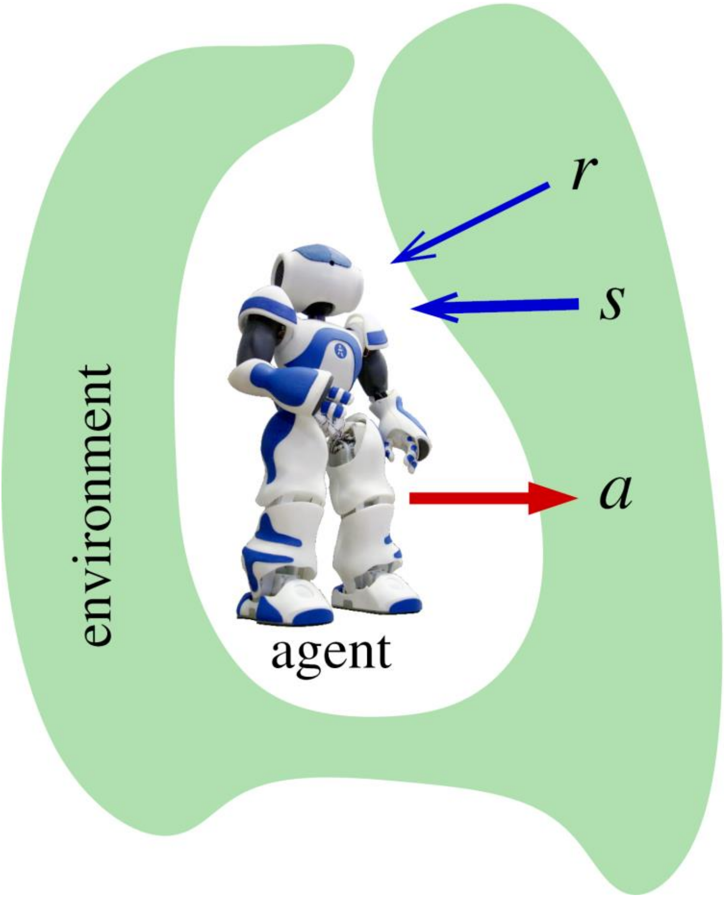
  $r = r(s`, s, a)$
  - might depend only on s`

An ***MDP*** is a tuple (*S*, *A*, *P*, *R*)

i.e. the *sets* of states, actions, transition probs., rewards

Markov Decision Process

The agent`s actions will be oriented towards gaining maximum *accumulated future reward*, which is the ***Return***.



How is the Return estimated?

- This will be done recursively:

  Return at current time *t*

  = *Return* at next time *t+1*

  + *Reward* obtained during transition

RL: The agent will store its best estimates of the return either

- as values *V* depending on states *s* or
- as values *Q* depending on transitions *(s,a)*

Bellman Equation – Dynamic Programming

How much reward will the agent accumulate in the future?

- **Return**:

  $$R(t) = r(t+1) + \gamma \cdot r(t+2) + \gamma^2 \cdot r(t+3) + \dots = \Sigma_{t`} \gamma^{t`} \cdot r(t+t`)$$
  $$= r(t+1) + \gamma \cdot R(t+1)$$

  - ***Discount factor γ*** <1: more distant rewards count less
  - Formula is recursive → *R* can be re-estimated at any time

- Agent`s estimate of the return (***expected return / Q-values***):

  $$Q^\pi(s; a) = r(t+1) + \gamma \cdot Q^\pi(s`; a`)$$

  - The estimate depends on the ***policy π***,
    i.e. which action strategy is the agent using.

- Alternative: estimate return by ***state values***:

  $$V^\pi(s) = r(t+1) + \gamma \cdot V^\pi(s`)$$

  (some algorithms use Q-values, others state values)

Markov Decision Process

From learning, the agent now has good estimates about the Return.

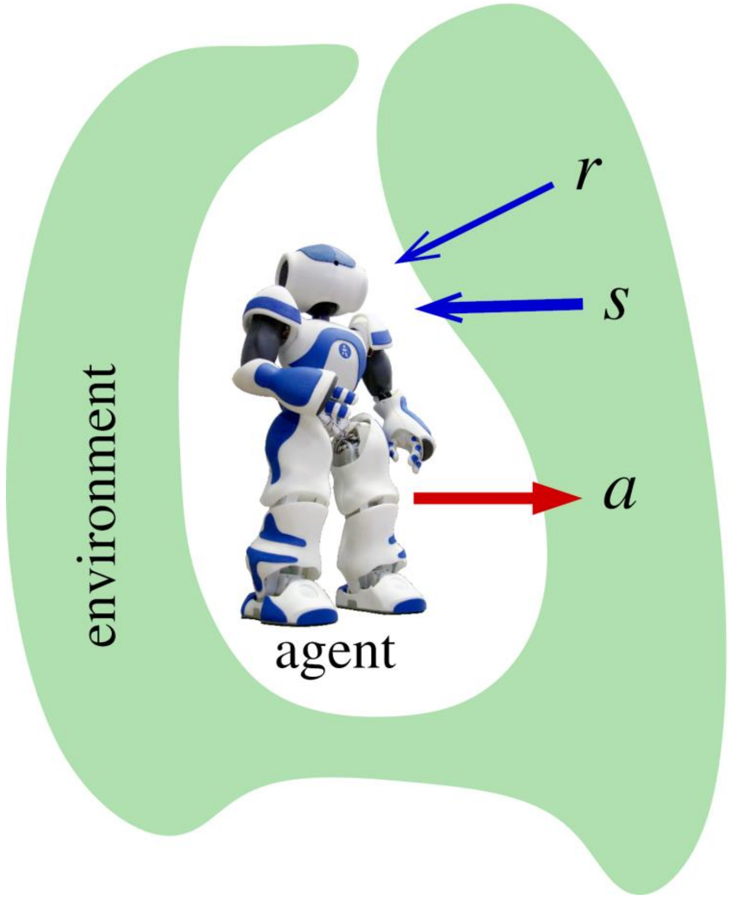
How does the agent use that knowledge to choose its actions?

- Follow an action selection strategy:
  - Prefer those actions that lead to the largest Return at the next time step.

The mapping of states to actions is called the agent`s ***policy***.

- A better policy will lead to larger Return.
- A better estimate of Return will allow better policies.

Over time, estimate of Return, and policy, will converge.



Maximising Return (Exploiting) *while Exploring*

Policy: the mapping from states to actions (here: discrete actions)

Action selection strategies are, e.g.:

- **greedy** action selection: $a = \text{argmax}\, Q(s; a)$ $\leftarrow$ no exploration!
- **ε-greedy** action selection:

  $i^* := \text{argmax}_{i'}\, Q(s; a_{i'})$ $\leftarrow$ the “best” action

  $P(a_{i*}=1) = 1-\varepsilon$ $\leftarrow$ choose best action a_{i*} with probability (else, any action, randomly)

- **Boltzmann (softmax) action selection**:

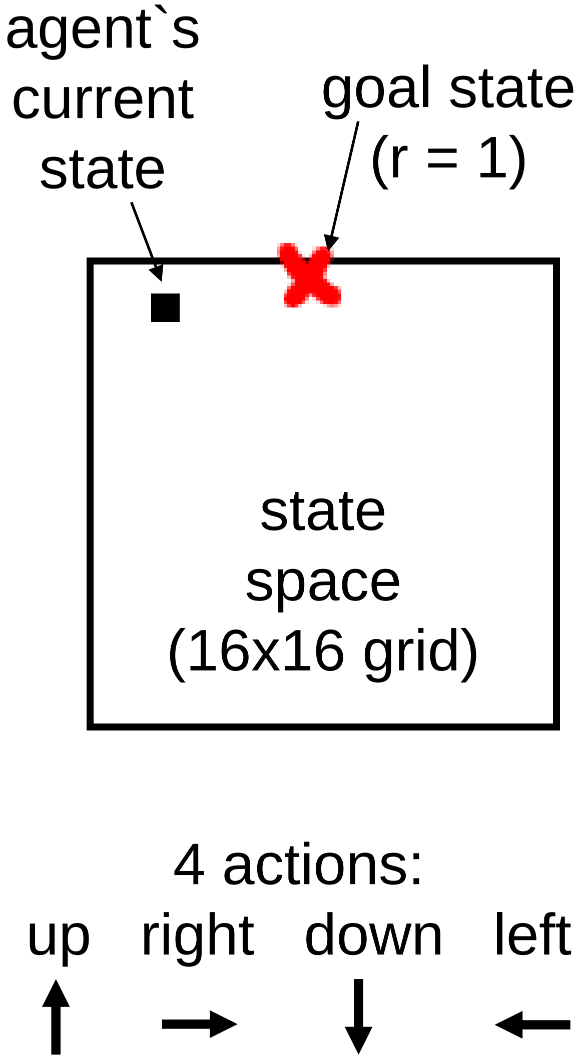
$$P(a_i=1) = \exp(\beta \cdot Q(s; a_i)) \,/\, \Sigma_{i'} \exp(\beta \cdot Q(s; a_{i'}))$$

- small ε or large $\beta \rightarrow$ prefer large-Q actions (exploitation)
- large ε or small $\beta \rightarrow$ choose more randomly (exploration)

Outline

- Agents
- Markov Decision Process (MDP), Estimation of Return, Action Selection
- Tabular vs. Deep RL
- TD-learning, SARSA, Actor-Critic
- Example Applications

An Example Scenario for Tabular RL



Objective: learn to move to rewarded state.

One-hot state encoding: s = (0, ..., 1, ..., 0)

One-hot action encoding: a = (0, 1, 0, 0)

Learning: sample (s, a) and (r, s`, a`) over many trials, and learn values Q(s, a).

the Q-table

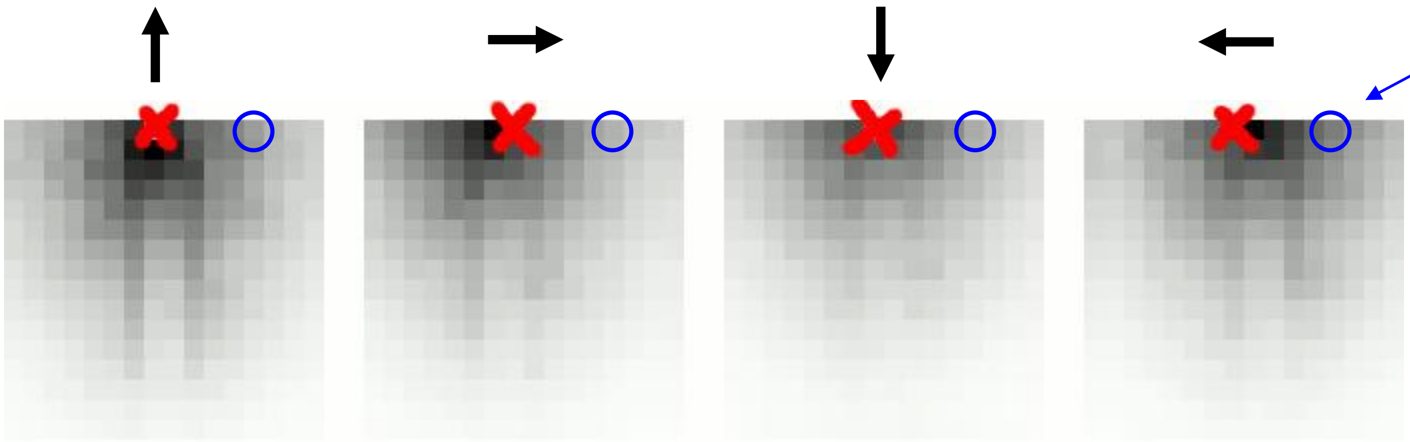
Initialize agent to random start state every time after achieving goal.

Meanwhile, agent will use learnt knowledge: ε-greedy or Boltzmann action selection prefers actions with large Q-values.

After learning: go straight to goal by choosing max-Q action in each state.

An Example Scenario – Learning Dynamics

- Values $Q(s,a)$ will be initialized with 0 for all s, a.
- First, non-zero Q-values will build up near the rewarded state (i.e. after accidentally hitting the reward).



"←" is the best action for this state

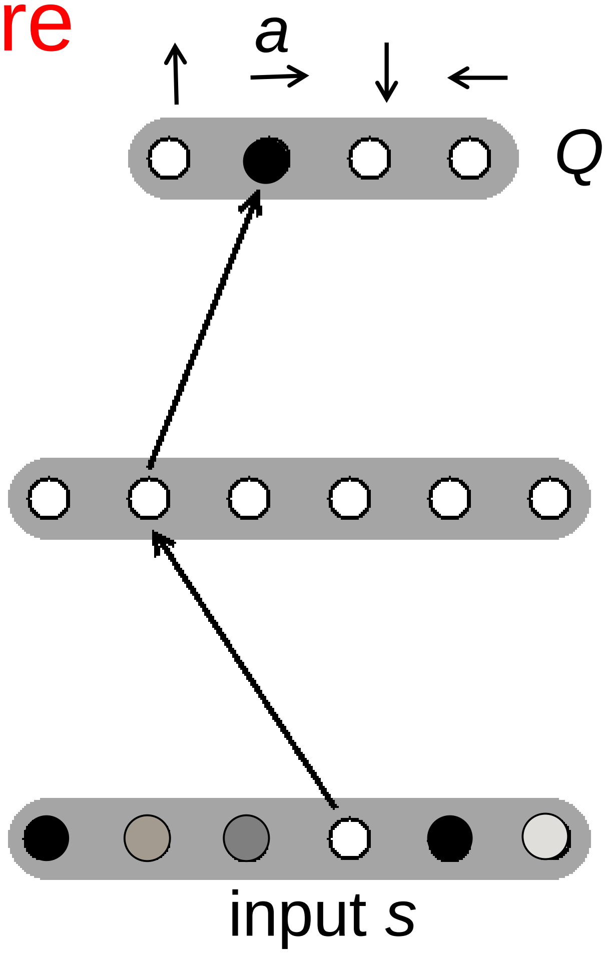
The resulting Q-values for the 4 actions (dark=large)

- After convergence, Q-values will "ramp up" towards the goal state (slope depends on γ).
- For a given state s, that Q-value with the best action will be the largest

Deep RL Architecture

Function approximation with a (deep) neural network

- Network input: *s*
- Network output: *Q(a)*
  - One unit for each action
  - Action selection based on those *Q*



Alternative:

- Network input: *(s, a)*
- Network output: *Q*
  - one output unit
- → for action selection, need multiple computations of output

Outline

- Agents
- Markov Decision Process (MDP), Estimation of Return, Action Selection
- Tabular vs. Deep RL
- TD-learning, SARSA, Actor-Critic
- Example Applications

Temporal Difference (TD) Learning
(Learning during one Step/Transition)

- Current estimated value at current state *s* and action *a*: $Q(s; a)$
- Estimated value at next state *s'* if then performing *a'*: $Q(s'; a')$
- Reward obtained during transition (or on arrival at *s'*): r
- Bellman equation allows a better estimate of the current value:

$$Q(s; a) = r + \gamma \cdot Q(s'; a')$$

- TD-learning advises to adapt the Q-value for the current (s,a):

$$Q(s; a) \leftarrow Q(s; a) + \underset{\text{learning rate}}{\eta} \cdot \underbrace{(r + \gamma \cdot Q(s'; a') - Q(s; a))}_{\text{TD error } \delta}$$

- The TD error (rather: the square of it; compare with the L2 norm) is usable like a cost function to update the parameters of a function that estimates *Q* given *s* and *a* as its inputs.

function approximation
e.g. with a neural model

The SARSA Algorithm (s,a,r,s',a')

- Init: read state *s*, select an action *a*, compute *Q(s,a)*
- Repeat until end of trial (e.g. when goal reached, out of bounds, etc.):
  - *Execute* action *a*
  - Read reward *r* and new state *s'*
  - *Select* next action *a'* (using ε-greedy or Boltzmann action selection)
  - Read new Q-value: $Q(s',a')$ (from Q-table or approximating function)
  - Compute TD error: $\delta = r + \gamma \cdot Q(s`; a`) - Q(s; a)$
  - Update Q-table entry: $Q(s; a) \leftarrow Q(s; a) + \eta \cdot \delta$ (TD-learning)

    Or: update parameters of approximating function to minimize δ
  - Set variables for next iteration: $s \leftarrow s'$, $a \leftarrow a'$, $Q \leftarrow Q'$

Repeat many trials (e.g. from diverse initial states) until entire state space is well learnt.

TD-variations: Q-, SARSA-, Actor-critic Learning

- Q-learning: update based on next ***best*** possible estimates
  $Q(s, a) \leftarrow Q(s, a) + \eta\,(r + \gamma\,\max_{a'} Q(s', a') - Q(s, a))$

  "**off-policy**" algorithm, because Q-value is **not** computed based on the actually chosen action a'

- SARSA: update estimates based on next ***chosen*** action
  $Q(s, a) \leftarrow Q(s, a) + \eta\,(r + \gamma\, Q(s', a') - Q(s, a))$

  "**on-policy**", because Q (or V) values are computed using the actually chosen action a' and next state s'

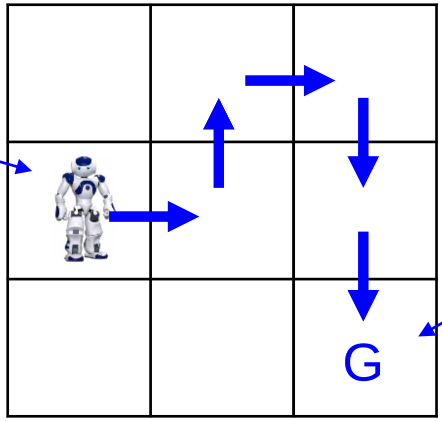
- Actor-Critic: update of ***state value***
  $V(s) \leftarrow V(s) + \eta\,(r + \gamma\, V(s') - V(s))$

  (requires separate learning of the policy)

Actor-critic Learning – Quantitative Visualization

- update of ***state value:*** $V(s) \leftarrow V(s) + \eta \underbrace{(r + \gamma\, V(s') - V(s))}_{\delta}$
- 3x3 grid world
- reward *r=1* provided if agent reaches state at position (3,3)
- Init: *V(s) = 0* at all states

agent at initial state (typically randomly chosen)



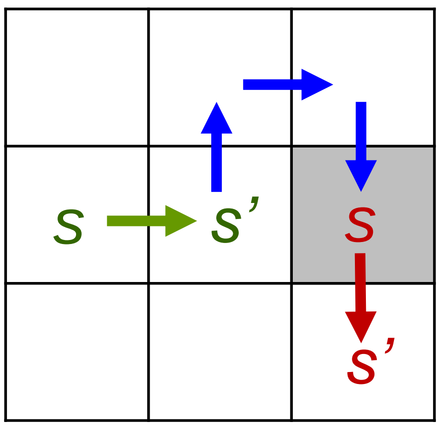
goal state (*r=1* will be provided)

Actor-critic Learning – Quantitative Visualization

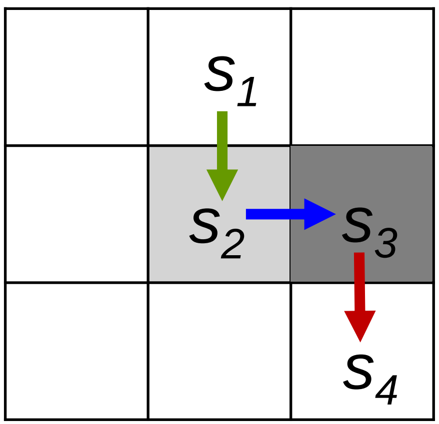
learning rate η=0.5 discount factor γ=0.9

- update of ***state value:*** $V(s) \leftarrow V(s) + \eta\,(r + \gamma\, V(s') - V(s))$

Rollout #1



Rollout #2 (new init state)



1st step: $V(s) = V(s') = 0$
$\rightarrow \delta=0$, hence nothing learnt
$\rightarrow$ same for 2nd, 3rd & 4th step

5th step (into goal state s'):
$V(s) = 0 + 0.5\cdot(1+0.9\cdot0-0) = 0.5$

1st step: $V(s_1) = V(s_2) = 0 \;\rightarrow \delta=0$

2nd step (s_2 to s_3):
$V(s_2) = 0 + 0.5\cdot(0+0.9\cdot0.5-0) = 0.5\cdot0.45$

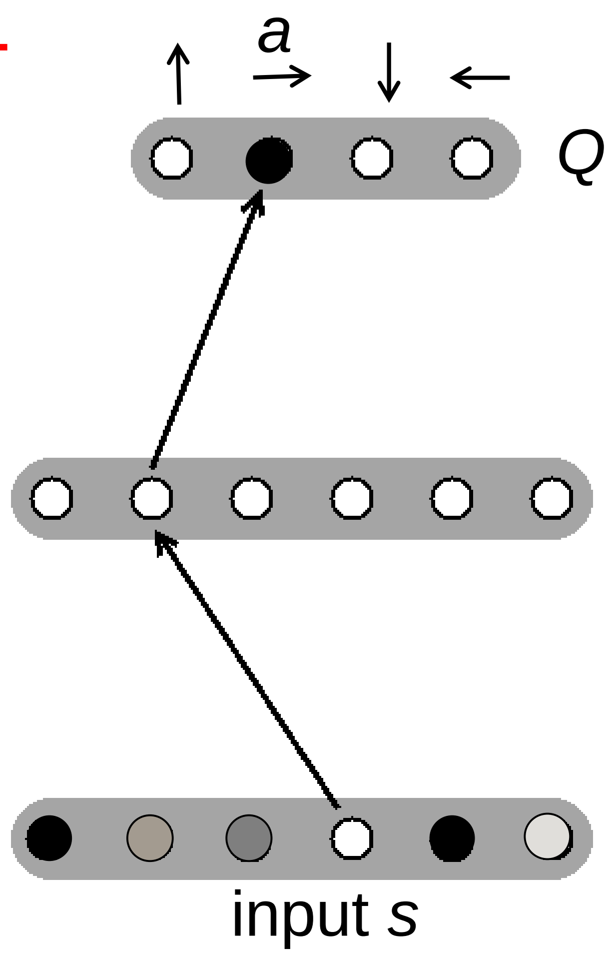
3rd step (s_3 to goal state s_4):
$V(s_3) = 0.5 + 0.5\cdot(1+0.9\cdot0-0.5) = 0.75$

TD Learning in Deep RL

- Network input: *s*
- Network output: *Q(a|s)*
  - One unit for each action
  - Action selection based on those *Q*
- Taking one step provides all info needed to compute the TD error:

$$\delta = \underbrace{r + \gamma \cdot Q(s`; a`)}_{\text{targeted output}} - \underbrace{Q(s; a)}_{\text{actual output}}$$

- Adapt network parameters by error backpropagation of the TD error
  - (apply TD error only to the action unit *a*, i.e. the action that caused this experience)



Experience Replay

Performing many trials in the environment can be costly.

Solution: learn multiple times from the experience:

- Store all ***experiences*** *(s, a, r, s', a')* in replay memory
  - They represent environmental behavior
  - Independent of learnt parameters
- Randomly sample experiences and learn from them
  - One experience is sufficient to compute one TD-update
  - Compute the needed *Q* (or *V*) values with current parameters

Advantage: Experience Replay supplies a *homogeneous distribution of learning samples*, independent of current exploratory behaviour, which may be biased

- In practice: Replay buffer is finite, drop old experiences
  - Earliest agent behaviour may be irrelevant anyway

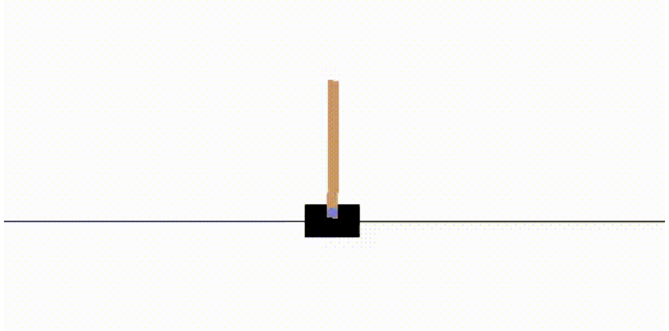
Outline

- Agents
- Markov Decision Process (MDP), Estimation of Return, Action Selection
- Tabular vs. Deep RL
- TD-learning, SARSA, Actor-Critic
- Example Applications

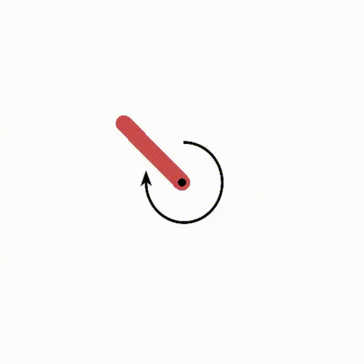
gym.openai.com I/II

Reimplementation of classical problems, e.g.:

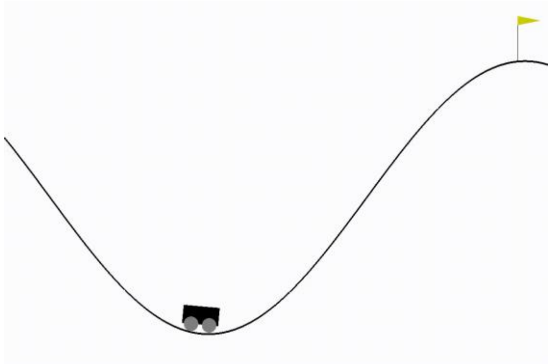
Cart Pole (balancing)



Pendulum Upswing



Mountain car



State: angle, angular speed, x-position

Actions: left or right force

Reward: =1 if near vertical

More difficult than cart pole.

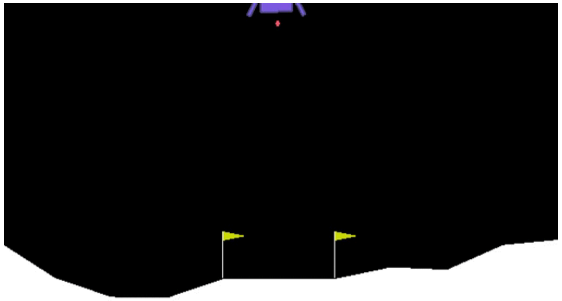
Similar behaviour to pendulum upswing.

All actions are discrete; states also mostly discretized.

gym.openai.com II/II

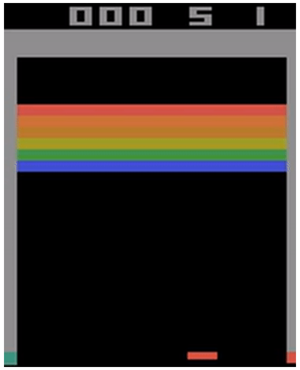
Collection of many newer problems, e.g.:

Lunar Lander



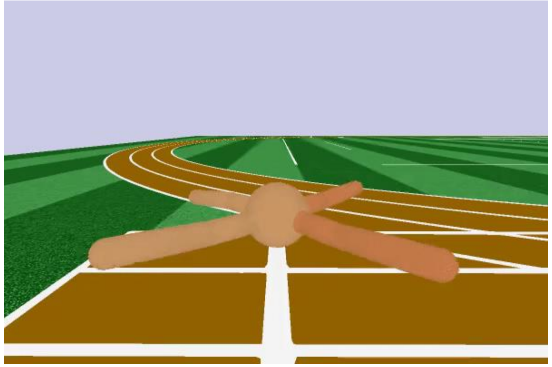
Continuous actions

Breakout



High-dim. input (full image)

Ant (RoboSchool)



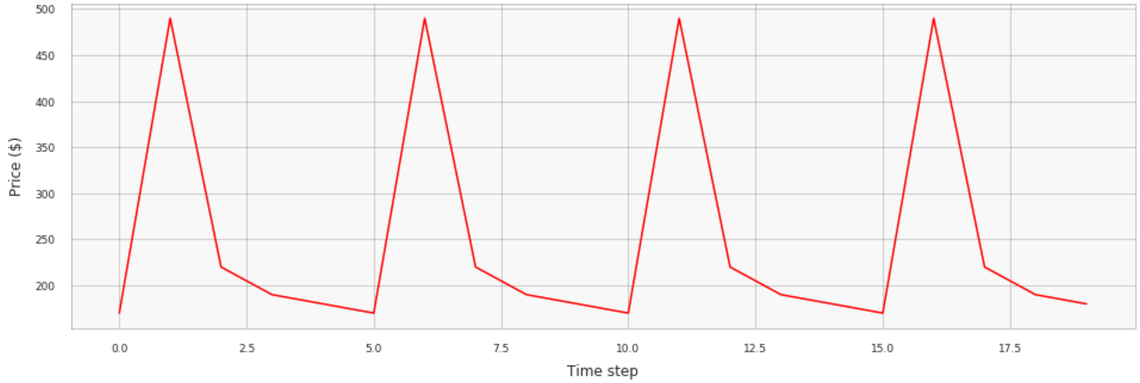
Complex physical behaviour

All environments have compatible interface for easy test of RL algorithms in many environments.

RL Application: Price Optimization

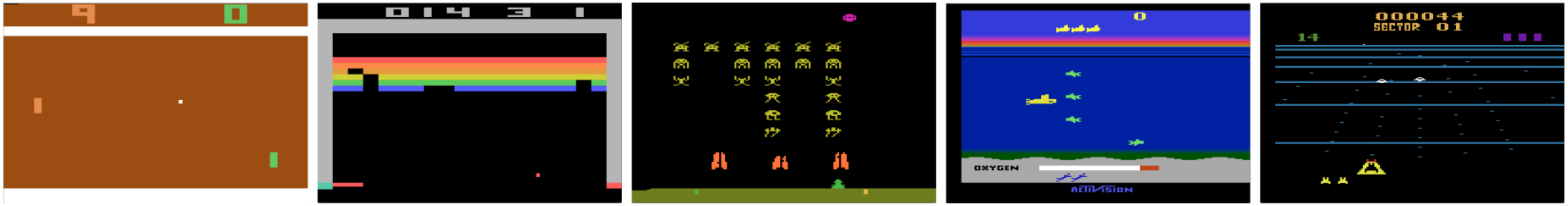
Commerce application: set a product pricing policy

- Requires a model of customer behavior: buy more when price drops
- Define the environment:
  - Encode the state s_t at time step t as a vector of prices p for all previous time steps concatenated with one-hot encoding of the time step itself:
    - $s_t = (p_{t-1}, p_{t-2}, \ldots, p_0, 0, \ldots) \mid (0, \ldots, 1, \ldots, 0)$
  - The action a is an index in the array of valid price levels
  - The reward r is the profit of the seller
- Result: a complex pricing strategy with price surges and discounts → Hi-Lo pricing strategy used by many retailers



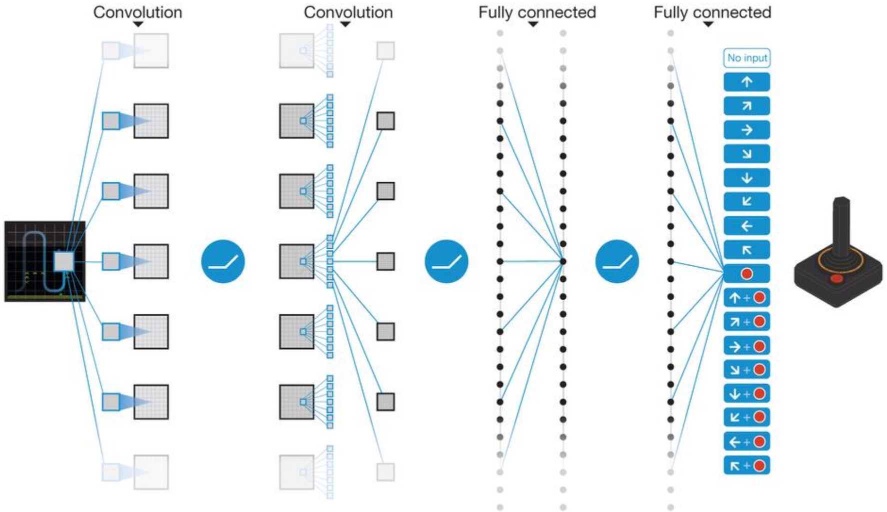
https://blog.griddynamics.com/deep-reinforcement-learning-for-supply-chain-and-price-optimization/

Deep RL Application: Atari Playing



- Multiple Atari Games (at some games still worse than humans)
  - Deep Reinforcement Learning (RL) (Mnih et al., 2015)

input
(screen)

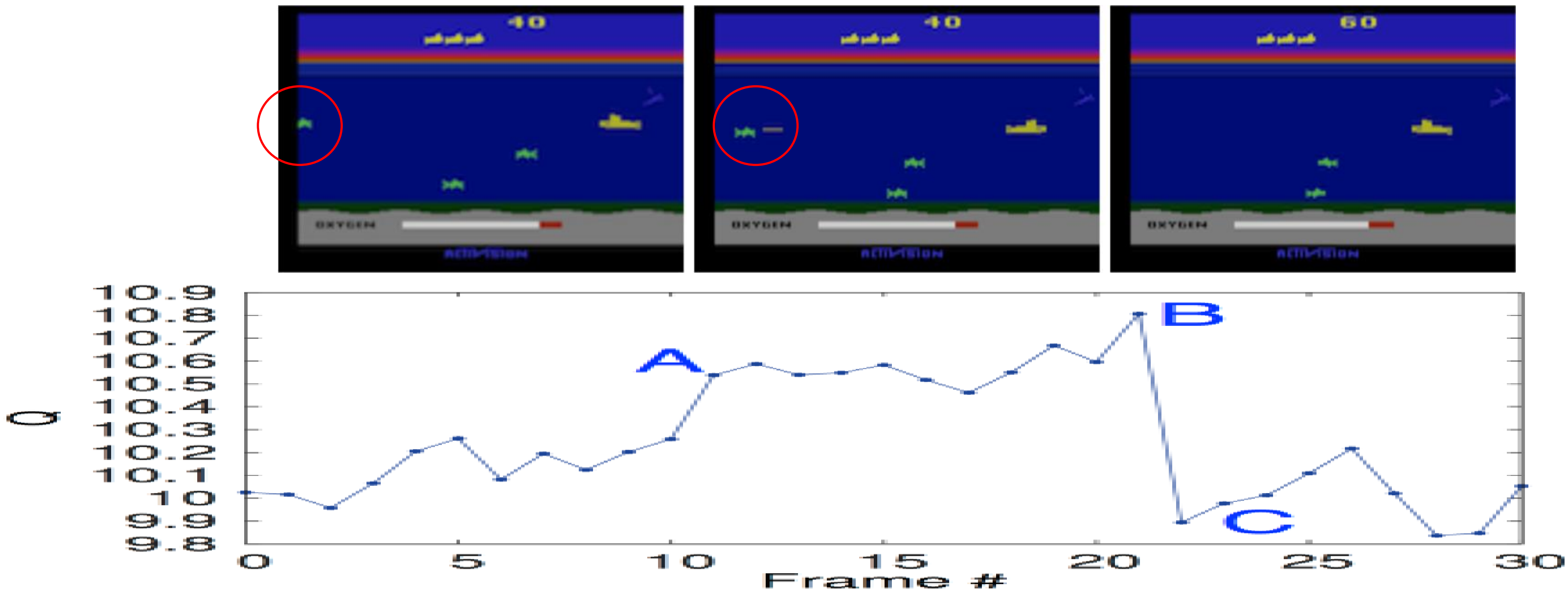


output
(Q-values of
discrete
actions,
determined
by RL)

CNN with ReLU

Deep RL – Q values in Seaquest

A enemy appears **B** before impact **C** baseline



- Raised Q-values reflect the upcoming reward, given when the missile hits the enemy (after **B**)

Mnih et al. (2014) Playing Atari with Deep Reinforcement Learning.

Robot Arms Picking Objects with Deep RL

- Real-world RL difficult as requires much training experience
  - Early examples: pendulum balance & swing-up
- Real deep RL:
  - Robot arms learn to pick objects using top-mounted camera
  - Large-scale: 14 robots; two months; 800.000 pick attempts

- CNN processes images
- Simplified pick actions from top:
  - always from top, given height
  - only x-, y-pos & angle learnt to maximize grasp success

Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection (2016) https://arxiv.org/abs/1603.02199

RL for Language Generation

- Learn to conduct dialogues between two virtual agents:
  - Reward sequences that display three useful conversational properties: informativity, coherence, and ease of answering.
  - Using policy gradient methods

  Deep RL for Dialogue Generation https://arxiv.org/abs/1606.01541

- Learn to play text games:
  - Reward depends on the finally achieved state.
  - Deep reinforcement relevance network (DRRN) represents action and state spaces with separate embedding vectors, which are combined with an interaction function to approximate the Q-function.

  Deep RL with a Natural Language Action Space
  https://arxiv.org/abs/1511.04636

Summary

- RL: agents learn to find action strategies given temporally delayed rewards (which extends (un)supervised learning)
- MDP supplies the theoretical framework
  - Bellman equation for recursive estimation of state/Q value
  - Repetitive exploration of agent
- Tabular & Deep RL
- Many applications and promising research direction
- Further reading & links:
  - Sutton & Barto: Reinforcement Learning: An Introduction (available online as 2nd edition in progress)
  - Difference SARSA vs. Q-learning: https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/
  - Software: https://ray.readthedocs.io/en/latest/rllib.html