Reinforcement learning

Jiří Kléma

Department of Computer Science,
Czech Technical University in Prague

Lecture based on **AIMA book** and its accompanying slides

https://cw.fel.cvut.cz/wiki/courses/b4m36smu

Reinforcement learning (RL) in a nutshell

Imagine playing a new game whose rules you don't know;
after a hundred or so moves, your opponent announces, "You lose".

Russell and Norvig
Introduction to Artificial Intelligence

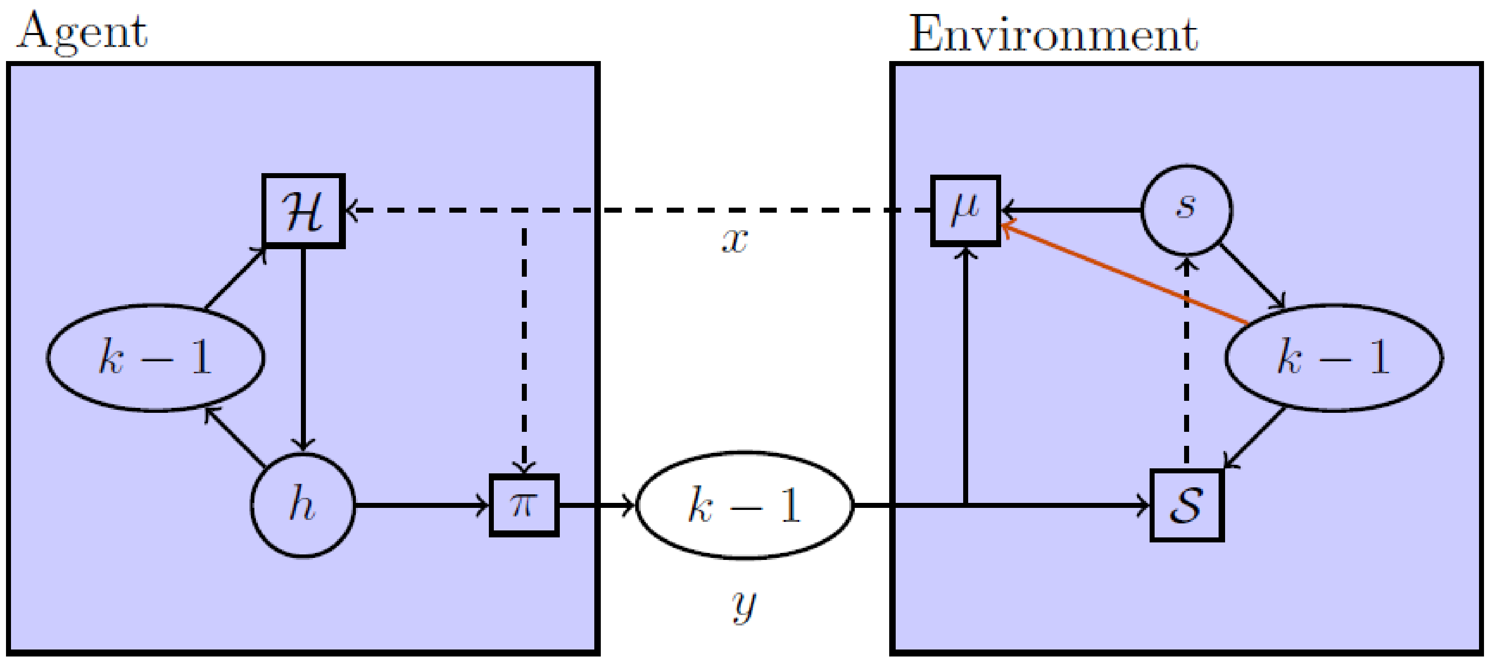
Agenda

- A formal definition of RL
  - the distinction from MDP known from ZUI course,
  - a definition in terms of our SMU general framework,
  - (necessary) simplifications from the most general form.
- Passive learning
  - model-free and model-based approaches,
  - direct utility estimation, adaptive dynamic programming (ADP), temporal difference (TD) learning,
  - illustrative, but not really useful in practice.
- Active learning
  - exploration-exploitation dilemma,
  - active ADP agent, Q-learner based on TD learning.
- Generalizations and applications
  - utility function approximations in large state spaces.

Reinforcement learning slightly more formally

- Agent interacts with an environment,
- it must learn to act optimally in it,
- we assume that the world is a Markov decision process (MDP),
- however, neither its transition model nor reward function is known,
- the goal is to learn the optimal policy,
- the main difficulties
  - action effects unknown and non-deterministic
  - rewards can be infrequent/delayed,
  - greedy decision making can be far from optimal,
  - world is large and complex.

Reinforcement learning in the SMU general framework



Sequential on-line learning assumed

The environment is fully observable . . . $o_k = s_k$

Deterministic rewards given by the current state . . . $\mu_r(r_k|s_{k-1}, y_{k-1}) \rightarrow R(s_{k-1})$

Agent's hypotheses may differ . . . the most complex form captures the transition model $\mathcal{S}(s_k|s_{k-1}, y_{k-1})$, the reward function $R(s_{k-1})$ and uses them to compute the utility functions $U(s)$ or $Q(s, a)$

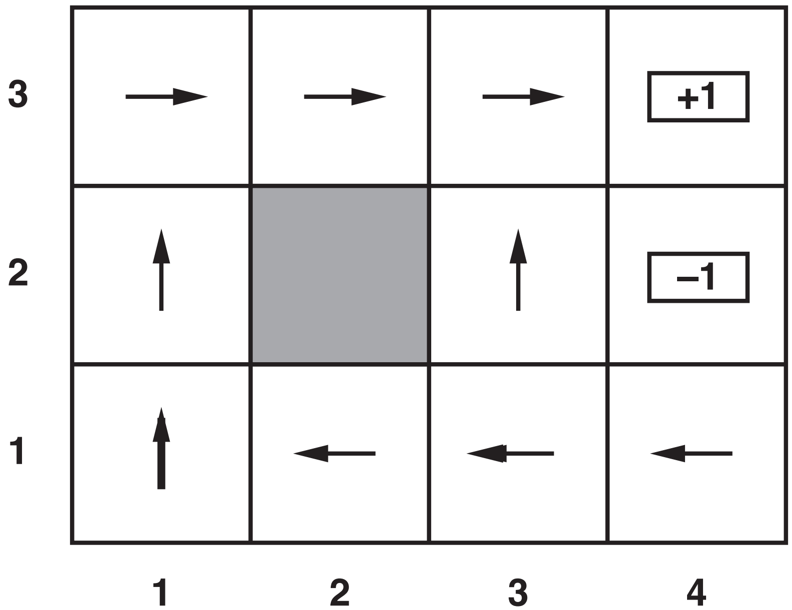
Passive and active learning

- Passive learning
  - the agent employs a fixed policy π,
  - it learns how good the policy is by interaction with the environment, e.g., it learns $U^\pi(s)$,
  - analogous to policy evaluation in policy iteration, (however, the big difference is that the model of the environment is unknown now).
- Active learning
  - the agent searches for an optimal (or at least good) policy,
  - it **explores** (many) different actions in (many) different states,
  - analogous to learning and solving the underlying MDP.

Model-based and model-free learning

- Model-based learning
  - learn the MDP model ($\mathcal{S}$ and R), or an approximation of it,
  - use the model to guess the state utility and find optimal policy,
  - more difficult to tailor to the environment
    (as straightforward application can be slow).
- Model-free learning
  - derive the optimal policy without explicitly learning the environment model,
  - typically stem from the estimation of Q, the utility of state-action pairs,
  - easier to apply independently of the environment,
    may fail in complex worlds.

Passive reinforcement learning



	1	2	3	4
3	0.812	0.868	0.918	+ 1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388

- Simply follow the fixed policy for a long time (many epochs if episodes exist),
- use interaction/epochs as training sequence(s) in learning $U^{\pi*}(s)$

$(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$

$(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (3,2)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$

$(1,1)_{-.04} \rightarrow (2,1)_{-.04} \rightarrow (3,1)_{-.04} \rightarrow (3,2)_{-.04} \rightarrow (4,2)_{-1}$

Passive reinforcement learning

- Approach 1: direct utility estimation (DUE)
  - the most simple approach, model-free,
  - learn the expected **reward-to-go** from the observed rewards-to-go,
  - reward-to-go of a state s is the sum of the (discounted) rewards from that state until a terminal state is reached,
  - the expected reward-to-go matches the true state utility given the policy,
  - an observed reward-to-go represents a sample of this quantity for each state visited,
  - estimate $U^\pi(s)$ as (running) average total reward of epochs containing s (calculating from s to the end of epoch),
  - assume $\gamma = 1$: reward-to-go$((1,2)) = (0.76 + 0.84)/2 = 0.8$

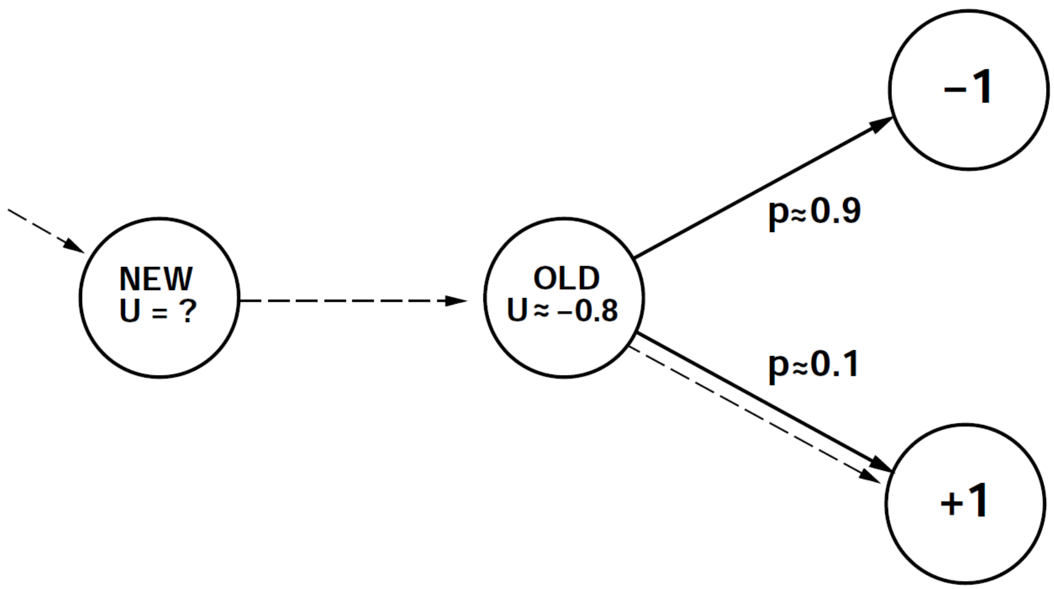
$(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$ ... 0.76

$(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (3,2)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$ ... 0.84

$(1,1)_{-.04} \rightarrow (2,1)_{-.04} \rightarrow (3,1)_{-.04} \rightarrow (3,2)_{-.04} \rightarrow (4,2)_{-1}$

Passive reinforcement learning

- Approach 1: direct utility estimation
  - easy, reduces RL to supervised learning
    (samples = pairs of state and observed reward-to-go),
  - convergence is guarranteed for infinitely many trials,
  - **big disadvantage is its slow convergence**,
  - why? does not employ dependence among state utilities!
    $U^\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{S}(s'|s,\pi(s))U^\pi(s')$



Passive reinforcement learning

- Approach 1: adaptive dynamic programming (ADP)
  - a complex model-based method,
  - step 1: it learns the transition model $\mathcal{S}$ and the reward function R
    * $R(s)$ is deterministic, whenever a new state entered, store the observed reward value as $R(s)$,
    * to learn $\mathcal{S}(s'|s,a)$:, keep track of how often you get to state s' given that you're in state s and do action a,
    * as a result, the agent obtains MDP and "knows" the environment somehow,
  - step 2: (based on the underlying MDP) it performs policy evaluation (a part of policy iteration, the MDP solution method)
    * in principle, it solves a system of n Bellman equations (n is the number of states) $U^\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{S}(s'|s,\pi(s))U^\pi(s')$
    * the equations are linear, can be done in $\mathcal{O}(n^3)$.

ADP algorithm

```
function PASSIVE_ADP_AGENT(percept) returns an action
 inputs: percept, the current state s' and reward signal r'
 static: π, a fixed policy
  mdp, an MDP with model S, rewards R, discount γ
  U, a table of utilities, initially empty
  Nsa, a state-action frequency table, initially zero
  Nsas', a state-action-state frequency table, initially zero
  s, a the previous state and action, initially null
 if s' is new then do U[s']= r'; R[s']= r'
 if s is not null, then do
  increment Nsa[s,a] and Nsas'[s,a,s']
  for each t such that Nsas'[s,a,t] is nonzero do
   S[s,a,t] = Nsas'[s,a,t] / Nsa[s,a]
 U = POLICY_EVALUATION(π, U, mdp)
 if TERMINAL?[s'] then s,a=null else s,a = s',π[s']
 return a
```

Passive reinforcement learning

- Approach 3: temporal difference (TD) learning
  - avoids the computational expense of full policy evaluation,
  - approximate the exact state utility,
    $U^\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{S}(s'|s, \pi(s))U^\pi(s')$
  - in the equation above, instead of summing over all successors, only adjust the utility of the state based on the successor observed in the trial
    $U^\pi(s) \sim R(s) + \gamma U^\pi(s')$
  - the transition model not needed – model-free,
  - example
    - suppose you see that after the first trial:
      $U^\pi(1,3) = 0.84$ and $U^\pi(2,3) = 0.92$,
    - if (1,3) $\rightarrow$ (2,3) happens all the time, you would expect to see:
      $U^\pi(1,3) = R(1,3) + U^\pi(2,3) = -0.04 + 0.92 = 0.88$,
    - observed 0.84 in the first trial is little lower,
    - we might want to (slightly) move it towards 0.88.

Passive reinforcement learning

- Approach 3: temporal difference (TD) learning
  - update rule for transition from s to s'
    $U^\pi(s) = U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$
  - α is the learning rate,
  - $R(s) + \gamma U^\pi(s')$ gives a new noisy sample of utility based on next state,
  - analogy in online mean estimation

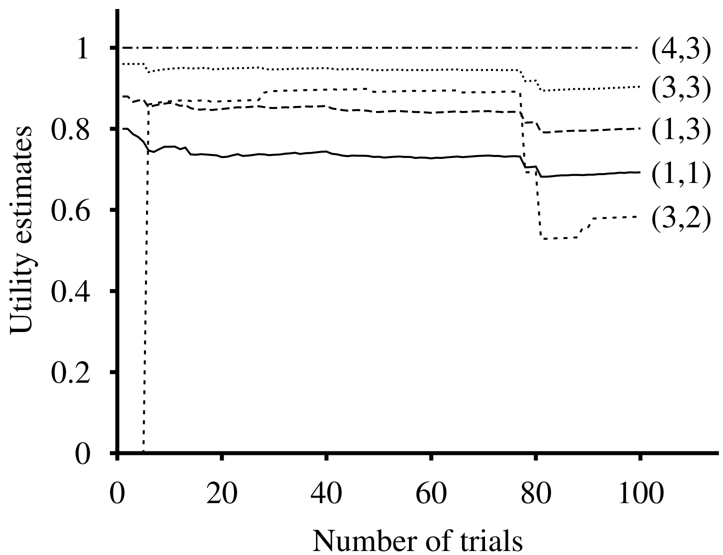
$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n+1}\Big(x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i\Big) =$$
$$= \hat{X}_n + \frac{1}{n+1}(x_{n+1} - \hat{X}_n)$$

  - the update rule maintains a mean of noisy utility samples,
  - if the learning rate decreases with the number of samples, it will eventually converge to true utility values.

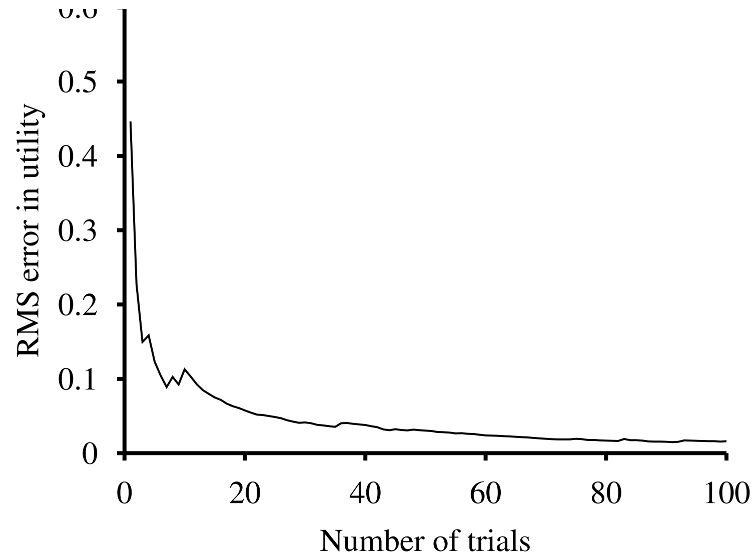
Passive RL – comparison

- Direct utility estimation
  - simple to implement, model-free,
  - each update is fast,
  - does not exploit state dependence and thus converges slowly,
- adaptive dynamic programming
  - harder to implement, model-based,
  - each update is a full policy evaluation (expensive),
  - fully exploits state dependence, fastest convergence in terms of epochs,
  - in active agents, exploration hard to be interleaved with model building,
- temporal difference learning
  - another model-free a., update speed and implementation similar to DUE,
  - partially exploits state dependence,
    does not adjust to **all** possible successors,
  - convergence in between DUE and ADP.

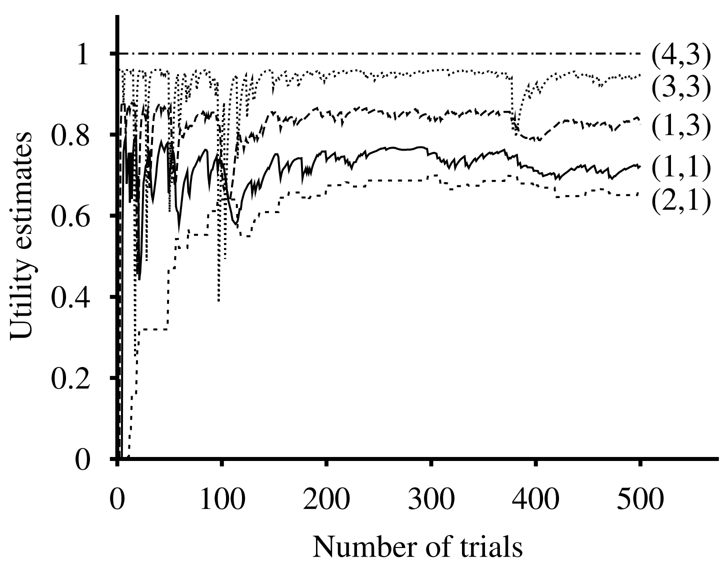
ADP vs TD



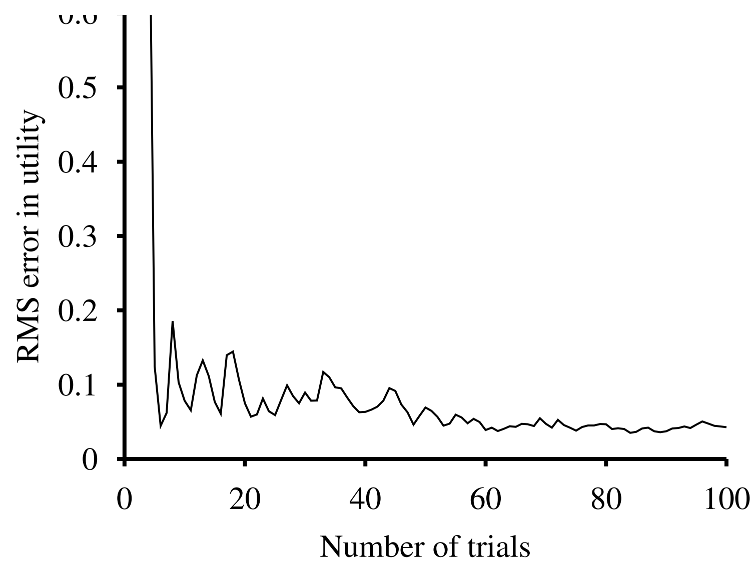
Utility estimates
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
Number of trials
(4,3)
(3,3)
(1,3)
(1,1)
(3,2)



RMS error in utility
0.5
0.4
0.3
0.2
0.1
0
0
20
40
60
80
100
Number of trials



Utility estimates
1
0.8
0.6
0.4
0.2
0
0
100
200
300
400
500
Number of trials
(4,3)
(3,3)
(1,3)
(1,1)
(2,1)



RMS error in utility
0.5
0.4
0.3
0.2
0.1
0
0
20
40
60
80
100
Number of trials

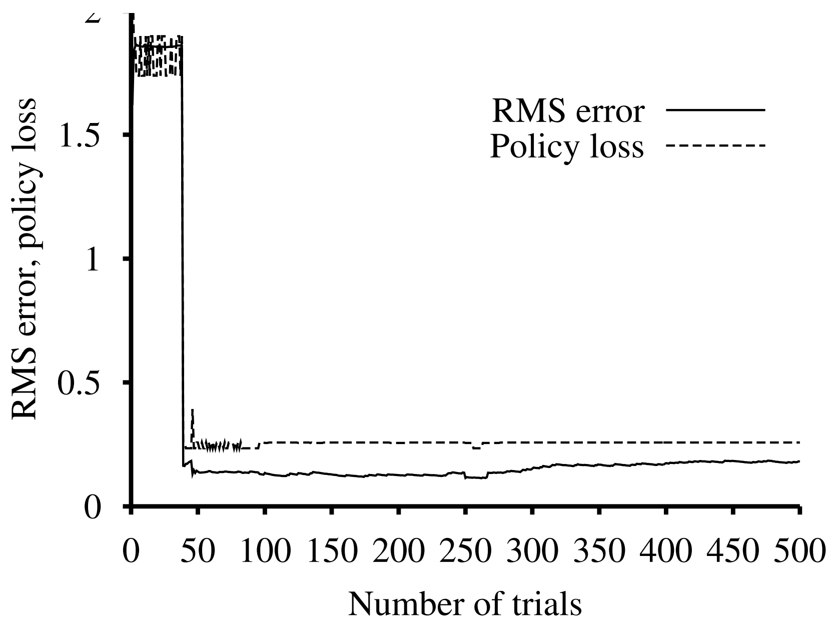
Active reinforcement learning

- the ultimate agent's goal is to find an optimal policy
  - another criteria could be learning time and performance during learning,
- let us start with the **active ADP agent**,
- the smallest change from the passive ADP could be
  - do not keep the policy fixed,
  - always take the action that maximizes the expected reward
    - given the existing environment model and utility estimates (utility estimates follow from the model, the model influenced by policy),
  - this action can easily be reached through the Bellman iterative equation

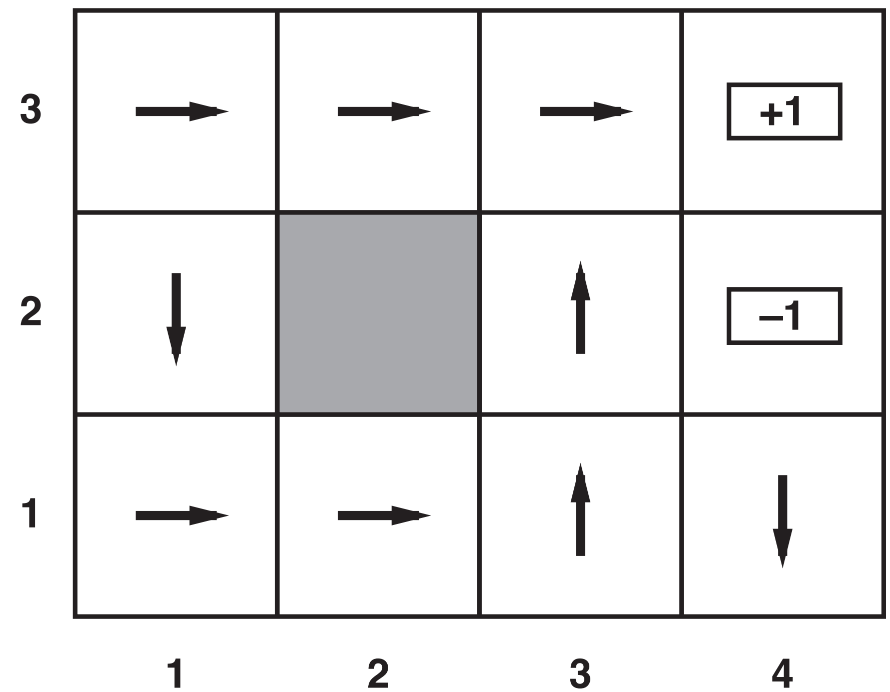
$$U(s) = R(s) + \gamma \max_a \sum_{s'} \mathcal{S}(s'|s,a)U(s')$$

  - we have a greedy active ADP agent,
- what happens?

Greedy active ADP agent in a grid world



The error development in a typical sequence of trials.



The suboptimal policy the agent converges in this particular sequence.

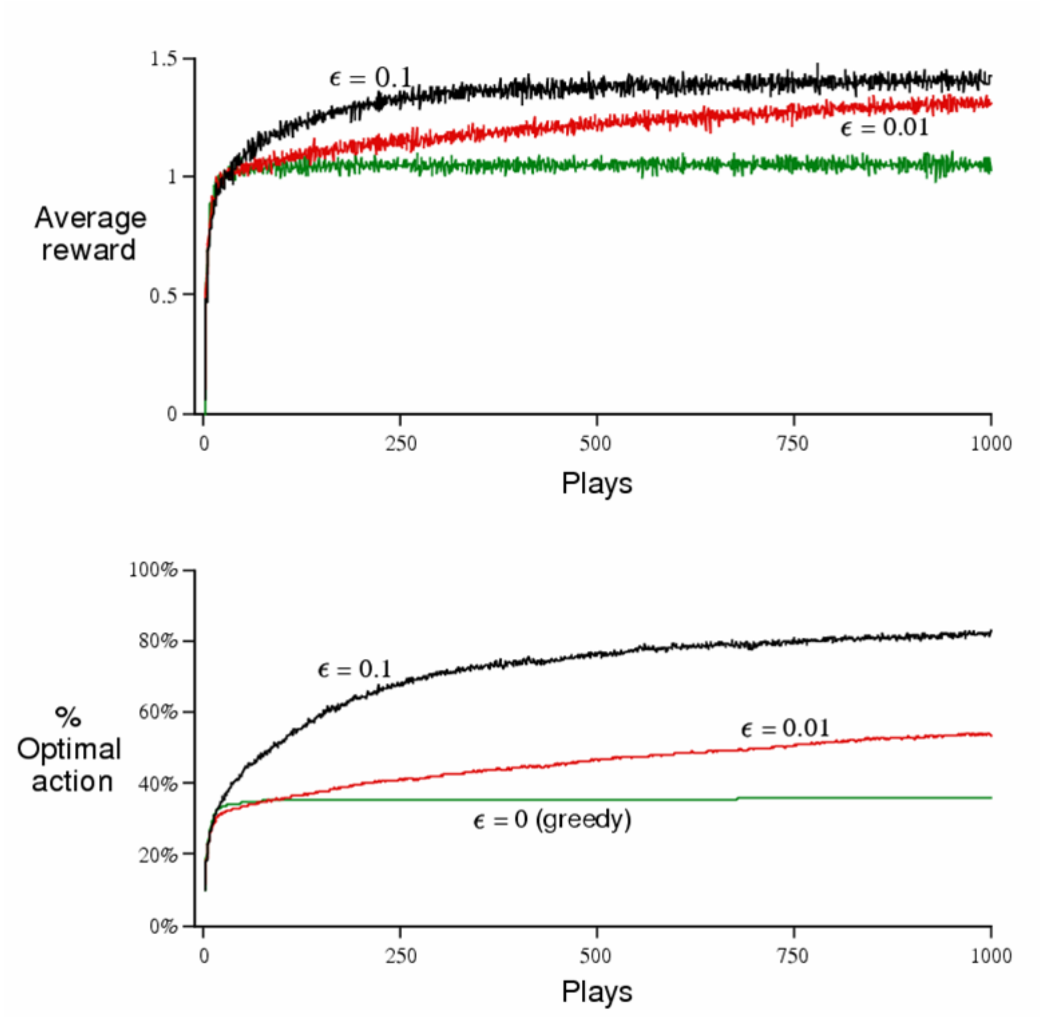
Greedy active ADP agent

- what happened?
  - choosing an optimal action lead to suboptimal results,
  - the learned model does not match the true environment,
  - the set of trials proposed and observed by the agent proves insufficient to build a good model,
- solution
  - the agent needs more training experience . . .
    . . . and more variability in the experience.

Exploration vs exploitation

- N-armed bandit
  - n actions with an unknown stochastic reward (and fixed parameters),
  - goal: maximize reward over M trials,
- Greedy strategy – pure exploitation
  - take the current best action, maximize the immediate reward,
  - gets stuck in a rut,
- Decide independetly of rewards – pure exploration
  - e.g., random or the least often applied, (quickly) learn the environment,
  - the environment is known, but never puts this knowledge into practice,
- "Nearly optimal" actions – a compromise
  - e.g, ε-greedy strategy,
  - only in ε situations random choice, greedy otherwise,
  - it also helps in non-stationary environment.

Exploration vs exploitation in 10-armed bandit



Sutton, Barto: Reinforcement Learning: An Introduction

Alternative action selection strategies

- ε-greedy strategy distinguishes two action categories only
  - the currently best and the other actions,
- SOFTMAX strategy
  - action a probability at time t is a function of its relative value,
  - important: the reward is relative, it must be related to other actions too

$$p(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

  - why so popular: a generalization of logistic function, differentiable, minimizes cross-entropy,
  - τ ... temperature – a high τ makes the actions equiprobable, for $\tau \rightarrow 0$ converges to the greedy strategy, τ should decrease with time,
- other strategies: interval estimates.

Optimistic utilities

- we converge even faster if actions not tried very often get higher weights,
- modified Bellman equations with optimistic utilities $U^+(s)$

$$U^+(s) = R(s) + \gamma \max_a f\Big(\sum_{s'} \mathcal{S}(s'|s,a)U^+(s), N(s,a)\Big)$$

  – $N(s,a)$ is the number of times a was taken in s,

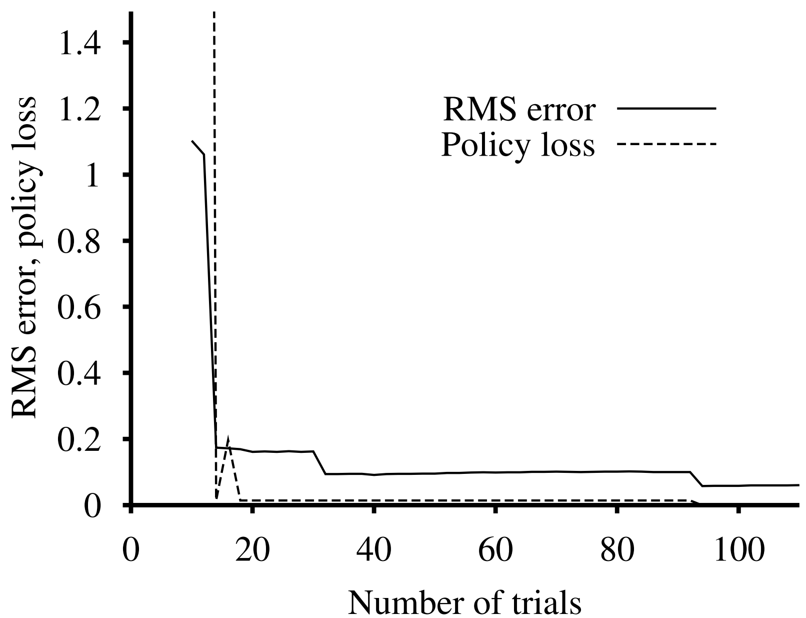
- The exploration function $f(u,n)$ determines how greed (preference for high values of a) is traded off against curiosity (preference for low values of n)
  – f should increase with expected utility u,
  – f should decrease with the number of trials n,
- Simple exploration function
  – $\forall n < N : f(u,v) = R^+$ (the best possible reward obtainable),
  – otherwise: $f(u,v) = u$.

Action selection strategies – summary

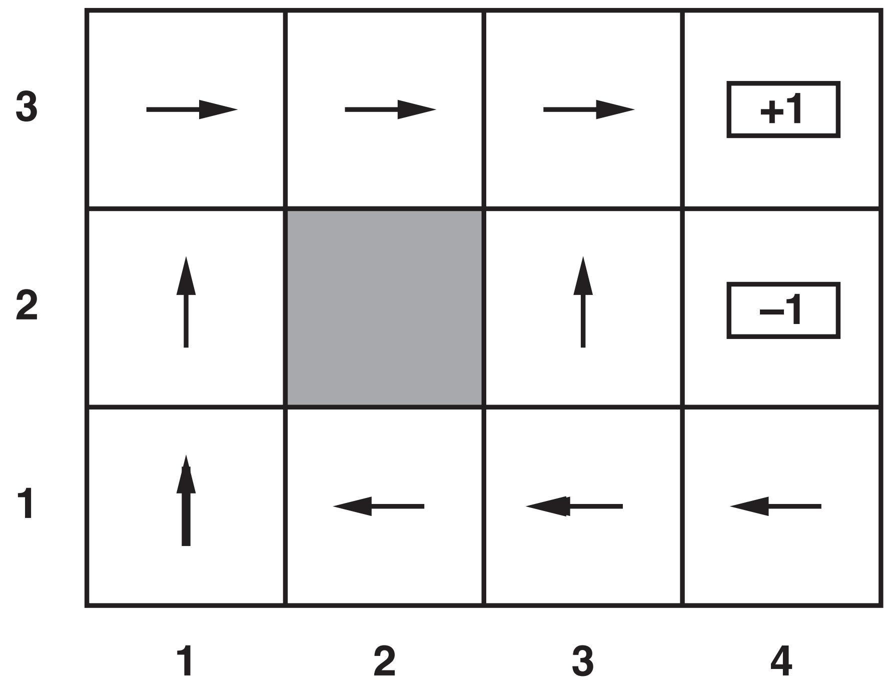
- An exact solution to exploration-exploitation problem usually intractable,
- fortunately, it is possible to come up with a reasonable exploration method that eventually leads to optimal behavior by the agent,
- any such exploration method needs to be Greedy in the Limit of Infinite Exploration (GLIE)
  - must try each action in each state an unbounded number of times so that it does not miss any optimal action,
  - must eventually become greedy,
  - example: ϵ-greedy, optimistic utility.

Exploratory active ADP agent in a grid world

- The only change in the exploratory active ADP agent
  - use a GLIE action selection scheme.



The rapid error development in a typical sequence of trials.



The policy the agent converges in this particular sequence.

Q-learning

- Now, we will aim at the active agent using TD learning
  - model-free, simpler to implement and often more computationally efficient,
- however, state utilities without model not sufficient for changing the policy
  - remember: $a_{next}^{greedy} = \arg\max_a \sum_{s'} \mathcal{S}(s'|s,a)U(s')$
- we will learn the optimal action-value function $Q(s,a)$ instead of $U(s)$
  - $Q(s,a)$ is the expected value of taking action a in state s and following the optimal policy thereafter,

$$U(s) = \max_a Q(s,a)$$

- the next action in s with Q-values

$$a_{next}^{greedy} = \arg\max_a Q(s,a)$$

Q-learning

- The utility update rule with the state utilities

$$U(s) = R(s) + \gamma \max_a \sum_{s'} \mathcal{S}(s'|s,a)U(s')$$

- after the substitution for U by Q

$$Q(s,a) = R(s) + \gamma \sum_{s'} \mathcal{S}(s'|s,a)U(s')$$

$$Q(s,a) = R(s) + \gamma \sum_{s'} \mathcal{S}(s'|s,a) \max_{a'} Q(s',a')$$

- at the moment, we still have the active ADP with the environment model
  - the model not needed for action selection,
  - however, employed in Q-value update.

Q-learning without a model

- The model removal step stemming from TD principle

$$U^\pi(s) = U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

$$Q^\pi(s,a) = Q^\pi(s,a) + \alpha(R(s) + \gamma \max_{a'} Q^\pi(s',a') - Q^\pi(s,a))$$

- the new estimate of $Q(s,a)$ stems from the old $Q(s,a)$ modified by the weighted difference between the old $Q(s,a)$ estimate and its new noisy sample after taking action a,
- SARSA is a sibling approach with a slightly modified update rule

$$Q^\pi(s,a) = Q^\pi(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q^\pi(s,a))$$

  - a' is the action actually taken in s',
  - e.g., in exploratory agents, a' is different from the greedy action considered in Q-learning.

TD Q-Learning algorithm

```
initialize Q(s,a) arbitrarily for all a and s
initialize s and r to the current state and reward observed
loop
  select action a according to explore/exploit policy
   the selection is based on current Q(s,a)
  receive immediate reward r' and observe the new state s'
  use the tuple (s,a,s',r') to update Q(s,a)
   Q(s,a)=Q(s,a)+α(r'+γ max_a'Q(s',a')-Q(s,a))
  s=s',r=r'
end loop
eventually return the greedy policy π
```

Notes on Q-learning

- Q-learning vs SARSA
  - Q-learning employs off-policy update that beteer learns for misleading policies,
  - SARSA's on-policy update is more realistic, works with utility that matches the policy.
- Q-learning convergence
  - guaranteed to converge to an optimal policy,
  - very general procedure because it is model free,
  - converges slower than ADP agent in terms of epochs
    it does not enforce consistency among values through the model.

References

:: Resources and reading

- Russell, Norvig: **AI: A Modern Approach**, 3rd edition, Prentice Hall, 2010.
  - Chapter 21 on Reinforcement learning,
- Sutton, Barto: **Reinforcement Learning: An Introduction**, 2nd edition, MIT Press, Cambridge, 2012.