TDDE56: Reinforcement Learning

Fredrik Heintz
Dept. of Computer Science, Linköping University
fredrik.heintz@liu.se
@FredrikHeintz

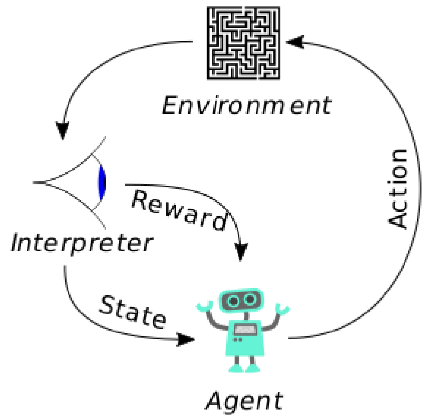
LINKÖPINGS UNIVERSITET

Reinforcement Learning Basic Concept

- *Reinforcement Learning is learning what to do – how to map situations to actions – so as to maximum a numerical reward.*

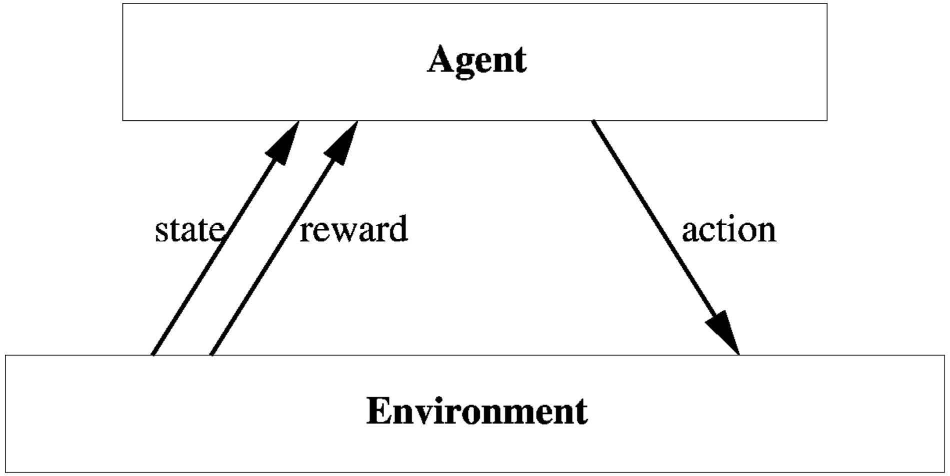
  Reinforcement Learning: An introduction
  Sutton & Barto

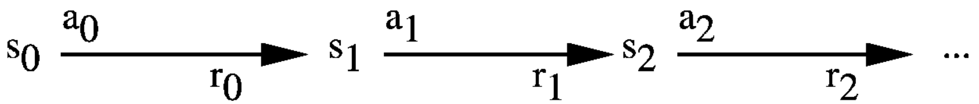
- Rather than learning from explicit training data, or discovering patterns in static data, reinforcement learning discovers the best option (highest reward) from trial and error.
- Inverse Reinforcement Learning
  - Learn reward function by observing an expert
  - "Apprenticeship learningapprenticeship learning"
  - E.g. Abbeel et al. *Autonomous Helicopter Aerobatics through Apprenticeship Learning*



A Reinforcement Learning Problem

- The environment
- The reinforcement function *r(s,a)*
  - Pure delay reward and avoidance problems
  - Minimum time to goal
  - Games
- The value function *V(s)*
  - Policy $\pi: S \rightarrow A$
  - Value $V^{\pi}(s) := \Sigma_i \gamma^i r_{t+i}$
- Find the optimal policy π^* that maximizes $V^{\pi*}(s)$ for all states *s*.





Goal: Learn to choose actions that maximize

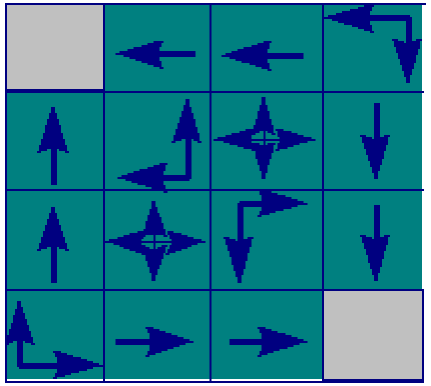
$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0<\gamma<1$

RL Value Function - Example

A minimum time to goal world

0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

Value function for random movement



Optimal policy

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

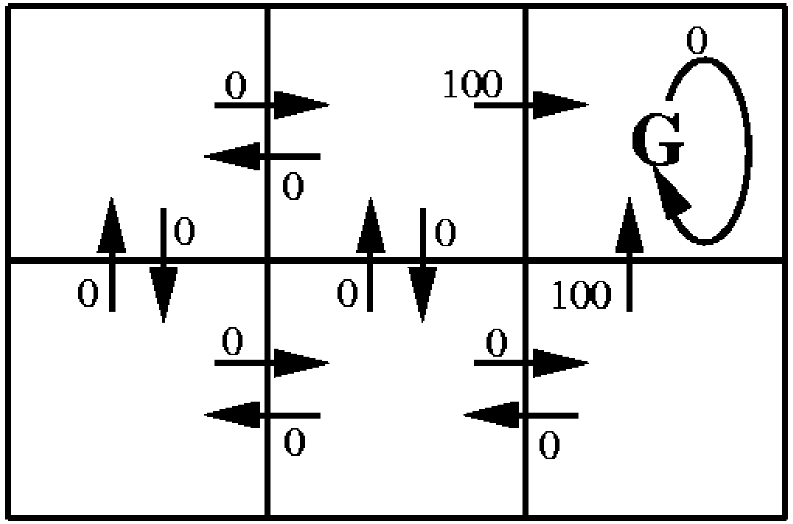
Optimal value function

Markov Decision Processes

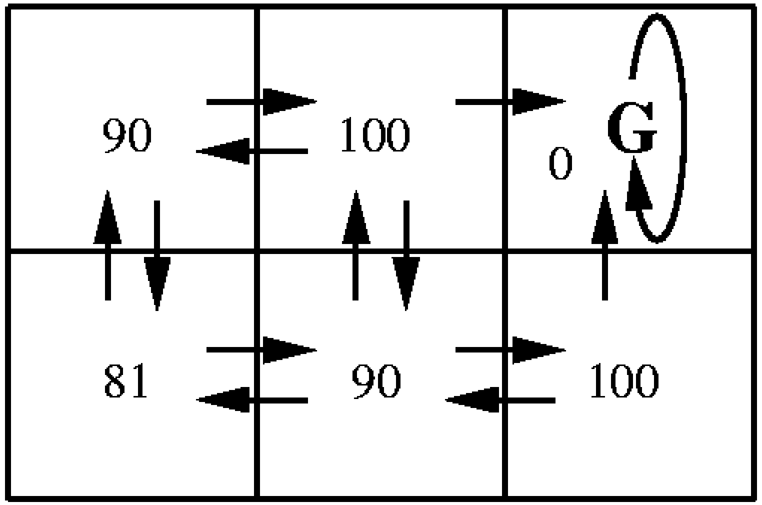
Assume:

- finite set of states S, finite set of actions A
- at each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$
- then receives immediate reward r_t
- and state changes to s_{t+1}
- Markov assumption: $s_{t+1} = \delta(s_t, a_t)$ and $r_t = \mathrm{r}(s_t, a_t)$
  - i.e. r_t and s_{t+1} depend only on current state and action
  - functions δ and r may be non-deterministic
  - functions δ and r not necessarily known to the agent

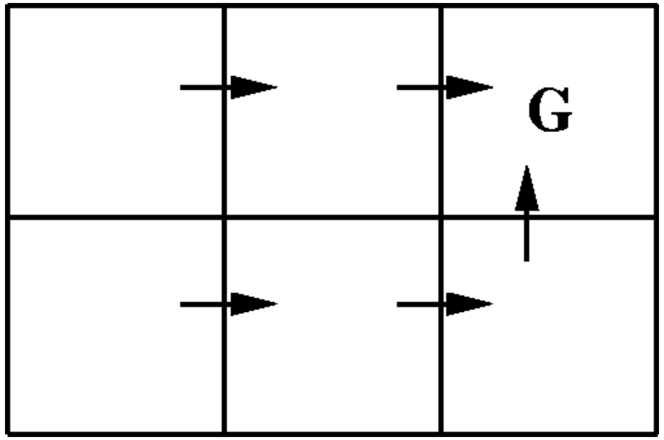
MDP Example



r(s,a)



V*(s)



An optimal policy

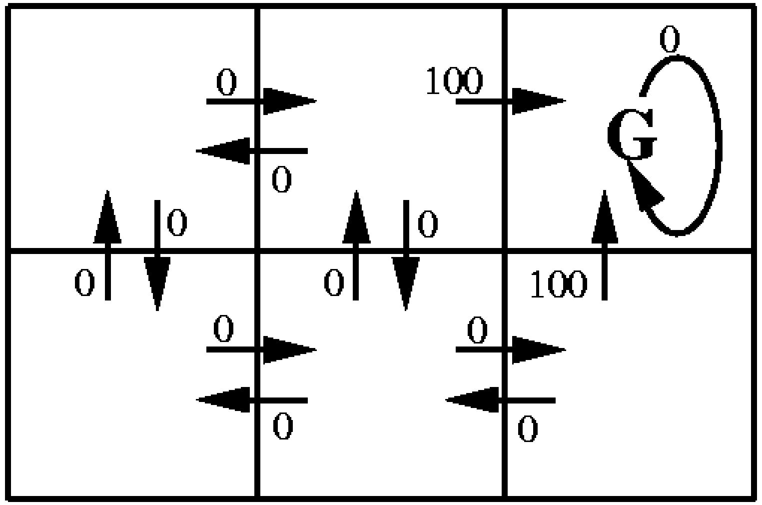
The Q-Function

Optimal policy:

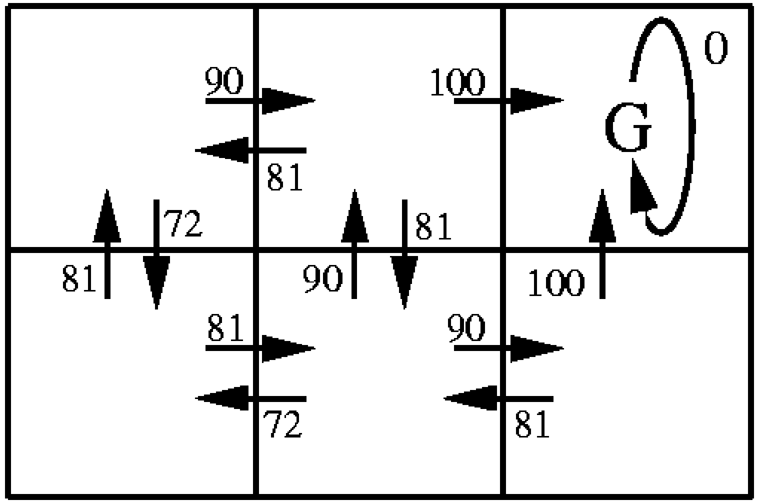
- $\pi^*(s) = \text{argmax}_a[r(s,a) + \gamma V^*(\delta(s,a))]$
- Doesn't work if we don't know r and δ.

The Q-function:

- $Q(s,a) := r(s,a) + \gamma V^*(\delta(s,a))$
- $\pi^*(s) = \text{argmax}_a Q(s,a)$



r(s,a)



Q(s,a)

The Q-Function

- Note Q and V* closely related:
  $V^*(s) = \max_{a'} Q(s,a')$
- Therefore Q can be written as:

  $Q(s_t, a_t) := r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) =$

  $r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$
- If $Q^\wedge$ denote the current approximation of Q then it can be updated by:

  $Q^\wedge(s,a) := r + \gamma \max_{a'} Q^\wedge(s',a')$

Q-Learning for Deterministic Worlds

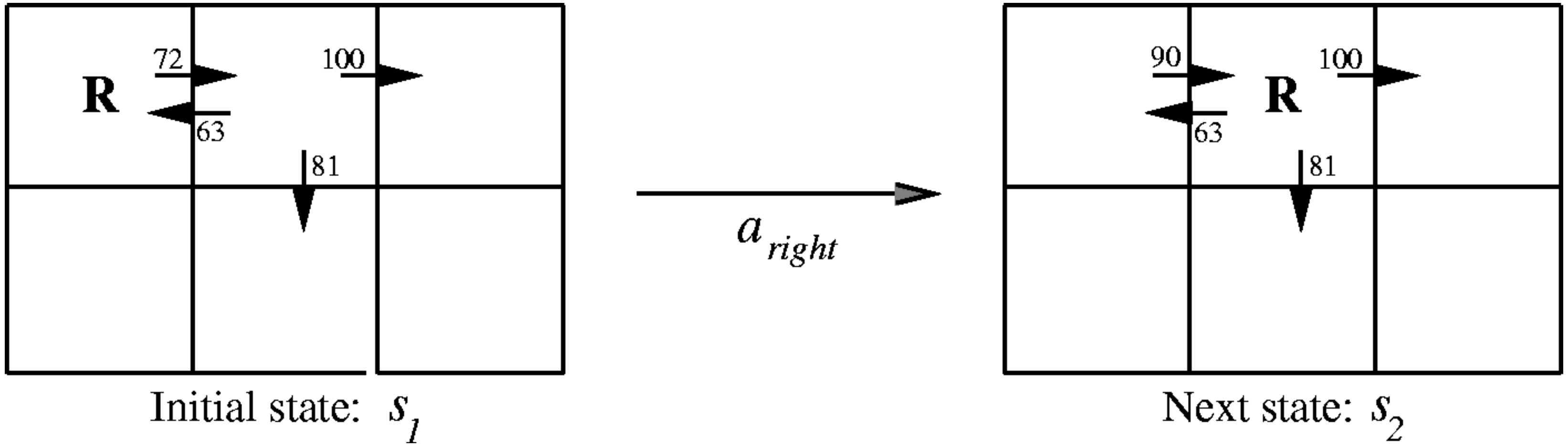
For each s, a initialize table entry $\hat{Q}(s,a) := 0$.

Observe current state s.

Do forever:

1. Select an action a and execute it
2. Receive immediate reward r
3. Observe the new state s'
4. Update the table entry for $\hat{Q}(s,a)$:
   $\hat{Q}(s,a) := r + \gamma \max_{a'} \hat{Q}(s',a')$
5. $s := s'$

Q-Learning Example



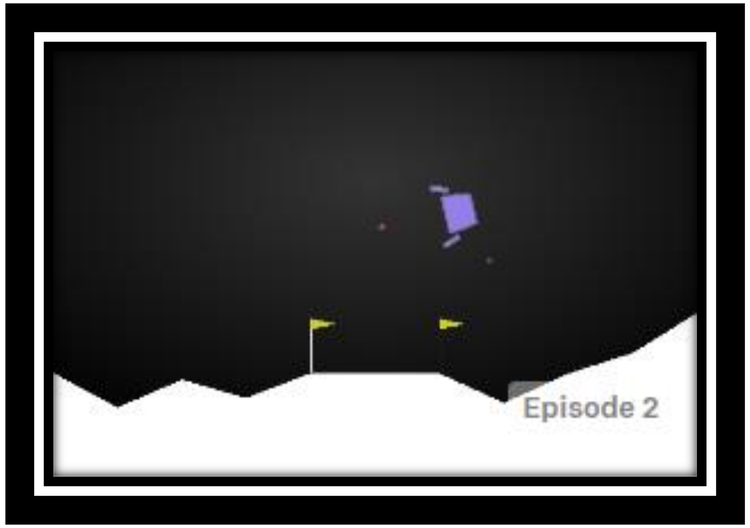
$$\hat{Q}(s_1, a_{right}) := r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
$$:= 0 + 0.9 \max\{63, 81, 100\}$$
$$:= 90$$

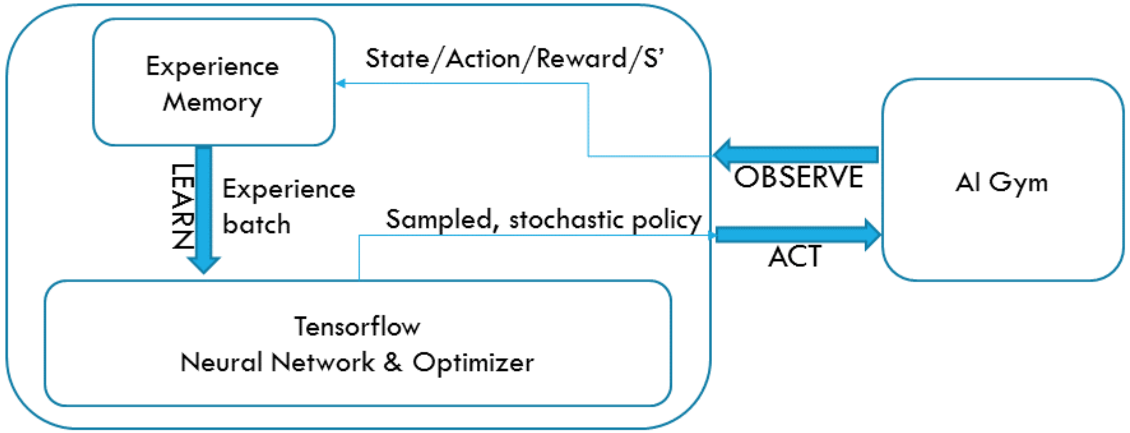
Q-Learning Continued

- Exploration
  - Selecting the best action
  - Probabilistic choice
- Improving convergence
  - Update sequences
  - Remember old state-action transitions and their immediate reward
- Non-deterministic MDPs
- Temporal Difference Learning

Reinforcement Learning – Neural Networks as Function Approximators

- To tackle a high-dimensional state space or continous states we can use a neural network as function approximator
- Lunar Lander experiment
  - 8 continous/discrete states
    - XY-Pos, XY-Vel, Rot, Rot-rate, Leg1/Leg2 ground contact
  - 4 discrete actions
    - Left thrust
    - Right thrust
    - Main engine thrust
    - NOP
  - Rewards
    - Move from top to bottom of the screen (+ ~100-140)
    - Land between the posts (+100)
    - Put legs on ground (+10 per leg)
  - Penalties
    - Using main engine thrust (-0.3 per frame)
    - Crashing (-100)
- Solved using Stochastic Policy Gradients





Reinforcement Learning Neural Networks as Function Approximators