

Reinforcement Learning

Belöningsbaserad Inläring

- 1 Defining the Problem
 - Framework
 - Role of Reward
 - Simplifying Assumptions
 - Central Concepts
- 2 Known Environment
 - Bellmans Equation
 - Solving Techniques
- 3 Unknown Environment
 - Monte-Carlo Method
 - Temporal-Difference
 - Q-Learning
 - Sarsa-Learning
- 4 Improvements
 - Importance of Making Mistakes
 - Eligibility Trace

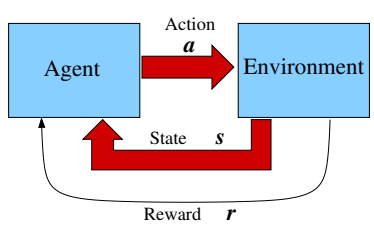
- 1 Defining the Problem
 - Framework
 - Role of Reward
 - Simplifying Assumptions
 - Central Concepts
- 2 Known Environment
 - Bellmans Equation
 - Solving Techniques
- 3 Unknown Environment
 - Monte-Carlo Method
 - Temporal-Difference
 - Q-Learning
 - Sarsa-Learning
- 4 Improvements
 - Importance of Making Mistakes
 - Eligibility Trace

Reinforcement Learning
 Learning of a behavior without explicit information about correct actions

- A **reward** (belöning) gives information about success
- The reward does not necessarily arrive *when* you do something good
Temporal credit assignment
- The reward does not say *what* was good
Structural credit assignment

Model of the learning situation

- An **agent** interacts with its **environment** (omgivning)
- The agent makes **actions** (handlingar)
- Actions affects the environments **state** (tillstånd)
- The agent can observe the environments state
- The agent receives **reward** (belöning) from the environment



Task for the Agent
 Find a behavior which maximizes the expected *total* reward.

How long future should we consider?

- Finite Horizon

$$\max \left[\sum_{t=0}^h r_t \right]$$

- Infinite Horizon

$$\max \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

Requires discount (**nedskrivning**) of future reward
 ($0 < \gamma < 1$)

Role of Reward
Reward Function

The Reward function controls which task should be solved

- Game (Chess, Backgammon)
 Reward only at the end: +1 when winning, -1 when loosing
- Avoiding mistakes (cycling, balancing, ...)
 Reward -1 at the end (when failing)
- Find a short/fast/cheap path to a goal
 Reward -1 at each step

Central Concepts
The Agents Internal Representation

- **Policy**
 The action chosen by the agent for each state

$$\pi(s) \mapsto a$$

- **Value Function**
 Expected total future reward from s when following policy π

$$V^\pi(s) \mapsto \mathfrak{R}$$

Simplifying Assumptions

Simplifying Assumptions

- Discrete time
- Finite number of actions a_i

$$a_i \in a_1, a_2, a_3, \dots, a_n$$

- Finite number of states s_i

$$s_i \in s_1, s_2, s_3, \dots, s_m$$

- Environment is a stationary *Markov Decision Process*
 Reward and next state depends only on s , a and chance
- Deterministic or non-deterministic environment

Central Concepts

The values of a state depends on the current policy.

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

V with an optimal policy

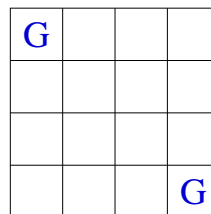
0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

V with a random policy

Central Concepts

Classical model problem: *Grid World*

- Each **state** is represented by a position in a grid
- The agent **acts** by moving to other positions



Trivial labyrinth

Reward: -1 at each step until a goal state (G) is reached

- 1 Defining the Problem
 - Framework
 - Role of Reward
 - Simplifying Assumptions
 - Central Concepts
- 2 **Known Environment**
 - Bellmans Equation
 - Solving Techniques
- 3 **Unknown Environment**
 - Monte-Carlo Method
 - Temporal-Difference
 - Q-Learning
 - Sarsa-Learning
- 4 **Improvements**
 - Importance of Making Mistakes
 - Eligibility Trace

Bellmans Equation

Model of the Environment

- Where does an action take us?

$$\delta(s, a) \mapsto s'$$
- How much reward do we receive?

$$r(s, a) \mapsto \mathfrak{R}$$

The values of different states are interrelated
Bellmans Equation:

$$V^\pi(s) = r(s, \pi(s)) + \gamma \cdot V^\pi(\delta(s, \pi(s)))$$

Solving Techniques

Can we solve *Bellmans equation*?

$$V^\pi(s) = r(s, \pi(s)) + \gamma \cdot V^\pi(\delta(s, \pi(s)))$$

- Direct solution (linear equation system)
- Iteratively (*value iteration*)

$$V_{k+1}^\pi(s) \leftarrow r(s, \pi(s)) + \gamma \cdot V_k^\pi(\delta(s, \pi(s)))$$

Solving Techniques

How do we find an **optimal policy** π^* ?
 Easy if the optimal value function V^* was known:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} (r(s, a) + \gamma \cdot V^*(\delta(s, a)))$$

Optimal version of Bellmans equation

$$V^*(s) = \max_a (r(s, a) + \gamma \cdot V^*(\delta(s, a)))$$

Hard to solve

Policy iteration:

Interleaved calculation of policy and values

- 1 Defining the Problem
 - Framework
 - Role of Reward
 - Simplifying Assumptions
 - Central Concepts
- 2 Known Environment
 - Bellmans Equation
 - Solving Techniques
- 3 Unknown Environment
 - Monte-Carlo Method
 - Temporal-Difference
 - Q-Learning
 - Sarsa-Learning
- 4 Improvements
 - Importance of Making Mistakes
 - Eligibility Trace

Monte-Carlo Method

Normal scenario: $r(s, a)$ and $\delta(s, a)$ are not known
 V^π must be estimated by **experience**

Monte-Carlo Method

- Start at a random s
- Follow π , store the rewards and s_t
- When the goal is reached, update $V^\pi(s)$ -estimation for all visited states with the future reward we actually received

Very slow convergence

Temporal-Difference

Temporal Difference

Idée behind Temporal Difference:

Use that there are two estimates of the value of a state:
before and after

- Estimate **before** the action

$$V^\pi(s_t)$$

- Estimate **after** the action

$$r_{t+1} + \gamma \cdot V^\pi(s_{t+1})$$

Important observation:
 The second estimate is better!

Update the value estimate towards the better

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \eta [r_{t+1} + \gamma \cdot V^\pi(s_{t+1}) - V^\pi(s_t)]$$

Measure of the **surprise / disappointment**

Learns *considerably faster* than the Monte-Carlo method

Problem:

Even if we have a good estimate of V we are not able to compute π since the agent does not have δ and r !

Trick:

Estimate $Q(s, a)$ instead of $V(s)$

$Q(s, a)$: Expected total reward when doing a from s .

$$\pi(s) = \operatorname{argmax}_a Q(s, a)$$

$$V^*(s) = \max_a Q^*(s, a)$$

How can we learn Q ?

The Q -function can also be learned using Temporal-Difference

$$Q(s, a) \leftarrow Q(s, a) + \eta \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

s' is the next state.

Small problem: max-operation requires a search through all possible actions

SARSA-learning

Almost the same as Q-learning, but one uses the **current policy** to select a' :

$$Q(s, a) \leftarrow Q(s, a) + \eta [r + \gamma Q(s', a') - Q(s, a)]$$

The name comes from the experience-tuples structure:

$$\langle s, a, r, s', a' \rangle$$

- 1 Defining the Problem
 - Framework
 - Role of Reward
 - Simplifying Assumptions
 - Central Concepts
- 2 Known Environment
 - Bellmans Equation
 - Solving Techniques
- 3 Unknown Environment
 - Monte-Carlo Method
 - Temporal-Difference
 - Q-Learning
 - Sarsa-Learning
- 4 Improvements
 - Importance of Making Mistakes
 - Eligibility Trace

What do we do when...

- The environment is not fully observable
- There are way too many states
- The states are not discrete
- The agent is acting in continuous time

The Exploration–Exploitation dilemma

If an agent strictly follows a greedy policy based on the current estimate of Q , learning is not guaranteed to converge to Q^*

Simple solution:

Use a policy which has a certain probability of "making mistakes"

- **ϵ -greedy**
Sometimes (with probability ϵ) make a random action instead of the one that seems best (greedy)
- **Softmax**
Assign a probability to choose each action depending on how good they seem

Accelerated learning

Idéa: TD updates can be used to improve not only the last state's value, but also states we have visited earlier.

$$\forall s, a : Q(s, a) \leftarrow Q(s, a) + \eta [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \cdot e$$

e is a remaining trace (**eligibility trace**) encoding how long ago we were in s doing a .

Often denoted **TD(λ)** where λ is the time constant of the trace e