Reinforcement Learning

Belöningsbaserad Inlärning

Reinforcement Learning

Learning of a behavior without
explicit information about correct actions

Reinforcement Learning

Learning of a behavior without
explicit information about correct actions

- A reward (belöning) gives information about success

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

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

Model of the learning situation

- An agent interacts with its environment (omgivning)

Model of the learning situation

- An agent interacts with its environment (omgivning)
- The agent makes actions (handlingar)

Model of the learning situation

- An agent interacts with its environment (omgivning)
- The agent makes actions (handlingar)
- Actions affects the environments state (tillstånd)

Model of the learning situation

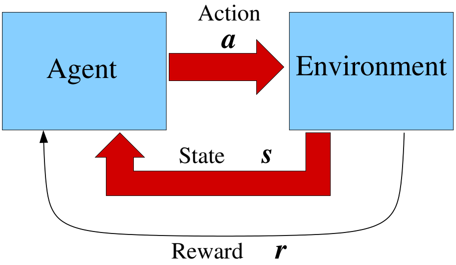
- An agent interacts with its environment (omgivning)
- The agent makes actions (handlingar)
- Actions affects the environments state (tillstånd)
- The agent can observe the environments state

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

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* revard.

Task for the Agent

Find a behavior which maximizes the expected *total* revard.

How long future should we consider?

Task for the Agent

Find a behavior which maximizes the expected *total* revard.

How long future should we consider?

- Finite Horizon

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

Task for the Agent

Find a behavior which maximizes the expected *total* revard.

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 $(0 < \gamma < 1)$

Reward Function

The Reward function controls which task should be solved

Reward Function

The Reward function controls which task should be solved

- Game (Chess, Backgammon)

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

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 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)

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 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

Simplifying Assumptions

Simplifying Assumptions

- Discrete time

Simplifying Assumptions

- Discrete time
- Finite number of actions a_i

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

Simplifying Assumptions

- Discrete time
- Finite number of actions a_i

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

- Finite number of states s_i

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

Simplifying Assumptions

- Discrete time
- Finite number of actions a_i

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

- Finite number of states s_i

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

- Environment is a stationary *Markov Decision Process*

Simplifying Assumptions

- Discrete time
- Finite number of actions a_i

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

- Finite number of states s_i

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

- Environment is a stationary *Markov Decision Process*
  Reward and next state depends only on s, a and chance

Simplifying Assumptions

- Discrete time
- Finite number of actions a_i

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

- Finite number of states s_i

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

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

The Agents Internal Representation

The Agents Internal Representation

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

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

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
  $$V^{\pi}(s) \mapsto \Re$$

Classical model problem: *Grid World*

Classical model problem: *Grid World*

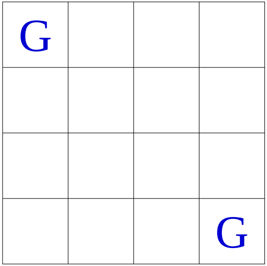
- Each state is represented by a position in a grid

Classical model problem: *Grid World*

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

Classical model problem: *Grid World*

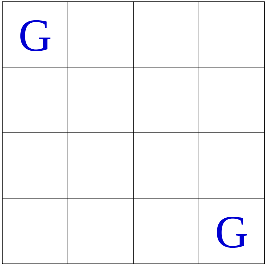
- Each state is represented by a position in a grid
- The agent acts by moving to other positions



Trivial labyrinth

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

The values of a state depends on the current policy.

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

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

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

Model of the Environment

Model of the Environment

- Where does an action take us?

$$\delta(s, a) \mapsto s'$$

Model of the Environment

- Where does an action take us?

$$\delta(s, a) \mapsto s'$$

- How much reward do we receive?

$$r(s, a) \mapsto \Re$$

Model of the Environment

- Where does an action take us?

$$\delta(s, a) \mapsto s'$$

- How much reward do we receive?

$$r(s, a) \mapsto \Re$$

The values of different states are interrelated

Model of the Environment

- Where does an action take us?

$$\delta(s, a) \mapsto s'$$

- How much reward do we receive?

$$r(s, a) \mapsto \Re$$

The values of different states are interrelated
Bellmans Equation:

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

Can we solve *Bellmans equation*?

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

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)

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*)

$$V^\pi_{k+1}(s) \leftarrow r(s, \pi(s)) + \gamma \cdot V^\pi_k(\delta(s, \pi(s)))$$

How do we find an optimal policy $\pi^{\star}$?

How do we find an optimal policy $\pi^\star$?
Easy if the optimal value function $V^\star$ was known:

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

How do we find an optimal policy $\pi^\star$?
Easy if the optimal value function $V^\star$ was known:

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

Optimal version of Bellmans equation

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

How do we find an optimal policy $\pi^\star$?
Easy if the optimal value function $V^\star$ was known:

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

Optimal version of Bellmans equation

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

Hard to solve

How do we find an optimal policy $\pi^\star$?
Easy if the optimal value function $V^\star$ was known:

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

Optimal version of Bellmans equation

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

Hard to solve

Policy iteration:

Interleaved calculation of policy and values

Normal scenario: $r(s, a)$ and $\delta(s, a)$ are not known

Normal scenario: $r(s, a)$ and $\delta(s, a)$ are not known

V^π must be estimated by experience

Normal scenario: $r(s, a)$ and $\delta(s, a)$ are not known

V^π must be estimated by experience

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

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

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 stated with the future reward we actually received

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 stated with the future reward we actually received

Very slow convergence

Temporal Difference

Temporal Difference

Idéa behind Temporal Difference:

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

Temporal Difference

Idéa behind Temporal Difference:

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

- Estimate before the action

$$V^{\pi}(s_t)$$

Temporal Difference

Idéa 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!

Important observation:

The second estimate is better!

Update the value estimate towards the better

Important observation:

The second estimate is better!

Update the value estimate towards the better

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

Important observation:

The second estimate is better!

Update the value estimate towards the better

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

Measure of the surprise / disappointment

Important observation:

The second estimate is better!

Update the value estimate towards the better

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

Measure of the surprise / disappointment

Learns *considerably faster* that 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!

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 $V(s)$

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.

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.

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

$$V^{\star}(s) = \max_{a} Q^{\star}(s, a)$$

How can we learn Q?

How can we learn Q?

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

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.

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

SARSA-learning

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

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

SARSA-learning

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

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

The name comes from the experience-tuples structure:

$$< s, a, r, s', a' >$$

What do we do when...

What do we do when...

- The environment is not fully observable

What do we do when...

- The environment is not fully observable
- There are way too many states

What do we do when...

- The environment is not fully observable
- There are way too many states
- The states are not discrete

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 $Q^\star$

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 $Q^\star$

Simple solution:
Use a policy which has a certain probability of "making mistakes"

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 $Q^\star$

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)

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 $Q^\star$

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

Accelerated learning

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

Accelerated learning

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

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

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

Accelerated learning

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

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

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