Reinforcement learning

Applied artificial intelligence (EDA132)
Lecture 13
2012-04-26
Elin A. Topp

Material based on course book, chapter 21 (17),
and on lecture "Belöningsbaserad inlärning / Reinforcement learning"
by Örjan Ekeberg, CSC/Nada, KTH, autumn term 2006 (in Swedish)

Outline

- Reinforcement learning (chapter 21, with some references to 17)
  - Problem definition
    - Learning situation
    - Roll of the reward
    - Simplified assumptions
    - Central concepts and terms
  - Known environment
    - Bellman's equation
    - Approaches to solutions
  - Unknown environment
    - Monte-Carlo method
    - Temporal-Difference learning
    - Q-Learning
    - Sarsa-Learning
  - Improvements
    - The usefulness of making mistakes
    - Eligibility Trace

Outline

- Reinforcement learning (chapter 21, with some references to 17)
  - Problem definition
    - Learning situation
    - Roll of the reward
    - Simplified assumptions
    - Central concepts and terms
  - Known environment
    - Bellman's equation
    - Approaches to solutions
  - Unknown environment
    - Monte-Carlo method
    - Temporal-Difference learning
    - Q-Learning
    - Sarsa-Learning
  - Improvements
    - The usefulness of making mistakes
    - Eligibility Trace

Reinforcement learning

Learning of a behaviour (a strategy, a skill) without access to a right / wrong measure for actions and decisions taken.

Reinforcement learning

Learning of a behaviour (a strategy, a skill) without access to a right / wrong measure for actions and decisions taken.

With the help of a *reward*, a measure is given, of *how well things are going*

Reinforcement learning

Learning of a behaviour (a strategy, a skill) without access to a right / wrong measure for actions and decisions taken.

With the help of a *reward*, a measure is given, of *how well things are going*

Note: The reward is not given in direct connection with a good choice of action *(temporal credit assignment)*

Reinforcement learning

Learning of a behaviour (a strategy, a skill) without access to a right / wrong measure for actions and decisions taken.

With the help of a *reward*, a measure is given, of *how well things are going*

Note: The reward is not given in direct connection with a good choice of action *(temporal credit assignment)*

Note: The reward does not tell what exactly it was, that made the "good" action *(structural credit assignment)*

Real life examples

Real life examples

Riding a bicycle

Powder skiing

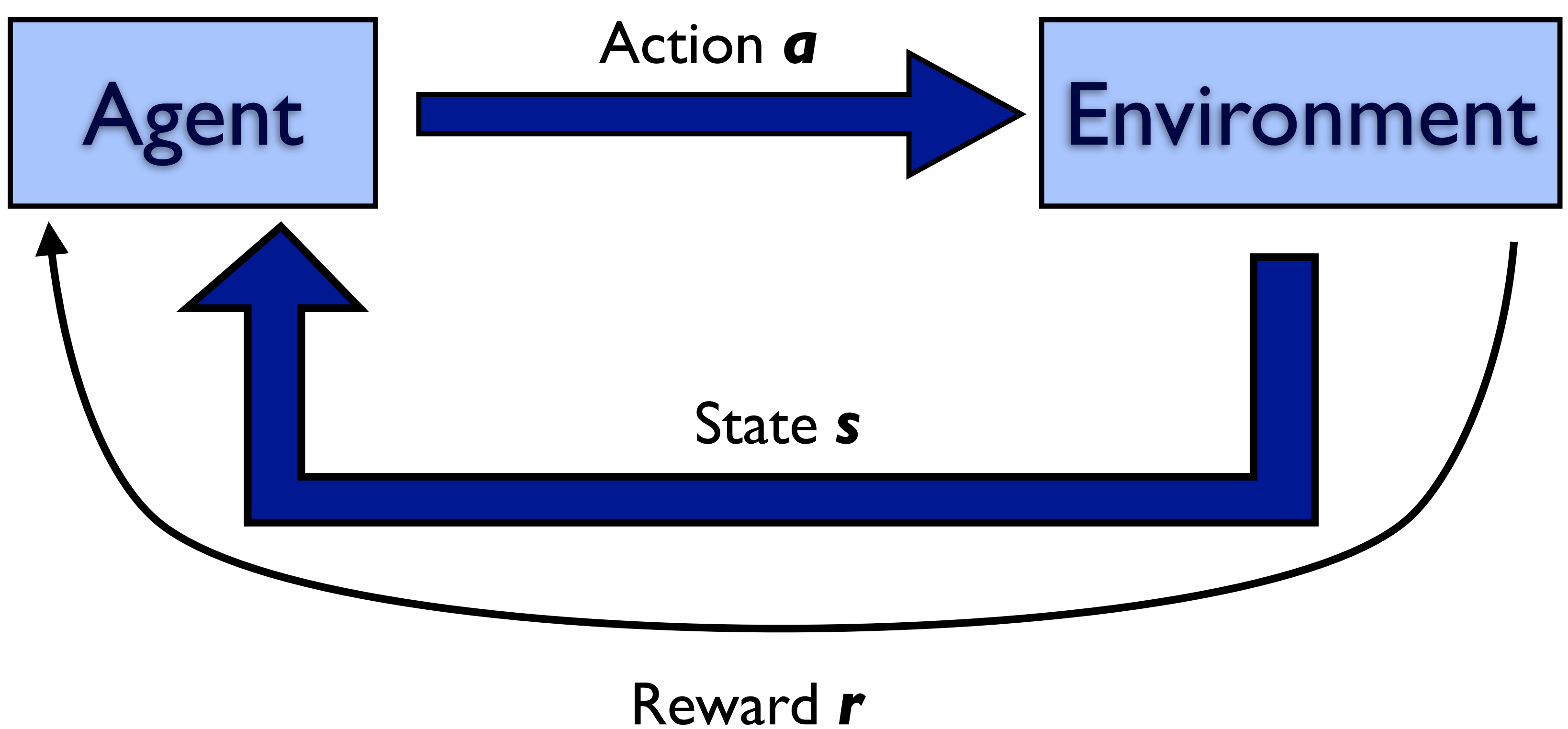
Learning situation: A model

An *agent* interacts with its *environment*

The agent performs *actions*

Actions have *influence* on the environment's *state*

The agent *observes* the environment's *state* and *receives* a *reward* from the environment



Learning situation: The agent's task

The task:

Find a behaviour (action sequence) that maximises the overall reward

How long into the future should we spy?

Finite time horizon:

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

Infinite time horizon:

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

with γ being a discount factor for future rewards ($0 < \gamma < 1$)

The reward function's roll

The reward function depends on the type of task

The reward function's roll

The reward function depends on the type of task

- Game (Chess, Backgammon): Reward is given only in the end of the game, +1 for "win", -1 for "loose"

The reward function's roll

The reward function depends on the type of task

- Game (Chess, Backgammon): Reward is given only in the end of the game, +1 for "win", -1 for "loose"
- Avoid mistakes (Riding a bike, Learning to fly according to hitchhiker's guide): Reward -1 when failing (falling)

The reward function's roll

The reward function depends on the type of task

- Game (Chess, Backgammon): Reward is given only in the end of the game, +1 for "win", -1 for "loose"
- Avoid mistakes (Riding a bike, Learning to fly according to hitchhiker's guide): Reward -1 when failing (falling)
- Find the shortest / cheapest / fastest path to a goal: Reward -1 for each step

A classic example: Grid World

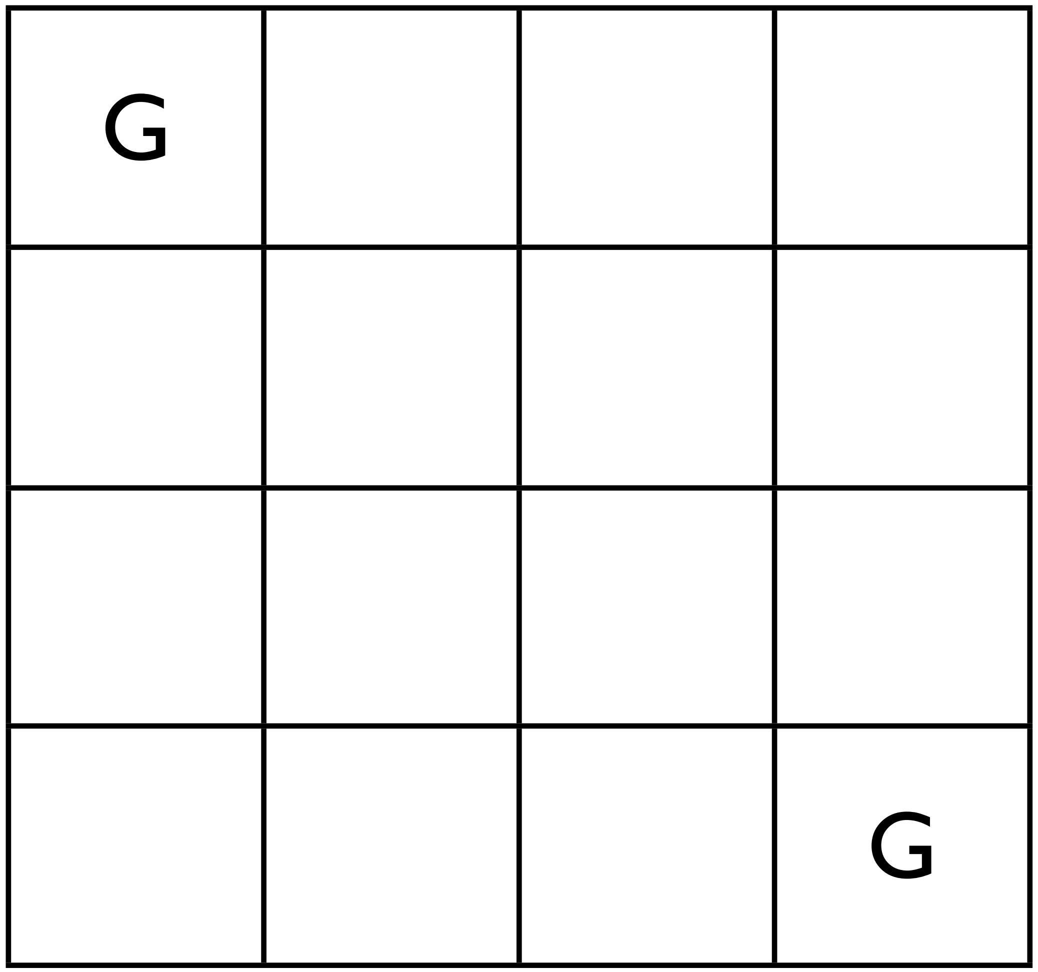
Simplified "Wumpus world" with just two gold pieces

G			
			G

A classic example: Grid World

Simplified "Wumpus world" with just two gold pieces

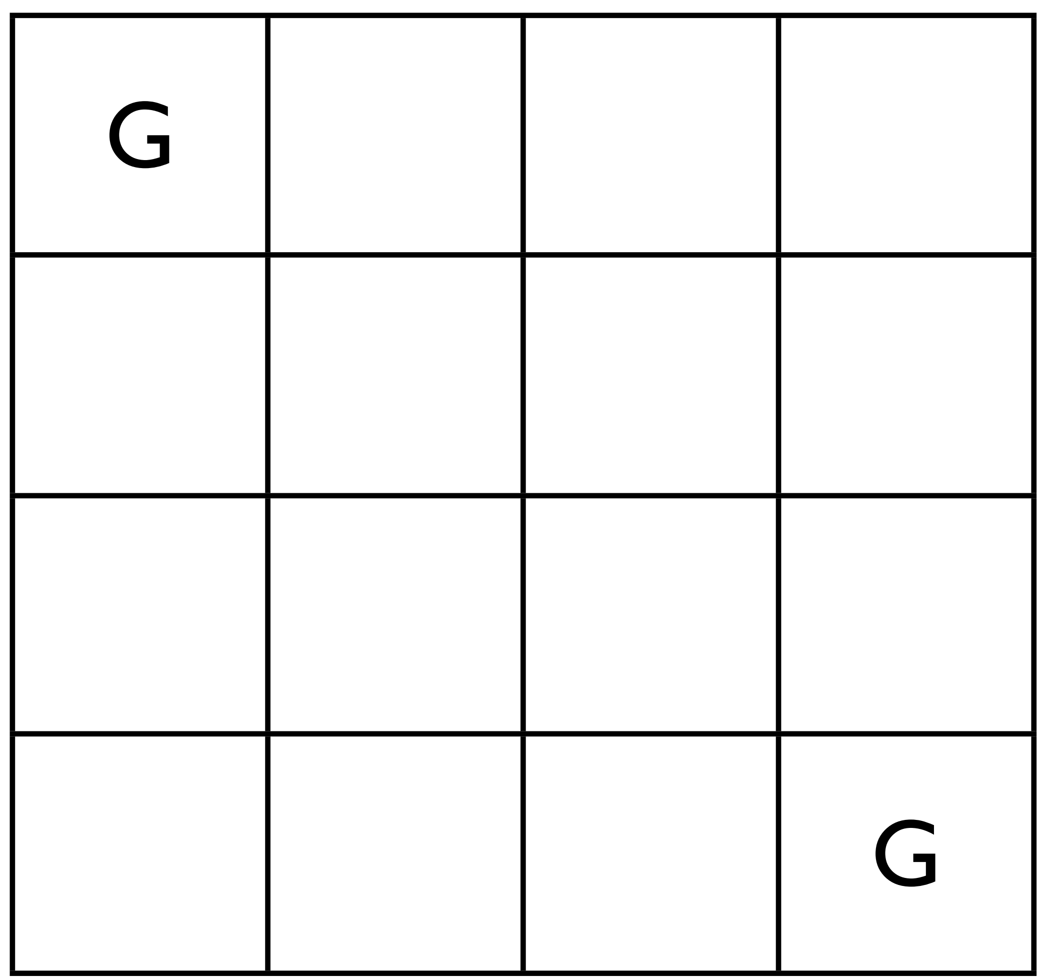
- Every state s_j is represented by a field in the grid



A classic example: Grid World

Simplified "Wumpus world" with just two gold pieces

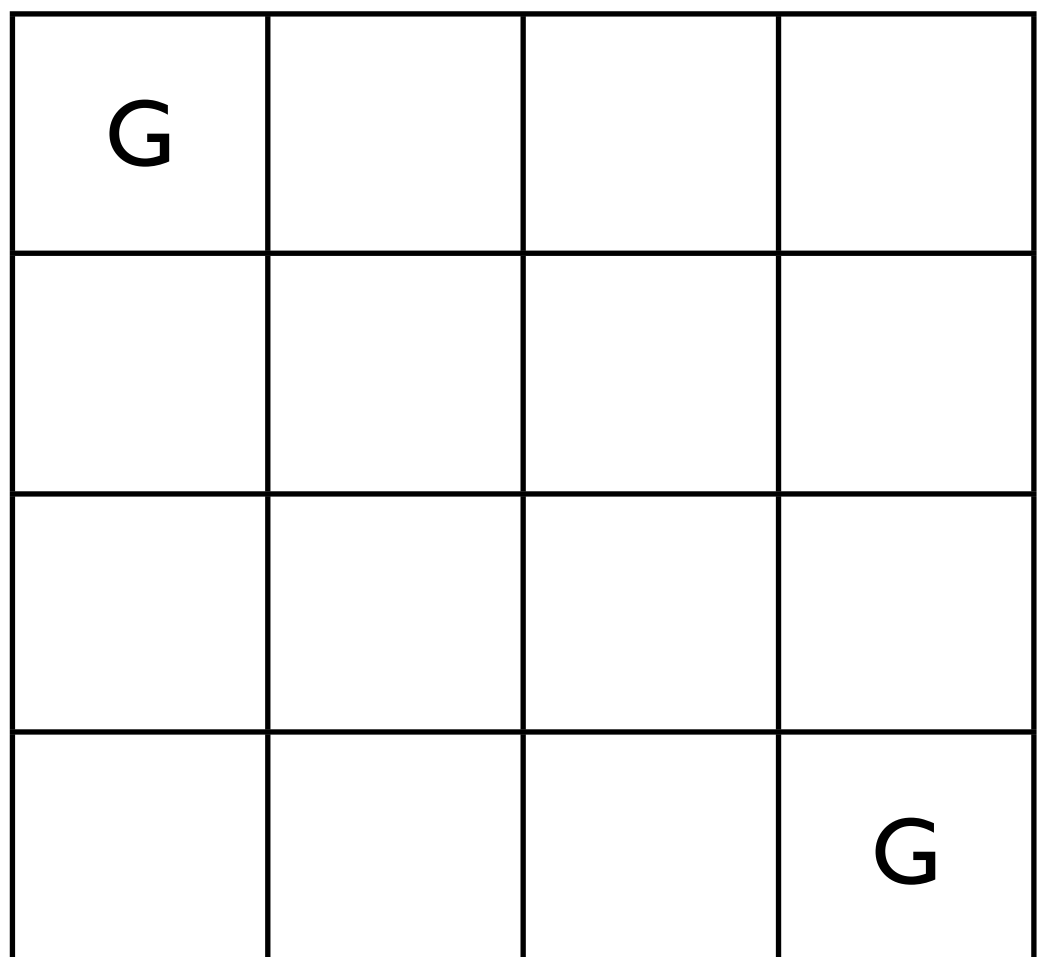
- Every state s_j is represented by a field in the grid
- Action a the agent can choose consists of moving one step to a neighbouring field



A classic example: Grid World

Simplified "Wumpus world" with just two gold pieces

- Every state s_j is represented by a field in the grid
- Action *a* the agent can choose consists of moving one step to a neighbouring field
- *Reward*: -1 in every step until one of the goals (G) is reached.



Simplifying assumptions

Simplifying assumptions

We assume for now:

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)
- Finite number of possible actions a_i

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

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)
- Finite number of possible actions a_i

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

- Finite number of states s_j

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

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)
- Finite number of possible actions a_i

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

- Finite number of states s_j

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

- The context is a constant MDP (*Markov Decision Process*), where reward and new state *s'* only depend on *s*, *a*, and (random) *noise*

Simplifying assumptions

We assume for now:

- Discrete time (steps over time)
- Finite number of possible actions a_i

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

- Finite number of states s_j

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

- The context is a constant MDP (*Markov Decision Process*), where reward and new state s' only depend on s, a, and (random) *noise*
- Environment is observable

The agent’s internal representation

The agent's internal representation

- An agent's *policy* π is the "rule" after which the agent chooses its action a in a given state s

$$\pi(s) \longmapsto a$$

The agent’s internal representation

- An agent’s *policy* π is the “rule” after which the agent chooses its action a in a given state s

$$\pi(s) \longmapsto a$$

- An agent’s *utility function* U describes the expected future reward given s, when following policy π

$$U^{\pi}(s) \longmapsto \Re$$

Grid World: A state's value

A state's value depends on the chosen policy

Grid World: A state's value

A state's value depends on the chosen policy

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

U with optimal policy

Grid World: A state's value

A state's value depends on the chosen policy

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

U with optimal policy

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

U with random policy

A 4x3 world

- Fixed policy - passive learning.

A 4x3 world

- Fixed policy - passive learning.
- Always start in state (1,1).

A 4x3 world

- Fixed policy - passive learning.
- Always start in state (1,1).
- Do trials, observe, until terminal state is reached, update utilities

A 4x3 world

- Fixed policy - passive learning.
- Always start in state (1,1).
- Do trials, observe, until terminal state is reached, update utilities
- Eventually, agent learns *how good the policy is* - it can evaluate the policy and test different ones

A 4x3 world

- Fixed policy - passive learning.
- Always start in state (1,1).
- Do trials, observe, until terminal state is reached, update utilities
- Eventually, agent learns *how good the policy is* - it can evaluate the policy and test different ones
- Policy as described in the left grid is optimal with rewards of -0.04 for all reachable, nonterminal states, and without discounting.

A 4x3 world

- Fixed policy - passive learning.
- Always start in state (1,1).
- Do trials, observe, until terminal state is reached, update utilities
- Eventually, agent learns *how good the policy is* - it can evaluate the policy and test different ones
- Policy as described in the left grid is optimal with rewards of -0.04 for all reachable, nonterminal states, and without discounting.

R	R	R	+1
U		U	-1
U	L	L	L

A 4x3 world

- Fixed policy - passive learning.
- Always start in state (1,1).
- Do trials, observe, until terminal state is reached, update utilities
- Eventually, agent learns *how good the policy is* - it can evaluate the policy and test different ones
- Policy as described in the left grid is optimal with rewards of -0.04 for all reachable, nonterminal states, and without discounting.

R	R	R	+1
U		U	-1
U	L	L	L

0.812	0.868	0.918	+1
0.762		0.660	-1
0.705	0.655	0.611	0.388

Outline

- Reinforcement learning (chapter 21, with some references to 17)
  - Problem definition
    - Learning situation
    - Roll of the reward
    - Simplified assumptions
    - Central concepts and terms
  - Known (observable) environment
    - Bellman's equation
    - Approaches to solutions
  - Unknown environment
    - Monte-Carlo method
    - Temporal-Difference learning
    - Q-Learning
    - Sarsa-Learning
  - Improvements
    - The usefulness of making mistakes
    - Eligibility Trace

Environment model

Environment model

- Where do we get in each step?

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

Environment model

- Where do we get in each step?

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

- What will the reward be?

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

Environment model

- Where do we get in each step?

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

- What will the reward be?

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

Environment model

- Where do we get in each step?

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

- What will the reward be?

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

The utility values of different states obey *Bellman's equation*, given a fixed policy π:

$$U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot U^{\pi}(\delta(s, \pi(s)))$$

Solving the equation

Solving the equation

There are two ways of solving *Bellman's equation*

$$U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot U^{\pi}(\delta(s, \pi(s)))$$

Solving the equation

There are two ways of solving *Bellman's equation*

$$U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot U^{\pi}(\delta(s, \pi(s)))$$

- Directly: $U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot \sum_{s'} P(s' \mid s, \pi(s))\, U^{\pi}(s')$

Recap: Random policy

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

$$U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot \sum_{s'} P(s' \mid s, \pi(s))\, U^{\pi}(s')$$

Solving the equation

There are two ways of solving (this "optimal" version of) *Bellman's equation*

$$U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot U^{\pi}(\delta(s, \pi(s)))$$

- Directly: $U^{\pi}(s) = r(s, \pi(s)) + \gamma \cdot \sum_{s'} P(s' \mid s, \pi(s))\, U^{\pi}(s')$

- Iteratively (*Value / utility iteration*), stop when equilibrium is reached, i.e., "nothing happens"

$$U^{\pi}_{k+1}(s) \longleftarrow r(s, \pi(s)) + \gamma \cdot U^{\pi}_{k}(\delta(s, \pi(s)))$$

Bayesian reinforcement learning

Bayesian reinforcement learning

A remark:

Bayesian reinforcement learning

A remark:

One form of reinforcement learning integrates Bayesian learning into the process to obtain the transition model, i.e., $P(s' \mid s, \pi(s))$

Bayesian reinforcement learning

A remark:

One form of reinforcement learning integrates Bayesian learning into the process to obtain the transition model, i.e., $P(s' \mid s, \pi(s))$

This means to assume a prior probability for each hypothesis on how the model might look like and then applying Bayes' rule to obtain the posterior.

Bayesian reinforcement learning

A remark:

One form of reinforcement learning integrates Bayesian learning into the process to obtain the transition model, i.e., $P(\, s' \mid s, \pi(s))$

This means to assume a prior probability for each hypothesis on how the model might look like and then applying Bayes' rule to obtain the posterior.

We are not going into details here!

Finding optimal policy and value function

Finding optimal policy and value function

How can we find an *optimal policy* π^*?

Finding optimal policy and value function

How can we find an *optimal policy* π^*?

That would be easy if we had the *optimal value / utility function* U^*:

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

Finding optimal policy and value function

How can we find an *optimal policy* π^*?

That would be easy if we had the *optimal value / utility function* U^*:

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

Apply to the "optimal version" of Bellman's equation

$$U^*(s) = \max_{a}\left(r(s, a) + \gamma \cdot U^*(\delta(s, a))\right)$$

Finding optimal policy and value function

How can we find an *optimal policy* π^*?

That would be easy if we had the *optimal value / utility function* U^*:

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

Apply to the "optimal version" of Bellman's equation

$$U^*(s) = \max_{a}\left(r(s,a) + \gamma \cdot U^*(\delta(s,a))\right)$$

Tricky to solve ... but possible:

Combine policy and value iteration by switching in each iteration step

Policy iteration

Policy iteration

Policy iteration provides exactly this switch.

Policy iteration

Policy iteration provides exactly this switch.

For each iteration step k:

$$\pi_k(s) = \underset{a}{\mathrm{argmax}}\left(r(s, a) + \gamma \cdot U_k(\delta(s, a))\right)$$

$$U_{k+1}(s) = r(s, \pi_k(s)) + \gamma \cdot U_k(\delta(s, \pi_k(s)))$$

Outline

- Reinforcement learning (chapter 21, with some references to 17)
  - Problem definition
    - Learning situation
    - Roll of the reward
    - Simplified assumptions
    - Central concepts and terms
  - Known environment
    - Bellman's equation
    - Approaches to solutions
  - Unknown environment
    - Monte-Carlo method
    - Temporal-Difference learning
    - Q-Learning
    - Sarsa-Learning
  - Improvements
    - The usefulness of making mistakes
    - Eligibility Trace

Monte Carlo approach

Monte Carlo approach

Usually the reward $r(s, a)$ and the state transition function $\delta(s, a)$ are unknown to the learning agent.

Monte Carlo approach

Usually the reward *r(s, a)* and the state transition function *δ(s, a)* are unknown to the learning agent.

(What does that mean for learning to ride a bike? $\boxed{}$)

Monte Carlo approach

Usually the reward $r(s, a)$ and the state transition function $\delta(s, a)$ are unknown to the learning agent.

(What does that mean for learning to ride a bike?)

Monte Carlo approach

Usually the reward $r(s, a)$ and the state transition function $\delta(s, a)$ are unknown to the learning agent.

(What does that mean for learning to ride a bike? $\boxed{}$)

Still, we can estimate U^* from *experience*, as a *Monte Carlo approach* will do:

- Start with a randomly chosen s
- Follow a policy π, store rewards and s_t for the step at time t
- When the goal is reached, update the $U^\pi(s)$ estimate for all visited states s_t with the future reward that was given when reaching the goal
- Start over with a randomly chosen s ...

Monte Carlo approach

Usually the reward $r(s, a)$ and the state transition function $\delta(s, a)$ are unknown to the learning agent.

(What does that mean for learning to ride a bike? $\boxed{}$)

Still, we can estimate U^* from *experience*, as a *Monte Carlo approach* will do:

- Start with a randomly chosen s
- Follow a policy π, store rewards and s_t for the step at time t
- When the goal is reached, update the $U^\pi(s)$ estimate for all visited states s_t with the future reward that was given when reaching the goal
- Start over with a randomly chosen s ...

Converges slowly...

Temporal Difference learning

Temporal Difference learning

Temporal Difference learning ...

Temporal Difference learning

Temporal Difference learning ...

... uses the fact that there are two estimates for the value of a state:

Temporal Difference learning

Temporal Difference learning ...

... uses the fact that there are two estimates for the value of a state:

before and *after* visiting the state

Temporal Difference learning

Temporal Difference learning ...

... uses the fact that there are two estimates for the value of a state:

before and *after* visiting the state

Temporal Difference learning

Temporal Difference learning ...

... uses the fact that there are two estimates for the value of a state:

before and *after* visiting the state

Temporal Difference learning

Temporal Difference learning ...

... uses the fact that there are two estimates for the value of a state:

before and *after* visiting the state

Or: What the agent believes *before* acting

$U^{\pi}(s_t)$

Temporal Difference learning

Temporal Difference learning ...

... uses the fact that there are two estimates for the value of a state:

before and *after* visiting the state

Or: What the agent believes *before* acting

$U^{\pi}(s_t)$

and *after* acting

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

Applying the estimates

Applying the estimates

The second estimate in the *Temporal Difference* learning approach is obviously "better", ...

Applying the estimates

The second estimate in the *Temporal Difference* learning approach is obviously "better", ...

... hence, we update the overall approximation of a state's value towards the more accurate estimate

Applying the estimates

The second estimate in the *Temporal Difference* learning approach is obviously "better", ...

... hence, we update the overall approximation of a state's value towards the more accurate estimate

$$U^{\pi}(s_t) \longleftarrow U^{\pi}(s_t) + \alpha[r_{t+1} + \gamma \cdot U^{\pi}(s_{t+1}) - U^{\pi}(s_t)]$$

Applying the estimates

The second estimate in the *Temporal Difference* learning approach is obviously "better", ...

... hence, we update the overall approximation of a state's value towards the more accurate estimate

$$U^{\pi}(s_t) \longleftarrow U^{\pi}(s_t) + \alpha[r_{t+1} + \gamma \cdot U^{\pi}(s_{t+1}) - U^{\pi}(s_t)]$$

Which gives us a measure of the "surprise" or "disappointment" for the outcome of an action.

Applying the estimates

The second estimate in the *Temporal Difference* learning approach is obviously "better", ...

... hence, we update the overall approximation of a state's value towards the more accurate estimate

$$U^{\pi}(s_t) \longleftarrow U^{\pi}(s_t) + \alpha[r_{t+1} + \gamma \cdot U^{\pi}(s_{t+1}) - U^{\pi}(s_t)]$$

Which gives us a measure of the "surprise" or "disappointment" for the outcome of an action.

Converges significantly faster than the pure Monte Carlo approach.

Q-learning

Q-learning

Problem:

Q-learning

Problem:

even if U is appropriately estimated, it is not possible to compute π, as the agent has no knowledge about δ and r, i.e., it needs to learn also that.

Q-learning

Problem:

even if U is appropriately estimated, it is not possible to compute π, as the agent has no knowledge about δ and r, i.e., it needs to learn also that.

Solution (trick): Estimate $Q(s, a)$ instead of $U(s)$:

Q-learning

Problem:

even if U is appropriately estimated, it is not possible to compute π, as the agent has no knowledge about δ and r, i.e., it needs to learn also that.

Solution (trick): Estimate $Q(s, a)$ instead of $U(s)$:

$Q(s, a)$: Expected total reward when choosing a in s

Q-learning

Problem:

even if U is appropriately estimated, it is not possible to compute π, as the agent has no knowledge about δ and r, i.e., it needs to learn also that.

Solution (trick): Estimate $Q(s, a)$ instead of $U(s)$:

$Q(s, a)$: Expected total reward when choosing a in s

$$\pi(s) = argmax\ Q(s, a)$$

Q-learning

Problem:

even if U is appropriately estimated, it is not possible to compute π, as the agent has no knowledge about δ and r, i.e., it needs to learn also that.

Solution (trick): Estimate $Q(s, a)$ instead of $U(s)$:

$Q(s, a)$: Expected total reward when choosing a in s

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

Q-learning

Problem:

even if U is appropriately estimated, it is not possible to compute π, as the agent has no knowledge about δ and r, i.e., it needs to learn also that.

Solution (trick): Estimate $Q(s, a)$ instead of $U(s)$:

$Q(s, a)$: Expected total reward when choosing a in s

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

$$U^*(s) = max\, Q^*(s, a)$$

Q-learning

Problem:

even if U is appropriately estimated, it is not possible to compute π, as the agent has no knowledge about δ and r, i.e., it needs to learn also that.

Solution (trick): Estimate $Q(s, a)$ instead of $U(s)$:

$Q(s, a)$: Expected total reward when choosing a in s

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

$$U^*(s) = \max_{a} Q^*(s, a)$$

Q-learning

Problem:

even if *U* is appropriately estimated, it is not possible to compute π, as the agent has no knowledge about δ and *r*, i.e., it needs to learn also that.

Solution (trick): Estimate $Q(s, a)$ instead of $U(s)$:

$Q(s, a)$: Expected total reward when choosing *a* in *s*

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

$$U^*(s) = \max_{a} Q^*(s, a)$$

Learning Q

Learning Q

How can we learn Q?

Learning Q

How can we learn Q?

Also the Q-function can be learned using the Temporal Difference approach:

Learning Q

How can we learn *Q*?

Also the *Q*-function can be learned using the Temporal Difference approach:

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

Learning Q

How can we learn Q?

Also the Q-function can be learned using the Temporal Difference approach:

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

Learning Q

How can we learn Q?

Also the Q-function can be learned using the Temporal Difference approach:

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

With s' being the next state that is reached when choosing action a'

Learning Q

How can we learn *Q*?

Also the *Q*-function can be learned using the Temporal Difference approach:

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

With *s'* being the next state that is reached when choosing action *a'*

Again, a problem: the *max* operator requires obviously a search through all possible actions that can be taken in the next step...

SARSA-learning

SARSA-learning

SARSA-learning works similar to *Q-learning*, but it is the *currently active policy* that controls the actually taken action *a'*:

SARSA-learning

SARSA-learning works similar to *Q-learning*, but it is the *currently active policy* that controls the actually taken action a':

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

SARSA-learning

SARSA-learning works similar to *Q-learning*, but it is the *currently active policy* that controls the actually taken action a':

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

SARSA-learning

SARSA-learning works similar to *Q-learning*, but it is the *currently active policy* that controls the actually taken action a':

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

Got its name from the "experience tuples" having the form

State-Action-Reward-State-Action

SARSA-learning

SARSA-learning works similar to *Q-learning*, but it is the *currently active policy* that controls the actually taken action a':

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

Got its name from the "experience tuples" having the form

State-Action-Reward-State-Action

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

Outline

- Reinforcement learning (chapter 21, with some references to 17)
  - Problem definition
    - Learning situation
    - Roll of the reward
    - Simplified assumptions
    - Central concepts and terms
  - Known environment
    - Bellman's equation
    - Approaches to solutions
  - Unknown environment
    - Monte-Carlo method
    - Temporal-Difference learning
    - Q-Learning
    - Sarsa-Learning
  - Improvements
    - The usefulness of making mistakes
    - Eligibility Trace

Improvements and adaptations

What can we do, when ...

- ... the environment is not fully observable?
- ... there are too many states?
- ... the states are not discrete?
- ... the agent is acting in continuous time?

Allowing to be wrong sometimes

Allowing to be wrong sometimes

Exploration - Exploitation dilemma: When following one policy based on the current estimate of Q, it is not guaranteed that Q actually converges to Q^* (the optimal Q).

Allowing to be wrong sometimes

Exploration - Exploitation dilemma: When following one policy based on the current estimate of Q, it is not guaranteed that Q actually converges to Q^* (the optimal Q).

A simple solution: Use a policy that has a certain probability of "being wrong" once in a while, to explore better.

Allowing to be wrong sometimes

Exploration - Exploitation dilemma: When following one policy based on the current estimate of Q, it is not guaranteed that Q actually converges to Q^* (the optimal Q).

A simple solution: Use a policy that has a certain probability of "being wrong" once in a while, to explore better.

- ε-*greedy*: Will sometimes (with probability ε) pick a random action instead of the one that looks best (*greedy*)

Allowing to be wrong sometimes

Exploration - Exploitation dilemma: When following one policy based on the current estimate of Q, it is not guaranteed that Q actually converges to Q^* (the optimal Q).

A simple solution: Use a policy that has a certain probability of "being wrong" once in a while, to explore better.

- ε-*greedy*: Will sometimes (with probability ε) pick a random action instead of the one that looks best (*greedy*)
- *Softmax*: Weighs the probability for choosing different actions according to how "good" they appear to be.

ε-greedy Q-learning

ε-greedy Q-learning

A suggested algorithm (ε-greedy implementation, given some “black box”, that produces *r* and *s*’, given *s* and *a*)

ε-greedy Q-learning

A suggested algorithm (ε-greedy implementation, given some "black box", that produces r and s', given s and a)

- Initialise $Q(s, a)$ arbitrarily $\forall s, a$, choose learning rate α and discount factor γ

ε-greedy Q-learning

A suggested algorithm (ε-greedy implementation, given some "black box", that produces r and s', given s and a)

- Initialise $Q(s, a)$ arbitrarily $\forall s, a,$ choose learning rate α and discount factor γ
- Initialise s

ε-greedy Q-learning

A suggested algorithm (ε-greedy implementation, given some "black box", that produces r and s', given s and a)

- Initialise $Q(s, a)$ arbitrarily $\forall s, a$, choose learning rate α and discount factor γ
- Initialise s
- Repeat for each step:
    - Choose a from s using ε-greedy policy based on $Q(s, a)$
    - Take action a, observe reward r, and next state s'
    - Update $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    - replace s with s'

ε-greedy Q-learning

A suggested algorithm (ε-greedy implementation, given some "black box", that produces r and s', given s and a)

- Initialise $Q(s, a)$ arbitrarily $\forall s, a,$ choose learning rate α and discount factor γ
- Initialise s
- Repeat for each step:
    - Choose a from s using ε-greedy policy based on $Q(s, a)$
    - Take action a, observe reward r, and next state s'
    - Update $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    - replace s with s'

until T steps.

ε-greedy Q-learning

A suggested algorithm (ε-greedy implementation, given some "black box", that produces r and s', given s and a)

- Initialise $Q(s, a)$ arbitrarily $\forall s, a$, choose learning rate α and discount factor γ
- Initialise s
- Repeat for each step:
  - Choose a from s using ε-greedy policy based on $Q(s, a)$
  - Take action a, observe reward r, and next state s'
  - Update $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
  - replace s with s'

until T steps.

Speeding up the process

Idea: the Time Difference (TD) updates can be used to improve the estimation also of states where the agent has already been earlier.

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

With e the *eligibility trace*, telling how long ago the agent visited s and chose action a

Often called *TD(* λ*)*, with λ being the time constant that describes the "*annealing rate*" of the trace.

Application examples

- Game playing.
  - A. Samuel's checkers program (1959). Remarkable: did not use any rewards... but was managed to converge anyhow...
  - G. Tesauro's backgammon program from 1992, first introduced as Neurogammon, with a neural network representation of Q(s, a). Required an expert for tedious training ;-) The newer version TD-gammon learned from self-play and rewards at the end of the game according to generalised TD-learning. Played quite well after two weeks of computing time ...
- Robotics
  - Classic example: the inverse pendulum (cart-pole). Two actions: jerk right or jerk left (bang-bang control). First learning algorithm to this problem applied in 1968 (Michie and Chambers), using a real cart!
  - More recently: Pancake flipping ;-)

Flipping ... a piece of (pan)cake?

Video from programming-by-demonstration.org
(Dr. Sylvain Calinon & Dr. Petar Kormushev)

Homework for Machine Learning

- Homework 3 is related to machine learning, announced on the course page
- Choose between 3a, 3b, 3c (or do several), but only one (the best) will contribute in the end as homework 3
- 3c is in the area of today's lecture (slides will be provided after the lecture ;-)
- The task: get a little two-legged agent ("robot") to learn to "walk"
- Some programming effort is involved (instructions provided)
- Main idea is to explore different reinforcement learning approaches and compare their effect on the agent's success (or failure...) and report on the experience
- A series of images for "animation" of the agent is provided
- Support methods for the "animation" of the agent's walk are provided in Matlab and Python (transferring to Java should also be easily possible, the Matlab code is less than 30 lines long)

Homework for Machine Learning cont'd

- Seemingly "simple" task - just doing it gives a grade 3 at maximum.
- BUT: the important part of this task is the INTERPRETATION and DISCUSSION of results, which should be done in a thoroughly prepared and written REPORT. Please make sure you have read the instructions carefully before starting the work!

- Deadline for handing in: May 10, 2012.