Introduction to Reinforcement Learning

Michael Baumann

University of Paderborn

Research Group Knowledge-Based Systems
Prof. Dr. Kleine Büning

winter term 2013

UNIVERSITY OF PADERBORN
Knowledge-Based Systems

Outline

Motivation

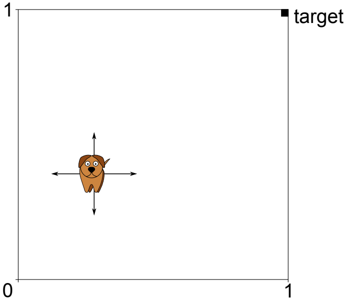
- How can an agent learn to choose optimal actions in each state to achieve its goals?
  - → Learning from interaction
  - → Reward and punishment

Definition (Reinforcement Learning [Sutton and Barto, 1998])

"Reinforcement learning is learning what to do–how to map situations to actions–so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them."

(Single) Agent Systems in a Nutshell

- Agent:
  - Situated in an environment
  - Subject of learning
  - Perceives (probably only a portion) of the environment's state
    $\rightarrow$ e.g. sonar, camera, ...
  - Can perform actions to act in or change the environment
    $\rightarrow$ e.g. move, turn, ...
- Environment:
  - Everything outside the agent
  - Observable state
  - Offers reward / punishment (RL)



Classification of Learning Techniques

- Supervised Learning needs labeled training samples
- Unsupervised Learning has no information on the correct solution; similar structures are found
- Reinforcement Learning uses a (delayed) feedback of the environment as measure, without stating the correct solution

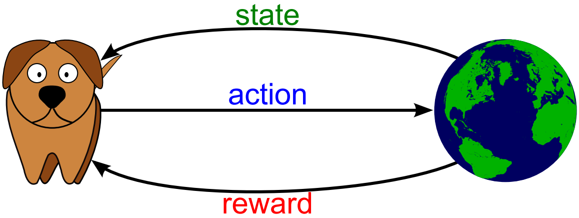
Example 1 (Reinforcement Learning).

Scenario: An agent hast to learn a board game
Formulation: The agent receives a *reward* if it won the game and a *punishment* (negative reward) if it loses. All other situation result in neutral feedbacks.

Outline

Reinforcement Learning

. . . in a Nutshell



- Interaction with environment via states and actions
- Reward as feedback for the last action
- Agent discovers usability of actions during learning
- Goal: Find policy, that maximizes discounted returns

Reinforcement Learning in Single Agent Systems

- Reward function offers numerical rewards for state-action pairs

→ Goal: Learn successful policy for any state

→ Maximizing this reward leads to proper behavior

- No labeled examples, i.e. no information on correct behavior
- Agent is not told which action to choose

→ Trial-and-error

- Learning agent has often no knowledge about its environment
- Difficulty: Current actions may influence future rewards
- Formulation as Markov Decision Process

The Markov Property

- Environment's behavior may depend on the complete history:

$$P\left[s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_0, s_0, a_0\right]$$

- If the state signal has the Markov Property, the response at $t+1$ only depends on the state and action at time t:

$$P\left[s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\right]$$

- If the system has the Markov property, both probability distributions are equal!

Example 2.

Configuration of all pieces on the board in checkers.

Markov Decision Process

Definition 1 (Markov Decision Process).

A Markov Decision Process is defined by a tuple (S, A, r, δ) with:

- *Finite set of states S*
- *Finite set of actions A*
- *Reward function r*
- *State transition function δ*
- *The state signal has the Markov property*

- $s_t \in S$ is perceived in time t and $a_t \in A$ is executed
- Environment responds with $r_t = r(s_t, a_t) \in \mathbb{R}$ and transitions to state $s_{t+1} = \delta(s_t, a_t)$
- δ, r are part of the environment and may be unknown

Policy

- Policy π determines agent's behavior:

$$\pi : S \rightarrow A$$

- $\pi(s_t) = a_t$ decides upon action in state s_t
- But: what is the optimal policy?

Example 3.

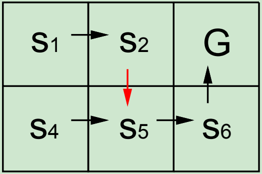


Figure: Illustration of a policy
[Mitchell, Machine Learning]

$$\pi(s_1) = a_{\text{right}}$$

$$\vdots$$

$$\pi(s_6) = a_{\text{up}}$$

(State) Value Function

- Goal: Learn a policy that maximizes the sum of discounted rewards
- Cumulated discounted value $V^{\pi}(s_t)$ starting in s_t following arbitrary policy π:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- Discount factor $0 \leq \gamma < 1$: value of delayed rewards in relation to immediate rewards
  - Reward received i steps in future are discounted by γ^i
  - $\gamma \rightarrow 0$ consider only immediate rewards
  - $\gamma \rightarrow 1$ higher influence of distant rewards
- Again: Optimal policy?

Optimal Policy

- Agent has to learn policy π that maximizes $V^{\pi}(s)$ for all states s
- Optimal Policy $\pi^{\star}$:

$$\pi^{\star} : V^{\pi^{\star}}(s) \geq V^{\pi}(s) \; \forall s \in S, \forall \pi$$

- $V^{\pi^{\star}}(s)$ is the sum of discounted rewards for an optimal policy starting in s
- $V^{\star}(s)$ is short for $V^{\pi^{\star}}(s)$

$\Rightarrow$ The agent's goal is to learn an optimal policy $\pi^{\star}$

Learning an Optimal Policy

- For $V^\star$ it holds:

$$V^\star(s_1) > V^\star(s_2) \Leftrightarrow \text{ agent prefers } s_1 \text{ over } s_2$$

- But: $V^\star$ values states and not actions!
- Optimal action in state s is action a, that maximizes the sum of reward $r(s, a)$ and $V^\star$-value of the successor state:

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

(State-Action) Value Function

- $Q(s, a)$ is the maximal discounted cumulated reward that can be received after executing action a in state s

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

- Assumption: agent follows an optimal policy after performing action a
- Till now, $\pi^\star(s)$ requires δ and r to be known:

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

- $\pi^\star(s)$ in terms of $Q(s, a)$:

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

- Sufficient to learn $Q(s, a)$

Outline

Q-Learning

- How to learn from delayed rewards?
  $\rightarrow$ Iterative approximation
- Close relation between $V^\star(s)$ and $Q(s,a)$:

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

- Recursive formulation of $Q(s,a)$:

$$\begin{aligned} Q(s,a) &= r(s,a) + \gamma V^\star(\delta(s,a)) \\ &= r(s,a) + \gamma \max_{a'} Q(\delta(s,a),a') \end{aligned}$$

$\rightarrow$ Core idea of Q-Learning (Watkins 1989)

Q-Learning Algorithm

- $\widehat{Q}$ is the agent's estimation of Q
- Agent stores estimated Q-values for each state-action pair
- The agent performs action a in state s and observes the reward r and the successor state s'
- Update of the Q-estimation after each step

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

- Update only requires $\widehat{Q}$
- r and s' are known to the agent because the update is performed **after** the environment's reaction

Q-Learning Algorithm

Q Learning

$\forall s, a$ initialize $\widehat{Q}(s, a)$
loop
    observe state s
    select action a and execute it
    receive immediate reward r
    observe new state s'
    update $\widehat{Q}(s, a)$:

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

$\widehat{Q}$ estimation of Q

γ discount factor

Initialization

- All elements zero
- Random
- Heuristic initialization
- ...

Q-Learning in a Grid World

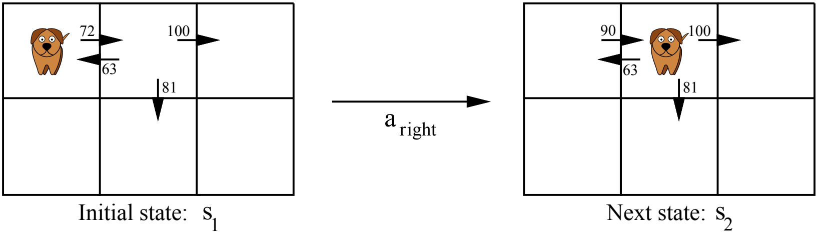


Figure: Grid World
adapted from [Mitchell, Machine Learning]

$$
\begin{aligned}
\widehat{Q}(s_1, a_{\text{right}}) &= r + \gamma \max_{a'} \widehat{Q}(s_2, a') \\
&= 0 + 0.9 \cdot \max\{63, 81, 100\} \\
&= 90
\end{aligned}
$$

Action Selection

- Agent has to perform an action $a \in A$ in each step
- Always choosing action $a = \underset{a'}{\operatorname{argmax}} Q(s, a')$
  - Exploits gained knowledge
  - **But:** Prefers state-action pairs with high values in the beginning
  - Important: visit unknown state-action pairs (s, a) to gain new information (exploration)

$\rightarrow$ Exploration/exploitation Trade-off

ε-greedy: Choose a random action with probability ε and with probability $(1 - \varepsilon)$ an action with highest Q-value

Convergence

Q-Learning with a tabular representation of the knowledge converges to the real Q-values under following assumptions:

1. The system is a deterministic MDP
2. The rewards are bound:

$$\forall s \forall a: \ |r(s,a)| \leq c$$

3. All state-action pairs (s, a) are visited infinitely often

Non-Determinism: Q-Function

- Noisy states and/or erroneous actuators $\Rightarrow$ probabilistic $\delta(s, a)$ and $r(s, a)$
- Non-deterministic MDP: δ and r only depend on s, a
- V^π now expected value of discounted cumulated reward:

$$V^\pi(s_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

- Adjustment of Q-function:

$$\begin{aligned} Q(s, a) &= E[r(s, a) + \gamma V^\star(\delta(s, a))] \\ &= E[r(s, a)] + \gamma E[V^\star(\delta(s, a))] \\ &= E[r(s, a)] + \gamma \sum_{s'} P(s' \mid s, a) V^\star(s') \end{aligned}$$

- $P(s' \mid s, a)$ is probability to transition to s' after executing a in s

Non-Deterministic Q-Learning

- Recursive definition of Q-function:

$$Q(s,a) = E[r(s,a)] + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$$

- Basic Q-update does not converge anymore
  $\rightarrow$ Different rewards for same (s,a) introduce "bouncing"
- Decaying weight α_t for adaptation:

$$\widehat{Q}_t(s,a) = (1-\alpha_t)\widehat{Q}_{t-1}(s,a) + \alpha_t \left(r + \gamma \max_{a'} \widehat{Q}_{t-1}(s',a') \right)$$

- Convergence to optimal solution if $0 < \alpha_t \leq 1$ and

$$\sum_{i=1}^{\infty} \alpha_t = \infty \text{ and } \sum_{i=1}^{\infty} (\alpha_t)^2 < \infty$$

$\rightarrow$ E.g. depending on number of visits of state-action pair:
$\alpha_t = \frac{1}{1+visits_t(s,a)}$

Properties of Q-Learning

- With Q-Learning the estimation $\widehat{Q}$ converges to the true Q-values
- Learning optimal policies with delayed rewards
- No need of domain knowledge
- The Q-value comprises all information on the expected discounted cumulated reward for any state-action pair
- In reality "sufficiently" many visits to each state-action pair are often enough

Outline

Problems with Reinforcement Learning

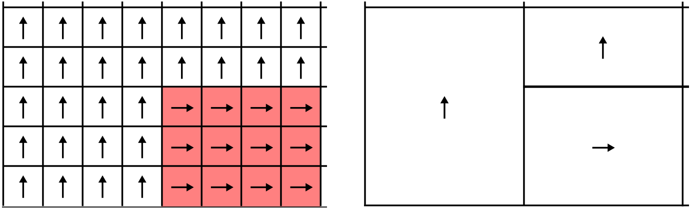
- Length of training: Q-Learning needs **many** iterations (even on small problems)
- Problem of temporal credit assignment: Hard to determine, which action (in a long sequence) led to a later reward
- Exploration: The agent's behavior influences the distribution of trainings examples ($\rightarrow$ exploration vs. exploitation)
- Partially observable states: Often no access to the complete state of the environment (e.g. camera vision of a robot)
- Frequent assumption in RL algorithms: Tabular representation
  - No generalization
  - Storage / time needed to derive good policy

Generalization

Motivation

- Issues with reinforcement learning in large/continuous state spaces
  - Storage needed for all state-action pairs
  - Time required to become sufficiently acquainted with each pair
- Generalization: Apply knowledge to unseen but similar states

$\Rightarrow$ Reduce size of state space



Assumption: Similar states require similar behavior.

Related Work

Generalization is not new, other approaches include:

- Tile Coding: [Sherstov and Stone, 2005], [Whiteson et al., 2007], [Lin and Wright, 2010]
- Basis functions: [Ernst et al., 2005], [Munos and Moore, 2002]
- Vector Quantization: [Lee and Lau, 2004]

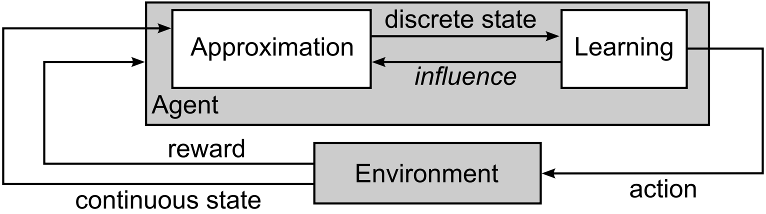
Existing approaches often ...

- need domain knowledge
- are computationally expensive
- are storage intensive
- have non-adaptive approximation shapes

Often: computational issues or domain knowledge assumed

GNG-Q

General Approach



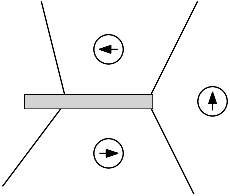
- Parallel learning of behavior and representation

$\Rightarrow$ Q-Learning + Growing Neural Gas = **GNG-Q**

Use information from learning to adjust representation.

GNG-Q

State Regions

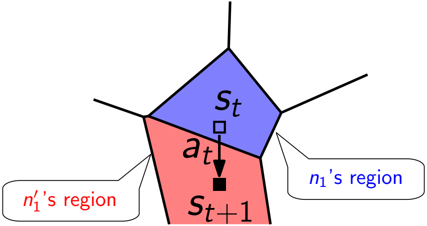


- Assumption: Similar states require similar behavior
- Build state regions with states that
  - are similar regarding some measure
  - require the same behavior

GNG-Q respects similarity in state **and** action space.

GNG-Q

Q-Update



- Standard Q-update rule:

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha\Big[r(s_t, a_t) + \gamma \max_{a' \in A} Q_t(s_{t+1}, a')\Big]$$

- Adopted Q-update for (s_t, a_t):

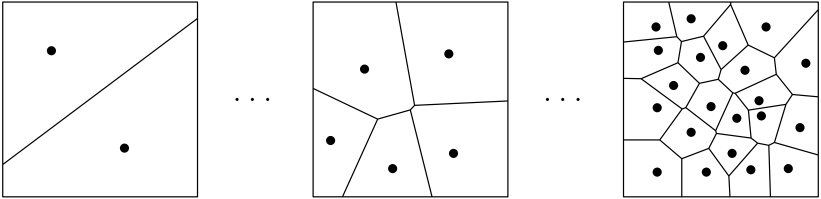
$$Q_{t+1}(n_1, a_t) = (1-\alpha)Q_t(n_1, a_t) + \alpha\Big[r(s_t, a_t) + \gamma \max_{a' \in A} Q_t(n_1', a')\Big]$$

Q-function for state regions defined by neurons.

GNG-Q

Adjusting the Approximation

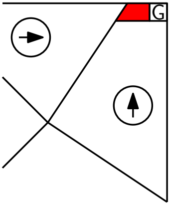
- Start with coarse approximation, initially consisting of two regions
- Q-Learning is applied to the current approximation
- Adjust approximation (refine and adapt)

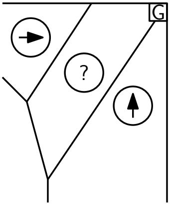


Refinement based on information gained during learning.

GNG-Q

Refinement

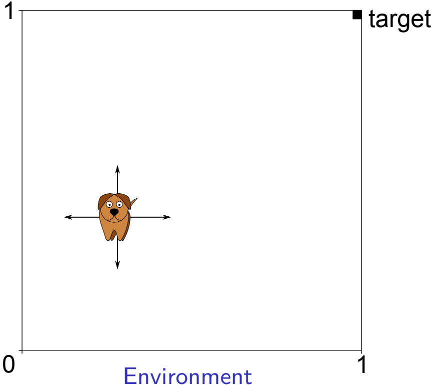




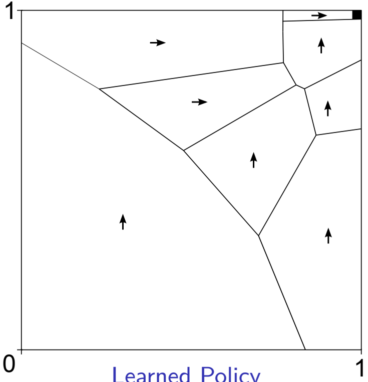
- Refine regions with incompatible states
  $\Rightarrow$ Count changes in the policy for each region

Frequent changes of the policy indicate the need to split.

Experimental Evaluation



Environment



Learned Policy

- Set of states: $S = \{(x, y) \mid x, y \in [0, 1] \subset \mathbb{R}\}$
- Set of actions: one step of size 0.05
- Task: learn the shortest path from all positions to the goal
- Reward: 0 for action leading to the goal, -1 else

Outline

Areas of Application

- Manufacturing optimization
- Scheduling: E.g. assignment of cabs to passengers
- Artificial intelligence for board games
- Robot soccer
- ...

Credits

Singleagent Reinforcement Learning

The sections on Singleagent Reinforcement Learning and Q-Learning are mainly based on [Sutton and Barto, 1998] and Chapter 13 of [Mitchell, 1997]. The Section on Reinforcement Learning in Realistic Scenarios is based on [Baumann and Kleine Büning, 2011] and [Baumann et al., 2012]

References I

[Baumann and Kleine Büning, 2011] M. Baumann and H. Kleine Büning (2011).
State aggregation by growing neural gas for reinforcement learning in continuous state spaces.
In *The Tenth International Conference on Machine Learning and Applications (ICMLA 2011)*, pages 430–435. IEEE Computer Society.

References II

[Baumann et al., 2012] M. Baumann, T. Klerx, and H. Kleine Büning (2012).
Improved state space aggregation with growing neural gas in high-dimensional state spaces.
In *The 5th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS@ECAI 2012)*, pages 27–36.

[Ernst et al., 2005] D. Ernst, P. Geurts, and L. Wehenkel (2005).
Tree-based batch mode reinforcement learning.
Journal of Machine Learning Research, 6:503–556.

[Lee and Lau, 2004] I. S. Lee and H. Y. Lau (2004).
Adaptive state space partitioning for reinforcement learning.
Engineering Applications of Artificial Intelligence, 17:577–588.

References III

[Lin and Wright, 2010] S. Lin and R. Wright (2010).
Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning.
In *AAAI Workshops: Workshop on Abstraction, Reformulation, and Approximation (WARA-2010)*.

[Mitchell, 1997] T. M. Mitchell (1997).
Machine Learning.
McGraw-Hill.

[Munos and Moore, 2002] R. Munos and A. Moore (2002).
Variable resolution discretization in optimal control.
Machine Learning Journal, 49:291–323.

References IV

[Sherstov and Stone, 2005] A. A. Sherstov and P. Stone (2005).
Function approximation via tile coding: Automating parameter choice.
In *SARA 2005*. Springer Verlag.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto (1998).
Reinforcement Learning: An Introduction.
The MIT Press.

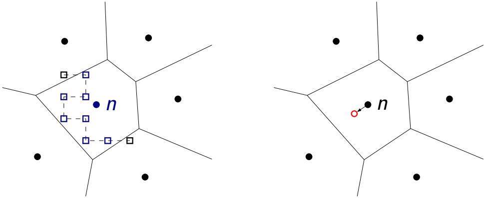
[Whiteson et al., 2007] S. Whiteson, M. E. Taylor, and P. Stone (2007).
Adaptive tile coding for value function approximation.
Technical report, University of Texas at Austin.

Appendix

GNG-Q

Criteria for Adjustments

- Count changes in policy (error of neuron)
- Movement only, if neuron's error is larger than Δ (e.g. $\Delta = 1$)



- Refine the approximation *after* an episode, if
  - $\sum_{n \in N_t} \mathrm{error}(n) > |n_t|$ and
  - At least λ_{insert} episodes have passed since the last refinement

"errors of all neurons are small" $\Leftrightarrow$ "policy stabilized"