Reinforcement Learning

Machine Learning B

708.062 08W 1sst KU

Dr. Stefan Häusler

Institut für Grundlagen der Informationsverarbeitung

Technische Universität Graz

Course WS 2010/11

Second homework set

Theory and practice of reinforcement learning (RL):

- Two theoretical problems
- On- vs. off-policy learning
- Function approximation
- RL with self-play

Assignment 4

4 RL theory I [3 P]

Prove **Corollary 1.1** (p. 7) from the script *Theory of Reinforcement Learning* [3]:

For every policy π there exists a deterministic policy π' such that $\pi' \geq \pi$. As a special case: If there exists a stochastic optimal policy π, then there exists also a deterministic optimal policy π' such that $\pi' \geq \pi$.

Assignment 4

Corollary to

Lemma 1.1. *Assume π and π' are (possibly stochastic) policies with*

$$E_{\pi'}\left[Q^{\pi}(s,\pi'(s))\right] \geq V^{\pi}(s)\ \forall s \in S$$

Then

$$V^{\pi'}(s) \geq E_{\pi'}\left[Q^{\pi}(s,\pi'(s))\right] \geq V^{\pi}(s)\ \forall s \in S$$

- Comparing policies through value functions
- Shows that we can concentrate on deterministic policies

Assignment 5

5 RL theory II [3 P]

Assume that for a given continuing MDP with discount factor $\gamma < 1$ we modify the reward signal by either

a) adding a constant d to all rewards

b) multiplying every reward with a constant $k > 0$

c) linearly transforming the reward signal to $k \cdot r + d$, $k > 0$

d) Let (s_{max}, a_{max}) be the state-action pair that leads to the highest possible immediate reward $r_{max} = max_{s,a} r(s,a)$ in the MDP. Set $r(s_{max}, a_{max}) \leftarrow r_{max} + d,\ d > 0$

e) Let (s_{min}, a_{min}) be the state-action pair that leads to the lowest possible immediate reward $r_{min} = min_{s,a} r(s,a)$ in the MDP. Set $r(s_{min}, a_{min}) \leftarrow r_{min} - d,\ d > 0$

Can this change the optimal policy of the MDP?

Assignment 5

Optimal policy:

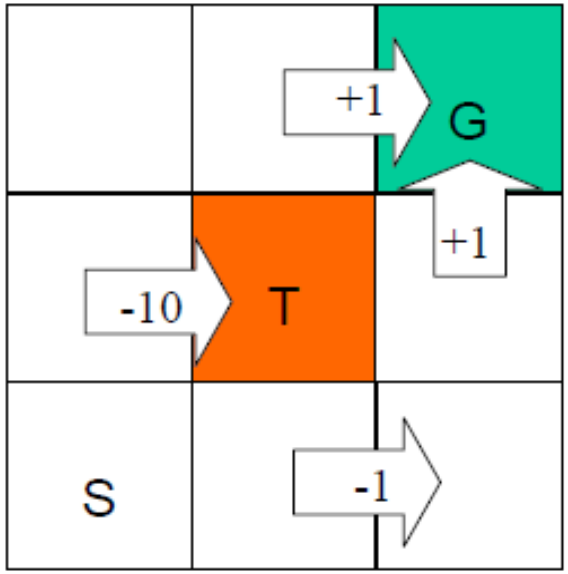
- Highest possible value / return at every state
- Return: $$R_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum^{\infty} \gamma^k r_{t+k+1}$$
- Value function: $$V^{\pi}(s) = E_{\pi}\left[R_t | s_t = s\right] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s\right]$$

Modifications:

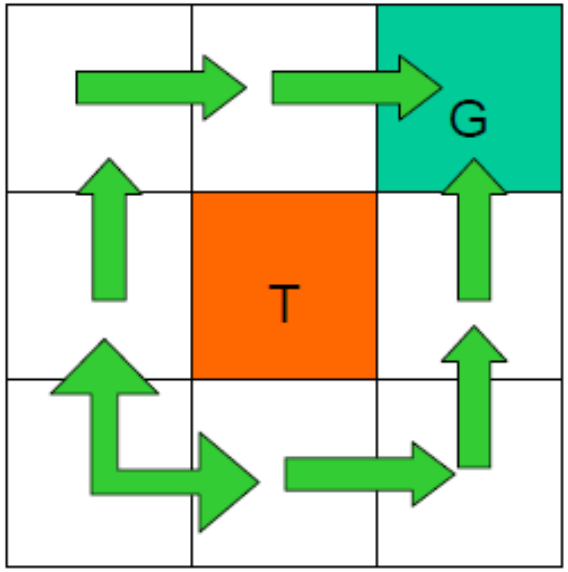
- Linear scaling of all rewards
- Change only maximum / minimum reward

Hint: Carefully read preconditions (γ<1, continuing,…)

Example

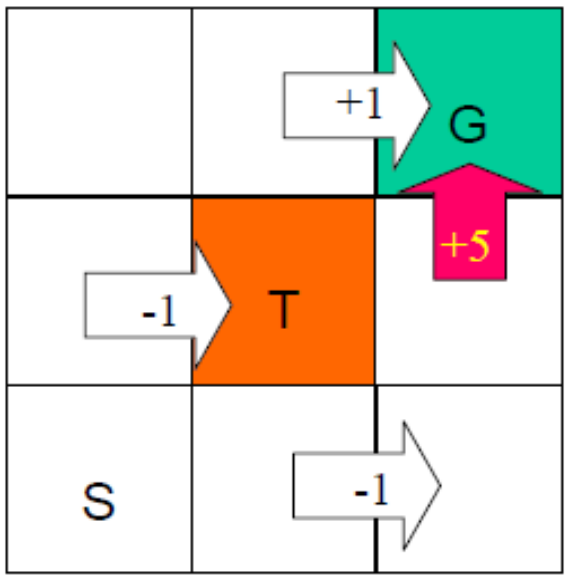


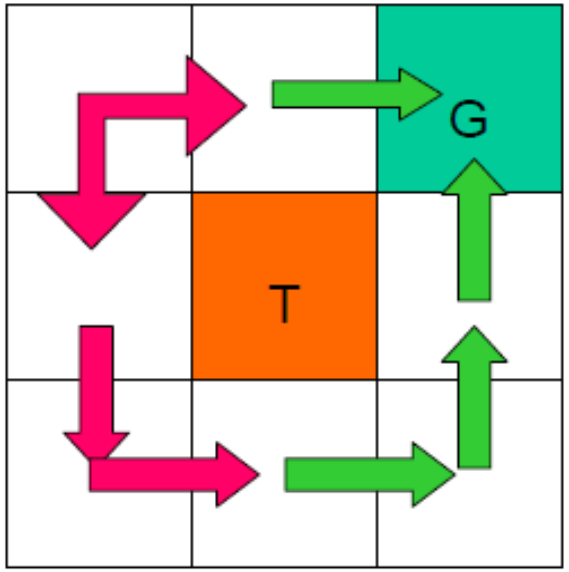
$\gamma = 1$



Optimal policy

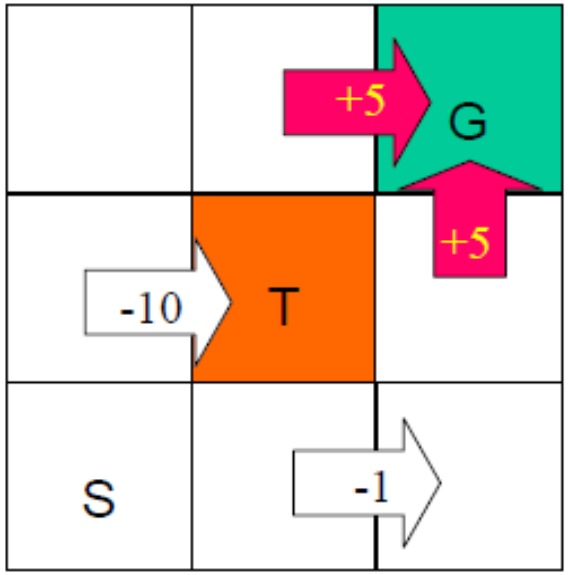
Example



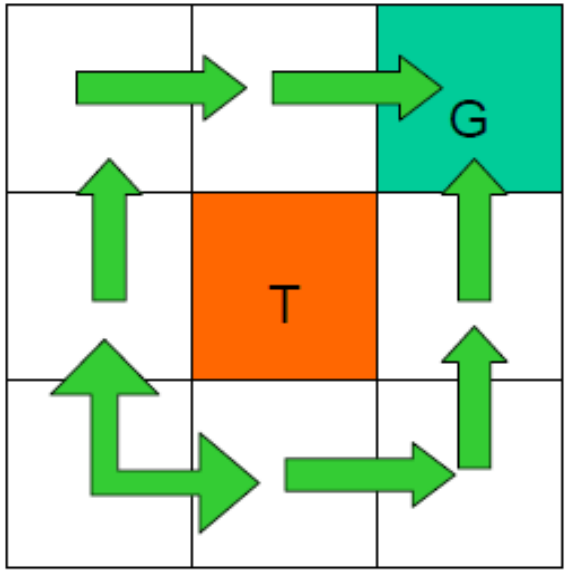


New optimal policy!

Example

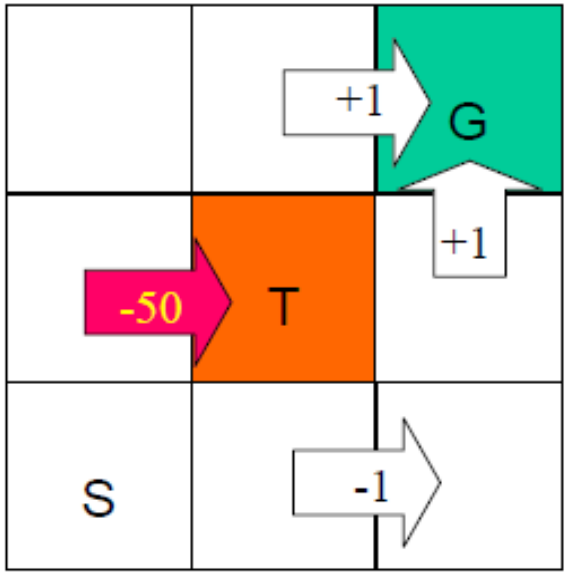


No change if we simultaneously change both goal rewards

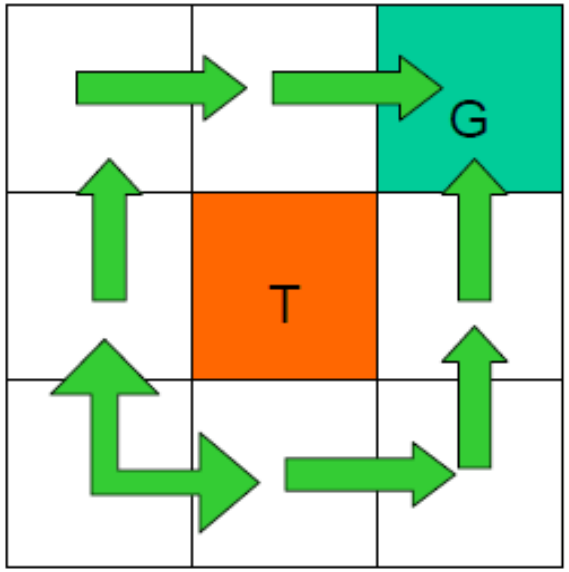


New optimal policy!

Example



Modification of negative reward does not change anything here



New optimal policy!

Programming tasks

MATLAB Reinforcement Learning Toolbox by Jose Antonio Martin (http://www.dacya.ucm.es/jam/download.htm)

You need to modify the MATLAB code

- set up state and action space
- define reward function
- select learning algorithm + parameters

Not using the toolbox is OK - but more work for you!

Assignment 6

6 RL application I: On- and off-policy learning [3 P]

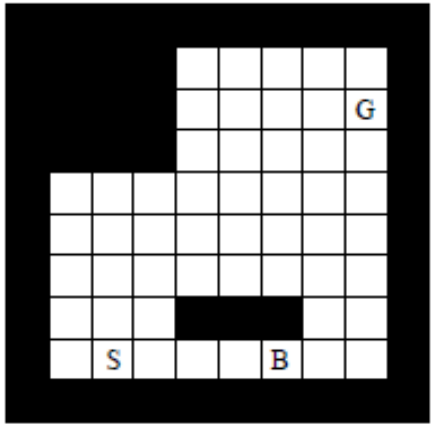


Figure 2: Gridworld with bonus state.

Use *Q-Learning* and *SARSA* without eligibility traces to learn policies for this task. Use ϵ-greedy action selection with a constant $\epsilon = 0.1$. Measure and plot the online performance of both learning algorithms (i.e. average reward per episode),

RL algorithms

Policy Evaluation (the prediction problem):
For a given policy π, compute the state-value function V^π.

$$
\begin{aligned}
V^\pi(s) &= E_\pi[R_t | s_t = s] \\
&= E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s] \\
&= E_\pi[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s] \\
&= E_\pi[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s]
\end{aligned}
$$

Monte Carlo methods

Dynamic Programming
Temporal difference learning

R_t … actual return following time t

γ … discount factor

$E^\pi[.]$ … expectation value w.r.p to policy π

Temporal difference learning

Recall:

Simple every-visit Monte Carlo method

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t) \right]$$

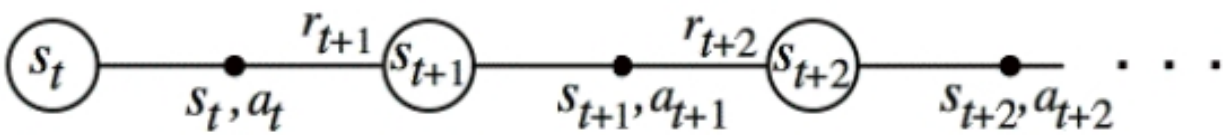
target: the actual return after time t

The simplest TD method,TD(0):

$$V(s_t) \leftarrow V(s_t) + \alpha \left[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{target: an estimate of the return}} - V(s_t) \right]$$

Learning a Q-function

Estimate Q^{π} for the current policy π.



After each transition from a non-terminal state s_t, do

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

Compare: $V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)\right]$

If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0.$

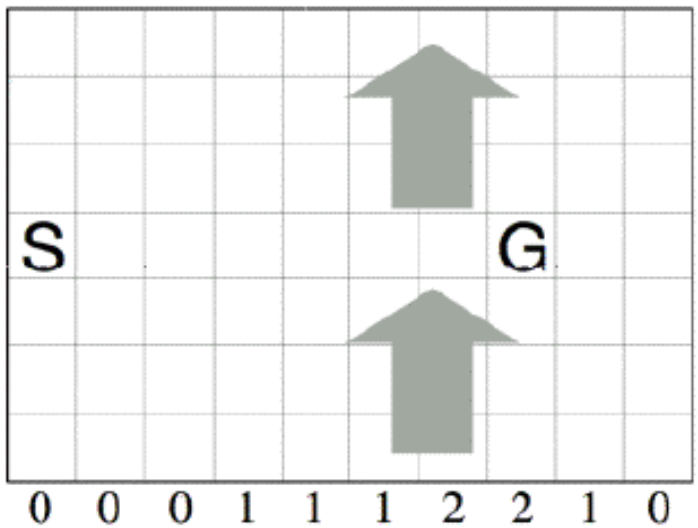
SARSA: On-policy TD learning

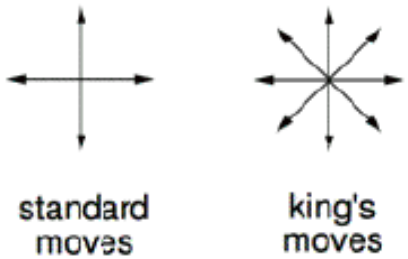
One-step SARSA-learning:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma\, Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right]$$

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Example: Windy grid-world





Undiscounted, episodic, reward = –1 until the goal is reached.

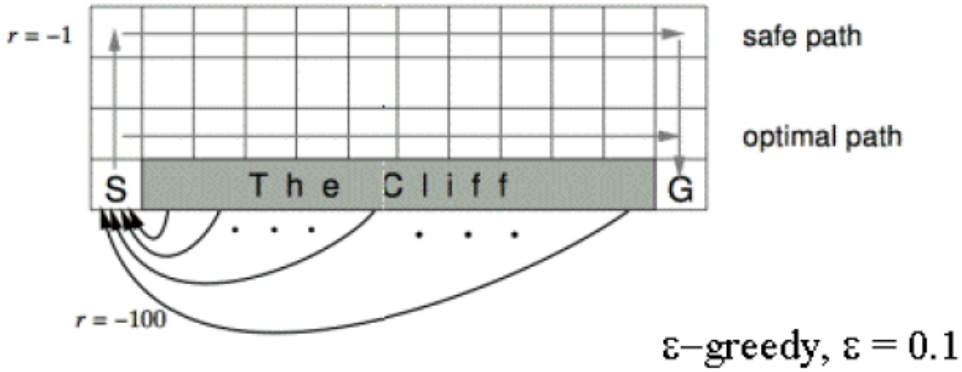
Q-learning: Off-policy TD learning

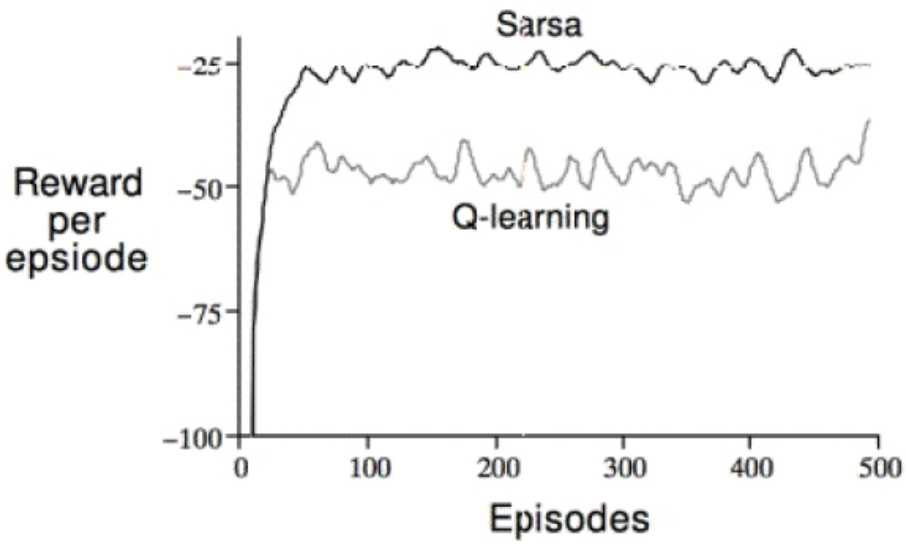
One-step Q-learning:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right]$$

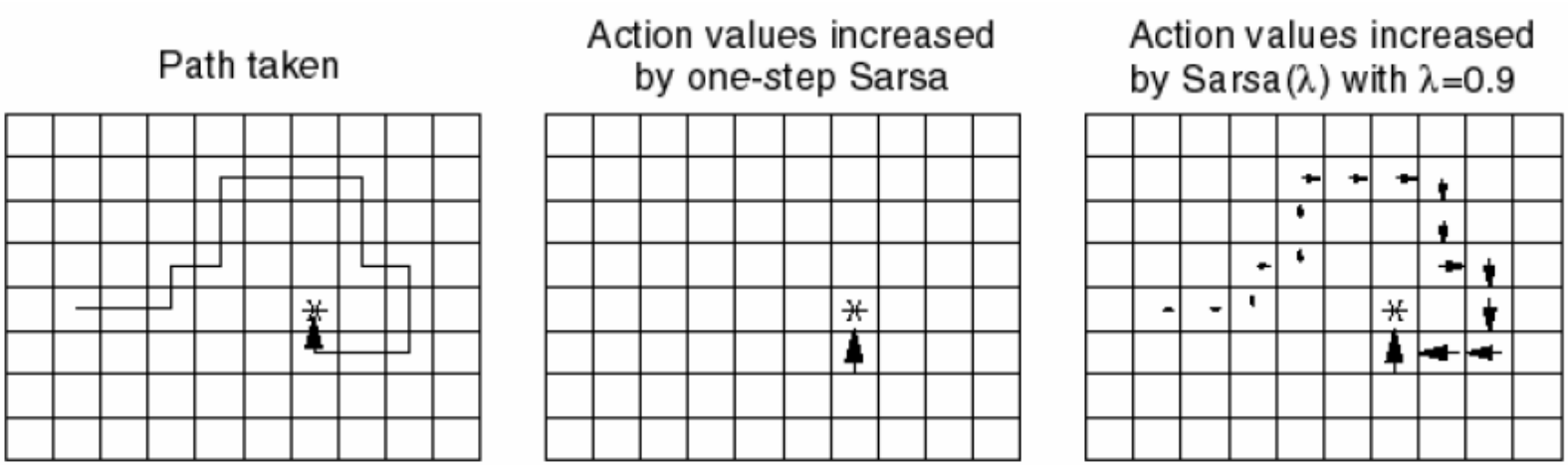
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$;
  until s is terminal

Example: Cliff-walking





Eligibility traces



Within one trial more information about how to get to the goal.

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases}$$

Can considerably accelerate learning

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$$

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

Sarsa(λ): On-policy TD learning with eligibility traces

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s,a

Repeat (for each episode) :

  Initialize s,a

  Repeat (for each step of episode) :

    Take action a, observe r,s'

    Choose a' from s' using policy derived from Q (e.g. ? - greedy)

    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$

    $e(s,a) \leftarrow e(s,a) + 1$

    For all s,a :

      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$

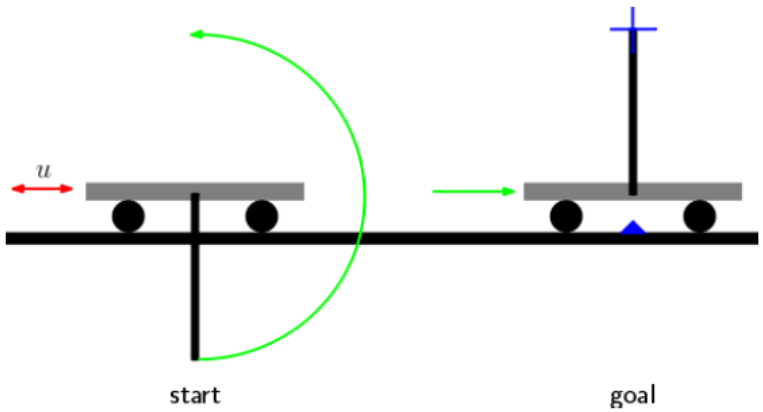
      $e(s,a) \leftarrow \gamma \lambda e(s,a)$

    $s \leftarrow s'; a \leftarrow a'$

  Until s is terminal

Assignment 7

7 RL application II: Function approximation [3 P]



$(xc, \dot{xc}, \varphi, \dot{\varphi})^T$

xc is the position of the cart

φ is the pole angle

Solve the cart-pole problem with the SARSA(λ) algorithm and linear function approximation.

Gradient descent methods

$$\vec{\theta}_t = \left(\theta_t(1), \theta_t(2), \ldots, \theta_t(n)\right)^T$$ transpose

Assume V_t is a (sufficiently smooth) differentiable function of $\vec{\theta}_t$, for all $s \in S$. Define

$$MSE(\theta_t) = \sum_{s \in S} P(s)\left[V^{\pi}(s) - V_t(s)\right]^2$$

$P(s)$ ... The **on-policy distribution**: the distribution created while following the policy being evaluated. Stronger results are available for this distribution.

Linear value function approximation (I)

Gradient descent on the value function

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \alpha/2 \nabla_{\vec{\theta}_t} [V^\pi(s_t) - V_t(s_t)]^2$$
$$= \vec{\theta}_t + \alpha [V^\pi(s_t) - V_t(s_t)] \nabla_{\vec{\theta}_t} V^\pi(s_t)$$

- Step size decreases appropriately
- On-line sampling (states sampled from the on-policy distribution)
- Converges to parameter vector $\vec{\theta}_\infty$ with property:

$$MSE(\vec{\theta}_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma} MSE(\vec{\theta}^*)$$

best parameter vector

(Tsitsiklis & Van Roy, 1997)

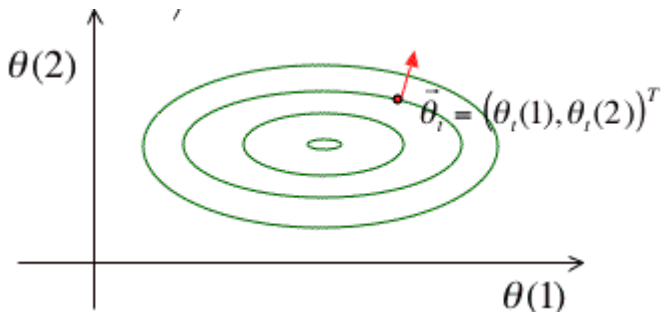
Linear value function approximation (II)

Represent states as feature vectors:

for each $s \in S$:

$$\vec{\phi}_s = \left(\phi_s(1), \phi_s(2), \ldots, \phi_s(n)\right)^T$$

$$V_t(s) = \vec{\theta}_t^T \, \vec{\phi}_s = \sum_{i=1}^{n} \theta_t(i) \, \phi_s(i)$$



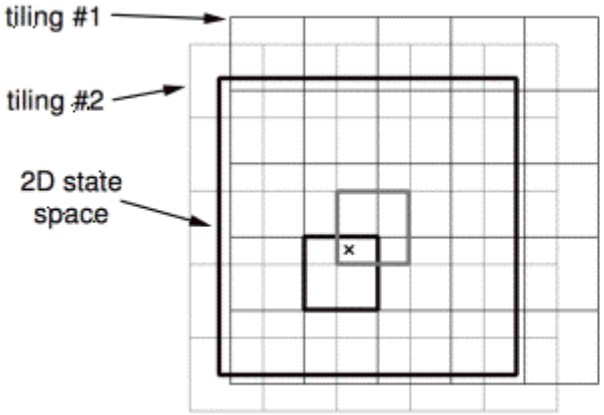
The gradient is very simple: $\nabla_{\vec{\theta}} V_t(s) = \vec{\phi}_s$

Linear value function approximation (III)

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
  $\vec{e} = 0$
  $s \leftarrow$ initial state of episode
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $\vec{e} \leftarrow \gamma\lambda\vec{e} + \nabla_{\vec{\theta}} V(s)$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$ i.e. $\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha[V^{\pi}(s_t) - V_t(s_t)]\nabla_{\vec{\theta}_t} V^{\pi}(s_t)$
    $s \leftarrow s'$
  until s is terminal

What feature vectors to chose? (I)

Tile coding:



Shape of tiles ⇒ Generalization

#Tilings ⇒ Resolution of final approximation

Binary feature for each tile

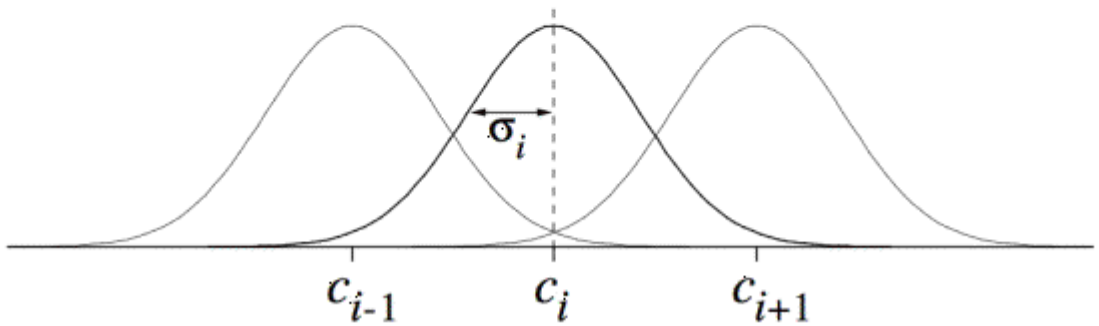
Number of features present at any one time is constant

Binary features means weighted sum easy to compute

What feature vectors to chose? (II)

RBF coding:

$$\phi_s(i) = \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right)$$



How to turn this into a control problem

How to estimate the Q function?

The general gradient-descent rule:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\left[v_t - Q_t(s_t,a_t)\right]\nabla_{\vec{\theta}} Q(s_t,a_t)$$

Gradient-descent Sarsa(λ) (backward view):

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\,\delta_t\,\vec{e}_t$$

where

$$\delta_t = \overbrace{r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1})}^{v_t} - Q_t(s_t,a_t)$$

$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \nabla_{\vec{\theta}}\vec{Q}_t(s_t,a_t)$$

Convergence is not guaranteed anymore.

Use a different parameter vector for each action. Update the one for a_t.

Assignment 8

8 RL application III: Self-play [3* P]

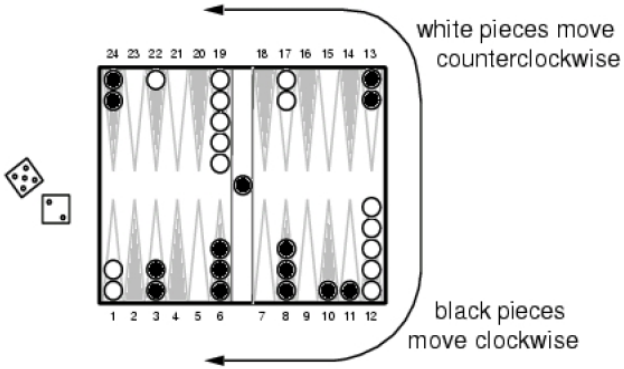
Learn a successful game strategy for a simple two-player game via self-play. The choice of the game and the learning algorithm is up to you. Suggestions for two-player games are *Tic-Tac-Toe*, *Blackjack* or *Nim*. If you want to choose you own game please send an email to *haeusler@igi.tugraz.at* before you start with the implementation. Implement the RL learning algorithm and the game environment in MATLAB using the Reinforcement Learning MATLAB Toolbox [6]. Evaluate the performance of your method. Document the implementation and the results in such a way that anybody can reproduce them effortless.

Learning with self-play

RL is becoming increasingly used in Computer Games, e.g. Black and White

Biggest Success Story of RL: World-class Backgammon program [Tesauro,1995]

- **TD-Gammon** is learning autonomously
- Little prior knowledge
- Most computer games rely heavily on knowledge of AI designer
- Training by playing against itself, i.e. self-play



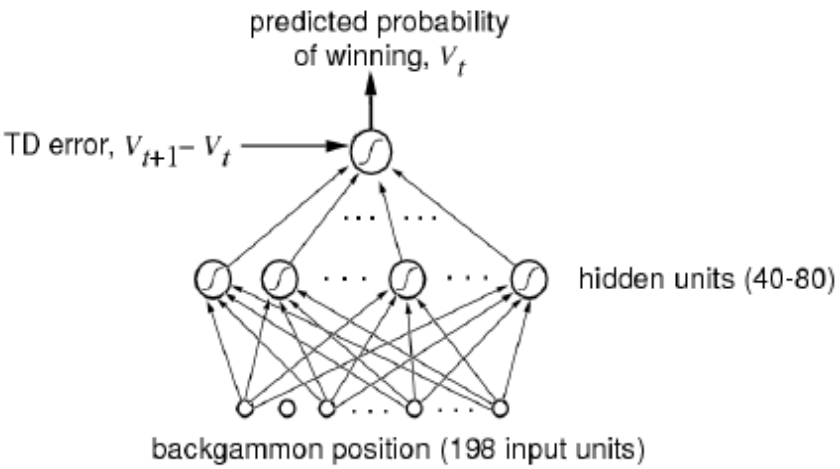
TD Gammon: How it works (I)

Evaluation Function:

- Represented by Neural Network
- NN yields probability of winning for every position

Input representation:

- binary representation of current board position
- Later: complex Backgammon-features



TD Gammon: How it works (II)

Training

- Plays training matches against itself
- Rewards: +1 for victory, –1 for loss, otherwise 0
- undiscounted: $\gamma = 1$
- TD(λ) learning improves estimation for non-terminal positions
- Neural Network is trained (via Backprop) for new estimated winning probabilities

Initially random weights

Backpropagation of TD-Error

Performance

- Increases with number of training matches
- Up to 1.5 Mio. training matches required for learning

TD Gammon: Results

Original version (no prior knowledge)

- plays comparable to other programs

Improvement: Feature Definition, hidden Neurons

- Use prior backgammon knowledge to define sensible features (not raw board position)
- Among top 3 players in the world (human and computer)

Signs of Creativity

- TD-Gammon played some opening moves differently than human grandmasters
- Statistical evaluation: TD-Gammon moves are better
- Today humans play the same moves!

Task for assignment 8

Find a 2-player game with a small state set, e.g. Tic-Tac-Toe, Blackjack, Nim

- suitable for tabular learning (no approximation)

Let your program learn with self-play.

Evaluate playing strength.