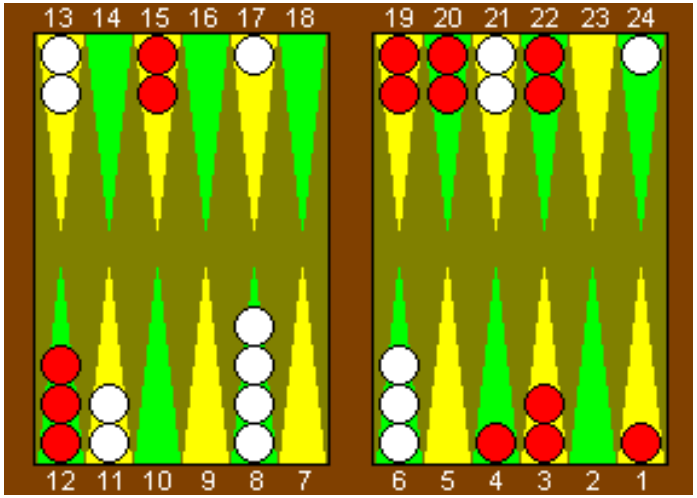
Reinforcement Learning



Lektor Dr.techn. Alexander K. Seewald
Österreichisches Forschungsinstitut
für Artificial Intelligence

alex@seewald.at / alex.seewald.at

Reinforcement Learning

Choose optimal actions (control strategies, policies) based on experience gained by acting in an environment.

- learning without a teacher
- learning by trial and error

Typical scenarios

- Mobile robot navigation, movement (leg control) and goal learning, Game Playing (Backgammon, Othello), ...

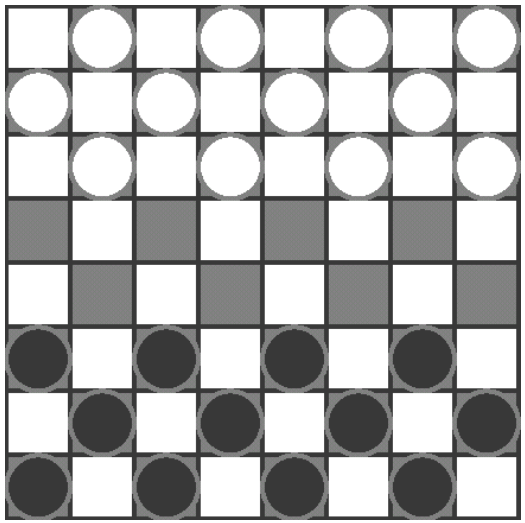
Central problem

- Delayed reward (no immediate classification / feedback available for most actions)

$\Rightarrow$ *Temporal Credit Assignment Problem*

alex@seewald.at / alex.seewald.at

Example: Learning to Play Checkers



Initial formulation of learning task

- Task: play checkers
- Performance measure: percent of games won in world tournament
- Training experience: games played against itself

To be defined

- Exact type of knowledge to be learned (target concept)
- A representation for this target concept
- A learning mechanism

alex@seewald.at / alex.seewald.at

Evaluation Function V

Possible evaluation function V for our checkers problem

- if b is a final board state and is won: V(b) = 100
- if b is a final board state and is lost: V(b) = -100
- if b is a final board state and is drawn: V(b) = 0

- otherwise (if b is not a final board state): V(b)=V(f(b)), where f(b) is the best final board state reachable from b by optimal play $\Rightarrow$ V(f(b)) $\in$ {-100, 0, 100}

Non-operational definition! (i.e. we would need to know optimal play a priori to determine f(b) and thus V(f(b)) – but this is what we are looking for!)

$\Rightarrow$ Learn operational approximation V' of ideal target V

alex@seewald.at / alex.seewald.at

Representation of V'

Characterize board state via set of six features (very simple board representation due to [Simon, 1963])

- x_1 : number of black pieces on the board
- x_2 : number of white pieces on the board
- x_3 : number of black kings on the board
- x_4 : number of white kings on the board
- x_5 : number of black pieces threatened by white
- x_6 : number of white pieces threatened by black

Represent V'(b) as linear function of these 6 features:

$$V'(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Learning problem: Learn values for coefficients $w_{0\text{-}6}$ so that V'(b) gives high values for good/promising board states, and low values for undesirable states.

More complex approximations are possible, e.g. all regression learners, neural networks, rote learning, abitrary numeric function of board features etc..

alex@seewald.at / alex.seewald.at

We still need...

Training examples for V: $[b, V_{train}(b)]$ where b = board state, described by 6 features; $V_{train}(b)$ = evaluation of b (ideally: value of optimal function V(b) – or at least a feasible approximation)

An example: $[(x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0), +100]$

= final board state, won for black because $x_2 = 0$

Function approximation (= learning) algorithm L for learning target function $V': B \rightarrow R$, given finite number of examples $(b, V_{train}(b))$

Problem: Correct evaluation V(b) is known only for final states, but is unknown for all intermediate states!

Solution (simple form of *Q-learning*)**:** Estimate training value $V_{train}(b))$ by using the current estimate (V') of the quality of b's successor state (because that state is one step closer to the final state, whose true evaluation is known):

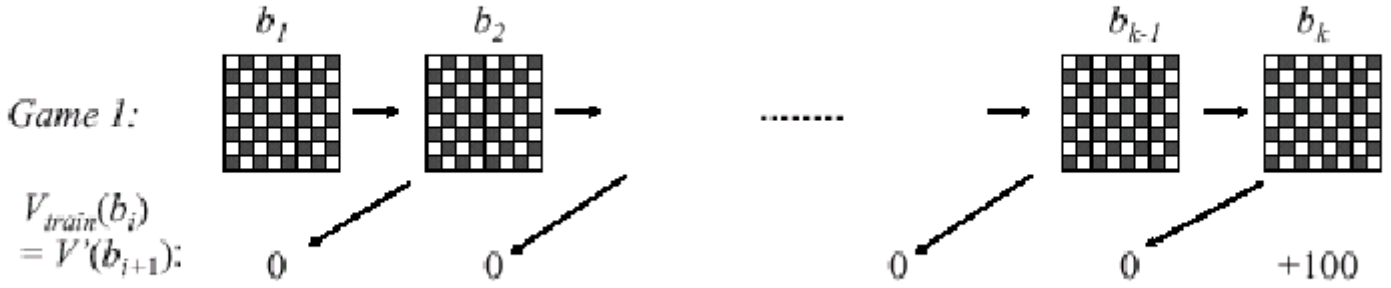
- if b is a final board state and is won/lost/drawn: $V_{train}(b) = +100/-100/0$
- otherwise (i.e., if b is not a final board state): $V_{train}(b) = V'(successor(b))$

Effect: Stepwise back-propagation of information (estimated values) from final game states to intermediate states. Update after each game.

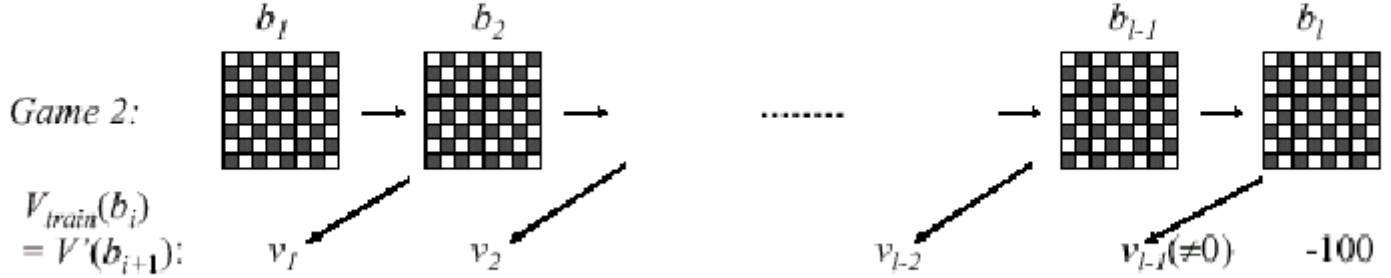
alex@seewald.at / alex.seewald.at

Example: Learning Checkers

1. Initialize V' with $w_0 = w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = 0$
2. Play game 1, using current (initialized) function V' as 'guidance'



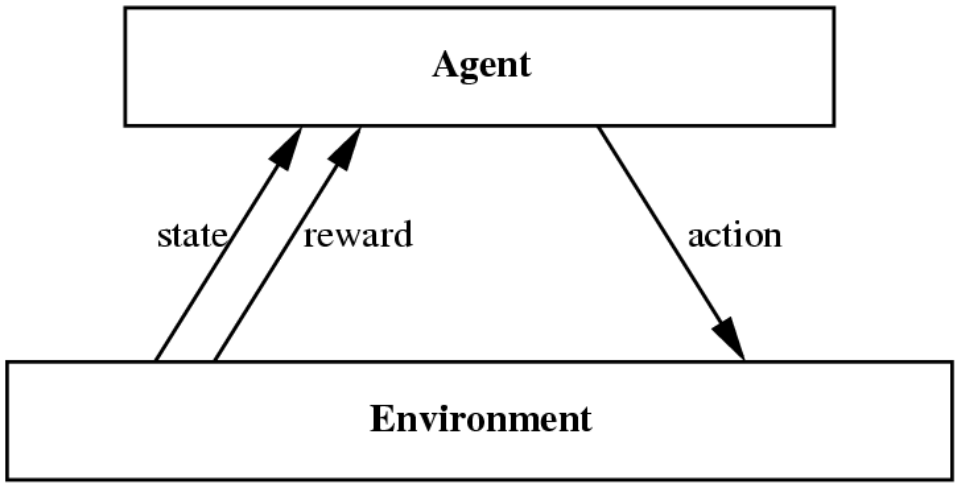
3. Estimate training values $V_{train}(b_i)$ for the states produced in this game (assume: the game was won)
4. Use learning algorithm L to update V' based on new examples $([b_1, V_{train}(b_1)], \ldots [b_k, V_{train}(b_k)])$
5. Play game 2, using newly updated function V' as guidance



6. update V', play, estimate evaluation for new training examples, update V', play, ...

alex@seewald.at / alex.seewald.at

Def.: Reinforcement Learning



$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$

Goal: Learn to choose actions that maximize

$\mathbf{V(s_0)} = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0<\gamma<1$

An **agent** interacting with its **environment**. The agent exists in an environment described by some set of possible **states *S***. It can perform any of a set of possible **actions *A***. Each time it performs an action a_t in some state s_t the agent receives a real-valued **reward r_t** that indicates the immediate value of this state-action transition.

This produces a sequences of states s_i , actions a_i , and immediate rewards r_i as shown on the left. The agent's task is to learn a **control policy π: S$\rightarrow$ A** that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

alex@seewald.at / alex.seewald.at

Refined Definition RL & Solutions

Learn optimal policy π^*: S→A (i.e. policy that maximizes $V_\pi(s)$ for all starting states s – not only for s_0!)

$$\pi^* = \arg\max_\pi V_\pi(s), \forall s$$

How to learn?

- If both $r(s,a)$ and state transition function $\delta(s,a)$ are known: Efficient solution via **dynamic programming** techniques.
- If not, more general approach is needed: **Q-Learning**

Problem: Cannot learn π^*: S→A directly: Training data is just sequence of immediate rewards $r(s_i, a_i)$.

⇒Learn V*: S→R instead, just like in Checkers example!

alex@seewald.at / alex.seewald.at

Q-Learning

Optimal evaluation function V*: $S \rightarrow R$

- Agent should prefer state s_1 over state s_2 whenever $V^*(s_1) > V^*(s_2)$
- Optimal action in state *s* is action *a* that maximizes immediate reward r(s,a) plus V* of immediate successor state (discounted by γ):

  $\pi^*(s) = \mathit{argmax}_a(r(s,a) + \gamma V^*(\delta(s,a)))$ where state-transition function δ(s,a) gives the state resulting from applying action *a* to state *s*

Problem: V*(s) can be usefully applied to make decisions only if reward function r(s,a) and state transition function δ(s,a) are known – which they are not!

Solution: Learn evaluation function for pairs (state,action) instead:

$$Q: S \times A \rightarrow R$$

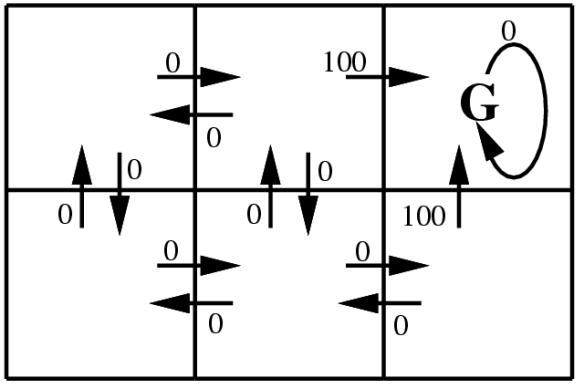
$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a)) \quad \Leftrightarrow \quad \pi^*(s) = \mathit{argmax}_a\, Q(s,a)$$

substitute $V^*(s) = \max_{a'} Q(s,a')$: $Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a),a')$

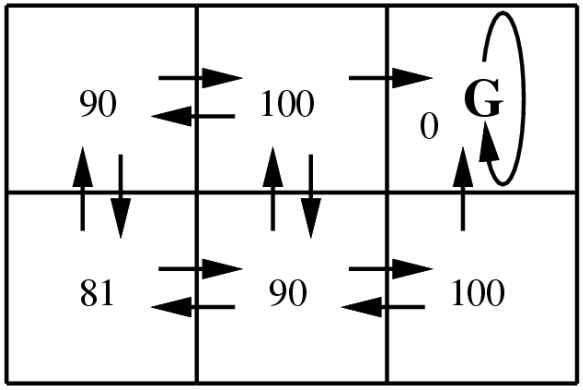
Key idea: Iterative approximation, i.e. using current approximation Q' of successor state to improve estimate of Q' in previous state.

alex@seewald.at / alex.seewald.at

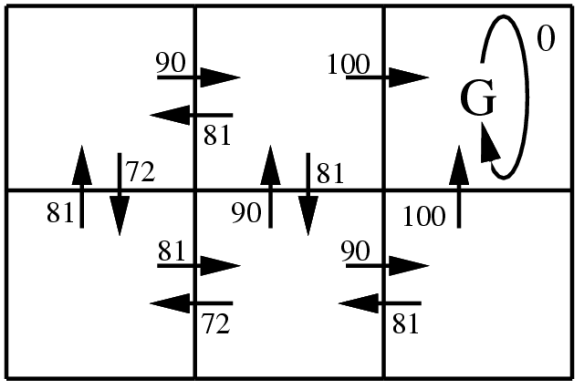
Example World



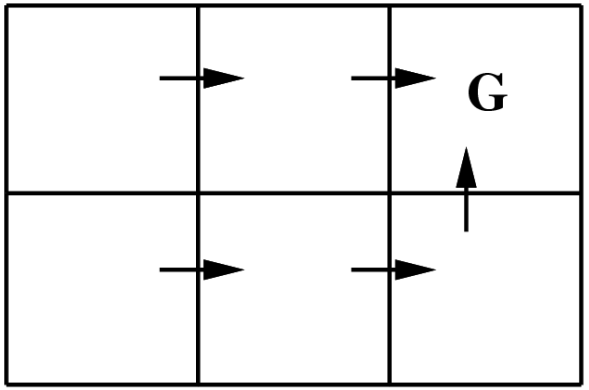
r(s,a) immediate reward values



$V_{\pi^*}(s)$ values for γ=0.9



Q(s,a) values



One optimal policy

alex@seewald.at / alex.seewald.at

Algorithm for learning Q

Representation of Q': large table with separate entry for each <s,a> state-action pair.

For each s, a initialize the table entry $Q'(s,a)$ to zero
Observe the current state s
Do forever:
- Select an action a and execute it
- Receive immediate reward r
- Observe the new state $succ(s)=\delta(s,a)$
- Update the table entry for $Q'(s,a)$ as follows:

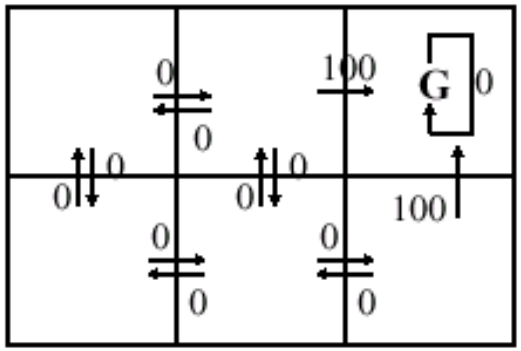
$$Q'(s,a) = r + \gamma \max_{a'} Q'(succ(s),a')$$

- $s=succ(s)$

alex@seewald.at / alex.seewald.at

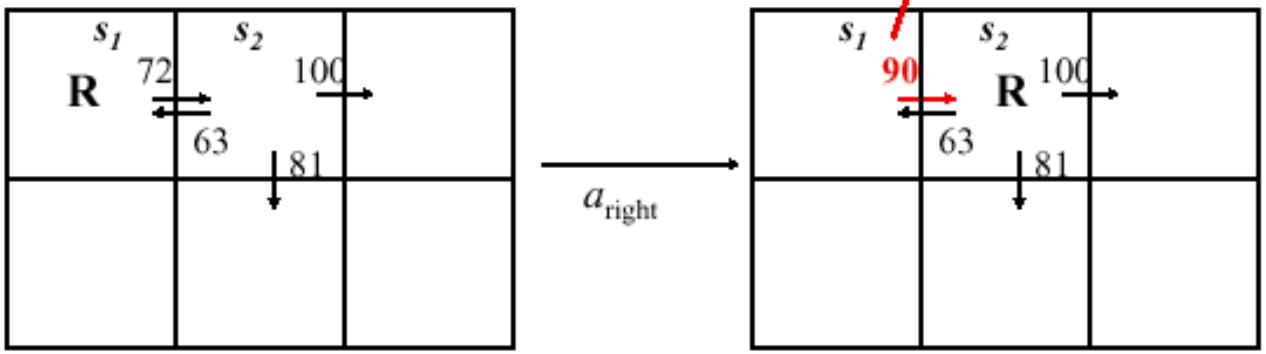
One Step of Q-Learning

Artificial world defined by functions δ and r:



$$Q'(s_1,a_{right}) \leftarrow r + \gamma \max_{a'} Q'(s_2,a')$$
$$= 0 + 0.9 \max\{63, 81, 100\}$$
$$= 90$$

One learning step ($\gamma = 0.9$):

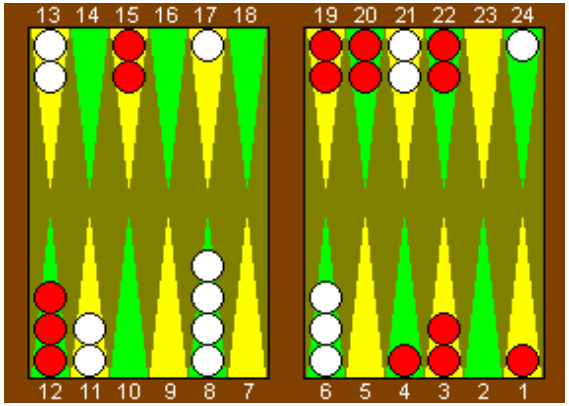


alex@seewald.at / alex.seewald.at

TD-Gammon

Major Success for Reinforcement Learning

- Started with only basic rules of Backgammon
- Learned by playing against itself via Temporal-Difference Learning (=generalization of Q-learning, using a neural network to approximate Q)
- Result: Excellent play at grandmaster level. In some cases, has even changed expert's judgement of best move.



However, later shown to rely on Backgammon' s specific structure - much simpler approaches work equally well. RL does not work so well for most other games, e.g. Go and Othello. Hand-tuning V' works better in some cases, e.g. the Checkers Champion Chinook (Schaeffer, 1997).

alex@seewald.at / alex.seewald.at