

Different Learning Scenarios

Supervised Learning

- A teacher provides the value for the target function for all training examples (labeled examples)
- concept learning, classification, regression

Reinforcement Learning

- The teacher only provides feedback but not example values

Semi-supervised Learning

- Only a subset of the training examples are labeled

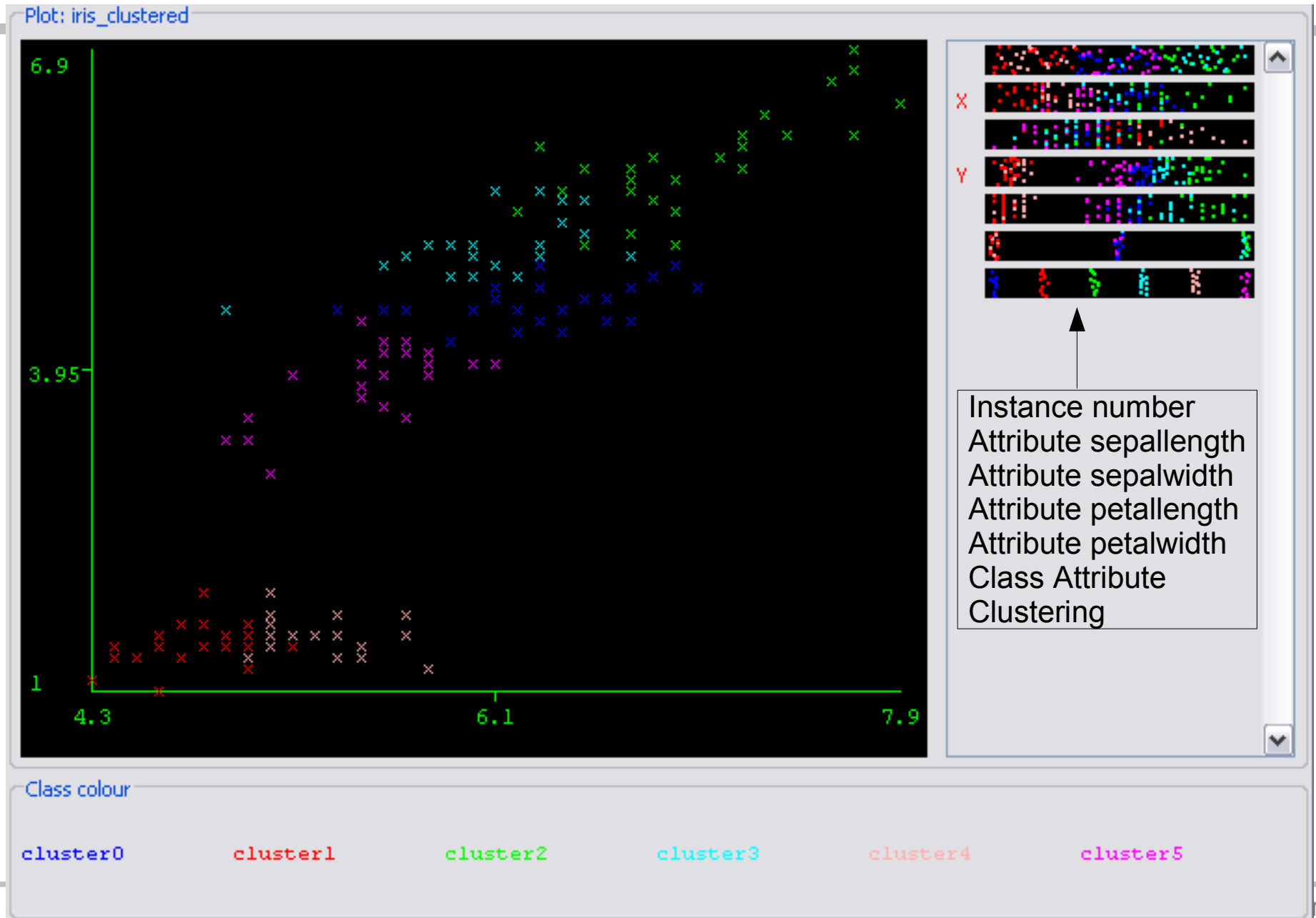
Unsupervised Learning

- There is no information except the training examples
- clustering, subgroup discovery, association rule discovery

Clustering

- Given:
 - a set of examples
 - in some description language (e.g., attribute-value)
 - no labels (-> unsupervised)
- Find:
 - a grouping of the examples into meaningful *clusters*
 - so that we have a high
 - **intra-class similarity**: similarity between objects in same cluster
 - **inter-class dissimilarity**: dissimilarity between objects in different clusters

6 clusters on Iris dataset



Clustering Algorithms

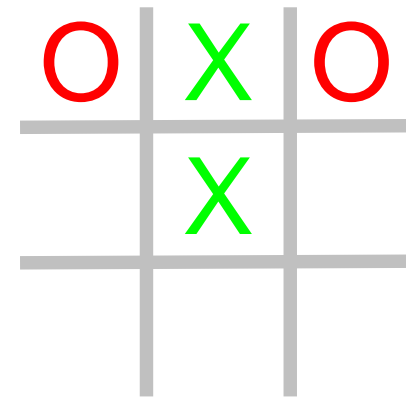
- k-means clustering
 - given a similarity metric (like k-NN algorithms)
 - initialize k cluster centers
 - iteratively assign examples to closest neighbor
 - until procedure converges
- bottom-up hierarchical clustering
 - each example is a cluster
 - iteratively merge clusters, similar to chi-merge
- Cobweb
 - incrementally build up a tree structure
 - each node/cluster can estimate a probability that an example belongs to this cluster
 - examples are sorted into the tree in a top-down way

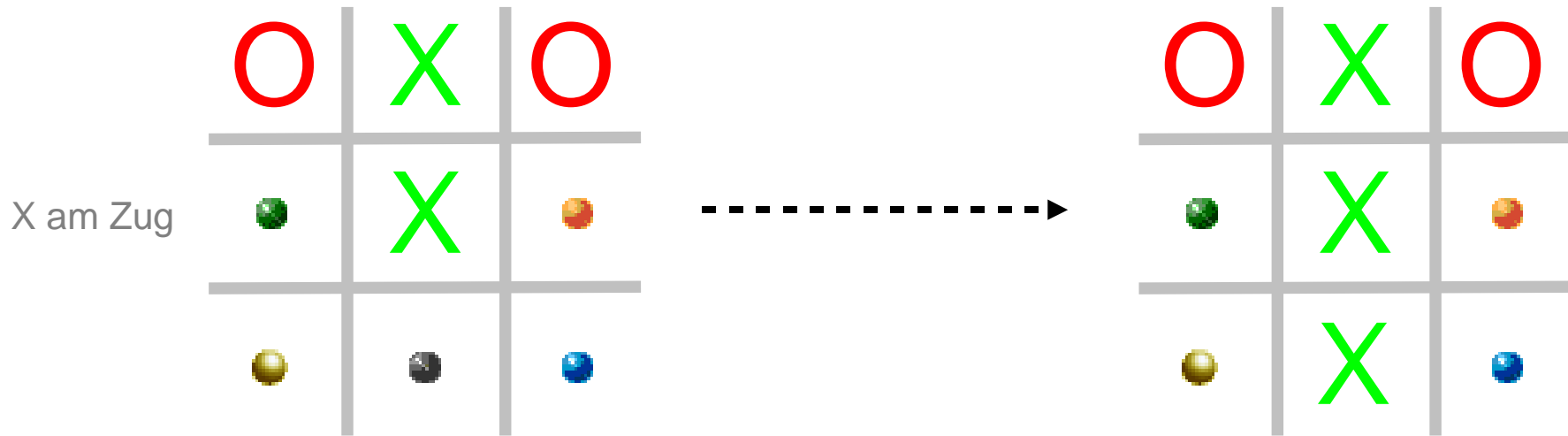
Reinforcement Learning

- Ziel:
 - Lernen von Bewertungsfunktionen durch Feedback (Reinforcement) der Umwelt (z.B. Spiel gewonnen/verloren).
- Anwendungen:
 - **Spiele:**
 - Tic-Tac-Toe: MENACE (Michie 1963)
 - Backgammon: TD-Gammon (Tesauro 1995)
 - Schach: KnightCap (Baxter et al. 2000)
 - **Andere:**
 - Elevator Dispatching
 - Robot Control
 - Job-Shop Scheduling

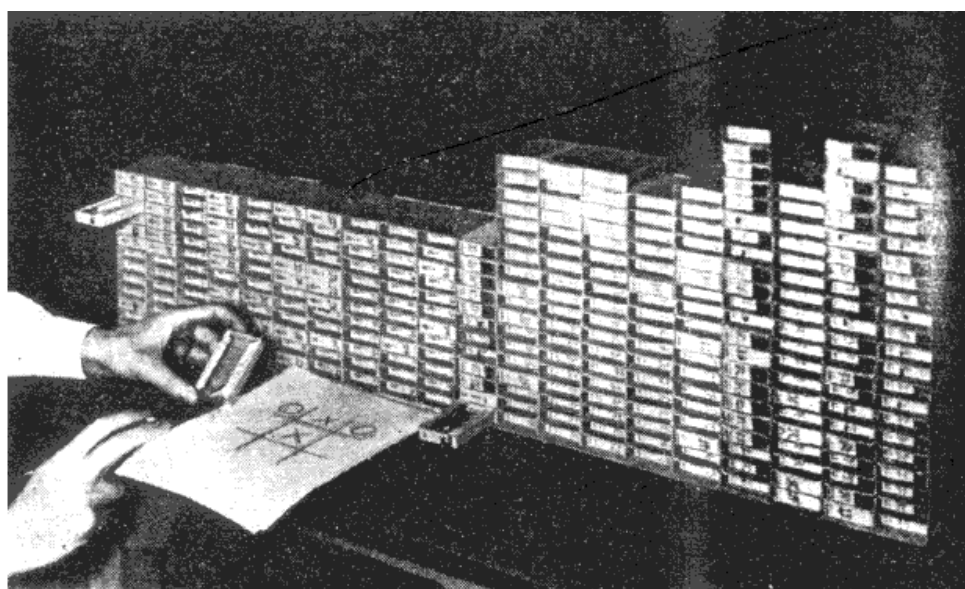
MENACE (Michie, 1963)

- Lernt Tic-Tac-Toe zu spielen
- Hardware:
 - 287 Zündholzschachteln (1 für jede Stellung)
 - Perlen in 9 verschiedenen Farbe (1 Farbe für jedes Feld)
- Spiel-Algorithmus:
 - Wähle Zündholzschachtel, die der Stellung entspricht
 - Ziehe zufällig eine der Perlen
 - Ziehe auf das Feld, das der Farbe der Perle entspricht





Zur Stellung passende Schachtel auswählen



Den der Farbe der gezogenen Kugel entsprechenden Zug ausführen



Eine Kugel aus der Schachtel ziehen



Reinforcement Learning in MENACE

- Initialisierung
 - alle Züge sind gleich wahrscheinlich, i.e., jede Schachtel enthält gleich viele Perlen für alle möglichen Züge
- Lern-Algorithmus:
 - Spiel **verloren** → gezogene Perlen werden einbehalten (*negative reinforcement*)
 - Spiel **gewonnen** → eine Perle der gezogenen Farbe wird in verwendeten Schachteln hinzugefügt (*positive reinforcement*)
 - Spiel remis → Perlen werden zurückgelegt (keine Änderung)
- führt zu
 - Erhöhung der Wahrscheinlichkeit, daß ein erfolgreicher Zug wiederholt wird
 - Senkung der Wahrscheinlichkeit, daß ein nicht erfolgreicher Zug wiederholt wird

Credit Assignment Problem

- Delayed Reward
 - Der Lerner merkt erst am Ende eines Spiels, daß er verloren (oder gewonnen) hat
 - Der Lerner weiß aber nicht, welcher Zug den Verlust (oder Gewinn verursacht hat)
 - oft war der Fehler schon am Anfang des Spiels, und die letzten Züge waren gar nicht schlecht
- Lösung in Reinforcement Learning:
 - Alle Züge der Partie werden belohnt bzw. bestraft (Hinzufügen bzw. Entfernen von Perlen)
 - Durch zahlreiche Spiele konvergiert dieses Verfahren
 - schlechte Züge werden seltener positiv verstärkt werden
 - gute Züge werden öfter positiv verstärkt werden

Reinforcement Learning - Formalization

- Learning Scenario
 - a learning agent
 - S : a set of possible states
 - A : a set of possible actions
 - a state transition function $\delta: S \times A \rightarrow S$
 - a reward function $r: S \times A \rightarrow \mathbb{R}$
- Markov property
 - rewards and state transitions only depend on last state
 - not on how you got into this state
- Environment:
 - the agent repeatedly chooses an action according to some *policy* $\pi: S \rightarrow A$
 - this will put the agent into a new state according to δ
 - in some states the agent receives feedback from the environment (*reinforcement*)

MENACE - Formalization

- Framework
 - states = matchboxes
 - actions = moves/beads
 - policy = prefer actions with higher number of beads
 - reward = game won/ game lost
 - *delayed* reward: we don't know right away whether a move was good or bad

Learning Task

find a policy that maximizes the cumulative reward

- **delayed reward**
 - reward for actions may not come immediately (e.g., game playing)
- therefore maximize **cumulative reward** $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
 - reward from "now" until the end of time
 - immediate rewards are weighted higher (*discount factor* γ)
- training examples
 - generated by interacting with the environment (trial and error)
 - Note:
 - training examples are not supervised (s, a_{max})
 - training examples are of the form (s, a, r)

Value Function

- Each policy can be assigned a value
 - the cumulative expected reward that the agent receives when s/he follows that policy

$$\begin{aligned} V^\pi(s_t) &= \sum_{i=0}^{\infty} \gamma^i r_{t+i} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots = \\ &= r_t + \gamma (r_{t+1} + \gamma r_{t+2} + \dots) = r(s_t, a_t) + \gamma V^\pi(\delta(s_t, a_t)) \end{aligned}$$

$s_{t+1} = \delta(s_t, a_t)$
↓

- Optimal policy
 - the policy with the highest expected value for all states s

$$\pi^* = \arg \max_{\pi} V^\pi(s)$$

- learning an optimal value function $V^*(s)$ yields an optimal policy

$$\pi^*(s) = \arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- BUT: using the optimal value function for action selection requires knowledge of functions r and δ

Q-function

- the Q-function does not evaluate states, but evaluates state-action pairs
 - the Q-function is the cumulative reward for starting in s , applying action a , and, in the resulting state s' , play optimally

$$Q(s, a) := r(s, a) + \gamma V^*(s') \quad [s' = \delta(s, a)]$$

→ the optimal value function is the maximal Q-function over all possible actions in a state $V^*(s) = \max_a Q(s, a)$

- **Bellman equation:** $Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a')$
 - the value of the Q-function for the current state s and an action a is the same as the sum of
 - the reward in the current state s for the chosen action a
 - the (discounted) value of the Q-function for the best action that I can play in the successor state s'

Learning the Q-function

- Basic strategy:
 - start with a some function \hat{Q} , and update it after each step
 - in MENACE: \hat{Q} returns for each box s and each action a the number of beads in the box
- update rule:
 - the Bellman equation will in general not hold for \hat{Q}
i.e., the left side and the right side will be different
→ new value of $\hat{Q}(s, a)$ is a weighted sum of both sides
 - weighted by a **learning rate** α

$$\begin{aligned}\hat{Q}(s, a) &\leftarrow (1 - \alpha)\hat{Q}(s, a) + \alpha(r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')) \\ &\leftarrow \hat{Q}(s, a) + \alpha[r(s, a) + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]\end{aligned}$$

new Q-value for state s and action a

old Q-value for this state/action pair

predicted Q-value for state s' and action a'

Q-learning (Watkins, 1989)

1. initialize all $\hat{Q}(s, a)$ with 0
2. observe current state s
3. loop
 1. select an action a and execute it
 2. receive the immediate reward and observe the new state s'
 3. update the table entry

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha [\underbrace{(r(s, a) + \gamma \max_{a'} \hat{Q}(s', a'))}_{\text{red}} - \underbrace{\hat{Q}(s, a)}_{\text{blue}}]$$

4. $s = s'$

Temporal Difference:

Difference between the estimate of the value of an action **before** and **after** performing the action.

→ Temporal Difference Learning

Miscellaneous

- Weight Decay:

- α decreases over time, e.g. $\alpha = \frac{1}{1 + \text{visits}(s, a)}$

- Convergence:

it can be shown that Q-learning converges

- if every state/action pair is visited infinitely often
 - not very realistic for large state/action spaces
 - but it typically converges in practice under less

- Representation

- in the simplest case, $\hat{Q}(s, a)$ is realized with a look-up table with one entry for each state/action pair
- a better idea would be to have trainable function, so that experience in some part of the space can be generalized
- special training algorithms for, e.g., neural networks exist

SARSA

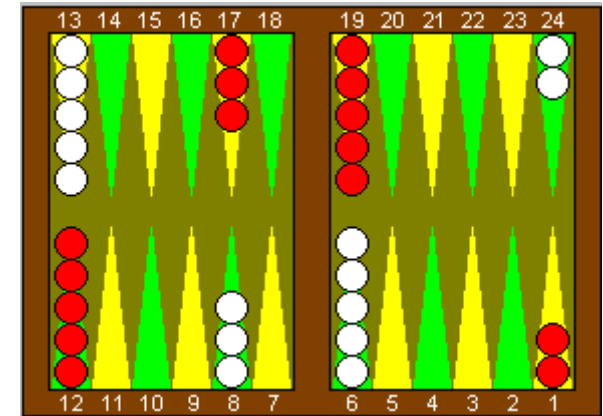
- performs *on-policy updates*
 - update rule assumes action a' is chosen according to current policy

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

- convergence if the policy gradually moves towards a policy that is greedy with respect to the current Q-function
- ε -greedy policies
 - choose random action with probability ε , otherwise greedy
 - trade off exploration vs. exploitation
 - **exploration** is necessary to get a wide variety of state action pairs
 - **exploitation** is necessary for convergence

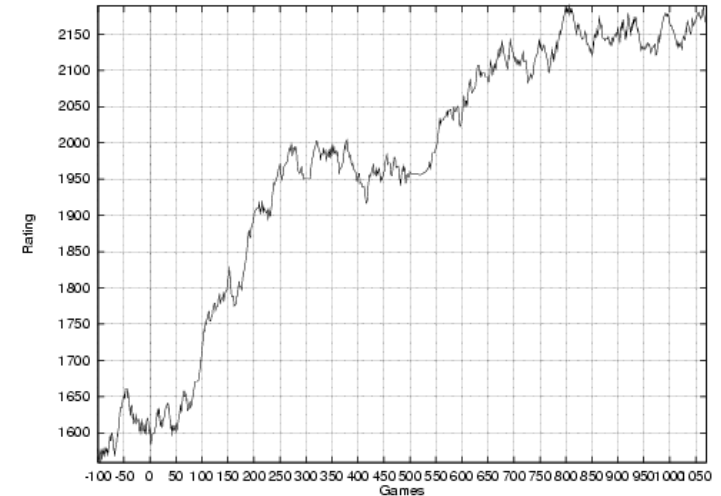
TD-Gammon (Tesauro, 1995)

- weltmeisterliches Backgammon-Programm
 - Entwicklung von Anfänger zu einem weltmeisterlichen Spieler nach 1,500,000 Trainings-Spiele gegen sich selbst (!)
 - Verlor 1998 WM-Kampf über 100 Spiele knapp mit 8 Punkten
 - Führte zu Veränderungen in der Backgammon-Theorie und ist ein beliebter Trainings- und Analyse-Partner der Spitzenspieler
- Verbesserungen gegenüber MENACE:
 - Schnellere Konvergenz durch Temporal-Difference Learning
 - Neutrales Netz statt Schachteln und Perlen erlaubt Generalisierung
 - Verwendung von Stellungsmerkmalen als Features



KnightCap (Baxter et al. 2000)

- Lernt meisterlich Schach zu spielen
 - Verbesserung von 1650 Elo (Anfänger) auf 2150 Elo (guter Club-Spieler) in nur ca. 1000 Spielen am Internet
- Verbesserungen gegenüber TD-Gammon:
 - Integration von TD-learning mit den tiefen Suchen, die für Schach erforderlich sind
 - Training durch Spielen gegen sich selbst -> Training durch Spielen am Internet



Reinforcement Learning Resources

- Book
 - On-line Textbook on Reinforcement learning
 - <http://www.cs.ualberta.ca/~sutton/book/the-book.html>
- Demos
 - Grid world
 - http://thierry.masson.free.fr/IA/en/qlearning_applet.htm
 - Robot learns to crawl
 - <http://iridia.ulb.ac.be/~fvandenb/qlearning/qlearning.html>
 - Pole Balancing Problem
 - <http://www.bovine.net/~jlawson/hmc/pole/sane.html>
- Reinforcement Learning Repository
 - tutorial articles, applications, more demos, etc.
 - <http://www-anw.cs.umass.edu/rlr/>