Maschinelles Lernen B

Lab Session 5
26. November 2004

Michael Pfeiffer
pfeiffer@igi.tugraz.at
<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>10th December</td>
<td>Next lab session</td>
</tr>
<tr>
<td>14th December</td>
<td>Submission for Problem Set 2</td>
</tr>
<tr>
<td>17th December</td>
<td>Presentation hour (PS2) 13.15 - 14.45 (!)</td>
</tr>
<tr>
<td>before Christmas</td>
<td>Problem Set 3 (Literature and Genetic Algorithms)</td>
</tr>
<tr>
<td>End of Semester</td>
<td>Deadline for Bonus Examples 5, 6</td>
</tr>
</tbody>
</table>
Working in Teams

• Team members can use the same source-code
  - submit once with name of all team-members

• What counts is interpretation of results!
• Everybody must write his/her own interpretation and report!
General Hints

- Repeat your experiments
  - good / bad luck should not influence your results (e.g. Exercise 2, 3)
- Try out different parameters
  - if it consistently does not work with one parameter setting try another one
- Report problems with RL Toolbox
Writing your report

• One document per exercise
• Document everything you do
  – parameter values
  – number and length of training runs
  – design choices
• Write clear scientific reports
  – include diagrams to illustrate your results
  – references to results from lecture, Sutton / Barto book or other sources
Exercise 2

- **Multipole – Example**
  - Updated version in new RL Toolbox (better initial parameters)
  - Alternative Version available on MLB-Homepage (does not use RL-Toolbox)

- **Output: trials until success (or failure)**
  - use this to define your performance measure

- **Isolate influence of single parameters**
  - fix „good“ parameters and vary only one
  - average over multiple training runs
On- / Off-policy learning

• Relevant for Problem 3

• Slides by Prof. Andrew G. Barto
  – http://www-all.cs.umass.edu/~barto/
Policy Evaluation (the prediction problem):
for a given policy \( \pi \), compute the state-value function \( V^\pi \)

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

**target**: an estimate of the return

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, modified by M. Pfeiffer
Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences
- Both MC and TD converge (under certain assumptions to be detailed later), but which is faster?
Learning An Action-Value Function

Estimate $Q^\pi$ for the current behavior policy $\pi$.

After every transition from a nonterminal state $s_t$, do this:

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

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

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, modified by M. Pfeiffer
Sarsa: On-Policy TD Control

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $a$, observe $r, s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-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

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, modified by M. Pfeiffer
Windy Gridworld

undiscounted, episodic, reward = –1 until goal
Results of Sarsa on the Windy Gridworld

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, modified by M. Pfeiffer
Q-Learning: Off-Policy TD Control

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., \( \varepsilon \)-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
Cliffwalking

$\varepsilon$-greedy, $\varepsilon = 0.1$
Control: Sarsa(\(\lambda\))

- Save eligibility for state-action pairs instead of just states

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

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

\[
\delta_t = r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)
\]
Sarsa(\(\lambda\)) Algorithm

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
Sarsa($\lambda$) Gridworld Example

- With one trial, the agent has much more information about how to get to the goal
  - not necessarily the best way
- Can considerably accelerate learning

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, modified by M. Pfeiffer
Watkins’ Q(\(\lambda\))

- How can we extend this to Q-learning?

- If you mark every state action pair as eligible, you backup over non-greedy policy
  
  - **Watkins.** Zero out eligibility trace after a non-greedy action. Do max when backing up at first non-greedy choice.
  
  - Other algorithms do not do that

\[
e_t(s, a) = \begin{cases} 
1 + \gamma \lambda e_{t-1}(s, a) & \text{if } s = s_t, a = a_t, Q_{t-1}(s_t, a_t) = \max_a Q_{t-1}(s_t, a) \\
0 & \text{if } Q_{t-1}(s_t, a_t) \neq \max_a Q_{t-1}(s_t, a) \\
\gamma \lambda e_{t-1}(s, a) & \text{otherwise}
\end{cases}
\]

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

\[
\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)
\]

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, modified by M. Pfeiffer
Learning via Self-Play

• Relevant for Example 6*

• Biggest Success Story of RL
  - TD – Gammon

• RL is becoming increasingly used in Computer Games
  - e.g. Black and White
TD Gammon [Tesauro, 1995]

- World-class Backgammon program

- TD-Gammon is learning autonomously
  - Little prior knowledge
  - Most computer games rely heavily on knowledge of AI designer

- Training by playing against itself
  - self-play
• Evaluation Function
  - Represented by Neural Network
  - NN yields probability of winning for every position
  - 2-ply ExpectiMiniMax-Search

• Input – Representation
  - binary representation of current board position
  - Later: complex Backgammon-features
• Training
  - Plays training matches against itself
  - Rewards: +1 für victory, –1 für loss, else 0
  - undiscounted: $\gamma = 1$
  - TD($\lambda$) 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
  - Increasing with number of training matches
  - Up to 1.5 Mio. training matches
• 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!
Does this approach always work?
- Works fine for many other games: Reversi, Checkers, Poker, Settlers of Catan, ...
- Not with Chess and Go
- Large number of training games is an important factor
- Element of chance appears to be important for the success of reinforcement learning
- Why?
  - Exploration is enhanced
  - Evaluation is learned also for non-standard moves
Exercise 6*

- Bonus example: 20 points
  - Deadline: End of semester
- Find a 2-player game with small state set, e.g. Tic-Tac-Toe, Blackjack, Nim
  - suitable for tabular learning (no approximation)
- Send me an E-mail, which game you have chosen
- Use existing open-source implementation or write your own (under Linux)
- Let your program learn with self-play
- Evaluate playing strength