Maschinelles Lernen B

Practical 1
08. October 2004

Michael Pfeiffer
pfeiffer@igi.tugraz.at
Organization

• Lecturer: DI Michael Pfeiffer
  - EMail: pfeiffer@igi.tugraz.at
  - Web: http://www.igi.tugraz.at/pfeiffer
  - Tel.: 873-5814

• Course Homepage:
  - http://www.igi.tugraz.at/lehre/MLB

• Course Newsgroup
  - news.tugraz.at/tu-graz.lv.maschinelles-lernen
Organization

• Place:
  - Practicals always in **i11**
  - **Lecture: Seminarraum I GI**, 1st floor

• Time:
  - usually Fridays, 13.15-14.00
  - sometimes 13.15-14.45
  - **Not every week!**
Registration

• Please register for two courses:
  – via TUG-Online
  – Maschinelles Lernen B VO (708.067)
  – Maschinelles Lernen B UE (708.068)
  – Deadline: 31st October 2004

• No registration for exams required yet
  – only for test at end of semester
What is the Practical about?

• Help you improve your understanding of the material from the lecture
  - Examples and Demos
  - Software for RL and GA
  - Discussion of your Questions

• Determining your Grade
  - 3 Exercise Sheets
  - Presentation Hours
  - 1 Test
How to get a Grade

- 3 Exercise Sheets:
  - Theory (RL)
  - Practical Solutions (RL)
  - Literature and Practical Solutions (RL, GA)

- Presentation Hours
  - You will present your solutions on blackboard
  - Solutions to be handed in on Tuesday (lecture) before presentation
  - I will inspect your solutions, but not correct all of them
  - Between presentation hours: no course on Friday
    - only occasional question hours or tutorials (announced on homepage and in newsgroup)

- Multiple Choice Test
  - end of semester
How to get a Grade

• You get points for every solution you submit
  – Points for exercise sheets and tests are added ⇒ grade
  – You do not lose points for minor errors
  – We will also discuss some wrong solutions

• Working together
  – You can form teams of max. 3 students (not for literature task)
  – Mention all team members on your submissions!
  – Discussing is OK, copying not!

• Copying from other students:
  – You lose all the points for the exercise sheet!
  – During presentation hours you must be able to explain every solution you have submitted!
1st Exercise Sheet: Theory

- Theoretical questions and proofs about Reinforcement Learning
  - Topics will be covered in lecture in the next weeks
- Issued: probably next week
- Presentation hour: 12th November (13.00-15.00)

* - Tasks:
  - every exercise sheet contains several *-tasks
  - more complex bonus exercises for those interested
  - earn you points like usual exercises!
Further Exercise Sheets

• Nr. 2: Practical Exercises:
  - Programming examples using RL Toolbox
  - Issued: 3rd November
  - Tutorial for RL Toolbox: 5th November
  - Presentation: before Christmas

• Nr. 3: Literature:
  - Summarize two research papers about RL or GA + programming exercise
  - papers are put on our homepage during the course
  - Issued: before Christmas
  - Presentation: end of semester (practical exercises only)
Reinforcement Learning

- Short summary of the first lecture

- Slides by Prof. Andrew G. Barto
  - [http://www-all.cs.umass.edu/~barto/](http://www-all.cs.umass.edu/~barto/)
Reinforcement Learning

Training Info = evaluations ("rewards" / "penalties")

Objective: get as much reward as possible
Complete Agent

- Temporally situated
- Continual learning and planning
- Object is to **affect** the environment
- Environment is stochastic and uncertain
The $n$-Armed Bandit Problem

- Choose repeatedly from one of $n$ actions; each choice is called a **play**
- After each play $a_t$, you get a reward $r_t$, where $E[r_t | a_t] = Q^*(a_t)$
  
  These are unknown **action values**
  Distribution of $r_t$ depends only on $a_t$

- Objective is to maximize the reward in the long term, e.g., over 1000 plays

To solve the $n$-armed bandit problem, you must **explore** a variety of actions and the **exploit** the best of them
Action-Value Methods

• Methods that adapt action-value estimates and nothing else, e.g.: suppose by the $t$-th play, action $a$ had been chosen $k_a$ times, producing rewards $r_1, r_2, \ldots, r_{k_a}$, then

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a} \quad \text{“sample average”}$$

• $\lim_{{k_a \to \infty}} Q_t(a) = Q^*(a)$
The Exploration/Exploitation Dilemma

• Suppose you form estimates

\[ Q_t(a) \approx Q^*(a) \]  

action value estimates

• The **greedy** action at \( t \) is

\[ a^*_t = \arg\max_a Q_t(a) \]

\[ a_t = a^*_t \Rightarrow \text{exploitation} \]

\[ a_t \neq a^*_t \Rightarrow \text{exploration} \]

• You can’t exploit all the time; you can’t explore all the time

• You can never stop exploring; but you should always reduce exploring
ε-Greedy Action Selection

• Greedy action selection:
  \[ a_t = a_t^* = \arg \max_a Q_t(a) \]

• ε-Greedy:
  \[ a_t = \begin{cases} 
  a_t^* \text{ with probability } 1 - \varepsilon \\
  \text{random action with probability } \varepsilon 
  \end{cases} \]

... the simplest way to try to balance exploration and exploitation
10-Armed Testbed

• \( n = 10 \) possible actions
• Each \( Q^*(a) \) is chosen randomly from a normal distribution: \( \eta(0,1) \)
• each \( r_t \) is also normal: \( \eta(Q^*(a_t),1) \)
• 1000 plays
• repeat the whole thing 2000 times and average the results
ε-Greedy Methods on the 10-Armed Testbed

- Average reward vs. number of plays
- Percentage of optimal action vs. number of plays

ε = 0.1
ε = 0.01
ε = 0 (greedy)
Questions

• Which method performs best if the reward variance is
  - higher (e.g. 10)
  - zero?

• What if action values are non-stationary?

• Which method will perform best in the long run?
  - cumulative reward and optimal action selection
Optimistic Initial Values

• All methods so far depend on $Q_0(a)$, i.e., they are biased.
• Suppose instead we initialize the action values *optimistically*, i.e., on the 10-armed testbed, use $Q_0(a) = 5$ for all $a$.
Optimistic Initial Values

• Another method for exploration

• What are the differences to $\varepsilon$-greedy?
  - duration of exploration

• How does this method work for non-stationary problems?
  - adaptation to changing tasks
Softmax Action Selection

- Softmax action selection methods grade action probs. by estimated values.
- The most common softmax uses a Gibbs, or Boltzmann, distribution:

\[
\text{Choose action } a \text{ on play } t \text{ with probability }
\]
\[
\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}},
\]

where \( \tau \) is the “computational temperature”
Boltzmann Distribution

- No action has probability 0
- How would you select an action from this?
An Extended Example: Tic-Tac-Toe

Assume an imperfect opponent:
—he/she sometimes makes mistakes

} x’s move
} o’s move
} x’s move
} o’s move
} x’s move
MiniMax - Approach

• Assume perfect player
  – perfect strategy exists: never lose

• Search until leaf of game tree
  – recursively back-up minimax - values (e.g. +1 / 0 / -1 for winning / draw / losing)
  – MIN-player: \( \rightarrow \) successor state with minimal value
  – MAX-player: \( \rightarrow \) successor with maximal value

• Does not yield optimal performance against fixed (fallible) player!
An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

<table>
<thead>
<tr>
<th>State</th>
<th>V(s) – estimated probability of winning</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>.5</td>
</tr>
<tr>
<td>☐</td>
<td>.5</td>
</tr>
<tr>
<td></td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>?</td>
</tr>
<tr>
<td>☐☐☐☐☐</td>
<td>1 win</td>
</tr>
<tr>
<td>☐☐☐☐☐</td>
<td></td>
</tr>
<tr>
<td>☐☐☐☐☐</td>
<td>0 loss</td>
</tr>
<tr>
<td>☐☐☐☐☐</td>
<td></td>
</tr>
<tr>
<td>☐☐☐☐☐</td>
<td>0 draw</td>
</tr>
</tbody>
</table>

2. Now play lots of games. To pick our moves, look ahead one step:

To pick our moves, look ahead one step:

Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a greedy move.

But 10% of the time pick a move at random; an exploratory move.
RL Learning Rule for Tic-Tac-Toe

• our move
• opponent's move
• our move
• opponent's move
• our move

starting position

• “Exploratory” move

s – the state before our greedy move
s’ – the state after our greedy move

We increment each \( V(s) \) toward \( V(s') \) – a backup:

\[
V(s) \leftarrow V(s) + \alpha [V(s') - V(s)]
\]

a small positive fraction, e.g., \( \alpha = .1 \)
the step-size parameter
Questions

• How could we improve the learning speed?

• How does the learned strategy perform against different opponents (e.g. MiniMax)?

• How could we improve the performance against different types of opponents?
• How do we learn to play against arbitrary opponents?
• What problems may arise then?

• For larger games (e.g. chess, Go, ...) it is impossible to search an entire game tree with MiniMax
  − we need to cut-off search and evaluate positions
  − no training examples for evaluations

• RL can be used to learn evaluation functions
Learning Action Sequences

• Until now:
  - n-Armed Bandit
    • n actions, only one state
  - Tic-Tac-Toe
    • only evaluate states

• Usual problems (e.g. Gridworld):  
  - multiple actions and multiple states  
  - sequential decision making:
    • maximize the sum of rewards that follow an action
    • rewards can be delayed
Next week

• Lecture:
  - more about learning from delayed reward
  - definition of the reinforcement learning problem
  - Exercise Sheet 1

• No Practical!