Reinforcement Learning

Viktor Seifert

Seminar: Knowledge Engineering und Lernen in Spielen

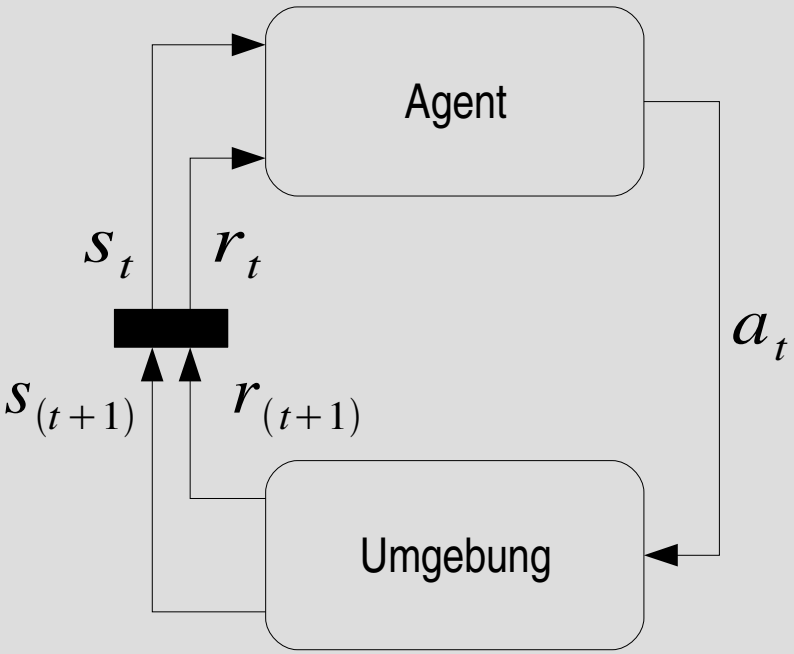
SS06

Prof. Johannes Fürnkranz

Übersicht

- 1. Definition
- 2. Allgemeiner Lösungsansatz
- 3. Temporal-Difference Learning
- 4. Funktionsapproximation
- 5. Vor- & Nachteile (pers. Einschätzung)
- 6. Beispiele

1. Definition



- Agent:
  - „die KI“
  - wählt nächste Aktion
- Umgebung:
  - gibt Zustand & Reward an Agent

1. Definition - Begriffe

- Agent:
  - entscheidet über nächste Aktion anhand des aktuellen Zustandes
  - gehört dazu: alles was Agent *beliebig* ändern kann
- Umgebung:
  - alles „außerhalb“ des Agenten
    - auch Wahrnehmung des Agenten !
  - bestimmt Rewards & Zustände
  - bestimmt mögliche Aktionen

1. Definition - Begriffe

- Zustand:
  - ist genaue Konfiguration der Umgebung
  - Agent darf/muss Aktion wählen
  - Nachfolgezustand ist nächste Entscheidungsmöglichkeit des Agenten
    - z. B. im Schach:
      - Agent ist schwarzer Spieler
      - Zustände sind am Anfang des Zuges des schwarzen Spielers
      - d.h.: Sicht *eines* Spielers
  - bei Echtzeit: Zustände müssen nicht gleichen zeitlichen Abstand haben

1. Definition - Begriffe

- Reward:
  - Belohnung oder Strafe für Handlungen des Agenten
  - kann und wird verzögert auftreten (delayed Reward)
  - für Agent auch nicht ersichtlich: welche Aktion hat Reward ausgelöst
  - z. B.: im Schach:
    - +1 für Sieg
    - 0 für Remis
    - -1 für Niederlage

1. Definition - Begriffe

- Return:
  - ist Reward aufsummiert über die Zeit
  - Agent versucht diesen zu maximieren
- Aufgabentypen:
  - episodisch:
    - Aufgabe ist in Episoden unterteilt, mit Anfangs- & Endzuständen
    - Return = Summe der Rewards
  - nicht episodisch:
    - Aufgabe läuft fortwährend weiter
    - Rewards werden mit Faktor < 1 discountet
    - d.h. Rewards die weit in der Zukunft liegen werden abgwertet

1. Definition

- Aufgabe des Agenten:
  - Kontrolle der Umgebung & Maximieren des Returns
  - soll sich durch Erfahrung verbessern
- Vergleich zu anderen ML Techniken:
  - kein Supervised Learning
  - d.h.: keine Aussage darüber welche Aktion die *richtige* gewesen wäre
  - sondern Performance Feedback mit (verzörten) Rewards
  - d.h.: keine Aussage darüber *welche* Aktion gut oder schlecht gewesen ist
  - Agent hat direkten Einfluss auf Umgebung

1. Definition

- Grundsätzlich: jede Methode die die Aufgabe löst ist Reinforcement Learning
- z.B.: Evolutionsstragien, ...
- aber: sie nutzen die Struktur der Umgebung nicht direkt aus (Rewards)
- weiteres in nächsten Abschnitt

1. Definition - Zusammenfassung

- Agent, Aktionen, Umgebung
- Zustände & Rewards
- Agent versucht Return zu maximieren
- Lernt durch Erfahrung

2. Allgemeiner Lösungsansatz Bestandteile

- Policy
  - $\pi(s, a)$
  - Wahrscheinlichkeit dass Agent in Zustand s Aktion a wählt
  - oben: theoretische Sicht
  - praktische Beispiele:
    - e-greedy: mit Warscheinlichkeit e beliebige Aktion, sonst beste
    - soft-max: wähle zufällige Aktion, mit höherer Wahrscheinlichkeit für bessere Aktionen
- Bewertungsfunktion
  - gibt erwarteten Return an
  - d.h.: Return verrechnet mit der Wahrscheinlichkeit ihn zu bekommen

2. Allgemeiner Lösungsansatz Bewertungsfunktion

- Arten:
  - Zustandswertefunktion: $V^{\pi}(s)$
    - bewertet Zustand *vor* eigener Aktion unter einer Policy π
    - man sollte Nachfolgezustand sicher voraussagen können (Modell der Umgebung)
  - Zustandswertefunktion für Afterstates:
    - wenn nächster Zustand nicht eindeutig
    - bewertet Zustand *nach* eigener Aktion (-> Afterstate)
    - kein Modell benötigt, aber Vorhersage über Aftersate
  - Aktionswertefunktion: $Q^{\pi}(s, a)$
    - bewertet Aktion a in Zustand s
    - benötigt keine Aussagen über Zukunft

2. Allgemeiner Lösungsansatz

- 1. Policy Evaluation:
  - durch Anwendung der Policy wird die Wertefunktion der Policy bestimmt
  - geschieht durch beobachten von Rewards
- 2. Policy Improvement:
  - verändere Policy so dass bessere Aktionen wahrscheinlicher werden
  - geschieht bei den obigen Beispielen automatisch
- 3. Wiederhole 1 & 2 bis optimale Funktion und Policy vorliegen

2. Allgemeiner Lösungsansatz
Aufgaben der Policy

- Exploitation:
  - nutze aktuelles Wissen über die Umwelt um möglichst gute Aktionen zu wählen
  - je besser das aktuelle Wissen desto besser die Leistung des Agenten
  - bei schlechtem Wissen keine Verbesserung der Leistung durch Erfahrung
- Exploration:
  - wähle irgendeine, nicht beste, Aktion um herauszufinden ob diese besser ist als die aktuell beste
  - mehr Exploration -> Performance verschlechtert sich
  - aber auch Möglichkeit neue gute Aktionen zu finden

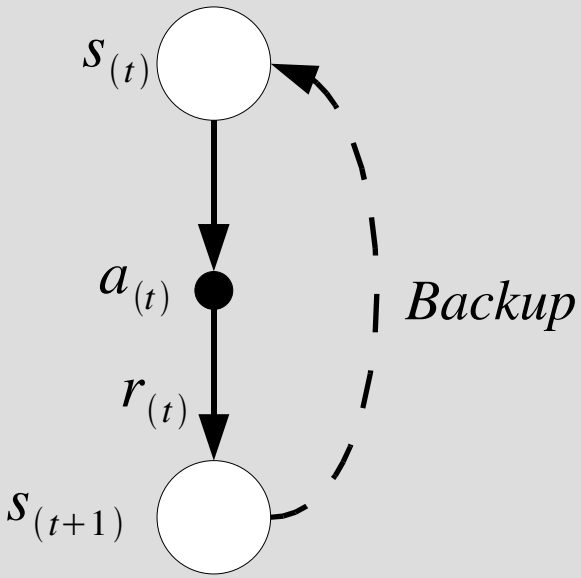
2. Allgemeiner Lösungsansatz

- Dynamic Programming:
  - bestimme Wert eines Zustandes durch „sweep“ durch den *gesamten* Zustandsraum
  - ineffizient bei großen Zustandsmengen
    - z.B.: Backgammon hat ca. 10^22 Zustände
- Monte Carlo Ansatz:
  - beobachte ganze Episode
  - benutze diese um Funktion zu verbessern
  - kein on-line-Lernen möglich

2. Allgemeiner Lösungsansatz Zusammenfassung

- Policy, Bewertungsfunktion
- Policy Evaluation, Policy Improvement
- Exploitation & Exploration, trade-off

3. Temporal-Difference Learning



- 1. Wähle Aktion
- 2. Beobachte nächsten Reward & nächsten Zustand
- 3. Passe den Wert des voherigen Zustandes anhand des Rewards und des Wertes des neuen Zustandes an

$$V(s_{(t)}) <- V(s_{(t)}) + \alpha[r_{(t)} + \gamma V(s_{(t+1)}) - V(s_{(t)})]$$

3. Temporal-Difference Learning

- vorherige Folie: TD(0)-Algorithmus
- Backup eines Zustandes mit Information die erst nach einiger Zeit verfügbar ist -> Temporal-Difference
- Wert des Zustandes wird anhand einer anderen Schätzung geschätzt -> Bootstrapping
- Backups können online erfolgen

3. Temporal-Difference Learning

- Variationsmöglichkeiten:
- 1-step-backups -> n-step-backups (TD(lambda))
- on-policy & off-policy
- Zustände & Aktionen

4. Funktionsapproximation

- bisher implizit angenommen:
- Werte der Funktion sind in einer Tabelle gespeichert
- also pro Zustand 1 Eintrag in der Tabelle
- für grössere Probleme ungeeignet
  - z.B.: Backgammon 10^22 Zustände
- keine Generalisierung
  - nur bereits besuchte Zustände können richtig beurteilt werden

4. Funktionsapproximation

- daher: Funktionsapproximation
- Funktion nicht mehr tabellarisch gespeichert sondern durch andere Funktion approximiert (Beispiele folgen)
- Speicherverbrauch sinkt dramatisch
- Generalisierung auf nicht besuchte Zustände möglich
- aber: nicht alle Zustände können gleich gut beurteilt werden, wegen geringerer Parameterzahl
- deswegen Fokussierung auf häufig besuchte Zustände

4. Funktionsapproximation

- Beispiele:
- lineare Funktion
- neuronales Netz
  - Backpropagation von Fehlern $r_{(t+1)}+V(s_{(t+1)})-V(s_{(t)})$
- Tabelle aber Zustände nicht mit allen Merkmalen gespeichert

3. & 4. Zusammenfassung

- Backups von Zustandswerten anhand von Beobachtungen
- in vielen Fällen zu viele Zustände um explizit zu speichern
- deswegen: Funktionsapproximation
- Speichereduzierung & Generalisierung

5. Vor- & Nachteile von RL

- Vorteile:
  - Anpassungsfähigkeit
    - verschiedene Gegenspieler
    - veränderte Regeln
  - automatisiertes Lernen bzw. Lösungssuche
    - Ziel bekannt, aber nicht dessen Lösung
    - keine menschlichen Resourcen vorhanden, aber Rechner
  - kann aus Erfahrung anderer lernen (siehe Beispiel 2)
  - bei vielen Spielen sind Rewards relativ leicht zu erkennen

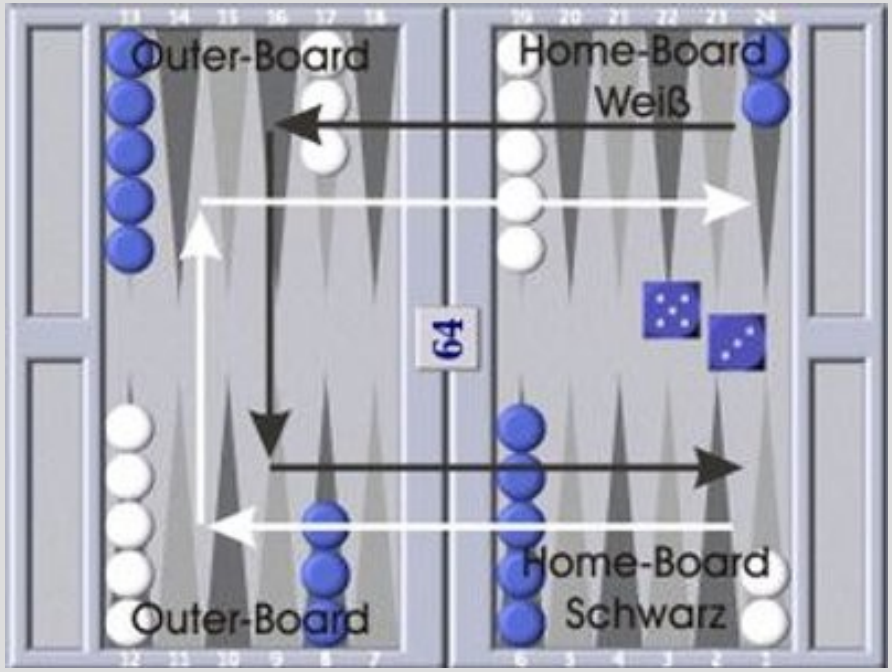
5. Vor- & Nachteile von RL

- Nachteile:
  - muss erst lernen:
    - braucht Zeit und Rechenkapazität
  - Erfolg ist nicht garantiert
    - auch nicht dessen Qualität
  - recht komplex
  - viele Parameter
    - kann Experimentieren notwendig machen
  - nicht immer ersichtlich was Rerwards sind

Reinforcement Learning

Fragen?

Beispiel 1
TD-Gammon



- Backgammon: sehr beliebtes Brettspiel
- 2 Spieler: Schwarz & Weiß
- Würfelwurf bestimmt über Zugmöglichkeiten
- Zugrichtung in Grafik
- Spieler gewinnt wenn alle seine Steine vom Brett entfernt wurden

Beispiel 1
TD-Gammon

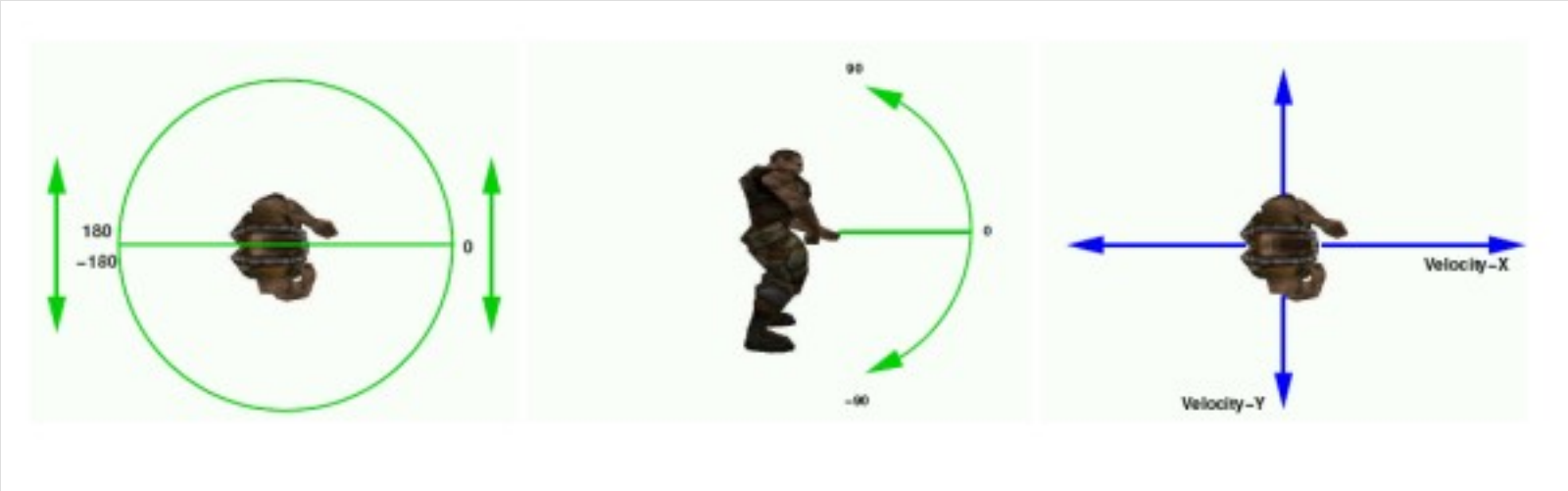
- TD-Gammon: Programm von Gerry Tesauro
- TD(lambda) Algorithmus
- Funktionsapproximation durch multi-layer neural network & backpropagation von TD-errors
- generierte Erfahrung durch Spielen gegen sich selbst
- Version 0.0 enthielt kein Backgammon Wissen
- war gleichauf mit allen vorherigen Programmen
- in der Version 1.0 wurde das Netz mit Backgammon Wissen ergänzt
- war für menschliche Spieler bereits herausfordernd

Beispiel 1
TD-Gammon

- weitere Versionen enthielten eine two-ply Suche und mehr hidden-units
- V 3.0 enthält eine three-ply Suche
- spielt auf Weltmeister Niveau
- sein Erfolg und weitere Analysen veränderten die Weise in der menschliche Spieler Eröffnungen spielen

Beispiel 2
Agent für Quake2 Deathmatches

- Quake2
- Agent sollte lernen Deathmatches zu spielen
- Erfahrung aus aufgezeichneten Matches von menschlichen Spielern (Demos)
- genaues Ziel: Moving & Aiming
- Zustand:
  - 3d-Postion der Figur auf der Karte
  - Abstand zum nächsten Gegner
  - horizontaler & vertikaler Winkel zu diesem Gegner



- Aktionen
  - Änderung des Sichtfeldes
    - YAW -180 bis +180 Grad (linkes Bild)
    - PITCH -90 bis +90 Grad (mittleres Bild)
  - Änderung der Bewegungsgeschwindigkeit
    - in x-Richtung von -400 bis +400
    - in y-Richtung von -200 bis +200

Beispiel 2
Agent für Quake2 Deathmatches

- für Sicht und Bewegung jeweils 1 neurales Netz
- Lernziele & Erfolge
  - Learning Efficient Paths
    - abgeschaut von menschlichen Spielern
    - Bot lernte einen möglichst guten (kreisförmigen) Weg um möglichst effizient Gegenstände einzusammeln
  - Learning to Run Crossed Paths
    - wenn die Pfade aus erstem Ziel sich kreuzen ist das Lernen fehlerhaft weil vorherige Zustände betrachet werden
    - Verbesserung durch beachten der 2 vorherigen Zustände

Beispiel 2
Agent für Quake2 Deathmatches

- Lernziele & Erfolge forts.
  - Learning to Switch between Movement and Aiming Behaviors:
    - vorherige Ansätze immitieren nur das Laufen
    - aber nicht das Umschalten von 2 versch. Verhaltensweisen, in diesem Fall: Moving & Aiming
    - entsteht wenn einem ein Gegner über den Weg läuft
    - die 2 neuralen Netze reichen nicht mehr
    - gute Erfolge mit Clustern der Daten mit Self-Organized-Maps

Reinforcement Learning

Vielen Dank für die Aufmerksamkeit

Noch Fragen?

Quellen

- http://www2.informatik.hu-berlin.de/Forschung_Lehre/wm/mldm2004/ReinforcementLernen.pdf
- Richard S. Sutton & Andrew G. Barto: Reinforcement Learning, An Introduction
- http://de.wikipedia.org/wiki/Backgammon
- Learning Human-like Opponent Behavior for Interactive Computer Games; Christian Bauckhage, Christian Thurau, and Gerhard Sagerer