Lernfähige autonome Roboter

Christian Kungel, Kornelius Nägele, Jan Tammen

Seminar Robotik WS 06/07
Fakultät Informatik,
HTWG Konstanz

14. Dezember 2006

Gliederung

Das RoboCup-Projekt

- internationales Forschungs- und Bildungsprojekt
- Ziel: Förderung der Erforschung von KI und mobilen autonomen Robotern
- Veranstaltung von Wettbewerben

By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.

Das RoboCup-Projekt

- internationales Forschungs- und Bildungsprojekt
- Ziel: Förderung der Erforschung von KI und mobilen autonomen Robotern
- Veranstaltung von Wettbewerben

By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.

Das RoboCup-Projekt

- internationales Forschungs- und Bildungsprojekt
- Ziel: Förderung der Erforschung von KI und mobilen autonomen Robotern
- Veranstaltung von Wettbewerben

By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.

Das RoboCup-Projekt

- internationales Forschungs- und Bildungsprojekt
- Ziel: Förderung der Erforschung von KI und mobilen autonomen Robotern
- Veranstaltung von Wettbewerben

By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- SmallSize
- MiddleSize
- FourLegged
- Humanoid

- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid

- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid

- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid

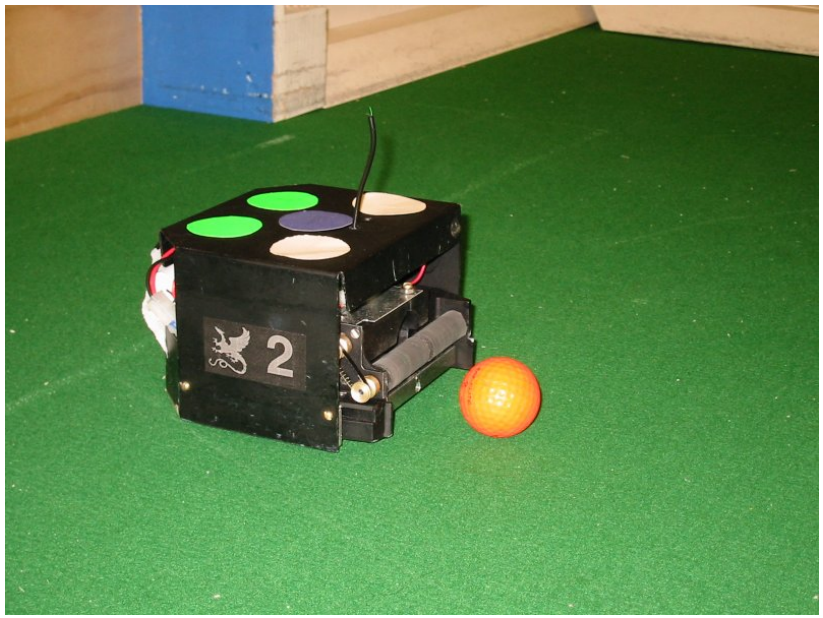
- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid



- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid

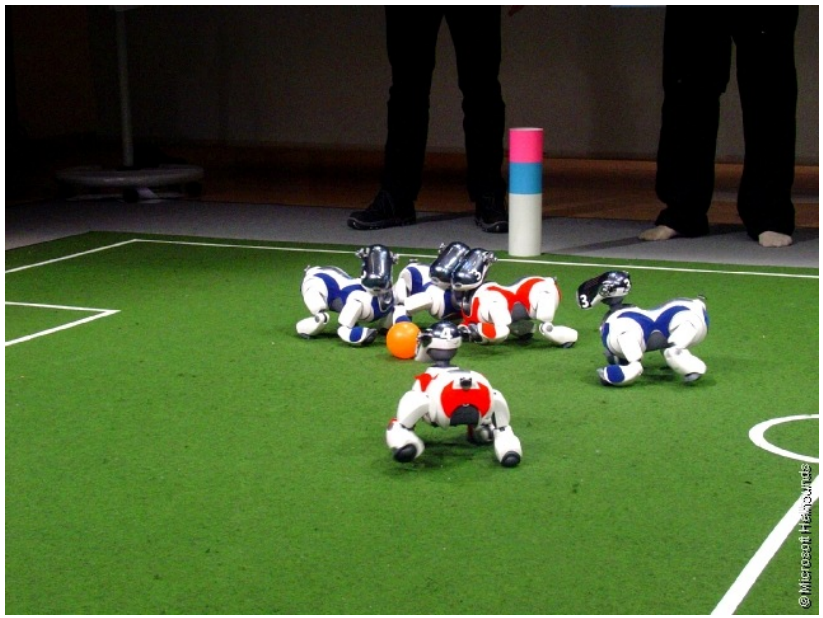
- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid



- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid

- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid

- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid

- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Der RoboCup

Internationale Wettbewerbs- und Konferenzveranstaltung

Ligen

- Simulation (2D und 3D)
- Small-Size
- Middle-Size
- Four-Legged
- Humanoid

- RoboCupRescue: Rescue Robots und Simulation
- RoboCupJunior: Soccer, Rescue, Dance

Gliederung

Simulationsliga: „Mindstormers“

Funktionsweise der Simulation

Client-Server-Architektur
Server simuliert Weltmodell und liefert Sensordaten an Clients
11 Clients (eigener Thread) pro Team
pro Zyklus (6000 à 100 ms) Ausführen eines Kommandos

Herausforderungen:

- verrauschte Sensorinformationen, nur indirekte Kommunikation möglich
- kooperatives Spiel
- begrenzte Kondition

Simulationsliga: „Mindstormers“

Funktionsweise der Simulation

Client-Server-Architektur
Server simuliert Weltmodell und liefert Sensordaten an Clients
11 Clients (eigener Thread) pro Team
pro Zyklus (6000 à 100 ms) Ausführen eines Kommandos

Herausforderungen:

- verrauschte Sensorinformationen, nur indirekte Kommunikation möglich
- kooperatives Spiel
- begrenzte Kondition

Simulationsliga: „Mindstormers“

Funktionsweise der Simulation

Client-Server-Architektur
Server simuliert Weltmodell und liefert Sensordaten an Clients
11 Clients (eigener Thread) pro Team
pro Zyklus (6000 à 100 ms) Ausführen eines Kommandos

Herausforderungen:

- verrauschte Sensorinformationen, nur indirekte Kommunikation möglich
- kooperatives Spiel
- begrenzte Kondition

Simulationsliga: „Mindstormers“

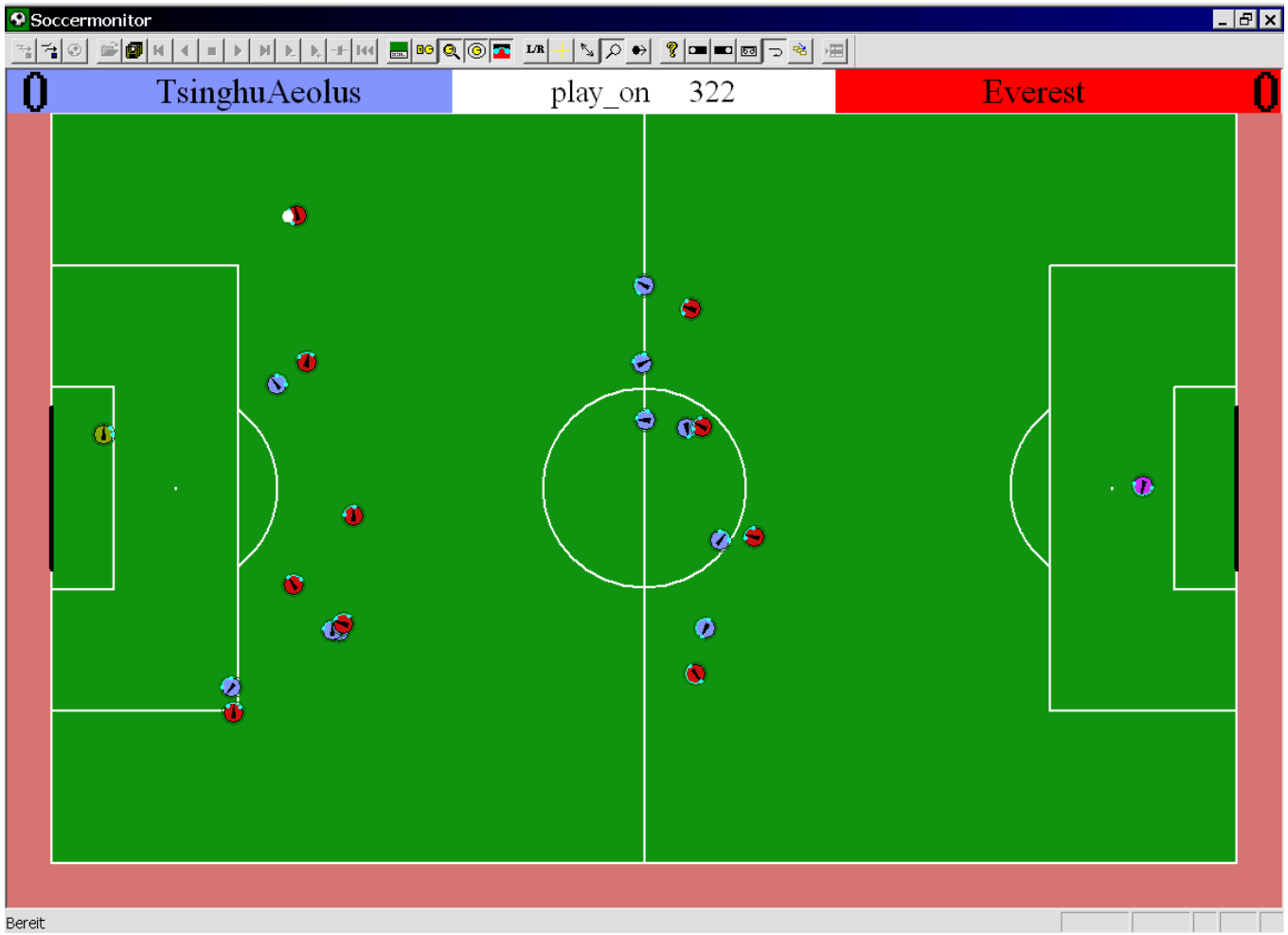
Funktionsweise der Simulation

Client-Server-Architektur
Server simuliert Weltmodell und liefert Sensordaten an Clients
11 Clients (eigener Thread) pro Team
pro Zyklus (6000 à 100 ms) Ausführen eines Kommandos

Herausforderungen:

- verrauschte Sensorinformationen, nur indirekte Kommunikation möglich
- kooperatives Spiel
- begrenzte Kondition

Visualisierung der Simulation



Lernen von Einzelfähigkeiten

- Beispiele: Abfangen eines Balls, ballhalten, dribbeln.
- zusammengesetzt aus Elementaraktionen wie *kick()* oder *dash()*.
- Ablauf des Trainings:
  1. zufälliger Startzustand
  2. Auswahl einer Elementaraktion mit niedrigen Kosten
  3. nach Kostenvergabe: Aktualisieren der Wertefunktion

Lernen von Einzelfähigkeiten

- Beispiele: Abfangen eines Balls, ballhalten, dribbeln.
- zusammengesetzt aus Elementaraktionen wie *kick()* oder *dash()*.
- Ablauf des Trainings:
  1. zufälliger Startzustand
  2. Auswahl einer Elementaraktion mit niedrigen Kosten
  3. nach Kostenvergabe: Aktualisieren der Wertefunktion

Lernen von Einzelfähigkeiten

- Beispiele: Abfangen eines Balls, ballhalten, dribbeln.
- zusammengesetzt aus Elementaraktionen wie *kick()* oder *dash()*.
- Ablauf des Trainings:
  1. zufälliger Startzustand
  2. Auswahl einer Elementaraktion mit niedrigen Kosten
  3. nach Kostenvergabe: Aktualisieren der Wertefunktion

Lernen von Einzelfähigkeiten

- Beispiele: Abfangen eines Balls, ballhalten, dribbeln.
- zusammengesetzt aus Elementaraktionen wie *kick()* oder *dash()*.
- Ablauf des Trainings:
  1. zufälliger Startzustand
  2. Auswahl einer Elementaraktion mit niedrigen Kosten
  3. nach Kostenvergabe: Aktualisieren der Wertefunktion

Lernen von Einzelfähigkeiten

- Beispiele: Abfangen eines Balls, ballhalten, dribbeln.
- zusammengesetzt aus Elementaraktionen wie *kick()* oder *dash()*.
- Ablauf des Trainings:
  1. zufälliger Startzustand
  2. Auswahl einer Elementaraktion mit niedrigen Kosten
  3. nach Kostenvergabe: Aktualisieren der Wertefunktion

Lernen von Einzelfähigkeiten

- Beispiele: Abfangen eines Balls, ballhalten, dribbeln.
- zusammengesetzt aus Elementaraktionen wie *kick()* oder *dash()*.
- Ablauf des Trainings:
  1. zufälliger Startzustand
  2. Auswahl einer Elementaraktion mit niedrigen Kosten
  3. nach Kostenvergabe: Aktualisieren der Wertefunktion

Gliederung

Reinforcement Learning

- (Verstärkendes) Lernprinzip „Zuckerbrot und Peitsche“
  - Kein Lehrer/keine Überwachung notwendig
- Es wird keine Information über eine Lösungsstrategie vorausgesetzt
- Eigenständiges Lernen einer Lösungsstrategie durch (geschickte) Interaktion mit der Umwelt
- Praktische Umsetzung:
  - Agent bekommt für jede ausgeführte Aktion (k)eine Belohnung (Reward)
  - Anhand dieser versucht er eine optimale Handlungsstrategie (Policy) zu finden, die die Belohnung maximiert

Reinforcement Learning

- (Verstärkendes) Lernprinzip „Zuckerbrot und Peitsche“
  - Kein Lehrer/keine Überwachung notwendig
- Es wird keine Information über eine Lösungsstrategie vorausgesetzt
- Eigenständiges Lernen einer Lösungsstrategie durch (geschickte) Interaktion mit der Umwelt
- Praktische Umsetzung:
  - Agent bekommt für jede ausgeführte Aktion (k)eine Belohnung (Reward)
  - Anhand dieser versucht er eine optimale Handlungsstrategie (Policy) zu finden, die die Belohnung maximiert

Reinforcement Learning

- (Verstärkendes) Lernprinzip „Zuckerbrot und Peitsche“
  - Kein Lehrer/keine Überwachung notwendig
- Es wird keine Information über eine Lösungsstrategie vorausgesetzt
- Eigenständiges Lernen einer Lösungsstrategie durch (geschickte) Interaktion mit der Umwelt
- Praktische Umsetzung:
  - Agent bekommt für jede ausgeführte Aktion (k)eine Belohnung (Reward)
  - Anhand dieser versucht er eine optimale Handlungsstrategie (Policy) zu finden, die die Belohnung maximiert

Reinforcement Learning

- (Verstärkendes) Lernprinzip „Zuckerbrot und Peitsche“
  - Kein Lehrer/keine Überwachung notwendig
- Es wird keine Information über eine Lösungsstrategie vorausgesetzt
- Eigenständiges Lernen einer Lösungsstrategie durch (geschickte) Interaktion mit der Umwelt
- Praktische Umsetzung:
  - Agent bekommt für jede ausgeführte Aktion (k)eine Belohnung (Reward)
  - Anhand dieser versucht er eine optimale Handlungsstrategie (Policy) zu finden, die die Belohnung maximiert

Reinforcement Learning

- (Verstärkendes) Lernprinzip „Zuckerbrot und Peitsche“
  - Kein Lehrer/keine Überwachung notwendig
- Es wird keine Information über eine Lösungsstrategie vorausgesetzt
- Eigenständiges Lernen einer Lösungsstrategie durch (geschickte) Interaktion mit der Umwelt
- Praktische Umsetzung:
  - Agent bekommt für jede ausgeführte Aktion (k)eine Belohnung (Reward)
  - Anhand dieser versucht er eine optimale Handlungsstrategie (Policy) zu finden, die die Belohnung maximiert

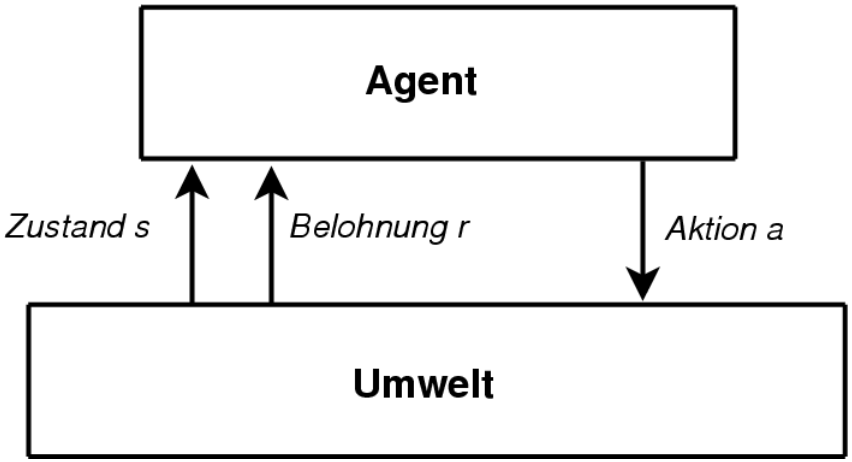
Reinforcement Learning

- (Verstärkendes) Lernprinzip „Zuckerbrot und Peitsche“
  - Kein Lehrer/keine Überwachung notwendig
- Es wird keine Information über eine Lösungsstrategie vorausgesetzt
- Eigenständiges Lernen einer Lösungsstrategie durch (geschickte) Interaktion mit der Umwelt
- Praktische Umsetzung:
  - Agent bekommt für jede ausgeführte Aktion (k)eine Belohnung (Reward)
  - Anhand dieser versucht er eine optimale Handlungsstrategie (Policy) zu finden, die die Belohnung maximiert

Reinforcement Learning

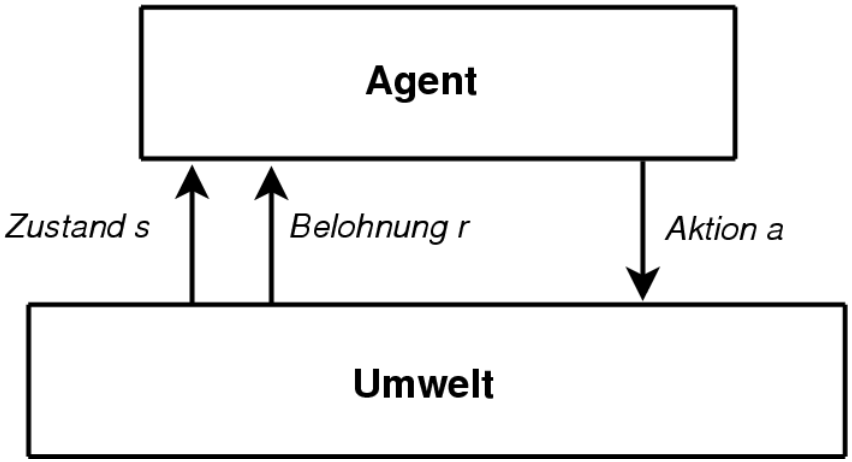
- (Verstärkendes) Lernprinzip „Zuckerbrot und Peitsche“
  - Kein Lehrer/keine Überwachung notwendig
- Es wird keine Information über eine Lösungsstrategie vorausgesetzt
- Eigenständiges Lernen einer Lösungsstrategie durch (geschickte) Interaktion mit der Umwelt
- Praktische Umsetzung:
  - Agent bekommt für jede ausgeführte Aktion (k)eine Belohnung (Reward)
  - Anhand dieser versucht er eine optimale Handlungsstrategie (Policy) zu finden, die die Belohnung maximiert

Reinforcement-Szenario



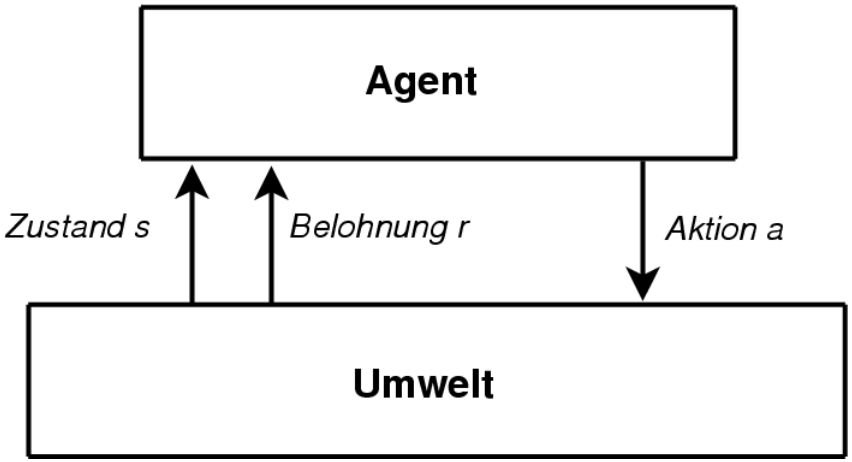
- Annahmen: deterministischer Fall, diskreter Zustandsraum mit endlicher Anzahl von Aktionen
  - Zustandsübergangsfunktion $\delta(s, a)$
  - Rewardfunktion $r(s, a)$
- Fundamentale Frage des RL Problems: „Wie finde ich die optimale Wertefunktion?“

Reinforcement-Szenario



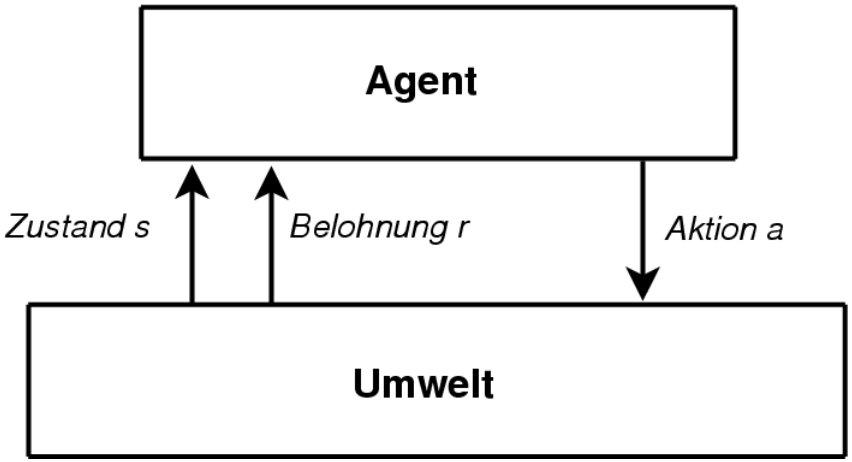
- Annahmen: deterministischer Fall, diskreter Zustandsraum mit endlicher Anzahl von Aktionen
  - Zustandsübergangsfunktion $\delta(s, a)$
  - Rewardfunktion $r(s, a)$
- Fundamentale Frage des RL Problems: „Wie finde ich die optimale Wertefunktion?“

Reinforcement-Szenario



- Annahmen: deterministischer Fall, diskreter Zustandsraum mit endlicher Anzahl von Aktionen
  - Zustandsübergangsfunktion $\delta(s, a)$
  - Rewardfunktion $r(s, a)$
- Fundamentale Frage des RL Problems: „Wie finde ich die optimale Wertefunktion?“

Reinforcement-Szenario



- Annahmen: deterministischer Fall, diskreter Zustandsraum mit endlicher Anzahl von Aktionen
  - Zustandsübergangsfunktion $\delta(s, a)$
  - Rewardfunktion $r(s, a)$
- Fundamentale Frage des RL Problems: „Wie finde ich die optimale Wertefunktion?“

Gliederung

Monte-Carlo (MC)

Monte-Carlo-Methoden basieren auf Schätzungen der Belohnungsfunktion. Das heisst, die Belohnung für Zustand X wird aufgrund der Erfahrung der vorherigen Zustände geschätzt. Hierbei geht der Algorithmus „episodisch“ vor, sprich er besucht nicht alle Zustände

- Nachteile:
  - Keine Garantie für ein globales Optimum der Strategie, da nicht alle Zustände besucht werden müssen
  - Nur für episodische Tasks geeignet

Monte-Carlo (MC)

Monte-Carlo-Methoden basieren auf Schätzungen der Belohnungsfunktion. Das heisst, die Belohnung für Zustand X wird aufgrund der Erfahrung der vorherigen Zustände geschätzt. Hierbei geht der Algorithmus „episodisch“ vor, sprich er besucht nicht alle Zustände

- Nachteile:
  - Keine Garantie für ein globales Optimum der Strategie, da nicht alle Zustände besucht werden müssen
  - Nur für episodische Tasks geeignet

Monte-Carlo (MC)

Monte-Carlo-Methoden basieren auf Schätzungen der Belohnungsfunktion. Das heisst, die Belohnung für Zustand X wird aufgrund der Erfahrung der vorherigen Zustände geschätzt. Hierbei geht der Algorithmus „episodisch“ vor, sprich er besucht nicht alle Zustände

- Nachteile:
  - Keine Garantie für ein globales Optimum der Strategie, da nicht alle Zustände besucht werden müssen
  - Nur für episodische Tasks geeignet

Dynamisches Programmieren (DP)

Der Value-Iteration-Algorithmus [Bellman, 1957] gehört zu den Methoden des dynamischen Programmierens. Es werden alle Zustände durchlaufen, und die aktuelle Belohnung auf die Bewertung des vorherigen Zustands und der dort ausgewählten Aktion übertragen ($\hat{V}_{dp}$-Funktion).

- Weitere DP Algorithmen: Policy Iteration, Policy Improvement, Policy Iteration
- Nachteile:
  - Benötigt komplettes Umgebungsmodell
  - Die komplette Value-Funktion wird zum evaluieren und verbessern eines Zustandes benötigt $\rightarrow$ Speicherbedarf

Dynamisches Programmieren (DP)

Der Value-Iteration-Algorithmus [Bellman, 1957] gehört zu den Methoden des dynamischen Programmierens. Es werden alle Zustände durchlaufen, und die aktuelle Belohnung auf die Bewertung des vorherigen Zustands und der dort ausgewählten Aktion übertragen ($\hat{V}_{dp}$-Funktion).

- Weitere DP Algorithmen: Policy Iteration, Policy Improvement, Policy Iteration
- Nachteile:
  - Benötigt komplettes Umgebungsmodell
  - Die komplette Value-Funktion wird zum evaluieren und verbessern eines Zustandes benötigt $\rightarrow$ Speicherbedarf

Dynamisches Programmieren (DP)

Der Value-Iteration-Algorithmus [Bellman, 1957] gehört zu den Methoden des dynamischen Programmierens. Es werden alle Zustände durchlaufen, und die aktuelle Belohnung auf die Bewertung des vorherigen Zustands und der dort ausgewählten Aktion übertragen ($\hat{V}_{dp}$-Funktion).

- Weitere DP Algorithmen: Policy Iteration, Policy Improvement, Policy Iteration
- Nachteile:
  - Benötigt komplettes Umgebungsmodell
  - Die komplette Value-Funktion wird zum evaluieren und verbessern eines Zustandes benötigt $\rightarrow$ Speicherbedarf

Dynamisches Programmieren (DP)

Der Value-Iteration-Algorithmus [Bellman, 1957] gehört zu den Methoden des dynamischen Programmierens. Es werden alle Zustände durchlaufen, und die aktuelle Belohnung auf die Bewertung des vorherigen Zustands und der dort ausgewählten Aktion übertragen ($\hat{V}_{dp}$-Funktion).

- Weitere DP Algorithmen: Policy Iteration, Policy Improvement, Policy Iteration
- Nachteile:
  - Benötigt komplettes Umgebungsmodell
  - Die komplette Value-Funktion wird zum evaluieren und verbessern eines Zustandes benötigt $\rightarrow$ Speicherbedarf

Gliederung

Temporal Difference Learning (TD)

- kombiniert Ideen der Monte-Carlo Strategie und die Vorgehensweise des Dynamischen Programmierens (DP)
- Hauptunterschied zu DP: TD ist ein „Modellfreies“ Verfahren

Temporal Difference Learning (TD)

- kombiniert Ideen der Monte-Carlo Strategie und die Vorgehensweise des Dynamischen Programmierens (DP)
- Hauptunterschied zu DP: TD ist ein „Modellfreies“ Verfahren

Q-Learning

- Variante des Temporal Difference Learning
- Typisches Merkmal: kein Modell (weder in der Lern- noch in der Aktionsauswahlphase) $\Rightarrow$ „Lernen in unbekannter Umgebung“
  - Konkret: Weder die Zustandsübergangsfunktion $\delta(s, a)$ noch die Belohnungsfunktion $r(s, a)$ sind bekannt.
- Ziel des Q-Lernens: Lernen einer Aktion-Wert-Funktion (Q-Funktion) durch Exploration, die den Nutzen eines Zustandsübergangs durch Auswahl einer bestimmten Aktion wiederspiegelt
  - Q-Funktion liefert für jeden Zustandsübergang einen Nutzenwert $\hat{Q}(s, a)$ unter Annahme, dass danach π^* (optimale Policy) folgt

Q-Learning

- Variante des Temporal Difference Learning
- Typisches Merkmal: kein Modell (weder in der Lern- noch in der Aktionsauswahlphase) $\Rightarrow$ „Lernen in unbekannter Umgebung“
  - Konkret: Weder die Zustandsübergangsfunktion $\delta(s, a)$ noch die Belohnungsfunktion $r(s, a)$ sind bekannt.
- Ziel des Q-Lernens: Lernen einer Aktion-Wert-Funktion (Q-Funktion) durch Exploration, die den Nutzen eines Zustandsübergangs durch Auswahl einer bestimmten Aktion wiederspiegelt
  - Q-Funktion liefert für jeden Zustandsübergang einen Nutzenwert $\hat{Q}(s, a)$ unter Annahme, dass danach π^* (optimale Policy) folgt

Q-Learning

- Variante des Temporal Difference Learning
- Typisches Merkmal: kein Modell (weder in der Lern- noch in der Aktionsauswahlphase) $\Rightarrow$ „Lernen in unbekannter Umgebung“
  - Konkret: Weder die Zustandsübergangsfunktion $\delta(s, a)$ noch die Belohnungsfunktion $r(s, a)$ sind bekannt.
- Ziel des Q-Lernens: Lernen einer Aktion-Wert-Funktion (Q-Funktion) durch Exploration, die den Nutzen eines Zustandsübergangs durch Auswahl einer bestimmten Aktion wiederspiegelt
  - Q-Funktion liefert für jeden Zustandsübergang einen Nutzenwert $\hat{Q}(s, a)$ unter Annahme, dass danach π^* (optimale Policy) folgt

Q-Learning

- Variante des Temporal Difference Learning
- Typisches Merkmal: kein Modell (weder in der Lern- noch in der Aktionsauswahlphase) $\Rightarrow$ „Lernen in unbekannter Umgebung“
  - Konkret: Weder die Zustandsübergangsfunktion $\delta(s, a)$ noch die Belohnungsfunktion $r(s, a)$ sind bekannt.
- Ziel des Q-Lernens: Lernen einer Aktion-Wert-Funktion (Q-Funktion) durch Exploration, die den Nutzen eines Zustandsübergangs durch Auswahl einer bestimmten Aktion wiederspiegelt
  - Q-Funktion liefert für jeden Zustandsübergang einen Nutzenwert $\hat{Q}(s, a)$ unter Annahme, dass danach π^* (optimale Policy) folgt

Q-Learning

- Variante des Temporal Difference Learning
- Typisches Merkmal: kein Modell (weder in der Lern- noch in der Aktionsauswahlphase) $\Rightarrow$ „Lernen in unbekannter Umgebung“
  - Konkret: Weder die Zustandsübergangsfunktion $\delta(s, a)$ noch die Belohnungsfunktion $r(s, a)$ sind bekannt.
- Ziel des Q-Lernens: Lernen einer Aktion-Wert-Funktion (Q-Funktion) durch Exploration, die den Nutzen eines Zustandsübergangs durch Auswahl einer bestimmten Aktion wiederspiegelt
  - Q-Funktion liefert für jeden Zustandsübergang einen Nutzenwert $\hat{Q}(s, a)$ unter Annahme, dass danach π^* (optimale Policy) folgt

Q-Learning Algorithmus [Watkins, 1989]

1. Initialisiere $\hat{Q}(s,a) = 0 \; \forall \; s \in S$ und $a \in A$, s ist Startzustand
2. Wiederhole solange der Agent lebt bzw. solange wie Änderungen von Q klein genug sind
   - wähle eine Aktion a und führe sie aus bzw. nach Ablauf des ersten Iterationsdurchgangs wähle optimale Aktion a^*
   - erhalte kurzfristige Belohnung (Reward) $r \in R$ und neuen Zustand $s' \in S$
   - $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$, wobei $\gamma =$ Discountfaktor mit $0 < \gamma < 1$
   - $s := s'$
3. Gib optimale Policy $\pi^*(s) = argmax_a \hat{Q}(s,a)$.

Q-Learning Algorithmus [Watkins, 1989]

1. Initialisiere $\hat{Q}(s,a) = 0 \; \forall \; s \in S$ und $a \in A$, s ist Startzustand
2. Wiederhole solange der Agent lebt bzw. solange wie Änderungen von Q klein genug sind
   - wähle eine Aktion a und führe sie aus bzw. nach Ablauf des ersten Iterationsdurchgangs wähle optimale Aktion a^*
   - erhalte kurzfristige Belohnung (Reward) $r \in R$ und neuen Zustand $s' \in S$
   - $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$, wobei $\gamma =$ Discountfaktor mit $0 < \gamma < 1$
   - $s := s'$
3. Gib optimale Policy $\pi^*(s) = argmax_a \hat{Q}(s,a)$.

Q-Learning Algorithmus [Watkins, 1989]

1. Initialisiere $\hat{Q}(s,a) = 0 \; \forall \; s \in S$ und $a \in A$, s ist Startzustand
2. Wiederhole solange der Agent lebt bzw. solange wie Änderungen von Q klein genug sind
   - wähle eine Aktion a und führe sie aus bzw. nach Ablauf des ersten Iterationsdurchgangs wähle optimale Aktion a^*
   - erhalte kurzfristige Belohnung (Reward) $r \in R$ und neuen Zustand $s' \in S$
   - $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$, wobei $\gamma =$ Discountfaktor mit $0 < \gamma < 1$
   - $s := s'$
3. Gib optimale Policy $\pi^*(s) = argmax_a \hat{Q}(s,a)$.

Q-Learning Algorithmus [Watkins, 1989]

1. Initialisiere $\hat{Q}(s,a) = 0 \; \forall \; s \in S$ und $a \in A$, s ist Startzustand
2. Wiederhole solange der Agent lebt bzw. solange wie Änderungen von Q klein genug sind
   - wähle eine Aktion a und führe sie aus bzw. nach Ablauf des ersten Iterationsdurchgangs wähle optimale Aktion a^*
   - erhalte kurzfristige Belohnung (Reward) $r \in R$ und neuen Zustand $s' \in S$
   - $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$, wobei γ = Discountfaktor mit $0 < \gamma < 1$
   - $s := s'$
3. Gib optimale Policy $\pi^*(s) = argmax_a \hat{Q}(s,a)$.

Q-Learning Algorithmus [Watkins, 1989]

1. Initialisiere $\hat{Q}(s,a) = 0 \; \forall \; s \in S$ und $a \in A$, s ist Startzustand
2. Wiederhole solange der Agent lebt bzw. solange wie Änderungen von Q klein genug sind
   - wähle eine Aktion a und führe sie aus bzw. nach Ablauf des ersten Iterationsdurchgangs wähle optimale Aktion a^*
   - erhalte kurzfristige Belohnung (Reward) $r \in R$ und neuen Zustand $s' \in S$
   - $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$, wobei γ = Discountfaktor mit $0 < \gamma < 1$
   - $s := s'$
3. Gib optimale Policy $\pi^*(s) = argmax_a \hat{Q}(s,a)$.

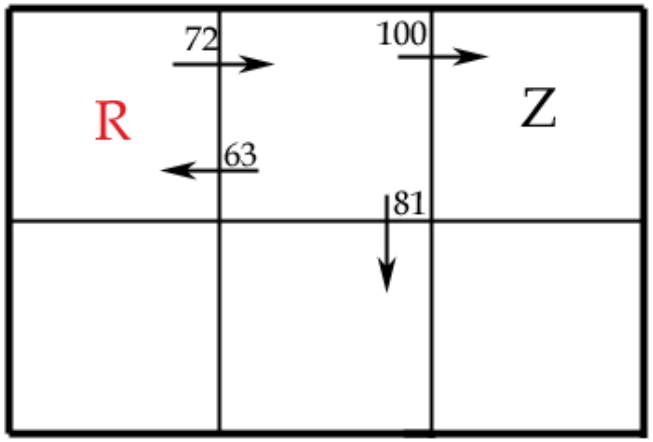
Q-Learning Algorithmus [Watkins, 1989]

1. Initialisiere $\hat{Q}(s,a) = 0 \; \forall \; s \in S$ und $a \in A$, s ist Startzustand
2. Wiederhole solange der Agent lebt bzw. solange wie Änderungen von Q klein genug sind
   - wähle eine Aktion a und führe sie aus bzw. nach Ablauf des ersten Iterationsdurchgangs wähle optimale Aktion a^*
   - erhalte kurzfristige Belohnung (Reward) $r \in R$ und neuen Zustand $s' \in S$
   - $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$, wobei γ = Discountfaktor mit $0 < \gamma < 1$
   - $s := s'$
3. Gib optimale Policy $\pi^*(s) = argmax_a \hat{Q}(s,a)$.

Q-Learning Algorithmus [Watkins, 1989]

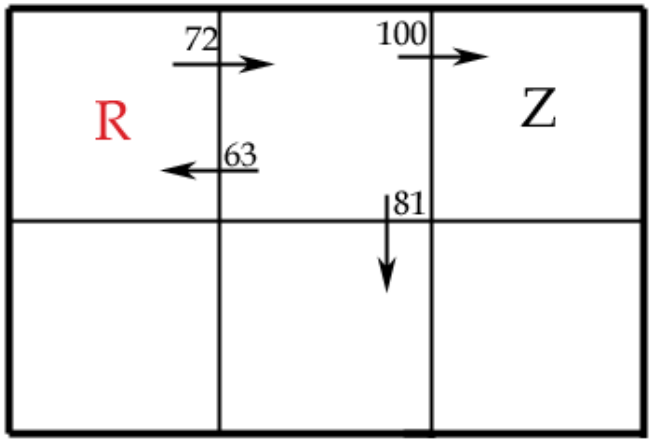
1. Initialisiere $\hat{Q}(s,a) = 0 \; \forall \; s \in S$ und $a \in A$, s ist Startzustand
2. Wiederhole solange der Agent lebt bzw. solange wie Änderungen von Q klein genug sind
   - wähle eine Aktion a und führe sie aus bzw. nach Ablauf des ersten Iterationsdurchgangs wähle optimale Aktion a^*
   - erhalte kurzfristige Belohnung (Reward) $r \in R$ und neuen Zustand $s' \in S$
   - $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$, wobei γ = Discountfaktor mit $0 < \gamma < 1$
   - $s := s'$
3. Gib optimale Policy $\pi^*(s) = argmax_a \hat{Q}(s,a)$.

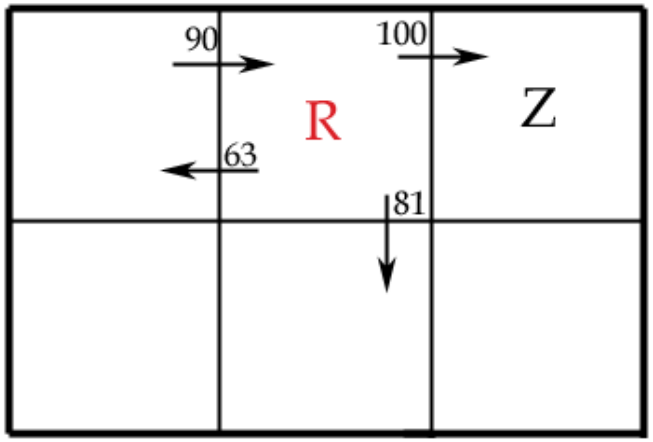
Q-Learning Beispiel



- Update der Q-Werte
- Q-Funktion: $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s', a')$
- hier im Beispiel: $0 + 0.9 * max 63, 81, 100 = 0.9 * 100 = 90$

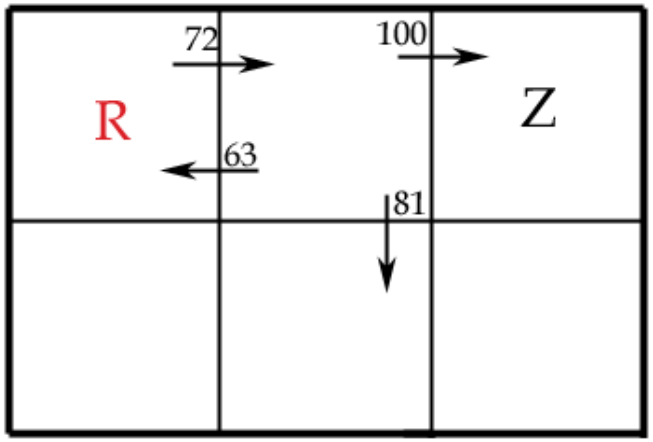
Q-Learning Beispiel

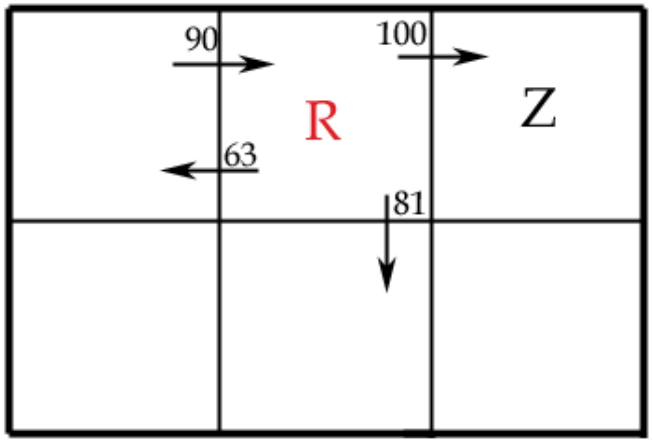




- Update der Q-Werte
- Q-Funktion: $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$
- hier im Beispiel: $0 + 0.9 * max 63, 81, 100 = 0.9 * 100 = 90$

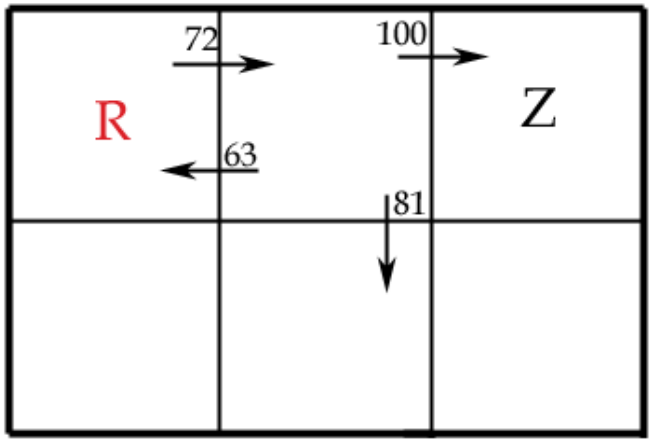
Q-Learning Beispiel

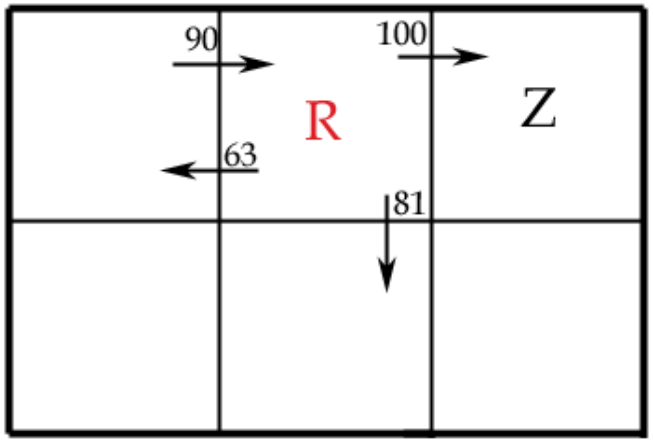




- Update der Q-Werte
- Q-Funktion: $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$
- hier im Beispiel: $0 + 0.9 * max 63, 81, 100 = 0.9 * 100 = 90$

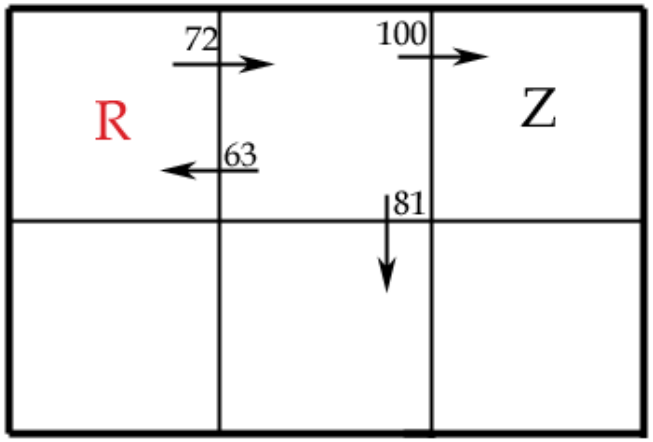
Q-Learning Beispiel

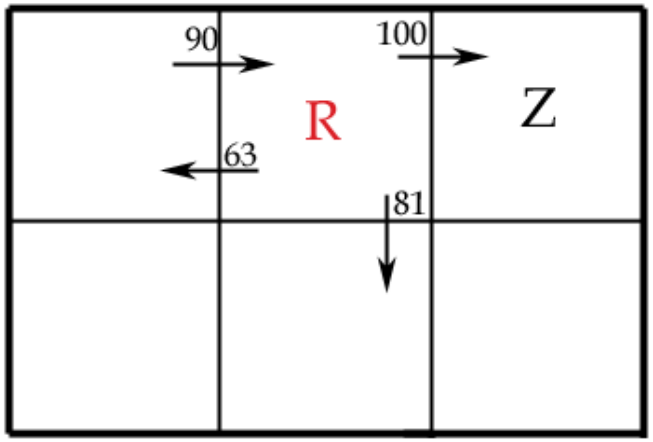




- Update der Q-Werte
- Q-Funktion: $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$
- hier im Beispiel: $0 + 0.9 * max 63, 81, 100 = 0.9 * 100 = 90$

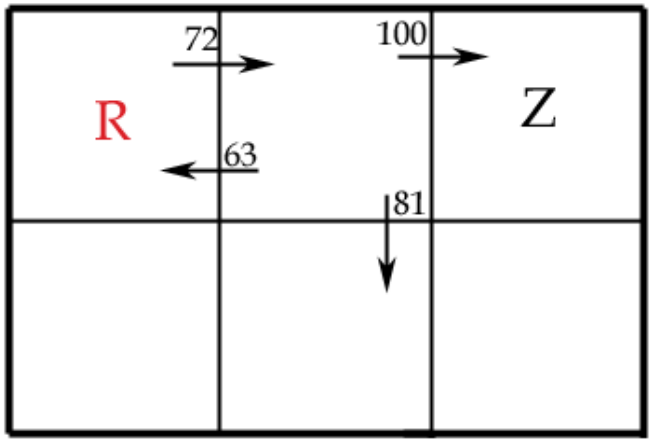
Q-Learning Beispiel

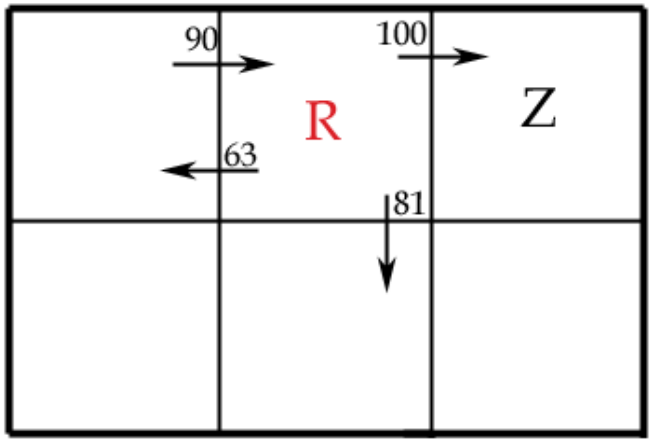




- Update der Q-Werte
- Q-Funktion: $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$
- hier im Beispiel: $0 + 0.9 * max 63, 81, 100 = 0.9 * 100 = 90$

Q-Learning Beispiel





- Update der Q-Werte
- Q-Funktion: $\hat{Q}(s,a) := r + \gamma * max_{a' \in A} \hat{Q}(s',a')$
- hier im Beispiel: $0 + 0.9 * max63, 81, 100 = 0.9 * 100 = 90$

Gliederung

Exploration vs. Exploitation

- Wann lernt der Agent?
  - Exploration (Erforschung): Ausprobieren einer neuen Aktion, um mögliche, eventuell bessere Strategien zu finden $\Rightarrow$ „Lernphase“
  - Exploitation (Ausnutzung): Wahl der bislang „besten“ Aktionen $\Rightarrow$ „Anwendungsphase“
- Optimale Vorgehensweise: „Mittelweg“ zwischen Lern- und Anwendungsphase

Exploration vs. Exploitation

- Wann lernt der Agent?
  - Exploration (Erforschung): Ausprobieren einer neuen Aktion, um mögliche, eventuell bessere Strategien zu finden $\Rightarrow$ „Lernphase“
  - Exploitation (Ausnutzung): Wahl der bislang „besten“ Aktionen $\Rightarrow$ „Anwendungsphase“
- Optimale Vorgehensweise: „Mittelweg“ zwischen Lern- und Anwendungsphase

Exploration vs. Exploitation

- Wann lernt der Agent?
  - Exploration (Erforschung): Ausprobieren einer neuen Aktion, um mögliche, eventuell bessere Strategien zu finden ⇒ „Lernphase“
  - Exploitation (Ausnutzung): Wahl der bislang „besten“ Aktionen ⇒ „Anwendungsphase“
- Optimale Vorgehensweise: „Mittelweg“ zwischen Lern- und Anwendungsphase

Exploration vs. Exploitation

- Wann lernt der Agent?
  - Exploration (Erforschung): Ausprobieren einer neuen Aktion, um mögliche, eventuell bessere Strategien zu finden $\Rightarrow$ „Lernphase“
  - Exploitation (Ausnutzung): Wahl der bislang „besten“ Aktionen $\Rightarrow$ „Anwendungsphase“
- Optimale Vorgehensweise: „Mittelweg“ zwischen Lern- und Anwendungsphase

Gliederung

Einführung

- große Menge an vernetzten Neuronen
- Gewichte an den Kanten simulieren selektive Weitergabe von echten Neuronen
- mehrere Schichten:
  - Eingabeschicht, mit allen Eingangsparamtern verbunden
  - verdeckte Schicht(en), mit Ausgängen der Eingabeschicht verbunden
  - Ausgabeschicht, mit Ausgängen der verdeckten Schicht verbunden

Einführung

- große Menge an vernetzten Neuronen
- Gewichte an den Kanten simulieren selektive Weitergabe von echten Neuronen
- mehrere Schichten:
  - Eingabeschicht, mit allen Eingangsparamtern verbunden
  - verdeckte Schicht(en), mit Ausgängen der Eingabeschicht verbunden
  - Ausgabeschicht, mit Ausgängen der verdeckten Schicht verbunden

Einführung

- große Menge an vernetzten Neuronen
- Gewichte an den Kanten simulieren selektive Weitergabe von echten Neuronen
- mehrere Schichten:
  - Eingabeschicht, mit allen Eingangsparamtern verbunden
  - verdeckte Schicht(en), mit Ausgängen der Eingabeschicht verbunden
  - Ausgabeschicht, mit Ausgängen der verdeckten Schicht verbunden

Einführung

- große Menge an vernetzten Neuronen
- Gewichte an den Kanten simulieren selektive Weitergabe von echten Neuronen
- mehrere Schichten:
  - Eingabeschicht, mit allen Eingangsparamtern verbunden
  - verdeckte Schicht(en), mit Ausgängen der Eingabeschicht verbunden
  - Ausgabeschicht, mit Ausgängen der verdeckten Schicht verbunden

Einführung

- große Menge an vernetzten Neuronen
- Gewichte an den Kanten simulieren selektive Weitergabe von echten Neuronen
- mehrere Schichten:
  - Eingabeschicht, mit allen Eingangsparamtern verbunden
  - verdeckte Schicht(en), mit Ausgängen der Eingabeschicht verbunden
  - Ausgabeschicht, mit Ausgängen der verdeckten Schicht verbunden

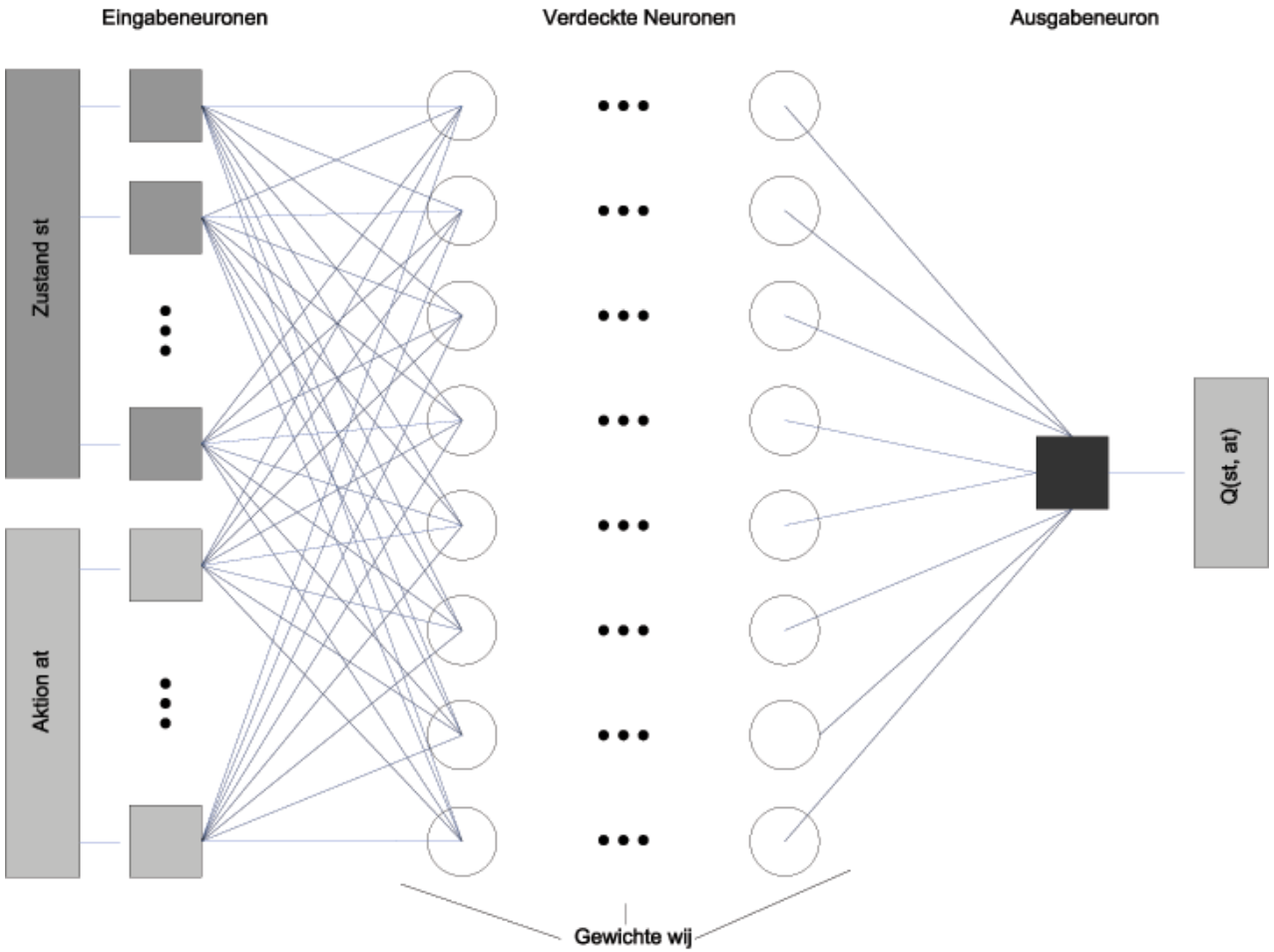
Einführung

- große Menge an vernetzten Neuronen
- Gewichte an den Kanten simulieren selektive Weitergabe von echten Neuronen
- mehrere Schichten:
  - Eingabeschicht, mit allen Eingangsparamtern verbunden
  - verdeckte Schicht(en), mit Ausgängen der Eingabeschicht verbunden
  - Ausgabeschicht, mit Ausgängen der verdeckten Schicht verbunden

Gliederung

Bedeutung für Reinforcement Learning

Abbildung der Q-Funktion als Neuronales Netz



Gliederung

„Mindstormers“: Lernen von Teamfähigkeiten

- Problem: Anzahl der Spieler wirkt sich exponentiell auf Aktionsmenge aus
- Kooperation erreichen durch: gemeinsames Reinforcement-Signal
- Ablauf der Lernphase: (überwachtes Lernen)
  1. Ermittlung der erfolgreichen Aktionen
  2. Berechnung des Folgezustands (approximativ)
  3. Bewertung des Folgezustands über neuronales Netz
  4. Auswahl der Aktion mit geringstem Wert

„Mindstormers“: Lernen von Teamfähigkeiten

- Problem: Anzahl der Spieler wirkt sich exponentiell auf Aktionsmenge aus
- Kooperation erreichen durch: gemeinsames Reinforcement-Signal
- Ablauf der Lernphase: (überwachtes Lernen)
  1. Ermittlung der erfolgreichen Aktionen
  2. Berechnung des Folgezustands (approximativ)
  3. Bewertung des Folgezustands über neuronales Netz
  4. Auswahl der Aktion mit geringstem Wert

„Mindstormers“: Lernen von Teamfähigkeiten

- Problem: Anzahl der Spieler wirkt sich exponentiell auf Aktionsmenge aus
- Kooperation erreichen durch: gemeinsames Reinforcement-Signal
- Ablauf der Lernphase: (überwachtes Lernen)
  1. Ermittlung der erfolgreichen Aktionen
  2. Berechnung des Folgezustands (approximativ)
  3. Bewertung des Folgezustands über neuronales Netz
  4. Auswahl der Aktion mit geringstem Wert

„Mindstormers“: Lernen von Teamfähigkeiten

- Problem: Anzahl der Spieler wirkt sich exponentiell auf Aktionsmenge aus
- Kooperation erreichen durch: gemeinsames Reinforcement-Signal
- Ablauf der Lernphase: (überwachtes Lernen)
  1. Ermittlung der erfolgreichen Aktionen
  2. Berechnung des Folgezustands (approximativ)
  3. Bewertung des Folgezustands über neuronales Netz
  4. Auswahl der Aktion mit geringstem Wert

„Mindstormers“: Lernen von Teamfähigkeiten

- Problem: Anzahl der Spieler wirkt sich exponentiell auf Aktionsmenge aus
- Kooperation erreichen durch: gemeinsames Reinforcement-Signal
- Ablauf der Lernphase: (überwachtes Lernen)
  1. Ermittlung der erfolgreichen Aktionen
  2. Berechnung des Folgezustands (approximativ)
  3. Bewertung des Folgezustands über neuronales Netz
  4. Auswahl der Aktion mit geringstem Wert

„Mindstormers“: Lernen von Teamfähigkeiten

- Problem: Anzahl der Spieler wirkt sich exponentiell auf Aktionsmenge aus
- Kooperation erreichen durch: gemeinsames Reinforcement-Signal
- Ablauf der Lernphase: (überwachtes Lernen)
  1. Ermittlung der erfolgreichen Aktionen
  2. Berechnung des Folgezustands (approximativ)
  3. Bewertung des Folgezustands über neuronales Netz
  4. Auswahl der Aktion mit geringstem Wert

„Mindstormers“: Lernen von Teamfähigkeiten

- Problem: Anzahl der Spieler wirkt sich exponentiell auf Aktionsmenge aus
- Kooperation erreichen durch: gemeinsames Reinforcement-Signal
- Ablauf der Lernphase: (überwachtes Lernen)
  1. Ermittlung der erfolgreichen Aktionen
  2. Berechnung des Folgezustands (approximativ)
  3. Bewertung des Folgezustands über neuronales Netz
  4. Auswahl der Aktion mit geringstem Wert

Gliederung

Middle-Size-Liga: „Tribots“

Regeln der Middle-Size-Liga

- bis zu 4 Roboter pro Team
- max. 50 cm Durchmesser
- Spielfeld: 12 x 8 m
- Dauer: 2 x 10 Minuten

Herausforderungen:

- Selbstlokalisation durch Kamera
- kooperatives Spiel
- begrenzte Rechenkapazität (onboard-Rechner)

Middle-Size-Liga: „Tribots“

Regeln der Middle-Size-Liga

- bis zu 4 Roboter pro Team
- max. 50 cm Durchmesser
- Spielfeld: 12 x 8 m
- Dauer: 2 x 10 Minuten

Herausforderungen:

- Selbstlokalisation durch Kamera
- kooperatives Spiel
- begrenzte Rechenkapazität (onboard-Rechner)

Middle-Size-Liga: „Tribots“

Regeln der Middle-Size-Liga

- bis zu 4 Roboter pro Team
- max. 50 cm Durchmesser
- Spielfeld: 12 x 8 m
- Dauer: 2 x 10 Minuten

Herausforderungen:

- Selbstlokalisation durch Kamera
- kooperatives Spiel
- begrenzte Rechenkapazität (onboard-Rechner)

Middle-Size-Liga: „Tribots“

Regeln der Middle-Size-Liga

- bis zu 4 Roboter pro Team
- max. 50 cm Durchmesser
- Spielfeld: 12 x 8 m
- Dauer: 2 x 10 Minuten

Herausforderungen:

- Selbstlokalisation durch Kamera
- kooperatives Spiel
- begrenzte Rechenkapazität (onboard-Rechner)

Middle-Size-Liga: „Tribots“

Regeln der Middle-Size-Liga

- bis zu 4 Roboter pro Team
- max. 50 cm Durchmesser
- Spielfeld: 12 x 8 m
- Dauer: 2 x 10 Minuten

Herausforderungen:

- Selbstlokalisation durch Kamera
- kooperatives Spiel
- begrenzte Rechenkapazität (onboard-Rechner)

Middle-Size-Liga: „Tribots“

Regeln der Middle-Size-Liga

- bis zu 4 Roboter pro Team
- max. 50 cm Durchmesser
- Spielfeld: 12 x 8 m
- Dauer: 2 x 10 Minuten

Herausforderungen:

- Selbstlokalisation durch Kamera
- kooperatives Spiel
- begrenzte Rechenkapazität (onboard-Rechner)

Middle-Size-Liga: „Tribots“

Regeln der Middle-Size-Liga

- bis zu 4 Roboter pro Team
- max. 50 cm Durchmesser
- Spielfeld: 12 x 8 m
- Dauer: 2 x 10 Minuten

Herausforderungen:

- Selbstlokalisation durch Kamera
- kooperatives Spiel
- begrenzte Rechenkapazität (onboard-Rechner)

Verarbeitung von Sensorinformationen

- Sammeln von Daten über Sensoren (Kamera und Odometrie)
- Bestimmen der Position über Spielfeldmarkierungen
- Bestimmen der Ballposition und -geschwindigkeit
- begrenzte Rechenleistung erfordert sehr simple Bildverarbeitung (Farberkennung)
- Erstellen eines Umweltbildes aus den Sensordaten, darauf basierend Entscheidung fällen

Verarbeitung von Sensorinformationen

- Sammeln von Daten über Sensoren (Kamera und Odometrie)
- Bestimmen der Position über Spielfeldmarkierungen
- Bestimmen der Ballposition und -geschwindigkeit
- begrenzte Rechenleistung erfordert sehr simple Bildverarbeitung (Farberkennung)
- Erstellen eines Umweltbildes aus den Sensordaten, darauf basierend Entscheidung fällen

Verarbeitung von Sensorinformationen

- Sammeln von Daten über Sensoren (Kamera und Odometrie)
- Bestimmen der Position über Spielfeldmarkierungen
- Bestimmen der Ballposition und -geschwindigkeit
- begrenzte Rechenleistung erfordert sehr simple Bildverarbeitung (Farberkennung)
- Erstellen eines Umweltbildes aus den Sensordaten, darauf basierend Entscheidung fällen

Verarbeitung von Sensorinformationen

- Sammeln von Daten über Sensoren (Kamera und Odometrie)
- Bestimmen der Position über Spielfeldmarkierungen
- Bestimmen der Ballposition und -geschwindigkeit
- begrenzte Rechenleistung erfordert sehr simple Bildverarbeitung (Farberkennung)
- Erstellen eines Umweltbildes aus den Sensordaten, darauf basierend Entscheidung fällen

Verarbeitung von Sensorinformationen

- Sammeln von Daten über Sensoren (Kamera und Odometrie)
- Bestimmen der Position über Spielfeldmarkierungen
- Bestimmen der Ballposition und -geschwindigkeit
- begrenzte Rechenleistung erfordert sehr simple Bildverarbeitung (Farberkennung)
- Erstellen eines Umweltbildes aus den Sensordaten, darauf basierend Entscheidung fällen

Lernen von Einzelfähigkeiten und Regelungsaufgaben

- Verfahren aus Simulation nicht direkt übertragbar
- Ansatz: Training an simuliertem Roboter, dann übertragen auf realen
- Allerdings geht optimale Eigenschaft verloren
- Lösung: optimierter Q-Learning-Algorithmus, erlaubt Training am realen Roboter mit geringem Aufwand
- Ansteuerung der Fahrwerksmotoren auch durch RL lernbar

Lernen von Einzelfähigkeiten und Regelungsaufgaben

- Verfahren aus Simulation nicht direkt übertragbar
- Ansatz: Training an simuliertem Roboter, dann übertragen auf realen
- Allerdings geht optimale Eigenschaft verloren
- Lösung: optimierter Q-Learning-Algorithmus, erlaubt Training am realen Roboter mit geringem Aufwand
- Ansteuerung der Fahrwerksmotoren auch durch RL lernbar

Lernen von Einzelfähigkeiten und Regelungsaufgaben

- Verfahren aus Simulation nicht direkt übertragbar
- Ansatz: Training an simuliertem Roboter, dann übertragen auf realen
- Allerdings geht optimale Eigenschaft verloren
- Lösung: optimierter Q-Learning-Algorithmus, erlaubt Training am realen Roboter mit geringem Aufwand
- Ansteuerung der Fahrwerksmotoren auch durch RL lernbar

Lernen von Einzelfähigkeiten und Regelungsaufgaben

- Verfahren aus Simulation nicht direkt übertragbar
- Ansatz: Training an simuliertem Roboter, dann übertragen auf realen
- Allerdings geht optimale Eigenschaft verloren
- Lösung: optimierter Q-Learning-Algorithmus, erlaubt Training am realen Roboter mit geringem Aufwand
- Ansteuerung der Fahrwerksmotoren auch durch RL lernbar

Lernen von Einzelfähigkeiten und Regelungsaufgaben

- Verfahren aus Simulation nicht direkt übertragbar
- Ansatz: Training an simuliertem Roboter, dann übertragen auf realen
- Allerdings geht optimale Eigenschaft verloren
- Lösung: optimierter Q-Learning-Algorithmus, erlaubt Training am realen Roboter mit geringem Aufwand
- Ansteuerung der Fahrwerksmotoren auch durch RL lernbar

Fazit Reinforcement Learning

- Koexistenz von handprogrammierten und gelernten Modulen möglich.
- Erlernte Fähigkeiten waren den handgefertigten anfangs überlegen.
- Zunehmend Ersetzen der erlernten durch handprogrammierte, optimierte Module.
- Mögliche Anwendungsgebiete: Regelungstechnik in der Industrie, reaktives Scheduling.

Fazit Reinforcement Learning

- Koexistenz von handprogrammierten und gelernten Modulen möglich.
- Erlernte Fähigkeiten waren den handgefertigten anfangs überlegen.
- Zunehmend Ersetzen der erlernten durch handprogrammierte, optimierte Module.
- Mögliche Anwendungsgebiete: Regelungstechnik in der Industrie, reaktives Scheduling.

Fazit Reinforcement Learning

- Koexistenz von handprogrammierten und gelernten Modulen möglich.
- Erlernte Fähigkeiten waren den handgefertigten anfangs überlegen.
- Zunehmend Ersetzen der erlernten durch handprogrammierte, optimierte Module.
- Mögliche Anwendungsgebiete: Regelungstechnik in der Industrie, reaktives Scheduling.

Fazit Reinforcement Learning

- Koexistenz von handprogrammierten und gelernten Modulen möglich.
- Erlernte Fähigkeiten waren den handgefertigten anfangs überlegen.
- Zunehmend Ersetzen der erlernten durch handprogrammierte, optimierte Module.
- Mögliche Anwendungsgebiete: Regelungstechnik in der Industrie, reaktives Scheduling.

Danke.

Fragen?