Bestärkendes Lernen

M. Heckel

16. Mai 2019

Inhalt

Bisher genutzter Ansatz

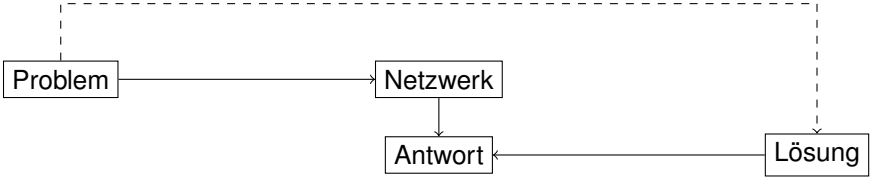
Bisher genutzter Ansatz

- Neuronales Netz trainieren:

Bisher genutzter Ansatz

- Neuronales Netz trainieren:

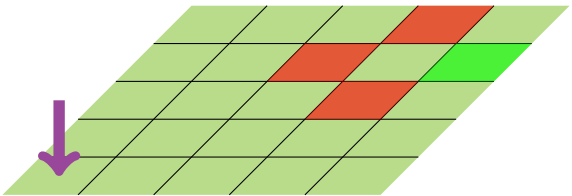
- Neuronales Netz testen:



Situation bei RL-Problemen

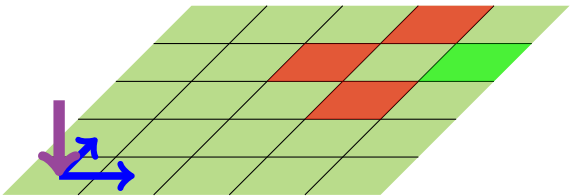
Situation bei RL-Problemen

- Eine **Umgebung** (gesamte Spielwelt) befindet sich in einem **Zustand** s



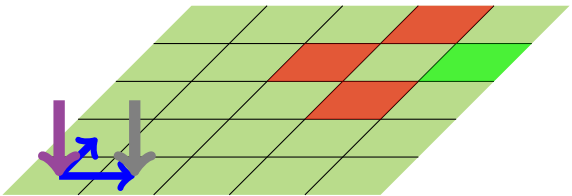
Situation bei RL-Problemen

- Eine **Umgebung** (gesamte Spielwelt) befindet sich in einem **Zustand** s
- Der Spieler/**Aktor** wählt eine **Aktion** a



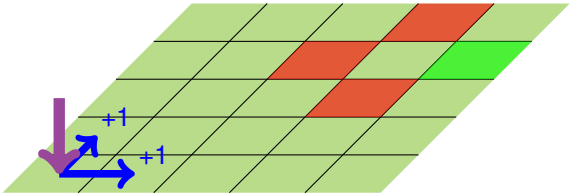
Situation bei RL-Problemen

- Eine **Umgebung** (gesamte Spielwelt) befindet sich in einem **Zustand** s
- Der Spieler/**Aktor** wählt eine **Aktion** a
- Aus einem Zustand s und einer Aktion a ergibt sich ein **Folgezustand** s'



Policy Gradient

- Das Netzwerk lernt, welche Aktion a in einem gegebenen Zustand s wie gut ist



Schwierigkeiten des Policy Gradient

- Netz muss trainiert werden (Problem und dessen Lösung zum Training (Gradientenoptimierung) benötigt)

Schwierigkeiten des Policy Gradient

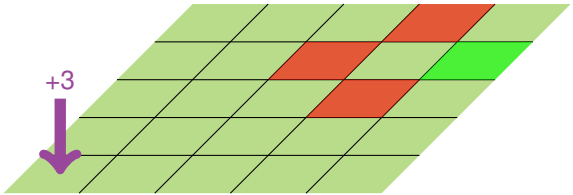
- Netz muss trainiert werden (Problem und dessen Lösung zum Training (Gradientenoptimierung) benötigt)
- Wert einer Aktion in einem Zustand hängt meistens von den folgenden Aktionen ab

Schwierigkeiten des Policy Gradient

- Netz muss trainiert werden (Problem und dessen Lösung zum Training (Gradientenoptimierung) benötigt)
- Wert einer Aktion in einem Zustand hängt meistens von den folgenden Aktionen ab
- Bei komplexen Umgebungen nicht möglich, den Wert einer Aktion direkt zu berechnen

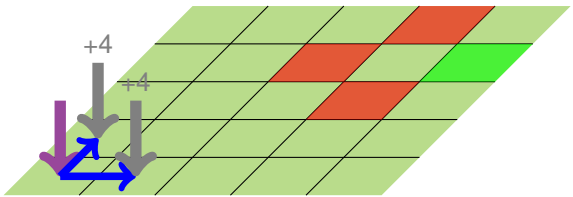
Value Function

- Das Netzwerk gibt aus, wie gut es ist, in einem Zustand s zu sein



Value Function

- Das Netzwerk gibt aus, wie gut es ist, in einem Zustand s zu sein
- Das Wählen einer Aktion a in einem Zustand s führt zu einem Folgezustand s'



Schwierigkeiten der Value Function

- Netz muss trainiert werden (Problem und dessen Lösung zum Training (Gradientenoptimierung) benötigt)

Schwierigkeiten der Value Function

- Netz muss trainiert werden (Problem und dessen Lösung zum Training (Gradientenoptimierung) benötigt)
- Wert eines Zustandes hängt meistens von den folgenden Aktionen (bzw. sich daraus ergebenden Zuständen) ab

Schwierigkeiten der Value Function

- Netz muss trainiert werden (Problem und dessen Lösung zum Training (Gradientenoptimierung) benötigt)
- Wert eines Zustandes hängt meistens von den folgenden Aktionen (bzw. sich daraus ergebenden Zuständen) ab
- Bei komplexen Umgebungen nicht möglich, den Wert eines Zustands direkt zu berechnen

Model-Ansatz

- **Problem:** Bei Anwendungen mit physischen Komponenten: Training schwierig
  - Wenn das Netz falsch entscheidet: (fatale) Auswirkungen
  - Umgebung ist ggf. schwer zurück zu setzen
  - Geschwindigkeit des Lernens ggf. durch physische Grenzen beschränkt

Model-Ansatz

- **Problem:** Bei Anwendungen mit physischen Komponenten: Training schwierig
  - Wenn das Netz falsch entscheidet: (fatale) Auswirkungen
  - Umgebung ist ggf. schwer zurück zu setzen
  - Geschwindigkeit des Lernens ggf. durch physische Grenzen beschränkt
- **Lösung:** Simulation der Umgebung erstellen und Netz mit der Simulation trainieren

Markov-Entscheidungsprozess

- Möglichkeit, Entscheidungsprobleme mathematisch zu formulieren
- $MEP = (S, A, t, r)$

Markov-Entscheidungsprozess

- Möglichkeit, Entscheidungsprobleme mathematisch zu formulieren
- $MEP = (S, A, t, r)$
  - S: Menge aller Zustände
  - A: Menge aller möglichen Aktionen
  - t: Übergangsfunktion der Zustände, aus dem Zustand s und der Aktion a folgt der Folgezustand s'
    $t(s, a) = s'$
  - r: Wert v einer Aktion a in einem Zustand s
    $r(s, a) = v$

Markov-Entscheidungsprozess

- Möglichkeit, Entscheidungsprobleme mathematisch zu formulieren
- $MEP = (S, A, t, r)$
  - S: Menge aller Zustände
  - A: Menge aller möglichen Aktionen
  - t: Übergangsfunktion der Zustände, aus dem Zustand s und der Aktion a folgt der Folgezustand s'
    $t(s, a) = s'$
  - r: Wert v einer Aktion a in einem Zustand s
    $r(s, a) = v$
- **Lösung:** Funktion $\pi(s) = a$, die einen Zustand auf die in diesem Zustand beste Aktion mappt

Exploration

- Wenn das Netzwerk gute Policies gelernt hat: Verwenden dieser Policies

Exploration

- Wenn das Netzwerk gute Policies gelernt hat: Verwenden dieser Policies
- **Problem:** Vielleicht gibt es eine bessere Policy, die das Netz nicht entdeckt

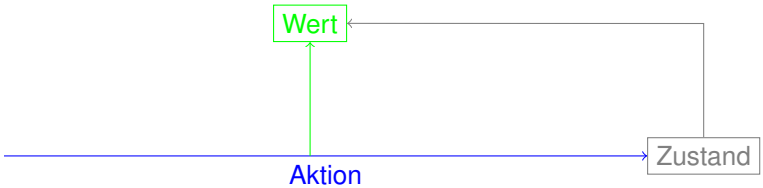
Exploration

- Wenn das Netzwerk gute Policies gelernt hat: Verwenden dieser Policies
- **Problem:** Vielleicht gibt es eine bessere Policy, die das Netz nicht entdeckt
- **Lösung:** Exploration erzwingen

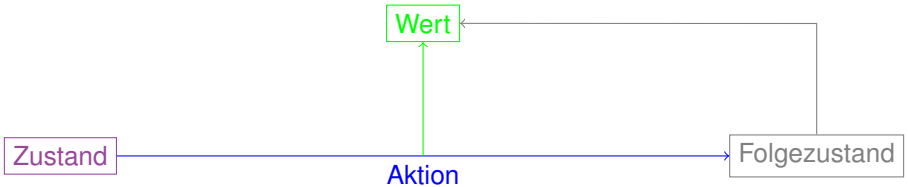
Exploration

- Wenn das Netzwerk gute Policies gelernt hat: Verwenden dieser Policies
- **Problem:** Vielleicht gibt es eine bessere Policy, die das Netz nicht entdeckt
- **Lösung:** Exploration erzwingen
  - Definieren einer Wahrscheinlichkeit ε, mit der das Netz eine zufällige Aktion wählt

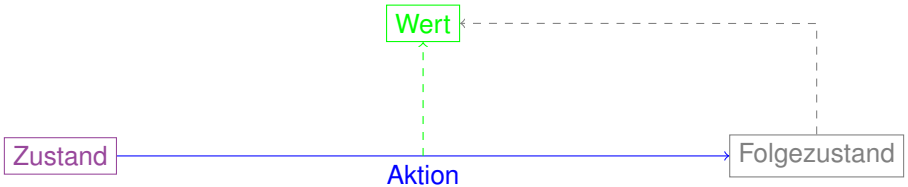
Typen von RL Problemen



Typen von RL Problemen



Typen von RL Problemen



Typen von RL Problemen

- **Problem:** Wert einer Aktion oder eines Zustands nicht direkt berechenbar
- **Lösung:**
  - Erfahrungen in einem Puffer speichern $(s, a, s', v = 0)$
  - Bei Spielende ist der Wert des letzten Tupel bekannt (gewonnen, verloren, unentschieden etc.)
  - Berechnen aller vorherigen v-Werte mit Discount-Factor
  - Trainieren des Netzes mit den Werten aus dem Puffer

Q-Learning

- Bestimmen des Q-Wertes einer Aktion *a* in einem Zustand *s*

Q-Learning

- Bestimmen des Q-Wertes einer Aktion *a* in einem Zustand *s*
- Je höher der Q-Wert desto besser die Aktion in dem entsprechenden Zustand

Q-Learning

- Bestimmen des Q-Wertes einer Aktion *a* in einem Zustand *s*
- Je höher der Q-Wert desto besser die Aktion in dem entsprechenden Zustand
- Wählen der Aktion mit dem höchsten Q-Wert

Q-Learning

- Bestimmen des Q-Wertes einer Aktion a in einem Zustand s
- Je höher der Q-Wert desto besser die Aktion in dem entsprechenden Zustand
- Wählen der Aktion mit dem höchsten Q-Wert
- Berechnung mittels Bellman Gleichung:

$$Q(s,a) = r + \gamma \times (\max(Q(s',a')))$$

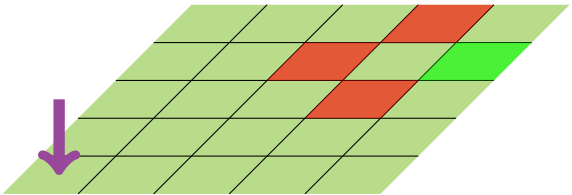
Q-Learning

- Bestimmen des Q-Wertes einer Aktion a in einem Zustand s
- Je höher der Q-Wert desto besser die Aktion in dem entsprechenden Zustand
- Wählen der Aktion mit dem höchsten Q-Wert
- Berechnung mittels Bellman Gleichung:

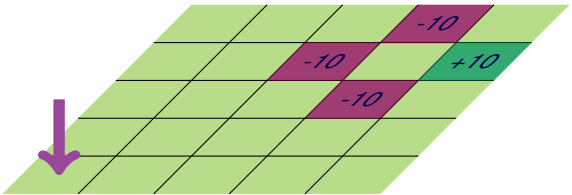
$$Q(s,a) = r + \gamma \times (\max(Q(s',a')))$$

  - r: Sofortiger Reward (z.B. bei ungültigen Zug oder Spielende)
  - γ: Discount-Factor

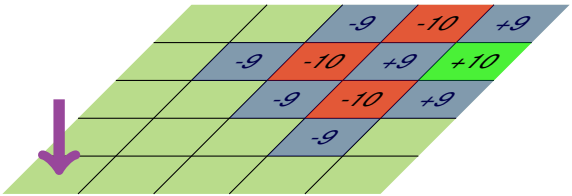
Q-Learning



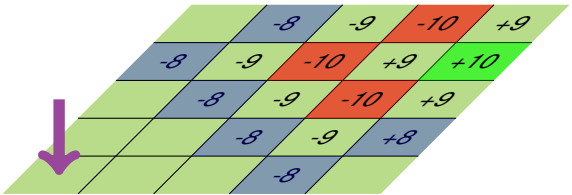
Q-Learning



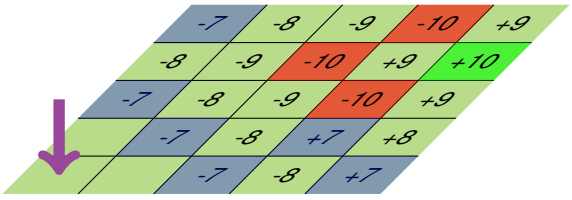
Q-Learning



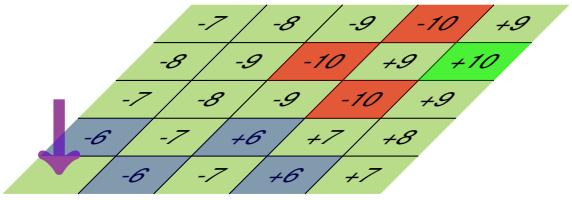
Q-Learning



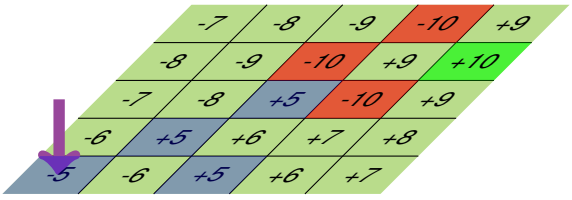
Q-Learning



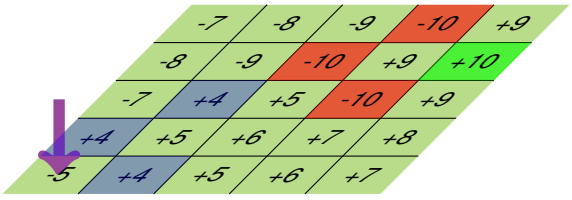
Q-Learning



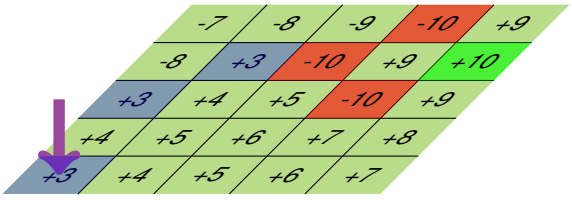
Q-Learning



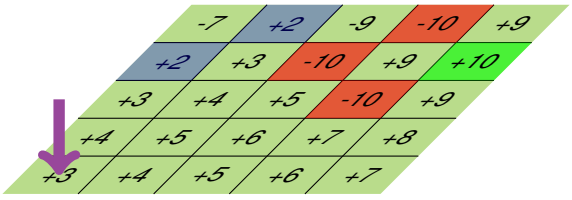
Q-Learning



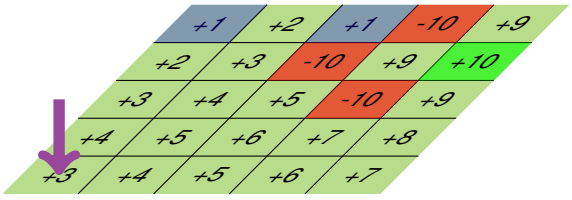
Q-Learning



Q-Learning



Q-Learning



Q-Learning

- Bei einfachen Problemen:
  - Alle Q-Werte berechnen (“Brute-Force”)

Q-Learning

- Bei einfachen Problemen:
  - Alle Q-Werte berechnen (“Brute-Force”)
- Bei komplexen Problemen:
  - Berechnung aller Q-Werte nicht möglich bzw. zu aufwändig
  - Neuronales Netz verwenden, um Q-Werte zu lernen

Deep-Q-Learning

- Q-Learning mit Hilfe eines Neuronalen Netzes mit mehreren Schichten

Deep-Q-Learning

- Q-Learning mit Hilfe eines Neuronalen Netzes mit mehreren Schichten
- ggf. zweites neuronales Netz zum Berechnen der Q-Werte (ein Netz für Aktion, ein Netz für Bewertung)

Deep-Q-Learning

- Q-Learning mit Hilfe eines Neuronalen Netzes mit mehreren Schichten
- ggf. zweites neuronales Netz zum Berechnen der Q-Werte (ein Netz für Aktion, ein Netz für Bewertung)
- ggf. Speichern gesammelter Erfahrungen in einen Puffer (s, a, s', v)

Deep-Q-Learning

- Q-Learning mit Hilfe eines Neuronalen Netzes mit mehreren Schichten
- ggf. zweites neuronales Netz zum Berechnen der Q-Werte (ein Netz für Aktion, ein Netz für Bewertung)
- ggf. Speichern gesammelter Erfahrungen in einen Puffer (s, a, s', v)
- für Videospiele: Zusätzliche Convolutional Layers zur Bildverarbeitung

A3C

- Asynchronous
  - Jeder Agent hat eine eigene Version der Umgebung, mit der er unabhängig von anderen Agents agieren kann
  - Mehrere Agents lernen parallel
  - Gelerntes wird in einem globalen Netz zusammengeführt

A3C

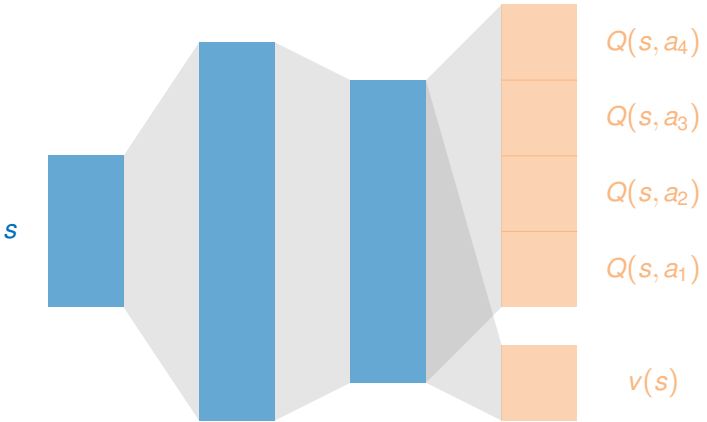
- Asynchronous
  - Jeder Agent hat eine eigene Version der Umgebung, mit der er unabhängig von anderen Agents agieren kann
  - Mehrere Agents lernen parallel
  - Gelerntes wird in einem globalen Netz zusammengeführt

- Actor-Critic
  - Actor-Teil: Policy Gradient
  - Critic-Teil: Value Function

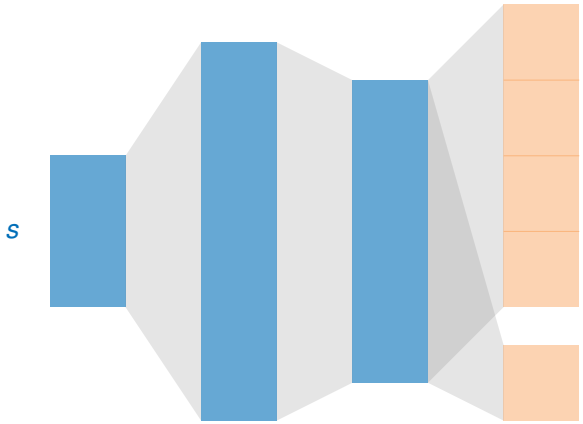
A3C

- Asynchronous
  - Jeder Agent hat eine eigene Version der Umgebung, mit der er unabhängig von anderen Agents agieren kann
  - Mehrere Agents lernen parallel
  - Gelerntes wird in einem globalen Netz zusammengeführt
- Advantage
  - Ein Zustand hat einen Q-Wert $v(s)$, der angibt, wie gut dieser Zustand ist
  - Eine Policy gibt die Q-Werte eines Zustands und einer Aktion an $\pi(s, a)$
  - Der Vorteil einer Aktion $A(a)$ ergibt sich aus: $A(a) = \pi(s, a) - v(s)$
- Actor-Critic
  - Actor-Teil: Policy Gradient
  - Critic-Teil: Value Function

A3C



A3C



$Q(s, a_4)$ $A(s, a_4) = Q(s, a_4) - v(s)$

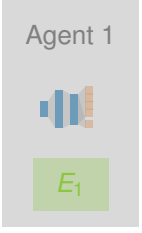
$Q(s, a_3)$ $A(s, a_3) = Q(s, a_3) - v(s)$

$Q(s, a_2)$ $A(s, a_2) = Q(s, a_2) - v(s)$

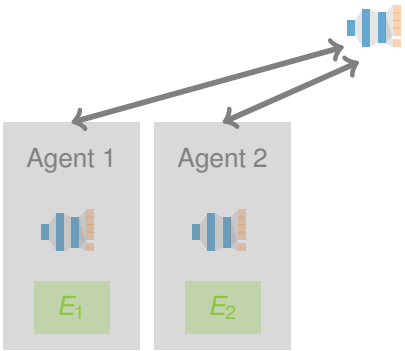
$Q(s, a_1)$ $A(s, a_1) = Q(s, a_1) - v(s)$

$v(s)$

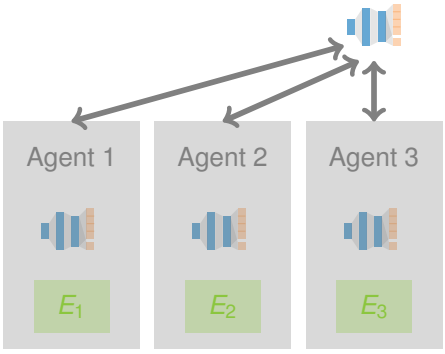
A3C



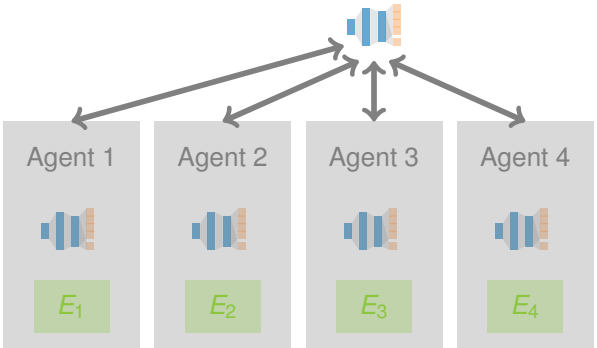
A3C



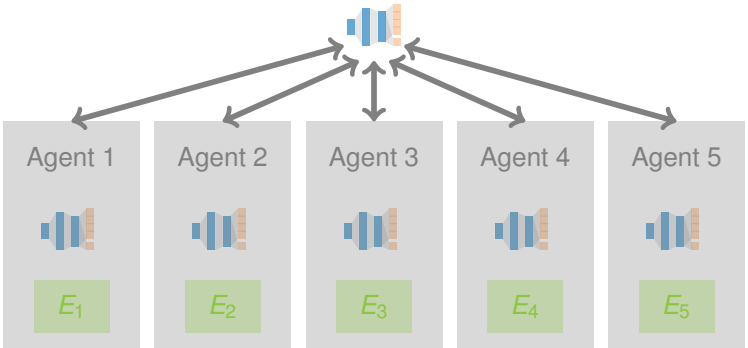
A3C



A3C



A3C



Quellen

- https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2
- https://arxiv.org/pdf/1602.01783.pdf
- https://jaromiru.com/2017/02/16/lets-make-an-a3c-theory/
- https://www.cs.rice.edu/~vardi/dag01/givan1.pdf