

TT2 Problemstellung 3 : Ausarbeitung

Pascal Jäger, Stefan Münchow, Armin Steudte,
Milena Rötting, Svend-Anjes Pahl, Carsten Noetzel, Oliver Steenbuck

01.06.2012

Inhaltsverzeichnis

1 Fragestellung 1: V-Werte / Q-Werte	1
2 Fragestellung 2: SARSA / Q-Learning bei kleinem ϵ	2
2.1 On-Policy: SARSA	2
2.2 Off-Policy: Q-Learning	2
2.3 Verhalten bei kleinem ϵ	3
3 Fragestellung 3: kontinuierliche Zustandsvariablen	3
3.1 Partitionierung	3
3.2 Abstraktion	3
3.3 Neuronale Netze	4
4 Fragestellung 4: Kniffel	4

Abbildungsverzeichnis

1 Sarsa: On Policy TD Control Algorithmus	2
2 Q Learning: Off Policy TD Control Algorithmus	3

1 Fragestellung 1: V-Werte / Q-Werte

Die Value-Function $V(s)$ definiert die mittlere Belohnung, wenn man einer Strategie π ab dem Zustand s folgt. Um die mittlere Belohnung zu bestimmen, ist es notwendig entweder eine Strategie π vorgegeben zu haben, welche bewertet werden soll (Policy Evaluation), oder es muss das Modell der Umwelt, das heißt die Zustandsübergangsfunktion δ und die Belohnungsfunktion r , bekannt sein. Ist das Modell der Umwelt bekannt, so kann mit Hilfe von beispielsweise der Value-Iteration die optimale Strategie π^* ermittelt werden (Policy Improvement).

Möchte man eine Strategie verbessern und besitzt kein Modell der Umwelt, so reicht die Bewertung einzelner Zustände nicht aus. Die Zustandsbewertung gibt keine Auskunft darüber die Wahl welcher Aktion zur Zustandsbewertung geführt hat. Man besitzt zwar die aktuelle Strategie π , diese wird jedoch kontinuierlich verbessert und damit ist kein Rückschluss einer Aktion auf eine Zustandsbewertung mehr möglich. (Wieso geht das bei Modell der Umwelt?)

Aus diesem Grund wird nicht nur die Bewertung des Zustands für die aktuelle Aktion der Strategie gespeichert, sondern es wird die Bewertung jeder Aktion für einen Zustand in einer Action-State-Funktion gespeichert. Auf diese Weise ist es möglich, die Strategie π zu verändern und das Wissen über die Bewertung einzelner Aktionen zu behalten. Durch die Speicherung der Bewertung zu jeder einzelnen Aktion vervielfacht sich der Speicheraufwand erheblich.

2 Fragestellung 2: SARSA / Q-Learning bei kleinem ϵ

2.1 On-Policy: SARSA

Basis für den SARSA-Algorithmus sind die Q-Werte, also Zustands-Aktions-Paare $Q(s, a)$. Die Bewertung eines solchen Zustands-Aktions-Paares erfolgt durch die unten genannte Gleichung welche auch in Abbildung 1 zu finden ist.

Der erwartete Wert $Q(s, a)$ für eine Aktion a im Zustand s wird dabei aktualisiert durch, den bereits ermittelten, erwarteten Wert $Q(s, a)$ und dem Schrittfaktor α , der mit der Summe aus Reward für den Folgezustand r_{t+1} und der Differenz aus dem discounted Reward für das Folge-Zustands-Aktions-Paar $\gamma Q(s_{t+1}, a_{t+1})$ und dem erwarteten Wert $Q(s, a)$ addiert wird.

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

In Abbildung 1 ist zu erkennen, dass die Aktionen a' im Zustand s' in Abhängigkeit von der Policy gewählt werden, die zum Beispiel ϵ -Greedy sein kann. Das heißt es wird die meiste Zeit diejenige Aktion a' ausgewählt die den größten erwarteten Gewinn verspricht und mit einer Wahrscheinlichkeit ϵ eine zufällige Aktionen, um die Umgebung weiter zu explorieren und nicht deterministisch zu sein.

```

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

```

Abbildung 1: Sarsa: On Policy TD Control Algorithmus

2.2 Off-Policy: Q-Learning

Auch das Q-Learning Verfahren setzt auf den Q-Werten auf und berechnet diese über die unten genannte Formel, die Teil des Algorithmus in Abbildung 2 ist.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (2)$$

Der Unterschied zum Sarsa-Algorithmus ist hierbei der Teil $\gamma \max_a Q(s_{t+1}, a)$. Anstelle den erwarteten Return $Q(s_{t+1}, a_{t+1})$ für die gewählte Aktion a' im Zustand s' mit dem Discount-Faktor γ zu multiplizieren wie es beim Sarsa-Algorithmus gemacht wird, wird hier diejenige Aktion bestimmt, die den größten Reward verspricht und deren Reward $\max_a Q(s_{t+1}, a)$ mit dem Discount-Faktor multipliziert. **Damit wird der Q-Wert $Q(s, a)$ unabhängig von der Strategie, allein auf Basis des Returns und der Q-Werte benachbarter Zustände aktualisiert!** Unabhängig davon, welche Aktion die ϵ -Greedy Policy wählt, wird immer der Q-Wert zur Berechnung genommen, die den höchsten Reward verspricht und nicht der, der durch die Policy gewählten Nachbarn.

```

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.,  $\epsilon$ -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

```

Abbildung 2: Q Learning: Off Policy TD Control Algorithmus

2.3 Verhalten bei kleinem ϵ

Das Verhalten vom Sarsa-Algorithmus nähert sich dem vom Q-Learning an, da mit kleiner werdendem ϵ die Wahrscheinlichkeit sinkt eine zufällige Aktion auszuwählen. Bei sehr kleinem ϵ wird vermehrt die Aktion a' ausgewählt, die den größten Reward verspricht, ähnlich wie es beim Q-Learning mit $\max_a Q(s_{t+1}, a)$ getan wird. Damit nähert sich das Verhalten von Sarsa dem des Q-Learnings an.

3 Fragestellung 3: kontinuierliche Zustandsvariablen

Für den Fall, dass die Zustandsvariablen kontinuierlich sind werden sie diskretisiert. Sollte der Zustandsraum durch die Diskretisierung zu groß werden, gibt es verschiedene Möglichkeiten, um den Zustandsraum zu verkleinern bzw. handhabbar zu machen.

3.1 Partitionierung

Bei der Partitionierung wird ein großer Zustandsraum in mehrere kleine Räume aufgeteilt. Ein Beispiel für die Partitionierung des Zustandsraumes ist Schach, bei dem der Zustandsraum in Eröffnung, Mittel- und Endspiel unterteilt werden kann.

3.2 Abstraktion

Ein Beispiel für die Abstraktion ist Kniffel, in dem bspw. die Reihenfolge in der die Würfel geworfen wurden ignoriert wird und sie immer der Größe nach sortiert werden, um den Zustandsraum zu verkleinern. In diesem Fall werden dadurch nicht einmal Informationen verloren.

3.3 Neuronale Netze

Die Tabelle der Q der V Werte kann durch ein neuronales Netz ersetzt werden, welches als Input die Variablen a und s erhält und als Ziel-Output den Q -Wert besitzt. Für jede Aktualisierung des Q -Werts gibt es ein Trainingsbeispiel mit genannten Inputs und Outputs. Durch das neuronale Netz kann die Funktion $Q(s, a)$, welche für unendlich viele Eingaben definiert ist, auf einen endlichen Raum reduziert werden. Statt eines neuronalen Netzes ist es auch möglich, andere überwachte Lernverfahren, wie beispielsweise Support Vector Machines zu verwenden, um den unendlichen Zustandsraum zu generalisieren und somit zu begrenzen.

Nachfolgendes
stammt aus
dem Ertel

4 Fragestellung 4: Kniffel

Dynamische Programmierung scheidet auf Grund des fehlenden Modells und der großen Zustandsmenge aus.

Q -Learning würde erst am Ende bewerten, bei Kniffel ist es aber wichtig die Zwischenschritte zu kennen?

Monte Carlo speichert bereits bewertete Paare und erlaubt daher die Bewertung jeden Schrittes. Ist aber speicheraufwendig. Auf Grund der Problembeschaffenheit müssen alle Verfahren zusätzlichen Aufwand betreiben, um den Zustandsraum klein zu bekommen.

http://www.haw-hamburg.de/fileadmin/user_upload/SchulCampus/Downloads/Arbeitskreis_Informatik/Kniffel_Agenten.pdf *Todo list

siehe den
Link

Nachfolgendes stammt aus dem Ertel	4
siehe den Link	4