

# (Deep) Reinforcement Learning - Policy Gradient

Thilo Stegemann

Hochschule für Technik und Wirtschaft  
Master Student der Angewandten Informatik  
12459 Berlin, Wilhelminenhofstraße 75A  
Email: t.stegemann@gmx.de

*Abstract*—The abstract goes here.

## I. EINFÜHRUNG

This demo file is intended to serve as a “starter file” for IEEE conference papers produced under L<sup>A</sup>T<sub>E</sub>X using IEEEtran.cls version 1.8b and later. I wish you the best of success.

mds

August 26, 2015

## II. REINFORCEMENT LEARNING (RL)

Sutton und Barto [1] definieren ein Standard Reinforcement Learning Framework, in diesem interagiert ein Agent mit einer Umgebung. Dieses standard RL-Framework wird nachfolgend ausführlich beschrieben, denn es ist eine elementare Grundlage für das Verständnis des Policy Gradient Verfahrens. Der Agent lernt, in einer ihm unbekannten Umgebung, durch Versuch und Irrtum eine Strategie  $\pi$  (eng. Policy). Der Agent kann in einer bestimmten Umgebung bestimmte Aktionen  $a$  ausführen. Ist die Menge der Aktionen begrenzt, dann ist es ein diskreter Aktionsraum  $A$ . Eine unbegrenzte Menge von Aktionen bezeichnet einen kontinuierlichen Aktionsraum. Die Umgebung bestimmt die Aktionsmenge und der Agent entscheidet welche Aktion aus dieser Menge ausgewählt wird. Eine Aktion kann eine Positionsangabe, eine Richtung oder etwas viel komplexeres sein.

### A. Markov-Entscheidungsprozess

Sutton und Barto [1] definieren die Umgebung als einen Markov-Entscheidungsprozess (eng. **Markov Decision Process**, kurz MDP). Der Agent interagiert demnach mit einem MDP und erhält von diesem einen Zustand und eine Belohnung. Das MDP erhält vom Agenten eine ausgewählte Aktion. Ein MDP ist ein sequentielles Entscheidungsproblem. Eine Sequenz  $\dots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2} \dots$  beschreibt die Interaktion des Agenten mit dem MDP. Ein Zustand ist eine Darstellung der Umwelt zu einem Zeitpunkt  $t$ . Die Zustände bestimmen, welche Aktionen ausgeführt werden können d.h. es existiert eine Funktion  $A(S_t)$ . Ergebnis dieser Funktion ist eine zulässige Menge von Aktionen  $A_t$  in einem Zustand  $S_t$ . Der Agent erhält eine direkte Belohnung  $R_{t+1}$  für das Ausführen einer Aktion  $A_t$  in einem Zustand  $S_t$ . Ein Beispiel: Es existiert ein Tic Tac Toe Spiel mit 9 Spielfeldern. Das Tic Tac Toe Spiel ist die Umgebung. Eine Darstellungsmöglichkeit

des Spielfelds ist z.B. eine 3x3 Matrix. Jedes Element der Matrix wird mit einem Leerzeichen ( ' ') initialisiert. Der Startzustand  $s_0 \in S$  ist eine 3x3 Matrix, indem jedes Element der Matrix ein Leerzeichen ist. Ein Spieler kann Kreuze ('X') in die Matrix einfügen und der andere Spieler Kreise ('O'). Der gesamte Zustandsraum  $S$  eines Tic Tac Toe Spiels, kann abgebildet werden. Die leeren Matrixelemente bestimmen die Positionen auf die Spielsteine gesetzt werden können.  $A(s_0) = \{(0,0), (0,1), \dots, (2,1), (2,2)\}$  und die Policy  $\pi(s_0) = a_0$  legt fest, welche Aktion  $a_0$  im Startzustand  $s_0$  ausgeführt werden soll ( $a_0 \in A(s_0)$ ). Eine direkte Belohnung  $R_{t+1}(s_t, a_t)$  ungleich 0 erhält der Agent für dieses Beispiel erst, wenn ein Spieler gewinnt oder verliert. Ein weiteres Beispiel: Ein Modellhubschrauber soll eigenständig lernen zu fliegen ohne abzustürzen und bestimmte Manöver auszuführen. In diesem Beispiel ist die Steuerung des Modellhubschraubers die zu lernende Strategie des RL Algorithmus. Eigenschaften der Umwelt sind unter anderem: Luftdruck, Position und Geschwindigkeit des Hubschraubers, Windgeschwindigkeit, Treibstoff. Die Modellierung der Zustände in diesem zweiten Beispiel ist ungleich komplexer verglichen mit dem ersten Beispiel. Zustände und Aktionen können sehr komplexe Objekte sein, dahingegen ist die Belohnung ein numerischer Wert und kein komplexes Objekt.

Die numerische Belohnung ist eine Bewertung der Aktion des Agenten. Der Agent versucht diese numerische Belohnung, über die Zeit, zu maximieren. Der Agent kann eine Belohnung von +10 erhalten (Hubschrauber um 360 Grad gedreht), wenn er in einem Zustand  $s$  eine Aktion  $a$  mit  $s \in S$  und  $a \in A$  ausführt. Der Agent kann auch eine negative numerische Belohnung von z.B. -23 erhalten (Hubschrauber abgestürzt). Der genaue Wert der Belohnung wird von der Umgebung definiert und der Agent kann diesen Wert nicht direkt beeinflussen oder verändern. Allein die Entscheidungen d.h. die Aktionsauswahl des Agenten entscheidet über die erhaltene Belohnung  $r$ . Die Umgebung definiert eine Belohnungsfunktion  $R_s^a = E\{r_{t+1} | s_t = s, a_t = a\}$ . Diese Funktion ist eine Abbildung von Zustand-Aktionspaaren auf erwartete Belohnungen.

In einer realen Umgebung kann es passieren, dass der Agent selbst (z.B. ein einfacher Laufroboter oder ein Hubschrauber) eine Signalstörung empfängt oder eine Fehlfunktion seiner mechanischen Steuereinheiten hat. Die Umgebungsgegeben-

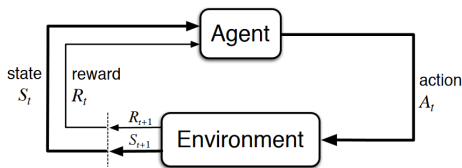


Fig. 1. Die Interaktion zwischen Agent und Umgebung nach [1].

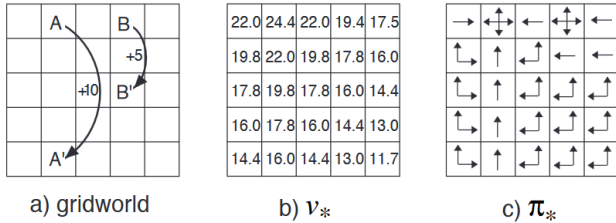


Fig. 2. Optimale Lösungen des Rasterwelt Beispiels nach [1].

heiten können sich ebenfalls verschieben oder verändern (z.B. wechselnde Windrichtungen, Windstärken-Schwankungen, andere autonom agierende Agenten). Man bezeichnet diese Möglichkeiten als die Dynamiken der Umgebung. Abgebildet werden diese Dynamiken mittels Wahrscheinlichkeiten. Die Zustandsübergangswahrscheinlichkeit das ein Agent in Zustand  $s_t$  mit der Aktion  $a_t$  in den Zustand  $s'_t$  übergeht ist definiert durch die Wahrscheinlichkeit  $P_{ss'}^a = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ . Die Umgebungsdynamik beeinflusst auch die Policy  $\pi$  und die Belohnungsfunktion  $R_s^a$ .

### B. Wertefunktionsapproximation

Die nachfolgenden Ausführungen basieren auf den Erläuterungen von D. Silver vgl. [4]: Eine nicht approximierte Wertefunktion ist eine Wertetabelle. In dieser Wertetabelle hat jeder Zustand  $s$  einen Eintrag der Form  $V(s)$  oder jedes Zustands-Aktionspaar  $s, a$  hat einen Eintrag  $Q(s, a)$ . Für große Markov-Entscheidungsprozesse existieren zu viele Zustände und Aktionen, um diese in der Wertetabelle speichern zu können. Das eigentliche Lernen der Wertefunktion ist zudem sehr langsam, weil für jeden Zustand ein individueller Wert gelernt werden muss. Die Lösung für große MDPs ist daher eine geschätzte (eng. estimated) Wertefunktion. Die Schätzung erfolgt mittels Funktionsapproximation:

$V_\theta(s) \approx V^\pi(s)$ , dabei ist  $V^\pi(s)$  die wahre Zustands-Wertefunktion, diese definiert wie viel erwartete Belohnung kann der Agent wirklich erhalten, wenn er der Strategie  $\pi$  folgt und sich in Zustand  $s$  befindet.

$Q_\theta(s, a) \approx Q^\pi(s, a)$ , dabei ist  $Q^\pi(s, a)$  die wahre Q-Funktion, welche für ein Zustands-Aktionspaar  $s, a$  definiert, wie hoch die wahre zu erwartende akkumulierte Belohnung ist, unter Berücksichtigung der Strategie  $\pi$ .

$V_\theta(s)$  und  $Q_\theta(s, a)$  sind Funktionsapproximationen, welche sich an die wahren Wertefunktionen annähern sollen. Die approximierten Funktionen generalisieren von bereits besuchten/bekannten Zuständen auf unbekannte/noch nicht be-

suchte Zustände. In der Regeln ist die Anzahl der Parameter im Parametervektor  $\theta$  viel kleiner als die Anzahl der Zustände des MDPs. Die Annäherung erfolgt durch die Modifikation des Parametervektors  $\theta$  (wird auch oft als  $w$  bezeichnet). Dieser Parametervektor wird mittels Monte-Carlo oder Temporal-Differenz gelernt. Im nächsten Abschnitt wird das gerade beschriebene Konzept der Funktionsapproximation auf die Strategiefunktion angewendet.

### C. Strategiefunktionsapproximation

Eine Strategie  $\pi$  (eng. Policy) definiert das Verhalten des Agenten als Funktion von Zuständen  $s$ , mit  $s \in S$  und  $S$  ist der gesamte Zustandsraum des MDPs. Bei allen Wertefunktions-basierten Methoden wird eine Wertefunktion gelernt und mittels dieser kann implizit eine Strategie erstellt werden (z.B. mit  $\epsilon - greedy$  vgl. [4]). Innerhalb dieser Arbeit werden speziell die Methoden erläutert, welche Strategien  $\pi_\theta$  (bzw. den Parametervektor  $\theta$ ) direkt lernen (ausgenommen der Actor-Critic-Methode, diese verwendet sowohl Strategie- als auch Wertefunktionen). Eine stochastische approximierte Strategiefunktion wird als  $\pi(s, a, \theta) = Pr\{a_t = a | s_t = s, \theta\}$  geschrieben. Die Strategie ist stochastisch (nicht-deterministisch) d.h. sie beschreibt Wahrscheinlichkeiten für das Auswählen von Aktionen. Eine deterministische (nicht-stochastische) Strategie legt jeweils exakt eine Aktion für einen Zustand fest. Eine weitere Beschreibung der Strategiefunktion wäre  $\pi_\theta(s, a) = P[a | s, \theta]$ , d.h.  $\pi_\theta(s, a)$  gibt die Wahrscheinlichkeitsverteilung über den möglichen Aktionen  $a$  im Zustand  $s$  an und durch den Parametervektor  $\theta$  kann diese Wahrscheinlichkeitsverteilung verändert werden. Das optimieren des Parametervektors  $\theta$  der Strategiefunktion  $\pi_\theta$  ist Ziel des Policy Gradient Verfahrens und wird im nächsten Abschnitt ausführlich behandelt.

## III. POLICY GRADIENT THEOREM

Die nachfolgenden Erläuterungen basieren auf der wissenschaftlichen Arbeit unter anderem von R. S. Sutton [2]. Der Agent soll eine approximierte optimale Strategie  $\pi_\theta(s, a)$  mittels des Policy Gradienten Verfahrens erlernen. Für ein besseres Verständnis des Policy Gradient Verfahrens wird zuerst die allgemeine gradientenbasierte lokale Optimierung erläutert und anschließend wird erklärt was eine Zielfunktion ist und wie diese dargestellt werden kann. Die letzten beiden Teilabschnitte beschreiben die Anwendung des Gradientenverfahrens auf die Zielfunktion (Policy Gradient) und eine Unterscheidung verschiedener Policy Gradient Methoden.

### A. Gradientenbasierte lokale Optimierung

Definition eines Gradienten nach G. Hoever [3, vgl. S. 213] und D. Silver [4]: In einem Gradienten werden die verschiedenen partiellen Ableitungen einer Zielfunktion  $J(\theta)$  zusammengefasst. Zu einer Zielfunktion  $J : \mathbb{R}^n \rightarrow \mathbb{R}$  heißt (falls die partiellen Ableitungen existieren)

$$\nabla f(x) := grad f(x) := \left( \frac{\partial f}{\partial x_1}(x), \dots, \frac{\partial f}{\partial x_n}(x) \right)$$

Gradient von  $f$  im Punkt  $x$ . ( $\nabla f$  wird "nabla  $f$ " gelesen.) Der Gradient einer Funktion weist in die Richtung des Steilsten Anstiegs. Senkrecht zum Gradienten ändert sich der Funktionswert nicht. G.Hoever [3, vgl. S. 214] liefert ebenfalls eine Erklärung des Gradientenverfahrens: Sucht man ausgehend von einem Startpunkt  $x^{(0)}$  eine Maximalstelle (Gradient Ascent), so geht man ein Stück in die Richtung des Gradienten, also

$$\begin{aligned}x^{(1)} &= x^{(0)} + \alpha_0 \cdot \nabla f(x^{(0)}), \\x^{(2)} &= x^{(1)} + \alpha_1 \cdot \nabla f(x^{(1)}),\end{aligned}$$

allgemein:

$$x^{(i+1)} = x^{(i)} + \alpha_i \cdot \nabla f(x^{(i)}).$$

Dabei beschreibt  $\alpha_i$  die Schrittweite. Die genaue Wahl dieser Schrittweite ist oft nicht ganz einfach, wird an dieser Stelle jedoch nicht näher erläutert. Sucht man eine Minimalstelle (Gradient Descent), so setzt man entsprechend

$$x^{(i+1)} = x^{(i)} - \alpha_i \cdot \nabla f(x^{(i)}).$$

Bei der univariaten und multivariaten Regression (siehe maschinelles Lernen: vorhersage von kontinuierlichen Werten [5]) wird das Gradienten Abstiegsverfahren verwendet, um eine Kostenfunktion zu minimieren. Diese Kostenfunktion berechnet die Qualität einer Hypothese (z.B. mittels dem Fehler der kleinsten Quadrate, eng. least-squared-errors). Bei Problemen des Reinforcement Learnings ist die Kostenfunktion definiert durch eine Zielfunktion  $\rho(\pi)$  und der Agent versucht eine Strategie zu erlernen für die, die Zielfunktion maximal wird. Der Agent verändert die Strategie mittels des Parametervektors  $\theta$  unter Verwendung eines Gradienten Anstiegsverfahrens. Vielmehr ist die Kostenfunktion bei RL Problemen eher als Gewinnfunktion oder Belohnungsfunktion zu bezeichnen, da diese maximiert und nicht minimiert werden soll. Wie genau sieht die mit dem Gradienten Anstiegsverfahren zu maximierende Funktion  $\rho(\pi)$  aus?

### B. Zielfunktion

Die Qualität (Performance) einer Policy berechnet die Zielfunktion (eng. objective function oder criterion)  $\rho(\pi)$ . R. S. Sutton [2] unterscheidet zwei verschiedene Formulierungen der Zielfunktion  $\rho(\pi)$ :

$$\begin{aligned}\rho_{avR}(\pi) &= \lim_{n \rightarrow \infty} \frac{1}{n} E\{r_1 + r_2 + \dots + r_n | \pi\} \\&= \sum_s d^\pi(s) \sum_a \pi(s, a) R_s^a,\end{aligned}$$

ist die durchschnittliche Belohnung, diese bewertet die Strategie bezüglich der zu erwartenden Langzeitbelohnung pro Schritt. Eine Langzeitbelohnung (eng. long-term reward) ist die Aufsummierung der Belohnungen für eine Entscheidungssequenz. Der Term  $d^\pi(s) = \lim_{t \rightarrow \infty} Pr\{s_t = s | s_0, \pi\}$  beschreibt die stationäre Verteilung der Zustände unter  $\pi$ , d.h.  $d^\pi(s)$  gibt an, wie hoch die Wahrscheinlichkeit dafür ist, jeden Zustand  $s_t$  zu erreichen, bedingt durch die Policy  $\pi$  und unabhängig vom Startzustand  $s_0$ , wenn  $t$  gegen unendlich

strebt. Sprachliche Formulierung der Funktion  $\rho_{avR}(\pi)$ : Es existiert eine Wahrscheinlichkeit, dass sich der Agent in Zustand  $s$  befindet und es existiert eine Wahrscheinlichkeit, dass der Agent eine Aktion  $a$  auswählt ( $a$  abhängig von  $\pi$ ). Berechnet wird die durchschnittliche Belohnung pro Zeitschritt, wobei die Belohnung  $R_s^a$  von den beiden vorher erwähnten Wahrscheinlichkeiten  $s$  und  $a$  abhängt. Der Wert eines Zustands-Aktionspaares, unter Berücksichtigung einer Policy  $\pi$ , ist für die Zielfunktion der durchschnittlichen Belohnung  $\rho_{avR}(\pi)$  wie folgt definiert:

$$Q^\pi(s, a) = \sum_{t=1}^{\infty} E\{r_t - \rho_{avR}(\pi) | s_0 = s, a_0 = a, \pi\},$$

mit  $\forall s \in S, a \in A$ . Die Funktion berechnet die Summe der Differenzen zwischen einer sofortigen Belohnung ( $r_t$ ) in jedem Zeitschritt  $t$  und der Zielfunktion  $\rho_{avR}(\pi)$ , also der durchschnittlichen Belohnung pro Zeitschritt, für ein Zustands-Aktionspaar unter Verwendung einer Policy  $\pi$ . R. S. Sutton notiert die Zielfunktion  $\rho_{avR}(\pi)$  als  $\rho(\pi)$ , in dieser Arbeit wird für die durchschnittliche erwartete Belohnung pro Zeitschritt die Notation  $\rho_{avR}(\pi)$  verwendet (für **average Reward** ähnlich D. Silver [4]). Die zweite Formulierung der Zielfunktion  $\rho(\pi)$  berücksichtigt einen genau festgelegten Startzustand  $s_0$  und ausschließlich von diesem Startzustand ausgehende Langzeitbelohnungen:

$$\rho_{s_0}(\pi) = E\left\{\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_0, \pi\right\}$$

und

$$Q^\pi(s, a) = E\left\{\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | s_t = s, a_t = a, \pi\right\}.$$

Dabei ist  $\gamma \in [0, 1]$  ein Abschwächungsfaktor ( $\gamma = 1$  ist nur in endlichen abzählbaren Sequenzen einzusetzen). Die stationäre Verteilung der Zustände  $d^\pi(s)$  ist bei der Startzustandsformulierung eine abgeschwächte Gewichtung der vorgefundenen Zustände angefangen in Zustand  $s_0$  und unter Berücksichtigung der Policy  $\pi$ :  $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr\{s_t = s | s_0, \pi\}$ .

### C. Policy Gradient

$$\Delta\theta \approx \alpha \frac{\partial \rho}{\partial \theta} \quad (1)$$

$$\frac{\partial \rho}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s, a)}{\partial \theta} Q^\pi(s, a) \quad (2)$$

## IV. POLICY GRADIENT METHODEN

### A. Finite Difference Policy Gradient

### B. Monte-Carlo Policy Gradient (REINFORCE)

### C. Actor-Critic Policy Gradient

## REFERENCES

- [1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts, London, England: MIT Press, 2012.

- [2] R. S. Sutton and D. McAllester and S. Singh and Y. Mansour, *Policy Gradient Methods for Reinforcement Learning with Function Approximation*, 180 Park Avenue, Florham Park, NY 07932: AT&T Labs, 1999.
- [3] G. Hoever, *Höhere Mathematik kompakt*, 2. Auflage, Springer Spektrum, 2014
- [4] D. Silver, *Online Course Reinforcement Learning*, <http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>, London, England: Google Deep Mind, 2015
- [5] I. Goodfellow and Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.