Čebyšëv-Approximation im Verstärkenden Lernen

Jannik Schürg

3. August 2015

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Jochen Garcke

Zweitgutachter: Prof. Dr. Marc Alexander Schweitzer

Institut für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Danksagung

Das Maschinelle Lernen und speziell die Entwicklung des Verstärkenden Lernens steckt voller Ideen, die elegant, einfach und zugleich sehr erfolgreich sind. Die Fülle an beeindruckenden und vielfältigen Anwendungen in diversen Disziplinen sprechen für sich. Vielen Dank an Herrn Prof. Dr. Garcke für den Vorschlag zu diesem außergewöhnlichen Thema und die gute Betreuung.

Inhaltsverzeichnis

1. Einleitung

Maschinen haben Schwierigkeiten beim Lösen von vielen interessanten Aufgaben, die Menschen hingegen sehr effizient beherrschen. So ist das Gehirn beispielsweise in der Lage, innerhalb von Bruchteilen einer Sekunde ein bekanntes Gesicht wiederzuerkennen, obwohl es möglicherweise viele Jahre gealtert ist. Ganz allgemein sind Lebewesen in der Lage, sehr schnell relevante Informationen aus optischen, akkustischen und sensorischen Eindrücken zu filtern um sie für das Lösen einer gestellten Aufgabe zu benutzen — zum Beispiel Autofahren. Sogar das Lösen der Aufgaben selbst wird erst erlernt: Neue Verhaltensweisen, die zu einer effizienten Lösung führen, können *gelernt* werden, d.h. der Fundus an Verhalten wird um neues, selbstständig erschaffenes Material erweitert und kann sogar auf verwandte Probleme übertragen werden.

Der Natur ist mit dem Gehirn eine Konstruktion gelungen, die Menschen seit langer Zeit in einer Maschine zu reproduzieren versuchen. Zum einen für ein besseres Verständnis, zum anderen um uns die universellen Eigenschaften und Fähigkeiten zu Nutze zu machen und das Vorbild aus der Natur sogar an Kapazität zu übertreffen. In dieser Allgemeinheit ist dies zwar noch nicht absehbar, jedoch werden seit den 50ern stetig Erfolge in diese Richtung verbucht. Zahlreiche Teilprobleme werden untersucht und bilden heute Teile des Spektrums *Künstliche Intelligenz*.

Maschinelles Lernen Im Jahr 1996 gelang es dem Computer Deep Blue den amtierenden Schachweltmeister zu schlagen. Die Überlegenheit von Deep Blue liegt in den schnellen und parallelen Rechenfähigkeiten und der sorgfältigen *Programmierung*, in welche zahlreiche Erkenntnisse aus Schachpartien eingeflossen sind. Die Maschine selbst hat, grob gesagt, lediglich die schnelle Ausführung von Vorberechnungen und Bewertungsfunktionen beigesteuert. Zudem wurde zwischen den Partien das Verhalten der Maschine händisch vom Team angepasst und verändert — Das Maschinelle Lernen versucht einen anderen Weg zu gehen. Angewandt auf die

Schachpartie von Deep Blue, könnte man sagen dem Computer wird nun auch die Aufgabe der Programmierung übertragen. Etwas formaler gesagt, erhält der Computer eine Beschreibung des Systems (Schachbrett, Figuren, mögliche Züge) und bekommt mitgeteilt, was das Ziel ist (eine Siegposition erreichen). Jetzt soll der Computer aus den gegebenen Daten eine optimale Strategie selbstständig *programmieren*. Die Frage im Maschinellen Lernen ist: Wie geschieht das?

Ein Zweig des Maschinellen Lernens beginnt mit der Idee, der Maschine Lösungen vorzugeben und sich so auf die Entwicklung von Algorithmen zu konzentrieren, die in diesen Daten *Verknüpfungen von Mustern* erkennen können, um damit eine Vorhersagefähigkeit für neue Daten zu entwickeln (die Vorhersage entspricht dann einem Lösungsversuch).

Eine weitere Möglichkeit ist die Angabe von Belohnungen, die bewerten wie gut ein erreichter Zustand ist. So wäre eine Siegposition im Schach mit einer Belohnung versehen. Es wird nun nach Algorithmen gesucht, die in der Lage sind, möglichst hohe Belohnungen zu erreichen. Dabei wird auf die Vorgabe von Lösungen verzichtet — die Maschine muss per *try and error* einen Weg finden und kann nur an Hand der erhaltenen Belohnungen arbeiten. Diese Form des Lernens ist auch in menschlichem Verhalten zu beobachten. Die Bestärkung in Form von Belohnungen und das daraus folgende Erlernen einer belohnungsmaximierenden Strategie wird *Verstärkendes Lernen* genannt.

Wie erfolgreich das sein kann, zeigt ein vielbeachtetes Beispiel von Tesauro aus den 90ern [Tes95]: Was Deep Blue im Schach geleistet hat, hat das Programm TD-Gammon für das Brettspiel Backgammon versucht. Mit dem Unterschied, dass die Strategie durch Verstärkendes Lernen berechnet wurde und so in einem gewissem Sinne weniger händische Vorarbeit nötig war. TD-Gammon hat Weltklasseniveau erreicht und das im Gegensatz zu Deep Blue ohne einen Lehrer oder eine Lehrerin in Backgammon (*Programmierung*). Eine Belohnung war auf Siegpositionen gesetzt. Durch diese unvoreingenommene Herangehensweise wurden Spielstrategien für Backgammon entdeckt, die Menschen bisher übersehen hatten.

Ein weiteres Beispiel ist eine Anfang des Jahres erschienene, polarisierende Veröffentlichung, wo mit Hilfe von Verstärkendem Lernen Strategien für 49 klassische Atari-Spiele (Space Invaders...) berechnet wurden[1], die erfahrene

[1] `https://youtu.be/iqXKQf2BOSE`

menschliche Spieler schlugen [Mni+15]. Dabei erhielt die Maschine als Eingabe lediglich ein Foto des Bildschirms und die möglichen Tasten. Als Belohnung wurde die erreichte Punktzahl benutzt.

Idee der Verfahren In dieser Arbeit werden wir wichtige Algorithmen, die zum Lösen von Problemen im Verstärkenden Lernen entwickelt wurden, präsentieren und eine Variation davon testen. Die Algorithmen funktionieren dabei anschaulich nach folgender Vorgehensweise: Beim Durchlaufen der verschiedenen Zustände des Systems, verknüpft die Maschine die erhaltenen Belohnungen mit den entsprechenden Zuständen. Steht die Maschine nun erneut vor der Aufgabe, kann sie die gewonnen Einschätzungen der zu erwartenden Belohnungen benutzen um eine Strategie zu entwickeln: »Gehe dahin, wo die höchste Belohnung erwartet wird«. Dabei werden die Einschätzungen entweder bestärkt oder korrigiert, so dass eine mögliche Korrektur der Strategie beim nächsten Start möglich ist. Die Details werden in Kapitel 2 ausgearbeitet.

Eines der dabei auftretenden Probleme ist, dass die Anzahl an möglichen Zuständen zu groß ist, um u.a. für alle eine Einschätzung abzuspeichern. Es sind Methoden nötig, die die Einschätzungen in komprimierter Form möglichst gut darstellen. Die erwähnte Variation, welche wir hier beschreiben und untersuchen möchten, bezieht sich auf eine solche Methode.

Mathematische Zusammenfassung Wir beschreiben Linearkombinationen von Čebyšëv-Polynomen als Approximation der Wert-Funktion im Verstärkenden Lernen und vergleichen sie mit der in [KOT11] vorgeschlagenen und verwandten Fourier-Basis. Dies wird experimentell ebenfalls in Kombination mit einer Ausdünnung der Basis mittels Hyperbolischen Kreuzes untersucht.

2. Verstärkendes Lernen

In diesem Kapitel werden wir den Algorithmus beschreiben und herleiten, der für diese Arbeit relevant ist. Dabei wird auf eine maßtheoretisch völlig rigorose Handhabung verzichtet und der Fokus auf die Entwicklung der Algorithmen gelegt. Für fehlende Details ist ein Literaturverweis angegeben. Die strukturelle Vorgehensweise und Begriffe sind aus [SB98; BT96; Ber95b; Ber95a] übernommen.

Wir werden im folgenden Abschnitt die betrachtenden Probleme formal definieren und danach zunächst die Lösung mit Hilfe von *Dynamischer Programmierung* betrachten. Aus der Bellman-Gleichung erhalten wir die Verfahren zur *Wert-* und *Strategie-Iteration*, welche im Verstärkenden Lernen eine grundlegende Rolle spielen. Schließlich wird über den Vergleich zwischen *Monte-Carlo-* und *TD-Verfahren* die Funktionsweise und Motivation des SARSA(λ)-Algorithmus deutlich. Den Abschluss bildet die Beschreibung einer adaptiven Schrittweitensteuerung und der linearen Funktionsapproximation der Wert-Funktion im SARSA(λ)-Algorithmus.

2.1. Modellierung des Problems

Beginnen wir zunächst mit einer groben Beschreibung des zu modellierenden Problems: Wir betrachten ein System mit verschiedenen *Zuständen* $s \in S$, das in einer Folge von diskreten Zeitschritten durch jeweilige Wahl einer *Aktion* $a \in A$ in einen neuen Zustand wechselt. Der neue Zustand muss nicht deterministisch vorhersehbar sein, sondern wird über eine Wahrscheinlichkeitsverteilung $\mathcal{P}$ beschrieben.

Am Anfang befindet sich das System in einem Startzustand $s_0 \in S$. Pro Zeitschritt müssen wir nun eine *Aktion* a_0 aus einer Menge von zulässigen Aktionen A_{s_0} für den Zustand s_0 wählen. Die gewählte Aktion bringt das System in einen neuen Zustand s_1 und einen Zeitschritt weiter. Die Möglichkeiten für s_1 sind durch die Wahrscheinlichkeitsverteilung $\mathcal{P}(s_0, a_0, \cdot)$ gegeben, die angibt wie $s_1 \in S$ in Abhängigkeit von s_0 und a_0 verteilt ist.

Im k-ten Zeitschritt befinden sich das System also in einem Zustand s_k, wir wählen eine Aktion $a_k \in A_{s_k}$ und das System wechselt in den Zustand s_{k+1}, verteilt nach $\mathcal{P}(s_k, a_k, \cdot)$. Außerdem erhalten wir für jede gewählte Aktion eine Belohnung $\mathcal{R}(s_k, a_k, s_{k+1}) \in \mathbb{R}$.

Das Problem besteht nun darin, unter Kenntnis des aktuellen Zustandes s_k und ggf. von $\mathcal{P}$ zu lernen, die Aktionen a_k so zu wählen, dass die Summe der Belohnungen[1] maximal wird.

Diese Beschreibung mathematisch zu präzisieren, ist Ziel des Abschnittes. Von den zahlreichen Varianten, die für dieses Problem existieren, wurde hier eine Auswahl getroffen, die für die Arbeit relevant ist. Die eingeführte Notation behält im Folgenden ihre obige Interpretation bei.

Grundlegende Strukturen

Für die folgenden Definition ist der Begriff des *stochastischen Kerns* wichtig, welcher eine Verallgemeinerung der Übergangsmatrix in *Markov-Ketten* (s.u.) auf allgemeine (überabzählbare) Räume darstellt. Für die Darstellung der Algorithmen spielt nur die Eigenschaft (i) der folgenden Definition eine Rolle.

Definition 2.1.1. Seien $(\Omega, \mathcal{A})$ und $(\Omega', \mathcal{A}')$ Messräume. Eine Abbildung $K \colon \Omega \times \mathcal{A}' \to [0, 1]$ heißt *stochastischer Kern*, falls gilt:

(i) Für alle $x \in \Omega$ ist $K(x, \cdot)$ ein Wahrscheinlichkeitsmaß auf $(\Omega', \mathcal{A}')$.

(ii) Für alle $A \in \Omega'$ ist $K(\cdot, A)$ eine Ω-messbare Funktion.

Nun können wir den zentralen Begriff zur Modellierung von Problemen im Verstärkenden Lernen definieren.

Definition 2.1.2. Sei $S \subseteq \mathbb{R}^d$ $(d \in \mathbb{N})$ ein polnischer Raum[a] mit Borelscher σ-Algebra $\mathcal{B}(S)$ sowie $A \neq \emptyset$ eine endliche Menge. Ein *Markov-Entscheidungsprozess* M ist ein Tupel

$$M = (S, A, \{A_s\}_{s \in S}, \mathcal{P}, \mathcal{R})$$

[1] Durch einen Vorzeichenwechsel ist die Minimierung von Kosten, man könnte sagen *Schmerzen*, ebenfalls möglich. Der Belohnungsgedanke ist allerdings etwas moderner als das *Lernen durch Schmerzen*.

wobei $\emptyset \neq A_s \subseteq A$ für alle $s \in S$, $\mathcal{P} : (S \times A) \times \mathcal{B}(S) \to [0,1]$ ein stochastischer Kern[b] und $\mathcal{R} \colon \Gamma \times S \to \mathbb{R}$ ist, wobei

$$\Gamma \coloneqq \{(s,a) \mid s \in S,\ a \in A_s\}$$

M heißt deterministisch, falls es für alle $(s,a) \in S \times A$ ein $s' \in S$ gibt mit $\mathcal{P}(s,a,\{s'\}) = 1$.

[a] D.h. S ist ein separabler und vollständig metrisierbarer topologischer Raum. Bspw. $\mathbb{N}^n$, $\mathbb{Z}^n$ oder $\mathbb{R}^n$ mit Diskreter bzw. Standard-Topologie.

[b] Dabei ist $S \times A$ mit der Produkt-Topologie versehen und A mit der Potenzmenge als σ-Algebra.

Bemerkung. Um die allgemeine Theorie sauber aufbauen zu können, müssten an Γ zusätzliche Messbarkeitbedingungen gestellt werden [BS96, S. 188f]. Auch für $\mathcal{R}$ sind zusätzliche Bedingungen und Fallunterscheidungen nötig, je nach untersuchtem Problem (s.u.) [BS96, S. 213f]. Diese Art von Bemerkung wiederholt sich in den weiteren Definition und Resultaten und es sei hier stellvertretend auf [BS96] für eine rigorose Beschreibung und Untersuchung hingewiesen, wenn nichts anderes angegeben ist.

Wir nutzen die abkürzende Notation

$$\begin{aligned} \mathcal{P}(s,a,\{s'\}) &= p_{ss'}(a) \\ \mathcal{R}(s,a,s') &= r_{ss'}(a) \end{aligned} \tag{2.1}$$

und lassen bei der Nennung von M in Zukunft das Tupel weg.

Für die Wahl der Aktionen a_k definiert man den Begriff der Strategie[2].

Definition 2.1.3 ([BS96, S. 190])**.** Sei M ein Markov-Entscheidungsprozess und $\mathcal{A} = \{X \mid X \subset A\}$ die Potenzmenge von A. Eine *Strategie* ist eine Folge $\pi = \{\mu_i\}_{i=0}^{\infty}$ wobei

$$\mu_k \colon S \times \mathcal{A} \to [0,1]$$

ein stochastischer Kern mit $\mu_k(s, A_s) = 1$ für alle $k \in \mathbb{N}$ und $s \in S$ ist. Eine Strategie $\pi = \{\mu_i\}_{i=0}^{\infty}$ heißt *stationär*, falls $\mu_0 = \mu_i$ für alle $i \in \mathbb{N}$ gilt. In diesem Fall identifizieren wir π mit μ_0 in der Notation.

Falls es für alle k ein $(s,a) \in \Gamma$ gibt mit $\mu_k(s,\{a\}) = 1$, so nennt

[2] In der Literatur *policy*.

man die Strategie *deterministisch*. In diesem Fall fassen wir μ_k auf als Funktion $\mu_k \colon S \to A_k$ mit $\mu_k(s) = a$.

Die Menge aller Strategien für M nennen wir Π_M. Ferner sei

$$\begin{aligned}\Pi_M^{\mathrm{s}} &= \{\pi \in \Pi_M \mid \pi \text{ ist stationär}\}\\ \Pi_M^{\mathrm{d}} &= \{\pi \in \Pi_M \mid \pi \text{ ist deterministisch}\}\\ \Pi_M^{\mathrm{sd}} &= \Pi_M^{\mathrm{s}} \cap \Pi_M^{\mathrm{d}}\end{aligned}$$

Mit einer Strategie π und einem Markov-Entscheidungsprozess M lässt sich ein *Markov-Prozess* bzw. eine *Markov-Kette* konstruieren.

Definition 2.1.4 ([Kle13, Def. 17.3])**.** Sei $S \subseteq \mathbb{R}^d$ ($d \in \mathbb{N}$) ein polnischer Raum mit Borelscher σ-Algebra $\mathcal{B}(S)$ und $X = \{X_k\}_{k=0}^{\infty}$ eine Folge von Zufallsvariablen in S. Außerdem sei $\{\mathrm{P}_s\}_{s\in S}$ eine Familie von Wahrscheinlichkeitsmaßen. Dann heißt X *Markov-Prozess*, falls gilt:

(i) Für jedes $s \in S$ ist $\mathrm{P}_s[X_0 = s] = 1$.

(ii) Die Abbildung $K \colon S \times \mathcal{B}(S)^{\otimes \mathbb{N}_0} \to [0,1]$, $(s, B) \mapsto \mathrm{P}_s[X \in B]$ ist ein stochastischer Kern.

Gilt zusätzlich

(iii) Für jedes $A \in \mathcal{B}(S)$, jedes $s \in S$ und alle $i, j \in \mathbb{N}_0$ ist

$$\mathrm{P}_s[X_{i+j} \in A \mid \sigma(X_k, k < j)] = K_i(X_j, A) \quad \mathrm{P}_s\text{-f.s.} \tag{2.2}$$

mit

$$K_i(s, A) \coloneqq K(s, \{y \in S^{\mathbb{N}_0} \mid y_i \in A\}) = \mathrm{P}_s[X_i \in A]$$

so heißt M *Markov-Kette*.

Bemerkung. Die Gleichung 2.2 heißt *Markov-Eigenschaft*. Für die Definition und Details der bedingten Wahrscheinlichkeit $P[X_{i+j} \in A \mid \sigma(X_k, k < j)]$ siehe [Kle13, S. 177]. Auch den dort definierten, bedingten Erwartungswert werden wir später benutzen.

Die Markov-Eigenschaft bedeutet, dass die Wahrscheinlich für einen Zustand X_{i+j} zum Zeitpunkt $i + j$ — gegeben der ersten j Zustände — bis

auf Nullmenge gleich

$$\mathrm{P}_{X_j}[X_i \in A]$$

ist. D.h. als wäre wegen (i) die Kette mit dem Zustand X_j gestartet. Ist S abzählbar, so kann stattdessen gefordert werden, dass für alle $n \in \mathbb{N}$, alle

$$k_1 < \cdots < k_n < k \in \mathbb{N}_0$$

und $s_1, \ldots, s_n, s \in S$ gilt:

$$P[X_k = s \mid X_{k_1} = s_1, \ldots, X_{k_n} = s_n] = P[X_k = s \mid X_{k_n} = s_n]$$

Hat man nun eine Strategie $\pi = \{\mu_k\}_{k=0}^{\infty}$ und einen Markov-Entscheidungsprozess M, so lässt sich mit den stochastischen Kernen $\{K_k\}_{k=0}^{\infty}$

$$\begin{aligned} K_k \colon S \times \mathcal{B}(S) &\to [0,1] \\ (s, B) &\mapsto \sum_{a \in A_k} \mu_k(s, \{a\})\, \mathcal{P}(s, a, B) \end{aligned}$$

ein Markov-Prozess $(x_k)_{k=0}^{\infty}$ auf S konstruieren mit weiteren Zufallsvariablen[3] $(u_k)_{k=0}^{\infty}$, wobei x_k den Zustand und u_k die gewählte Aktion beschreiben. Diese Notation gilt im Folgenden als Konvention. Die Erwartungswerte für einen zur Strategie π gehörenden Markov-Prozess mit Startwert s werden mit $\mathbb{E}_\pi[\,\cdot \mid x_0 = s\,]$ notiert. Falls π stationär ist, ist M eine Markov-Kette.

Die obige formale Definition einer Markov-Kette — welche zur wahrscheinlichkeitstheoretischen Formulierung des Systems bei einer gegebenen Strategie benutzt wird — ist für diese Arbeit lediglich zur Illustration der stochastischen Hintergründe und zur Verknüpfung mit der Theorie vorhanden, da alle nachfolgenden Resultate ohne genaue Angabe der nötigen technischen Voraussetzungen dargestellt werden und somit die Details der Definition in den Hintergrund rücken. Der Zweck dieses ersten, formelleren Abschnittes ist es, mit Hilfe der zitierten Literatur die Möglichkeit zu bieten, die Theorie parallel nachzuvollziehen — so weit rigorose Theorie überhaupt existiert.

Wir haben nun genug Struktur, um uns im nächsten Schritt der Formulierung des oben beschriebenen Maximierungs-Problem zu widmen.

[3] Der Messraum der Maße P_s wird entsprechend konstruiert, dass dort Zustand-Aktion-Abfolgen modelliert werden können.

Formulierung des Maximierungs-Problem

Wir können unter gewissen Voraussetzungen für eine Strategie $\pi = \{\mu_k\}_{k=0}^{\infty}$ und $\gamma \in (0, 1]$ den Erwartungswert für die zu erhaltenden Belohnungen

$$V_N^\pi(s) \coloneqq \mathbb{E}_\pi \left[\sum_{k=0}^{N-1} \gamma^k \mathcal{R}(x_k, u_k, x_{k+1}) \,\middle|\, x_0 = s \right]$$

in den ersten N Schritten und für die Startzustände $s \in S$ betrachten. Dabei modelliert $\gamma < 1$ den Fall, dass uns weit in der Zukunft liegende Belohnungen weniger bzw. zeitnahe Belohnungen stärker interessieren. Ist $\gamma = 1$, so spricht man vom *Stochastischen Kürzeste-Wege-Problem.*

Unter weiteren Voraussetzungen existiert neben $V_N^\pi(s)$ und dem folgenden Supremum

$$V_N^*(s) \coloneqq \sup_{\pi \in \Pi_M} V_N^\pi(s)$$

auch der für uns interessante Grenzfall $N \to \infty$ und wir dürfen unter Zusatzbedingungen die Grenzwerte vertauschen:

$$V^\pi(s) \coloneqq \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}(x_k, u_k, x_{k+1}) \,\middle|\, x_0 = s \right] \tag{2.3}$$

$$V^*(s) \coloneqq \sup_{\pi \in \Pi_M} V^\pi(s) \tag{2.4}$$

Diese Funktionen (2.3) und (2.4) spielen eine zentrale Rolle in den beschriebenen Verfahren. Die Funktion $V^*(s)$ heißt *Wert-Funktion*[4] für den Zustand s. Sie gibt an, was die optimal zu erreichende Belohnung für einen Start in s ist.

Das Problem lässt sich nun wie folgt formulieren:

Definition 2.1.5. Sei $\gamma \in (0, 1]$ und M ein Markov-Entscheidungsprozess. Wir bezeichnen das Finden einer Strategie π mit $V^\pi(s) = V^*(s)$ für alle $s \in S$ als *Reinforcement-Learning-Problem.*

Die Existenz solcher *optimalen* Strategien, lässt sich unter gewissen Voraussetzungen zeigen. In einigen Fällen ist nur die Existenz von *ε-optimalen* Strategien bekannt, die das Supremum beliebig gut approximieren.

[4] In der Literatur: *value function.*

Insgesamt lässt sich für die Modellierung und die zahlreichen Hinweise auf weitere Voraussetzungen sagen, dass endliche oder abzählbar-unendliche Zustandsräume S in Verbindung mit einer endlichen Schrittzahl oder $\gamma < 1$ in der Regel keine großen Schwierigkeiten verursachen. Für diese S sind die meisten getroffenen Aussagen hier und im Folgenden unproblematisch und mit grundlegenden Kenntnissen zu zeigen. Schwierigkeiten verursachen überabzählbare S und der Fall unendlicher Schrittzahlen. Es sind maßtheoretische und analytische Anstrengungen nötig und die Verbindung mit $\gamma = 1$ erschwert die Situation. Wir befassen uns im Abschnitt 2.5.1 mit der algorithmischen Handhabung großer S.

2.2. Dynamische Programmierung

Die Dynamische Programmierung ist das älteste hier vorgestellte Verfahren zur Lösung von Optimierungsproblemen der beschriebenen Art und wurde von Richard Bellmann in den 50ern als Prinzip erkannt und popularisiert. Bis heute ist die Idee sehr erfolgreich und auf eine Vielzahl von Bereichen erweitert worden. Das zugrunde liegende Optimalitätsprinzip (*Principle of Optimality*) lautet:

> An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions. ([Bel57, S. 83])

Im Falle des Kürzesten-Wege-Problems in der Graphentheorie, nutzt der Bellman-Ford-Algorithmus dieses Prinzip, konkret die Tatsache, dass für einen kürzesten s–t-Weg, jeder Teilweg v–t ebenfalls ein kürzester Weg ist.

Sei M ein Markov-Entscheidungsprozess. Wir beschränken uns auf den Fall deterministischer Strategien, da für viele Probleme eine optimale deterministische Strategie existiert (siehe bspw. [Put94, S. 134ff]). Die Wert-Funktion nach N Zeitschritten ist dann:

$$V_N^*(s) = \max_{\pi \in \Pi_M^{\mathrm{d}}} \mathbb{E}_\pi \left[\sum_{k=0}^{N-1} \gamma^k \mathcal{R}(x_k, \mu_k(x_k), x_{k+1}) \,\middle|\, x_0 = s \right] \tag{2.5}$$

Für $N = 1$ lässt sich $V_1^*(s)$ ausrechnen:

$$V_1^*(s) = \max_{u \in A_s} \mathbb{E}\big[\, \mathcal{R}(s, u, x_1) \mid x_0 = s, u_0 = u \,\big]$$

Das Optimalitätsprinzip liefert nun die Idee, mit der Wahl für die Nte-Aktion zu beginnen und dann rückwärts alle vorherigen Aktionen zu berechnen. In der Tat zeigt man durch Aufteilen der Strategien in Teilstrategien und mit Hilfe der genauen Konstruktion des Markov-Prozesses, dass sich (2.5) rekursiv schreiben lässt als

$$V_{k+1}^*(s) = \max_{u \in A_s} \mathbb{E}\big[\, \mathcal{R}(s,u,x_1) + \gamma V_k^*(x_1) \;\big|\; x_0 = s, u_0 = u \,\big]$$

Unter zusätzlichen Annahmen, die u.a. Konvergenz, Wohldefiniertheit und Vertauschung von Grenzwerten sichern, erhält man

$$\begin{aligned} V^*(s) &= \lim_{k\to\infty} V_k^*(s) \\ &= \max_{u \in A_s} \mathbb{E}\big[\, \mathcal{R}(s,u,x_1) + \gamma V^*(x_1) \;\big|\; x_0 = s, u_0 = u \,\big] \end{aligned} \tag{2.6}$$

Die Gleichung (2.6) heißt *Bellman-Gleichung* und gilt in vielen betrachteten Problemen, auch die Wahl einer Strategie, die jeweils rechts das Maximum annimmt, ist dort eine optimale Stratgie (siehe Theorem 2.2). Wie man sieht, ist diese im Fall von (2.6) sogar stationär.

Auch für die Wert-Funktion V^π lässt sich mit $\pi \in \Pi_M^{\mathrm{sd}}$ auf gleiche Weise eine Gleichung aufstellen:

$$\begin{aligned} V_{k+1}^\pi(s) &= \mathbb{E}_\pi \left[\sum_{i=0}^{k} \gamma^i \mathcal{R}(x_i, \pi(x_i), x_{i+1}) \;\middle|\; x_0 = s \right] \\ &= \mathbb{E}_\pi \left[\mathcal{R}(s,\pi(s),x_1) + \gamma \sum_{i=0}^{k-1} \gamma^i \mathcal{R}(x_{i+1}, \pi(x_{i+1}), x_{i+2}) \;\middle|\; x_0 = s \right] \\ &= \mathbb{E}_\pi \big[\, \mathcal{R}(s,\pi(s),x_1) + \gamma V_k^\pi(x_1) \;\big|\; x_0 = s \,\big] \end{aligned}$$

Analog zu (2.6) gilt nun

$$V^\pi(s) = \mathbb{E}_\pi \big[\, \mathcal{R}(s,\pi(s),x_1) + \gamma V^\pi(x_1) \;\big|\; x_0 = s \,\big] \tag{2.7}$$

Sei $\mathrm{F}(S)$ der Raum der reellen Funktionen auf S und $\pi \in \Pi_M^{\mathrm{sd}}$. Wir definieren die folgenden Operatoren, sofern die Ausdrücke wohldefiniert sind existieren:

$$\begin{aligned} \mathcal{T} \colon \mathrm{F}(S) &\to \mathrm{F}(S) \\ V &\mapsto \Big(s \mapsto \max_{u \in A_s} \mathbb{E}\big[\, \mathcal{R}(s,u,x_1) + \gamma V(x_1) \;\big|\; x_0 = s, u_0 = u \,\big] \Big) \end{aligned}$$

$$\mathcal{T}_\pi \colon \mathrm{F}(S) \to \mathrm{F}(S)$$
$$V \mapsto \Big(s \mapsto \mathbb{E}_\pi \big[\mathcal{R}(s, \pi(s), x_1) + \gamma V(x_1) \mid x_0 = s \big] \Big)$$

Ferner setzen wir für $k \in \mathbb{N}_0$

$$\mathcal{T}^k V \coloneqq \mathcal{T}^{k-1} V \qquad \mathcal{T}_\pi^k V \coloneqq \mathcal{T}_\pi^{k-1} V$$
$$\mathcal{T}^1 V \coloneqq \mathcal{T} V \qquad \mathcal{T}_\pi^1 V \coloneqq \mathcal{T} V$$

und

$$\mathcal{T}^\infty V \coloneqq \lim_{k \to \infty} \mathcal{T}^k V \qquad \mathcal{T}_\pi^\infty V \coloneqq \lim_{k \to \infty} \mathcal{T}_\pi^k V$$

Die Bellman-Gleichung lässt sich also als die Fixpunktgleichung

$$V^* = \mathcal{T} V^* \tag{2.8}$$

schreiben und man kann die folgenden Monotonieeigenschaften einfach zeigen:

Lemma 2.1. *Sei $V, \bar{V} \in \mathrm{F}(S)$ mit $V(s) \leq \bar{V}(s)$„ $s \in S$. Dann gilt für jede stationäre Strategie π*

$$(\mathcal{T}^k V)(s) \leq (\mathcal{T}^k \bar{V})(s), \quad s \in S,\ k \in \mathbb{N}_0$$
$$(\mathcal{T}_\pi^k V)(s) \leq (\mathcal{T}_\pi^k \bar{V})(s), \quad s \in S,\ k \in \mathbb{N}_0$$

Das folgende, wichtige Resultat ist für viele Fälle korrekt. Die Beweise unterscheiden sich jedoch stark, siehe die Bemerkungen auf Seite 14.

Theorem 2.2 ([BT96, S. 22])**.** *Unter gewissen Voraussetzungen gilt mit einer bestimmten Menge $\mathrm{B}(S) \subseteq \mathrm{F}(S)$:*

(a) Die eindeutige Lösung von (2.8) ist V^*.

(b) Es ist

$$\mathcal{T}^\infty J = V^* \tag{2.9}$$

für alle $J \in \mathrm{B}(S)$.

(c) Eine stationäre Strategie π ist genau dann optimal, falls

$$\mathcal{T}_\pi V^* = \mathcal{T} V^*$$

gilt.

(d) Für gewisse stationäre Strategien π ist V^π die eindeutige Lösung von $V^\pi = \mathcal{T}_\pi V^\pi$ und es gilt

$$\mathcal{T}_\pi^\infty J = V^\pi \tag{2.10}$$

für alle $J \in \mathrm{B}(S)$.

Für endliche S lässt sich der aus (2.6) abgeleitete Dynamische-Programmierung-Algorithmus schreiben als Lösung des Gleichungssystems

$$V(s) = \max_{u \in A_s} \sum_{s' \in S} p_{ss'}(u) \Big(\mathcal{R}(s,u,s') + \gamma V(s') \Big), \quad s \in S \tag{2.11}$$

bzw. lautet das Gleichungssystem wegen (2.2) im Fall endlicher Schritte

$$V_k(s) = \max_{u_k \in A_s} \sum_{s' \in S} p_{ss'}(u_k) \Big(\mathcal{R}(s,u_k,s') + \gamma V_{k-1}(s') \Big) \tag{2.12}$$

mit $s \in S$, $k = 1, \ldots, N$ und $V_0^*(s)$ eine Funktion, die die Belohnung für die Terminierung im Zustand s beschreibt.

2.2.1. Wert- und Strategie-Iteration

Da die Gleichung (2.9) sehr wichtig ist, wollen wir sie hier beispielhaft unter konkreten Voraussetzungen zeigen. Sei dafür $\gamma < 1$, M ein deterministischer Markov-Entscheidungsprozess und $\mathcal{R} < C$ beschränkt. D.h. wir können $\pi \in \Pi_M^{\mathrm{sd}}$ annehmen. Es folgt, dass V^* beschränkt ist:

$$\sup_{\pi \in \Pi_M^{\mathrm{sd}}} |V^\pi(s)| \leq \sup_{\pi \in \Pi_M^{\mathrm{sd}}} \sum_{k=0}^{\infty} \gamma^k |\mathcal{R}(x_k, \pi(x_k), x_{k+1})| < \frac{C}{1-\gamma}$$

Wir wählen $\mathrm{B}(S) = \{ f \colon S \to \mathbb{R} \mid f \text{ beschränkt} \}$. Mit der Supremumsnorm ist $(\mathrm{B}(S), \|\cdot\|_\infty)$ ein Banachraum und es gilt für $f, g \in \mathrm{B}(S)$:

$$\begin{aligned} \|\mathcal{T}_\pi f - \mathcal{T}_\pi g\|_\infty &= \sup_{s \in S} \Big| \mathbb{E}_\pi \big[\gamma(f(x_1) - g(x_1)) \mid x_0 = s \big] \Big| \\ &= \gamma \sup_{s \in S} |f(x_1(s)) - g(x_1(s))| \leq \gamma \|f - g\|_\infty \end{aligned}$$

Das zeigt, dass $\mathcal{T}_\pi$ eine Kontraktion ist. Dabei bezeichnet $x_1(s) \in S$ den Zustand, den wir erhalten, falls $u_0 = \pi(s)$ und $x_0 = s$ gewählt wird. Durch die Betrachtung des Supremums auf der linken Seite über π und eine Vertauschung folgt, dass auch $\mathcal{T}$ eine Kontraktion ist. Außerdem gilt für $s \in S$:

$$\begin{aligned} |(\mathcal{T}f)(s)| &\leq \max_{u \in A_s} \left(|\mathcal{R}(s, u, x_1(s))| + \gamma |f(x_1(s))| \right) \\ &\leq C + \|f\|_\infty \end{aligned}$$

Also ist $\mathcal{T}f \in \mathrm{B}(S)$ und der Banachsche Fixpunktsatz liefert nun einen eindeutigen Fixpunkt $J^* \in \mathrm{B}(S)$ mit $J^* = \mathcal{T}J^*$ und

$$J^* = \lim_{k \to \infty} \mathcal{T}^k J = \mathcal{T}^\infty J$$

für alle $J \in \mathrm{B}(S)$. Mit der Eindeutigkeit folgt $J^* = V^*$, also (2.9).

Die durch (2.9) definierte Fixpunkt-Iteration

$$J_0 \coloneqq J \qquad\qquad J_{k+1} \coloneqq \mathcal{T}J_k$$

wird als *Wert-Iteration* bezeichnet und stellt eine Möglichkeit dar V^* zu berechnen. Analog lässt sich mit (2.10) eine Fixpunkt-Iteration für V^π angeben.

Die Wert-Iteration kann in einigen Fällen nach endlich vielen Schritten abbrechen, also $J_N = V^*$. Im Allgemeinen sind allerdings endlich viele Iteration nicht genug, um die Wert-Funktion exakt zu berechnen. Die folgende *Strategie-Iteration* hat bessere Konvergenzeigenschaften.

Startend mit einer Strategie $\pi_k \in \Pi_M^{\mathrm{sd}}$ erhalten wir V^{π_k}, indem wir die Lösung von (2.7), d.h. von

$$\mathcal{T}_{\pi_k} J = J$$

berechnen.

Im zweiten Schritt wählen wir eine Strategie π_{k+1}, so dass

$$\mathcal{T}_{\pi_{k+1}} V^{\pi_k} = \mathcal{T} V^{\pi_k} \tag{2.13}$$

gilt. D.h. wir wählen π_{k+1} als eine Strategie, die das Maximum für einen Schritt der Wert-Iteration mit Startwert V^{π_k} annimmt:

$$\pi_{k+1}(s) = \arg\max_{u \in A_s} \mathbb{E}\left[\mathcal{R}(s, u, x_1) + V^{\pi_k}(x_1) \;\middle|\; x_0 = s, u_0 = u \right]$$

Das folgende Lemma zeigt nun beispielhaft ein paar Konvergenzeigenschaften.

Lemma 2.3. *Sei V^{π_k} eine Folge in $\mathrm{B}(S)$ und $V^*(s) < \infty$ für alle $s \in S$. Dann konvergiert die Folge V^{π_k} punktweise. Ist $B(S)$ endlich, so konvergiert die Folge gegen eine optimale Strategie. Insbesondere wird diese dann nach endlich vielen Schritten angenommen (trivial).*

Beweis. Es gilt für jedes $s \in S$ im Folgenden punktweise:

$$V^{\pi_k} \overset{(2.7)}{=} \mathcal{T}_{\pi_k} V^{\pi_k} \leq \mathcal{T} V^{\pi_k} \overset{(2.13)}{=} \mathcal{T}_{\pi_{k+1}} V^{\pi_k} \tag{2.14}$$

Wenden wir Lemma 2.1 auf (2.14) an, so folgt

$$\mathcal{T}_{\pi_{k+1}} V^{\pi_k} \leq \mathcal{T}_{\pi_{k+1}} (\mathcal{T}_{\pi_{k+1}} V^{\pi_k}) = \mathcal{T}^2_{\pi_{k+1}} V^{\pi_k}$$

Das wieder in (2.14) eingesetzt ergibt induktiv für alle $m \in \mathbb{N}$

$$V^{\pi_k} \leq \mathcal{T}^m_{\pi_{k+1}} V^{\pi_k}$$

und im Grenzwert $m \to \infty$ folgt wegen (2.10)

$$V^{\pi_k} \leq V^{\pi_{k+1}}$$

Die erzeugte Folge ist also punktweise monoton und durch V^* beschränkt, also konvergent.

Falls Gleichheit gilt, ist

$$V^{\pi_{k+1}} \overset{(2.7)}{=} \mathcal{T}_{\pi_{k+1}} V^{\pi_{k+1}} = \mathcal{T}_{\pi_{k+1}} V^{\pi_k} \overset{(2.13)}{=} \mathcal{T} V^{\pi_k} = \mathcal{T} V^{\pi_{k+1}}$$

Also erfüllt $V^{\pi_{k+1}}$ die Bellman-Gleichung und ist deswegen optimal, bzw. wegen Eindeutigkeit bereits V^*. Das zeigt auch den zweiten Teil. □

Für endliche S ist die Forderung endlicher $\mathrm{B}(S)$ häufig erfüllt. Sonst erzeugt das Verfahren in vielen Fällen zumindest eine Folge, die die Wert-Funktion beliebig gut approximiert.

Ein Problem der exakten Strategie-Iteration (Algorithmus 2.1) ist in Zeile 5 die möglicherweise teure Berechnung von $V^{\pi_{k+1}}$, da mehrmals über den gesamten Zustandsraum iteriert werden muss. Eine Möglichkeit dies zu verbessern ist es, die Berechnung zu ersetzen durch eine endliche Fixpunkt-Iteration

$$J_{k+1} \leftarrow \mathcal{T}^m_{\pi_{k+1}} J_{\pi_k}, \quad m \in \mathbb{N}$$

Algorithmus 2.1 Exakte Strategie-Iteration

Input: π, V^π

1: $k \leftarrow 0$
2: $\pi_k \leftarrow \pi$, $J_k \leftarrow V^\pi$
3: **repeat**
4: $\quad \pi_{k+1} \leftarrow \pi$ mit $\mathcal{T}_\pi J_k = \mathcal{T} J_k$ ▷ »*policy improvement*«
5: $\quad J_{k+1} \leftarrow J$ mit $\mathcal{T}_{\pi_{k+1}} J = J$ ▷ »*policy evaluation*«
6: $\quad k \leftarrow k + 1$
7: **until** $J_k = J_{k-1}$
8: **return** π_k

welche gegen $V^{\pi_{k+1}}$ konvergiert. Wählt man $m = 1$, so erhält man wegen $\mathcal{T}_{\pi_{k+1}} J_k = \mathcal{T} J_k$ (Zeile 4) wieder die Wert-Iteration.

Die allgemeine Idee, die beiden Abschnitte *policy improvement* und *policy evaluation* zu kombinieren, bildet die Grundlage bei vielen Verfahren im Verstärkenden Lernen. Wir werden in den weiteren Abschnitten daher auf diese Iterationen zurückkommen. Für Details und eine Reihe von Varianten siehe [BT96, S. 25ff]. In [SB98, S. 89ff] befinden sich zusätzlich numerische Experimente.

2.3. Monte-Carlo-Verfahren

Die Dynamische Programmierung (2.11) und (2.12) ist für große (endliche) S teuer und benötigt Kenntnis der Zustandsdynamiken, d.h. der Funktionen $p_{ss'}(u)$. Sind diese nicht bekannt, aber können für eine gegebene Strategie einzelne Ereignisse der zugehörigen Markov-Kette, also Folgen $\{(x_k, u_k)\}_{k=0}^{\infty} \subseteq S \times A$, durch Simulation oder Messung erhalten werden, so kommen so genannte *Monte-Carlo-Verfahren* in Frage, die wir hier kurz präsentieren um die Besonderheiten der TD-Verfahren später durch einen Vergleich besser hervorheben und verstehen zu können. Außerdem werden nötige Begrifflichkeiten und Ideen bereits eingeführt. Die Vorgehensweise wurde aus [SB98, S. 111ff] und [BT96, S. 180ff] übernommen, wo sich Beispiele, Experimente und genaue Informationen bzgl. Voraussetzungen finden lassen. Wir beschränken uns auf den Fall endlicher S.

Da in der Praxis keine unendlichen $\{(x_k, u_k)\}_{k=0}^{\infty}$ realisierbar sind, fordern wir von einem Markov-Entscheidungsprozess weitere Eigenschaften, die sicherstellen, dass wir nach endlich vielen Schritten aufhören können:

Definition 2.3.1. Ein Markov-Entscheidungsprozess mit $m \in \mathbb{N}$ Zuständen heißt *episodisch*, falls es einen *terminalen Zustand* $t \in S$ gibt, so dass gilt:

(i) Für alle $s \in S$ und $u \in A_t$ ist $r_{ts}(u) = 0$ und

$$p_{ts}(u) = \begin{cases} 1 & \text{wenn } s = t \\ 0 & \text{sonst} \end{cases}$$

(ii) Es gibt ein $\pi \in \Pi_M^{\mathrm{s}}$ mit

$$\mathrm{P}[\, x_m = t \mid x_0 = s \,] > 0$$

Solche π nennen wir *zulässig*. Für nicht zulässige $\pi \in \Pi_M^{\mathrm{s}}$ gilt $V^{\pi}(s) = -\infty$ für mindestens ein $s \in S$.

Bemerkung. Bei der zugehörigen Markov-Kette spricht man bei t von einem *absorbierenden Zustand*. Die Eigenschaft (ii) ist notwendig für die *Irreduzibilität* der Markov-Kette.

Entsprechend kann man nun Episoden[5] betrachten.

Definition 2.3.2. Sei M ein episodischer Markov-Entscheidungsprozess und $\pi \in \Pi_M^s$ zulässig. Eine Teilmenge $\{(x_k, u_k)\}_{k=0}^d \subseteq S \times A$ eines Ereignis der Markov-Kette heißt *Episode*, falls x_d ein terminaler Zustand ist und alle x_i mit $i < d$ nicht terminal sind.

Ist nun eine Folge von Episoden $\{(x_{ij}, u_{ij})\}_{j=0}^{d_i}$ gegeben, so können wir mit Hilfe des Gesetzes der großen Zahlen eine Approximation an V^π durch Mittelwertbildung der einzelnen Belohnungssummen berechnen. Dafür ist nun nur noch $\mathcal{R}(s, u, s')$ nötig.

Kennt man $V(s) \approx V^\pi(s)$, aber nicht die Übergangswahrscheinlichkeiten, so ist es nicht mehr direkt möglich aus V wieder eine Strategie $\tilde{\pi}$ für ein *policy improvement* zu gewinnen. Es müsste eine maximierende Aktion

$$\arg\max_{u \in A_s} \mathbb{E}\big[\, \mathcal{R}(s, u, x_1) + V(x_1) \;\big|\; x_0 = s, u_0 = u \,\big]$$

gewählt werden. Dieser Erwartungswert ist nun wegen fehlender Kenntnis der Systemdynamik nicht mehr direkt berechenbar.

Eine wichtige Idee ist es, stattdessen die Wert-Funktion für Zustand-Aktion-Paare $(s, a) \in S \times A$ zu berechnen, d.h. die erwartete Belohnung für den Fall, dass im Zustand s die Aktion a gewählt wurde. Diese spezielle Wert-Funktion lautet

$$\begin{aligned} Q^\pi(s, a) &:= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}(x_k, u_k, x_{k+1}) \;\middle|\; x_0 = s, u_0 = a \right] \\ &= \mathbb{E}_\pi\big[\, \mathcal{R}(s, a, x_1) + \gamma V^\pi(x_1) \;\big|\; x_0 = s, u_0 = a \,\big] \end{aligned} \tag{2.15}$$

Dann kann durch Wahl von

$$\tilde{\pi}(s) := \arg\max_{u \in A_s} Q^\pi(s, u)$$

eine neue Strategie berechnet werden (*policy improvement*). Durch Erzeugung von Episoden für $\tilde{\pi}$ können wir nun wieder iterieren und erhalten unter gewissen Voraussetzungen Konvergenz gegen eine optimale Strategie.

[5] Man könnte auch von *samples* sprechen.

Algorithmus 2.2 Monte-Carlo Strategie-Evaluation

Input: $\{(x_{ij}, u_{ij})\}_{j=0}^{d_i},\ i = 0, \ldots, d$

1: $Q(s,a) \leftarrow 0, \quad (s,a) \in \Gamma$
2: $m(s,a) \leftarrow 0, , \quad (s,a) \in \Gamma$ ▷ Anzahl Vorkommen von (s, a)
3: **for all** $i = 0, \ldots, d$ **do**
4: **for all** $j = 1, \ldots, d_i$ **do**
5: $m(x_{ij}, u_{ij}) \leftarrow m(x_{ij}, u_{ij}) + 1$
6: $\alpha \leftarrow \left|m(x_{ij}, u_{ij})\right|^{-1}$
7: $r \leftarrow \sum_{k=j}^{d_i - 1} \gamma^k \mathcal{R}(x_{ik}, u_{ik}, x_{i(k+1)})$
8: $Q(x_{ij}, u_{ij}) \leftarrow Q(x_{ij}, u_{ij}) + \alpha \left(r - Q(x_{ij}, u_{ij})\right)$
9: **end for**
10: **end for**
11: **return** Q

Man beachte, dass in Algorithmus 2.2 die Zeile 8 gerade den Mittelwert rekursiv berechnet. Wichtig ist hier auch anzumerken, dass es $(s,a) \in \Gamma$ geben könnte, die in einer Episode mehrmals vorkommen. Dann ist das Update in Zeile 8 nicht mehr mit unabhängigen Teilepisoden, d.h. das Gesetz der großen Zahlen muss nicht erfüllt sein. Man kann zeigen, dass Konvergenz dennoch vorliegt. Alternativ führt man das Update nur beim ersten Besuch pro Episode aus, was für die Praxis — also eine endliche Anzahl an Episoden — vorteilhafter ist, da der Fehler schneller abfällt im Vergleich zum mehrmaligen Update.

Eine Voraussetzung für Konvergenz ist, dass jedes Paar $(s,a) \in \Gamma$ in den Episoden unendlich oft auftaucht, um eine Approximation berechnen zu können. Eine Möglichkeit dies sicher zu stellen ist, Episoden zu generieren, die in s mit a beginnen. Dieses Vorgehen heißt *exploring starts*, was so genannt wird, da durch den Start in (s,a) neue, möglicherweise profitablere Bereiche der Wert-Funktion »entdeckt« werden. Siehe [BT96, S. 238ff] für Beispiele, wo diese *exploration* illustriert wird und tatsächlich nötig ist.

Die freie Wahl des Anfangszustandes für Episoden ist in der Praxis eine starke Forderung. Sie kann jedoch durch die Benutzung von ε-Strategien ersetzt werden:

Definition 2.3.3. Sei $\pi \in \Pi_M^{\mathrm{s}}$ zulässig und $0 < \varepsilon < 1$. Dann nennen

wir π eine ε-*Strategie*, falls

$$\pi(s,a) \geq \frac{\varepsilon}{|A_s|}$$

für alle $(s,a) \in \Gamma$ gilt.

Haben wir nun $Q^{\pi_k}(s,a)$ mit einer ε-Strategie π_k und ist a_s^* eine $Q^{\pi_k}(s,\cdot)$ maximierende Aktion, so können wir mit

$$\pi_{k+1}^{\varepsilon}(s,a) \coloneqq \begin{cases} \frac{\varepsilon}{|A_s|}, & \text{falls } a \neq a_s^* \\ 1 - \frac{\varepsilon}{|A_s|}\left(|A_s| - 1\right), & \text{falls } a = a_s^* \end{cases}$$

eine ε-Strategie definieren, die *greedy* bzgl. a_s^* ist, d.h. mit einer Wahrscheinlichkeit von ε wird nicht die optimale Aktion, sondern eine andere zufällige gewählt. Diese Strategie können wir nun nutzen, um Episoden zu generieren, die mit positiver Wahrscheinlichkeit alle erreichbaren Zustände enthält. So erhalten wir eine Strategie, die unter den ε-Strategien optimal ist.

Bildlich gesprochen halten ε-Strategien alle Wege durch den Zustandsraum offen und es ist gewährleistet, dass in den erzeugten Episoden immer auch die Zustände auftauchen, die nach aktueller Iterierten der Strategie nicht als optimal betrachtet werden. Auch hier spielt die entstehende *exploration* eine entscheidende Rolle, um die gesamte Wert-Funktion nach möglichen profitableren Alternativwegen abzusuchen. Allerdings muss auch eine gewisse Balance zwischen *exploration* und *exploitation*, der Nutzung der aktuellen — möglicherweise schlechten — Wert-Abschätzung bestehen, da sonst keine optimale Strategie gefunden (lang- und kurzfristige Belohnung werden falsch eingeschätzt).

Der Parameter ε muss per Hand gewählt werden. Um Konvergenz gegen eine optimale Strategie zu erhalten gibt es verschiedenen Möglichkeiten. Beispielsweise kann man mit den Iterationen den Parameter ε abfallen lassen. Eine Übersicht über verschiedene Varianten, Bezüge zur Psychologie und Neurobiologie und ein adaptives Verfahren für ε finden sich in [Tok13].

Die hier vorgestellten Ideen, durch Generierung von Episoden in Verbindung mit bspw. ε-Strategien die Wert-Funktion $Q^{\pi}(s,a)$ zu berechnen, um so wieder die üblichen *policy evaluation* bzw. *policy improvement* Schritte durchführen zu können, sind in den folgenden Verfahren und auch allgemein im Verstärkenden Lernen sehr wichtig. Es ist in der Praxis häufig einfacher,

Episoden zu generieren, als Zustandsdynamiken zu berechnen — man denke an Systeme die durch gewöhnliche Differentialgleichungen beschrieben werden. Aber auch für große S kommen exakte Verfahren wie der Dynamische-Programmierung-Algorithmus nicht mehr in Frage. Der bekannte *Curse Of Dimensionality* geht ebenfalls auf Bellman zurück [Bel57, S. ix].

2.4. TD-Verfahren

Die Idee der *temporal-difference*-Verfahren (TD) hat zu vielen Algorithmen geführt, die heute als Verstärkendes Lernen bezeichnet werden. Richard Sutton und Andrew Barto haben Anfang der 80er in der Lernpsychologie eine neue Modellierung vorgestellt [SB81], die folgende experimentelle Beobachtung erklären kann: Lernfähige Lebewesen sind in der Lage, zukünftig zu erwartende Belohnungen zu lernen (persönliche Wert-Funktion) und sie passen den Wert der erwarteten Belohnung während des Lernvorgangs nach jedem Zeitschritt an, *basierend auf ihrer aktuellen Erwartung des folgenden Zustandes*. D.h. statt sich die erhaltenen Belohnungen und die damit verbunden Zustände über eine lange Zeit zu merken und dann rückwirkend mit den tatsächlichen Belohnungen die Erwartungen anzupassen, geschieht das sofort. Erhält das Lebewesen im Zustand s eine Belohnung r und ist nun im Zustand s', so wird $r + \gamma V(s')$ als Abschätzung für die erwartete Belohnung benutzt. Betrachtet wird nun der so genannte *TD-Fehler*

$$\Big[r + \gamma V(s')\Big] - V(s)$$

um mit diesem die Erwartung anzupassen:

$$V(s) \leftarrow V(s) + \alpha \Big([r + \gamma V(s')] - V(s)\Big)$$

In den Messungen sollte der TD-Fehler um den Erwartungswert fluktuieren. Indem während des Lernprozesses nun die *Lernrate* α gesenkt wird, entsteht eine Art Mittelwertbildung. Schauen wir in die Zeile 8 von Algorithmus 2.2 zurück, werden die Parallelen deutlich.

Wir geben in diesem Abschnitt zunächst eine weitere Herleitung und theoretische Grundlage des Verfahrens mit Hilfe von *stochastischer Approximation* an. Danach verallgemeinern wir das Verfahren und erhalten TD(λ), was als Spezialfall auch den Monte-Carlo-Algorithmus enthält.

2.4.1. Stochastische Approximation

In diesem Abschnitt wollen wir ein Approximationsverfahren angeben, dass sowohl die Berechnung von V^π im Monte-Carlo-Algorithmus darstellt, als auch das TD-Verfahren und dessen Verallgemeinerungen. Tatsächlich werden wir sehen, dass die Lösung des zunächst allgemeinen Problems die Herleitung vereinfacht. Wir wollen das Problem betrachten Fixpunkt-Gleichungen der Form

$$J(s) = \mathbb{E}\big[\, F(J(s)) \,\big]$$

zu lösen, wobei die Erwartungswerte nur durch Beispielereignisse berechnet werden können. Eine ausführliche Abhandlung der stochastischen Approximation auch im Zusammenhang mit TD(λ) befindet sich in [KY03]. In [BS96, S 132ff und S. 154ff] findet sich die stochastische Approximation mit einem stärkeren Fokus auf die beschriebenen Verfahren. Aus den in diesem Kapitel bekannten Gründen für nicht näher spezifizierte Voraussetzungen beschränken wir uns auch hier auf eine Motivation der Verfahren.

Sei $H\colon \mathbb{R}^n \to \mathbb{R}^n$. Um eine Lösung der Gleichung

$$\boldsymbol{x} = H(x), \quad \boldsymbol{x} \in \mathbb{R}^n \tag{2.16}$$

zu berechnen, können wir alternativ eine Nullstelle von

$$h(\boldsymbol{x}) \coloneqq H(\boldsymbol{x}) - \boldsymbol{x}$$

berechnen.

Falls H differenzierbar ist, ist eine Möglichkeit das Newton-Verfahren oder ein Quasi-Newton-Verfahren:

$$x_{k+1} = x_k + \alpha_k B_k h(x_k)$$

Eine Wahl die wir im Folgenden betrachten ist $B_k = \mathbb{1}$. Beispielsweise unter geeigneten Kontraktionsbedingungen konvergiert dieses Verfahren lokal.

Wir haben also die Iteration

$$x_{k+1} = x_k + \alpha_k(H(x_k) - x_k) \tag{2.17}$$

Welche auch als Fixpunktiteration von (2.16) mit Schrittweitensteuerung aufgefasst werden kann.

Wie bereits erwähnt, interessiert uns der Fall, dass $H(x)$ nicht direkt berechnet werden kann, sondern lediglich gestörte Werte

$$H(\boldsymbol{x}) + Y_t$$

wobei $\{Y_t\}_{t\in\mathbb{N}}$ eine Folge von Zufallsvariablen ist, die zusätzliche Eigenschaften erfüllt, so dass wir bspw. mit Hilfe des Gesetzes der großen Zahlen Konvergenz erhalten:

$$H(\boldsymbol{x}) = \lim_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} \xi(\boldsymbol{x})_t$$

Wobei $\{\xi(\boldsymbol{x})_t\}$ entsprechende Ereignisse sind.

Statt $H(x)$ für jeden Iterationsschritt von (2.17) möglichst genau zu berechnen, haben Herbert Robbins und Sutton Monro 1951 gezeigt [RM51], dass es für die Nullstellensuche genügt $\xi(x) \coloneqq \xi(\boldsymbol{x})_1$ als Approximation zu verwenden, falls die Folge $\{a_k\}$ in geeigneter Weise abfällt. Eine Bedingung, die in vielen Aussagen eine Rolle spielt ist

$$\sum_{k=0}^{\infty} a_k = \infty \quad \text{und} \quad \sum_{k=0}^{\infty} a_k^2 < \infty \tag{2.18}$$

Die allgemeine Iteration ist nun also

$$x_{k+1} = x_k + \alpha_k(\xi(x_k) - x_k) \tag{2.19}$$

Anpassung für Strategie-Iteration

Sei $S = \{s_i\}_{i=1}^{d}$ endlich und $\pi \in \Pi_M^{\mathrm{s}}$. Wir wollen nun V^π berechnen und identifizieren V^π dafür mit einem Vektor

$$\boldsymbol{v}_\pi = \Big(V^\pi(s_1), V^\pi(s_2), \ldots, V^\pi(s_d)\Big) \in \mathbb{R}^d$$

Nun erfüllt $\boldsymbol{v}_\pi$ eine Fixpunktgleichung

$$\boldsymbol{v}_\pi = \mathbb{E}_\pi\big[\, F(\boldsymbol{v}_\pi)\,\big]$$

mit einer geeigneten Zufallsvariable F auf $\mathbb{R}^d$, bspw. die Bellman-Gleichung (2.7) oder die ursprüngliche Definition (2.3). Man beachte auch,

dass pro Komponente die Erwartungswerte bzgl. verschiedener Maße P_s berechnet werden.

In der Simulation erhalten wir pro Zeitschritt in der Regel nicht für jeden Zustand einen Belohnungswert, sondern das erhaltene Ereignis bezieht sich nur auf einen Teil der Zustände, also auf Komponenten von $\boldsymbol{v}_\pi$. Am Beispiel der Monte-Carlo-Simulation sieht man, dass nur für die besuchten Zustände einer Episode Ereignisse vorhanden sind (die Summe der danach erhaltenen Belohnungen). Im TD-Verfahren bezieht sich ein Ereignis tatsächlich nur auf einen Zustand. Wir müssen also die Faktoren α_k komponentenweise betrachten, bzw. können α_k als Diagonalmatrix auffassen. Wir sprechen im Folgenden von einem Vektor

$$\alpha_k = \Big(\alpha_k(1), \alpha_k(2), \ldots, \alpha_k(d)\Big) \in \mathbb{R}^d$$

und meinen im Verfahren (2.19) die entsprechende Diagonalmatrix. Man nennt diese Variante die *asynchrone* Stochastische Approximation, siehe auch [KY03, Abschnitt 2.4.1].

Wir werden im Folgenden mangels Platz und nötiger Begrifflichkeiten diese Variante ohne Anspruch auf vollständige Sauberkeit genauer ausführen. Details finden sich in den angegebenen Büchern.

Sei $\{\xi_k\}_{k\in\mathbb{N}}$ eine Folge von Ereignissen und $I_i = \left\{\kappa(i)_j\right\}_{j=1}^{\infty}$ die aufsteigende Folge der Indizes, sodass $\xi_{\kappa(i)_j}$ ein Ereignis für s_i ist. Dann setzen wir

$$\alpha_k(i) \coloneqq 0, \quad k \notin I_i$$

wodurch für diese k dann $x_{k+1}(i) = x_k(i)$ wird.

Für die restlichen Faktoren muss die Folge entsprechend (2.18) gewählt werden, bspw.

$$\alpha_{\kappa(i)_j}(i) \coloneqq \frac{1}{j}$$

Bemerkung. Eine Sache, die hier ausgearbeitet werden müsste, ist die Tatsache, dass α_k so zu einer Zufallsvariable wird, da die besuchten Zustände und damit I_i von Zufallsvariablen abhängen. Ist man nur an einer Implementierung interessiert, spielt dies keine Rolle.

Wie man sieht, ist das Monte-Carlo-Verfahren ein Spezialfall und auch die folgenden Verfahren lassen sich über diese Approximation darstellen. Konkret werden wir immer die *policy evaluation* implementieren.

2.4.2. TD(λ)

Die Verallgemeinerung durch das TD(λ)-Verfahrens besteht darin, an Stelle die Belohnung des nächsten Schrittes als Approximationswert zu nutzen, stattdessen eine exponentielle Gewichtung (mit λ) der Belohnungen nach jeweils n Schritten zu betrachten. Das entstehende Verfahren wollen wir aus einer Fixpunktgleichung herleiten. Die bereits oben betrachteten TD-Fehler werden wir brauchen:

$$d_i \coloneqq \mathcal{R}(x_i, u_i, x_{i+1}) + \gamma V^\pi(x_{i+1}) - V^\pi(x_i)$$

Sei $\lambda \in (0, 1)$. Betrachten wir die auf l Schritte verallgemeinerte Bellman-Gleichung:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{i=0}^{l-1} \gamma^i \mathcal{R}(x_i, u_i, x_{i+1}) + \gamma^l V^\pi(x_l) \;\middle|\; x_0 = s \right]$$

Indem wir nun beide Seiten mit λ^{l-1} multiplizieren und über l summieren erhält man wegen $\sum_{l=1}^{\infty} \lambda^{l-1} = (1-\lambda)^{-1}$ und mit der zusätzlichen Forderung, dass $V^\pi(s)$ beschränkt ist (Majorisierte Konvergenz):

$$V^\pi(s) = (1-\lambda)\, \mathbb{E}_\pi \left[\sum_{l=1}^{\infty} \lambda^{l-1} \left(\sum_{i=0}^{l-1} \gamma^i \mathcal{R}(x_i, u_i, x_{i+1}) + \gamma^l V^\pi(x_l) \right) \;\middle|\; x_0 = s \right]$$

Wenn wir annehmen, dass die Ereignisse ab einer Schrittzahl N (Zufallsvariable) einen terminalen Zustand erreichen, sind die Summen endlich und wir können die Reihenfolge vertauschen:

$$\begin{aligned}
&\sum_{l=1}^{\infty} \lambda^{l-1}(1-\lambda) \left(\sum_{i=0}^{\infty} \gamma^i \mathcal{R}(x_i, u_i, x_{i+1}) \mathbb{1}_{\{i<l\}} + \gamma^l V^\pi(x_l) \right) \\
&\quad = \sum_{i=0}^{\infty} \gamma^i \mathcal{R}(x_i, u_i, x_{i+1}) \underbrace{\sum_{l=1}^{\infty} (\lambda^{l-1} - \lambda^l) \mathbb{1}_{\{i<l\}}}_{=\lambda^i} + \sum_{l=1}^{\infty} (\lambda^{l-1} - \lambda^l) \gamma^l V^\pi(x_l) \\
&\quad = \sum_{i=0}^{\infty} (\lambda\gamma)^i \, \mathcal{R}(x_i, u_i, x_{i+1}) + \sum_{i=0}^{\infty} \left(\lambda^i - \lambda^{i+1} \right) \gamma^{i+1} V^\pi(x_{i+1})
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=0}^{\infty} (\lambda\gamma)^i \left(\mathcal{R}(x_i, u_i, x_{i+1}) + \gamma V^\pi(x_{i+1}) \right) - \sum_{i=0}^{\infty} (\lambda\gamma)^{i+1} V^\pi(x_{i+1}) \\
&= \sum_{i=0}^{\infty} (\lambda\gamma)^i \left(\mathcal{R}(x_i, u_i, x_{i+1}) + \gamma V^\pi(x_{i+1}) - V^\pi(x_i) \right) \\
&\quad + \sum_{i=0}^{\infty} \left[(\lambda\gamma)^i V^\pi(x_i) - (\lambda\gamma)^{i+1} V^\pi(x_{i+1}) \right] && \text{»Teleskopsumme«} \\
&= \sum_{i=0}^{\infty} (\lambda\gamma)^i d_i + V^\pi(x_0)
\end{aligned}
$$

Wir erhalten also die Fixpunktgleichung

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} (\lambda\gamma)^i d_i \,\middle|\, x_0 = s \right] + V^\pi(s) \tag{2.20}$$

für V^π.

Sei $S = \{s_i\}_{i=1}^d$. Wir bezeichnen mit v_i die i-te Komponente von einem $\boldsymbol{v} \in \mathbb{R}^d$ und mit $\mathbb{1}_i$ den kanonischen i-ten Einheitsvektor. Wir identifizieren die Zufallsvariablen x_k der Zustände mit Zufallsvariablen ι_k von Indizes, so dass $x_k = s_{\iota_k}$ gilt. Sei $H \colon \mathbb{R}^d \to \mathbb{R}^d$,

$$H(\boldsymbol{v})_i \coloneqq \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} (\lambda\gamma)^i \left(\mathcal{R}(s_{\iota_k}, u_k, s_{\iota_{k+1}}) + \gamma v_{\iota_{k+1}} - v_{\iota_k} \right) \,\middle|\, \iota_0 = i \right] + v_i$$

Gesucht ist nun eine Lösung von

$$\boldsymbol{v} = H(\boldsymbol{v})$$

mittels stochastischer Approximation.

Dafür sei $\{(s_{i_k}, u_k)\}_{k=0}^{\infty}$ ein Ereignis der Markov-Kette und

$$\delta_k(\boldsymbol{v}) \coloneqq \mathcal{R}(s_{i_k}, u_k, s_{i_{k+1}}) + \gamma v_{i_{k+1}} - v_{i_k} \in \mathbb{R}$$

ein entsprechendes Ereignis von d_i. Um nun $H(\boldsymbol{v})$ in einer Iteration der Stochastischen Approximation durch ein Sample zu nähern, betrachten wir

$$\xi_k(\boldsymbol{v}) \coloneqq \left(\sum_{i=0}^{\infty} (\lambda\gamma)^i \delta_{k+i}(\boldsymbol{v}) + v_{i_k} \right) \mathbb{1}_{i_k}$$

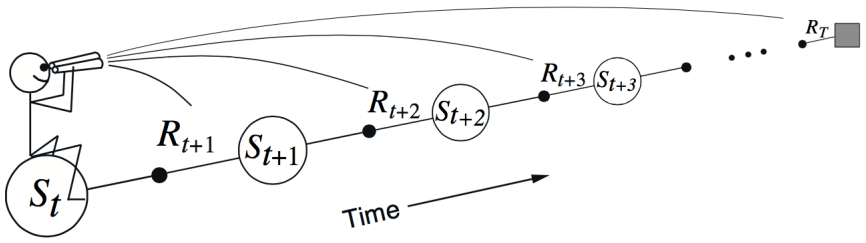

Abbildung 2.1.: Die Vorwärts-Interpretation von TD(λ): Der Lernende »sieht« die folgenden Belohnungen. Entnommen aus [SB98, S. 171].

$$= \left(\sum_{i=k}^{\infty} (\lambda\gamma)^{i-k} \, \delta_i(\boldsymbol{v}) + v_{i_k} \right) \mathbb{1}_{i_k} \in \mathbb{R}^d$$

Also die Summe aus (2.20) wird im Schritt k für eine Komponente — also einen Eintrag der Wert-Funktion für einen Zustand s_{ι_k} — genähert, indem die Näherung aus dem Teil der Ereignisse erzeugt wird, der bei k, folglich mit Startzustand s_{ι_k}, beginnt: $\left\{(s_{i_j}, u_j)\right\}_{j=k}^{\infty}$.

Dann lautet mit geeigneter Wahl von α_k (s.o.) die Iteration

$$\begin{aligned} \boldsymbol{v}^{(k+1)} &:= \boldsymbol{v}^{(k)} + \alpha_k \left(\xi_k\Big(\boldsymbol{v}^{(k)}\Big) - \boldsymbol{v}^{(k)} \right) \\ &= \boldsymbol{v}^{(k)} + \alpha_k \sum_{i=k}^{\infty} (\lambda\gamma)^{i-k} \, \delta_i\Big(\boldsymbol{v}^{(k)}\Big) \end{aligned} \tag{2.21}$$

Man beachte, dass dies für $\lambda = 0$, mit $0^0 = 1$, dem einfachen TD-Verfahren oben und für $\lambda = 1$ dem Monte-Carlo-Verfahren entspricht — bei entsprechendem α_k.

Betrachten wir die ersten Samples in den jeweiligen Komponenten:

$$\begin{aligned} (\lambda\gamma)^0 \, \delta_0\Big(\boldsymbol{v}^{(0)}\Big) \;+\; (\lambda\gamma)^1 \, \delta_1\Big(\boldsymbol{v}^{(0)}\Big) \;+\; (\lambda\gamma)^2 \, \delta_2\Big(\boldsymbol{v}^{(0)}\Big) \;+\; \dots \\ (\lambda\gamma)^0 \, \delta_1\Big(\boldsymbol{v}^{(1)}\Big) \;+\; (\lambda\gamma)^1 \, \delta_2\Big(\boldsymbol{v}^{(1)}\Big) \;+\; \dots \\ +\; (\lambda\gamma)^0 \, \delta_2\Big(\boldsymbol{v}^{(2)}\Big) \;+\; \dots \end{aligned}$$

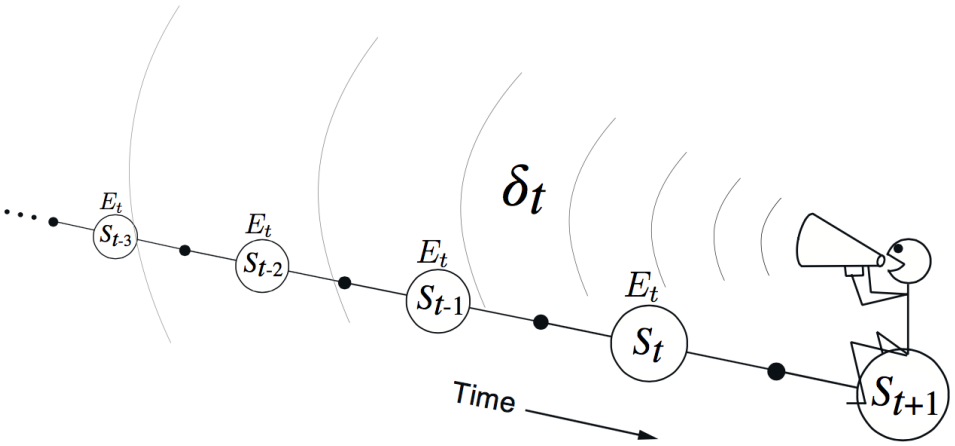

Abbildung 2.2.: Die Rückwärts-Interpretation von TD(λ). Entnommen aus [SB98, S. 175].

Der Vektor $\boldsymbol{v}^{(i)}$ unterscheidet sich von $\boldsymbol{v}^{(i+1)}$ höchstens in der Komponente ι_i. D.h. für $i < j$ unterscheiden sich die Vektoren $\boldsymbol{v}^{(i)}$ und $\boldsymbol{v}^{(j)}$ maximal in den Komponenten $\iota_i, \iota_{i+1}, \ldots, \iota_{j-1}$. Der TD-Fehler $\delta_j(\boldsymbol{v})$ hängt nur von den Komponenten ι_j und ι_{j+1} ab. Im Fall, dass außer der terminalen Zustände kein Zustand mehr als einmal in dem Ereignis auftaucht, ist also mit Startwert $\boldsymbol{v}^{(0)} = 0$:

$$\delta_j\left(\boldsymbol{v}^{(i)}\right) = \delta_j\left(\boldsymbol{v}^{(j)}\right)$$

Die Samples lassen sich daher schreiben als

$$\begin{aligned}(\lambda\gamma)^0\,\delta_0\left(\boldsymbol{v}^{(0)}\right) \;+\; (\lambda\gamma)^1\,\delta_1\left(\boldsymbol{v}^{(1)}\right) \;+\; (\lambda\gamma)^2\,\delta_2\left(\boldsymbol{v}^{(2)}\right) \;+\; \ldots\\ (\lambda\gamma)^0\,\delta_1\left(\boldsymbol{v}^{(1)}\right) \;+\; (\lambda\gamma)^1\,\delta_2\left(\boldsymbol{v}^{(2)}\right) \;+\; \ldots\\ +\; (\lambda\gamma)^0\,\delta_2\left(\boldsymbol{v}^{(2)}\right) \;+\; \ldots\end{aligned}$$

Falls α_k konstant ist, ist es nun möglich, das Verfahren *online* zu implementieren, d.h. wir können die TD-Fehler aufaddieren, sobald die entsprechenden Ereignisse verfügbar sind — bspw. während einer Messung oder Simulation. Ein Vorteil ist unter anderem, dass nicht bis auf das Ende einer Episode gewartet werden muss. Der Markov-Entscheidungsprozess muss nicht einmal episodisch sein. Außerdem ist so in Kombination mit *policy improvement*

ein Lernen nach jedem Zeitschritt möglich. Allgemein sieht die Iteration nun so aus:

$$\boldsymbol{v}^{(k+1)} := \boldsymbol{v}^{(k)} + \alpha_k \boldsymbol{e}^{(k)} \delta_k\Big(\boldsymbol{v}^{(k)}\Big)$$

Wobei $\boldsymbol{e}^{(k)} \in \mathbb{R}^d$ ist und die entsprechenden Faktoren realisiert. Dabei wird $\boldsymbol{e}^{(k)}$ rekursiv berechnet und findet sich unter der Bezeichnung *eligibility trace*.

In der Praxis kommt es natürlich vor, dass Zustände in Ereignissen doppelt auftauchen. In diesem Fall ist die Rekursion zwar weiterhin anwendbar — und das wird auch erfolgreich getan —, allerdings entspricht sie formal nicht mehr der obigen Interpretation. Auch wird α_k in den Implementierungen nicht mehr konstant gehalten. Eine mögliche Variante ist es, beim wiederholten Auftauchen eines Zustandes im Ereignis die Approximationen zu kürzen

$$\boldsymbol{v}^{(k)} + \alpha_k \sum_{i=k}^{\infty} (\lambda\gamma)^{i-k}\, \delta_i\Big(\boldsymbol{v}^{(k)}\Big) \approx \boldsymbol{v}^{(k)} + \alpha_k \sum_{i=k}^{k+N} (\lambda\gamma)^{i-k}\, \delta_i\Big(\boldsymbol{v}^{(k)}\Big)$$

und im Schritt $k+N$ das möglicherweise ebenfalls gekürzte Reststück für eine neue Approximation zu verwenden. Man spricht von Restart − TD(λ) [SS96] im Gegensatz zur oben vorgestellten *akkumulativen* Version. Auch nur jeden Zustand maximal einmal zu updaten, wie bereits beim Monte-Carlo-Verfahren, ist möglich und man erhält unkorrelierte Ereignisse pro Komponente.

Die Stochastische Approximation ist breit anwendbar, und es sind zahlreiche Konvergenzaussagen unter vielen verschiedenen Voraussetzungen und Varianten bekannt. In [BS96, S. 219ff] wird Konvergenz unter allgemeineren Voraussetzungen an $\boldsymbol{e}^{(k)}$ gezeigt, was die erwähnten Fälle einschließen kann, obwohl sie nicht exakt der Vorwärtsinterpretation entsprechen.

Seijen und Sutton haben 2014 eine neue, vielversprechende Variante von TD(λ) vorgeschlagen [SS14], die rekursiv bzw. *online* implementiert werden kann *und* einer dort angegebenen Vorwärtsinterpretation immer exakt entspricht[6].

[6] Die Idee ist zuerst die gekürzten Samples aus den Daten $x_0, u_0, x_1, u_2, \ldots, x_l, u_l$ zu verwenden. Dies wird für jedes l wieder mit dem Startwart neu begonnen. Eine interessante Frage ist, ob die beschriebene Methode sich aus einer Bellman-Gleichung wie oben herleiten lässt, bzw. ob das Verfahren als eine Variante der Stochastischen Approximation interpretiert werden kann.

2.5. SARSA(λ)-Verfahren

Wir haben im vorigen Abschnitt vorgestellt, wie mit Hilfe des TD(λ)-Verfahrens eine Approximation an V^π berechnet werden kann. Natürlich können auch hier, wie bereits beim Monte-Carlo-Algorithmus beschrieben, ε-Strategien benutzt werden, um sicherzustellen, dass auch die Zustände besucht werden, die nach π Wahrscheinlichkeit 0 hätten (siehe Seite 24). Es wird in diesem Fall nicht V^π, sondern V^{π^ε} approximiert. Eine bereits erwähnte Möglichkeit nun eine optimale Strategie zu finden ist es, ε graduell zu verkleinern. Ansonsten können so genannte *off-policy* Varianten vom TD- oder Monte-Carlo-Verfahren genutzt werden. Diese nutzen intuitiv gesagt die Beobachtung einer anderen Strategie (bspw. π^ε) um Rückschlüsse auf π zu treffen. In der Wahrscheinlichkeitstheorie ist dieses Prinzip als *Importance Sampling* bekannt, in [SB98, S. 124ff] finden sich weitere Details.

Um nun mit V^π ein *policy improvement* durchführen zu können, müssen entweder die Zustandsdynamiken bekannt und berechenbar sein, oder man approximiert die Paare $Q(s, a)$ wie auf Seite 23 und kann so eine neue Strategie berechnen. Übrig bleibt die Frage, wann eine Strategie-Iteration stattfindet. Eine Möglichkeit ist, dies nach jedem Zeitschritt im TD(λ)-Verfahren durchzuführen, obwohl die gelernte Wert-Funktion möglicherweise ungenau ist. Bertsekas und Tsitsiklis sprechen in [BT96] deswegen von der *optimistischen* Strategie-Iteration. Der SARSA(λ)-Algorithmus bezeichnet gerade diese Variante.

Um den TD-Fehler zu berechnen ist das folgende, namens gebende Tupel nötig:

$$(s, a, r, s', a')$$

Wobei $s \in S$ der aktuelle Zustand ist, $a \in A_s$ die gewählte Aktion, $r \in \mathbb{R}$ die erhaltene Belohnung, s' der folgende Zustand und a' die Aktion, die darauf gewählt werden wird. Diese wird bspw. durch

$$a' \coloneqq \arg\max_{a \in A_s} Q(s, a)$$

bestimmt, oder nach der entsprechenden ε-Strategie. Im Algorithmus 2.3 geschieht dies in Zeile 9 (*policy improvement*).

Die Abbruchbedingung, Q ist Lösung, stellt in der Praxis Schwierigkeiten dar. Ein grundlegendes Problem ist, dass der Verfahren selbst im Falle von

Konvergenz nicht unbedingt eine optimale Strategie berechnet hat. Zum anderen ist es ein Problem, auf welche Weise eine optimale Strategie erkennbar ist. Die Bellmangleichung würde die Berechnung eines Erwartungswertes erfordern, was in der Regel unmöglich ist, bzw. hat gerade die Approximation von ebendiesem zur Bestimmung von Q geführt. In unseren Experimenten untersuchen wir den zeitlichen Verlauf der Belohnungen, welche das Verfahren während des Lernens oder nach einer festen Anzahl von Episoden erreicht hat. Interessant ist dabei die Frage, wie schnell das Verfahren gelernt hat und wie stabil die Lösung ist.

Für die erwähnte *restart* Variante, ist in Zeile 11 $e(s, a) \leftarrow 1$ zu wählen.

Algorithmus 2.3 SARSA(λ), akkumulativ, α konstant

Input: $\lambda, \gamma \in [0, 1]$, $\alpha \in (0, 1]$

1: Initialisiere $Q(s, a)$ beliebig für alle $(s, a) \in \Gamma$
2: **repeat**
3: $(s, a) \leftarrow$ Startzustand und erste Aktion der Simulation
4: $e(s, a) \leftarrow 0, \quad (s, a) \in \Gamma$
5: **repeat**
6: Führe in der Simulation die Aktion a aus
7: $s' \leftarrow$ Der neue Zustand des Systems
8: $r \leftarrow \mathcal{R}(s, a, s')$
9: $a' \leftarrow$ Wähle nächste Aktion mit Hilfe von Q
10: $\delta \leftarrow \left[r + \gamma Q(s', a')\right] - Q(s, a)$
11: $e(s, a) \leftarrow e(s, a) + 1$ ▷ Akkumulative Variante
12: **for all** $(s, a) \in \Gamma$ **do**
13: $Q(s, a) \leftarrow Q(s, a) + \alpha e(s, a)\delta$
14: $e(s, a) \leftarrow \lambda\gamma e(s, a)$
15: **end for**
16: $(s, a) \leftarrow (s', a')$
17: **until** s ist terminal
18: **until** Ziel erreicht

Zusammenfassung

Wir haben gesehen, dass SARSA(λ) als eine approximative Strategie-Iteration aus Abschnitt 2.2.1 interpretiert werden kann (Wechsel von Berechnung

V^{π_k} und π_{k+1}). Der erste Schritt im Kapitel über Monte-Carlo-Verfahren war es, V^{π_k} durch Samples zu approximieren, d.h. ohne Kenntnis über das Verhalten des Systems zu haben. Durch eine Aufteilung der Wert-Funktion in Zustands-Aktion-Paare $Q(s, a)$, ist in diesem Fall ebenfalls eine Berechnung von π_{k+1} möglich. Die TD-Methode hat schließlich die Samples für die erhaltene Belohnung per *bootstraping* verkürzt auf $r + \gamma Q$ und so zum einen eine *online* Berechnung ermöglicht und zum anderen durch Gewichtung verschiedener Längen eine ganze Klasse von Methoden erzeugt: $TD(\lambda)$.

Offensichtlich sind zahlreiche anderen Varianten dieses Algorithmus möglich, einige Möglichkeiten haben wir bereits erwähnt oder sind auf Grund der Darstellungen denkbar. So ist in Algorithmus 2.3 die Lernrate α noch konstant und die Implementierung für unendliche oder sehr große S ist in dieser Form entweder unmöglich oder unpraktikabel. Diesem Problem wird sich der nächste Abschnitt widmen. Eine Übersicht und ein experimenteller Vergleich verschiedener aktueller Algorithmen im Verstärkenden Lernen, findet sich in [DNP14].

2.5.1. Lineare Funktionsapproximation und Stochastische Gradienten-Abstiegs-Verfahren

Betrachtet man Reinforcement-Learning-Probleme, die bspw. durch physikalische Systeme — wie das Steuern eines Roboters — beschrieben werden, so treten in der Regel Zustandsräume S auf, die überabzählbare Teilmengen von $\mathbb{R}^d$ sind. Eine mögliche Formulierung dieser Probleme im Fall einer kontinuierlich modellierten Zeit geschieht über die Hamilton-Jacobi-Bellman-Gleichung, die als zeitkontinuierliche Variante der Bellman-Gleichung aufgefasst werden kann [Ber95a, S. 91].

Ist die Zeit jedoch diskretisiert, liegt ein Lösungsversuch mit den hier beschriebenen Verfahren für das Reinforcement-Learning-Problem nahe. Eine dabei auftretende Schwierigkeit ist die Speicherung bzw. Darstellung der Wert-Funktion: Es muss $Q(s, a)$ für alle $(s, a) \in \Gamma$ gespeichert und Γ in jeder Iterieren durchlaufen werden. Eine naheliegende Lösung ist die Diskretisierung von S, zum Beispiel durch Aufteilung mit Gittern verschiedener Formen. Diese Möglichkeit ist bekannt als *Tile Coding* [SB98, S. 204ff]. Nachteile sind, dass die Gitter durch zahlreiche Parameter spezifiziert werden müssen und eine naive Aufteilung mit der Anzahl an Dimensionen von S zu schnell wächst.

Wir suchen im Folgenden eine endliche Repräsentation der Zustände, die wir durch eine Funktion $\boldsymbol{F}\colon S \to \mathbb{R}^n$, $s \mapsto (f_1(s), \ldots, f_n(s))$ beschreiben. Das Ziel ist es, die Wert-Funktion mit Hilfe einer Linearkombination der f_i zu approximieren. Wir suchen also von s unabhängige Koeffizienten

$$\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n) \in \mathbb{R}^n$$

so dass

$$V(s) \approx \sum_{i=1}^{n} \theta_i f_i(s, a) = \boldsymbol{\theta}^\top \boldsymbol{F}(s) =: V_{\boldsymbol{\theta}}^{\pi}(s)$$

Die Komponenten f_i von $\boldsymbol{F}$ werden *Features* genannt. Im allgemeinen Umfeld des Maschinellen Lernens sprich man bei Features von Eigenschaften einer Struktur. Beispielsweise können in der Gesichtserkennung auf dem Bild eines Gesichtes die Eigenschaften »Augenfarbe«, »Nasenausmaße« usw. als Feature bezeichnet und benutzt werden.

Die Wahl der Features ist ein wichtiges Erfolgskriterium und nicht nur für unendliche S von Interesse. Sie sollten den Zustandsraum so weit wie möglich reduzieren in dem Sinn, dass er durch eine kleine, endliche Menge von Zahlen gut für das Problem charakterisiert wird. Wie man an dem Beispiel Gesichtserkennung sieht, kann die Auswahl und Berechnung der Features ein eigenes, schwieriges Unterproblem sein. Wir bleiben in diesem Abschnitt bei der mathematisch abstrakten Definition. In Kapitel 3 werden wir dafür (endliche) Čebyšëv- und Fourier-Basen betrachten, die pro Summand ausgewertet als Features benutzt werden.

Zunächst passen wir SARSA(λ) für die Benutzung von Features an. Da S größer ist als $\boldsymbol{\theta}$, wollen wir nicht fordern, dass die Approximation punktweise beliebig gut wird (gleichmäßige Konvergenz). Um die verfügbaren Freiheitsgrade möglichst effizient zu nutzen, ist unser Ziel die Güte der Approximation an besonders »wichtigen« Zuständen stärker zu gewichten. Wenn

$$\mu\colon \mathcal{B}(S) \to [0, 1]$$

ein Maß mit $\mu(S) = 1$ ist, so können wir die L^p-Norm betrachten

$$\begin{aligned} \|\cdot\|_{L^p}\colon L^p(S, \mathcal{B}(S), \mu) &\to [0, \infty) \\ f &\mapsto \left(\int_S |f|^p \,\mathrm{d}\mu\right)^{1/p} \end{aligned}$$

Dabei soll μ beschreiben, wie »wichtig« Zustände $X \in \mathcal{B}(S)$ sind — wir kommen später darauf zurück. Wir sprechen im folgenden von L^p und meinen $L^p(S, \mathcal{B}(S), \mu)$.

Mit dieser Norm lassen sich nun Fehler messen:

Definition 2.5.1. Sei $\pi \in \Pi^{\mathrm{s}}_M$ und μ, $\boldsymbol{F}$ wie oben mit $f_i \in L^2$ für alle $i = 1, \ldots, n$. Sei

$$\tilde{V}(F) \coloneqq \operatorname{span}\{f_1, \ldots, f_n\} \subseteq L^2$$

Wir bezeichnen mit $\mathcal{P}\colon L^2 \to \tilde{V}(F)$ die durch

$$\|f - \mathcal{P}f\|_{L^2} = \inf_{\boldsymbol{\theta} \in \mathbb{R}^n} \|f - V^\pi_{\boldsymbol{\theta}}\|_{L^2} \quad f \in L^2$$

eindeutig bestimmte Projektion. Falls $V^\pi \in L^2$ ist, so definieren wir

$$\mathrm{MSE}(\boldsymbol{\theta}) \coloneqq \frac{1}{2}\|V^\pi - V^\pi_{\boldsymbol{\theta}}\|^2_{L^2}$$
$$\mathrm{MSPE}(\boldsymbol{\theta}) \coloneqq \frac{1}{2}\|\mathcal{P}V^\pi - V^\pi_{\boldsymbol{\theta}}\|^2_{L^2}$$

Bevor wir zur Beschreibung kommen, überprüfen wir die Wohldefiniertheit.

Lemma 2.4. *Die Abbildung $\mathcal{P}$ ist wohldefiniert und linear.*

Beweis. Wir möchten den Projektionssatz für Hilberträume auf $\tilde{V}(F) \subseteq L^2$ anwenden. Dafür muss $\tilde{V}(F)$ konvex und abgeschlossen sein.

Da $\tilde{V}(F)$ ein Vektorraum ist, ist die Konvexität klar. Ein bekanntes Resultat der Funktionalanalysis besagt, dass endlich-dimensionale Unterräume von normierten Räumen vollständig sind. Auf metrischen Räumen ist Vollständigkeit jedoch hinreichend für Abgeschlossenheit.

Sei $\{u_1, \ldots, u_m\}$ eine Orthonormalbasis von $\tilde{V}(F)$. Auf Grund der Orthogonalitätseigenschaften von $\mathcal{P}$ gilt

$$\mathcal{P}(f) = \sum_{i=1}^{m} \langle f, u_i \rangle_{L^2} \, u_i$$

Also ist $\mathcal{P}$ linear. □

Um die Güte einer Approximation zu messen, verwenden wir den mittleren, quadratischen Fehler (*MSE*) und nehmen zum Zweck dieser Herleitung an, dass $V^\pi \in L^2$ ist. Wir haben also das Minimierungsproblem

$$\inf_{\boldsymbol{\theta}\in\mathbb{R}^n} \mathrm{MSE}(\boldsymbol{\theta})$$

Das ist äquivalent zur Minimierung von MSPE, da sich die Minima lediglich um eine Konstante unterscheiden (Satz des Pythagoras für Skalarprodukte). Falls die Features linear unabhängig sind (als Funktionen), gibt es genau einen Minimierer $\boldsymbol{\theta}$.

Bemerkung. Die Entscheidung MSE zu minimieren werden wir später relativieren. Sie ist nicht kanonisch, da es ebenso gute Alternativen gibt. Auch ist sie alleine nicht hinreichend oder notwendig für Konvergenz gegen eine optimale Strategie in SARSA. Diese Vorgehensweise dient zur Motivation des Verfahrens.

Lemma 2.5. *Es gilt*

$$\begin{aligned}\nabla\mathrm{MSPE}(\boldsymbol{\theta}) &= \int_S \left(\boldsymbol{\theta}^\top \boldsymbol{F}(s) - \mathcal{P}V^\pi(s)\right) \boldsymbol{F}(s)\,\mathrm{d}\mu(s)\\ &= \mathbb{E}_\mu\left[\left(\boldsymbol{\theta}^\top \boldsymbol{F} - \mathcal{P}V^\pi(s)\right)\boldsymbol{F}\right]\end{aligned}$$

Wobei $\mathbb{E}_\mu[\,\cdot\,]$ den Erwartungswert bzgl. des Maßes μ bezeichnet[7].

Beweis. Sei $g(\boldsymbol{\theta}, s) \coloneqq \left(\mathcal{P}V^\pi(s) - \boldsymbol{\theta}^\top \boldsymbol{F}(s)\right)^2$. Da der Maßraum endlich ist, ist $g(\boldsymbol{\theta}, \cdot) \in L^1$ für alle $\boldsymbol{\theta} \in \mathbb{R}^n$. Außerdem existiert

$$\frac{\partial g}{\partial \theta_i}(\boldsymbol{\theta}, s) = 2\left(\boldsymbol{\theta}^\top \boldsymbol{F}(s) - \mathcal{P}V^\pi(s)\right) f_i(s)$$

für alle $s \in S$. Es ist weiter für $0 < |t| < \delta$ und $i = 1, \ldots, n$:

$$\begin{aligned}\left|\frac{g(\boldsymbol{\theta} + t\mathbb{1}_i, s) - g(\boldsymbol{\theta}, s)}{t}\right| &= \left|t f_i(s)^2 + 2 f_i(s)\left(\mathcal{P}V^\pi(s) - \boldsymbol{\theta}^\top \boldsymbol{F}(s)\right)\right|\\ &\leq \delta f_i(s)^2 + 2\left|f_i(s)\right|\left|\mathcal{P}V^\pi(s) - \boldsymbol{\theta}^\top \boldsymbol{F}(s)\right| \in L^1\end{aligned}$$

[7] Die Doppeldeutigkeit in der Notation mit $\mathbb{E}_\pi[\,\cdot\,]$ bitten wir zu verzeihen.

Für die L^1-Abschätzung wurde im zweiten Term einmal die Hölder-Ungleichung benutzt. Mit dem Satz der majorisierten Konvergenz erhalten wir nun die Aussage. □

Um das Minimierungsproblem zu lösen, suchen wir einen kritischen Punkt von MSPE. Wir können dafür ein Gradienten-Verfahren verwenden:

$$\begin{aligned}\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \alpha_k \nabla \mathrm{MSPE}(\boldsymbol{\theta}_k) \\ &= \boldsymbol{\theta}_k + \alpha_k \mathbb{E}_\mu \left[\left(\mathcal{P}V^\pi - \boldsymbol{\theta}_k^\top \boldsymbol{F} \right) \boldsymbol{F} \right]\end{aligned}$$

Wir wollen versuchen, der Idee der Stochastischen Approximation folgend, hieraus eine Iterationsvorschrift zu motivieren. Sei $\{x_k, u_k\}_{k=0}^\infty$ ein Ereignis der Markov-Kette. Die Annahme ist nun, dass μ so gewählt ist, dass die in der Simulation auftauchenden Zustände nach μ verteilt sind.

Die Wahl für μ ist eine so genannte *Invariante Verteilung* der Markov-Kette $(X_n)_{n\in\mathbb{N}}$, sofern sie existiert [Kle13, Abschnitt 17.6]. Sie ist eine Verteilung mit

$$\int_S \mathrm{P}_s[\, X_1 \in B \,] \,\mathrm{d}\mu(s) = \mu(B) = \int_S \underbrace{\mathrm{P}_s[\, X_0 \in B \,]}_{=\mathbb{1}_B(s)} \mathrm{d}\mu(s)$$

für alle $B \in \mathcal{B}(S)$. Intuitiv heißt das, falls X_0 nach μ verteilt ist, so auch alle folgenden X_n. Unter einigen Bedingungen ist bekannt, dass die Verteilung der X_n gegen eine Invariante Verteilung konvergiert. Also kann μ als ein Maß für die Häufigkeit der auftretenden Zustände gesehen werden.

Sei ab hier

$$\boldsymbol{\phi}_i \coloneqq \boldsymbol{F}(x_i)$$

der Feature-Vektor für den Zustand x_i. Nehmen wir zunächst an, dass $\mathcal{P}V^\pi(s)$ bekannt ist. Dann lautet die Iteration, wenn wir wie bei der stochastischen Approximation nur ein Sample pro Schritt wählen um den Erwartungswert zu nähern:

$$\boldsymbol{\theta}_{k+1} \coloneqq \boldsymbol{\theta}_k + \alpha_k \left[\mathcal{P}V^\pi(x_k) - \boldsymbol{\theta}_k^\top \boldsymbol{\phi}_k \right] \boldsymbol{\phi}_k \tag{2.22}$$

Nun ist in der Implementierung $\mathcal{P}V^\pi(s)$ nicht exakt bekannt — diese Berechnung ist gerade das Ziel des Verfahrens. Unter technischen Voraussetzungen kann man zeigen [TV97, Lemma 6], dass die projizierte Variante

von (2.20) einen Fixpunkt J (in $\tilde{V}(F)$) besitzt mit[8]

$$\|J - V^\pi\|_{L^2} \leq c\|\mathcal{P}V^\pi - V^\pi\|_{L^2} \tag{2.23}$$

Wir haben also mit der Linearität von $\mathcal{P}$

$$\begin{aligned} J(s) &= \mathcal{P}\mathbb{E}_\pi\left[\sum_{i=0}^{\infty}(\lambda\gamma)^i\, d_i \;\middle|\; x_0 = s\right] + J(s) \\ d_i &= \mathcal{R}(x_k, u_k, x_{k+1}) + \gamma J(x_{k+1}) - J(x_k) \end{aligned} \tag{2.24}$$

Die Abschätzung (2.23) gibt uns mit (2.24) den Anlass nun für $\mathcal{P}V^\pi$ die folgenden Samples zu verwenden:

$$\begin{aligned} &\sum_{i=k}^{\infty}(\lambda\gamma)^{i-k}\delta_i + \boldsymbol{\theta}_k^\top \boldsymbol{\phi}_k \\ \delta_i &\coloneqq \mathcal{R}(x_i, u_i, x_{i+1}) + \gamma\boldsymbol{\theta}_k^\top \boldsymbol{\phi}_{i+1} - \boldsymbol{\theta}_k^\top \boldsymbol{\phi}_i \end{aligned}$$

Indem wir in (2.22) dies nun für $\mathcal{P}V^\pi$ einsetzen, erhalten wir

$$\boldsymbol{\theta}_{k+1} \coloneqq \boldsymbol{\theta}_k + \alpha_k\left[\sum_{i=k}^{\infty}(\lambda\gamma)^{i-k}\delta_i\right]\boldsymbol{\phi}_k \tag{2.25}$$

Diese Iteration lässt sich nun analog zu Abschnitt 2.4.2 in eine rekursive Alternativ-Form bringen. In diesem Fall benutzen die TD-Fehler δ_i i.A. in jedem Schritt der Rekursion (entspricht einem Summanden in (2.25)) eine andere Gewichtung $\boldsymbol{\theta}$, selbst in dem Fall, dass Zustände nicht doppelt besucht werden. Sind die Features eher lokal, also sind nur wenige $f_i(s)$ verschieden von Null, wird diese Abweichung reduziert.

In [TV97] wird gezeigt[9], dass das Verfahren unter geeigneten Voraussetzungen gegen ein J konvergiert, das die Abschätzung (2.23) erfüllt. Dort ist ebenfalls ein Divergenzbeispiel für einen nicht-linearen Ansatz. Wählt man statt MSPE einen anderen Fehler oder ein anderes Verfahren zur Lösung des Minimierungsproblems, so entstehen andere Verfahren. Eine Übersicht ist in [DNP14], auch über theoretische Resultate. Siehe ebenfalls [BT96, S. 284ff].

[8] Dort wird $\gamma \in (0, 1)$ angenommen und nur für endliche S gerechnet.

[9] Eine ausführliche und verallgemeinerte Version ist in [Van98].

Um SARSA nun mit Features zu implementieren, muss $Q(s,a)$ an Stelle von V^π approximiert werden: Sei $A = \{a_i\}_{i=0}^{m-1}$ die Menge der Aktionen. Wird im Schritt i nun die Aktion a_k gewählt, so ist der Feature-Vektor

$$\boldsymbol{\phi}_i \coloneqq \sum_{j=1}^{n} f_j(x_i) \mathbb{1}_{kn+j} \in \mathbb{R}^{nm} \tag{2.26}$$

Anschaulich wird der Feature-Vektor so in m Blöcke aufgeteilt, wobei der Block k mit den Werten der Features gefüllt wird, wenn die Aktion a_k eintritt. Alle anderen Blöcke bleiben leer. Eine spezielle Wahl von Features, die für Zustand-Aktion-Paare definiert sind, ist natürlich ebenfalls denkbar, wird hier allerdings nicht weiter verfolgt.

Zudem ändert sich in SARSA nun μ in jedem Schritt, da π ebenfalls iteriert wird.

2.5.2. Adaptive Lernratensteuerung

Ein Problem beim TD-Verfahren ist die Wahl der Lernrate α_k, da sie um Konvergenz zu erreichen nicht zu schnell und nicht zu langsam abfallen darf, siehe (2.18). Es sind zwar Konvergenzaussagen auch für konstante α_k bekannt, jedoch funktionieren abfallende Werte in der Praxis und in Kombination mit Features besser. Dabney und Barto haben eine adaptive Lernratensteuerung für das TD-Verfahren vorgestellt [DB12]. Das heißt der Wert wird während des Lernens auf Basis erhaltener Daten mit Hilfe einer Heuristik angepasst. Diese werden wir hier kurz präsentieren und auch später verwenden. Das Verfahren geht von einer linearen Approximation der Wert-Funktion wie oben beschrieben aus. Wir übernehmen im Folgenden die Notation von oben, dabei spielt es keine Rolle ob wir $Q(s,a)$ oder V^π approximieren.

Ziel des Verfahrens ist es, die α_k nicht zu groß und nicht zu klein zu wählen. Für zu kleine α ist die Konvergenz langsam, für zu große kann das Verfahren divergieren. Wir beschränken uns auf skalare α_k, d.h. eine spezielle Anpassung pro einzelner Komponente findet hier nicht statt. Die Idee bzw. Intuition der Lernratensteuerung ist:

> Wenn wir im Schritt k einen neuen Wert der Parameter $\boldsymbol{\theta}_{k+1}$ berechnet haben, so sollten sich die bisherigen TD-Fehler bei

einer Neuberechnung mit $\boldsymbol{\theta}_{k+1}$ verbessern, in dem Sinne, dass

$$\delta_i' = \left[r_i + \gamma \boldsymbol{\theta}_{k+1}^\top \boldsymbol{\phi}_{i+1}\right] - \boldsymbol{\theta}_{k+1}^\top \boldsymbol{\phi}_i \tag{2.27}$$

gegenüber

$$\delta_i = \left[r_i + \gamma \boldsymbol{\theta}_{k}^\top \boldsymbol{\phi}_{i+1}\right] - \boldsymbol{\theta}_{k}^\top \boldsymbol{\phi}_i$$

im Absolutbetrag nicht größer ist und kein Vorzeichenwechsel stattfindet, für $i \leq k$.

Ein Vorzeichenwechsel würde bedeuten die Lernrate war zu groß und wir sind beim Vergleich der neuen Approximation $\boldsymbol{\theta}_{k+1}\boldsymbol{\phi}_{i+1}$ mit der Messung über das Ziel (Gleichheit) hinaus, die Anpassung hätte weniger stark sein müssen.

Um nun eine Schranke für α_k zu berechnen, betrachten wir (2.27):

$\boldsymbol{\theta}_{k+1} = \ldots$

$$\begin{aligned}
\delta_i' &= \left[r_i + \gamma \boldsymbol{\theta}_{k+1}^\top \boldsymbol{\phi}_{i+1}\right] - \boldsymbol{\theta}_{k+1}^\top \boldsymbol{\phi}_i \\
&= \left[r_i + \gamma \left(\boldsymbol{\theta}_k + \alpha_k \delta_k e_k\right)^\top \boldsymbol{\phi}_{i+1}\right] - \left(\boldsymbol{\theta}_k + \alpha_k \delta_k e_k\right)^\top \boldsymbol{\phi}_i \\
&= \left(\left[r_i + \gamma \boldsymbol{\theta}_k^\top \boldsymbol{\phi}_{i+1}\right] - \boldsymbol{\theta}_k^\top \boldsymbol{\phi}_i\right) + \alpha_k \delta_k e_k^\top \left(\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i\right) \\
&= \delta_i + \alpha_k \delta_k e_k^\top \left(\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i\right)
\end{aligned} \tag{2.28}$$

Die Bedingung, dass $\left|\delta_i'\right| \leq \left|\delta_i\right|$ und dass kein Vorzeichenwechsel stattgefunden hat, führt für den Fall $\delta_i = 0$ auf $\delta_i' = 0$. Wenn $\delta_k e_k^\top \left(\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i\right) \neq 0$ ist, müsste also $\alpha_k = 0$ gewählt werden. Sonst lässt sich für $\delta_i \neq 0$ die Bedingung schreiben als

$$\begin{aligned}
&\frac{\delta_i'}{\delta_i} \in [0, 1] \\
\overset{(2.28)}{\Leftrightarrow} \quad &\alpha_k e_k^\top \left(\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i\right) \frac{\delta_k}{\delta_i} \in [-1, 0]
\end{aligned} \tag{2.29}$$

Ist $\delta_k e_k^\top \left(\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i\right) > 0$, so müsste auch hier $\alpha_k = 0$ gewählt werden.

Für $\delta_k e_k^\top \left(\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i\right) < 0$ und $i = k$ können wir (2.29) umformen und erhalten

$$0 < \alpha_k \leq -\frac{1}{e_k^\top \left(\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i\right)} = \left|e_k^\top \left(\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i\right)\right|^{-1} \tag{2.30}$$

als äquivalente Bedingung für unsere Forderungen für $i = k$. Wir können die Schranke (2.30) durch geeignete[10] $i < j$ mit (2.29) verkleinern:

$$\alpha_k \leq \min \left\{ \frac{-\delta_i}{\delta_k e_k^\top (\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i)} \,\middle|\, 0 \leq i \leq k,\ \delta_i \neq 0, e_k^\top (\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i) \frac{\delta_k}{\delta_i} < 0 \right\}$$

Als Heuristik für die Schrittweitensteuerung ignorieren wir nun die Fälle, wo nach obigen Forderungen $\alpha_k = 0$ sein müsste und setzen stattdessen

$$\alpha_k \coloneqq \begin{cases} \min \left\{ \alpha_{k-1}, \left| e_k^\top (\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i) \right|^{-1} \right\} & \text{falls } e_k^\top (\gamma \boldsymbol{\phi}_{i+1} - \boldsymbol{\phi}_i) < 0 \\ \alpha_{k-1} & \text{sonst} \end{cases}$$

Mit Startwert $\alpha_0 \coloneqq 1$.

In [DB12] finden sich numerische Experimente zum Vergleich mit anderen Schrittweitensteuerungen. Der Vorteil dieser Methode ist, dass der Parameter α_k völlig entfällt.

[10] In [DB12] wird ein wenig ungenau mit Vorzeichen in Kombination mit Äquivalenzumformungen von Ungleichungen umgegangen. Die hier angegebene Wahl berücksichtigt, dass i.A. $\operatorname{sign}(\delta_i) \neq \operatorname{sign}(\delta_k)$ ist, bzw. $\delta_i = 0$ zulässig ist. In der Tat ist nicht zu erwarten, dass die neuen Parameter $\boldsymbol{\theta}_k$ gleichzeitig eine bessere Approximation für alle $s \in S$ ermöglichen.

Algorithmus 2.4 SARSA(λ), akkumulativ, Lernratensteuerung, Features

Sei $A = \{a_k\}_{k=0}^{m-1}$. Wir bezeichnen mit $\boldsymbol{\phi}(s,a)$ den Feature-Vektor aus (2.26) und mit $\boldsymbol{e}[a]$ die Indizes $kn+1$ bis $kn+n$ von $\boldsymbol{e}$, wobei $a = a_k$.

Input: $\lambda, \gamma \in [0,1]$, $\boldsymbol{F}\colon S \to \mathbb{R}^n$

$\alpha \leftarrow 1$
$\boldsymbol{\theta} \leftarrow 0 \in \mathbb{R}^{nm}$
repeat
  $(s,a) \leftarrow$ Startzustand und erste Aktion der Simulation
  $\boldsymbol{e} \leftarrow 0 \in \mathbb{R}^{nm}$
  repeat
    Führe in der Simulation die Aktion a aus
    $s' \leftarrow$ Der neue Zustand des Systems
    $r \leftarrow \mathcal{R}(s,a,s')$
    $a' \leftarrow$ Wähle nächste Aktion mit $\boldsymbol{\theta}^\top \boldsymbol{\phi}(s,\cdot)$
    $\delta \leftarrow \left[r + \gamma \boldsymbol{\theta}^\top \boldsymbol{\phi}(s',a')\right] - \boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a)$
    $e[a] \leftarrow e[a] + 1$
    if $\boldsymbol{e}^\top(\gamma\boldsymbol{\phi}(s',a') - \boldsymbol{\phi}(s,a)) < 0$ **then**
      $\alpha \leftarrow \min\left\{\alpha, \left|\boldsymbol{e}^\top(\gamma\boldsymbol{\phi}(s',a') - \boldsymbol{\phi}(s,a))\right|^{-1}\right\}$
    end if
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{e} \delta \boldsymbol{\phi}(s,a)$
    $\boldsymbol{e} \leftarrow \lambda\gamma\boldsymbol{e}$
    $(s,a) \leftarrow (s',a')$
  until s ist terminal
until Q ist Lösung

3. Čebyšëv- und Fourier-Approximation

Wir haben in Abschnitt 2.5.1 gesehen, wie wir einen großen Zustandsraum $S \subseteq \mathbb{R}^d$ auf die Speicherung eines Koeffizientenvektors $\boldsymbol{\theta}$ reduzieren können, falls so genannte Features gegeben sind:

$$\begin{aligned} \boldsymbol{F}\colon S &\to \mathbb{R}^n \\ s &\mapsto (f_1(s), f_2(s), \ldots, f_n(s)) \end{aligned}$$

Dabei heißen die f_i Basis-Funktionen.

Diese Features sollten im Idealfall folgende Eigenschaften aufweisen:

1. Sie sollten effizient berechenbar sein und die Wert-Funktionen »gut« approximieren.
2. Die Anzahl sollte möglichst klein sein (hier mit n bezeichnet).
3. Sie sollten von möglichst wenigen Parametern abhängen, die nicht automatisch gewählt werden (wie λ und γ bei SARSA).
4. Sie sollten möglichst universell sein, also auf eine breite Klasse von Zustandsräumen anwendbar sein.

Es ist üblich Basis-Funktionen per Hand zu konstruieren, so dass *problemabhängig* eine Reduzierung erreicht wird, die hoffentlich die relevanten Informationen immer noch darstellt. TD-Gammon [Tes95] konnte erst mit der Hinzufügung von speziell gewählten Features ein Weltklasse-Niveau erreichen. Natürlich sind Methoden wünschenswert, die nicht einer manuellen Vorarbeit bedürfen, so dass lediglich eine Belohnungsfunktion, Zustände und Aktionen vorgegeben werden. Es gibt Vorschläge, problemabhängige Basen im Verfahren automatisch zu konstruieren (einen Vorschlag, sowie Verweise auf andere Verfahren finden sich in [MM07]), allerdings werden

wir dies hier nicht weiter verfolgen. In [KOT11] wurden klassische Fourier-Basen als Features beschrieben und experimentell getestet. Die Fourier-Basis hat, gemessen an ihrer Einfachheit, in den untersuchten Experimenten erstaunlich gute Ergebnisse im Vergleich zu bisherigen Basen gezeigt, obwohl sie üblicherweise für periodische Funktionen benutzt wird. Die mit der Fourier-Basis verwandten Čebyšëv-Polynome haben in nicht-periodischen Fällen erfahrungsgemäß bessere Approximationseigenschaften, was uns veranlasst hat diese als Basis zu testen (Kapitel 4). In diesem Kapitel stellen wir die Čebyšëv-Polynome vor und vergleichen sie in einem allgemeinen Approximationsumfeld mit der Fourier-Basis.

Die Frage der Norm In Abschnitt 2.5.1 haben wir das Stochastische Gradienten-Verfahren mit Hilfe der L^2-Norm bzgl. des Maßes μ hergeleitet. Wie bereits erwähnt, ändert sich μ jedoch (sofern überhaupt existent), so bald wir π iterieren, was im SARSA-Algorithmus in jedem Schritt geschieht. Der direkte Wechsel von *policy evaluation* und *improvement* stellt somit für die Analyse eine Herausforderung dar. Unabhängig von diesem *optimistischen*, schnellen Wechsel ist jedoch nicht gesichert, dass eine bzgl. der L^2-Norm optimale Approximation auch zu Konvergenz gegen eine optimale Strategie führt: Viele Wert-Funktionen enthalten Zustände, die zwar in S euklidisch nahe sind, aber deren Wert sich stark unterscheidet, Unstetigkeiten sind üblich. Ist die Approximation $\tilde{Q}$ der exakten Wert-Funktion nun in der L^2-Norm gut, könnte der punktweise Fehler dennoch sehr groß sein, zu groß um eine Strategie zu verbessern. Es gilt zwar (2.23), allerdings ist die Strategie

$$\tilde{\pi}(s) \coloneqq \arg\max_{a \in A_s} \tilde{Q}(s, a)$$

nun abhängig von der punktweisen Auswertung und könnte bedeutend schlechter sein. Die Forderung punktweiser Genauigkeit im Sinne von $\|\cdot\|_\infty$ wäre L^∞ bzw. gleichmäßige Konvergenz. Es wird klar, dass allein die Formulierung von direkten, mathematischen Forderungen an eine Basis nicht trivial ist, zumal über die Wert-Funktionen wenig Eigenschaften (bspw. Stetigkeit) bekannt bzw. realistisch sind.

Wir stünden also vor der Aufgabe, eine unbekannte Klasse an Funktionen (die Wert-Funktionen) bzgl. einer unbekannten Norm bestmöglich zu approximieren. Um dennoch qualitative Aussagen über Fourier- und

Čebyšëv-Basen machen zu können, geben wir allgemeine Approximationseigenschaften an, die einen Eindruck der Stärken und Schwächen vermitteln sollen.

3.1. Definition und elementare Eigenschaften

Sei

$$F\colon \partial B_1(0) \to \mathbb{R}$$

eine Funktion auf dem Einheitskreis. Wir können eine 2π-periodische Funktion auf $\mathbb{R}$ erhalten, indem wir $f(x) \coloneqq F(\mathrm{e}^{\mathrm{i}x})$ definieren. Es ist bekannt, dass sich f in einem gewissen Sinne mit der Fourier-Basis $\left\{\mathrm{e}^{\mathrm{i}nx}\right\}_{n\in\mathbb{Z}}$ identifizieren lässt:

$$f(x) \sim \sum_{n=-\infty}^{\infty} c_n \mathrm{e}^{\mathrm{i}nx}$$

Dabei sind zahlreiche Formen der Konvergenz bekannt: Konvergenz in verschiedenen Normen (L^p, punktweise, gleichmäßig) unter entsprechenden Voraussetzungen an f, ebenso wie Konvergenz unter verschiedenen Summationsarten der Fourier-Reihe (Cesáro- und Abel-Summation). Die Theorie ist sehr umfangreich mit einem breiten Anwendungsbereich, welcher beständig erweitert wird.

Falls f gerade (also F spiegelsymmetrisch zur x-Achse) ist, fallen alle ungeraden Anteile der Basis weg und die Darstellung reduziert sich auf

$$f(x) \sim \sum_{n=0}^{\infty} a_n \cos(nx), \quad a_n \coloneqq \begin{cases} c_n + c_{-n}, & \text{falls } n \neq 0 \\ c_0 & \text{sonst} \end{cases} \tag{3.1}$$

falls die Summationsreihenfolge getauscht werden darf. Mit dem Koordinatenwechsel

$$\Psi\colon [-1,1] \to [0,\pi], \quad x \mapsto \arccos x$$

erhalten wir eine Funktion $g \coloneqq f \circ \Psi$, welche eine Hälfte von F und wegen der Symmetrie damit ganz F darstellt. Für $\phi \in [0,\pi]$ gilt somit, wenn $x = \cos\phi$ ist:

$$g(x) = f(\phi) \sim \sum_{n=0}^{\infty} a_n \cos(n\phi) = \sum_{n=0}^{\infty} a_n \cos(n \arccos x) \tag{3.2}$$

Diese Vorgehensweise lässt sich auch umkehren: Falls $g\colon [-1,1] \to \mathbb{R}$ ist, so können wir durch

$$F\colon \partial B_1(0) \to \mathbb{R}, \qquad \mathrm{e}^{\mathrm{i}\phi} \mapsto g(\cos\phi)$$

eine Funktion auf dem Einheitskreis definieren und erhalten wieder eine Identifikation für g wie in (3.2), bzw. eine gerade, 2π-periodische Funktion f.

Definition 3.1.1. Für $n = 0, 1, 2, \ldots$ heißt

$$\begin{aligned} T_n\colon [-1,1] &\to \mathbb{R} \\ x &\mapsto \cos(n \arccos x) \end{aligned}$$

das *n-te Čebyšëv-Polynom*, wobei $\arccos\colon [-1,1] \to [0,\pi]$ bijektiv sein soll.

Sei p_n die Menge der reellen, eindimensionalen Polynome vom Grad n oder kleiner.

Es ist $T_0(x) = 1$ und $T_1(x) = x$. Dass auch für höhere n die Čebyšëv-Polynome tatsächlich Polynome im gewohnten Sinne sind, zeigt das folgende Lemma.

Lemma 3.1. *Für alle $n = 0, 1, 2, \ldots$ und alle $x \in [-1,1]$ gilt:*

$$T_{n+1}(x) = 2x\,T_n(x) - T_{|n-1|}(x) \tag{3.3}$$

Beweis. Für $n = 0$ ist die Aussage klar. Mit dem Additionstheorem folgt nun für $n > 0$:

$$\begin{aligned} T_{n+1}(x) + T_{|n-1|} &= \cos\big((n+1)\arccos x\big) + \cos\big((n-1)\arccos x\big) \\ &= 2\cos\big(n\arccos(x)\big)\cos\big(\arccos(x)\big) \\ &= 2x\,T_n(x) \end{aligned}$$

□

Also ist bspw. $T_2(x) = 2x^2 - 1$ und $T_4(x) = 8x^4 - 8x^2 + 1$. Wir setzen im Folgenden T_n auf $\mathbb{R}$ durch das entsprechende Polynom fort.

Korollar 3.2. *Die Funktionen $\{T_k\}_{k=0}^n$ bilden eine Basis für p_n.*

Beweis. Der Vektorraum p_n hat Dimension $n+1$. Nach (3.3) hat T_k Grad k, also ist $\{T_k\}_{k=0}^n$ linear unabhängig. □

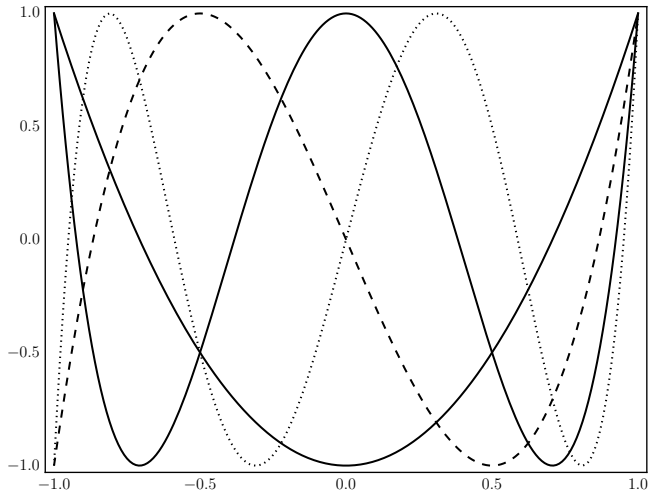

Abbildung 3.1.: Die Polynome T_2 bis T_5.

Orthogonalität und Konvergenz

Da die geraden Teile der Fourier-Basis, also $\{\cos(n\cdot)\}_{n\in\mathbb{N}}$, bzgl. des Standard-$L^2$-Skalarproduktes orthogonal sind, überträgt sich diese Eigenschaft mit dem Diffeomorphismus

$$\Psi\colon (-1,1) \to (0,\pi), \quad x \mapsto \arccos(x)$$

auch auf die Čebyšëv-Polynome: Für $f, g \in L^2((0,\pi))$ gilt

$$\int_{(0,\pi)} fg \,\mathrm{d}\lambda = \int_{(-1,1)} ((fg)\circ\Psi)\,|\Psi'|\,\mathrm{d}\lambda = \int_{(-1,1)} \frac{f(\Psi(x))g(\Psi(x))}{\sqrt{1-x^2}}\,\mathrm{d}\lambda(x)$$

Dabei bezeichnet λ das Lebesgue-Maß. Sei nun

$$L^2_\Psi := \left\{ f\colon (-1,1)\to\mathbb{R} \,\middle|\, f\circ\Psi^{-1} \in L^2((0,\pi)) \right\}$$

und

$$\langle f, g\rangle_\Psi \coloneqq \int_{-1}^{1} \frac{f(x)g(x)}{\sqrt{1-x^2}}\,\mathrm{d}\lambda(x)$$

das zugehörige Skalarprodukt auf L^2_Ψ, bzw. $\|\cdot\|_\Psi$ die zugehörige Norm. Analog definieren wir die *Sobolev-Räume* $W^{m,2}_\Psi$. Die nächsten beiden Theoreme geben Kriterien für Konvergenz an, welche von der Fourier-Reihe vererbt werden.

Theorem 3.3 ([Alt13, Satz 7.12])**.** *Sei $f \in W^{m,2}_\Psi$ und*

$$P_n f \coloneqq \sum_{k=0}^{n} a_k T_k, \quad a_k = \frac{\langle f, T_k\rangle_\Psi}{\langle T_k, T_k\rangle_\Psi}$$

Dann gilt

$$\|f - P_n f\|_\Psi = \mathcal{O}\left(n^{-m}\right)$$

Für $m = 0$, also $f \in L^2_\Psi$, gilt $P_n f \to f$ in L^2_Ψ.

Bemerkung. Es ist

$$\langle T_k, T_k\rangle = \int_0^\pi \cos^2(nx)\,\mathrm{d}x = \begin{cases} \pi, & \text{falls } k > 0 \\ \frac{\pi}{2}, & \text{falls } k = 0 \end{cases}$$

Siehe (3.1) für eine Erklärung, warum $a_k = 2a_0$ für $k \neq 0$.

Wie die Fourier-Reihe, konvergiert also auch also auch die *Čebyšëv-Reihe* in L^2 und der Fehler ist von der Regularität abhängig. Ist f stetig (differenziebar) lässt sich die Konvergenz punktweise zeigen:

Theorem 3.4. *Sei $f\colon [-1,1] \to \mathbb{R}$. Ist f stückweise stetig, so gilt für alle $x \in (-1,1)$*

$$\lim_{n\to\infty} P_n f(x) = \frac{\lim_{y\downarrow x} f(y) + \lim_{y\uparrow x} f(y)}{2} \tag{3.4}$$

Falls f stetig ist, gilt $P_n f \to f$ punktweise. Ist f stetig differenziebar, so ist die Konvergenz gleichmäßig. Falls f m-mal stetig differenziebar

ist, gilt

$$\|f - P_n f\|_\infty = \mathcal{O}\left(n^{-m}\right)$$

Bemerkung. Gleichung (3.4) besagt, dass die Čebyšëv-Reihe an Sprungstellen gegen den Mittelwert und an stetigen Punkten punktweise konvergiert.

Die Bedingung für gleichmäßige Konvergenz lässt sich deutlich abschwächen: Es genügt für stetige f, dass f beschränkte Variation hat oder die *Dini-Lipschitz-Bedingung* erfüllt [MH02, Abschnitt 5.3.2].

Wir werden weiter unten auf diese Konvergenzaussagen zurückkommen und ein paar Unterschiede zwischen Fourier- und Čebyšëv-Reihe illustrieren. Das nächste Theorem beantwortet die Frage, wie gut die Partialsumme $P_n f$ die Funktion f bzgl. einer Norm approximiert.

Theorem 3.5 ([MH02, Abschnitt 5.4])**.** *Sei $(V,\|\cdot\|)$, $V \subseteq L^2_\Psi$ ein normierter Funktionenraum mit $p_n \subseteq V$ und $P_n \colon V \to p_n$ wie oben. Ist P_n bzgl. $\|\cdot\|$ stetig, so gilt für $f \in V$:*

$$\|f - P_n f\| \leq (1 + \|P_n\|) \min_{p \in p_n} \|f - p\|$$
$$\|f - P_n f\| \leq \|\mathbb{1} - P_n\| \min_{p \in p_n} \|f - p\|$$

Dabei ist $\|P_n\|$ die induzierte Operator-Norm.

Bemerkung. Das Minimum existiert, da

$$\|\boldsymbol{a}\|_n \coloneqq \left\| \sum_{i=0}^{n} a_i x_i \right\|, \quad \boldsymbol{a} = (a_0, \ldots, a_n)$$

eine Norm auf $\mathbb{R}^{n+1}$ definiert und deswegen die Funktion

$$\phi \colon \mathbb{R}^{n+1} \to \mathbb{R}, \quad \boldsymbol{a} \mapsto \left\| f - \sum_{i=0}^{n} a_i x_i \right\|, \quad |\phi(\boldsymbol{a}) - \phi(\boldsymbol{b})| \leq \|\boldsymbol{a} - \boldsymbol{b}\|_n$$

stetig ist, d.h. sie nimmt bspw. auf der kompakten Menge

$$\left\{ \boldsymbol{a} \in \mathbb{R}^{n+1} \ : \ |\phi(\boldsymbol{a})| \leq \|f\| \right\}$$

ein (globales) Minimum an.

Es ist also garantiert, dass die Approximation in einer gewissen Nähe einer optimalen Approximation ist. Die Aussage folgt aus einer analogen Aussage für die Fourier-Basis. Dort lässt sich weiter zeigen, dass für

$$V = \{f \colon \mathbb{R} \to \mathbb{R} \mid f \text{ stetig und } 2\pi\text{-periodisch}\}$$

die entsprechende Partialsumme F_n in der Operatornorm, verglichen mit allen *linearen* Projektionen H_n in die trigonometrischen Polynome, ein Minimum darstellt:

$$\|F_n\|_\infty \leq \|H_n\|_\infty$$

Die Fourier-Reihe konvergiert also »am schnellsten« in dieser Klasse, während für Polynome die Resultate nicht direkt übertragbar sind, obwohl die Praxis-Erfahrungen dies vermuten lassen. Es sind zumindest einige Klassen bekannt, für welche die Čebyšëv-Polynome ebenfalls optimal sind [MH02, Abschnitt 5.6].

Die Čebyšëv-Polynome spielen auch in der Funktionsinterpolation eine wichtige Rolle. Die Eigenschaften können elegant und ähnlich zur hier beschrieben Orthogonalität dargestellt werden, worauf wir hier allerdings nicht weiter eingehen, da bei der Approximation der Wert-Funktion die Stützstellen nicht frei wählbar sind, sondern die Samples durch die besuchten Zustände entstehen. Die Interpolationseigenschaften und vieles Weitere findet sich in [MH02], sowie in Standard-Lehrbüchern der Numerik.

Verhalten am Rand und an Sprungstellen

In diesem Abschnitt werden wir uns exemplarisch an drei Beispielen das Verhalten der Partialsummen von Fourier- und Čebyšëv-Reihe anschauen. Es werden einige Phänomene sichtbar, die aus der Praxis für diese Basen bekannt sind.

Wir schauen uns Funktionen

$$f \colon [-1, 1] \to \mathbb{R}$$

an und berechnen die Čebyšëv-Koeffizienten

$$a_k \coloneqq \frac{\langle f, T_k \rangle}{\langle T_k, T_K \rangle}, \quad k = 0, \ldots, n$$

bzw. wird die Funktion für die Fourier-Reihe mit einer affinen Projektion von $[0, \pi]$ auf $[-1, 1]$ verkettet und gerade nach $[-\pi, \pi]$ fortgesetzt:

$$\tilde{f}(x) \coloneqq f\left(\frac{2}{\pi}|x| - 1\right)$$

Danach sind die Koeffizienten für die Kosinus-Reihe:

$$b_0 \coloneqq \frac{1}{\pi}\int_0^\pi \tilde{f}(x)\cos(0x)\,\mathrm{d}x = \frac{1}{\pi}\int_0^\pi \tilde{f}(x)\,\mathrm{d}x$$
$$b_k \coloneqq \frac{2}{\pi}\int_0^\pi \tilde{f}(x)\cos(kx)\,\mathrm{d}x, \quad k = 1, \ldots, n$$

Bemerkung. Diese Vorgehensweise werden wir auch für die Benutzung als Features verwenden, wie in [KOT11] beschrieben.

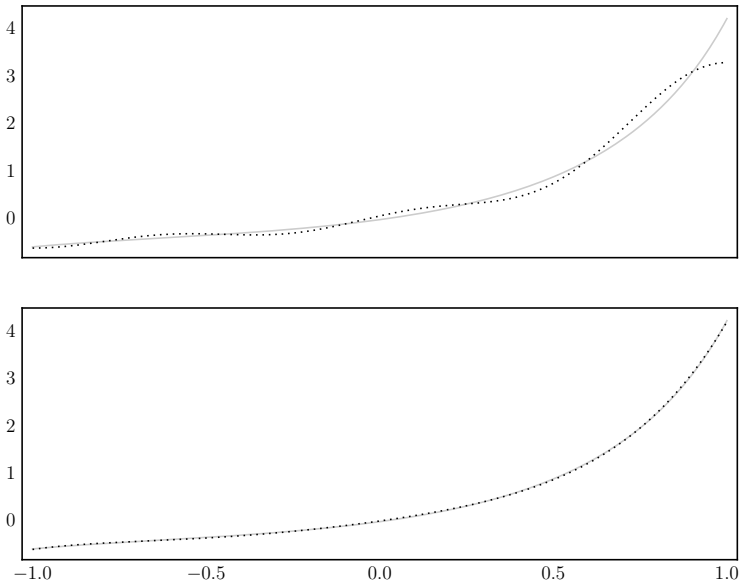

Abbildung 3.2.: L^2-Bestapproximation für die Fourier-Basis oben und die Čebyšëv-Basis unten für f.

Das erste Beispiel ist $f(x) \coloneqq \exp(x)\tan(x)$. Diese Funktion ist auf $[-1, 1]$ glatt, daher erwarten wir nach Theorem 3.4 sehr schnelle, gleichmäßige

k	0	1	2	3	4	5
a_k	0.0146	1.0386	0.7600	0.5534	1.0368	0.8038
b_k	0.4748	−1.3528	0.6381	−0.4144	0.2414	−0.1859

Konvergenz. Abbildung 3.2 zeigt die Approximation für $n = 5$. Die Čebyšëv-Approximation hat einen maximalen Fehler der Ordnung 10^{-2} während die Fourier-Basis die eher geraden Teile nicht gut darstellt und am Rand (rechts) einen größeren Fehler hat. Der Grund ist, dass in der Fourier-Darstellung dort tatsächlich kein *Rand* ist, sondern die Funktion periodisch fortgesetzt wird. Die nach unten gehende Kurve deutet dies an. Es liegt jedoch punktweise Konvergenz vor und wegen der beschränkten Variation ebenfalls gleichmäßige, wenn auch langsamer.

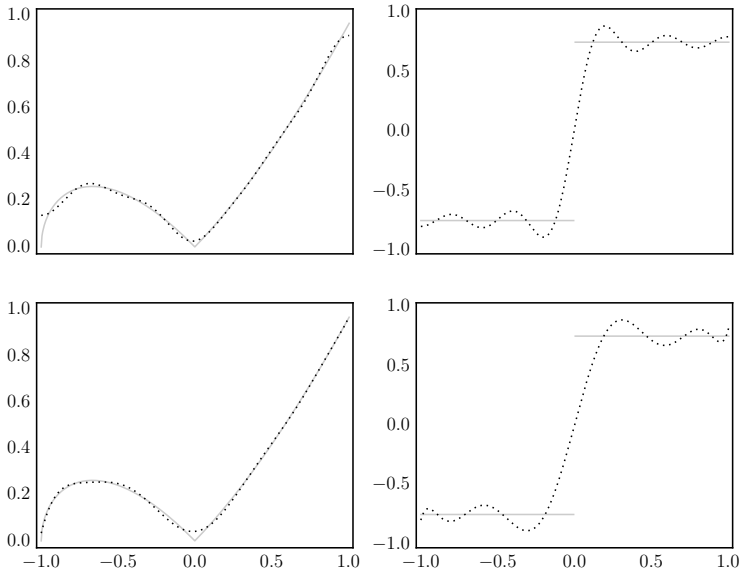

Abbildung 3.3.: L^2-Bestapproximation für die Fourier-Basis oben und die Čebyšëv-Basis unten für f_1 links und f_2 rechts.

Die anderen Beispiele sind

$$f_1(x) \coloneqq \sqrt{x^3 + x^2}, \quad f_2(x) \coloneqq \operatorname{sign} x$$

Dabei ist f_1 in 0 nicht differenzierbar und f_2 ist stückweise konstant mit einer Sprungstelle in 0. Abbildung 3.3 zeigt die Approximation für $n = 10$. Für f_1 zeigt sich an den Rändern das gleiche Bild wie bereits beim ersten Beispiel. In diesem Fall scheint allerdings die Fourier-Basis an der Stelle 0 besser zu sein, während die Čebyšëv-Polynome die anderen Stellen mit weniger Schwingungen darstellen. In beiden Fällen liegt gleichmäßige Konvergenz vor.

Im Fall von f_2 zeigt sich an der Sprungstelle zuerst, dass der Wert der beiden Approximationen dem Mittelwert aus (3.4) entspricht. In allen anderen Punkten ist die Konvergenz punktweise. Es wird außerdem ein bekanntes Problem sichtbar, dass als *Gibbssches Phänomen* bekannt ist: An der Sprungstelle schwingt die Fourier-Basis über. Betrag und Richtung bleiben dabei unabhängig von n gleich, nur die Breite wird kleiner. Die Konvergenz ist somit — sieht man von dem Fehler in 0 ab — nicht gleichmäßig sondern hat immer einen nach unten beschränkten, maximalen Fehler. Dieses Überschwingen überträgt sich auch auf die Čebyšëv-Darstellung.

Die Beispiele deuten an, dass im direkten Vergleich Čebyšëv-Polynome der Fourier-Basis vorzuziehen sind, falls die zu approximierende Funktion nicht von einer besonderen Periodizität gekennzeichnet ist. Sind die Funktionen nur stückweise stetig und haben Sprungstellen, bekommen beide Basen Probleme, die insbesondere für die Darstellung der Wert-Funktion die bereits beschriebenen, schlechten Auswirkungen auf ein *policy improvement* haben könnten: Zustände nahe der Sprungstelle haben einen großen Approximations-Fehler.

Bemerkung. Das Überschwingen ist bei einem Interpolationsansatz mit Hilfe der Čebyšëv-Polynome in diesem Fall geringer. In der Literatur sind außerdem Methoden bekannt, um das Phänomen zu dämpfen [Can+06, S. 56ff].

3.2. Approximation in höheren Dimensionen

In diesem Abschnitt werden wir die Čebyšëv- und Fourier-Basen in höhere Dimensionen übertragen und die Benutzung im Zusammenhang mit Zustandsräumen darstellen.

Da eine exakte Konvergenz-Analyse nicht Ziel dieses Kapitels ist, haben wir uns für die Angabe der Konvergenz-Eigenschaften auf den eindimen-

sionalen Fall zur Einfachheit beschränkt. Viele Eigenschaften des vorigen Absatzes übertragen sich jedoch bei geeigneter Anpassung der Voraussetzungen auf den mehrdimensionalen Fall. Für einen Einblick und Referenzen zu detailierteren Abhandlungen siehe [MH02, Abschnitt 5.3.3].

Wir wählen zur Einfachheit hier in jeder Dimension die gleiche Approximationsordnung n und erhalten für $\boldsymbol{x} \in [-1,1]^d$ die Basis-Darstellung

$$\boldsymbol{x} \sim \sum_{\boldsymbol{c} \leq n} a_{\boldsymbol{c}} T_{c_1}(x_1) T_{c_2}(x_2) \cdot \ldots \cdot T_{c_d}(x_d) \tag{3.5}$$

Wobei die Summe über alle $\boldsymbol{c} = (c_1, \ldots, c_d)^\top \subset \mathbb{N}^d$ mit $c_i \leq n$ für $i = 1, \ldots, d$ gemeint ist. Sei

$$\begin{aligned} T_{\boldsymbol{c}} \colon [-1,1]^d &\to \mathbb{R} \\ \boldsymbol{x} &\mapsto T_{c_1}(x_1) T_{c_2}(x_2) \cdot \ldots \cdot T_{c_d}(x_d) \end{aligned}$$

Wir nehmen an, dass

$$S \subseteq [a_1, b_1] \times [a_2, b_2] \times \ldots [a_d, b_d], \quad d \in \mathbb{N}$$

ist mit $a_i, b_i \in \mathbb{R}$ und $a_i < b_i$ für $i = 1, \ldots, d$. Durch die affine Abbildung

$$\begin{aligned} \boldsymbol{\psi} \colon S &\to [-1,1]^d \\ (s_1, \ldots, s_d) &\mapsto \left(2\frac{s_1 - b_1}{b_1 - a_1} - 1, \ldots, 2\frac{s_d - b_d}{b_d - a_d} - 1 \right) \end{aligned}$$

werden wir S auf den Würfel $[-1,1]^d$ abbilden. Wenn nun $\{\boldsymbol{\zeta}_i\}_{i=1}^N$ eine Aufzählung der Koeffizienten $\boldsymbol{c}$ aus (3.5) ist, so definieren wir die Feature-Funktion mit Ordnung n als:

$$\begin{aligned} \boldsymbol{F}_{\mathrm{c}} \colon S &\to \mathbb{R}^N \\ \boldsymbol{s} &\mapsto \left((T_{\boldsymbol{\zeta}_1} \circ \boldsymbol{\psi})(\boldsymbol{s}), \ldots, (T_{\boldsymbol{\zeta}_N} \circ \boldsymbol{\psi})(\boldsymbol{s}) \right) \end{aligned}$$

Also eine Aufreihung aller Basis-Funktionen aus (3.5) ausgewertet an der Stelle $\boldsymbol{s}$.

Für die Fourier-Reihe nehmen wir entsprechend an, dass $\boldsymbol{\psi}$ affin nach $[0, \pi]^d$ abbildet. Wir setzen die Funktion gerade und 2π-periodisch fort. Dort ist die Darstellung wegen $\exp(x)\exp(y) = \exp(x+y)$:

$$\begin{aligned} \boldsymbol{F}_{\mathrm{f}} \colon S &\to \mathbb{R}^N \\ \boldsymbol{s} &\mapsto \left(\cos\left({\boldsymbol{\zeta}_1}^\top \boldsymbol{\psi}(s) \right), \ldots, \cos\left({\boldsymbol{\zeta}_N}^\top \boldsymbol{\psi}(s) \right) \right) \end{aligned}$$

3.3. Ausgedünnte Basen: Hyperbolisches Kreuz

Ein Problem bei der Erweiterung auf d Dimensionen ist, dass bei Ordnung n die Anzahl an Features $(n+1)^d$ beträgt. Für große d ist die Berechnung also schnell nicht mehr handhabbar (*Curse of Dimensionality*). Einen Lösungsvorschlag den wir hier testen möchten, ist die Ausdünnung der Basis an Hand des *Hyperbolischen Kreuzes*: Statt die Basisfunktionen mit Indizes $c_i \leq n$ für $i = 1, \ldots, n$ zu wählen, beschränken wir uns auf die Indexmenge

$$H(\kappa) \coloneqq \left\{ \boldsymbol{c} \in \mathbb{N}^d \;\middle|\; (c_1 + 1) \cdot \ldots \cdot (c_d + 1) \leq \kappa + 1,\ \|\boldsymbol{c}\|_\infty \leq n \right\}$$

Bemerkung. Es ist üblich die Beschränkung $\|\boldsymbol{c}\|_\infty \leq n$ wegzulassen, allerdings war diese Verfeinerung in der gewählten Implementierung leicht umsetzbar. Mit $n = \kappa$ entspricht dies der üblichen Definition.

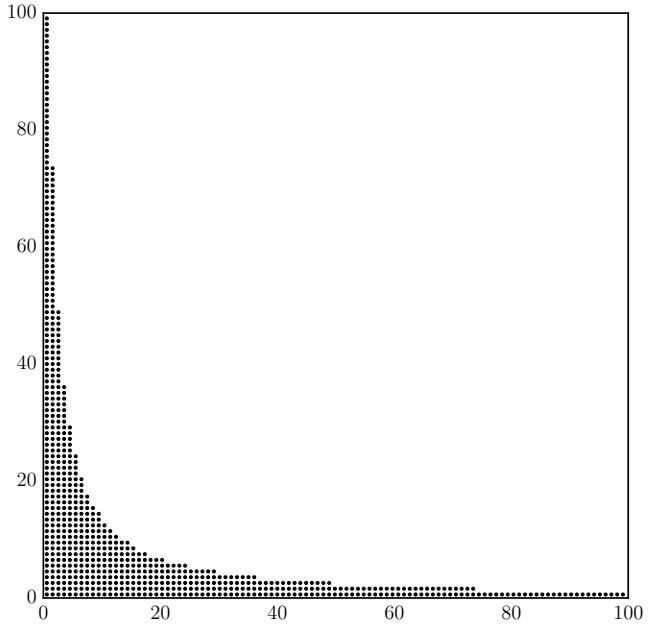

Abbildung 3.4.: Die Punkte in $H(150)$ für $n = 100$ und $d = 2$.

Während im eindimensionalen Fall klar ist, in welcher Reihenfolge bzgl. Konvergenz die Basis-Funktionen hinzugefügt werden (aufsteigender Ord-

nung), so ist die Wahl in höheren Dimensionen nicht mehr eindeutig. Es hat sich gezeigt, dass Basen, eingeschränkt auf das Hyperbolische Kreuz, häufig immer noch sehr gute Approximationseigenschaften haben, auch im Vergleich zum Tensor-Ansatz oben. Der Grund ist, dass die Koeffizienten unter gewissen Glattheitsbedingungen mit dem Produkt der Ordnungen bzw. Frequenzen abfallen [Hal92]. Wir erreichen also mit einer reduzierten Anzahl von

$$|H(\kappa)| = \mathcal{O}\left(\kappa 2^{-d} \left(\log\left(1+\kappa\right)\right)^{d-1}\right)$$

Basis-Funktionen eine gute Approximation.

Bemerkung. Der Faktor 2^{-d} kommt hinzu, da wir wegen der geraden Fortsetzung die Frequenzen/Ordnungen aus $\mathbb{N}^d$ statt $\mathbb{Z}^d$ wählen.

4. Numerische Experimente

Wir werden in diesem Kapitel die Čebyšëv-Polynome an Hand von drei Beispielen als Features im SARSA-Algorithmus testen, während die Fourier-Basis, wie in [KOT11] beschrieben, als Vergleich dienen soll. Wir übernehmen die Wahl der Beispiele, um an die Experimente dort mit anderen Basis-Funktionen anknüpfen zu können.

4.1. Das Labyrinth-Problem

In diesem Problem muss gelernt werden in einem Labyrinth zu einer festgelegten Stelle zu gelangen. Das Labyrinth wird durch Blöcke modelliert und ist endlich. Ein Block ist entweder eine Wand, ein Weg, das Ziel oder eine Grube, welche vermieden werden muss. Befindet sich das System an einer Stelle $(x, y) \in \mathbb{N}^2$ (dem Zustand), so sind je nach Situation der Wände die vier Aktionen »oben«, »unten«, »links« und »rechts« möglich und bewegen das System in der entsprechenden Richtung einen Block weiter. Beim Ziel-Block und den Gruben handelt es sich um terminale Zustände. Ziel der Aufgabe ist es, möglichst schnell zum Ziel zu gelangen. Die Belohnungs-Funktion wird gewählt als

$$\mathcal{R}(s, a, s') \coloneqq \begin{cases} -1 & \text{falls } s' \text{ ein Weg ist} \\ -\Delta & \text{falls } s' \text{ eine Grube ist} \\ 0 & \text{sonst} \end{cases}$$

wobei $\Delta \gg 0$. Mit der Wahl $\gamma \coloneqq 1$ erhalten wir so ein Stochastisches-Kürzeste-Wege-Problem, wobei die Zustandsdynamiken deterministisch sind.

Beispiel: Einfaches Labyrinth

Abbildung 4.1 zeigt das 8×4-Labyrinth, welches wir im ersten Schritt testen möchten. Der Zustandsraum ist endlich, wird allerdings für die Benutzung

mit Čebyšëv- bzw. Fourier-Features als kontinuierlich aufgefasst. Die Auswertung findet immer nur an diskreten Stellen statt.

Abbildung 4.1.: Das Test-»Labyrinth«

In Abbildung 4.2 sind die Ergebnisse der Čebyšëv-Basis: Es wurde getestet, wie viele Schritte der SARSA-Algorithmus ($\lambda = 0.9$) bei einer Folge von Episoden benötigt. Auf der x-Achse ist die Nummer der Episode und auf der y-Achse die benötigen Schritte in dieser Episode. Diese Werte wurden über 100 Wiederholungen des Experiments gemittelt und mit Standardabweichung als Fehler-Balken angegeben.

Nach bereits drei Episoden ist in beiden Fällen ein gutes Ergebnis erreicht und der Algorithmus hat nach fünf Episoden einen optimalen Weg (17 Schritte) gefunden und eine Standardabweichung von 0 in allen folgenden Episoden. Für $n = 4$ ist die Schrittzahl etwas instabiler am Anfang, insbesondere sind mehr Schritte in den ersten Episoden nötig. Es ist zu beachten, dass die Anzahl an Features sich von 10 ($n = 4$) auf 14 ($n = 5$) vergrößert hat und $|S| = 32$ ist.

Die Fourier-Basis erreichte in diesem Experiment für $n < 6$ nicht das Ziel (Abbruch nach 1000 Schritten in jeder Episode), auch die Ausdünnung der Basis war für kleinere n nicht erfolgreich. In Abbildung 4.3 sind die Ergebnisse der Fourier-Basis für $n = 6$ (46 Features) und $n = 10$ auf $H(10)$ (29 Features) zu sehen. Die Ergebnisse sind vergleichbar mit den Čebyšëv-Polynomen darunter. Beide Verfahren konvergieren nach maximal fünf Episoden gegen eine optimale Strategie, wobei die ausgedünnten Basen trotz niedrigerer Featureanzahl besser sind. Die Fourier-Basis zeigt bei $n = 6$ leichte Instabilitäten.

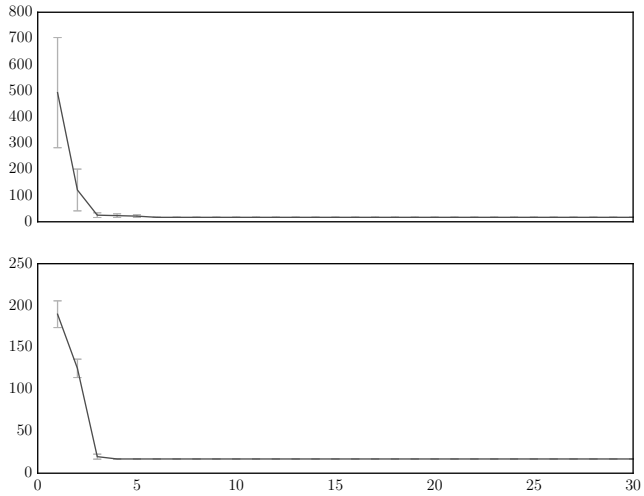

Abbildung 4.2.: Čebyšëv-Basis mit $n = 4$ auf $H(4)$ oben, bzw. $n = 5$ auf $H(5)$ unten.

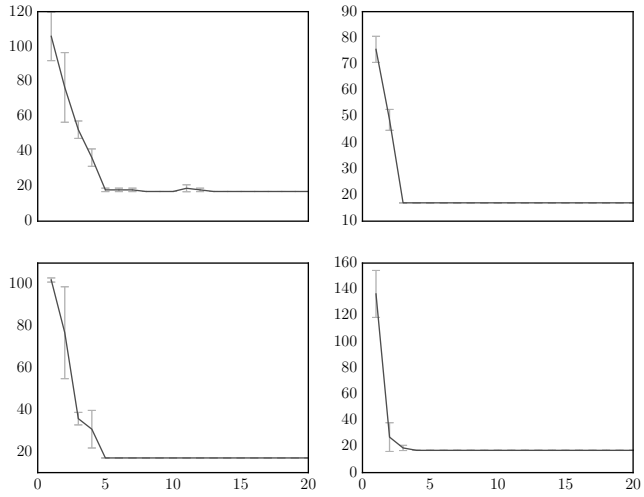

Abbildung 4.3.: Fourier-Basis oben mit $n = 6$ links und $n = 10$ rechts auf $H(10)$. Čebyšëv-Basis unten mit gleicher Konfiguration.

Beispiel: Labyrinth mit Grube

Das Labyrinth aus Abbildung 4.4 stellt ein Negativbeispiel dar.

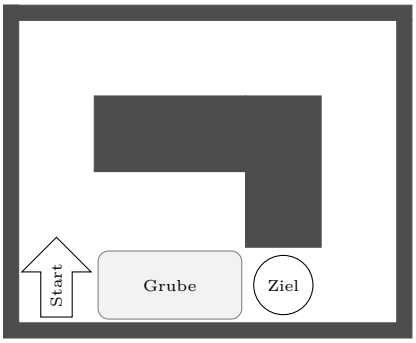

Abbildung 4.4.: Labyrinth mit Grube

Der SARSA-Algorithmus hat mit Čebyšëv- und Fourier-Features für größere Δ kein brauchbares Verhalten gezeigt: Entweder wurde das Ziel nicht erreicht (Abbruch nach 1000 Schritten), oder die Episode endete in der Grube. In einigen wenigen Fällen wurde der optimale Weg gefunden. Eine plausible Erklärung für diese Beobachtungen liefert das Verhalten der Basen an Sprungstellen. Bei diesem diskreten Zustandsraum ist Stetigkeit zwar nicht definiert, allerdings ist der Erwartungswert für die Belohnung an Ziel und Grube stark unterschiedlich. Die nicht-lokalen Auswirkungen von Updates auf $\boldsymbol{\theta}$ beeinflussen alle Werte in der Nähe der Grube, so auch das Ziel. Liegt der Auswertungspunkt (S wird kontinuierlich eingebettet) nun ungünstig (Gibbsches Phänomen), wird der Schritt ins Ziel nicht als günstig angesehen, bzw. scheint ein möglichst frühes Enden in der Grube als optimal.

Das dies tatsächlich ein Problem der gewählten Approximation an die Wert-Funktion ist, zeigt der Vergleich mit der exakten Abspeicherung, also ohne Features: In diesem Fall erreicht der Algorithmus mit Hilfe einer ε-Strategie nach weniger als 100 Schritten eine optimale Strategie. Die großen Differenzen verursachen den gewählten Features Probleme[1].

[1] Eine mögliche Idee um dies zu verbessern, ist die Auswertung und Mittelung an mehreren Punkten, welche alle zum selben Block gehören. So könnten die Überschläge evtl. herausgemittelt werden.

4.2. Das Mountain-Car-Problem

Wir wenden uns dem bekannten Mountain-Car-Problem zu: Ein Auto steht in einem Tal (eindimensional) und soll es verlassen, allerdings ist nicht genug Kraft vorhanden, um die Steigung direkt zu bewältigen. Der Algorithmus soll einen Weg finden, die nötige Geschwindigkeit zu erreichen. Die Lösung besteht aus mehrmaligem Vor- und Zurückfahren, was selbstständig erlernt werden wird. Abbildung 4.5 ist ein Plot der verwendeten Funktion für das »Tal«.

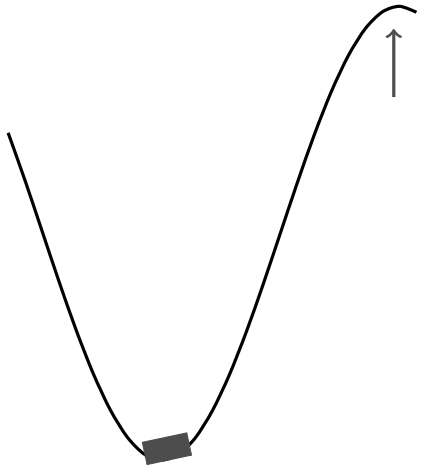

Abbildung 4.5.: Das Rechteck (Auto) möchte trotz schwachem Motor das Tal verlassen bis zur Stelle, die mit dem Pfeil markiert ist.

Der Zustandsraum ist $S = [-1.2, 0.6] \times [-0.07, 0.07]$, wobei die erste Koordinate die Position und die zweite die Geschwindigkeit darstellt. Ein terminaler Zustand ist erreicht, falls die Position größer als 0.5 ist. In jedem Zeitschritt bestehen nun drei Aktionsmöglichkeiten: Beschleunigung nach vorne, nach hinten oder keine Beschleunigung. Die Systemdynamiken sind in [SB98, S. 214] beschrieben, dabei wird Geschwindigkeit und Position entsprechend S beschränkt. Der Startzustand ist $(-0.5, 0)$ und wir benutzen $\gamma = 1$. Als Belohnungswert wählen wir

$$\mathcal{R}(s, a, s') := \begin{cases} 0 & \text{falls } s' \text{ terminal ist} \\ -1 & \text{sonst} \end{cases}$$

und erhalten somit ebenfalls ein Stochastisches-Kürzeste-Wege-Problem mit deterministischen Zustandsdynamiken. Allerdings ist nun der Zustandsraum nicht mehr endlich.

In Abbildung 4.6 sind die Schrittzahlen pro Episode für Fourier- und Čebyšëv-Basis mit $n \in \{5, 16\}$ zu sehen. Die Feature-Anzahl beträgt 36 bzw. 289. Auf eine Mittelung wurde verzichtet, da die jeweiligen Experimente kein einheitliches Bild zeigten. Das Streudiagramm soll einen Eindruck von den Schranken der erreichten Schrittzahlen vermitteln. In den ersten Episoden fällt die benötigte Schrittzahl bis zum Erreichen des Ziels in der Regel auf die Größenordnung 200, was das Ergebnis aus [KOT11] bestätigt. In den Episoden 20 bis 100 bleiben die Schrittzahlen mit der Fourier-Basis zwar meistens um oder unter 200, allerdings gibt es einige schlechte Durchläufe/Ausreißer und zum anderen wird keine Stabilität erreicht. Der Plot für die Čebyšëv-Polynome zeigt hier eine höhere Stabilität und bessere Ergebnisse: Die Schrittzahlen bleiben ab etwa Episode 30 relativ konstant — wenn auch pro Durchlauf auf unterschiedlichen Werten — und es werden Strategien berechnet, die schneller sind (bis zu einer Lösung mit 104 Schritten[2]).

Die Ergebnisse für $n = 80$ auf dem Hyperbolischen Kreuz mit 373 Features sind in Abbildung 4.7 zu sehen. Die Čebyšëv-Basis erreicht ebenfalls bessere Schrittzahlen und fällt dabei schneller ab. Zudem ist die Bandbreite der erreichten Schrittzahlen geringer. Im Gegensatz dazu streut die Fourier-Basis stärker und fällt außerdem kaum unter 200.

Der einzige Teil des Experiments, der vom Zufall abhängt, ist im *policy improvement*. Wir benutzen zwar keine ε-Strategie, allerdings wird bei der Wahl einer maximierenden Aktion

$$\arg\max_{a \in A_s} Q(s, a)$$

unter allen maximalen eine zufällige gezogen. Dieser Schritt ist Auslöser für die stark unterschiedlichen Verläufe und zeigt, dass die Verfahren nicht stabil bzgl. dieser Wahlen sind. Es spielt für das erreichte Endergebnis eine Rolle, welche Wahlen getroffen werden. Wenn Konvergenz zu einer stabilen Lösung vorhanden ist, dann nicht innerhalb der ersten 100 Episoden.

[2] Die besten dem Autor bekannten Ergebnisse aus der Literatur liegen bei 103.

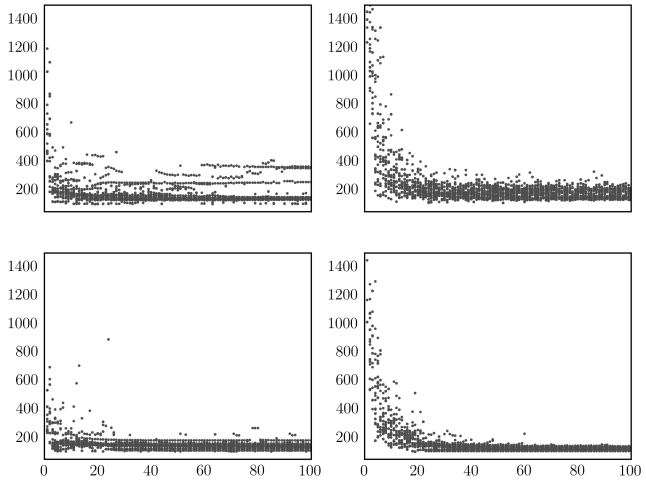

Abbildung 4.6.: Fourier-Basis oben mit $n = 5$ links und $n = 16$ rechts. Čebyšëv-Basis unten mit gleicher Konfiguration.

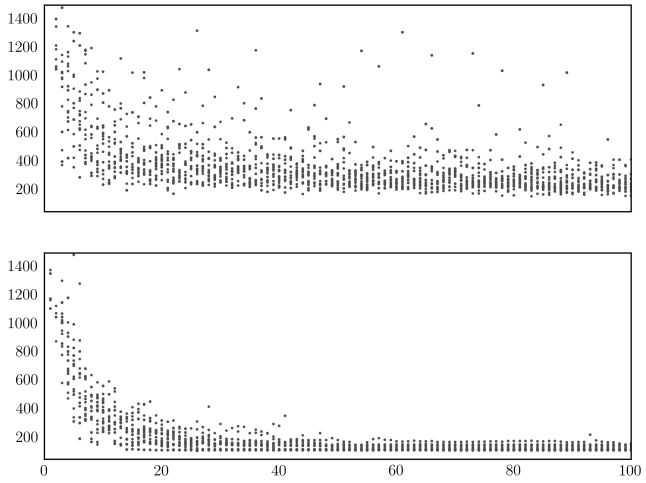

Abbildung 4.7.: Fourier-Basis oben mit $n = 80$ auf $H(80)$. Čebyšëv-Basis unten mit gleicher Konfiguration.

4.3. Der Acrobot

Das letzte betrachtete Problem ist eine vereinfachte Modellierung einer Aufgabe, die auch vielen Menschen Schwierigkeiten verursacht: Der Aufschwung an einem Reck. Die Modellierung besteht aus zwei festen Stangen, die durch ein Gelenk miteinander und mit einem weiteren Gelenk an einem festen Punkt verbunden sind, siehe Abbildung 4.8. Das Ziel besteht darin, in möglichst kurzer Zeit mit der Spitze der Stange über die Horizontale hinaus zu gelangen.

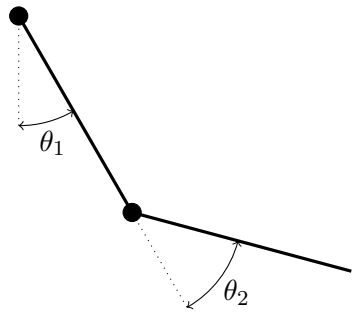

Abbildung 4.8.: Der Acrobot muss mit dem Ende über die gestrichelte Horizontale hinaus.

Der Zustandsraum ist $S = [-\pi, \pi]^2 \times [-4\pi, 4\pi] \times [-9\pi, 9\pi]$ und beschreibt die beiden Winkel und deren Winkelgeschwindigkeiten. Die terminalen Zustände sind die mit

$$-\cos(\theta_1) - \cos(\theta_1 + \theta_2) > 1$$

Die Belohnungsfunktion wählen wir wie beim Mountain-Car, auch ist $\gamma = 1$. Im System stehen drei Aktionen zur Wahl, die angeben in welche Richtung bzw. oder ob ein Drehmoment am mittleren Gelenk angesetzt werden soll. Die

Systemdynamiken werden mit Hilfe einer gewöhnlichen Differentialgleichung berechnet, wobei das klassische Runge-Kutta-Verfahren vierter Ordnung verwendet wurde. Eine Beschreibung der Differentialgleichungen ist bspw. in [SB98, S. 271]. Wir starten mit $(0, 0, 0, 0)^\top$, also in Ruhe hängend.

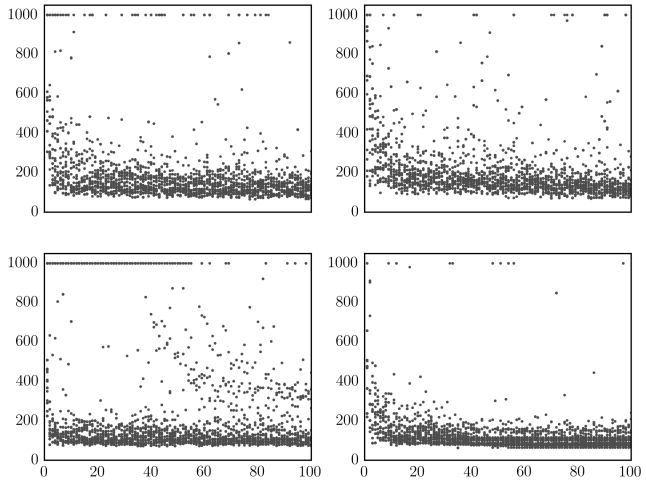

Abbildung 4.9.: Fourier-Basis oben mit $n = 5$ links und $n = 7$ rechts. Čebyšëv-Basis unten mit gleicher Konfiguration.

Zunächst die Ergebnisse für $n \in \{5, 7\}$ mit Featurezahl 1296 bzw. 4096. In Abbildung 4.9 setzt sich das Verhalten aus dem Mountain-Car-Experiment fort, jedoch mit einigen Besonderheiten. Für $n = 5$ tauchen in beiden Basen Episoden auf, die zwischenzeitlich nicht das das Ziel erreichen, was an den Punkten auf der Höhe 1000 zu erkennen ist, dort wurden Episoden abgebrochen. Die Fourier-Basis hat hier bessere Streueigenschaften. Dieses Problem wird mit $n = 7$ besser und es zeigen sich wieder die bereits beschriebenen Streueigenschaften. Dabei werden in beiden Basen Strategien erreicht, die nur ca. 65 Schritte benötigen, mit einem Minimum bei der Čebyšëv-Basis von 62 Schritten[3].

In Abbildung 4.10 wurde für $n = 50$ auf $H(50)$ mit 1264 Features getestet. Die Čebyšëv-Polynome zeigen wie schon beim Mountain-Car-Problem eine geringe Streuung, ein gutes Ergebnis (Miniumum bei 63) und einen schnellen

[3]Die besten dem Autor bekannten Ergebnisse aus der Literatur liegen bei 61.

Abfall der Schrittanzahlen. Anders als vorher hat auch die Fourier-Basis hier gute Ergebnisse, wenn auch die Ausreißer etwas häufiger sind.

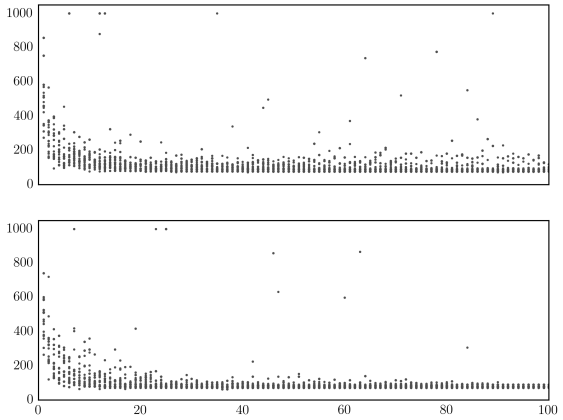

Abbildung 4.10.: Fourier-Basis oben mit $n = 50$ auf $H(50)$. Čebyšëv-Basis unten mit gleicher Konfiguration.

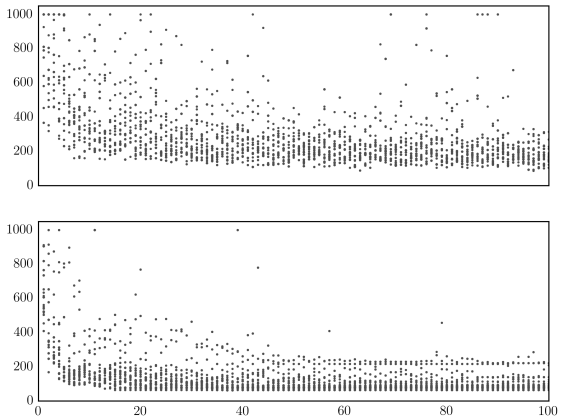

Abbildung 4.11.: Fourier-Basis oben mit $n = 9$ (10000 Features) und Čebyšëv-Basis unten mit gleicher Konfiguration.

Fazit und Diskussion der Experimente

In den getesteten Beispielen hat die Čebyšëv-Basis sich als mindestens vergleichbar gute und in der Regel sogar als bessere Basis gegenüber der Fourier-Basis gezeigt. Zusammen mit dem Hyperbolischen Kreuz wurden in praktikabler Rechenzeit brauchbare Schrittzahlen erreicht, die im Vergleich zur Fourier-Basis weniger bis gar keinen Schwankungen unterlagen.

Eine Information, die die gewählten Streudiagramme nicht abbildet, ist wie oft und in welchem Zusammenhang Ausreißer auftreten. Bei aufeinanderfolgenden Punkten gleicher Höhe ist nicht ablesbar, ob diese zu einem Durchgang gehören und wie oft diese Schrittzahl in der entsprechenden Episode erreicht wurde[4]. Dennoch liefern die Streudiagramme ein deutlicheres Indiz für die Kapazitäten und Probleme der Basis, als gemittelte Werte: Da die Schrittzahl pro Episode sowie deren Verlauf stark schwanken, würden so Schrittzahlen gemittelt, die nach verschiedenen Anzahlen von Lernschritten und Wahlen von maximierenden Aktionen entstanden sind.

Das gewählte Testverfahren zielte wie auch in [KOT11] auf Einfachheit ab. Es sind zahlreiche weitere Untersuchungen in jedem einzelnen Experiment denkbar und auch in sofern interessant, um Stärken und Schwächen noch besser zu verstehen. Das Gruben-Beispiel des Labyrinths zeigt, dass die Basen keineswegs universell einsetzbar sind und bereits an einfachen Problemen scheitern können. Allerdings waren die Ergebnisse deutlich besser, wenn Δ so gewählt wurde, dass die Belohnung für die Grube nur knapp schlechter war, als der Weg zum Ziel. Die Probleme der Basen bei große Gradienten sollten möglichst einfach demonstriert werden. Das Labyrinth bietet eine einfache Möglichkeit, um Wert-Funktionen mit einer bestimmten Geometrie zu erzeugen.

Eine weitere Fragestellung, die nicht beantwortet wurde, ist, ob und wie die Eigenschaften in Problemen mit höherer Dimension übertragbar sind. Die Versuche mit ähnlich einfachen Ansätzen (Ordnung und Hyperbolisches Kreuz variieren) wie hier, hatten alle das Problem, dass nicht eine Episode in einem terminalen Zustand beendet wurde, sondern immer das Episodenlimit — welches sehr weit von einer optimalen Strategie entfernt ist — erreicht wurde. Interessant wäre, was die Gründe dafür sind und ob

[4] Aus den Einzelverläufen, die hier nicht abgebildet sind, zeigt sich, dass die Čebyšëv-Basis im Gegensatz zur Fourier-Basis häufig eine gleichbleibende Schrittzahl erreicht.

eine einfache Korrektur möglich ist.

Wie bereits in [KOT11] beschrieben, scheint die Fourier- bzw. Čebyšëv-Basis als Wahl für allgemeine Tests nützlich zu sein, da sie im Vergleich zu anderen Basen einfach definiert und berechnet werden kann, während die erzielten Resultate gut oder sogar besser sind, inbesondere mit der einfachen Basis-Ausdünnung durch das Hyperbolische Kreuz (ein Parameter).

A. Implementierung

Die Experimente wurden aufbauend auf der Python-Software-Bibliothek *RLPy* [Ger+13] implementiert. Diese Bibliothek stellt verschiedene Test-Probleme, Strategie-Wahlen, Schrittweitensteuerungen, Basis-Funktionen usw. in einer modularen Art und Weise zur Verfügung. Die Zufallsgeneratoren werden so initialisiert, dass eine exakte Reproduktion der Ergebnisse möglich ist.

Änderungen an RLPy Die verwendete Version unterscheidet sich von der Release-Version zum einen durch eine fehlerbehobene Schrittweitensteuerung nach Dabney und zum anderen, neben einigen kleineren Korrekturen, durch eine Implementierung der Čebyšëv-Basis-Darstellung und der Auswertung auf dem Hyperbolischen Kreuz. Die (eindimensionalen) Čebyšëv-Polynome werden von NumPy [J+01] an den entsprechenden Stellen berechnet und für die Tensor-Variante mit Hilfe des äußeren Produktes schnell berechnet. Außerdem wurde die in [KOT11] beschriebene Vor-Skalierungsmöglichkeit umgesetzt.

Das Hyperbolische Kreuz hat sich als Python-Modul als zu langsam für die Experimente herausgestellt und wurde daher in C implementiert (zu finden in `rlpy/rlpy/Tools/crosscalc.c`). Bei der Initialisierung der Klasse werden die Koeffizienten aus dem Kreuz in einer hierarchischen Liste gespeichert, so dass durch eine Wiederbenutzung bereits berechneter Teilfaktoren in der Auswertung Zeit gespart werden kann (Teilfaktoren tauchen in mehreren Basis-Funktionen auf).

Alle Daten aus den Experimenten, die Programme die zur Durchführung ebendieser benutzt wurden, die Python-Skripte, die zum Plotten verwendet wurden, sowie die LaTeX-Dateien (mit Abbildungen etc.) befinden sich auf der beigelegten *Compact Disk*.

Clenshaw-Algorithmus für Čebyšëv-Partialsummen

In diesem Abschnitt beschreiben wir die numerische Implementierung und Stabilität der Berechnung von

$$f(x) \coloneqq \sum_{k=0}^{n} a_k T_k(x), \quad x \in [-1,1]$$

Die Berechnung erfolgt mit Hilfe des *Clenshaw-Algorithmus* und der Rekursionsbeziehung (3.3).

Algorithmus A.1 Clenshaw-Algorithmus für Čebyšëv-Polynome

Input: $(a_0, a_1, \ldots, a_n),\ n > 0$ und $x \in [-1,1]$
Output: $\sum_{k=0}^{n} a_k T_k(x)$

1: $b_{n+2} \leftarrow 0,\ b_{n+1} \leftarrow 0$
2: **for all** $k = n, n-1, \ldots, 1, 0$ **do**
3: $\quad b_k \leftarrow 2x b_{k+1} - b_{k+2} + a_k$
4: **end for**
5: **return** $b_0 - b_1 x$

Theorem A.1. *Algorithmus A.1 ist korrekt und benötigt* $3(n+2)$ *elementare Rechenoperationen.*

Beweis. Es gilt $a_k = b_k - 2x b_{k+1} + b_{k+2}$ für $k = 0, 1, \ldots, n$. Dann ist

$$\begin{aligned}\sum_{k=0}^{n} a_k T_k(x) &= \sum_{k=0}^{n} \left(b_k - 2x b_{k+1} + b_{k+2}\right) T_k(x) \\ &= \sum_{k=0}^{n} b_k T_k(x) - \sum_{k=1}^{n+1} b_k 2x\, T_{k-1}(x) + \sum_{k=2}^{n+2} b_k T_{k-2}(x)\end{aligned}$$

Siehe (3.3)

$$\begin{aligned} &= (b_0 + b_1 x) - b_1 2x + \sum_{k=2}^{n+2} b_k \underbrace{\left(T_k(x) - 2xT_{k-1}(x) + T_{k-2}(x)\right)}_{=0} \\ &= b_0 - b_1 x \end{aligned}$$

In jedem Schritt der Schleife sind drei Rechenoperationen, plus zwei am Ende. Mit dem Speichern von $2x$ folgt die Behauptung. □

Wir schließen den Abschnitt mit einer Abschätzung des Rechenfehlers.

Theorem A.2 ([MH02, Abschnitt 2.4.2])**.** *Seien $\bar{b}_k$ die berechneten Werte von b_k und*

$$\bar{b}_k = 2x\bar{b}_{k+1} - \bar{b}_{k+2} + a_k - \varepsilon_k$$

der Fehler in Zeile 3. Sei S_n die exakte Lösung und $\bar{S}_n = \bar{b}_0 - x\bar{b}_1$ die gestörte. Dann gilt

$$\left|S_n - \bar{S}_n\right| \leq \|\boldsymbol{\varepsilon}\|_1, \quad \boldsymbol{\varepsilon} = (\varepsilon_0, \ldots, \varepsilon_k)^\top$$

Beweis. Sei $\delta_k \coloneqq b_k - \bar{b}_k$. Dann gilt

$$\delta_k = 2x\left(b_{k+1} - \bar{b}_{k+1}\right) - \left(b_{k+2} - \bar{b}_{k+2}\right) + \varepsilon_k = 2x\delta_{k+1} - \delta_{k+2} + \varepsilon_k$$

Also erfüllt δ_k die gleiche Rekursionsgleichung wie b_k, jedoch mit ε_k an Stelle von a_k. Da wir annehmen, dass $\delta_{k+1} = \delta_{k+2} = 0$ ist, gilt somit

$$\delta_0 - x\delta_1 = \sum_{k=0}^{n} \varepsilon_k T_k(x)$$

und wir erhalten

$$\begin{aligned} \left|S_n - \bar{S}_n\right| &= \left|(b_0 - xb_1) - \left(\bar{b}_0 - x\bar{b}_1\right)\right| = |\delta_0 - x\delta_1| = \left|\sum_{k=0}^{n} \varepsilon_k T_k(x)\right| \\ &\leq \sum_{k=0}^{n} |\varepsilon_k||T_k(x)| \leq \max_{i=1,\ldots,n} |T_i(x)| \sum_{k=0}^{n} |\varepsilon_k| \leq \sum_{k=0}^{n} |\varepsilon_k| = \|\boldsymbol{\varepsilon}\|_1 \end{aligned}$$

Wegen $|T_i(x)| \leq 1$

□

Literatur

[Alt13] H.W. Alt. *Lineare Funktionalanalysis: Eine anwendungsorientierte Einführung*. Springer Berlin Heidelberg, 2013. ISBN: 9783662083857.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957. ISBN: 069107951X.

[Ber95a] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control. Vol. 1.* Athena Scientific, 1995. ISBN: 1886529124.

[Ber95b] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control. Vol. 2.* Athena Scientific, 1995. ISBN: 1886529132.

[BS96] D.P. Bertsekas und S.E. Shreve. *Stochastic Optimal Control: The Discrete Time Case*. Athena scientific optimization and computation series. Athena Scientific, 1996. ISBN: 9781886529038.

[BT96] D.P. Bertsekas und J.N. Tsitsiklis. *Neuro-dynamic Programming*. Anthropological Field Studies. Athena Scientific, 1996.

[Can+06] Claudio Canuto u. a. *Spectral Methods. Fundamentals in Single Domains*. Springer-Verlag Berlin Heidelberg, 2006. ISBN: 9783540307266.

[DB12] William Dabney und Andrew G Barto. »Adaptive Step-Size for Online Temporal Difference Learning.« In: *AAAI*. 2012.

[DNP14] Christoph Dann, Gerhard Neumann und Jan Peters. »Policy evaluation with temporal differences: A survey and comparison«. In: *The Journal of Machine Learning Research* 15.1 (2014), S. 809–883.

[Ger+13] Alborz Geramifard u. a. *RLPy: The Reinforcement Learning Library for Education and Research*. Apr. 2013.

[Hal92] Klaus Hallatschek. »Fouriertransformation auf dünnen Gittern mit hierarchischen Basen«. In: *Numerische Mathematik* 63.1 (1992), S. 83–97.

[J+01] Eric Jones, Travis Oliphant, Pearu Peterson u. a. *SciPy: Open source scientific tools for Python*. 2001–. URL: `http://www.scipy.org/`.

[Kle13] A. Klenke. *Wahrscheinlichkeitstheorie*. Springer-Lehrbuch Masterclass. Springer Berlin Heidelberg, 2013. ISBN: 9783642360183.

[KOT11] George Konidaris, Sarah Osentoski und Philip Thomas. »Value function approximation in reinforcement learning using the Fourier basis«. In: *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*. 2011, S. 1468–1473.

[KY03] H.J. Kushner und G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Applications of Mathematics. Springer, 2003. ISBN: 987654321.

[MH02] J.C. Mason und D.C. Handscomb. *Chebyshev Polynomials*. CRC Press, 2002. ISBN: 9781420036114.

[MM07] S. Mahadevan und M. Maggioni. »Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes«. In: *The Journal of Machine Learning Research* 8 (2007), S. 2169–2231.

[Mni+15] Volodymyr Mnih u. a. »Human-level control through deep reinforcement learning«. In: *Nature* 518.7540 (Feb. 2015), S. 529–533. ISSN: 0028-0836. URL: `http://dx.doi.org/10.1038/nature14236%2010.1038/nature14236%20http://www.nature.com/nature/journal/v518/n7540/abs/nature14236.html%5C#supplementary-information`.

[Put94] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994. ISBN: 0471727822.

[RM51] Herbert Robbins und Sutton Monro. »A Stochastic Approximation Method«. English. In: *The Annals of Mathematical Statistics* 22.3 (1951), S. 400–407. ISSN: 00034851.

[SB81] Richard S Sutton und Andrew G Barto. »Toward a modern theory of adaptive networks: expectation and prediction.« In: *Psychological review* 88.2 (1981), S. 135.

[SB98] Richard S Sutton und Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[SS14] Harm V Seijen und Rich Sutton. »True Online TD (lambda)«. In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 2014, S. 692–700.

[SS96] Satinder P Singh und Richard S Sutton. »Reinforcement learning with replacing eligibility traces«. In: *Machine learning* 22.1-3 (1996), S. 123–158.

[Tes95] Gerald Tesauro. »Temporal Difference Learning and TD-Gammon«. In: *Commun. ACM* 38.3 (März 1995), S. 58–68. ISSN: 0001-0782. DOI: 10.1145/203330.203343. URL: http://doi.acm.org/10.1145/203330.203343.

[Tok13] Michel Tokic. »Reinforcement Learning mit adaptiver Steuerung von Exploration und Exploitation«. Diss. Universität Ulm, Institut für Neuroinformatik, 2013. URL: http://vts.uni-ulm.de/doc.asp?id=8696.

[TV97] John N Tsitsiklis und Benjamin Van Roy. »An analysis of temporal-difference learning with function approximation«. In: *Automatic Control, IEEE Transactions on* 42.5 (1997), S. 674–690.

[Van98] Benjamin Van Roy. »Learning and Value Function Approximation in Complex Decision Processes«. Diss. Massachusetts Institute of Technology, Mai 1998. URL: http://www.stanford.edu/~bvr/pubs/thesis.pdf.