Reinforcement Lernen mit Regularisierungsnetzwerken

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

im

Fachbereich Physik, Mathematik und Informatik

der

Johannes Gutenberg-Universität Mainz

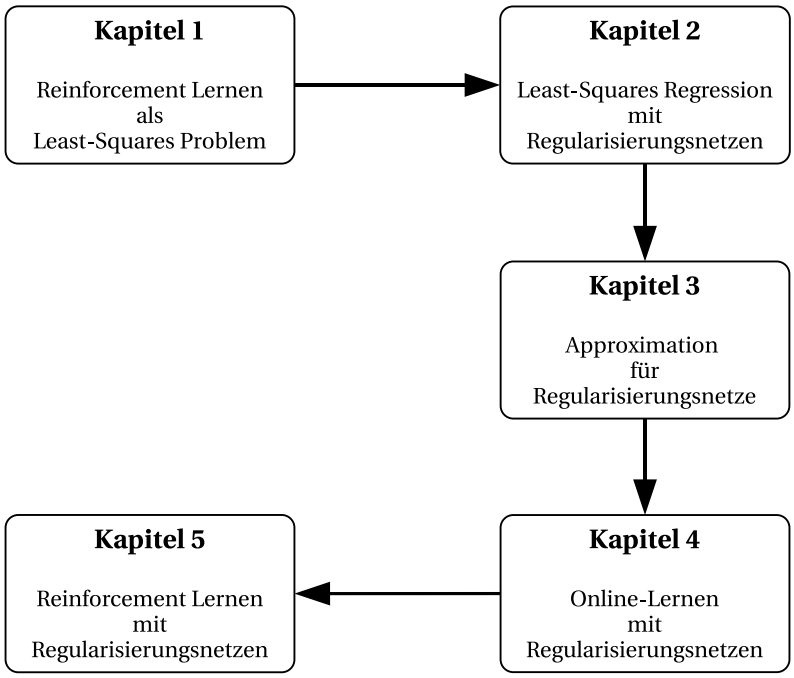
vorgelegt von

Tobias Jung

aus

Wiesbaden

November 2007



Datum der mündlichen Prüfung: 16. November 2007

Gutachter:

Prof. Dr. Elmar Schömer (Universität Mainz)
Dr. Daniel Polani (University of Hertfordshire, U.K.)

D77 – Mainzer Dissertation

Verwendete Abkürzungen

RL	Reinforcement Lernen
RN	Regularisierungsnetze
RR	Ridge Regression, *Seite 34*
Kernel-RR	Kernel Ridge Regression, *Seite 38*
GPR	Gauß-Prozeß Regression, *Seite 42*
OLS/OMP	Orthogonales Least-Squares/Matching Pursuit, *Seite 68*
PCA	Hauptkomponenten-Analyse (principal components analysis), *Seite 33*
SVD	Singulärwertzerlegung (singular value decomposition), *Seite 32*
ICD	unvollständige Cholesky-Zerlegung (incomplete cholesky decomposition), *Seite 48*
SR	Subset of Regressors Approximation, *Seite 46*
GCV	Generalisierte Kreuz-Validierung (generalized cross validation), *Seite 36*
TD	Temporal-Difference Lernen, *Seite 12*
Sarsa(λ)	auf OPI und Online-Lernen basierendes Standardverfahren in RL, *Seite 15*
API	Approximative Politik-Iteration, *Seite 11*
APE	Approximative Politik-Evaluation, *Seite 16*
OPI	Optimistische Politik-Iteration, *Seite 13*
LSTD	Least-Squares Temporal-Difference Lernen, *Seite 22*
LSPE	Least-Squares Policy Evaluation, *Seite 78*
BRM	Minimierung des Bellman-Residuums, *Seite 18*
CMAC	Cerebellar model articulation controller (auch Tilecoding), *Seite 9*

Zusammenfassung

Die Arbeit behandelt das Problem der Skalierbarkeit von Reinforcement Lernen auf hochdimensionale und komplexe Aufgabenstellungen. Unter Reinforcement Lernen versteht man dabei eine auf approximativem Dynamischen Programmieren basierende Klasse von Lernverfahren, die speziell Anwendung in der Künstlichen Intelligenz findet und zur autonomen Steuerung simulierter Agenten oder realer Hardwareroboter in dynamischen und unwägbaren Umwelten genutzt werden kann. Dazu wird mittels Regression aus Stichproben eine Funktion bestimmt, die die Lösung einer 'Optimalitätsgleichung' (Bellman) ist und aus der sich näherungsweise optimale Entscheidungen ableiten lassen. Eine große Hürde stellt dabei die Dimensionalität des Zustandsraums dar, die häufig hoch und daher traditionellen gitterbasierten Approximationsverfahren wenig zugänglich ist. Das Ziel dieser Arbeit ist es, Reinforcement Lernen durch nichtparametrisierte Funktionsapproximation (genauer, Regularisierungsnetze) auf – im Prinzip beliebig – hochdimensionale Probleme anwendbar zu machen. Regularisierungsnetze sind eine Verallgemeinerung von gewöhnlichen Basisfunktionsnetzen, die die gesuchte Lösung durch die Daten parametrisieren, wodurch die explizite Wahl von Knoten/Basisfunktionen entfällt und so bei hochdimensionalen Eingaben der 'Fluch der Dimension' umgangen werden kann. Gleichzeitig sind Regularisierungsnetze aber auch lineare Approximatoren, die technisch einfach handhabbar sind und für die die bestehenden Konvergenzaussagen von Reinforcement Lernen Gültigkeit behalten (anders als etwa bei Feed-Forward Neuronalen Netzen).

Allen diesen theoretischen Vorteilen gegenüber steht allerdings ein sehr praktisches Problem: der Rechenaufwand bei der Verwendung von Regularisierungsnetzen skaliert von Natur aus wie $\mathcal{O}(n^3)$, wobei n die Anzahl der Daten ist. Das ist besonders deswegen problematisch, weil bei Reinforcement Lernen der Lernprozeß *online* erfolgt – die Stichproben werden von einem Agenten/Roboter erzeugt, während er mit der Umwelt interagiert. Anpassungen an der Lösung müssen daher sofort und mit wenig Rechenaufwand vorgenommen werden. Der Beitrag dieser Arbeit gliedert sich daher in zwei Teile:

Im ersten Teil der Arbeit formulieren wir für Regularisierungsnetze einen effizienten Lernalgorithmus zum Lösen allgemeiner Regressionsaufgaben, der speziell auf die Anforderungen von Online-Lernen zugeschnitten ist. Unser Ansatz basiert auf der Vorgehensweise von Recursive Least-Squares, kann aber mit konstantem Zeitaufwand nicht nur neue Daten sondern auch neue Basisfunktionen in das bestehende Modell einfügen. Ermöglicht wird das durch die 'Subset of Regressors' Approximation, wodurch der Kern durch eine stark reduzierte Auswahl von Trainingsdaten approximiert wird, und einer gierigen Auswahlwahlprozedur, die diese Basiselemente direkt aus dem Datenstrom zur Laufzeit selektiert.

Im zweiten Teil übertragen wir diesen Algorithmus auf approximative Politik-Evaluation mittels Least-Squares basiertem Temporal-Difference Lernen, und integrieren diesen Baustein in ein Gesamtsystem zum autonomen Lernen von optimalem Verhalten. Insgesamt entwickeln wir ein in hohem Maße dateneffizientes Verfahren, das insbesondere für Lernprobleme aus der Robotik mit kontinuierlichen und hochdimensionalen Zustandsräumen sowie stochastischen Zustandsübergängen geeignet ist. Dabei sind wir nicht auf ein Modell der Umwelt angewiesen, arbeiten weitestgehend unabhängig von der Dimension des Zustandsraums, erzielen Konvergenz bereits mit relativ wenigen Agent-Umwelt Interaktionen, und können dank des effizienten Online-Algorithmus auch im Kontext zeitkritischer Echtzeitanwendungen operieren. Wir demonstrieren die Leistungsfähigkeit unseres Ansatzes anhand von zwei realistischen und komplexen Anwendungsbeispielen: dem Problem RoboCup-Keepaway, sowie der Steuerung eines (simulierten) Oktopus-Tentakels.

Abstract

This thesis aims at learning autonomously optimal behavior for high-dimensional control tasks using reinforcement learning with a kernel-based approach. Harnessing the representational power of kernel-based methods we hope to escape the so-called 'curse of dimensionality', which otherwise implies an exponential growth in the number of basis functions. Specifically, we apply regularization networks as underlying function approximator in least-squares based policy evaluation. The samples used to build this approximation are generated online, from an agent interacting with the environment. This poses an enormous computational challenge since kernel methods inherently scale with $\mathcal{O}(n^3)$, where n is the number of training samples.

Our first contribution hence is an efficient recursive implementation of regularization networks, particularly tailored for online learning. This is made possible by using the subset of regressors approximation which approximates the kernel using a vastly reduced number of basis functions. To select this subset we employ sparse greedy online selection that automatically constructs a dictionary of basis functions directly from the data stream. Since parsimoniousness is of great importance for efficiency in an online-setting, we extend the original procedure to additionally incorporate a notion of relevance, i.e. the reduction of error in the regression task, thereby eliminating redundancy on-the-fly and further reducing the number of basis functions necessary. The resulting new online algorithm is evaluated on a number of benchmark regression problems and shown to perform well (an order of magnitude faster with only a slight decrease of performance) in comparison with state-of-the-art offline methods.

We then apply this algorithm to carry out approximate policy evaluation and embed it into an approximate policy iteration framework suitable for optimal control. The result is a reinforcement learning method that works online and in a model-free fashion, allows non-deterministic transitions, is very data-efficient and effortlessly scales to high-dimensional state-spaces. We demonstrate this using some high-dimensional benchmarks (simulations), among them RoboCup-Keepaway and the recently proposed control of an octopus-arm.

Danksagung

Zu großem Dank bin ich meinem im Herbst 2005 verstorbenen Doktorvater Prof. Dr. Thomas Uthmann verpflichtet, der diese Arbeit auf zweierlei Weise möglich gemacht hat; einmal, indem er mit seiner Vorlesung mein Interesse an Künstlicher Intelligenz geweckt hat. Und zum zweiten, indem er mich als Doktorand in seine Arbeitsgruppe geholt hatte und mich während meiner Forschungstätigkeit in vielerlei Hinsicht unterstützend begleitet hat.

Ganz herzlich bedanken möchte ich mich auch bei Dr. Daniel Polani, der keine Mühe gescheut und sich auf das Wagnis einer Betreuung aus der Ferne eingelassen hat, sowie bei Prof. Dr. Elmar Schömer, der als Ersatz-Doktorvater eingesprungen ist und sich in das Thema vertieft hat, obwohl dieses nicht direkt zu seinem Arbeitsgebiet gehört hat.

Diese Dissertation entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Informatik der Johannes Gutenberg-Universität Mainz. Bedanken möchte ich mich in diesem Zusammenhang nicht zuletzt auch für die freundliche Atmosphäre und Unterstützung die ich dabei erfahren habe, und die nicht unwesentlich dazu beigetragen hat, daß die Promotion auch in turbulenteren Zeiten normal weiter laufen konnte.

Für meine Eltern

Inhaltsverzeichnis

Kapitel 1

Einleitung

Worin beschrieben wird, welches Problem auf welche Weise diese Arbeit zu lösen gedenkt

Sequentielle Entscheidungs- oder Belohnungsprozesse sind ein Formalismus, mit dem wir eine große Klasse von praktischen Problemen aus unterschiedlichen Bereichen einheitlich darstellen und lösen können. Beispielsweise läßt sich damit im Bereich Operations Research das Problem modellieren, wie eine Menge von Ressourcen (Mitarbeiter, Produktionsmittel, Geld) über einen langen Zeitraum maximal gewinnbringend eingesetzt werden kann. Oder es läßt sich damit, im Zusammenhang mit Kontrolltheorie, die optimale zeitliche Steuerung komplizierter technischer Anlagen oder Prozesse beschreiben. Speziell und vor allem sind sie aber auch für die künstliche Intelligenz von großem Interesse, weil sich mit ihnen zum ersten Mal vernünftig komplexes rationales Verhalten von Agenten und damit das Problem der autonomen Steuerung simulierter oder realer Roboter abbilden läßt. Bei allen diesen Aufgaben handelt es sich um ein spezielles Optimierungsproblem, das durch folgende gemeinsame Eigenschaften charakterisiert wird:

- die Aufgabe verlangt nicht eine einzelne Entscheidung, sondern eine ganze Abfolge davon
- das Endresultat hängt in komplizierter Weise von der gesamten Abfolge der zuvor getroffenen Entscheidungen und ihren jeweiligen Konsequenzen ab.
- das Resultat jeder einzelnen Entscheidung kann von zufälligen (nicht-kontrollierbaren) Faktoren abhängen, deren genaue Natur/Systematik nicht exakt beschrieben werden kann oder schlicht unbekannt ist.

Im allgemeinen sind für solche Aufgaben optimale Entscheidungsstrategien (im folgenden *Politik* genannt) nicht bekannt und selbst für einen menschlichen Experten ist es mitunter sehr schwierig und aufwendig, eine ausreichend gute Politik zu formulieren. In der künstlichen Intelligenz wird unter dem Etikett „Reinforcement Lernen“ (eine spezielle Form von approximativem Dynamischen Programmieren) daher folgender Lösungsansatz verfolgt: wir betrachten Algorithmen, die selbständig gute Politiken für sequentielle Entscheidungsprobleme *erlernen*, einfach indem sie mit dem Problem interagieren (verschiedene Handlungsalternativen ausprobieren) und sich merken, welche Aktionen zu einem guten Endergebnis geführt haben und welche nicht. Eine zentrale Rolle nimmt dabei eine globale Bewertungsfunktion ein, die für jede mögliche Situation und Entscheidung deren absolute Güte (im Hinblick auf den langfristigen Nutzen) quantifiziert. Das Ziel von Reinforcement Lernen ist es, diese Funktion zu erlernen. Ist die Funktion erst einmal bekannt bzw. gelernt worden, können daraus leicht optimale (oder zumindest gute) Entscheidungen abgeleitet werden.

In den folgenden Abschnitten der Einleitung wollen wir, bevor wir ausführlicher erläutern, worin der Beitrag dieser Dissertation besteht, zunächst etwas näher ausführen, wie dieser Sachverhalt mathematisch modelliert wird und dann erläutern, wie die zugehörigen elementaren Lösungsverfahren von Dynamischen Programmieren (DP) und Reinforcement Lernen (RL) aussehen und welche Schwierigkeiten dabei auftreten.

1.1 Dynamisches Programmieren

Die Wertefunktion

Der Ausgangspunkt ist die folgende grundlegende Situation: wir betrachten ein dynamisches System, dessen Entwicklung diskret in der Zeit voranschreitet und von getroffenen Entscheidungen (*Aktionen*) vorangetrieben wird. Zu jedem Zeitpunkt $t = 0, 1, 2, \ldots$ befindet sich das System in einem Zustand $s_t \in \mathcal{S}$ (wobei die Menge aller möglichen Zustände $\mathcal{S}$ eine endliche Menge, z.B. die Menge aller Brettpositionen im Spiel Schach, oder unendlich, z.B. $\mathcal{S} \subset \mathbb{R}^d$, sein kann) und geht, unter Auswahl einer Aktion $a_t \in \mathcal{A}$ (wobei die Menge aller möglichen Aktionen $\mathcal{A}$ ebenfalls eine endliche Menge, z.B. die Menge aller Spielzüge bei Schach, oder unendlich, z.B. $\mathcal{A} \subset \mathbb{R}^d$, sein kann) in einen neuen Zustand s_{t+1} über. Der Einfachheit halber betrachten wir zunächst endliche Zustandsräume; außerdem werden wir in der gesamten Arbeit auch nur endliche Aktionsräume behandeln.

Die Zustandsübergänge von s_t nach s_{t+1} dürfen im allgemeinen Fall stochastisch sein und werden mittels einer Verteilung

$$s_{t+1} \sim \mathrm{P}(\cdot\,|s_t, a_t)$$

modelliert, hängen also nur von dem vorangehenden Zustand s_t und der gewählten Aktion a_t ab (Markov-Eigenschaft).

Unter einer *Politik* π verstehen wir eine Abbildung $\pi : \mathcal{S} \to \mathcal{A}$, die zu jedem Zustand s eine Aktion $\pi(s)$ vorschreibt. Werden die Aktionen gemäß einer festen Politik π ausgewählt, dann reduziert sich das System auf einen Markov-Prozeß über $\mathcal{S}$ mit Übergangswahrscheinlichkeiten $\mathrm{P}(s_{t+1}|s_t, \pi(s_t))$.

Jede Auswahl einer Aktion (und der damit einhergehende Zustandsübergang) ist mit einer reellwertigen Größe, der *Belohnung* (oder 1-Schritt Auszahlung) $r_{t+1} = \mathrm{R}(s_{t+1}|s_t, a_t)$ verbunden, die von s_t, a_t und dem eingetretenen Folgezustand s_{t+1} abhängt. Die Belohnung kann als eine künstliche Hilfsgröße betrachtet werden, aus der wir die uns eigentlich interessierende Quantität „Bewertung“ konstruieren werden. Die grundlegende Annahme ist, daß die Belohnungsfunktion stets bekannt ist (üblicherweise läßt sie sich in der Praxis zu einem konkret vorliegenden Lernproblem meist auch sehr leicht spezifizieren).

Mit jeder Politik π assoziieren wir eine über den Zuständen definierte *Wertefunktion* $V^\pi : \mathcal{S} \to \mathbb{R}$. Sie wird als Erwartungswert aller zukünftigen Belohnungen (die erwartete Gesamtauszahlung)

$$V^\pi(s) = \mathbb{E}\left\{ \sum_{t \geq 0} \gamma^t \mathrm{R}(s_{t+1}|s_t, \pi(s_t)) \;\middle|\; s_0 = s \right\} \quad \forall s \tag{1.1}$$

definiert, wenn $s_0 = s$ und die Folgezustände für $t = 0, 1, 2 \ldots$ gemäß $s_{t+1} \sim \mathrm{P}(\cdot\,|s_t, \pi(s_t))$ erzeugt werden. Der Parameter $\gamma \in [0, 1)$ wird als Diskontfaktor bezeichnet und reduziert den Beitrag von weit in der Zukunft erfolgenden Ereignissen. [1]

Beispiel: Mountain-Car

Ein sehr einfaches für den Rest der Arbeit aber repräsentatives „Spielzeug“-Problem mag uns das soeben Gesagte anschaulich illustrieren. Beim Problem „Mountain-Car“ aus Sutton und Barto (1998) handelt es sich um ein zeit-diskretes Steuerungsproblem mit kontinuierlichem Zustandsraum und deterministischer Übergangsfunktion. Ferner ist es ein episodisches Problem mit ausgezeichneten Terminalzuständen. Die Aufgabe besteht, wie in Abbildung 1.1 dargestellt, darin, ein Fahrzeug in die gewünschte Endposition auf den rechten Hügel zu bringen. Für ein direktes Erklimmen des Hügels reicht jedoch die Motorkraft nicht aus, stattdessen muß zunächst der gegenüberliegende Hügel angesteuert werden, um genügend Schwung aufzubauen.

[1] $\gamma = 1$ ist möglich, wenn sichergestellt ist, daß jeder Zustand unter der aktuellen Politik mit Wahrscheinlichkeit > 0 erreicht werden kann und es einen absorbierenden Zustand (Terminalzustand) gibt.

Der Zustand des Systems wird beschrieben durch die Position und Geschwindigkeit $s_t = (x, \dot{x}) \in \mathcal{S}$, wobei $\mathcal{S} = [-1.2, 0.5] \times [-0.07, 0.07]$. Mögliche Steuerbefehle sind „links“, „rechts“ oder „keine Änderung“, der Aktionsraum ist $\mathcal{A} = \{-1, 0, 1\}$. Die deterministische Zustandsübergangsfunktion lautet

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_{t+1} \\ \dot{x}_{t+1} &= \dot{x}_t + 0.001 a_t - 0.0025 \cos(3x_t). \end{aligned}$$

Der linke Rand ist von einer Mauer umgrenzt: ist $x_{t+1} \leq -1.2$, so wird $x_{t+1} := -1.2$ und $\dot{x}_{t+1} := 0$ gesetzt. Das Problem gilt als gelöst, sobald $x_{t+1} \geq 0.5$.

Als nächstes müssen wir noch eine Belohnungsfunktion angeben. Verwenden wir etwa

$$\mathrm{R}(s_{t+1}|s_t, a_t) = \begin{cases} 0 & \text{falls } x_{t+1} \geq 0.5 \\ -1 & \text{sonst} \end{cases}$$

dann ist der Wert eines Zustandes proportional zur Anzahl der notwendigen Schritte zum Ziel und die optimale Politik wird diejenige sein, die den jeweils kürzesten Weg (mit den wenigsten Schritten) wählt. Alternativ könnten wir in der Belohnungsfunktion auch den Energieverbrauch einbeziehen und beispielsweise Gasgeben bestrafen. In diesem Fall würden wir dann nicht mehr den schnellsten Weg zum Ziel, sondern denjenigen mit dem geringsten Energieaufwand suchen. Allein durch die Wahl der Belohnungsfunktion spezifizieren wir also, welches dahinterstehende Entscheidungsproblem gelöst werden soll.

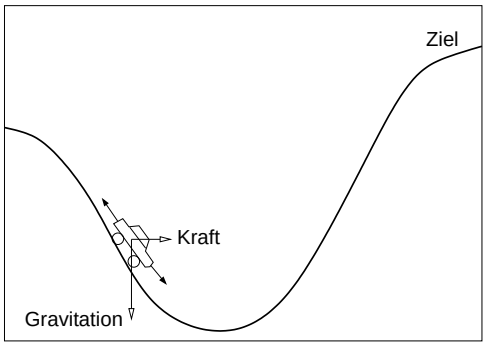


Abbildung 1.1: Das Mountain-Car Problem. Ziel ist, den Wagen mit möglichst wenig Schritten auf den rechten Hügel zu steuern.

Politik-Evaluation

Um die erwartete Gesamtauszahlung in Gleichung (1.1) zu einem gegebenen π auszurechnen, machen wir von einem Standardergebnis des Dynamischen Programmierens Gebrauch, welches besagt, daß (1.1) äquivalent geschrieben werden darf als

$$V^\pi(s) = \mathbb{E}_{s_{t+1} \sim \mathrm{P}(\cdot\,|s_t, \pi(s_t))} \left\{ \mathrm{R}(s_{t+1}|s_t, \pi(s_t)) + \gamma V^\pi(s_{t+1}) \;\middle|\; s_t = s \right\} \quad \forall s \tag{1.2}$$

d.h. daß die Erwartung über die gesamte Zukunft in (1.1) ersetzt werden darf nur durch die Erwartung über den unmittelbaren Nachfolgezustand.

Für eine endliche Zustandsmenge mit N Zuständen kann die gesamte Wertefunktion als $N \times 1$ Vektor V^π mit Einträgen $[V^\pi]_i = V^\pi(i)$ geschrieben werden. In diesem Fall ist (1.2) ein lineares Gleichungssystem mit N Variablen und kann mit numerischen Standardverfahren exakt nach der unbekannten V^π aufgelöst werden.

Alternativ können wir (1.2) auch als lineare Fixpunktgleichung auffassen: definieren wir den Bellman-Operator $\mathcal{T}_\pi : \mathbb{R}^N \to \mathbb{R}^N$ mittels

$$\big(\mathcal{T}_\pi V\big)(s) := \mathbb{E}_{s' \sim \mathrm{P}(\cdot\,|s, \pi(s))} \left\{ \mathrm{R}(s'|s, \pi(s)) + \gamma V(s') \right\} \quad \forall s,$$

dann ist V^π Lösung der Fixpunktgleichung $V = \mathcal{T}_\pi V$. Der Bellman-Operator $\mathcal{T}_\pi$ ist eine Kontraktion (bzgl. $\mathcal{L}_\infty$-Norm), die gesuchte Lösung V^π kann daher näherungsweise mittels Fixpunktiteration $V_{k+1} = \mathcal{T}_\pi V_k$ bestimmt werden.

Optimale Politik, optimale Wertefunktion und Werte-Iteration

Gemeinhin wollen wir uns aber nicht bloß damit zufriedengeben, die Wertefunktion einer beliebigen Politik ausrechnen zu können. Was uns vielmehr interessiert, ist eine *optimale* Politik, die uns zur bestmöglichen (maximalen) Gesamtauszahlung führt. Eine optimale Politik ist eine (nicht notwendigerweise eindeutig bestimmte) Politik π^* mit der Eigenschaft $V^{\pi^*}(s) \geq V^\pi(s)\ \forall s, \forall \pi$. [2] Die zu jeder optimalen Politik π^* gehörende Wertefunktion ist die (eindeutig bestimmte) optimale Wertefunktion $V^* := V^{\pi^*}$.

Die optimale Wertefunktion V^* erfüllt eine zu (1.2) analoge Beziehung, die Bellman-Optimalitätsgleichung:

$$V^*(s) = \max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V^*(s') \right\} \quad \forall s.$$

Sie ist somit ebenfalls Lösung eines Systems nicht-linearer Fixpunktgleichungen und kann analog zum Fall Politik-Evaluation iterativ bestimmt werden. Definieren wir den Bellman-Operator $\mathcal{T}_* : \mathbb{R}^N \to \mathbb{R}^N$ mittels

$$\big(\mathcal{T}_* V\big)(s) := \max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V(s') \right\} \quad \forall s,$$

dann ist V^* Lösung von $V = \mathcal{T}_* V$. Wie zuvor, so kann auch hier gezeigt werden, daß $\mathcal{T}_*$ eine Kontraktion (bzgl. $\mathcal{L}_\infty$-Norm) ist und die gesuchte Lösung daher mittels der Fixpunktiteration $V_{k+1} = \mathcal{T}_* V_k$ näherungsweise bestimmt werden kann. Das dazugehörende Verfahren wird im Dynamischen Programmieren als *Werte-Iteration* bezeichnet und kann (unter bestimmten Voraussetzungen) durch geschicktes Anordnen der Reihenfolge in der die Aktualisierungen vorgenommen werden, effizient durchgeführt werden.

Politik-Ableitung: gierige Politik und Politik-Verbesserung

Ist die optimale Wertefunktion V^* (und damit die zu erwartende Gesamtauszahlung eines jeden Zustandes) erst einmal bekannt, so können wir diese als Grundlage nehmen, um zukünftig für jeden beliebigen Zustand s die jeweils optimale Aktion

$$\operatorname*{argmax}_{a \in \mathcal{A}} \mathbb{E}_{s' \sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V^*(s') \right\}$$

auszuwählen.

Etwas allgemeiner, wenn immer wir aus einer beliebigen Wertefunktion V' eine Politik π' mittels

$$\pi'(s) := \operatorname*{argmax}_{a \in \mathcal{A}} \mathbb{E}_{s' \sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V'(s') \right\} \quad \forall s$$

erzeugen, so heißt dieser Vorgang (gierige) *Politik-Ableitung* und π' ist die bezüglich V' *gierige* Politik. Eine optimale Politik π^* geht durch gierige Politik-Ableitung aus V^* hervor.

Das Politik-Verbesserungs-Theorem besagt dann folgendes: geht eine Politik π_2 aus einer Wertefunktion V^{π_1} zu einer Politik π_1 durch gierige Politik-Ableitung hervor, dann ist die neue Politik π_2 mindestens genauso gut wie π_1, d.h. es gilt $V^{\pi_2}(s) \geq V^{\pi_1}(s)\ \forall s$. Gilt Gleichheit $\forall s$, dann ist die Politik optimal, ansonsten ist für mindestens ein s die Aussage $V^{\pi_2}(s) \geq V^{\pi_1}(s)$ erfüllt. Mit gieriger Politik-Ableitung ist es somit allgemein möglich, aus einer vorgegebenen Politik eine noch bessere zu machen. Die optimale Wertefunktion V^* ist diejenige Wertefunktion, bei der gierige Politik-Ableitung zu keiner weiteren Verbesserung mehr führt.

[2] Zu jedem sequentiellen Markov-Entscheidungsprozeß mit endlichem Zustands- und Aktionsraum gibt es eine deterministische optimale Politik (Bertsekas und Tsitsiklis, 1996).

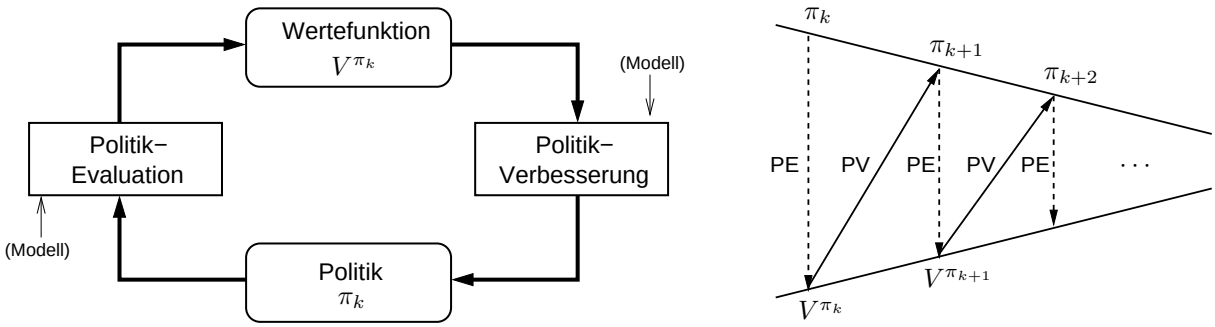


Abbildung 1.2: Politik-Iteration iteriert Evaluation und Verbesserung (linke Seite) und ist ein Prozeß, der zu immer besseren und schließlich der optimalen Politik führt (rechte Seite).

Zusammenfassung: Werte-Iteration und Politik-Iteration

Um die optimale Politik zu einem vorliegenden sequentiellen Entscheidungsproblem zu bestimmen, sind somit zweierlei Vorgehensweisen möglich:

1. **Werte-Iteration:** Wir berechnen V^* direkt durch Fixpunktiteration mit $\mathcal{T}_*$ (d.h. starte mit $V_0 \equiv 0$ und iteriere $V_{k+1} = \mathcal{T}_* V_k$ für $k = 0, 1, 2, \ldots$, Konvergenz $V_k \to V^*$). Sobald V^* hinreichend genau bekannt ist, kann daraus dann mit gieriger Politik-Ableitung die optimale Politik π^* bestimmt werden.

2. **Politik-Iteration:** Wir bestimmen V^{π^*} indirekt durch Iteration von Politiken π_k. Wie in Abbildung 1.2 gezeigt, beginnen wir mit einer initialen Politik π_0 und führen dann solange abwechselnd die Schritte Politik-Evaluation und Politik-Verbesserung aus, bis eine resultierende Wertefunktion V^{π_k} invariant unter ihrer weiteren Anwendung geworden ist, d.h. $V^{\pi_{k+1}} = V^{\pi_k}$ für ein k gilt. Dann ist π_k eine optimale Politik. Weil es bei einem endlichen Zustandsraum nur endlich viele deterministische Politiken gibt, ist die Konvergenz sichergestellt. Schon an dieser Stelle lohnt es sich vor allem im Hinblick auf die noch kommenden Entwicklungen (wo Politik-Iteration eine tragende Rolle spielen wird) besonders darauf hinzuweisen, daß es sich dabei um zwei ineinander verschachtelte Prozesse auf unterschiedlichen Zeitskalen handelt:

   - der äußere Prozeß ist eine Iteration über Politiken $\pi_1, \pi_2, \ldots$
   - der innere Prozeß implementiert den Schritt Politik-Evaluation für das jeweilige π_i und ist seinerseits eine Iteration, wenn V^{π_i} durch Fixpunktiteration für $\mathcal{T}_{\pi_i}$ bestimmt wird (d.h. starte mit $V_0 \equiv 0$ und iteriere $V_{k+1} = \mathcal{T}_{\pi_i} V_k$ für $k = 0, 1, 2, \ldots$, Konvergenz $V_k \to V^{\pi_i}$).

Warum das nicht ausreicht?

So schön und elegant die soeben beschriebene Theorie auch anmutet, so wenig läßt sie sich in dieser Form auch in die Praxis umsetzen. Zweierlei Dinge wurden bislang (stillschweigend) vorausgesetzt, mit denen reale Probleme in der Regel nicht aufwarten können:

- Zustandsraum endlich (und klein!)
- Modell, d.h. Übergangswahrscheinlichkeiten $\mathrm{P}(s_{t+1}|s_t, a_t)$ und Belohnungsfunktion $\mathrm{R}(s_{t+1}|s_t, a_t)$ bekannt.

Bei vielen Problemen mag der Zustandsraum zwar endlich, aber doch auch ungeheuer groß sein (z.B. Schach $\sim 10^{50}$ verschiedene Zustände). Bei anderen Problemen wieder (z.B. der Steuerung eines Roboters) ist der Zustandsraum von vorneherein kontinuierlich ($\mathcal{S} \subset \mathbb{R}^d$) und möglicherweise gar hochdimensional ($d = 10$, $d = 100$ oder mehr ist keine Seltenheit). In beiden Fällen können die Zustandswerte nicht

mehr einzeln als Aufzählung in einem *Vektor* erfaßt werden. Stattdessen müssen wir das gesuchte Objekt, die Wertefunktion, tatsächlich auch als *Funktion* auffassen und mit den Methoden der statistischen Funktionsapproximation behandeln.

Das zweite Problem ist die Notwendigkeit eines Modells $\mathrm{P}(s_{t+1}|s_t, a_t)$, $\mathrm{R}(s_{t+1}|s_t, a_t)$ um die Erwartungswertberechnung in den Schritten Politik-Evaluation und Politik-Verbesserung exakt durchführen zu können (siehe Abbildung 1.2). Häufig wird sich das exakte Bestimmen der Übergangswahrscheinlichkeiten als sehr schwierig gestalten (z.B. bei der Steuerung von Robotern in realen Welten), manchmal auch als zu aufwendig oder schlicht unmöglich (bei Schach muß der Gegenspieler, allgemeiner bei Multiagentensystemen muß das Verhalten aller anderen Agenten quantifiziert werden können). Eine Lösung dieses Problems ist es, den Erwartungswert durch Stichproben zu approximieren, die durch Simulation des Systems gewonnen werden können.

Lernen bei autonomen Agenten

Insbesondere können die Stichproben aber auf eine ganz besondere Weise erzeugt werden, nämlich als die Abfolge von Zuständen/Belohnungen, die eine Maschine (in der künstlichen Intelligenz ein autonomer Agent) *real beobachtet*, wenn sie mit dem System interagiert und die Aktionen gemäß der Politik tatsächlich ausführt.

Hierin liegt die eigentliche Stärke und das Reizvolle am Reinforcement Lernen: gegeben ein Agent, der eine bestimmte festgelegte Aufgabe ausführen soll, dann reicht es aus, wenn der Agent mit dem der Aufgabe zugrundeliegenden System real oder simulativ interagieren und

- den einer Aktion folgenden Zustand
- die dieser Aktion folgende Belohnung (1-Schritt Auszahlung)

beobachten kann um optimales Verhalten selbständig zu *erlernen*. Etwas zugespitzt bedeutet das beispielsweise für einen Schachcomputer, daß es ausreichend ist, lediglich gegen einen Gegner zu spielen, sich dabei zu merken, wie der Gegenspieler auf Züge reagiert und zu registrieren, wenn das Spiel gewonnen/verloren wurde, um mit der Zeit eine nahezu optimale Lösungsstrategie für dieses Problem zu erlernen (sofern der Gegner stationär ist und seine Strategie nicht ändert).

1.2 Approximatives Dynamisches Programmieren

Um einen funktionalen Zusammenhang zwischen Zuständen und deren erwarteter Gesamtauszahlung mit den Methoden der statistischen Funktionsapproximation erlernen zu können, sind zweierlei Dinge zu tun:

1. Wahl einer allgemeinen Funktionsapproximations-Architektur
2. Wahl eines Lernverfahrens für die unbekannten Parameter.

Da Funktionen im allgemeinen beliebig komplizierte Gebilde sein können, besteht der erste Schritt zunächst darin, eine kompakte Form der Repräsentation zu wählen, so daß die gesuchte Funktion nur von einer (relativ) *kleinen* Anzahl frei bestimmbarer Parameter abhängt. Genauer gesagt, die gesuchte (unbekannte) Wertefunktion $V^\pi : \mathcal{S} \to \mathbb{R}$ wird durch eine *parametrisierte* Funktion $\tilde{V} : \mathcal{S} \times \mathbb{R}^m \to \mathbb{R}$ ersetzt, so daß

$$\tilde{V}(s; \mathbf{w}) \approx V^\pi(s)$$

für eine bestimmte Belegung des Parametervektors $\mathbf{w} \in \mathbb{R}^m$ gilt (wodurch das ursprüngliche Problem vom Bestimmen einer *Funktion* auf das Finden einer Belegung des Parameter*vektors* reduziert wird). Anschließend müssen wir das eigentliche Dynamische Programmieren dahingehend abändern, daß nunmehr der Parametervektor $\mathbf{w}$ gelernt wird. Betrachten wir diese beiden Schritte nun im Detail:

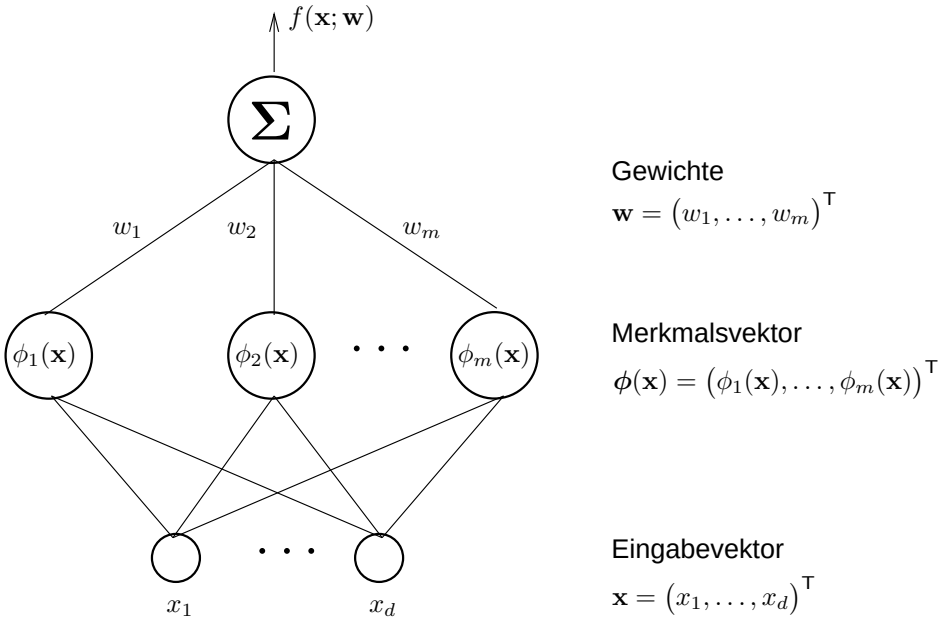


Abbildung 1.3: Graphische Repräsentation linear parametrisierter Modelle durch ein zweischichtiges Basisfunktionsnetzwerk. Das gezeigte Netz mit Basisfunktionen $\phi_1, \ldots, \phi_m$ und Gewichten $w_1, \ldots, w_m$ realisiert eine Funktion $f(\mathbf{x};\mathbf{w}) = \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x})$.

1.2.1 Wahl einer allgemeinen Funktionsapproximations-Architektur

Linear parametrisierte Modelle

Eine spezielle Architektur sind die linear parametrisierten (oder Basisfunktions-) Modelle. Hier betrachten wir die gesuchte Funktion als eine lineare Kombination der Form

$$\tilde{V}(s;\mathbf{w}) = \sum_{i=1}^{m} w_i \phi_i(s) \tag{1.3}$$

wobei die ϕ_i ihrerseits besonders ausgewählte Funktionen $\phi_i : \mathcal{S} \to \mathbb{R}$ und die w_i die zu bestimmenden Parameter oder Entwicklungskoeffizienten sind. Im Zusammenhang mit „Neuronalen Netzen“ spricht man in der Informatik von (1.3) auch als von einem Basisfunktionsnetzwerk; die ϕ_i werden als *Basisfunktionen* (Aktivierungsfunktionen von Neuronen) und die w_i als *Gewichte* bezeichnet. Abbildung 1.3 zeigt die Darstellung eines solchen Netzes. Definieren wir zur weiteren Abkürzung den Vektor der Basisfunktionsaktivierungen (transformierte Eingaben oder Merkmalsvektor bzw. Featurevektor genannt) $\boldsymbol{\phi}(s) := \big(\phi_1(s), \ldots, \phi_m(s)\big)^\mathsf{T}$ und den gesuchten Gewichtsvektor $\mathbf{w} := \big(w_1, \ldots, w_m\big)^\mathsf{T}$, dann wird auch optisch unmittelbar einsichtig, daß $\tilde{V}(s;\mathbf{w}) = \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(s)$ eine lineare Funktion bezüglich der Gewichte $\mathbf{w}$ ist.

Wahl und Bedeutung der Basisfunktionen

Mit der Wahl der einzelnen Basisfunktionen ϕ_i legen wir die allgemeine Form und Komplexität der zu lernenden Funktion fest. Die ϕ_i selbst kann man sich dabei als elementare Funktionen vorstellen, die jeweils einen ganz bestimmten Aspekt (ein *Merkmal*) aus einem Zustand extrahieren. Anstatt Zustände s direkt zu verwenden, operieren wir dann auf einer vektorwertigen Repräsentation von s mit den Ausprägungen $\phi_1(s), \ldots, \phi_m(s)$ in den m Merkmalen. Das Besondere dabei ist, daß wir auf diese Weise vom allgemeinen Objekt „Zustand“ auf Vektoren übergehen und daher die nachfolgenden Algorithmen unabhängig vom konkreten Aussehen der Zustände formulieren können. Als Eingabe dienen ausschließlich die Vektoren

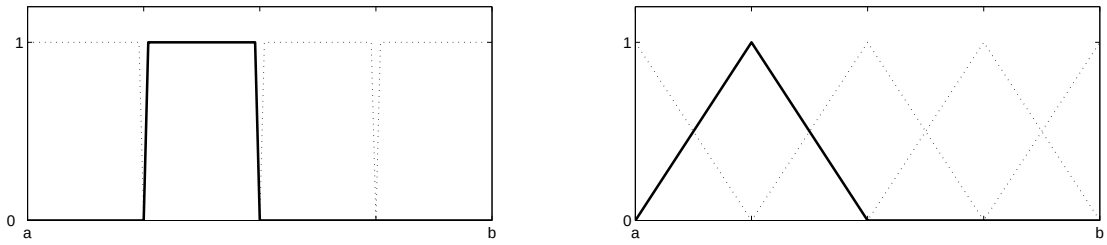


Abbildung 1.4: B-Splines 1. Ordnung (links) und 2. Ordnung (rechts) als System möglicher Basisfunktionen ϕ_i für den Zustandsraum $\mathcal{S} = [a, b]$.

$\boldsymbol{\phi}(s)$; für den Algorithmus spielt es daher keine Rolle mehr, ob der dahinter stehende Zustandsraum eine Menge diskreter Objekte oder ein kompaktes Gebiet im $\mathbb{R}^d$ ist.

Die schlechte Nachricht ist, daß es für die Wahl der ϕ_i zumindest im Fall *diskreter Zustandsräume* bislang kein allgemeingültiges Kochrezept gibt, sondern diese von Hand für jedes Problem maßgefertigt werden müssen. Auf diese Weise ist es einerseits zwar möglich, Wissen über das ungefähre Aussehen der gesuchten Funktion explizit einzubringen, was andererseits aber auch bedeutet, daß solches Expertenwissen (oder eine gute Vorahnung) erst einmal vorliegen muß. Betrachten wir hierfür wieder das Beispiel Schach. Ein Bestandteil der Bewertungsfunktion von Schachprogrammen ist typischerweise der Figurenvorteil. Für eine Brettposition s (Zustand) könnte eine parametrisierte Wertefunktion dann wie folgt aussehen:

$$\tilde{V}(s; \mathbf{w}) = w_1 \cdot \text{VBauern}(s) + w_2 \cdot \text{VSpringer}(s) + w_3 \cdot \text{VLäufer}(s) + w_4 \cdot \text{VTurm}(s) + w_5 \cdot \text{VDame}(s)$$

wobei VBauern(s) die Differenz zwischen der Anzahl eigener und der Anzahl gegnerischer Bauern in s ist (der Rest analog). Haben wir uns auf diese Weise für eine spezielle Parametrisierung entschieden, so müssen dann (mit einem Lernverfahren) die Parameter $w_1, \ldots, w_5$ so bestimmt werden, daß auf der Grundlage von $\tilde{V}$ eine gute Entscheidungsstrategie abgeleitet werden kann.[3]

Als ungleich einfacher gestaltet sich dagegen (zunächst) die Wahl der Basisfunktionen ϕ_i für den Fall *kontinuierlicher Zustandsräume* (weswegen wir uns in dieser Arbeit auch allein auf diesen Fall konzentrieren wollen). Eine simple Möglichkeit für den univariaten Fall wäre es z.B., den betrachteten Zustandsraum $\mathcal{S} = [a, b]$ in m gleichgroße Teilstücke aufzuteilen und darüber binäre Merkmale (aktiv oder nicht aktiv) zu betrachten (siehe auch Abbildung 1.4):

$$\phi_i(s) = \begin{cases} 1 & \text{falls } s \in [a + \frac{i-1}{m}(b-a),\ a + \frac{i}{m}(b-a)) \\ 0 & \text{sonst} \end{cases} \quad , i = 1 \ldots m$$

Etwas allgemeiner entspricht das den in der Approximation bekannten B-Splines 1. Ordnung. Abbildung 1.4 zeigt zusätzlich B-Splines höherer Ordnung, die ebenfalls als Basissystem Verwendung finden. Was B-Splines (zumindest für die Approximation niedrig-dimensionaler Probleme) so ungemein attraktiv macht, ist ihre Eigenschaft, einen kompakten Träger zu besitzen, d.h. nur auf einem kleinen Teil des Definitionsbereichs Werte ungleich Null anzunehmen. Diese Eigenschaft ist maßgeblicher Garant für eine effiziente technische Umsetzung und macht es überhaupt möglich, Modelle mit vielen (hundert-) tausenden Basisfunktionen zu betrachten.

Für den multivariaten Fall, d.h. der Zustandsraum ist ein d-dimensionaler Quader, können mehrdimensionale Spline-Basisfunktionen aus dem (Tensor-)Produkt der eindimensionalen Splines erstellt werden. Beispielsweise für den Fall $d = 2$ (siehe dazu auch Abbildung 1.5): seien ϕ_i^1, $i = 1, \ldots, m_1$ die univariaten Basisfunktionen entlang Koordinate x_1 und ϕ_j^2, $j = 1, \ldots, m_2$ diejenigen entlang Koordinate x_2. Dann ist das Produkt gleich

$$\phi_{i,j}(\mathbf{x}) = \phi_{i,j}(x_1, x_2) = \phi_i^1(x_1) \cdot \phi_j^2(x_2) \quad , i = 1 \ldots m_1, j = 1 \ldots m_2$$

[3] Viele Schachbücher schlagen die Faustregel $\mathbf{w} = (1, 3, 3, 5, 9)$ vor. Diese Belegung der Gewichte wurde allerdings mehr aus der umfangreichen Historie menschlicher Spielerfahrung gewonnen und nicht etwa von einer Maschine automatisch gelernt. Der von IBM entwickelte (spielstarke) Schachcomputer Deep Blue verwendet $\sim$ 8000 solcher Merkmale, deren Gewichte von menschlichen Experten festgelegt/getuned wurden (Campbell et al., 2002).

und die Approximation $f(\mathbf{x}; \mathbf{w})$ ist von der allgemeinen Form

$$f(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{m_1} \sum_{j=1}^{m_2} w_{i,j} \phi_{i,j}(\mathbf{x})$$

wobei die Anzahl der Gewichte/Basisfunktionen nun $m_1 \times m_2$ ist. Für den allgemeinen Fall von d Dimensionen kann diese Prozedur analog angewendet werden. Zu beachten ist allerdings, und das ist in der Tat der gravierende Nachteil, daß dann die Anzahl der Basisfunktionen exponentiell in d anwächst!

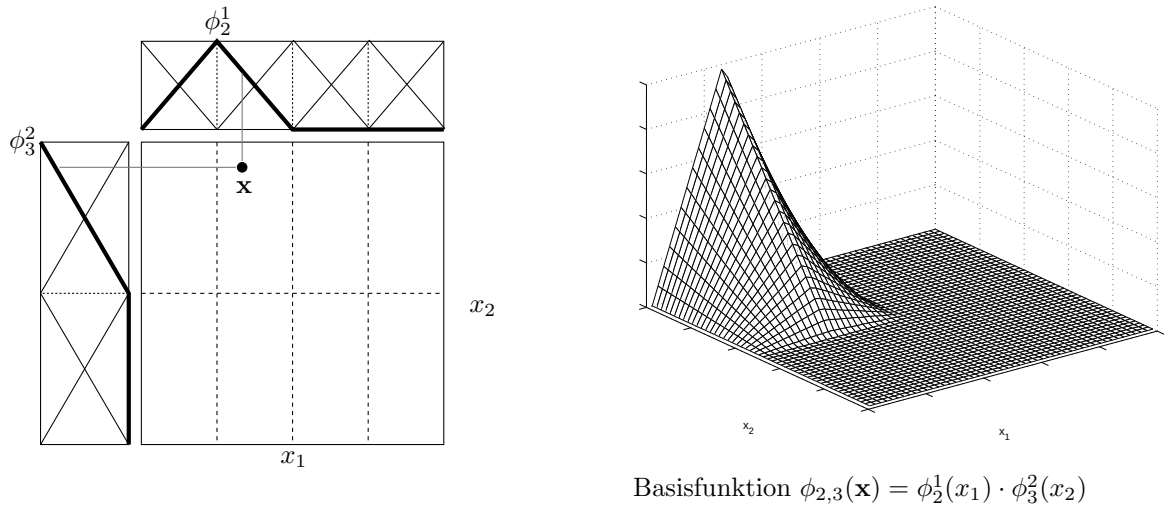
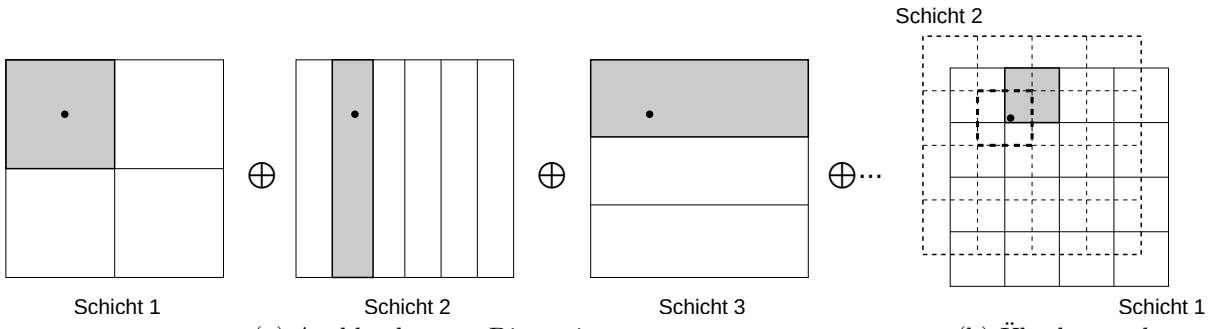


Abbildung 1.5: Verallgemeinerung von eindimensionalen Splines auf mehrdimensionale Räume, hier für 2 Dimensionen dargestellt. Die multivariate B-Spline Basis wird aus dem Tensorprodukt univariater B-Splines erzeugt.

Beispiel: Tilecoding oder CMAC

Speziell für den Einsatz mit Reinforcement Lernen schlagen Sutton und Barto (1998) das sogenannte *Tilecoding* vor, basierend auf dem CMAC Ansatz (cerebellar model articulation controller) von Albus (1975), welcher seinerseits lose (sehr lose) an die Funktionsweise des Gehirns angelehnt ist. In mancherlei Hinsicht ähnelt Tilecoding auch den sogenannten dünnen Gittern (Garcke, 2004). Tilecoding ist ein gitterbasiertes Verfahren mit binärer Aktivierungsfunktion (die Gitterzellen werden als „rezeptive“ Felder interpretiert) und ist aus einer oder mehreren Schichten aufgebaut. Jede Schicht überdeckt den Zustandsraum komplett, kann aber jede Dimension unterschiedlich fein auflösen oder auch ganz ausblenden (siehe Abbildung 1.6 links). Im allgemeinen nehmen einzelne Schichten aber nur eine grobe Rasterung vor; die feinere Auflösung wird dadurch erzielt, daß die Schichten dupliziert und versetzt eingesetzt werden (siehe Abbildung 1.6 rechts). Solange binäre Basisfunktionen verwendet werden, produziert Tilecoding stückweise konstante Funktionen.

Technisch kann Tilecoding wieder sehr effizient implementiert werden, weil die Anzahl der aktiven Elemente im Merkmalsvektor gerade gleich der Anzahl der Schichten ist und es trivial ist, den Index der jeweils aktiven Zelle zu bestimmen. Darüberhinaus mildert Tilecoding bis zu einem gewissen Grad das exponentielle Wachstum der Anzahl der Gewichte ab (weil nicht mehr das volle kartesische Produkt aller Dimensionen betrachtet wird), kann es aber letztenendes auch nicht verhindern. In der Praxis wird Tilecoding in Verbindung mit Reinforcement Lernen (auch aus Gründen der Laufzeiteffizienz) eher ad-hoc eingesetzt: die Rasterung wird vor dem Lernen durch Ausprobieren festgelegt und nicht adaptiv an den Approximationsfehler angepaßt.



(a) Ausblenden von Dimensionen (b) Überlappend

Abbildung 1.6: Tilecoding besteht aus der direkten Summe einzelner (inhomogener) Schichten, die jeweils eine komplette Partitionierung des Zustandsraumes vornehmen. Diese sind nicht homogen, jede einzelne Schicht kann dabei eine unterschiedliche Partitionierung realisieren.

1.2.2 Wahl eines Lernverfahrens für den unbekannten Parametervektor

Analog zu den beiden prinzipiellen Vorgehensweisen Werte-Iteration und Politik-Iteration im klassischen Dynamischen Programmieren betrachten wir, um im approximativen Dynamischen Programmieren die optimale Politik näherungsweise zu bestimmen, die Möglichkeiten

- approximative Werte-Iteration (AVI)
- approximative Politik-Iteration (API)

Approximative Werte-Iteration (AVI)

Approximative bzw. „gefittete“ Werte-Iteration geht den naheliegenden Weg und wählt zunächst eine kleine Zahl von N repräsentativen Zuständen $s_1, \ldots, s_N$ aus allen möglichen aus. Im $(k+1)$-ten Schritt der Iteration wird dann der Gewichtsvektor $\mathbf{w}_{k+1}$ einer parametrisierten Wertefunktion der Form $\tilde{V}(\cdot\,;\mathbf{w})$ so bestimmt, daß (zumindest für die repräsentativen Zustände)

$$\begin{aligned}\tilde{V}(s_i;\mathbf{w}_{k+1}) &\approx \big(\mathcal{T}_*\tilde{V}(\cdot\,;\mathbf{w}_k)\big)(s_i)\\ &= \max_{a\in\mathcal{A}} \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s_i,a)}\left\{\mathrm{R}(s'|s_i,a)+\gamma\tilde{V}(s';\mathbf{w}_k)\right\}, \quad i=1\ldots N\end{aligned}$$

gilt (wobei der Erwartungswert leicht durch Stichproben genähert werden kann). In jeder Iteration k ist somit ein Least-Squares Problem zu lösen:

$$\mathbf{w}_{k+1} = \operatorname*{argmin}_{\mathbf{w}} \left\{\sum_{i=1}^{N}\Big(\tilde{V}(s_i;\mathbf{w}) - \big(\mathcal{T}_*\tilde{V}(\cdot\,;\mathbf{w}_k)\big)(s_i)\Big)^2\right\},$$

die Lösung solcher Regressionsaufgaben werden wir noch ausführlicher in Kapitel 2 diskutieren.

Die gefittete Werte-Iteration selbst soll hier aber nur der Vollständigkeit halber Erwähnung finden. In der Theorie kann gegenwärtig nur gezeigt werden, daß sie für eine bestimmte Klasse von Funktionsapproximatoren, den sog. 'Averagern[4]' konvergiert (Gordon, 1995). Für andere Approximationsarchitekturen, wie den hier betrachteten linear parametrisierten, ist das im allgemeinen nicht der Fall und es lassen sich auch Gegenbeispiele für Divergenz konstruieren. Zwar *kann* in der Praxis das Verfahren durchaus auch mit anderen Architekturen funktionieren, unsere eigenen anfänglichen Experimente in diese Richtung lieferten aber keine zufriedenstellende Ergebnisse, weswegen wir diesen Ansatz in der Arbeit nicht weiter verfolgen wollen.

[4] Averagers sind Methoden, bei denen die Approximation $f(\mathbf{x};\mathbf{w})$ als gewichtete Summe der Soll-Werte geschrieben werden kann, die Gewichte nicht-negativ sind und sich auf Eins summieren (d.h. für Daten $\{\mathbf{x}_i, y_i\}_{i=1}^N$ ist $f(\mathbf{x};\mathbf{w}) = \sum_{i=1}^N w_i(\mathbf{x})y_i$ mit $w_i \geq 0$ und $\sum_{i=1}^N w_i(\mathbf{x}) = 1$, $\forall\mathbf{x}$). Zur Klasse der Averagers gehören die instanzbasierten Methoden wie Nearest Neighbour oder Kernel Smoothing, nicht aber die mächtigeren Basisfunktionsnetzwerke.

Approximative Politik-Iteration (API)

Wie zuvor operieren wir auch bei approximativer Politik-Iteration auf Politiken und führen abwechselnd die Schritte Politik-Evaluation und Politik-Verbesserung aus, mit der Maßgabe, nach und nach immer bessere Politiken zu produzieren (siehe Abbildung 1.7). Den Schritt Politik-Evaluation für eine Politik π_k können wir allerdings nicht exakt ausführen, weil wir erstens die Wertefunktion nicht exakt, sondern nur durch eine Approximation $\tilde{V}(\cdot\,;\mathbf{w}_k)$ darstellen und weil wir zweitens die Anwendung des Bellman-Operators $\mathcal{T}_{\pi_k}$ nur mit Hilfe einer endlichen Anzahl T simulierter Zustandsübergänge nachahmen können. Die Zustandsübergänge werden unter der aktuell betrachteten Politik π_k erzeugt und bestehen aus Tupeln von Zuständen s_{t+1} und erhaltener Belohnung r_{t+1}:

$$s_t \xrightarrow[r_{t+1}]{\pi_k(s_t)} s_{t+1} \xrightarrow[r_{t+2}]{\pi_k(s_{t+1})} s_{t+2} \longrightarrow \cdots$$

wobei $s_{t+1} \sim \mathrm{P}(\cdot\,|s_t, \pi_k(s_t))$ und $r_{t+1} = \mathrm{R}(s_{t+1}|s_t, \pi_k(s_t))$ für alle $t = 0, 1, 2, \ldots$. Zur Berechnung der Aktionen $\pi_k(s_t)$ ist es nicht notwendig, auch die Abbildung π_k durch einen eigenen Funktionsapproximator zu repräsentieren, weil π_k die gierige (deterministische) Politik bezüglich $\tilde{V}(\cdot\,;\mathbf{w}_{k-1})$ ist und bei einer endlichen (kleinen) Menge von Aktionen punktweise ausgewertet werden kann. Die Kenntnis des alten Gewichtsvektors $\mathbf{w}_{k-1}$ reicht hierfür aus (strenggenommen benötigen wir im Augenblick dazu auch noch ein Modell, doch dieses Problem werden wir später mit der erweiterten Q-Notation beseitigen).

Die zur Verfügung stehenden Daten $\{(s_{t-1}, s_t, r_t)\}_{t=1}^{T}$ werden dann an ein spezielles Verfahren weitergereicht, das aus diesen Daten einen neuen Gewichtsvektor $\mathbf{w}_k$ produziert, so daß am Ende $\tilde{V}(\cdot\,;\mathbf{w}_k) \approx V^{\pi_k}(\cdot)$ gilt, siehe dazu auch Abbildung 1.7. Ein Verfahren das so etwas leistet ist *Temporal-Difference Lernen*, welches wir im folgenden Abschnitt näher erläutern werden.

Den Schritt Politik-Verbesserung können wir auch nicht exakt ausführen, weil bei einer gierigen Politik-Ableitung aus einer Approximation $\tilde{V}(\cdot\,;\mathbf{w}_k)$ nicht mehr länger gewährleistet ist, daß π_{k+1} tatsächlich eine bessere Politik ist. Häufig läßt sich der Effekt „Chattering“ beobachten, bei dem die aktuelle Politik um die optimale oszilliert. Nichtsdestotrotz handelt es sich bei API um eine robuste Vorgehensweise; der Fehler zwischen der von API produzierten Approximation und der wirklichen optimalen Wertefunktion ist nach Bertsekas und Tsitsiklis (1996) in der $\mathcal{L}_\infty$-Norm beschränkt. So gilt beispielsweise (mit der Identifikation $\tilde{V}^{\pi_k} := \tilde{V}(\cdot\,;\mathbf{w}_k)$)

$$\limsup_{k\to\infty} \left\| \tilde{V}^{\pi_k} - V^* \right\|_\infty \leq \frac{2\gamma\varepsilon}{(1-\gamma)^2}$$

falls für alle Iterationen k der Fehler in der Approximation durch

$$\forall k = 0, 1, 2, \ldots \qquad \left\| \tilde{V}^{\pi_k} - V^{\pi_k} \right\|_\infty \leq \varepsilon$$

beschränkt ist. Neuere Ergebnisse dieser Art findet man in (Munos, 2003) und anderswo. Für uns entscheidend ist, daß vor allem in der Praxis API oft mit erstaunlich wenigen Iterationen bereits eine gute Politik liefert.

1.3 Reinforcement Lernen

Unter Reinforcement Lernen (RL) versteht man eine spezielle Architektur zum näherungsweisen Bestimmen einer optimalen Politik im Rahmen approximativen Dynamischem Programmieren. Die Bezeichnung „Reinforcement Lernen“ (Lernen durch Belohnen/Bestrafen) weckt dabei nicht von ungefähr Assoziationen mit natürlichem Lernen, viele der zugrundeliegenden Methoden entspringen der Nachbildung von Lernprozessen, die bei Tieren (oder Menschen) beobachtet werden konnten. Eine grundlegende Eigenschaft von RL ist, daß das optimale Verhalten von der Maschine (Agent) erworben werden soll, *während* sie mit dem System (d.h. in der Welt) real interagiert. Anpassungen am Gewichtsvektor, Verbesserung der Politik und das Sammeln von Daten (Stichproben von Zustandsübergängen) müssen daher in einen geschlossenen Kreislauf integriert werden. Zudem spielt auch die Laufzeit eine kritische Rolle, wir benötigen ein Verfahren, das die notwendigen Berechnungen in Echtzeit vornehmen kann.

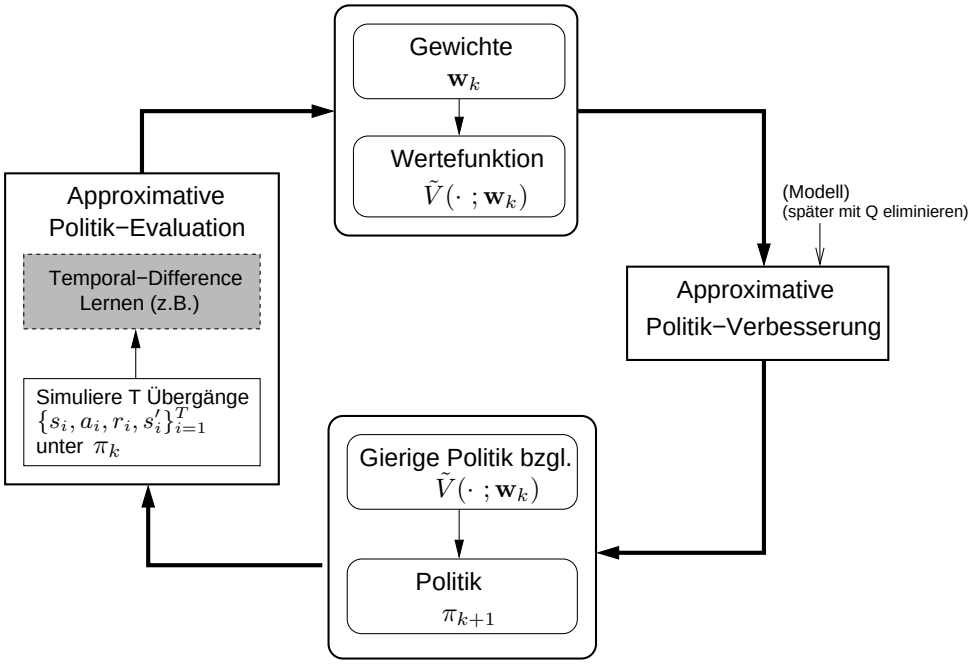


Abbildung 1.7: Approximative Politik-Iteration iteriert approximative Politik-Evaluation und approximative Politik-Verbesserung um immer bessere Politiken π_k zu produzieren (wofür es keine theoretische Gewähr mehr gibt, was aber in der Praxis dennoch gut zu funktionieren scheint).

1.3.1 Temporal-Difference Lernen für approximative Politik-Evaluation

Den Motor von RL bilden die Temporal-Difference Verfahren zur Implementation der approximativen Politik-Evaluation (APE). Der populärste Vertreter dieser Art ist TD(λ) von Sutton (1988). Dieses basiert auf gemittelten n-Schritt Schätzungen für die erwartete Gesamtauszahlung und einer linear parametrisierten Wertefunktion. TD(λ) ist ein iteratives Verfahren, welches bei jedem beobachteten Zustandsübergang für $t = 0, 1, 2, \ldots$ eine neue Approximation $\tilde{V}(\cdot\,;\mathbf{w}_k^t)$ für $V^{\pi_k}(\cdot)$ produziert. Zudem ist es ein Online-Verfahren, das bei dieser Operation nur den jeweils aktuellen Zustandsübergang berücksichtigt (vergleichbar mit den Least-Mean-Squares Varianten von Gradientenabstiegsverfahren im Bereich der Adaptiven Filter). Das TD(λ) Verfahren ist technisch sehr leicht zu implementieren. Es ist aber relativ aufwendig, auch anschaulich darzustellen und zu analysieren, weswegen wir an dieser Stelle nur den Algorithmus selbst angeben wollen. Für weitergehende Erläuterungen verweisen wir auf die Literatur (Bertsekas und Tsitsiklis, 1996; Tsitsiklis und Van Roy, 1997; Sutton und Barto, 1998).

Wie zuvor nehmen wir an, daß die Wertefunktion durch

$$\tilde{V}(\cdot\,;\mathbf{w}) = \sum_{i=1}^{m} w_i \phi_i(\cdot)$$

mit einem Gewichtsvektor $\mathbf{w}$ parametrisiert ist. Sei $\tilde{V}(\cdot\,;\mathbf{w}_k^t)$ die Approximation für V^{π_k} nachdem t Zustandsübergänge $s_0, s_1, s_2, \ldots, s_t$ mit Belohnungen $r_1, r_2, \ldots, r_t$ unter π_k beobachtet wurden. Bei jedem Zeitschritt i wählen wir dabei die Aktion $\pi_k(s_i)$ (die gierige Aktion bzgl. $\tilde{V}(\cdot\,;\mathbf{w}_{k-1})$) und gehen in den Zustand s_{i+1} mit Belohnung $\mathrm{R}(s_{i+1}|s_i, \pi_k(s_i))$ über.

Der grundlegende Bestandteil ist ein Korrekturterm, der sog. Temporal-Difference Term, der allgemein durch

$$d(s_i, s_{i+1}; \mathbf{w}) = \mathrm{R}(s_{i+1}|s_i, \pi(s_i)) + \gamma \tilde{V}(s_{i+1}; \mathbf{w}) - \tilde{V}(s_i; \mathbf{w})$$

definiert ist und in gewisser Weise einer Näherung für das Bellmanresiduum, der Differenz zwischen linker und rechter Seite in der Bellmangleichung (1.2), entspricht. Weiter betrachten wir einen sogenannten

Erinnerungs- oder Aktivierungsvektor („eligibility traces“) $\mathbf{z}_t$, der durch

$$\mathbf{z}_t := \sum_{i=0}^{t} (\gamma\lambda)^{t-i} \ \nabla_{\mathbf{w}} \tilde{V}(s_i; \mathbf{w}_k^t)$$

definiert ist und hier, weil $\nabla_{\mathbf{w}} \tilde{V}(s_i; \mathbf{w}_k^t) = \boldsymbol{\phi}(s_i)$ ist, als

$$\mathbf{z}_t = \sum_{i=0}^{t} (\gamma\lambda)^{t-i} \boldsymbol{\phi}(s_i)$$

geschrieben werden kann, wobei wie zuvor $\boldsymbol{\phi}(s_i) = \big(\phi_1(s_i), \ldots, \phi_m(s_i)\big)^{\mathsf{T}}$ ist. Der Parameter $\lambda \in [0, 1]$ steuert die Gewichtung der einzelnen n-Schritt Schätzungen für die erwartete Gesamtauszahlung und ist das namensgebende Lambda in TD(λ). Es ist bekannt, daß TD(λ) für die „richtige“ Wahl von λ besser als die Extremfälle TD(0) (entspricht ungefähr asynchroner Werte-Iteration) und TD(1) (entspricht Monte-Carlo Verfahren) funktionieren kann.

Der TD(λ) Algorithmus funktioniert nun wie folgt: bei jedem Zustandsübergang von s_t nach s_{t+1} unter $\pi_k(s_t)$ mit r_{t+1} berechnen wir zunächst den Temporal-Difference Term

$$d_t := d(s_t, s_{t+1}; \mathbf{w}_k^t) = \mathrm{R}(s_{t+1}|s_t, \pi_k(s_t)) + \gamma \tilde{V}(s_{t+1}; \mathbf{w}_k^t) - \tilde{V}(s_t; \mathbf{w}_k^t) \tag{1.4}$$

und aktualisieren dann zuerst den Gewichtsvektor (eine Form von Gradientenabstieg)

$$\mathbf{w}_k^{t+1} = \mathbf{w}_k^t + \alpha d_t \mathbf{z}_t \tag{1.5}$$

und anschließend den Aktivierungsvektor

$$\mathbf{z}_{t+1} = (\gamma\lambda)\mathbf{z}_t + \boldsymbol{\phi}(s_{t+1}) \tag{1.6}$$

wobei α ein Schrittweitenparameter (Lernrate) ist. Der Algorithmus wird mit einer anfänglichen Belegung für $\mathbf{w}_k^0$ und mit $\mathbf{z}_0 \equiv 0$ gestartet. Die Folge der Gewichtsvektoren $\mathbf{w}_k^1, \mathbf{w}_k^2, \ldots$ konvergiert gegen einen Gewichtsvektor $\mathbf{w}_k^*$, so daß $\tilde{V}(\cdot\ ; \mathbf{w}_k^*) \approx V^{\pi_k}(\cdot)$ ist (in gewissem Sinne und unter einer Reihe weiterer technischer Voraussetzungen, siehe Tsitsiklis und Van Roy, 1997).

Der Rechen- und Speicheraufwand für TD(λ) ist sehr gering; der Speicherbedarf ist insgesamt $\mathcal{O}(m)$ und der notwendige Rechenaufwand für (1.4), (1.5), (1.6) beträgt pro beobachtetem Zustandsübergang ebenfalls nur $\mathcal{O}(m)$ Operationen. Für spezielle Basisfunktionen mit kompaktem Träger kann der Aufwand noch erheblich weiter reduziert werden; so ist beispielsweise bei B-Splines 1. Ordnung immer nur eine Basisfunktion aktiv (ungleich Null), d.h. der Aufwand reduziert sich hier auf $\mathcal{O}(1)$.

1.3.2 Optimistische Politik-Iteration für optimale Steuerung

Optimistische Politik-Iteration (OPI)

Bei Politik-Iteration im klassischen Sinne würden wir den Schritt Politik-Evaluation vollständig bis zur Konvergenz der Wertefunktion ausführen (bzw. der Gewichtsvektoren $\mathbf{w}_k^t$ gegen $\mathbf{w}_k^*$) und erst im Anschluß eine neue verbesserte Politik ableiten. Auf diese Weise gehen wir allgemein aber recht verschwenderisch mit den gesammelten Daten um, denn wir halten auch dann noch an einer Politik fest, wenn schon längst absehbar geworden ist, daß sie verbessert werden muß. Vor allem in Anbetracht der Tatsache, daß jeder Zustandsübergang von einem Agenten teuer produziert werden muß, sollte es unser vorrangigstes Ziel sein, eine möglichst schnelle Konvergenz der Politik zur optimalen herbeizuführen.

Bei der optimistischen Politik-Iteration zögern wir daher die Politik-Verbesserung nicht mehr bis zur vollständigen Konvergenz der Politik-Evaluation hinaus, sondern beginnen bereits nach wenigen Evaluationsschritten mit einer Verbesserung (Bertsekas und Tsitsiklis, 1996). Die Idee ist informell die folgende: wenn bei Politik-Evaluation die Approximationen $\tilde{V}(\cdot\ ; \mathbf{w}_k^{t_1}), \tilde{V}(\cdot\ ; \mathbf{w}_k^{t_2}), \ldots$ für $t \to \infty$ gegen $\tilde{V}(\cdot\ ; \mathbf{w}_k^*)$ als Näherung für $V^{\pi_k}(\cdot)$ konvergieren, und gierige Politik-Ableitung aus V^{π_k} zu einer besseren Politik führt,

dann könnte doch auch bereits die Politik-Ableitung aus den einzelnen Iterierten $\tilde{V}(\cdot\,;\mathbf{w}_k^{t_1}), \tilde{V}(\cdot\,;\mathbf{w}_k^{t_2}), \ldots$ zu einer verbesserten Politik führen. Eine theoretische Absicherung dieser Vorgehensweise ist uns gegenwärtig nicht bekannt, aber in der Praxis funktioniert optimistische Politik-Iteration meist tadellos. Häufig wird die Politik-Verbesserung auch bereits nach jedem beobachteten Zustandsübergang durchgeführt, was möglich ist, weil TD(λ) ein iteratives Verfahren ist und der anfängliche Gewichtsvektor $\mathbf{w}_k^0 \equiv \mathbf{w}_{k-1}$ mit dem Gewichtsvektor $\mathbf{w}_{k-1}$ des vorangegangenen Politik-Evaluation Schrittes initialisiert werden kann.

Die Q-Funktion zur modellfreien Politik-Verbesserung

Um aus einer (parametrisierten) Wertefunktion V die gegenwärtig beste Aktion in einem Zustand s zu ermitteln, ist es notwendig

$$\operatorname*{argmax}_{a\in\mathcal{A}} \; \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V(s') \right\}$$

auszurechnen, womit wir aber noch immer vor dem Problem stehen, einen Erwartungswert bestimmen zu müssen. Mit einem sehr eleganten Ansatz von Watkins (1989) wird es möglich, auch diese Erwartungswertberechnung komplett zu umgehen. Hierzu wird eine erweiterte Notation eingeführt, bei der die Wertefunktion nicht mehr auf Zuständen, sondern auf Zustands-Aktionspaaren operiert.

Analog zur Definition der Wertefunktion $V^\pi : \mathcal{S} \to \mathbb{R}$ für Zustände $s \in \mathcal{S}$ unter Politik π in Gleichung (1.1) definieren wir nun die Q-Funktion $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ für Zustands-Aktionspaare (s,a) durch

$$Q^\pi(s,a) := \mathbb{E}\left\{ \mathrm{R}(s_1|s_0,a) + \sum_{t\geq 1} \gamma^t \mathrm{R}(s_{t+1}|s_t,\pi(s_t)) \;\Big|\; s_0 = s \right\} \quad \forall s,a \tag{1.7}$$

wobei $s_1 \sim \mathrm{P}(\cdot\,|s_0,a)$ und $s_{t+1} \sim \mathrm{P}(\cdot\,|s_t,\pi(s_t))$ für $t = 1,2,\ldots$. Die Q-Funktion erfüllt eine zur Bellmangleichung (1.2) analoge Fixpunktbeziehung

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V^\pi(s') \right\} \\ &= \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma Q^\pi(s',\pi(s')) \right\} \quad \forall s,a \end{aligned} \tag{1.8}$$

Der entscheidende Vorteil der Q-Funktion ist, daß sich aus ihr sehr leicht die beste Aktion ableiten läßt, denn

$$\operatorname*{argmax}_{a\in\mathcal{A}} \; \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V^\pi(s') \right\} = \operatorname*{argmax}_{a\in\mathcal{A}} \; Q^\pi(s,a).$$

Die optimale Q-Funktion Q^* wird definiert als

$$Q^*(s,a) := \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V^*(s') \right\} \quad \forall s,a.$$

Nach der Bellman-Optimalitätsgleichung für V^* gilt $V^* = \mathcal{T}_* V^*$ und daher ist

$$\begin{aligned} V^*(s) &= \max_{a\in\mathcal{A}} \; \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V^*(s') \right\} \\ &= \max_{a\in\mathcal{A}} \; Q^*(s,a) \quad \forall s. \end{aligned}$$

Es folgt, daß auch Q^* eine Form der Bellman-Optimalitätsgleichung erfüllt, nämlich

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma V^*(s') \right\} \\ &= \mathbb{E}_{s'\sim \mathrm{P}(\cdot\,|s,a)} \left\{ \mathrm{R}(s'|s,a) + \gamma \max_{a'\in\mathcal{A}} \; Q^*(s',a') \right\}. \end{aligned}$$

Der Nachteil der Q-Funktion gegenüber der V-Funktion ist, daß sich die Anzahl der zu bestimmenden Werte erhöht. War bei exakter Politik-Evaluation im Falle von V^π noch ein Gleichungssystem in $|\mathcal{S}|$ Unbekannten zu lösen, so ist nun für Q^π ein Gleichungssystem in $|\mathcal{S}| \times |\mathcal{A}|$ Unbekannten zu lösen. Ähnlich verhält es sich, wenn wir Approximationen für Q betrachten:

Approximation der Q-Funktion

Auch zur Bestimmung von Q^π im Falle großer/kontinuierlicher Zustandsräume können wir nur wieder eine parametrisierte Approximation heranziehen, dergestalt daß

$$\tilde{Q}(s,a;\mathbf{w}) \approx Q^\pi(s,a)$$

für eine Belegung des Parametervektors $\mathbf{w}$ gilt. Wie zuvor betrachten wir linear parametrisierte Modelle, d.h.

$$\tilde{Q}(s,a;\mathbf{w}) = \sum_{i=1}^{m} w_i \phi_i(s,a)$$

wobei dieses mal die m Basisfunktionen $\phi_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ über Zuständen und Aktionen definiert sein müssen.

Das Standardverfahren: Sarsa(λ)=TD(λ) + OPI + Q-Funktion

Das Verfahren TD(λ) zur approximativen Politik-Evaluation läßt sich leicht auch auf Q-Funktionen übertragen, so daß wir bei jedem Schritt Iterierte $\tilde{Q}(\cdot\,;\mathbf{w}_k^t)$ produzieren, die gegen $\tilde{Q}(\cdot\,;\mathbf{w}_k^*)$ als Näherung für Q^{π_k} konvergieren. Insbesondere läßt sich dank der Q-Funktion auch leicht die extreme Form von optimistischer Politik-Iteration umsetzen, bei der auf jeden Evaluationsschritt unmittelbar eine Politik-Verbesserung folgt. Gewissermaßen lösen wir auf diese Weise das Problem der optimalen Steuerung direkt, ohne den expliziten Umweg Politik-Evaluation und -Verbesserung zu gehen. Diese spezielle Variante zum näherungsweisen Bestimmen der optimalen Politik wird manchmal auch Sarsa(λ) genannt und ist gegenwärtig das bekannteste und am weitesten verbreitete RL-Verfahren, siehe Sutton und Barto (1998).

Der Sarsa(λ) Algorithmus funktioniert folgendermaßen: nach t Zustandsübergängen sei $\tilde{Q}(\cdot\,;\mathbf{w}^t)$ die gegenwärtige Wertefunktion. Der lernende Agent führt Aktion a_t aus (wobei a_t die im vorausgegangenen Iterationsschritt ermittelte Aktion $a_t = \text{argmax}_a\ \tilde{Q}(s_t,a;\mathbf{w}^{t-1})$ ist) und beobachtet den nachfolgenden Zustand s_{t+1} sowie Belohnung $r_{t+1} = \mathrm{R}(s_{t+1}|s_t,a_t)$. Wir ermitteln nun die beste Aktion in s_{t+1} (Politik-Verbesserung)

$$a_{t+1} = \operatorname*{argmax}_a\ \tilde{Q}(s_{t+1},a;\mathbf{w}^t) \tag{1.9}$$

und berechnen den TD-Term

$$d_t := d([s_t,a_t],[s_{t+1},a_{t+1}];\mathbf{w}^t) = r_{t+1} + \gamma\tilde{Q}(s_{t+1},a_{t+1};\mathbf{w}^t) - \tilde{Q}(s_t,a_t;\mathbf{w}^t).$$

Anschließend aktualisieren wir den Gewichtsvektor

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \alpha d_t \mathbf{z}_t \tag{1.10}$$

und den Aktivierungsvektor

$$\mathbf{z}_{t+1} = (\lambda\gamma)\mathbf{z}_t + \boldsymbol{\phi}(s_{t+1},a_{t+1}),$$

wobei $\boldsymbol{\phi}(s_{t+1},a_{t+1}) = \big(\phi_1(s_{t+1},a_{t+1}),\ldots,\phi_m(s_{t+1},a_{t+1})\big)^\mathsf{T}$. Sarsa($\lambda$) ist ein On-Policy Verfahren, bei dem die Zustandsübergänge (Stichproben) unter der zu evaluierenden Politik generiert werden.

Exploration durch ε-gierige Aktionswahl

Um sicherzustellen, daß in den beobachteten Zuständen auch verschiedene Handlungsalternativen ausprobiert werden, und um zu verhindern, daß aufgrund der nur näherungsweisen durchführbaren Politik-Verbesserung die Politik zwischen zwei oder mehr nicht-optimalen Politiken oszilliert (d.h. der Agent steckenbleibt) wird in der Praxis die Politik mit einer stochastischen Komponente versehen. Bei der ε-gierigen Aktionswahl wird beispielsweise mit einer kleinen Wahrscheinlichkeit ε nicht die (subjektiv) beste, sondern eine zufällige Aktion ausgeführt.

Beispiel Mountain-Car

An dieser Stelle wollen wir nun demonstrieren, wie RL in der Praxis eingesetzt wird und wie für das Mountain-Car Beispiel (vgl. Seite 2) mit dem Standard-Verfahren Sarsa(λ) und Tilecoding (jede der 3 Aktionen wird durch ein Gitter mit 9 Schichten bestehend aus jeweils 9×9 Zellen diskretisiert) ein optimales Verhalten erlernt werden kann. Dazu verfahren wir nach immer demselben Schema:

Die Aufgabe wird als ein episodisches Lernproblem formuliert, bei dem der Agent in jeder einzelnen Epsiode in einem (zufälligen) Anfangszustand startet und solange Aktionen ausführt, bis er den Terminalzustand erreicht oder ein anderes Abbruchkriterium erfüllt ist. In jedem einzelnen Schritt einer Episode beobachtet der Agent den aktuellen Zustand, wählt seine Aktion gemäß (1.9) gierig bzgl. der gegenwärtigen Approximation für Q aus (ε-gierig), beobachtet den folgenden Zustand sowie Belohnung und modifiziert die Gewichte der Approximation gemäß der Sarsa(λ) Lernregel (1.10). Zu beachten ist, daß, typisch für RL, die vergebene Belohnung nur unmittelbaren Erfolg bewertet und ansonsten keine Hilfestellung zur Wahl 'richtiger' Aktionen liefert; sie ist 0 bei Erreichen des Zielzustands und -1 bei jedem anderen Schritt.

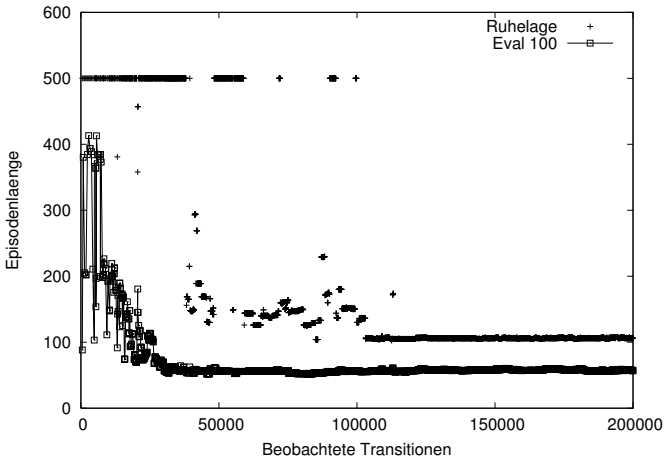
Zum Messen des Lernerfolgs wird zusätzlich nach jeder Episode die Performanz der gegenwärtig gelernten Q-Funktion ermittelt. Dazu wird eine Anzahl von Testepisoden durchgeführt, in denen der Agent nicht weiter lernt, sondern nur der aktuell gierigen Politik folgt. Abbildung 1.8 (oben) zeigt den Lernerfolg abhängig von der Zeit (Anzahl zum Lernen verwendeter Zustandsübergänge). Gezeigt werden zwei Graphen, erstens die Perfomanz beim Starten aus der Ruhelage (schwieriger) und zweitens die mittlere Performanz aus einer Menge von 100 festen Startzuständen. Es zeigt sich das typische Lernverhalten: zu Beginn sind die Episoden maximal lang (eine nach 500 Schritten erfolglose Episode wird abgebrochen) und der Agent irrt zunächst ahnungslos umher. Aber schon nach wenigen Episoden ist er in der Lage, von beinahe jeder Anfangsposition aus das Ziel zuverlässig anzusteuern, nach ungefähr 40000 Übergängen tritt keine merkliche Verbesserung mehr ein. Für die etwas schwierigere Ruhelage braucht der Agent etwas länger, hier wird erst nach ungefähr 100000 Übergängen das bestmögliche Ergebnis erreicht. Abbildung 1.8 (mitte) zeigt dann die am Ende gelernte Wertefunktion sowie zum visuellen Vergleich (unten) auch die tatsächliche optimale Wertefunktion, näherungsweise berechnet mit gefitteter Werteiteration über einem hochauflösenden Gitter.

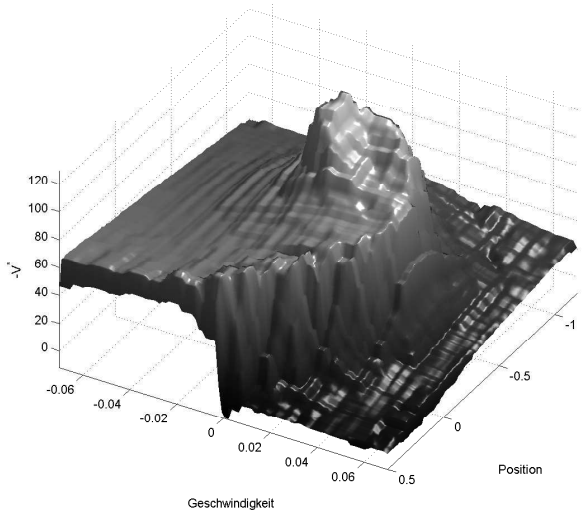
1.4 Approximative Politik-Evaluation mit Least-Squares Methoden

Das bislang vorgestellte RL-Verfahren TD(λ) ist ein Online Verfahren, das auf einer stochastischen Form von Gradientenabstieg basiert und nur die jeweils aktuelle Beobachtung heranzieht, um Anpassungen am Gewichtsvektor vorzunehmen. Der Vorteil ist, wie bereits erwähnt wurde, daß dabei pro Schritt nur geringer Rechenaufwand betrieben werden muß und wir algorithmisch eine einfache „Aktualisierungsvorschrift“ angeben können. Der Nachteil dabei ist, daß

1. TD(λ) ein reichlich ineffizientes Verfahren ist, was die Geschwindigkeit der Konvergenz im Verhältnis zur Anzahl der beobachteten Zustandsübergänge angeht
2. TD(λ) effektiv nur mit einer limitierten Klasse von Funktionsapproximatoren betrieben werden kann, bei der die Parametrisierung, d.h. die Basisfunktionen, *vor* dem Lernen festgelegt werden müssen. Alle konstruktiven Methoden, die die Form der Approximation abhängig vom Approximationsfehler adaptiv gestalten, sind bei dieser Art des Lernens nicht möglich.

An dieser Stelle wollen wir daher eine alternative Herangehensweise an das Problem APE betrachten, bei der die Aufgabe als ein klassisches Least-Squares Problem formuliert wird und für das wir eine geschlossene Lösung berechnen können (Bradtke und Barto, 1996; Boyan, 1999; Lagoudakis und Parr, 2003; Nedić und Bertsekas, 2003). Damit können alle beobachteten Daten zugleich einbezogen werden und wir eine weitaus schnellere Konvergenz erzielen. Indem wir das Problem APE auf statistische Regression zurückführen, eröffnet sich uns insbesondere aber auch (und das ist fast noch wichtiger) die Möglichkeit,





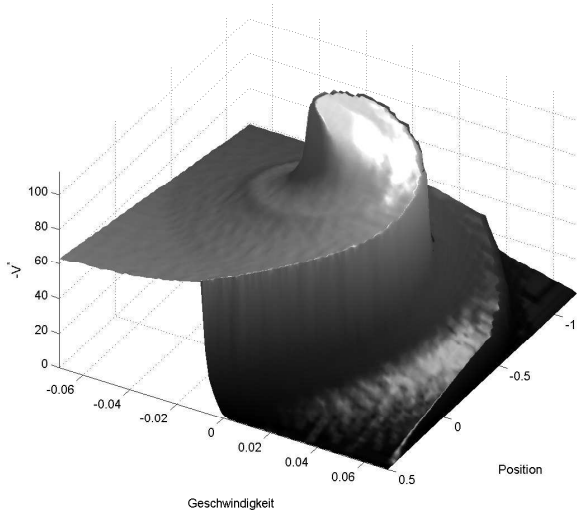


Abbildung 1.8: Reinforcement Lernen für das Mountain-Car Problem. Oben: Qualität der gelernten Lösung, abhängig von der Anzahl der beobachteten Daten. Das Ziel ist es, eine Politik zu lernen, die die Weglänge/Episodenlänge minimiert. Der Plot zeigt die Performanz der jeweils gierigen Politik, ermittelt nach jeder Lernepisode, bei zwei Startkonfigurationen: einmal für das Starten aus der Ruhelage (schwieriger) und zweitens die mittlere Performanz beim Starten aus einer Menge von 100 zufälligen, festgewählten Zuständen. Mitte: Approximation von V^* nach 200000 Zustandsübergängen mit Sarsa(λ) und Tilecoding. Unten: Approximation von V^* durch gefittete Werteiteration auf einem 200$\times$ 200 Gitter (3-Nearest Neighbour Funktionsapproximation).

auf die gesamte Bandbreite linearer statistischer Modelle zurückzugreifen und fehleradaptive Methoden zu verwenden.

Für die folgenden Erläuterungen nehmen wir der Einfachheit halber wieder einen endlichen Zustandsraum $\mathcal{S} = \{1, .., N\}$ an. Unser Ziel ist es, die Wertefunktion V^π zu einer gegebenen Politik π auszurechnen (das Gesagte würde in gleicher Weise auch für die Q-Funktion Q^π gelten). Für einen endlichen Zustandsraum $\mathcal{S}$ kann jede Wertefunktion V als $N \times 1$ Vektor geschrieben werden: $\mathbf{v} = \big(V(1), \ldots, V(N)\big)^\mathsf{T} \in \mathbb{R}^N$, die gesuchte Wertefunktion ist der Vektor $\mathbf{v}^\pi = \big(V^\pi(1), \ldots, V^\pi(N)\big)^\mathsf{T} \in \mathbb{R}^N$. Definiere die $N \times N$ Matrix $\mathbf{P}$ als Transitionsmatrix $[\mathbf{P}]_{ij} = \mathrm{P}(j|i, \pi(i))$ und den $N \times 1$ Vektor $\bar{\mathbf{R}}$ als Vektor der erwarteten Belohnungen, d.h. $[\bar{\mathbf{R}}]_i = \sum_j \mathrm{P}(j|i, \pi(i))\mathrm{R}(j|i, \pi(i))$. Den Bellmanoperator $\mathcal{T}_\pi : \mathbb{R}^N \to \mathbb{R}^N$ können wir dann als affine Abbildung $\mathcal{T}_\pi(\mathbf{v}) := \bar{\mathbf{R}} + \gamma\mathbf{P}\mathbf{v}$ schreiben.

1.4.1 Mit Modell

Wir betrachten zunächst den Fall, daß $\mathbf{P}$ und $\bar{\mathbf{R}}$ bekannt sind.

Exakte Bestimmung von V^π

Aufgrund der Bellmangleichung (1.2) wissen wir, daß das gesuchte V^π ein Fixpunkt unter $\mathcal{T}_\pi$ sein muß. Unter Verwendung obiger Bezeichnungen bedeutet das, daß der Vektor $\mathbf{v}^\pi$ Lösung des Gleichungssystems $\mathbf{v} = \mathcal{T}_\pi(\mathbf{v})$, oder ausgeschrieben

$$\mathbf{v} = \bar{\mathbf{R}} + \gamma\mathbf{P}\mathbf{v}, \tag{1.11}$$

in den N Unbekannten $\mathbf{v}$ ist. Somit folgt $\mathbf{v}^\pi = (\mathbf{I} - \gamma\mathbf{P})^{-1}\bar{\mathbf{R}}$ (wobei die Inverse existiert, wenn $\gamma < 1$).

Repräsentation von V durch lineare Funktionsapproximation

Jetzt nehmen wir wieder an, die Wertefunktionen V werden durch eine lineare Parametrisierung $\tilde{V}(\cdot\,; \mathbf{w})$ repräsentiert (i.a. $m < N$). Sei der $N \times 1$ Vektor $\tilde{\mathbf{v}} = \mathbf{\Phi}\mathbf{w}$ eine solche Wertefunktion über $\mathcal{S}$, wobei

$$\tilde{\mathbf{v}} = \begin{bmatrix} \tilde{V}(1; \mathbf{w}) \\ \vdots \\ \tilde{V}(N; \mathbf{w}) \end{bmatrix} \qquad \text{und} \qquad \mathbf{\Phi} = \begin{bmatrix} \text{—}\ \boldsymbol{\phi}(1)^\mathsf{T}\ \text{—} \\ \vdots \\ \text{—}\ \boldsymbol{\phi}(N)^\mathsf{T}\ \text{—} \end{bmatrix}$$

mit $\boldsymbol{\phi}(i) = \big(\phi_1(i), \ldots, \phi_m(i)\big)^\mathsf{T}$ und $\mathbf{w} = \big(w_1, \ldots, w_m\big)^\mathsf{T}$. Die m Basisfunktionen ϕ_i, d.h. die Spalten von $\mathbf{\Phi}$, seien linear unabhängig, so daß $\mathrm{Rang}(\mathbf{\Phi}) = m$ und der Bildraum $\mathrm{Bild}(\mathbf{\Phi}) := \{\mathbf{\Phi}\mathbf{w} \,|\, \mathbf{w} \in \mathbb{R}^m\}$, d.h. die Menge aller durch diese Parametrisierung darstellbaren Wertefunktionen, die Dimension m hat. Ist $m < N$, dann ist $\mathrm{Bild}(\mathbf{\Phi})$ ein echter Unterraum des $\mathbb{R}^N$ und im allgemeinen wird das echte $\mathbf{v}^\pi$ außerhalb von $\mathrm{Bild}(\mathbf{\Phi})$ liegen, d.h. $\mathbf{v}^\pi \notin \mathrm{Bild}(\mathbf{\Phi})$. Das Ziel ist es, $\tilde{\mathbf{v}} \in \mathrm{Bild}(\mathbf{\Phi})$ (d.h. den zugehörigen Gewichtsvektor $\mathbf{w}$) so zu bestimmen, daß $\tilde{\mathbf{v}} \approx \mathbf{v}^\pi$ gilt. Die beste (mit minimalem quadratischen Fehler) Approximation an $\mathbf{v}^\pi$ aus $\mathrm{Bild}(\mathbf{\Phi})$ kann dazu nicht genommen werden, weil $\mathbf{v}^\pi$ unbekannt ist. Stattdessen müssen wir auf irgendeine Weise die Information ausnutzen, daß das zu approximierende $\mathbf{v}^\pi$ ein Fixpunkt unter $\mathcal{T}_\pi$ ist. Naheliegend wären die folgenden zwei Möglichkeiten (siehe auch Abbildung 1.9):

1. Möglichkeit: Minimierung des Bellman-Residuums (BRM)

Ersetzen wir in (1.11) $\mathbf{v}$ durch $\tilde{\mathbf{v}}$, dann soll $\tilde{\mathbf{v}} = \mathcal{T}_\pi(\tilde{\mathbf{v}})$ bzw. $\mathbf{\Phi}\mathbf{w} = \mathcal{T}_\pi(\mathbf{\Phi}\mathbf{w})$ sein. Für den gesuchten Gewichtsvektor $\mathbf{w}$ erhalten wir daraus das (für $m < N$) überbestimmte Gleichungssystem $(\mathbf{\Phi} - \gamma\mathbf{P}\mathbf{\Phi})\mathbf{w} = \bar{\mathbf{R}}$. Zum Lösen dieses Systems suchen wir die Lösung minimaler $\mathcal{L}_2$-Norm $\hat{\mathbf{w}}$

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left\| (\mathbf{\Phi} - \gamma\mathbf{P}\mathbf{\Phi})\mathbf{w} - \bar{\mathbf{R}} \right\|^2, \tag{1.12}$$

die wir dann durch (genaueres dazu in Kapitel 2)

$$\hat{\mathbf{w}} = \big((\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Phi})^{\mathsf{T}}(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Phi})\big)^{-1}(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Phi})^{\mathsf{T}}\bar{\mathbf{R}} \tag{1.13}$$

erhalten. Alternativ können wir die Minimierungsaufgabe in (1.12) auch äquivalent als

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{N} \left\{ \boldsymbol{\phi}(i)^{\mathsf{T}}\mathbf{w} - \sum_{j=1}^{N} \mathrm{P}(j|i,\pi(i)) \Big[\mathrm{R}(j|i,\pi(i)) + \gamma \boldsymbol{\phi}(j)^{\mathsf{T}}\mathbf{w} \Big] \right\}^2 \tag{1.14}$$

schreiben.

Sei $\tilde{\mathbf{v}}_{\mathrm{BRM}} := \mathbf{\Phi}\hat{\mathbf{w}}$ die resultierende Approximation für $\mathbf{v}^{\pi}$. Für jede Wertefunktion $\mathbf{v}$ wird die Differenz $\mathbf{v} - \mathcal{T}_{\pi}(\mathbf{v})$ das Bellman-Residuum von $\mathbf{v}$ genannt. Für die wirkliche Lösung $\mathbf{v}^{\pi}$ ist das Bellman-Residuum gleich Null, bei einer Approximation $\tilde{\mathbf{v}}$ im allgemeinen natürlich nicht mehr. Konstruieren wir uns daher eine Approximation $\tilde{\mathbf{v}}_{\mathrm{BRM}}$ für $\mathbf{v}^{\pi}$ über (1.12), d.h. über Minimierung des Bellman-Residuums

$$\tilde{\mathbf{v}}_{\mathrm{BRM}} = \operatorname*{argmin}_{\tilde{\mathbf{v}} \in \mathrm{Bild}(\mathbf{\Phi})} \left\| \tilde{\mathbf{v}} - \mathcal{T}_{\pi}(\tilde{\mathbf{v}}) \right\|^2,$$

dann suchen wir, anschaulich gesagt, im Unterraum Bild($\mathbf{\Phi}$) nach demjenigen Vektor $\tilde{\mathbf{v}}$, bei dem eine Anwendung des Bellman-Operators $\mathcal{T}_{\pi}$ eine minimale Änderung (bzgl. der $\mathcal{L}_2$-Norm) bewirkt. Abbildung 1.9 skizziert diese Situation graphisch.

2. Möglichkeit: Approximation des Fixpunkts (LSTD)

Eine Alternative ist es, stattdessen nach einem $\tilde{\mathbf{v}} \in \mathrm{Bild}(\mathbf{\Phi})$ zu suchen, das näherungsweise die Fixpunkteigenschaft erfüllt. Dazu bestimmen wir die Lösung $\tilde{\mathbf{v}}_{\mathrm{LSTD}}$ der Fixpunktgleichung

$$\tilde{\mathbf{v}} = \mathcal{P}\mathcal{T}_{\pi}(\tilde{\mathbf{v}}) \tag{1.15}$$

in $\tilde{\mathbf{v}}$, wobei $\mathcal{P}$ die (orthogonale) Projektion von $\mathbb{R}^N$ auf Bild($\mathbf{\Phi}$) ist. Gesucht ist demnach der Fixpunkt des Bellmanoperators gefolgt von Projektion, d.h. dasjenige Element aus Bild($\mathbf{\Phi}$), bei dem die Anwendung von $\mathcal{T}_{\pi}$ orthogonal ist (siehe Abbildung 1.9). Mit der Darstellung $\tilde{\mathbf{v}} = \mathbf{\Phi}\mathbf{w}$ lautet (1.15)

$$\mathbf{\Phi}\mathbf{w} = \mathcal{P}(\bar{\mathbf{R}} + \gamma \mathbf{P}\mathbf{\Phi}\mathbf{w}). \tag{1.16}$$

Die Projektion $\mathcal{P}$ kann geschrieben werden als $\mathcal{P} = \mathbf{\Phi}(\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathsf{T}}$ (siehe Kapitel 2), so daß wir aus (1.16) die Gleichung

$$\mathbf{\Phi}\mathbf{w} = \mathbf{\Phi}(\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathsf{T}}(\bar{\mathbf{R}} + \gamma \mathbf{P}\mathbf{\Phi}\mathbf{w})$$

erhalten. Weil $\mathbf{\Phi}$ vollen Spaltenrang hat (d.h. die Spaltenvektoren linear unabhängig sind), ergibt sich nun weiter

$$\begin{aligned} 0 &= \mathbf{\Phi}\big((\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathsf{T}}(\bar{\mathbf{R}} + \gamma \mathbf{P}\mathbf{\Phi}\mathbf{w}) - \mathbf{w}\big) \\ &= (\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathsf{T}}(\bar{\mathbf{R}} + \gamma \mathbf{P}\mathbf{\Phi}\mathbf{w}) - \mathbf{w} \\ &= \mathbf{\Phi}^{\mathsf{T}}\bar{\mathbf{R}} + \gamma \mathbf{\Phi}^{\mathsf{T}}\mathbf{P}\mathbf{\Phi}\mathbf{w} - \mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi}\mathbf{w} \end{aligned}$$

und damit

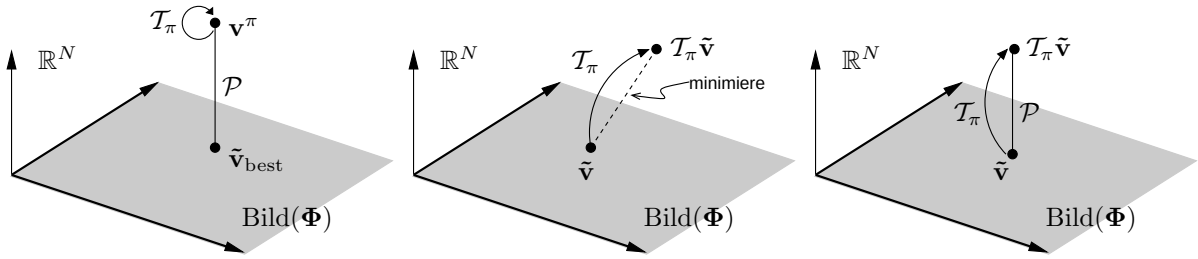
$$\mathbf{\Phi}^{\mathsf{T}}(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Phi})\mathbf{w} = \mathbf{\Phi}^{\mathsf{T}}\bar{\mathbf{R}},$$

so daß schließlich der Gewichtsvektor $\hat{\mathbf{w}}$ von $\tilde{\mathbf{v}}_{\mathrm{LSTD}} := \mathbf{\Phi}\hat{\mathbf{w}}$ durch

$$\hat{\mathbf{w}} = \big(\mathbf{\Phi}^{\mathsf{T}}(\mathbf{\Phi} - \gamma \mathbf{P}\mathbf{\Phi})\big)^{-1}\mathbf{\Phi}^{\mathsf{T}}\bar{\mathbf{R}} \tag{1.17}$$

berechnet wird.

Eine alternative Darstellung erhalten wir, wenn wir berücksichtigen, daß die Projektion $\mathcal{P}$ von $\mathcal{T}_{\pi}(\tilde{\mathbf{v}}) \in \mathbb{R}^N$ auf den Unterraum der darstellbaren Wertefunktionen auch als Least-Squares Problem interpretiert



(a) Gesucht: $\tilde{\mathbf{v}}_{\text{best}} := \mathcal{P}\mathbf{v}^\pi$

(b) Minimiere Bellman-Residuum $\tilde{\mathbf{v}}_{\text{BRM}} := \underset{\tilde{\mathbf{v}}\in\text{Bild}(\mathbf{\Phi})}{\text{argmin}} \ \|\tilde{\mathbf{v}} - \mathcal{T}_\pi(\tilde{\mathbf{v}})\|^2$

(c) Fixpunkt-Approximation $\tilde{\mathbf{v}}_{\text{LSTD}} = \underset{\tilde{\mathbf{v}}\in\text{Bild}(\mathbf{\Phi})}{\text{argmin}} \ \|\tilde{\mathbf{v}} - \mathcal{T}_\pi(\tilde{\mathbf{v}}_{\text{LSTD}})\|^2$

Abbildung 1.9: APE als ein Least-Squares Problem. Links: $\tilde{\mathbf{v}}_{\text{best}}$ ist die Bestapproximation von $\mathbf{v}^\pi$ aus Bild($\mathbf{\Phi}$). Diese kann aber nicht direkt bestimmt werden, weil $\mathbf{v}^\pi$ unbekannt ist. Mitte: Bei der Minimierung des Bellman-Residuums wird ein $\tilde{\mathbf{v}}$ gesucht, bei dem die Anwendung von $\mathcal{T}_\pi$ eine minimale Verschiebung bewirkt. Rechts: Bei der Approximation des Fixpunkts wird ein $\tilde{\mathbf{v}}$ gesucht, das Fixpunkt von $\mathcal{P}\mathcal{T}_\pi$ ist, d.h. bei dem die Anwendung von $\mathcal{T}_\pi$ orthogonal ist.

werden kann. Allgemein bedeutet Projektion eines $\mathbf{v}' \in \mathbb{R}^N$ auf den Unterraum Bild($\mathbf{\Phi}$) $\subset \mathbb{R}^N$: suche $\hat{\mathbf{v}} = \mathbf{\Phi}\hat{\mathbf{w}} \in \mathbb{R}^N$, so daß

$$\hat{\mathbf{v}} = \underset{\tilde{\mathbf{v}}\in\text{Bild}(\mathbf{\Phi})}{\text{argmin}} \ \|\tilde{\mathbf{v}} - \mathbf{v}'\|^2$$

ist. Bezogen auf den Gewichtsvektor $\hat{\mathbf{w}}$ in $\hat{\mathbf{v}} := \mathbf{\Phi}\hat{\mathbf{w}}$ heißt das

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^m}{\text{argmin}} \ \|\mathbf{\Phi}\mathbf{w} - \mathbf{v}'\|^2 = \underset{\mathbf{w}\in\mathbb{R}^m}{\text{argmin}} \ \sum_{i=1}^{N} \big\{\boldsymbol{\phi}(i)^\mathsf{T}\mathbf{w} - v'_i\big\}^2.$$

Setzen wir für $\mathbf{v}'$ den Vektor $\mathcal{T}_\pi(\tilde{\mathbf{v}}_{\text{LSTD}})$ mit $\tilde{\mathbf{v}}_{\text{LSTD}} = \mathbf{\Phi}\hat{\mathbf{w}}$ ein, so erhalten wir als alternative Darstellung von (1.15) die Fixpunktgleichung in $\hat{\mathbf{w}}$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^m}{\text{argmin}} \ \sum_{i=1}^{N} \left\{ \boldsymbol{\phi}(i)^\mathsf{T}\mathbf{w} - \sum_{j=1}^{N} \text{P}(j|i,\pi(i)) \Big[\text{R}(j|i,\pi(i)) + \gamma\boldsymbol{\phi}(j)^\mathsf{T}\hat{\mathbf{w}} \Big] \right\}^2 ,$$

was ähnlich aussieht wie (1.14), aber kein direktes Least-Squares Problem ist.

Anstelle der orthogonalen Projektion ist es möglich, die durch eine diagonale Matrix $\mathbf{D} = \text{diag}(d_1, \ldots, d_N)$ gewichtete zu nehmen. Versieht man den $\mathbb{R}^N$ mit dem Skalarprodukt $\langle \mathbf{x}, \mathbf{x}' \rangle_D := \mathbf{x}^\mathsf{T}\mathbf{D}\mathbf{x}'$ und Norm $\|\mathbf{x}\|_D := \sqrt{\mathbf{x}^\mathsf{T}\mathbf{D}\mathbf{x}} = \sum d_i x_i^2$, dann entspricht obige Projektion bzgl. $\|\cdot\|_D$ dem Problem

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmin}} \ \|\mathbf{\Phi}\mathbf{w} - \mathcal{T}_\pi(\mathbf{\Phi}\hat{\mathbf{w}})\|_D^2$$

was ausgeschrieben wieder gleich

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^m}{\text{argmin}} \ \sum_{i=1}^{N} d_i \left\{ \boldsymbol{\phi}(i)^\mathsf{T}\mathbf{w} - \sum_{j=1}^{N} \text{P}(j|i,\pi(i)) \Big[\text{R}(j|i,\pi(i)) + \gamma\boldsymbol{\phi}(j)^\mathsf{T}\hat{\mathbf{w}} \Big] \right\}^2 \tag{1.18}$$

ist. Die zugehörige Projektionsmatrix lautet $\mathcal{P} = \mathbf{\Phi}(\mathbf{\Phi}^\mathsf{T}\mathbf{D}\mathbf{\Phi})^{-1}\mathbf{\Phi}^\mathsf{T}\mathbf{D}$, so daß die Lösung $\hat{\mathbf{w}}$ in $\tilde{\mathbf{v}}_{\text{LSTD}} = \mathbf{\Phi}\hat{\mathbf{w}}$ statt durch (1.17) allgemeiner nun durch

$$\hat{\mathbf{w}} = \big[\mathbf{\Phi}^\mathsf{T}\mathbf{D}(\mathbf{\Phi} - \gamma\mathbf{P}\mathbf{\Phi})\big]^{-1}\mathbf{\Phi}^\mathsf{T}\mathbf{D}\bar{\mathbf{R}} \tag{1.19}$$

gegeben ist.

1.4.2 Ohne Modell

Erneut interessiert uns aber vor allem der Fall, daß die Übergangswahrscheinlichkeiten $\mathbf{P}$ und Belohnungsfunktion $\mathbf{R}$ nicht bekannt bzw. nicht verfügbar sind, und wir daher weder $\mathbf{\Phi} - \gamma\mathbf{P}\mathbf{\Phi}$ noch $\bar{\mathbf{R}}$ ausrechnen können.

Approximation durch Stichproben

Unter einer festen Politik reduziert sich der Entscheidungsprozeß auf einen Markov-Prozeß über $\mathcal{S}$ mit Übergangswahrscheinlichkeiten $[\mathbf{P}]_{ij} = \mathrm{P}(s_{t+1} = j | s_t = i)$. Sei $\mu(i)$, $i = 1, \ldots, N$ die stationäre Verteilung[5] der einzelnen Zustände und $\mathbf{D} = \mathrm{diag}(\mu(1), \ldots, \mu(N))$. Im folgenden betrachten wir die Variante Fixpunktapproximation, d.h. Lösung der Aufgabe (1.18). Sei

$$\mathbf{A} := \mathbf{\Phi}^\mathsf{T}\mathbf{D}(\mathbf{\Phi} - \gamma\mathbf{P}\mathbf{\Phi}) \qquad \text{und} \qquad \mathbf{b} := \mathbf{\Phi}^\mathsf{T}\mathbf{D}\bar{\mathbf{R}} \tag{1.20}$$

dann müssen wir, um nach (1.19) den Gewichtsvektor $\hat{\mathbf{w}}$ in $\tilde{\mathbf{v}}_{\mathrm{LSTD}} = \mathbf{\Phi}\hat{\mathbf{w}}$ zu bestimmen, das Gleichungssystem

$$\mathbf{A}\mathbf{w} = \mathbf{b} \tag{1.21}$$

lösen. Ohne Modell $\mathbf{P}$ und $\bar{\mathbf{R}}$ können $\mathbf{A}$ und $\mathbf{b}$ nicht exakt ausgerechnet werden. Wie zuvor können wir aber versuchen, beide Matrizen durch Stichproben, d.h. den bei laufendem Prozeß beobachteten Zustandsübergängen, wenigstens näherungsweise zu bestimmen.

Angenommen, wir beobachten die Folge von T Zustandsübergängen $s_0, s_1, s_2, \ldots, s_T$ und Belohnungen $r_1, \ldots, r_T$ wobei $s_{t+1} \in \{1 \ldots N\}$, mit $s_{t+1} \sim \mathrm{P}(\cdot\,|s_t, \pi(s_t))$ und $r_{t+1} = \mathrm{R}(s_{t+1}|s_t, \pi(s_t))$. Die stationäre Verteilung $\mu(i)$ von Zustand i kann jeweils durch die relative Häufigkeit eines besuchten Zustandes i in den beobachteten Daten approximiert werden: $\forall i = 1 \ldots N$ gilt

$$\mu(i) \approx \frac{1}{T}\sum_{t=0}^{T-1} \chi_{s_t}(i) \quad \text{wobei} \quad \chi_{s_t}(i) = \begin{cases} 1 & \text{falls } s_t = i \\ 0 & \text{sonst} \end{cases}$$

Ferner stellen wir fest, daß wir Matrix $\mathbf{A}$ aus (1.20) mit äußeren Produkten äquivalent darstellen können als

$$\mathbf{A} = \sum_{i=1}^{N} \mu(i)\boldsymbol{\phi}(i)\big(\boldsymbol{\phi}(i) - \gamma\sum_{j=1}^{N} \mathrm{P}(j|i, \pi(i))\boldsymbol{\phi}(j)\big)^\mathsf{T}.$$

Ähnliches gilt für Vektor $\mathbf{b}$:

$$\mathbf{b} = \sum_{i=1}^{N} \mu(i)\boldsymbol{\phi}(i)\sum_{j=1}^{N} \mathrm{P}(j|i, \pi(i))\mathrm{R}(j|i, \pi(i)).$$

Betrachten wir nun $\mathbf{A}$ genauer, dann zeigt sich, daß ($\mathbb{E}$ bezeichnet wieder den Erwartungswert)

$$\begin{aligned} \mathbf{A} &= \sum_{i=1}^{N} \mu(i)\boldsymbol{\phi}(i)\big(\boldsymbol{\phi}(i) - \gamma\,\mathbb{E}_{j\sim\mathrm{P}(\cdot\,|i)}\{\boldsymbol{\phi}(j)\}\big)^\mathsf{T} \\ &= \sum_{i=1}^{N} \mu(i)\mathbb{E}_{j\sim\mathrm{P}(\cdot\,|i)}\big\{\boldsymbol{\phi}(i)\big(\boldsymbol{\phi}(i) - \gamma\boldsymbol{\phi}(j)\big)^\mathsf{T}\big\} \end{aligned}$$

aus der gewichteten Summe von Matrizen der Form $\mathbb{E}_{j\sim\mathrm{P}(\cdot\,|i)}\big\{\boldsymbol{\phi}(i)\big(\boldsymbol{\phi}(i) - \gamma\boldsymbol{\phi}(j)\big)^\mathsf{T}\big\}$ besteht. Jeden einzelnen dieser Terme können wir durch die relative Häufigkeit annähern, mit der ein bestimmter Übergang in den Daten beobachtet wurde (d.h. ein Nachfolger j auf i folgt): $\forall i = 1 \ldots N$ gilt

$$\frac{1}{T}\sum_{t=0}^{T-1} \chi_{s_t}(i)\boldsymbol{\phi}(s_t)\big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)^\mathsf{T} \approx \mu(i)\mathbb{E}_{j\sim\mathrm{P}(\cdot\,|i)}\big\{\boldsymbol{\phi}(i)\big(\boldsymbol{\phi}(i) - \gamma\boldsymbol{\phi}(j)\big)^\mathsf{T}\big\}$$

[5] Die stationäre Verteilung $\boldsymbol{\mu} = \big(\mu(1), \ldots, \mu(N)\big)^\mathsf{T}$ erfüllt $\boldsymbol{\mu}^\mathsf{T}\mathbf{P} = \boldsymbol{\mu}$, sofern $\mathbf{P}$ irreduzibel ist. Ein episodisches Problem (mit absorbierenden Zuständen) wird formal so modifiziert, daß beim Erreichen eines Terminalzustands (dem Ende einer Episode) ein kostenloser Übergang in einen neuen Nicht-Terminalzustand erfolgt (dem Startpunkt einer neuen Episode).

Folglich ist

$$\begin{aligned}
&\frac{1}{T}\sum_{i=1}^{N}\sum_{t=0}^{T-1}\chi_{s_t}(i)\boldsymbol{\phi}(s_t)\big(\boldsymbol{\phi}(s_t)-\gamma\boldsymbol{\phi}(s_{t+1})\big)^{\mathsf{T}}\\
=&\frac{1}{T}\sum_{t=0}^{T-1}\boldsymbol{\phi}(s_t)\big(\boldsymbol{\phi}(s_t)-\gamma\boldsymbol{\phi}(s_{t+1})\big)^{\mathsf{T}}\sum_{i=1}^{N}\chi_{s_t}(i)\\
=&\frac{1}{T}\sum_{t=0}^{T-1}\boldsymbol{\phi}(s_t)\big(\boldsymbol{\phi}(s_t)-\gamma\boldsymbol{\phi}(s_{t+1})\big)^{\mathsf{T}}\\
=:&\frac{1}{T}\mathbf{A}_T
\end{aligned} \tag{1.22}$$

mit hinreichend vielen Stichproben T eine Approximation für $\mathbf{A}$ mit $\lim_{T\to\infty}\mathbf{A}_T/T=\mathbf{A}$ (fast sicher). Ähnlich folgern wir für $\mathbf{b}$, daß

$$\mathbf{b}_T := \sum_{t=0}^{T-1}\boldsymbol{\phi}(s_t)r_{t+1} \tag{1.23}$$

mit hinreichend vielen Stichproben eine Approximation für $\mathbf{b}$ ist, mit $\lim_{T\to\infty}\mathbf{b}_T/T=\mathbf{b}$ (fast sicher).

Fixpunkt-Approximation mit LSTD(0)

Ersetzt man in (1.21) $\mathbf{A}$ und $\mathbf{b}$ durch die aus den T beobachteten Zustandsübergängen gewonnenen Näherungen $\mathbf{A}_T$ und $\mathbf{b}_T$, so erhalten wir den Gewichtsvektor $\hat{\mathbf{w}}$ der Fixpunktapproximation $\tilde{\mathbf{v}}_{\mathrm{LSTD}}=\boldsymbol{\Phi}\hat{\mathbf{w}}$ durch den Ausdruck (der Faktor $1/T$ steht auf beiden Seiten und kann gekürzt werden)

$$\hat{\mathbf{w}} = \mathbf{A}_T^{-1}\mathbf{b}_T$$

oder, indem wir vom Lösen des Gleichungssystems wieder auf das zugehörige Least-Squares Problem zurückgehen, als Lösung von

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{t=0}^{T-1}\big\{\boldsymbol{\phi}(s_t)^{\mathsf{T}}\mathbf{w}-(r_{t+1}+\gamma\boldsymbol{\phi}(s_{t+1})^{\mathsf{T}}\hat{\mathbf{w}})\big\}^2. \tag{1.24}$$

Das Verfahren (1.24) zur Berechnung des Gewichtsvektors $\hat{\mathbf{w}}$ einer linear parametrisierten Wertefunktion in der Aufgabe Approximative Politik-Evaluation wird in der Literatur als Least-Squares Temporal-Difference Lernen (LSTD) bezeichnet. Der hier beschriebene Spezialfall LSTD(0) wurde zuerst von Bradtke und Barto (1996) vorgeschlagen.

Fixpunkt-Approximation mit LSTD(λ) und Vergleich mit TD(λ)

Zwischen dem Online-Verfahren TD(λ) und der geschlossenen Lösung von LSTD besteht ein enger Zusammenhang: es stellt sich heraus, daß der Gewichtsvektor $\mathbf{w}_t$ in TD(λ) gegen die Lösung $\mathbf{w}^*$ des Gleichungssystem

$$\mathbf{A}\mathbf{w}=\mathbf{b} \tag{1.25}$$

konvergiert (fast sicher), wobei $\mathbf{A}$ und $\mathbf{b}$ durch

$$\mathbf{A}=\boldsymbol{\Phi}^{\mathsf{T}}\mathbf{D}(\mathbf{I}-\gamma\mathbf{P})\sum_{k=0}^{\infty}(\gamma\lambda\mathbf{P})^k\boldsymbol{\Phi} \qquad \text{und} \qquad \mathbf{b}=\boldsymbol{\Phi}^{\mathsf{T}}\mathbf{D}\sum_{k=0}^{\infty}(\gamma\lambda\mathbf{P})^k\bar{\mathbf{R}}$$

definiert sind. Näheres dazu findet man in Tsitsiklis und Van Roy (1997). Für $\lambda=0$ sind $\mathbf{A}$ und $\mathbf{b}$ aber gerade identisch mit (1.20), womit einsichtig wird, daß in diesem Fall TD(0) gegen dieselbe Lösung wie LSTD(0) aus (1.24) konvergiert (mit wachsender Anzahl von beobachteten Übergängen).

Allgemeiner läßt sich LSTD aber auch für beliebige $\lambda \in [0,1]$ formulieren. Wie zuvor so können auch hier Matrix $\mathbf{A}$ und Vektor $\mathbf{b}$ durch Stichproben-Übergänge approximiert werden. Für die Details verweisen wir auf die oben angegebene Literatur; dort wird gezeigt, daß für

$$\mathbf{A}_T := \sum_{t=0}^{T-1} \mathbf{z}_t \big(\boldsymbol{\phi}(s_t) - \gamma \boldsymbol{\phi}(s_{t+1})\big)^\mathsf{T} \quad \text{mit} \quad \mathbf{z}_t = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \boldsymbol{\phi}(s_k)$$

und

$$\mathbf{b}_T := \sum_{t=0}^{T-1} \mathbf{z}_t r_{t+1}$$

wieder Konvergenzaussagen der Form $\lim_{T\to\infty} \mathbf{A}_T/T = \mathbf{A}$ und $\lim_{T\to\infty} \mathbf{b}_T/T = \mathbf{b}$ (fast sicher) gelten. Ersetzt man nun in (1.21) $\mathbf{A}$ und $\mathbf{b}$ durch die aus den beobachteten Zustandsübergängen gewonnenen Näherungen $\mathbf{A}_T$ und $\mathbf{b}_T$, dann erhalten wir den Gewichtsvektor $\hat{\mathbf{w}}$ wieder durch

$$\hat{\mathbf{w}} = \mathbf{A}_T^{-1} \mathbf{b}_T$$

oder, erneut über die zugehörige Least-Squares Formulierung ausgedrückt, als Lösung von

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{t=0}^{T-1} \Big\{ \boldsymbol{\phi}(s_t)^\mathsf{T} \mathbf{w} - \boldsymbol{\phi}(s_t)^\mathsf{T} \hat{\mathbf{w}} - \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} d(s_k, s_{k+1}; \hat{\mathbf{w}}) \Big\}^2 \tag{1.26}$$

wobei $d(s_k, s_{k+1}; \hat{\mathbf{w}})$ der schon aus TD(λ) bekannte Temporal-Difference Term

$$d(s_k, s_{k+1}; \hat{\mathbf{w}}) = \mathrm{R}(s_{k+1} | s_k, \pi(s_k)) + \gamma \boldsymbol{\phi}(s_{k+1})^\mathsf{T} \hat{\mathbf{w}} - \boldsymbol{\phi}(s_k)^\mathsf{T} \hat{\mathbf{w}}$$

ist. Das Verfahren (1.26) ist eine Verallgemeinerung von (1.24) auf beliebige $\lambda \in [0,1]$ und wird als LSTD(λ) bezeichnet. Es geht auf Boyan (1999) zurück. Sowohl LSTD(λ) als auch TD(λ) konvergieren gegen denselben Gewichtsvektor $\mathbf{w}^*$, der Lösung des Gleichungssystems (1.25), wie in Nedić und Bertsekas (2003) gezeigt wird. Der Vorteil von LSTD(λ) gegenüber TD(λ) ist, daß die Konvergenzgeschwindigkeit sehr viel höher ist, generell also weitaus weniger Zustandsübergänge beobachtet werden müssen, und die Lösung durch ein geschlossenes Least-Squares-artiges Verfahren bestimmt wird, womit u.a. die manchmal heikle Wahl der richtigen Lernrate α bei TD(λ) entfällt.

Ebenso wie TD(λ) so kann auch LSTD(λ) anstatt für die V-Funktion in analoger Weise auch für die Q-Funktion formuliert werden, was in Lagoudakis und Parr (2003) als LSTDQ bezeichnet wird. Auf diese Weise läßt sich dann wieder der Schritt Politik-Verbesserung ohne Modell durchführen, und wir können das effiziente LSTD Verfahren als Politik-Evaluationskomponente im Rahmen von approximativer Politik-Iteration einsetzen (vgl. Abbildung 1.7), um die optimale Politik näherungsweise zu bestimmen. In dieser Arbeit werden wir uns, da wir auch an praktischen Resultaten interessiert sind, hauptsächlich auf diesen Fall beziehen.

Was ist mit der Minimierung des Bellman-Residuums?

Eine analoge Vorgehensweise ist bei dem Ansatz Minimierung des Bellman-Residuums nur mit großen Einschränkungen möglich. Nach (1.13) erhalten wir zwar auch hier den Gewichtsvektor $\hat{\mathbf{w}}$ von $\tilde{\mathbf{v}}_{\mathrm{BRM}} = \boldsymbol{\Phi}\hat{\mathbf{w}}$ wie zuvor in (1.25) durch Lösen eines Gleichungssystems $\mathbf{A}\mathbf{w} = \mathbf{b}$, wobei dieses mal jedoch $\mathbf{A}$ anstatt

$$\mathbf{A} = \sum_{i=1}^{N} \mathbb{E}_{j\sim \mathrm{P}(\cdot\,|i)} \Big\{ \boldsymbol{\phi}(i) \big(\boldsymbol{\phi}(i) - \gamma \boldsymbol{\phi}(j)\big)^\mathsf{T} \Big\} \qquad \text{(LSTD)}$$

von der Form

$$\mathbf{A} = \sum_{i=1}^{N} \mathbb{E}_{j\sim \mathrm{P}(\cdot\,|i)} \big\{ \boldsymbol{\phi}(i) - \gamma \boldsymbol{\phi}(j) \big\} \mathbb{E}_{j\sim \mathrm{P}(\cdot\,|i)} \big\{ \boldsymbol{\phi}(i) - \gamma \boldsymbol{\phi}(j) \big\}^\mathsf{T} \qquad \text{(BRM)}$$

ist. Anders als bei LSTD können wir die einzelnen Terme der zweiten Summe BRM, d.h. für jeden Zustand i das Produkt $\mathbb{E}\{\boldsymbol{\phi}(i) - \gamma\boldsymbol{\phi}(j)\}\mathbb{E}\{\boldsymbol{\phi}(i) - \gamma\boldsymbol{\phi}(j)\}^\mathsf{T}$ nicht durch *eine* einzige Stichprobe

$$\big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)\big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)^\mathsf{T}$$

mit $s_{t+1} \sim \mathrm{P}(s_{t+1}|s_t = i)$ annähern. Um einen erwartungstreuen Schätzer für das Produkt zweier Zufallsvariablen zu erhalten, müssen stattdessen auch zwei voneinander unabhängig erzeugte Stichproben s_{t+1}, s'_{t+1} für den Folgezustand von $i = s_t$ verwendet werden:

$$\big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)\big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s'_{t+1})\big)^\mathsf{T},$$

siehe auch Baird (1995). In der Realität läßt sich das nicht oder nur sehr schwierig verwirklichen, weil die Stichproben aus real beobachteten Zustandsübergängen stammen und es praktisch unmöglich ist, solche „gedoppelten“ Paare zu erzeugen.

Die Minimierung des Bellman-Residuums zusammen mit Stichprobenapproximation ist für uns dennoch von Interesse. Für den Spezialfall deterministischer Probleme, bei denen Nachfolgezustände nicht probabilistisch folgen sondern deterministisch festgelegt sind, dürfen die beobachteten Nachfolgezustände in der beschriebenen Weise herangezogen werden, um eine Approximation für $\mathbf{A}$ aufzubauen. Zweitens besteht zu einem gewissen Grad auch die Hoffnung, daß der Defekt tolerierbar ist und sich auch dann gute Ergebnisse erzielen lassen, wenn das Problem nur „leicht“ nicht-deterministisch ist (d.h. Zustandsübergänge nur wenig von zufallsgesteuerten Faktoren/Ereignissen beeinträchtigt werden).

Für deterministische Probleme approximieren wir daher $\mathbf{A}$ durch

$$\mathbf{A}_T := \sum_{t=0}^{T-1} \big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)\big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)^\mathsf{T}$$

und $\mathbf{b}$ durch

$$\mathbf{b}_T := \sum_{t=0}^{T-1} \boldsymbol{\phi}(s_t) r_{t+1}.$$

Setzen wir die Approximation in das Gleichungssystem $\mathbf{A}\mathbf{w} = \mathbf{b}$ ein, so erhalten wir wieder, mit der gleichen Argumentation wie in (1.24), den Gewichtsvektor $\hat{\mathbf{w}}$ in $\tilde{\mathbf{v}}_{\mathrm{BRM}} = \boldsymbol{\Phi}\hat{\mathbf{w}}$ durch

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{t=0}^{T-1} \big\{\big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)^\mathsf{T}\mathbf{w} - r_{t+1}\big\}^2. \tag{1.27}$$

1.5 Aufbau und Inhalt der Arbeit

Das Problem

Reinforcement Learning ist ein theoretisch sehr machtvolles Instrument, mit dem im Bereich KI/Robotik selbständig optimales Verhalten näherungsweise gelernt werden könnte. Als äußerst problematisch erweist sich allein dessen praktische Umsetzung: um realistische Aufgabenstellungen mit RL/DP-Methoden erfolgreich zu lösen, ist, wie wir gesehen haben, Approximation unumgänglich. Dazu gibt es eine Reihe von einzelnen Techniken für die verschiedenen Komponenten der RL-Architektur, die erfolgreich zu einem Gesamtsystem zusammengeführt werden müssen. Gegenwärtig scheint das vor allem nur dann im großen Stil gut zu funktionieren, wenn das zu lösende Entscheidungsproblem einer speziellen Klasse mit einer bestimmten Struktur angehört, bei der Zustände, Aktionen, Übergangswahrscheinlichkeiten und Belohnungen von einer bekannten Form sind (z.B. die Aufgabenstellungen aus dem Bereich Operations Research).

Eines der zentralen Probleme ist die Darstellung der Wertefunktion als parametrisierte Funktion über den Zuständen (bzw. der Q-Funktion als parametrisierte Funktion über Zustands-Aktionspaaren). Hier müssen wir eine Menge von Basisfunktionen angeben, so daß die daraus zusammengesetzte Approximation möglichst genau die zugrundeliegende echte Funktion wiedergeben kann. Häufig sind die Zustände aber sehr hochdimensionale Vektoren, so daß in der Approximation bewährte aber nur für niedrigdimensionale Eingaberäume sinnvolle Methoden nicht anwendbar sind.

Unsere Lösung

Das Ziel dieser Arbeit ist es, die Skalierbarkeit von Reinforcement Lernen auf hochdimensionale Zustandsräume (unabhängig von einer speziellen Struktur) voranzutreiben. Dazu greifen wir auf nichtparametrisierte Ansätze aus der statistischen Regression zurück und verwenden das Werkzeug der sogenannten 'Regularisierungsnetzwerke', um die Wertefunktion aus den beobachteten Stichprobenübergängen zu lernen. Regularisierungsnetze (RN) sind eine Verallgemeinerung der eingangs beschriebenen gewöhnlichen Basisfunktionsnetze: die gesuchte Lösung wird nun direkt durch die Daten parametrisiert, so daß die explizite Wahl von Knoten/Basisfunktionen entfällt und daher bei hochdimensionalen Eingaben der 'Fluch der Dimension' umgangen werden kann. In vielen anderen Anwendungsgebieten des Überwachten Lernens sind RN, bzw. allgemeiner Gauß-Prozeß-Regression oder ähnliche kernbasierte Ansätze wie Support Vektor Maschinen, bereits etabliert und haben sich den anderen für hochdimensionale Eingaberäume geeigneten Verfahren (wie Feed-Forward Neuronalen Netzen) als mindestens ebenbürtig erwiesen. Gleichzeitig sind RN aber auch lineare Methoden, die technisch einfach handhabbar sind und für die die bestehenden Konvergenzaussagen von RL Gültigkeit behalten.

Das Problem von RN als kernbasierter Funktionsapproximator ist nicht mehr länger die Dimension der Daten, sondern in erster Linie der erforderliche Rechenaufwand. Um den Gewichtsvektor für RN zu bestimmen, ist allgemein ein lineares Gleichungssystem in n Variablen mit dichtbesetzter Systemmatrix zu lösen, wobei n die Anzahl der Daten ist. Bei RL haben wir es unter Umständen aber einerseits mit großen Datenmengen (> 10000) und andererseits mit einem Online-Lernproblem zu tun – zwei Aspekte, die sich nur schwer mit dem für gewöhnlich wie $\mathcal{O}(n^3)$ wachsenden Lösungsaufwand für RN vertragen.

Somit sind zwei Dinge zu tun: Erstens, einen effizienten Lösungsalgorithmnus für RN finden und formulieren, der auch und speziell im Fall von Online-Lernen anwendbar ist. Zweitens, diesen Lösungsalgorithmus auf RL übertragen, in ein Gesamtsystem zum näherungsweisen Lernen von optimaler Steuerung integrieren und praktisch demonstrieren, daß alle diese Einzelteile auch zusammen gut funktionieren. Unsere Arbeit gliedert sich demnach in zwei Teile:

1. **Effizienten Lösungsalgorithmus für Online-Lernen mit RN formulieren:**

   - Wir starten dazu in Kapitel 2, indem wir wichtige Hintergrundinformationen zum Lernen von Funktionen und der Anwendung von Regularisierung auf Least-Squares Regression kurz zusammenfassen. Das Kapitel endet mit einer Beschreibung der RN.
   - In Kapitel 3 erläutern wir dann eine Reihe von Approximationsverfahren, um die zuvor erklärten RN auch praktisch auszurechnen: eine auf partieller Gram-Schmidt Orthogonalisierung basierende Auswahl relevanter Basiselemente, sowie die 'Subset of Regressors'-Approximation, wodurch der Lösungsaufwand insgesamt stark reduziert wird und asymptotisch nur noch linear von der Anzahl der Daten abhängt. Beides sind Standardtechniken, die man in dieser oder in ähnlicher Form häufig im Zusammenhang mit kernbasiertem Lernen (insbesondere der Gauß-Prozeß Regression) erwähnt findet. Beide Techniken können aber nur im Fall von Offline-Lernen angewendet werden.
   - In Kapitel 4 übertragen wir daher diese Technik auf Online-Lernen mit RN. Wir formulieren einen neuen Lernalgorithmus, der wie das bekannte Verfahren Recursive Least-Squares funktioniert, aber zwei Neuerungen mitbringt: erstens, müssen wir bei der Herleitung explizit die Regularisierung und damit den zusätzlichen Strafterm mitberücksichtigen. Und zweitens, was noch viel wichtiger ist, kann unser Verfahren mit konstantem Zeitaufwand nicht nur neue Daten sondern auch neue Basisfunktionen in ein bereits bestehendes Modell einfügen. Die Basisfunktionen werden dabei zur Laufzeit direkt aus dem Datenstrom selektiert. Zur Selektion schlagen wir ein neues überwachtes Kriterium vor, das sowohl die 'Neuheit' eines Basisfunktionskandidaten (gegenüber den bisher ausgewählten), als auch die 'Relevanz' (Beitrag zur Verringerung des Fehlers) mitberücksichtigt. Selektierte Basisfunktionen, die sich zu einem späteren Zeitpunkt als irrelevant herausstellen, können mit ebenso wenig Aufwand auch wieder entfernt werden. Wir implementieren unseren Ansatz und demonstrieren dann anhand einer Reihe von Benchmarkdatensätzen, daß er beinahe ebenso effektiv (Performanz), aber deutlich effizienter (Laufzeit) als vergleichbare Offline-Varianten ist.

2. **RN auf RL übertragen:**

   - In Kapitel 5 übertragen wir schließlich den Lernalgorithmus auf das Problem approximativer Politik-Evaluation mit Least-Squares basiertem Temporal-Difference Lernen. Wir leiten einheitliche Rekursionsgleichungen für Online-Lernen mit RN für alle drei APE-Varianten BRM, LSTD-λ und LSPE-λ her. Wir integrieren diesen Baustein in ein auf approximativer Politik-Iteration basierendes Gesamtsystem zum autonomen und modellfreien Lernen von optimalen Verhalten. Anhand einer Auswahl typischer Benchmarkprobleme evaluieren wir dann unseren Ansatz; insbesondere mit den guten Resultaten (deutlich besser als die bisherige Bestmarke) beim herausfordernden RoboCup-Keepaway Problem und der Steuerung eines simulierten Oktopus-Tentakels demonstrieren wir die Leistungsfähigkeit unseres Ansatzes und seine Skalierbarkeit auch auf hochdimensionale Zuständsräume.

Der Einfachheit halber nehmen wir an, daß die Zustände reellwertige Vektoren und die Menge der möglichen Aktionen diskret und klein (< 10) ist. Unser vorgestellter Ansatz ist aber nicht darauf beschränkt, weil wir ohne Probleme durch eine andere Wahl des Kerns auch kontinuierliche Aktionen oder eine andere Struktur der Zustände behandeln können (beispielsweise können Kerne auch über Texte oder Graphen definiert werden, siehe etwa Shawe-Taylor und Cristianini (2004)).

Relevante Literatur

Der Ansatz, RN zur Approximation der Wertefunktion im Rahmen von RL zum autonomen Lernen von Verhalten in agentenbasierten Anwendungen und Robotik einzusetzen, ist, speziell vor dem Hintergrund Online-Lernen, neu.

Gegenwärtig gibt es nur eine zweite bekannte Arbeit, die Vergleichbares leistet: der GPTD-Ansatz (Gaussian Process Temporal Difference Learning) von Yaakov Engel (Engel et al., 2003; Engel, 2005; Engel et al., 2005a,b). Hier wird das Bellman-Residuum mittels Bayes'scher Regression (Gauß-Prozeß-Regression oder kurz GP-Regression) modelliert. Konzeptionell ist das ein anderer Ansatz. Technisch läuft GP-Regression allerdings auf dasselbe Gleichungssystem wie RN hinaus, so daß beide Ansätze von der Endformulierung äquivalent sind. Am Ende der Arbeit (Abschnitt 5.4.2) gehen wir genauer auf die Gemeinsamkeiten und Unterschiede ein.

Auf weitere relevante Literatur verweisen wir ansonsten dort im Text, wo es nötig wird.

Kapitel 2

Hintergrund: Lineare Modelle und Regularisierungsnetzwerke

Worin beschrieben wird, wie aus wenigen Daten eine ganze Funktion gelernt werden kann

Wie im vorangegangenen Kapitel dargelegt wurde, ist der zentrale Baustein von Reinforcement Lernen (RL) und approximativem Dynamischen Programmieren (ADP) das Lösen eines Approximationsproblems im Teilschritt APE. Etwas allgemeiner formuliert besteht die Aufgabe somit darin, aus wenigen beobachteten Daten (Stichproben einer unbekannten Funktion) Rückschlüsse auf die zugrundeliegende Funktion zu ziehen und eine Näherungslösung dafür zu erstellen. Aufgaben dieser Art findet man in vielen Anwendungsbereichen; neben dem hier adressierten Maschinellen Lernen traditionsgemäß in der Statistik und in manchen Feldern der angewandten Mathematik (häufig wurden äquivalente Verfahren auch unabhängig voneinder entwickelt oder zu einem späteren Zeitpunkt wiederentdeckt).

Als zentrales Werkzeug werden wir dazu in dieser Arbeit Regularisierungsnetze (RN) verwenden; diese sollen nun zunächst im Rahmen allgemeiner Funktionsapproximation vorgestellt werden (die Anwendung auf RL folgt dann in Kapitel 5). Gewissermaßen handelt es sich bei den RN um eine Verallgemeinerung linearer Basisfunktionsnetze auf unendlich viele Basisfunktionen, so daß die Wahl konkreter Basisfunktionen entfällt. Sie werden in dieser Form beschrieben von Girosi et al. (1995); Evgeniou et al. (2000) und sind funktional identisch mit den Verfahren Gauß-Prozeß Regression (Rasmussen und Williams, 2006) und Kernel Ridge Regression (Saunders et al., 1998), sowie eng verwandt mit den bekannten Verfahren Smoothing Splines bzw. deren Erweiterung auf höherdimensionale Räume, den Thin Plate Splines (Green und Silverman, 1994; Wahba, 1990), sowie auch den sogenannten Least-Squares Support Vector Machines (Suykens et al., 2002). Anstatt der funktionalanalytischen Herleitung von RN im Rahmen von Tikhonov-Regularisierung (Poggio und Girosi, 1989) wollen wir hier einen technisch einfacheren Weg gehen und sie über das 'Kernelisieren' von Ridge Regression einführen.

2.1 Vorbemerkung: Grundsätzliche Probleme beim Lernen von Funktionen

Wir betrachten im folgenden das Problem der Funktionsrekonstruktion (Funktionsapproximation) für unregelmäßige Daten aus höherdimensionalen Eingaberäumen. Gegeben seien n Datenpaare oder Trainingsdaten $\{\mathbf{x}_i, y_i\}_{i=1}^n$, wobei $\mathbf{x}_i \in \mathbb{R}^d$ die Eingaben (Inputs) und $y_i \in \mathbb{R}$ die zugehörigen beobachteten Ausgaben (Outputs) einer unbekannten Funktion $g : \mathbb{R}^d \to \mathbb{R}$ sind. Unser Ziel ist es, aus den gegebenen Trainingsdaten eine Funktion $f : \mathbb{R}^d \to \mathbb{R}$ zu lernen, die der den Daten zugrundeliegenden unbekannten Funktion g möglichst 'nahe' kommt. Insbesondere soll die gelernte Funktion f auch für neue Eingaben $\mathbf{x}$ eine gute *Vorhersage* $f(\mathbf{x})$ für $g(\mathbf{x})$ liefern.

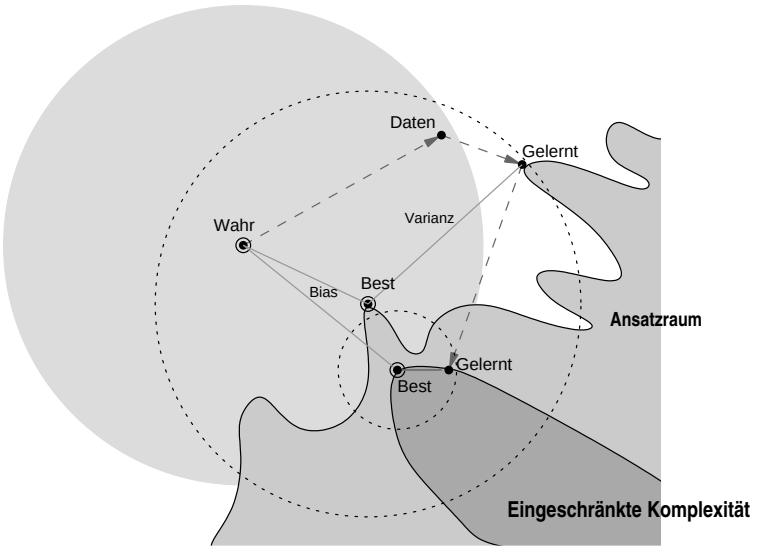


Abbildung 2.1: Illustration der Aufspaltung des erwarteten Vorhersagefehlers in Bias (Fehler aufgrund des Verfahrens) und Varianz (Fehler aufgrund der Daten). Betrachten wir als Ansatzraum der Lösung Funktionen hoher Komplexität, dann kann eine aus konkret vorliegenden Daten gelernte Lösung weit von der (theoretisch) bestmöglichen Lösung 'Best' abweichen (großer gestrichelter Kreis ist Varianz). Durch Regularisierung wird die Komplexität der möglichen Lösungen eingeschränkt: die bestmögliche Lösung 'Best' aus der Klasse von Funktionen eingeschränkter Komplexität ist geringfügig schlechter, d.h. der Bias wird erhöht. Andererseits werden die aus konkret vorliegenden Daten gelernten Lösungen aber viel näher um 'Best' herumgruppiert sein (kleiner gestrichelter Kreis), so daß insgesamt der erwartete Vorhersagefehler durch Regularisierung verringert werden kann. Abbildung nach Hastie et al. (2001).

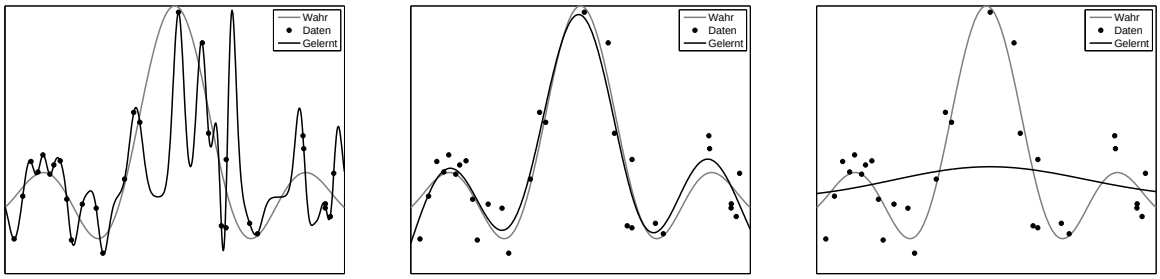
Technisch betrachtet reichen die Daten alleine jedoch nicht aus. Es handelt sich hierbei um ein sogenanntes schlecht gestelltes Problem, weil erstens sehr viele (unendlich viele) Lösungen möglich sind und zweitens die Lösung nicht stetig von den Daten abhängt (d.h. kleine Änderungen/Fehler in den Daten zu großen Änderungen/Fehlern in der gelernten Funktion führen können).

Um das Problem eindeutig lösbar zu machen und Stabilität herzustellen, muß man auf eine stark vereinfachende Zusatzannahme zurückgreifen. Im allgemeinen ist das die Annahme, daß das unbekannte g einen 'natürlichen' Sachverhalt wiedergibt und daher in gewisser Weise einfach oder 'glatt' (smooth) ist. Dabei soll 'glatt' heißen, daß die gesuchte Funktion nicht zu stark variiert, sondern für ähnliche Eingaben auch ähnliche Ausgaben liefert. Erreicht wird das, indem Freiheitsgrade eliminiert und die Komplexität der Lösung eingeschränkt oder *regularisiert* wird. Grob gesagt gibt es dafür u.a. folgende Möglichkeiten:

1. Wir betrachten als mögliche Ansatzfunktionen für f nur solche, die von vornherein eine bestimmte festgelegte Struktur haben (Basisfunktionsnetzwerke mit festen, wenigen Basisfunktionen). Die Komplexität wird primär über die Anzahl der Knoten/Basisfunktionen gesteuert.

2. Wir geben die Struktur von f nicht direkt vor (die Knoten/Basisfunktionen sind die Daten!), schränken aber die Komplexität explizit durch Bestrafen von 'Kurvigkeit' ein.

Die Fähigkeit, gute Vorhersagen zu liefern, hängt maßgeblich von der Stabilität und damit von der Komplexität der Lösung ab. In der Statistik betrachtet man dazu die *Bias-Varianz-Zerlegung* (etwa Hastie et al., 2001) des erwarteten Fehlers einer Vorhersage $f(\mathbf{x})$ für $g(\mathbf{x})$, wobei der Erwartungswert vom zufälligen Zustandekommen der Daten herrührt. Der 'Bias' ist ein Maß für die Genauigkeit; er beschreibt die Abweichung zwischen bestmöglicher Lösung (unter dem betrachteten Verfahren) und dem wirklichen g, unabhängig von den zur Verfügung stehenden Daten. Die 'Varianz' ist ein Maß für die



(a) Zu kompliziert (b) Guter Kompromiß (c) Zu einfach

Abbildung 2.2: Auswirkungen von Regularisierung und Modellkomplexität auf die gelernte Funktion. Links: die gelernte Funktion ist zu kompliziert (niedriger Bias, hohe Varianz). Mitte: die gelernte Funktion hat die richtige Komplexität (niedriger Bias, niedrige Varianz). Rechts: die gelernte Funktion ist zu einfach (hoher Bias, niedrige Varianz).

Streuung vom Erwartungswert; sie beschreibt, wie weit eine aus konkret vorliegenden Daten gelernte Lösung von der (theoretisch) bestmöglichen Lösung des Verfahrens im Mittel abweicht. In Abbildung 2.1 wird diese Situation skizziert und Abbildung 2.2 zeigt die Auswirkungen an einem konkreten Beispiel. Es gilt, zwischen den beiden Extremfällen zu komplizierte Lösung (niedriger Bias, hohe Varianz) und zu einfacher Lösung (hoher Bias, niedrige Varianz) den richtigen Kompromiß zu finden. Die Hoffnung ist, daß für ein kleines bißchen mehr an Bias (durch Einschränken der Komplexität) ein viel weniger an Varianz erzielt werden kann.

2.2 Einfache lineare Regression

Der einfachste strukturelle Zusammenhang ist der lineare. Hier nehmen wir an, daß die Ausgabe y entweder tatsächlich durch lineare Kombination der Eingaben $\mathbf{x} = \big(x_1, \ldots, x_d\big)^\mathsf{T}$ zustande kommt (was sehr unrealistisch ist), oder diese sich wenigstens gut dadurch erklären lassen kann. Das Modell ist demnach von der Form

$$y = w_1 x_1 + w_2 x_2 + \ldots w_d x_d + w_0. \tag{2.1}$$

wobei die unbekannten Gewichte (oder Parameter) $w_0, w_1, \ldots, w_d$ aus den Daten bestimmt werden müssen. Der Bias-Term w_0 wird üblicherweise nicht separat aufgeführt, sondern dem Parametervektor $\mathbf{w} = \big(w_0, \ldots, w_d\big)^\mathsf{T}$ hinzugefügt, dementsprechend werden die Eingaben um eine konstanten Eintrag zu $\mathbf{x} = \big(1, x_1 \ldots, x_d\big)^\mathsf{T}$ erweitert. Die zu lernende Funktion hat damit die Form $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\mathsf{T}\mathbf{x}$.

2.2.1 Gewöhnliches Least-Squares

Aufgabenstellung

Um den Gewichtsvektor $\mathbf{w}$ aus den Daten zu bestimmen, minimieren wir den quadratischen Fehler

$$\min_{\mathbf{w}\in\mathbb{R}^{d+1}} \; J(\mathbf{w}) = \sum_{i=1}^{n} \big(y_i - \sum_{j=0}^{d} x_{ij} w_j\big)^2 = \sum_i (y_i - \mathbf{x}_i^\mathsf{T}\mathbf{w})^2$$

auf den Trainingsdaten, was wir in kompakter Matrix-Schreibweise auch ausdrücken können als

$$\min_{\mathbf{w}\in\mathbb{R}^{d+1}} \; J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w}), \tag{2.2}$$

wobei die $n \times (d+1)$ Datenmatrix $\mathbf{X}$ aus den Zeilen $\mathbf{x}_i^\mathsf{T}$ besteht und $\mathbf{y}$ der $n \times 1$ Vektor $\mathbf{y} = \big(y_1, \ldots, y_n\big)^\mathsf{T}$ der Ausgaben ist. Die Lösung $\hat{\mathbf{w}}$ des quadratischen Optimierungsproblem erhalten wir, indem wir die Ableitung von (2.2) bezüglich $\mathbf{w}$ bestimmen und gleich Null setzen, durch den Ausdruck

$$\hat{\mathbf{w}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}, \tag{2.3}$$

der Lösung der sog. *Normalgleichung*. Setzen wir dieses $\hat{\mathbf{w}}$ in die zu lernende Funktion (2.1) ein, so erhalten wir als Vorhersage für die n Trainingsdaten den Vektor

$$\hat{\mathbf{y}} := \mathbf{X}\hat{\mathbf{w}} = \mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}$$

mit Einträgen $\hat{y}_i = \mathbf{x}_i^\mathsf{T}\hat{\mathbf{w}} = f(\mathbf{x}_i; \hat{\mathbf{w}})$.

Hintergrund für die Betrachtung des quadratischen Fehler ist das Gauß-Markov Theorem, welches besagt, daß der Least-Squares Schätzer der erwartungstreue Schätzer minimaler Varianz für $\mathbf{w}$ ist (vgl. Hastie et al., 2001). Zudem entspricht der Least-Squares Schätzer auch dem Maximum-Likelihood Schätzer für den Fall, daß die beobachteten Ausgaben einen additiven normalverteilten Fehler aufweisen, siehe Abschnitt 2.2.2. Allgemeiner ist das Least-Squares Kriterium aber nur eine Möglichkeit; daneben werden auch alternative Kostenfunktionen betrachtet, die zu anderen Formulierungen, wie z.B. den Support Vektor Maschinen[1] führen, siehe auch Schölkopf und Smola (2002); Shawe-Taylor und Cristianini (2004).

Geometrische Anschauung

Alternativ können wir Least-Squares Regression auch als Projektion auffassen (siehe Abbildung 2.3). Die n Einzelbeobachtungen $y_1, \ldots, y_n$ werden zu $\mathbf{y} = \big(y_1, \ldots, y_n\big)^\mathsf{T} \in \mathbb{R}^n$ zusammengefaßt. Die Spalten $\mathbf{x}_{(0)}, \mathbf{x}_{(1)} \ldots, \mathbf{x}_{(d)} \in \mathbb{R}^n$ der $n \times (d+1)$ Datenmatrix $\mathbf{X}$ spannen, sofern sie unabhängig sind, den $(d+1)$-dimensionalen Unterraum Bild($\mathbf{X}$) auf. Weil für jede Belegung des Gewichtsvektors $\mathbf{w}$ der Vektor $\mathbf{X}\mathbf{w}$ ein Element von Bild($\mathbf{X}$) ist, kann Aufgabe (2.2) auch als $\min_{\mathbf{z} \in \text{Bild}(\mathbf{X})} \|\mathbf{y} - \mathbf{z}\|^2$ ausgedrückt werden, d.h. gesucht ist dasjenige Element aus Bild($\mathbf{X}$), das minimalen (euklidischen) Abstand zu $\mathbf{y}$ hat. Dieses Element ist aber gerade $\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$, die orthogonale Projektion von $\mathbf{y}$ auf Bild($\mathbf{X}$), wobei für die Projektionsmatrix $\mathbf{H}$ in diesem Fall gilt:

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}. \tag{2.4}$$

Der Gewichtsvektor $\hat{\mathbf{w}}$ entspricht somit den Entwicklungskoeffizienten von $\hat{\mathbf{y}}$ in der (in aller Regel nicht orthogonalen) Basis der Spalten $\mathbf{x}_{(i)}$ von $\mathbf{X}$. Für den Beobachtungsvektor $\mathbf{y}$ erhalten wir die Zerlegung $\mathbf{y} = \hat{\mathbf{y}} + \hat{\mathbf{y}}^\perp$, in der das Residuum $\hat{\mathbf{y}}^\perp := \mathbf{y} - \hat{\mathbf{y}} = (\mathbf{I} - \mathbf{H})\mathbf{y}$ orthogonal ist, d.h. $\hat{\mathbf{y}}^\mathsf{T}\hat{\mathbf{y}}^\perp = 0$ gilt. Für den Abstand $\xi := \|\mathbf{y} - \hat{\mathbf{y}}\|^2$ bzw. äquivalent $\xi := J(\hat{\mathbf{w}})$ aus (2.2) erhalten wir $\xi = (\hat{\mathbf{y}}^\perp)^\mathsf{T}(\hat{\mathbf{y}}^\perp) = (\mathbf{y} - \hat{\mathbf{y}})^\mathsf{T}\hat{\mathbf{y}}^\perp = \mathbf{y}^\mathsf{T}\hat{\mathbf{y}}^\perp$.

Redundante Attribute und Rang-Defizit

Einen ersten Hinweis auf die Notwendigkeit von Regularisierung gibt uns folgende Problematik. Betrachten wir die Spalten von $\mathbf{X}$, die den einzelnen 'Attributen' der Beobachtungen entsprechen, dann ist es denkbar, daß die Attribute untereinander abhängig sind, d.h. daß eine Spalte (exakt) das Vielfache einer oder mehreren anderen Spalten ist. Beides hätte zur Folge, daß $\mathbf{X}$ nicht mehr vollen Rang $d+1$ hat. In der Realität werden solche Abhängigkeiten zwar selten exakt, aber doch zumindest näherungsweise auftreten, z.B. Größe und Gewicht erwachsener Menschen (im Idealfall). In diesen Fällen wird die Kreuzproduktmatrix $\mathbf{X}^\mathsf{T}\mathbf{X}$ schlecht konditioniert oder singulär sein, was die Inversenbildung in (2.3) erschwert.

Eine Möglichkeit das zu verhindern wäre es, von vorneherein die redundanten Attribute zu entfernen; wir würden dann lediglich eine reduzierte Auswahl aller Attribute betrachten. Diese relevante Teilmenge könnte beispielsweise durch sequentielle Vorwärtsselektion oder Rückwärtsselektion identifiziert werden.

[1] Bei SVM Regression wird statt der L_2-Kostenfunktion die ε-tolerante Kostenfunktion $|\cdot|_\varepsilon := \max(0, |\cdot| - \varepsilon|)$ verwendet, was zur Folge hat, daß das resultierende Optimierungsproblem zu einem quadratischen mit Nebenbedingungen wird und technisch aufwendig zu lösen ist. In der Praxis lassen sich mit SVM und RN in etwa gleichgute Ergebnisse erzielen, aufgrund der eleganteren Handhabbarkeit ziehen wir in der Arbeit die RN vor.

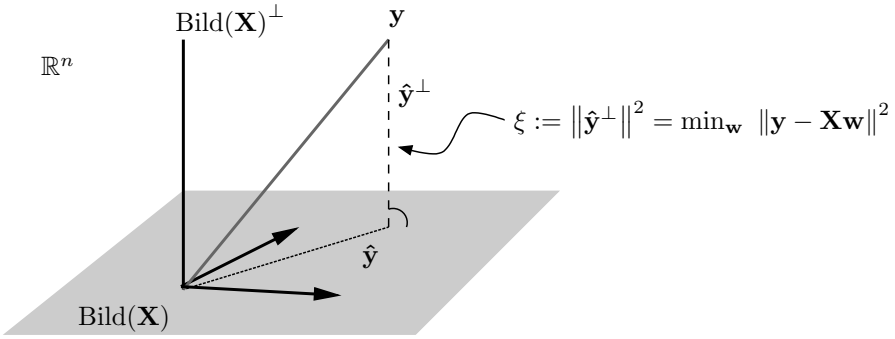


Abbildung 2.3: Least-Squares als Projektion des Beobachtungsvektors $\mathbf{y}$ auf den Bildraum von $\mathbf{X}$.

Eine zweite, alternative Strategie ist es, alle Attribute beizubehalten und auf die Singulärwertzerlegung oder QR-Zerlegung zurückzugreifen, um (2.2) numerisch stabil zu lösen (Golub und Van Loan, 1996). Im folgenden werden uns diese Strategie und Verfeinerungen davon interessieren. Zuvor wollen wir aber noch genauer klären, wie Abhängigkeit der Attribute mit der Stabilität der Lösung und damit einhergehend, der Vorhersageeigenschaft der gelernten Lösung, zusammenhängt.

2.2.2 Hinter den Kulissen

Im folgenden nehmen wir an, daß die Daten *zentriert* sind, d.h. die Spalten von $\mathbf{X}$ und die Beobachtungen $\mathbf{y}$ Mittelwert Null haben. Ist das der Fall, so kann auf einen expliziten Bias-Term w_0 verzichtet werden, was die folgenden Betrachtungen vereinfacht. Diese Annahme stellt keine Beschränkung der Allgemeinheit dar, denn der Bias-Term w_0 kompensiert ansonsten genau diese Transformation, es gilt $w_0 = \bar{\mathbf{y}} - \sum \bar{\mathbf{x}}_{(i)} \hat{w}_i$.

Ein einfaches Modell für Fehler und Maximum Likelihood Schätzer

Wir machen folgende grundlegende Modellannahme: die unbekannte Funktion sei tatsächlich linear, die Beobachtungen y_i aber seien fehlerhaft und mit einem additiven, identisch unabhängig normalverteilten Rauschen mit Varianz σ_N^2 behaftet, d.h.

$$y_i = \mathbf{x}_i^\mathsf{T} \mathbf{w} + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma_N^2). \tag{2.5}$$

In diesem Fall ist für eine feste Folge von unabhängigen Inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ der Vektor der zugehörigen Beobachtungen $\mathbf{y} = \big(y_1, \ldots, y_n\big)^\mathsf{T}$ ein Zufallsvektor, der ebenfalls normalverteilt gemäß $\mathbf{y} \sim \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma_N^2 \mathbf{I})$ ist: die Wahrscheinlichkeit $\mathbf{y}$ zu beobachten (die *likelihood*), gegeben Daten $\mathbf{X}$ und wirkliche Gewichte $\mathbf{w}$, ist aufgrund der Unabhängigkeit der Einzelbeobachtungen gerade gleich

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, w) = \prod_{i=1}^{n} \frac{1}{(2\pi\sigma_N^2)^{1/2}} \exp\left(-\frac{(y_i - \mathbf{x}_i^\mathsf{T}\mathbf{w})^2}{2\sigma_N^2}\right) \\ &= \frac{1}{(2\pi\sigma_N^2)^{n/2}} \exp\left(-\frac{\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2}{2\sigma_N^2}\right). \end{aligned} \tag{2.6}$$

Kennen wir $\mathbf{w}$ selbst aber nicht sondern wollen es auf der Grundlage von Daten schätzen oder lernen, so ist es nun eine naheliegende Vorgehensweise, denjenigen Parameter $\hat{\mathbf{w}}$ zu nehmen, der in (2.6) das Maximum erzielt. Maximieren von (2.6) ist aber äquivalent zum Minimieren des negativen Logarithmus von (2.6), so daß wir leicht einsehen können, daß der *maximum likelihood* Schätzer für $\mathbf{w}$ gerade die Lösung des Least-Squares Problems (2.2) ist.

Die Singulärwertzerlegung

Das Hauptwerkzeug zum Verständnis der zugrundeliegenden Problematik ist nun die Singulärwertzerlegung (SVD) von $\mathbf{X}$. Es sei $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$ die SVD der $n \times d$ Matrix $\mathbf{X}$, wobei die $n \times d$ Matrix $\mathbf{U} = \left[\mathbf{u}_1, \ldots, \mathbf{u}_d\right]$ orthonormale Spalten hat, d.h. es gilt $\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}$ und die $d \times d$ Matrix $\mathbf{V} = \left[\mathbf{v}_1, \ldots, \mathbf{v}_d\right]$ orthogonal ist, d.h. es gilt $\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{V}\mathbf{V}^\mathsf{T} = \mathbf{I}$. Die $d \times d$ Diagonalmatrix $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_d)$ enthält die Singulärwerte $\sigma_1 \geq \cdots \geq \sigma_d$. Existiert ein Index $p < d$, so daß für die nachfolgenden Singulärwerte $\sigma_p > \sigma_{p+1} = \ldots = \sigma_d = 0$ gilt, dann ist die Matrix $\mathbf{X}$ singulär mit Rang $p < d$. Die Spalten von $\mathbf{U}$ sind eine orthonormale Basis des Bildraums von $\mathbf{X}$, die orthonormalen Spalten von $\mathbf{V}$ spannen den Bildraum von $\mathbf{X}^\mathsf{T}$ und damit gleichbedeutend von $\mathbf{X}^\mathsf{T}\mathbf{X}$ auf.

Sei nun $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$ die SVD von $\mathbf{X}$. Ferner habe $\mathbf{X}$ vollen Rang d. Setzen wir die SVD in (2.3) ein, so folgt für $\mathbf{X}^\mathsf{T}\mathbf{X}$ zunächst $\mathbf{X}^\mathsf{T}\mathbf{X} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T})^\mathsf{T}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}) = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^\mathsf{T}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\mathsf{T}$, und wir können die Lösung $\hat{\mathbf{w}}$ dann schreiben als:

$$\hat{\mathbf{w}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} = (\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\mathsf{T})^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\mathsf{T}\mathbf{y} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\mathsf{T}\mathbf{y}$$

oder anders ausgedrückt

$$\hat{\mathbf{w}} = \sum_{i=1}^{d} \frac{\mathbf{u}_i^\mathsf{T}\mathbf{y}}{\sigma_i}\mathbf{v}_i. \tag{2.7}$$

Die Lösung $\hat{\mathbf{w}}$ des Least-Squares Problems ist demnach eine Linearkombination der rechten Singulärvektoren $\mathbf{v}_i$ der Datenmatrix $\mathbf{X}$. Setzen wir diese Darstellung von $\hat{\mathbf{w}}$ in $\hat{\mathbf{y}} = \mathbf{H}\mathbf{y} = \mathbf{X}\hat{\mathbf{w}}$ ein, dann erhalten wir $\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\mathsf{T} = \mathbf{U}\mathbf{U}^\mathsf{T}$ und somit gilt für die Approximation $\hat{\mathbf{y}}$ die als Bestapproximation bekannte Darstellung

$$\hat{\mathbf{y}} = \mathbf{U}\mathbf{U}^\mathsf{T}\mathbf{y} = \sum_{i=1}^{d}(\mathbf{u}_i^\mathsf{T}\mathbf{y})\mathbf{u}_i.$$

Für den Fall, daß $\mathbf{X}$ exakt Rang $p < d$ hat, summiert man in (2.7) nur über die ersten p Singulärvektoren, um eine Lösung $\hat{\mathbf{w}}$ minimaler Norm $\|\hat{\mathbf{w}}\|$ zu erhalten:

$$\hat{\mathbf{w}} = \sum_{i=1}^{p} \frac{\mathbf{u}_i^\mathsf{T}\mathbf{y}}{\sigma_i}\mathbf{v}_i. \tag{2.8}$$

Varianz der Lösung

Greifen wir wieder auf unsere Modellannahme (2.5) mit festen Eingaben $\mathbf{x}_1, \ldots, \mathbf{x}_n$ und zufälligem Beobachtungsvektor $\mathbf{y}$ zurück. Zur Abkürzung sei $\mathbf{X}^\dagger := (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}$. Dann ist auch die Lösung $\hat{\mathbf{w}} = \mathbf{X}^\dagger\mathbf{y}$ ein Zufallsvektor mit Erwartung $\mathbb{E}\{\hat{\mathbf{w}}\} = \mathbb{E}\{\mathbf{X}^\dagger(\mathbf{X}\mathbf{w} + \boldsymbol{\varepsilon})\} = \mathbf{w}$ und Varianz

$$\mathrm{Var}(\hat{\mathbf{w}}) = \mathrm{Var}(\mathbf{X}^\dagger\mathbf{y}) = \mathbf{X}^\dagger \mathrm{Var}(\mathbf{y})(\mathbf{X}^\dagger)^\mathsf{T} = \sigma_N^2\mathbf{X}^\dagger(\mathbf{X}^\dagger)^\mathsf{T} = \sigma_N^2(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}$$

wobei wir Gleichung (B.1)[2] verwendet haben. Mit Hilfe der SVD kann das wieder geschrieben werden als

$$\mathrm{Var}(\hat{\mathbf{w}}) = \sum_{i=1}^{d} \frac{\sigma_N^2}{\sigma_i^2}\mathbf{v}_i\mathbf{v}_i^\mathsf{T}. \tag{2.9}$$

Diese Darstellung zeigt, daß die Varianz der Lösung maßgeblich von der Größe der kleinsten Singulärwerte bestimmt wird: ein Rauschen oder Defekt in den Daten der Größenordnung $\mathcal{O}(\sigma_N)$ führt zu einem Rauschen der Größenordnung $\mathcal{O}(\sigma_N/\sigma_p)$ in der Lösung $\hat{\mathbf{w}}$, wobei σ_p der kleinste Singulärwert von $\mathbf{X}$ ist. Hat die Datenmatrix $\mathbf{X}$ also Singulärwerte, die signifikant kleiner sind als die Standardabweichung des Rauschens σ_N der Daten, dann wird die Lösung $\hat{\mathbf{w}}$ mit einem viel größeren Rauschen versehen sein, als es in dem Ursprungsproblem, den Daten selbst, der Fall war. Kleine Singulärwerte treten dann auf, wenn die Attribute näherungsweise abhängig sind.

[2] Es gilt: $\mathrm{Var}(\mathbf{A}\mathbf{z}) = \mathbf{A}\,\mathrm{Var}(\mathbf{z})\mathbf{A}^\mathsf{T}$

In der Praxis ist hohe Varianz ein deutliches Zeichen dafür, daß die gelernte Funktion schlechte Generalisierungseigenschaften besitzt (vgl. Abschnitt 2.1). Um die Generalisierungseigenschaften nachhaltig zu verbessern, wird daher im folgenden versucht, die Varianz und damit die Komplexität der Lösung einzuschränken. Mathematisch strenger behandelt findet sich dieser Sachverhalt auch in der statistischen Lerntheorie wieder, wo Vapnik (1998) das Prinzip der Strukturellen Risikominimierung zur Komplexitätskontrolle vorgeschlägt. Für das hier geschilderte Lernproblem läuft das aber auf dasselbe hinaus wie klassische Tikhonov-Regularisierung und das im folgenden beschriebene Verfahren Ridge Regression (Evgeniou et al., 2000).

2.3 Regularisierte lineare Regression und Ridge Regression

Neben solchen Methoden, die auf einer expliziten Elimination redundanter Attribute beruhen, gibt es auch einen zweiten, eleganteren Weg, um die Varianz (Komplexität) zu reduzieren: die sog. *Shrinkage* Methoden. Hierbei werden prinzipiell alle vorhandenen Attribute berücksichtigt, zum Auffinden der Lösung aber unterschiedlich stark gewertet; die Daten werden bezüglich einer bestimmten Basis entwickelt und die zugehörigen Entwicklungskoeffizienten werden dann 'geschrumpft'. Beispiele hierfür sind die (abgeschnittene) Singulärwertzerlegung auch bekannt als PCA-Regression (Jolliffe, 1986), Partial Least Squares (Wold, 1975), Ridge Regression (Hoerl und Kennard, 1970) und LASSO (Tibshirani, 1996).

Die beiden erst genannten Verfahren beruhen auf Projektion und führen, ebenso wie die explizite Selektion (Vorauswahl nicht-redundanter Attribute), zu einer Reduktion der Anzahl der Variablen im Modell. Anders als bei der expliziten Selektion werden dabei aber neue 'abgeleitete Attribute' gebildet, die durch Linearkombination aus den ursprünglichen Attributen hervorgehen. Ein Nachteil dabei ist, daß die Interpretation der Variablen als Attribute verloren geht. Das kann für den praktischen Einsatz (z.B. bei typischen Data-Mining Anwendungen), wo Attribute mit einer realen Bedeutung (z.B. Einkommen, Alter) verbunden sind, unerwünscht sein. Die beiden letztgenannten Verfahren Ridge Regression und LASSO kann man dagegen als kontinuierliche Selektion auffassen, die alle Attribute beibehält aber unterschiedlich stark gewichtet.

Allen diesen Verfahren gemein ist, daß sie von einem zusätzlichen Parameter, dem frei wählbaren Regularisierungsparameter, gesteuert werden. Dieser drückt aus, in welchem Ausmaß wir höheren Bias zugunsten niedrigerer Varianz in Kauf nehmen wollen. Im Falle projektionsbasierter Regularisierung ist der Parameter diskret (Dimension des Unterraums), bei den anderen Verfahren ist er kontinuierlich.

In den beiden folgenden Abschnitten wird zunächst die Wirkungsweise von Regularisierung anhand von PCA-Regression (PCR) und Ridge Regression (RR) vorgestellt. Anschließend werden wir die Verallgemeinerung von RR auf beliebige Merkmalsräume, das sogenannte Kernel-RR diskutieren, was uns schließlich zu dem zentralen Werkzeug dieser Arbeit, den Regularisierungsnetzwerken, führt.

Im folgenden gelte wieder die Annahme, daß das Problem zentriert ist, d.h. der Bias-Term nicht explizit erwähnt werden muß.

2.3.1 PCA-Regression (PCR)

Eine erste, sehr einfache Möglichkeit die Lösung zu stabilisieren und deren Varianz zu reduzieren ist die abgeschnittene Singulärwertzerlegung oder PCA-Regression. Angenommen, die Datenmatrix $\mathbf{X}$ habe vollen Rang d und (stark) abfallende Singulärwerte. Wir projizieren nun die Daten in einen Unterraum kleinerer Dimension $\hat{p} \leq d$ und lösen dann das gewöhnliche Least-Squares Problem zwischen den projizierten Daten und den Beobachtungen $\mathbf{y}$. Anteile der Daten außerhalb des Unterraums werden ignoriert.

PCA als unüberwachte Dimensionsreduktion

Der Unterraum wird mit der sog. Hauptachsentransformation (PCA) konstruiert und von den 'Richtungen' aufgespannt, bezüglich denen die Daten maximale Varianz haben. Die Hauptachsen sind definiert als

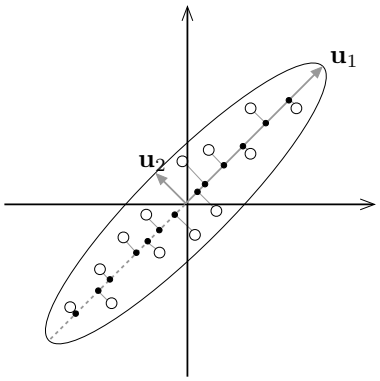


Abbildung 2.4: Veranschaulichung von PCA: $\mathbf{u}_1$ ist die Richtung maximaler Varianz der Daten (die Hauptrichtung). Die Projektion der Daten auf den Unterraum $\mathbf{u}_1$ liefert unter allen Unterräumen der Dimension 1 die beste Rekonstruktion der Daten.

die Eigenvektoren der Kovarianzmatrix $\mathbf{X}^\mathsf{T}\mathbf{X}/n$ der Stichproben, absteigend geordnet nach Größe der Eigenwerte. Ist $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$ die SVD der zentrierten Datenmatrix $\mathbf{X}$, dann gilt für die Eigenwertzerlegung $\mathbf{X}^\mathsf{T}\mathbf{X} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\mathsf{T}$, so daß die Hauptachsen gerade den rechten Singulärvektoren $\mathbf{v}_i$ (oder Eigenvektoren von $\mathbf{X}^\mathsf{T}\mathbf{X}$) entsprechen. Der erste Eigenvektor $\mathbf{v}_1$ mit Eigenwert σ_1^2 ist derjenige 1d-Unterraum, bezüglich dessen die Projektion der Daten maximale Varianz $\mathrm{Var}(\mathbf{X}\mathbf{v}_1) = \sigma_1^2/n$ hat. Der letzte Eigenvektor $\mathbf{v}_d$ zum kleinsten Eigenwert σ_d^2 liefert dagegen die Richtung mit der kleinsten Varianz σ_d^2/n. Die Projektion auf das Erzeugnis der ersten $\hat{p}$ Singulärvektoren behält daher möglichst viele Informationen bei und liefert unter allen möglichen Unterräumen mit Dimension $\hat{p}$ die beste Rekonstruktion der Daten (siehe Abbildung 2.4). PCA ist äquivalent zu einer abgebrochenen Entwicklung in den Singulärvektoren, bei der die letzten und kleinsten Singulärwerte herausgenommen werden, die *abgeschnittene* SVD.

Least-Squares auf PCA-induziertem Unterraum

In der PCA-Regression wählen wir als Regularisierungsparameter die effektive Dimension der Daten $\hat{p} \leq d$ und erhalten dann den Gewichtsvektor $\hat{\mathbf{w}}$ des Regressionsmodells mittels

$$\hat{\mathbf{w}} = \sum_{i=1}^{\hat{p}} \frac{\mathbf{u}_i^\mathsf{T}\mathbf{y}}{\sigma_i}\mathbf{v}_i. \tag{2.10}$$

Ähnlich zu dem Fall, daß $\mathbf{X}$ nicht vollen Rang $p < d$ hat, wo in (2.8) alle Singulärvektoren entfernt werden, die zu Singulärwerten $\sigma_{p+1} = \ldots = \sigma_d = 0$ gehören, wird in (2.10) der Beitrag entlang aller Singulärvektoren herausgefiltert, die zu 'kleinen' Singulärwerten $\sigma_{\hat{p}+1} \geq \ldots \geq \sigma_d > 0$ gehören. Auf diese Weise wird die Varianz der Lösung $\hat{\mathbf{w}}$ in (2.9) reduziert, was sich positiv auf die Generalisierungseigenschaften und Vorhersagequalität auswirken kann.

Zwei Anmerkungen: der mit PCA konstruierte Unterraum berücksichtigt nur die Varianz der Daten $\mathbf{X}$, nicht aber, ob die einzelnen Richtungen auch relevant sind für die Rekonstruktion der Beobachtungen $\mathbf{y}$. Allgemein wird das als *unüberwacht* bezeichnet; eine *überwachte* Variante wäre demgegenüber PLS (Partial Least Squares), welche wir hier aber nicht weiter erläutern werden. Ein zweiter Punkt ist es, für einen konkreten Datensatz den richtigen Regularisierungsparameter $\hat{p}$ zu finden; dieser kann durch Ausprobieren, Auffinden von Sprüngen im Eigenwertspektrum oder anderen Methoden, wie der generalisierten Kreuzvalidierung (GCV) gefunden werden (etwa Craven und Wahba, 1979; Louis, 1989; Hansen, 1998), siehe auch Seite 36.

2.3.2 Ridge Regression (RR)

Im Ansatz *Ridge Regression* (Hoerl und Kennard, 1970) wird die Lösung stabilisiert, indem man anstelle des gewöhnlichen quadratischen Fehlers in (2.2) noch zusätzlich einen expliziten Strafterm für die Kom-

plexität der Lösung einführt, oder äquivalent, ein restringiertes Optimierungsproblem betrachtet, bei dem die Norm der Lösung nach oben beschränkt wird. Auf diese Weise wird zu starkes Wachstum der Gewichte vermieden und die Varianz der Lösung verringert. Im Zusammenhang mit Neuronalen Netzen ist dieser Ansatz auch bekannt als *weight decay* (etwa Bishop, 1995).

Aufgabenstellung

Analog zu (2.2) betrachten wir nun den quadratischen Fehler mit zusätzlichem Strafterm

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2 \tag{2.11}$$

wobei der erste Term Nähe zu den Daten erzwingt, der zweite Term aber gleichzeitig die Komplexität des jeweiligen Kandidaten bestraft. Der reellwertige Regularisierungsparameter $\lambda > 0$ drückt aus, inwieweit wir einer der beiden gegensätzlichen Seiten Vorrang vor der anderen einräumen wollen. Wie zuvor findet man die Lösung $\hat{\mathbf{w}}$ nach Differentiation von (2.11) durch die generalisierte Normalgleichung

$$\hat{\mathbf{w}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}. \tag{2.12}$$

Für $\lambda \to 0$ reduziert sich RR auf gewöhnliches Least-Squares, für $\lambda \to \infty$ geht die Lösung $\hat{\mathbf{w}}$ gegen Null. Für $\lambda > 0$ ist die Kreuzproduktmatrix $(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})$ immer positiv definit. Als Vorhersage für die n Trainingsdaten erhalten wir nun $\hat{\mathbf{y}} = \mathbf{S}_\lambda\mathbf{y}$, wobei $\mathbf{S}_\lambda$ als

$$\mathbf{S}_\lambda := \mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}$$

definiert wird. Anders als $\mathbf{H}$ in (2.4) ist $\mathbf{S}_\lambda$ aber nicht mehr länger die orthogonale Projektion auf den Unterraum Bild($\mathbf{X}$).

Alternativ zu der auf (2.12) hinauslaufenden Formulierung kann (2.11) äquivalent auch als ein gewöhnliches Least-Squares Problem

$$\min_{\mathbf{w}} \; J_\lambda(\mathbf{w}) = \left\| \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{I} \end{bmatrix} \mathbf{w} \right\|^2$$

mit erweiterter Datenmatrix und Beobachtungsvektor geschrieben werden und dann mit Standardvefahren wie QR-Zerlegung numerisch stabil gelöst werden (Golub und Van Loan, 1996).

Hinter den Kulissen

Genaueren Aufschluß über die Funktionsweise von Ridge Regression erhalten wir wieder über die Basisentwicklung in den Singulärvektoren. Sei $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$ die SVD von $\mathbf{X}$. Setzen wir diese Darstellung in (2.12) ein, so erhalten wir

$$\hat{\mathbf{w}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} = (\mathbf{V}(\mathbf{\Sigma}^2 + \lambda\mathbf{I})\mathbf{V}^\mathsf{T})^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\mathsf{T}\mathbf{y}$$

oder anders gesagt

$$\hat{\mathbf{w}} = \sum_{i=1}^{d} f_i \frac{\mathbf{u}_i^\mathsf{T}\mathbf{y}}{\sigma_i}\mathbf{v}_i \tag{2.13}$$

wobei die f_i die sogenannten Filterfaktoren

$$f_i = \frac{\sigma_i^2}{\sigma_i^2 + \lambda}$$

sind.

Setzen wir diese Darstellung von $\hat{\mathbf{w}}$ wieder in $\hat{\mathbf{y}} = \mathbf{S}_\lambda\mathbf{y} = \mathbf{X}\hat{\mathbf{w}}$ ein, dann erhalten wir $\mathbf{S}_\lambda = \mathbf{U}\mathbf{\Sigma}^2(\mathbf{\Sigma}^2 + \lambda\mathbf{I})^{-1}\mathbf{U}^\mathsf{T}$ und somit gilt für $\hat{\mathbf{y}}$ die Darstellung

$$\hat{\mathbf{y}} = \mathbf{U}\mathbf{\Sigma}^2(\mathbf{\Sigma}^2 + \lambda\mathbf{I})^{-1}\mathbf{U}^\mathsf{T}\mathbf{y} = \sum_{i=1}^{d} f_i(\mathbf{u}_i^\mathsf{T}\mathbf{y})\mathbf{u}_i.$$

Betrachten wir die Filterfaktoren genauer, so sieht man folgendes: für $\sigma_i \gg \lambda$ gilt $f_i \to 1$, andererseits gilt $f_i \to 0$ für $\sigma_i \ll \lambda$. Die Filterfaktoren f_i haben somit den Effekt, den Beitrag von Singulärvektoren mit kleinen Singulärwerten, d.h. den Beitrag der Daten entlang Richtungen kleiner Varianz, zu unterdrücken. Vergleicht man die Darstellung von RR in (2.13) mit der Darstellung von PCR in (2.10), so ist festzustellen, daß beide Methoden gegenüber gewöhnlicher Regression in (2.7) prinzipiell das gleiche tun und Richtungen kleiner Varianz herausfiltern oder 'schrumpfen'. In PCR geschieht das diskret: die Filterfaktoren f_i haben Wert 1 für $i = 1, \ldots, p$ und Wert 0 für $i = \hat{p} + 1, \ldots, d$, d.h. eine Richtung ist entweder ganz oder gar nicht dabei. In RR erfolgt das Schrumpfen dagegen gradueller, und in der Praxis liefert RR gewöhnlich auch die besseren Ergebnisse (Frank und Friedman, 1993).

Zur Wahl des Regularisierungsparameters

Wie schon bei PCR so ist auch bei RR die Wahl des Regularisierungsparameters λ schwierig, weil es keine geschlossene optimale Lösung, sondern nur verschiedene Strategien gibt. Eine Möglichkeit wäre es, zuerst die Komplexität der Approximation festzulegen und dann λ entsprechend zu wählen. Ein in der Statistik bekanntes Maß für Komplexität einer Approximation ist die effektive Anzahl der Freiheitsgrade im Modell. Sie wird definiert als (Hastie et al., 2001):

$$df(\lambda) := \mathrm{Spur}(\mathbf{S}_\lambda) = \sum_{i=1}^{d} f_i.$$

Im Falle von gewöhnlichem Least-Squares ist $df := \mathrm{Spur}(\mathbf{H}) = d$, die Anzahl der (unabhängigen) Spalten von $\mathbf{X}$; für PCR ist es $\hat{p}$, die gewählte Dimension des Unterraums. Ein Vorteil dieses Komplexitätsmaßes ist die Unabhängigkeit vom verwendeten Verfahren; wir können so das Resultat verschiedener gelernter Modelle bei gleicher Komplexität direkt miteinander vergleichen. In der Praxis kann es zudem einfacher sein, die gewünschte Anzahl der Freiheitsgrade $df(\lambda)$ vorzugeben und dann das zugehörige λ auszurechnen, anstatt λ direkt zu suchen.

Eine zweite Strategie wäre es, ein Kriterium wie die generalisierte Kreuzvalidierung (GCV)

$$\mathrm{GCV}(\lambda) := \frac{n \sum_{i=1}^{n} \{y_i - f(\mathbf{x}_i; \hat{\mathbf{w}})\}^2}{\{n - df(\lambda)\}^2}$$

als Schätzung des Vorhersagefehlers zu verwenden (Craven und Wahba, 1979), und bezüglich des Parameters λ zu minimieren. Eine GCV-Kurve stellt den Vorhersagefehler gegenüber der Komplexität graphisch dar und ist von grob V-förmiger Gestalt. Als Regularisierungsparameter λ wählen wir denjenigen Wert, der das Minimum in der Kurve erzielt. Daneben gibt es noch eine ganze Reihe weiterer möglicher Vorgehensweisen, etwa Typ-II Maximum Likelihood im Rahmen Bayes'scher Regression (Rasmussen und Williams, 2006), dem Diskrepanzprinzip oder L-Kurvenkriterium im Rahmen schlecht gestellter Probleme (Hanke und Hansen, 1993; Hansen, 1998), oder anderen vor allem in der Statistik beheimateten Verfahren (Hastie et al., 2001).

2.4 Basisfunktionsmodelle und Kernel Ridge Regression

Die bislang betrachteten einfachen linearen Modelle werden häufig nicht ausdrucksstark genug sein, um reale Zusammenhänge zwischen den Ein- und Ausgaben sinnvoll abzubilden; eher selten wird die zu modellierende Ausgabegröße linear von den Eingaben abhängen. Mit einer geringen Modifikation läßt sich die Mächtigkeit dieser Modelle allerdings dramatisch aufwerten und auch auf nichtlineare Zusammenhänge ausdehnen. Anstelle der einfachen Inputvektoren $\mathbf{x}_i \in \mathbb{R}^d$ betrachten wir eine erweiterte Repräsentation und operieren nun mit einer transformierten Version $\boldsymbol{\phi}(\mathbf{x}_i) \in \mathbb{R}^m$ der ursprünglichen Inputvektoren $\mathbf{x}_i$. Der Vektor $\boldsymbol{\phi}(\cdot) = \big(\phi_1(\cdot), \ldots, \phi_m(\cdot)\big)^\mathsf{T}$ wird *Merkmalsvektor* oder *Featurevektor* genannt, die m (nichtlinearen) Transformationen $\phi_i : \mathbb{R}^d \to \mathbb{R}$, $i = 1 \ldots m$ werden als *Basisfunktionen* oder *Features* bezeichnet.

Alle bislang betrachteten Algorithmen könne in identischer Weise auch auf die erweiterte Repräsentation mit Merkmalsvektoren angewendet werden; hierzu werden einfach alle auftretenden $\mathbf{x}_i$ durch $\boldsymbol{\phi}(\mathbf{x}_i)$ ersetzt.

Die einfachen linearen Modelle können demnach als Spezialfall mit $\phi_i(\mathbf{x}) = x_i$ verstanden werden. Das aus nichtlinearen Basisfunktionen resultierende Modell ist nicht mehr länger linear in den Eingaben, sondern lediglich linear in den Merkmalen; statt (2.1) hat die gelernte Funktion f die Form

$$f(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) = \mathbf{w}^\mathsf{T} \phi(\mathbf{x}) \tag{2.14}$$

wobei $\mathbf{w} \in \mathbb{R}^m$ der zu bestimmende Gewichtsvektor ist.

Art und Anzahl der Basisfunktionen können frei gewählt werden und bestimmen den Ansatzraum, aus dem die gelernte Funktion f rekrutiert wird. In Abschnitt 1.2.1 hatten wir bereits Beispiele kennengelernt, die auf einer festen Menge von Basisfunktionen/Knoten beruhen, die allesamt *vor* dem Lernen der Funktion ausgewählt wurden. Für unsere Einsatzzwecke ist das aber weniger geeignet, weil die Anzahl der benötigten Basisfunktionen/Knoten im allgemeinen Fall exponentiell mit der Dimension des Eingaberaums anwächst ('curse of dimensionality'). Für unsere Anwendung auf Reinforcement Lernen entwickelt sich das zu einem Problem, da wir es dort mit relativ hochdimensionalen Eingaben zu tun haben werden; die Zustandsvektoren haben Dimensionalität von ca. 10–100 !

Im folgenden betrachten wir daher nichtparametrisierte – genauer gesagt kernbasierte – Methoden, bei denen die Wahl der Basisfunktionen komplett entfällt. Stattdessen müssen wir nur noch einen einzigen Parameter spezifizieren, nämlich die Kovarianzfunktion bzw. den positiv definiten *Kern* (eine symmetrische Funktion $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$), der besagt, wie stark die gesuchte Funktion variiert, d.h. in welchem Ausmaß die Ausgaben von im Eingaberaum benachbarten Punkten variieren dürfen. Prinzipiell betrachtet man also einen Funktionenraum und unendlich viele Basisfunktionen (abhängig vom gewählten Kern). Die zentrale Aussage des Repräsenter-Theorems (Kimeldorf und Wahba, 1971) ist aber, daß eine endliche Repräsentation der Lösung durch die Daten ausreichend ist, d.h. die Lösung in gewisser Weise wieder einem Basisfunktionsnetzwerk mit auf den Daten zentrierten Basisfunktionen entspricht. Formal handelt es sich bei dem Ansatzraum für f um einen Hilbertraum mit reproduzierendem Kern (RKHS); um an dieser Stelle nicht zu sehr vom Kurs abzuweichen, fassen wir diesbezüglich grundlegende Aussagen im Anhang zusammen.

In den folgenden Teilen soll nun die 'kernelisierte' Variante von RR, das sog. Kernel-RR, vorgestellt werden. Um die Analogie mit RR aufzuzeigen, werden wir dabei mit der Darstellung des Kerns als Abbildung in einen (hochdimensionalen) Merkmalsraum arbeiten und die gesuchte Lösung f bezüglich der Eigenfunktionen des Kerns entwickeln, d.h. die Eigenfunktionen als Basisfunktionen auffassen.

2.4.1 Primale Darstellung von Kernel-RR

Seien $\phi_i(\mathbf{x}) := \sqrt{\lambda_i}\varphi_i(\mathbf{x})$, $i = 1, \ldots, d_\mathcal{H}$ die Basisfunktionen, bezüglich derer wir die Approximation

$$f(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{d_\mathcal{H}} w_i \phi_i(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle$$

in gewohnter Weise parametrisieren. Dabei bezeichne $\varphi_i(\mathbf{x})$ die Eigenfunktionen und λ_i die Eigenwerte eines positiv definiten Kerns gemäß seiner Darstellung mit Hilfe von Mercer's Theorem (vgl. Anhang). Weiter sei $d_\mathcal{H}$ die Dimension des Merkmalsraums, die auch unendlich sein kann[3]. Wie gehabt betrachten wir wieder Merkmalsvektoren $\boldsymbol{\phi}(\mathbf{x}) = \big(\phi_1(\mathbf{x}), \ldots, \phi_{d_\mathcal{H}}(\mathbf{x})\big)^\mathsf{T}$ und eine Datenmatrix

$$\boldsymbol{\Phi} = \begin{bmatrix} — \ \boldsymbol{\phi}(\mathbf{x}_1)^\mathsf{T} \ — \\ \vdots \\ — \ \boldsymbol{\phi}(\mathbf{x}_n)^\mathsf{T} \ — \end{bmatrix} = \begin{bmatrix} \sqrt{\lambda_1}\varphi_1(\mathbf{x}_1) & \cdots & \sqrt{\lambda_{d_\mathcal{H}}}\varphi_{d_\mathcal{H}}(\mathbf{x}_1) \\ \vdots & & \vdots \\ \sqrt{\lambda_1}\varphi_1(\mathbf{x}_n) & \cdots & \sqrt{\lambda_{d_\mathcal{H}}}\varphi_{d_\mathcal{H}}(\mathbf{x}_n) \end{bmatrix}$$

Ist $d_\mathcal{H} > n$ [4], so ist es natürlich möglich, für jede endliche Menge von Trainingsdaten eine perfekte Interpolation zu erzielen; das Residuum $J(\hat{\mathbf{w}})$ von gewöhnlicher Least-Squares Approximation (2.2) wäre

[3] Das ist beispielsweise dann der Fall, wenn wir als Kern Gauß'sche Radiale Basisfunktionen verwenden.

[4] Was beinahe immer der Fall ist, denn schließlich betrachten wir Merkmalsräume aus genau dem Grund, Probleme die im Eingaberaum nicht linear lösbar sind, durch Abbildung in einen hochdimensionalen Merkmalsraum linear lösbar zu machen.

in diesem Fall exakt Null. Aus den bereits genannten Gründen wäre das Resultat davon für Vorhersagen jedoch wenig geeignet (vgl. Abbildung 2.2). Daher betrachten wir wieder die regularisierte Variante RR und formulieren analog zu (2.11) das quadratische Problem (dieses mal im Merkmalsraum)

$$\min_{\mathbf{w}} \; J_\lambda(\mathbf{w}) := \|\mathbf{y} - \mathbf{\Phi}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2 \tag{2.15}$$

Die Lösung $\hat{\mathbf{w}}$ erhalten wir durch Differentiation von (2.15) und analog zu (2.12) als

$$\hat{\mathbf{w}} = (\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\mathbf{\Phi}^\mathsf{T}\mathbf{y}. \tag{2.16}$$

2.4.2 Duale Darstellung von Kernel-RR

In der Praxis wird es ungünstig sein, die Lösung $\hat{\mathbf{w}}$ in der Form (2.16) auszurechnen, weil es sich dabei um ein $d_\mathcal{H} \times d_\mathcal{H}$ Problem handelt. Mit Hilfe des in der Literatur als 'Kernel-Trick' bezeichneten Schrittes können wir allerdings die Aufgabenstellung in (2.15), die im Augenblick noch explizit von der Dimension des Merkmalsraums abhängt, in eine duale Aufgabe umwandeln, die dann nur noch von der Anzahl der Daten abhängt. Alternativ könnte man auch mittels Nyström-Approximation[5] aus den Daten eine Näherung für $\boldsymbol{\phi}(\mathbf{x})$ erzeugen, womit sich das Problem auch in der primalen Form (2.15) lösen ließe (vgl. Williams und Seeger, 2001; Williams et al., 2002; Hoegarts et al., 2005).

Duale Variablen

Wenden wir die Sherman-Morrison-Woodbury Identität (B.3)[6] mit $\mathbf{A} := \lambda\mathbf{I}_{d_\mathcal{H}}$, $\mathbf{B} := \mathbf{\Phi}^\mathsf{T}$, $\mathbf{C} := \mathbf{\Phi}$ und $\mathbf{D} := \mathbf{I}_n$ auf die Matrix in (2.16) an, so kann diese äquivalent geschrieben werden als

$$(\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi} + \lambda\mathbf{I}_{d_\mathcal{H}})^{-1}\mathbf{\Phi}^\mathsf{T} = \lambda^{-1}\mathbf{I}_{d_\mathcal{H}}\mathbf{\Phi}^\mathsf{T}(\mathbf{\Phi}\lambda^{-1}\mathbf{I}_{d_\mathcal{H}}\mathbf{\Phi}^\mathsf{T} + \mathbf{I}_n)^{-1} = \mathbf{\Phi}^\mathsf{T}(\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T} + \lambda\mathbf{I}_n)^{-1}$$

Der Unterschied ist nun, daß auf der linken Seite die Inverse einer $d_\mathcal{H} \times d_\mathcal{H}$ Matrix berechnet werden muß, während wir es auf der rechten Seite lediglich mit einer $n \times n$ Matrix zu tun haben. Demnach ist (2.16) äquivalent zu

$$\hat{\mathbf{w}} = \mathbf{\Phi}^\mathsf{T}(\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T} + \lambda\mathbf{I})^{-1}\mathbf{y}.$$

Definieren wir nun duale Variablen $\hat{\boldsymbol{\alpha}} := (\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T} + \lambda\mathbf{I})^{-1}\mathbf{y}$, dann gilt $\hat{\mathbf{w}} = \mathbf{\Phi}^\mathsf{T}\hat{\boldsymbol{\alpha}}$. Somit kann die Lösung $\hat{\mathbf{w}}$ des primalen Problems (2.15) als Linearkombination der Zeilen von $\mathbf{\Phi}$, den Merkmalsvektoren $\boldsymbol{\phi}(\mathbf{x}_i)$ der einzelnen Daten, ausgedrückt werden, d.h.

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \hat{\alpha}_i \boldsymbol{\phi}(\mathbf{x}_i).$$

Für die gelernte Funktion $f(\cdot\,; \hat{\mathbf{w}}) = \langle \boldsymbol{\phi}(\cdot), \hat{\mathbf{w}} \rangle$ gilt folglich für beliebige Punkte $\mathbf{x}$ die Darstellung

$$f(\mathbf{x}; \hat{\boldsymbol{\alpha}}) = \sum_{i=1}^{n} \hat{\alpha}_i \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}_i) \rangle. \tag{2.17}$$

Der Kernel-Trick

Jetzt kommt der entscheidende Schritt: alle auftretenden Skalarprodukte zwischen Merkmalsvektoren können durch den Kern ersetzt werden; es gilt $\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle = k(\mathbf{x}, \mathbf{x}')$ für alle $\mathbf{x}, \mathbf{x}'$ (siehe Anhang). Die Matrix $\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T}$ besteht aus den Skalarprodukten der Daten im Merkmalsraum untereinander und kann durch die $n \times n$ Kernmatrix $\mathbf{K}$ mit $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ ersetzt werden. Weiter wird aufgrund von (2.17) nun

[5] Indem die kontinuierliche Eigenwertgleichung $\int k(\mathbf{x}, \mathbf{y})\varphi_i(\mathbf{x})d\mathbf{x} = \lambda_i\varphi_i(\mathbf{y})$ durch $\frac{1}{\ell}\sum_{\nu=1}^{\ell} k(\mathbf{x}_\nu, \mathbf{x}_j)\varphi_i(\mathbf{x}_\nu) = \lambda_i\varphi_i(\mathbf{x}_j)$ diskretisiert wird (in Stützstellen $\{\mathbf{x}_i\}_{i=1}^{\ell}$).

[6] Es gilt: $(\mathbf{A} + \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1} = \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}$

klar, daß die primalen Gewichte $\hat{\mathbf{w}}$ überhaupt nicht mehr gebraucht werden, um die gelernte Funktion darzustellen; die dualen Variablen $\hat{\boldsymbol{\alpha}}$ reichen hierfür aus.

Anstelle des primalen Problems (2.15) in den $d_{\mathcal{H}}$ Unbekannten w_i formulieren wir daher mit $\mathbf{w} = \boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\alpha}$ und $\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathsf{T}} = \mathbf{K}$ das duale Problem in n Unbekannten α_i:

$$\min_{\boldsymbol{\alpha}} \; J_\lambda(\boldsymbol{\alpha}) := \|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \lambda \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{K}\boldsymbol{\alpha} \tag{2.18}$$

Die Lösung $\hat{\boldsymbol{\alpha}}$ erhalten wir, weil $\mathbf{K}$ positiv definit und damit invertierbar ist, nach Differentiation von (2.18) als

$$\hat{\boldsymbol{\alpha}} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}. \tag{2.19}$$

Schließlich kann nach (2.17) die gelernte Funktion allein durch $\hat{\boldsymbol{\alpha}}$ ausgedrückt werden

$$f(\mathbf{x}; \hat{\boldsymbol{\alpha}}) = \sum_{i=1}^{n} \hat{\alpha}_i k(\mathbf{x}_i, \mathbf{x}). \tag{2.20}$$

Anschaulich kann man sich Kernel-RR aber auch wieder als ein Basisfunktionsnetzwerk vorstellen, bei dem n Basisfunktionen $k(\mathbf{x}_i, \cdot\,)$ auf den Daten plaziert werden.

Beispiel: Datensatz 'Sinc'

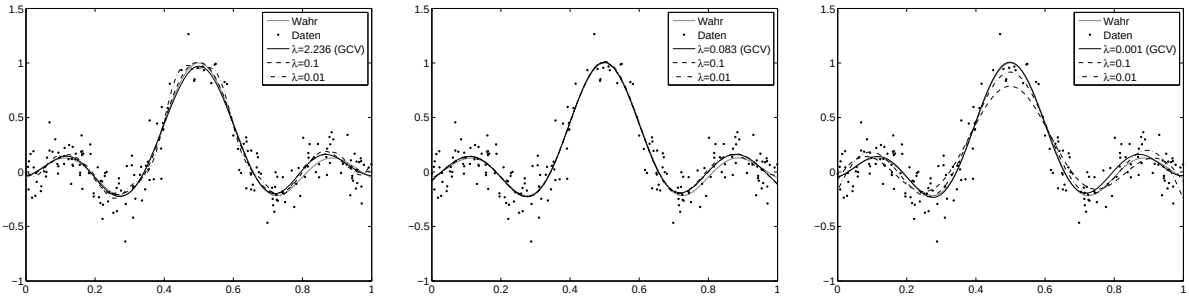
Illustrieren wir das Lernen einer Funktion anhand eines einfachen Beispiels. Gegeben seien 200 Stichproben einer (unbekannten) Funktion g, wobei hier $g(x) = \sin|x|/|x|$ gewählt wurde. Die Beobachtungen $y_i = g(x_i) + \varepsilon_i$ seien mit einem Fehler $\varepsilon_i \sim \mathcal{N}(0, 0.1)$ versehen, und die x_i seien zufällig gewählt mit $x_i \sim \mathcal{U}[-10, 10]$. Als Kern verwenden wir, wie in der Literatur weitestgehend üblich, die Gauß'schen Radialen Basisfunktionen (siehe Diskussion zur Wahl des Kerns im Anhang)

$$k(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{h}\right\}$$

mit uniformer Varianz $h/2$. Wir wollen nun demonstrieren, wie sich verschiedene Einstellungen der Metaparameter auf die durch Kernel-RR gelernte Funktion f auswirken. In diesem Fall sind das der Kernparameter h und der Regularisierungsparameter λ. Wir starten in Abbildung 2.5 (links) mit $h = 0.01$ (diesen Wert findet man in der Literatur häufig als den für dieses Testproblem günstigen Parameter) und $\lambda = 2.235$ (bestimmt durch Minimieren von GCV(λ)), sowie zum Vergleich $\lambda = 0.1$ und $\lambda = 0.01$ (manuell gewählte Erfahrungswerte ohne Parameterwahlstrategie). In Abbildung 2.5 (mitte und rechts) wiederholen wir das für $h = 0.05$ und $h = 0.1$; das zugehörige λ wird jeweils erneut mit GCV bestimmt. Auf einer unabhängigen Testmenge von 100 nicht-gestörten Beispiele von g erhalten wir die folgenden Vorhersagefehler (gemessen als MSE):

	$h = 0.1$	$h = 0.05$	$h = 0.01$
$\lambda = 0.1$	0.007452	0.000482	0.000901
$\lambda = 0.01$	0.002623	0.000559	0.002576
GCV	0.000492	0.000476	0.000656
	($\lambda = 0.00111$)	($\lambda = 0.08251$)	($\lambda = 2.23577$)

Prinzipiell ist die Wahl der richtigen Metaparameter immer eine komplizierte Angelegenheit, weil die Metaparameter in ihrer Wirkung aneinander gekoppelt sind, und daher nicht unabhängig voneinander, sondern gemeinsam bestimmt werden müssen. Möglichkeiten dazu bieten uns Optimierungsverfahren wie GCV oder Typ-II Maximum Likelihood, oder Kreuzvalidierung und gitterbasierte Suche; was in jedem Fall eine sehr aufwendige Prozedur darstellt. In dieser Arbeit werden wir Parameterwahl daher nur am Rande diskutieren und insbesondere bei der späteren Anwendung auf Reinforcement Lernen den aufwendigeren Optimierungsverfahren grobe gitterbasierte Suche oder schlichtes Ausprobieren vorziehen.



(a) $h = 0.01$ (b) $h = 0.05$ (c) $h = 0.1$

Abbildung 2.5: Lernen der Funktion $g(x) = \sin|x|/|x|$. Gezeigt wird die mit Kernel-RR aus fehlerhaften Beobachtungen (schwarze Kreise) rekonstruierte Funktion für unterschiedliche Einstellungen der Metaparameter h und λ.

Hinter den Kulissen: Kernel-RR

Wie kann man sich die Wirkung von Regularisierung im Falle von Kernel-RR vorstellen? Ähnlich wie bei gewöhnlichem RR so werden auch bei Kernel-RR der Einfluß bestimmter 'komplizierter' Anteile an der Lösung gedämpft. Sei $\mathbf{\Phi} = \mathbf{U\Sigma V}^\mathsf{T}$ die SVD von $\mathbf{\Phi}$, so daß $\mathbf{K} = \mathbf{\Phi\Phi}^\mathsf{T}$ die Eigenwertzerlegung $\mathbf{K} = \mathbf{U\Sigma}^2\mathbf{U}^\mathsf{T}$ hat. Der Vektor $\hat{\mathbf{y}}$ der Vorhersagen in den Trainingsdaten geht mit dem gelernten Modell $f(\cdot\,; \hat{\boldsymbol{\alpha}})$ nach (2.20) aus den Beobachtungen $\mathbf{y}$ mittels

$$\hat{\mathbf{y}} = \mathbf{K}(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

hervor. Sei wie zuvor $\mathbf{S}_\lambda := \mathbf{K}(\mathbf{K}+\lambda\mathbf{I})^{-1}$. Setzen wir die Eigenwertzerlegung von $\mathbf{K}$ in $\mathbf{S}_\lambda$ ein, so erhalten wir $\mathbf{S}_\lambda = \mathbf{K}(\mathbf{K} + \lambda\mathbf{I})^{-1} = \mathbf{U\Sigma}^2\mathbf{U}^\mathsf{T}(\mathbf{U\Sigma}^2\mathbf{U}^\mathsf{T} + \lambda\mathbf{I})^{-1} = \mathbf{U\Sigma}^2(\mathbf{\Sigma}^2 + \lambda\mathbf{I})^{-1}\mathbf{U}^\mathsf{T}$ und wegen $\hat{\mathbf{y}} = \mathbf{S}_\lambda\mathbf{y}$ wieder

$$\hat{\mathbf{y}} = \sum_{i=1}^{n} f_i(\mathbf{u}_i^\mathsf{T}\mathbf{y})\mathbf{u}_i,$$

wobei $f_i = \sigma_i^2/(\sigma_i^2+\lambda)$. Wie zuvor gilt $f_i \to 1$ für $\lambda \ll \sigma_i$ und $f_i \to 0$ für $\lambda \gg \sigma_i$. Das sieht soweit genauso wie RR aus, doch welche Bedeutung haben die $\mathbf{u}_i$ im Falle von Kernel-RR? Bei RR entsprachen die $\mathbf{u}_i$ den PCA-Richtungen und die Dämpfung durch f_i war besonders groß entlang den Richtungen kleiner Varianz der Daten. In gewissem Sinne wurde die Gesamtlösung also bevorzugt aus 'einfachen' Bauteilen, den Hauptrichtungen der Daten, zusammengesetzt.

Ähnlich verhält es sich auch bei Kernel-RR. Man kann zeigen, daß die Eigenvektoren $\mathbf{u}_i$ der Kernmatrix gegen die Eigenfunktionen $\varphi_i(\cdot)$ des Kerns konvergieren, wenn diese in den n Datenpunkten ausgewertet werden (im Sinne von $\mathbf{u}_i^{(j)} \approx \sqrt{n}\varphi_i(\mathbf{x}_j)$ für $j = 1, \ldots, n$, vgl. Williams und Seeger (2001); Hoegarts et al. (2005), wo dies mittels Nyström-Approximation gezeigt wird). Auch die Eigenfunktionen können anschaulich bezüglich ihrer 'Komplexität' charakterisiert werden: typischerweise sind die zu den ersten paar Eigenwerten gehörenden Eigenfunktionen solche, die langsam variieren und wenig Nullstellen haben, während die Eigenfunktionen zu den kleinen Eigenwerten stark oszillieren und viele Nullstellen haben.

In den Abbildungen 2.6 und 2.7 zeigen wir dies zur Illustration für das konkrete Beispiel aus Abbildung 2.5: wir plotten zunächst das Eigenspektrum des Gauß-Kerns für $h = 0.1$, $h = 0.05$, und $h = 0.01$ (mitte), die zugehörige Dämpfung durch die Filterkoeffizienten f_i für das durch jeweils mit GCV berechnete λ (rechts), und dann die Eigenfunktionen zu den ersten 12 Eigenwerten (absteigend geordnet), sowie deren durch f_i gedämpfte Version, aus der schließlich die Lösung aufgebaut wird (Abbildung 2.7). Zu beachten ist, daß wir die Eigenwerte/Eigenfunktionen näherungsweise durch die Daten des 'Sinc'-Beispiels mit der Nyström-Approximation berechnen und die gezeigten Eigenwerte in den Abbildungen daher nicht exakt sind[7]. Zusammengenommen sehen wir nun folgendes: die Eigenwerte des Gauß-Kerns fallen allgemein

[7] In Shawe-Taylor et al. (2005) wird eine exakte Formel für die Eigenzerlegung des Gaußkerns angegeben. Es zeigt sich, daß die Eigenwerte wie c^k abfallen, mit k deren Index und $c < 1$. In unserer Abbildung dagegen ist das aufgrund der Approximation für Eigenwerte ab einem Index von ca. 20-50 schon nicht mehr der Fall.

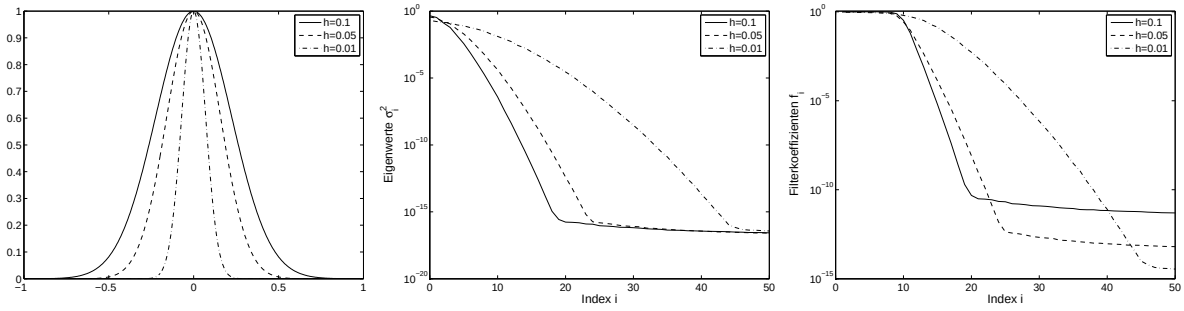


Abbildung 2.6: Links: Gauß-Kern für verschiedene Längenskalen h. Mitte: Eigenspektrum des Gauß-Kerns für verschiedene Längenskalen h. Rechts: Die entsprechende Dämpfung durch die Filterkoeffizienten für verschiedene Werte von h und mit GCV bestimmten λ (vgl. Abb. 2.5).

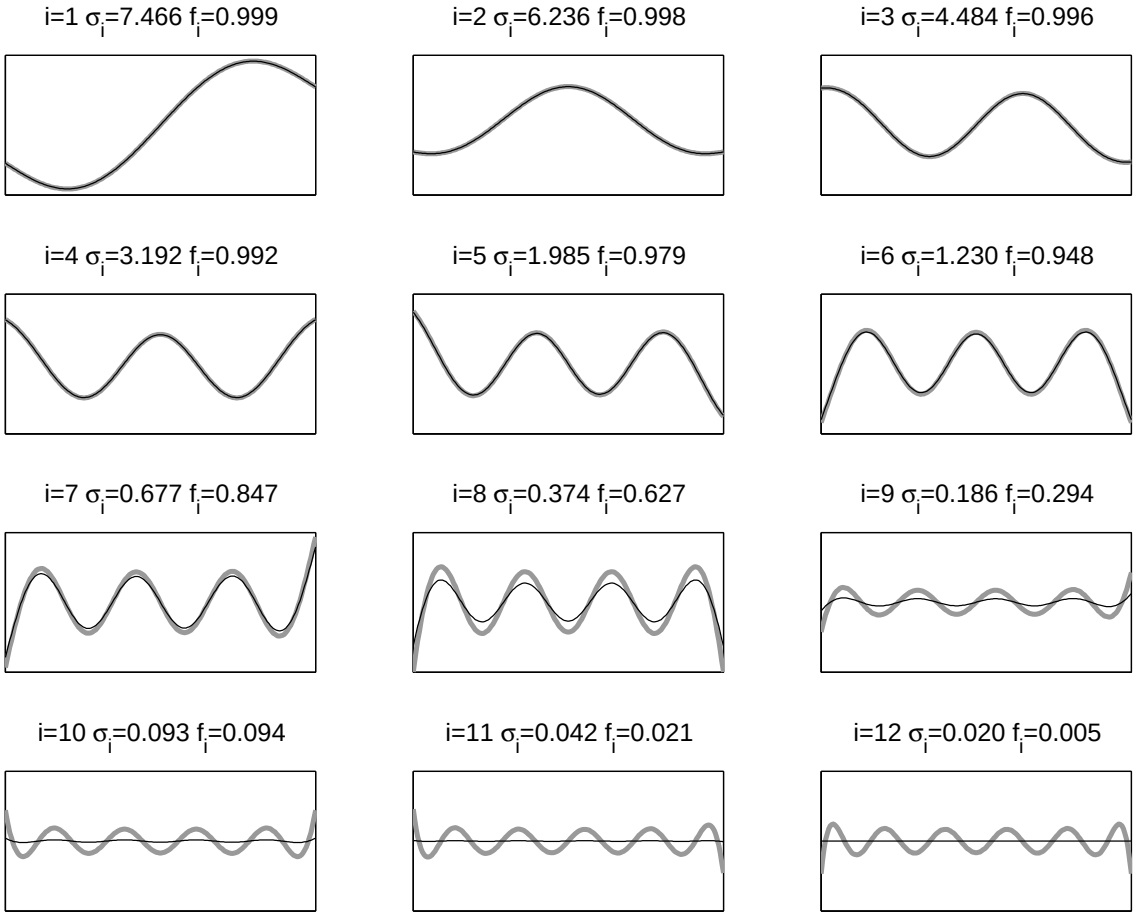


Abbildung 2.7: Die ersten 12 Eigenfunktionen des Gauß-Kerns für $h = 0.05$, geordnet nach absteigenden Eigenwerten und aufsteigender Komplexität. Gezeigt werden die jeweilige Eigenfunktion (graue Kurve) sowie die mit $\lambda = 0.083$ durch f_i gedämpfte Version (schwarze Kurve), korrespondierend zur GCV-Lösung Abb. 2.5 (mitte).

schnell ab, für den Fall $h = 0.1$ (d.h. der relativ 'breiten' Gaußglocke) noch schneller als für $h = 0.01$ (der 'spitzen' Gaußglocke). Die zu den Eigenwerten gehörenden Eigenfunktionen nehmen mit wachsendem Index in ihrer Komplexität zu (oszillieren schneller). Anhand der Filterkoeffizienten sehen wir dann, wie mit Regularisierung der Beitrag von Eigenfunktionen hoher Komplexität (korrespondiert zu Eigenwerten mit großem Index) gedämpft wird: die ersten 6 Eigenfunktionen werden beinahe vollständig übernommen, während alle Eigenfunktionen über dem Index 12 kaum noch Einfluß auf die Lösung haben ($f_i < 0.05$). Tendenziell wird eine Eigenfunktion also um so stärker gedämpft, je komplizierter sie ist; die zu lernende Funktion wird also bevorzugt aus einfachen Bauteilen zusammensetzt. Der genaue Umfang der Dämpfung hängt vom Abfallen des Eigenspektrums (und damit der gewählten Breite h des Gauß-Kerns) sowie dem Regularisierungsparameter λ ab: in Abbildung 2.6 sehen wir, daß das unterschiedlich starke Abfallen der Eigenwerte für die unterschiedliche Wahl von h durch die Wahl von λ kompensiert werden kann und die resultierende Dämpfung in dem relevanten Bereich (Index 1-12) nahezu identisch ist.

2.4.3 Alternative Herleitung und Literatur

Neben der hier betrachteten Herleitung von Regularisierungsnetzen (RN) als 'kernelisierte' Variante von Ridge Regression findet man RN in der Literatur häufig auch anders motiviert:

1. **Tikhonov-Regularisierung:** Das Lernen einer Funktion aus Daten $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ wird als Variationsproblem $\min_f \sum_i (y_i - f(\mathbf{x}_i))^2 + \lambda\Omega[f]$ formuliert. Der erste Term mißt den Approximationsfehler (und erzwingt Nähe zu den Daten), der zweite Term – in der Regularisierungstheorie Stabilisator genannt – bestraft gleichzeitig die Komplexität der Lösung (Tikhonov und Arsenin, 1977; Poggio und Girosi, 1989). Die Wahl von $\Omega[f]$ bestimmt das Aussehen der Funktion: für $\Omega[f] = \int (f''(x))^2 dx$ erhalten wir beispielsweise das Verfahren Smoothing Splines, die Basisfunktionen sind kubische Splines, wobei die Knotenmenge die Menge der Daten ist. Für $\Omega[f] = \|f\|_k^2$, der Norm im RKHS induziert durch eine positiv definite Funktion $k(\mathbf{x}, \mathbf{x}')$, kann allgemein gezeigt werden (Girosi et al., 1995), daß die Funktion f, die obiges Problem minimiert, von der Form $f(\mathbf{x}) = \sum_i c_i k(\mathbf{x}, \mathbf{x}_i)$ ist, und die Koeffizienten c_i aus den Daten durch Lösen des Gleichungssystems $(\mathbf{K} + \lambda\mathbf{I})\mathbf{c} = \mathbf{y}$ hervorgehen – also genau Gleichung (2.19) bei unserer Herleitung.

2. **Bayes'sche Regression:** Eine alternative Motivation für (Kernel)-RR ergibt sich mit dem Ansatz Bayes'scher Inferenz. Die gesuchte Funktion wird dabei nicht als Punktschätzung konstruiert, sondern über eine Wahrscheinlichkeitsverteilung (über den Gewichtsvektoren $\mathbf{w}$) modelliert. Man startet als Anfangshypothese (Apriori-Verteilung) für $\mathbf{w}$ mit einer Normalverteilung. Ausgehend von einem Modell der Form (2.5) (mit ebenfalls Normalverteilten Rauschen) für das Zustandekommen der Daten wird dann die Verteilung für $\mathbf{w}$ aufgrund der tatsächlich beobachteten Daten adaptiert: es gilt $p(\mathbf{w}|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})$ (Satz von Bayes), wobei $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ die Likelihood aus (2.6) ist. Das Produkt zweier Normalverteilungen ist wieder eine Normalverteilung, so daß die Aposteriori-Verteilung $p(\mathbf{w}|\mathbf{X}, \mathbf{y})$ ebenfalls eine Normalverteilung bleibt. Man kann nun zeigen, daß der Erwartungswert dieser Verteilung gerade die Lösung $\hat{\mathbf{w}}$ von RR ist, und der Quotient aus Varianz des Rauschens und Varianz des Priors dem Regularisierungsparameter λ entspricht. Für weitergehende Informationen und die Verallgemeinerung mit Kernen siehe Rasmussen und Williams (2006); das resultierende Lernverfahren ist als Gauß-Prozeß Regression (GP-Regression) bekannt. Soweit es diese Arbeit anbelangt, kann GP-Regression als technisch äquivalent zu Kernel-RR mit einem Gauß-Kern angesehen werden. Darüber hinaus bietet GP-Regression allerdings eine Reihe von konzeptionellen Vorteilen; etwa indem sie es erlaubt, die notwendigen Metaparameter (Varianz des Rauschens, Kernparameter) automatisch für einen vorliegenden Datensatz mittels Typ-II Maximum Likelihood und Automatic Relevance Determination einzustellen. Allerdings ist dazu zusätzlich ein aufwendiges Optimierungsproblem zu lösen, weswegen wir diesen Ansatz nicht für Online-Lernen verfolgen können.

Kapitel 3

Hintergrund: Methoden zur Approximation (offline)

Worin beschrieben wird, wie die zuvor formulierte Aufgabe für RN praktisch gelöst werden kann

Wie wir gesehen haben, laufen RN auf eine Parametrisierung der gesuchten Funktion f der Form

$$f(\cdot\,;\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot\,)$$

hinaus, wobei die Koeffizienten α_i bestimmt werden als Lösung von

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \; \|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \lambda \boldsymbol{\alpha}^\mathsf{T}\mathbf{K}\boldsymbol{\alpha}.$$

Unsere Hauptsorge wird es jetzt sein, selbige tatsächlich auch auszurechnen. Im allgemeinen Fall ist $\mathbf{K}$ eine vollbesetzte $n \times n$ Matrix, so daß der benötigte Aufwand ungünstig in der Anzahl n der Daten anwächst: der Rechenaufwand zum Lösen des Gleichungssystems beträgt $\mathcal{O}(n^3)$, der Speicherbedarf $\mathcal{O}(n^2)$ und am Ende wird jede einzelne Voraussage mit f $\mathcal{O}(n)$ Operationen kosten. Ab einer hinreichend großen Menge von Daten (z.B. ab $n > 5000$) wird das Lösen auf naive Weise so natürlich nicht mehr funktionieren.

Im folgenden beschreiben wir daher Methoden, mit denen der Kern, bzw. die Kernmatrix, aus einer Teilmenge der Daten (mit Kardinalität $m \ll n$) approximiert werden kann. Mit Hilfe dieser Approximation kann ein reduziertes Problem für RN formuliert werden, durch das die erforderlichen Kosten erheblich gesenkt werden können: der Rechenaufwand sinkt auf $\mathcal{O}(nm^2)$, der Speicherbedarf auf $\mathcal{O}(m^2)$ und jede zukünftige Vorhersage mit f wird dann nur noch $\mathcal{O}(m)$ Operationen kosten.

3.1 Rang-reduzierte Approximation von K

Starten wir mit der Motivation:

Warum hilft uns niedrigerer Rang?

Um den Gewichtsvektor $\boldsymbol{\alpha}$ zu bestimmen, müssen wir die Inverse von $(\mathbf{K} + \lambda\mathbf{I})$ bestimmen (oder das Gleichungssystem $(\mathbf{K}+\lambda\mathbf{I})\boldsymbol{\alpha} = \mathbf{y}$ anderweitig lösen). Hat $\mathbf{K}$ Rang q ($q < n$), dann kann $\mathbf{K}$ als $\mathbf{K} = \mathbf{Q}\mathbf{Q}^\mathsf{T}$ mit einer $n \times q$ Matrix $\mathbf{Q}$ geschrieben werden. Aufgrund von (B.2)[1] dürfen wir dann $(\mathbf{K} + \lambda\mathbf{I})^{-1}$ durch

$$(\mathbf{K} + \lambda\mathbf{I})^{-1} = (\mathbf{Q}\mathbf{I}_q\mathbf{Q}^\mathsf{T} + \lambda\mathbf{I}_n)^{-1} = \lambda^{-1}\mathbf{I}_n - \lambda^{-1}\mathbf{Q}(\lambda\mathbf{I}_q + \mathbf{Q}^\mathsf{T}\mathbf{Q})^{-1}\mathbf{Q}^\mathsf{T} \tag{3.1}$$

[1] Es gilt: $(\mathbf{A} + \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1}$. Mit $\mathbf{A} := \lambda\mathbf{I}_n$, $\mathbf{B} := \mathbf{Q}$, $\mathbf{D}^{-1} := \mathbf{I}_q$ und $\mathbf{C} := \mathbf{Q}^\mathsf{T}$ folgt die Behauptung.

ersetzen. Auf diese Weise wird die Inversion einer $n \times n$ Matrix in die Inversion einer $q \times q$ Matrix[2] umgewandelt.

Woher bekommen wir niedrigen Rang?

Allerdings wird für viele Kerne (so wie den hier betrachteten Gauß'schen RBF Kernen) die Kernmatrix $\mathbf{K}$ vollen Rang haben. Es besteht aber die Hoffnung, daß aufgrund der besonderen Glattheit der Kernfunktion die Matrix $\mathbf{K}$ zu gewissem Grad schnell abfallende Eigenwerte hat und daher beinahe ebensogut durch eine rangreduzierte Approximation repräsentiert werden könnte. Die bezüglich der Frobeniusnorm ($\|\mathbf{A}\|_F^2 := \sum_{i,j} |a_{ij}|^2$) optimale Rang-$q$ Approximation $\tilde{\mathbf{K}}$ von $\mathbf{K}$ ist

$$\tilde{\mathbf{K}} = \mathbf{U}_q \mathbf{\Lambda}_q \mathbf{U}_q^\mathsf{T} \tag{3.2}$$

wobei $\mathbf{\Lambda}_q$ die diagonale Matrix der q führenden Eigenwerte von $\mathbf{K}$ und $\mathbf{U}_q$ die Matrix der zugehörigen orthonormalen Eigenvektoren ist (Golub und Van Loan, 1996). Zunächst hilft uns das aber wenig weiter, weil das Berechnen der Eigenwertzerlegung ebenfalls eine $\mathcal{O}(n^3)$ Angelegenheit ist.

Allerdings kann man versuchen, die Eigenwerte und Eigenvektoren näherungsweise zu bestimmen und in (3.2) einzusetzen; in (Williams und Seeger, 2001) wird ein Verfahren beschrieben, das nur die Eigenwertzerlegung einer viel kleineren Teilmatrix vornimmt und dann mittels Nyström-Approximation[3] eine Näherung für die gesamte Eigenwertzerlegung berechnet.[4]

Approximation des Kerns durch Teilmengen

In Smola und Schölkopf (2000) wird eine zur Nyström-Approximation alternative, für die nachfolgenden Kapitel aber nützlichere Sichtweise beschrieben, die am Ende auf exakt dieselbe Approximation für $\mathbf{K}$ hinausläuft. Sei $\mathcal{D} := \{\tilde{\mathbf{x}}_i\}_{i=1}^m$ eine m-elementige Auswahl aller Trainingsdaten $\{\mathbf{x}_i\}_{i=1}^n$ (um einen doppelten Index zu vermeiden, markieren wir ausgewählte Elemente mit einer Tilde). Für jedes beliebige Argument $\mathbf{x}$ wollen wir nun versuchen, die Kernfunktion $k(\mathbf{x}, \cdot\,)$ durch Linearkombination der Kerne für $\tilde{\mathbf{x}}_i$ zu approximieren[5]:

$$k(\mathbf{x}, \cdot\,) \approx \sum_{i=1}^m a_i k(\tilde{\mathbf{x}}_i, \cdot\,).$$

Um geeignete Koeffizienten a_i zu bestimmen, minimieren wir im RKHS die Länge des Residuums:

$$\min_{\mathbf{a} \in \mathbb{R}^m} \left\| k(\mathbf{x}, \cdot\,) - \sum_{i=1}^m a_i k(\tilde{\mathbf{x}}_i, \cdot\,) \right\|_{\mathcal{H}}^2. \tag{3.3}$$

Verwendet man die Äquivalenz von Norm und Skalarprodukt, dann zeigt sich, daß

$$\delta_{\mathbf{x}} := \left\| k(\mathbf{x}, \cdot\,) - \sum_{i=1}^m a_i k(\tilde{\mathbf{x}}_i, \cdot\,) \right\|_{\mathcal{H}}^2$$

gerade gleich

$$\langle k(\mathbf{x}, \cdot\,), k(\mathbf{x}, \cdot\,) \rangle_{\mathcal{H}} - 2 \sum_i a_i \langle k(\mathbf{x}, \cdot\,), k(\tilde{\mathbf{x}}_i, \cdot\,) \rangle_{\mathcal{H}} + \sum_{i,j} a_i a_j \langle k(\tilde{\mathbf{x}}_i, \cdot\,), k(\tilde{\mathbf{x}}_j, \cdot\,) \rangle_{\mathcal{H}}$$

[2] Numerisch ist das sicherlich nicht die günstigeste Möglichkeit, aber wir erwähnen das hier um die erzielbaren Einsparungen deutlich zu machen. Eine unter diesem Gesichtspunkt bessere Alternative wäre die Cholesky-Zerlegung, wie von Fine und Scheinberg (2001) beschrieben.

[3] Vgl. Fußnote auf Seite 38

[4] Dieselbe Approximation wurde auch in (Fowkles et al., 2004) auf ein Problem aus dem Bereich Computer-Vision angewendet, um die ersten paar Eigenvektoren von sehr großen Matrizen ($> 10^6 \times 10^6$) auszurechnen.

[5] Genausogut könnten wir auch die Bilder im Merkmalsraum approximieren:

$$\boldsymbol{\phi}(\mathbf{x}) \approx \sum_{i=1}^m a_i \boldsymbol{\phi}(\tilde{\mathbf{x}}_i)$$

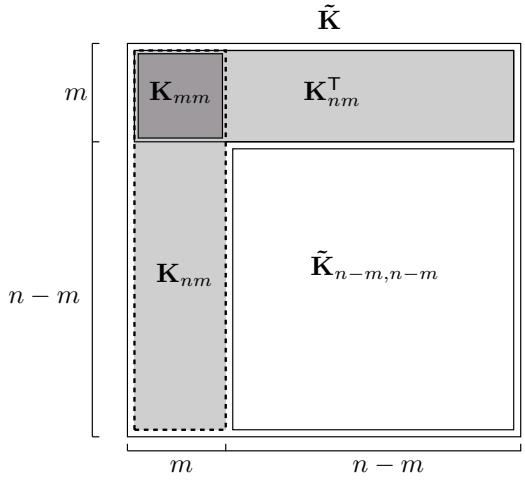


Abbildung 3.1: Approximation des Kerns $k(\mathbf{x}, \mathbf{x}')$ durch eine m-elementige Teilmenge liefert eine Rang-m Approximation $\tilde{\mathbf{K}}$ der Kernmatrix $\mathbf{K}$. Nur der Block $\tilde{\mathbf{K}}_{n-m,n-m}$ enthält approximierte Werte, alle anderen sind exakt.

ist, was wegen (A.1)[6] geschrieben werden kann als

$$k(\mathbf{x}, \mathbf{x}) - 2\mathbf{a}^{\mathsf{T}}\mathbf{k}_m(\mathbf{x}) + \mathbf{a}^{\mathsf{T}}\mathbf{K}_{mm}\mathbf{a}, \tag{3.4}$$

wobei zur Abkürzung der $m \times 1$ Vektor $\mathbf{k}_m(\mathbf{x}) := \big(k(\mathbf{x}, \tilde{\mathbf{x}}_1), \ldots, k(\mathbf{x}, \tilde{\mathbf{x}}_m)\big)^{\mathsf{T}}$ und die $m \times m$ Matrix $\mathbf{K}_{mm}$ mit $[\mathbf{K}_{mm}]_{ij} := k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ definiert wurden. Bilden wir die Ableitung von (3.4) bezüglich $\mathbf{a}$ und setzen das Resultat gleich Null, so erhalten wir für das gesuchte Minimum von (3.3) den Ausdruck

$$\mathbf{a} = \mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}). \tag{3.5}$$

Setzen wir die Lösung von (3.5) in (3.4) ein, so ergibt sich für die Länge des Residuums

$$\delta_{\mathbf{x}} = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}_m(\mathbf{x})^{\mathsf{T}}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}). \tag{3.6}$$

Insgesamt erhalten wir aus Symmetriegründen für beliebige $\mathbf{x}, \mathbf{x}'$ den approximierten Kern

$$k(\mathbf{x}, \mathbf{x}') \approx \sum_i a_i k(\tilde{\mathbf{x}}_i, \mathbf{x}') = \mathbf{a}^{\mathsf{T}}\mathbf{k}_m(\mathbf{x}') = \mathbf{k}_m(\mathbf{x})^{\mathsf{T}}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}') \tag{3.7}$$

und weiter auch eine Rang-m Approximation $\tilde{\mathbf{K}}$ der gesamten Kernmatrix $\mathbf{K}$ mittels

$$\tilde{\mathbf{K}} = \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^{\mathsf{T}} \tag{3.8}$$

wobei

$$\mathbf{K}_{nm} := \begin{bmatrix} — \ \mathbf{k}_m(\mathbf{x}_1)^{\mathsf{T}} \ — \\ \vdots \\ — \ \mathbf{k}_m(\mathbf{x}_n)^{\mathsf{T}} \ — \end{bmatrix}.$$

Ist darüberhinaus $\mathbf{x}$ ein Element der Teilmenge $\mathcal{D}$, dann folgt aus (3.5) gerade $\mathbf{a} = \mathbf{e}_i$, wobei $\mathbf{e}_i$ der i.te Einheitsvektor und $1 \leq i \leq m$ der Index von $\mathbf{x}$ in $\mathcal{D}$ ist. Folglich ist die Approximation in (3.7) exakt, sobald eines der beiden Argumente von $k(\mathbf{x}, \mathbf{x}')$ zur Teilmenge $\mathcal{D}$ gehört.

Die approximierte Kernmatrix $\tilde{\mathbf{K}}$ hat eine besondere Struktur, die in Abbildung 3.1 graphisch veranschaulicht wird. Der Einfachheit halber nehmen wir dazu an, daß die in $\mathcal{D}$ gewählten Daten gerade die ersten m Daten waren (ansonsten ordnen wir sie entsprechend um). Sowohl die $m \times m$ Matrix $\mathbf{K}_{mm}$ (die Kernmatrix der Daten in $\mathcal{D}$), als auch die $n \times m$ Matrix $\mathbf{K}_{nm}$ enthalten die exakten Werte und sind daher Teilmatrizen von $\mathbf{K}$. Lediglich der große $(n-m) \times (n-m)$ Block $\tilde{\mathbf{K}}_{n-m,n-m} = \mathbf{K}_{n-m,m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{n-m,m}^{\mathsf{T}}$ enthält approximierte Werte. Die Differenz $\mathbf{K}_{n-m,n-m} - \tilde{\mathbf{K}}_{n-m,n-m}$ ist auch als das Schurkomplement von $\mathbf{K}_{mm}$ in $\mathbf{K}$ bekannt.

[6] Es gilt die reproduzierende Eigenschaft: $f(\mathbf{x}) = \langle f(\cdot), k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}},\ \forall f \in \mathcal{H}$ und daher speziell $\langle k(\mathbf{x}, \cdot), k(\mathbf{y}, \cdot)\rangle = k(\mathbf{x}, \mathbf{y})$.

3.2 Ein reduziertes Problem

Die Approximation durch eine m-elementige Teilmenge der Daten (und im allgemeinen sind wir natürlich an $m \ll n$ interessiert) kann nun den Aufwand beim Lösen des Gleichungssystems erheblich reduzieren:

Ausnutzen der Rang-reduzierten Faktorisierung: die Nyström-Methode

Bei dieser Vorgehensweise, die u.a. von Williams und Seeger (2001) im Rahmen von GP-Regression und von Fine und Scheinberg (2001) im Rahmen von ε-SVM angewendet wurden, wird die Kernmatrix $\mathbf{K}$ im zugehörigen Optimierungsproblem durch eine faktorisierte Rang-m Approximation ersetzt. Im Falle von GP-Regression (äquivalent zu RN) wurde dementsprechend die (Nyström-) Approximation $\tilde{\mathbf{K}} = \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}$ aus (3.8) in das Gleichungssystem $(\mathbf{K} + \lambda\mathbf{I})\boldsymbol{\alpha} = \mathbf{y}$ eingesetzt und (3.1) ausgenutzt, um die Lösung mit dem reduzierten Aufwand $\mathcal{O}(nm^2)$ auszurechnen. Die Anzahl der zu bestimmenden Koeffizienten bleibt unverändert n. In der Praxis erweist sich diese häufig als 'Nyström' bezeichnete Variante vor allem dann als problematisch, wenn m klein und die Approximation daher mit relativ großen Fehlern behaftet ist. Auf Seite 51 illustrieren wir diesen Aspekt an einem konkreten Beispiel.

Die Subset of Regressors Approximation

Ein zweiter, weitergehender und konsequenterer Ansatz ist die sogenannte *Subset of Regressors* (SR) Approximation. Diese wird in der Literatur in ähnlicher Form an verschiedenen Stellen beschrieben und spiegelt die äquivalenten Ausprägungen des dieser Arbeit zugrundeliegenden Ansatz zur nicht-parametrisierten Funktionsapproximation wieder: in Wahba (1990); Luo und Wahba (1997) wird sie im Zusammenhang mit Splinemodellen, in Poggio und Girosi (1990); Smola und Schölkopf (2000); Rifkin (2002) im Zusammenhang mit RN und generalisierten Radialen Basisfunktionsnetzen und in Williams et al. (2002); Rasmussen und Williams (2006) im Zusammenhang mit GP-Regression genannt.

Bei SR werden *alle* Vorkommnisse des Kerns $k(\mathbf{x}, \mathbf{x}')$ durch die Approximation $\mathbf{k}_m(\mathbf{x})^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}')$ ersetzt. Das bedeutet folgendes: galt zunächst für Vorhersagen in beliebigen $\mathbf{x}$ allgemein

$$f(\mathbf{x}) = \mathbf{k}(\mathbf{x})^\mathsf{T}(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

mit $n \times 1$ Vektor $\mathbf{k}(\mathbf{x}) := \big(k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_n)\big)^\mathsf{T}$, so folgt nun durch das Ersetzen von $k(\mathbf{x}, \mathbf{x}')$ durch die Approximation zunächst $\mathbf{k}(\mathbf{x}) \approx \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x})$ und damit weiter

$$f(\mathbf{x}) \approx \mathbf{k}_m(\mathbf{x})^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^\mathsf{T}\big(\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^\mathsf{T} + \lambda\mathbf{I}\big)^{-1}\mathbf{y}.$$

Anwendung von (B.3)[7] zeigt

$$\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^\mathsf{T}\big(\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^\mathsf{T} + \lambda\mathbf{I}_n\big)^{-1} = \big(\mathbf{K}_{nm}^\mathsf{T}\lambda^{-1}\mathbf{I}_n\mathbf{K}_{nm} + \mathbf{K}_{mm}\big)^{-1}\mathbf{K}_{nm}^\mathsf{T}\lambda^{-1}\mathbf{I}_n$$

und damit wieder

$$f(\mathbf{x}) \approx \mathbf{k}_m(\mathbf{x})^\mathsf{T}\big(\mathbf{K}_{nm}^\mathsf{T}\mathbf{K}_{nm} + \lambda\mathbf{K}_{mm}\big)^{-1}\mathbf{K}_{nm}^\mathsf{T}\mathbf{y}.$$

Demnach reicht es also aus, nur noch m Variablen, korrespondierend zu den m Elementen der Teilmenge $\mathcal{D}$ zu betrachten.

Anstelle der ursprünglichen $n \times n$ Aufgabenstellung (2.18) reicht es aufgrund der SR-Approximation infolgedessen aus, lediglich die reduzierte $m \times m$ Aufgabenstellung

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^m} \; J_\lambda(\boldsymbol{\alpha}) := \|\mathbf{y} - \mathbf{K}_{nm}\boldsymbol{\alpha}\|^2 + \lambda\boldsymbol{\alpha}^\mathsf{T}\mathbf{K}_{mm}\boldsymbol{\alpha} \tag{3.9}$$

zu betrachten, deren Lösung

$$\boldsymbol{\alpha} = \big(\mathbf{K}_{nm}^\mathsf{T}\mathbf{K}_{nm} + \lambda\mathbf{K}_{mm}\big)^{-1}\mathbf{K}_{nm}^\mathsf{T}\mathbf{y} \tag{3.10}$$

[7] Es gilt $\mathbf{A}^{-1}\mathbf{B}(\mathbf{D} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} = (\mathbf{A} + \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1}$. Mit $\mathbf{A} := \mathbf{K}_{mm}$, $\mathbf{B} := \mathbf{K}_{nm}^\mathsf{T}$, $\mathbf{C} := \mathbf{K}_{nm}$ und $\mathbf{D} := \lambda\mathbf{I}_n$ folgt die Behauptung.

ist. Die gelernte Funktion f wird nur noch durch die Elemente von $\mathcal{D}$ repräsentiert:

$$f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i k(\tilde{\mathbf{x}}_i, \mathbf{x}) = \boldsymbol{\alpha}^\mathsf{T} \mathbf{k}_m(\mathbf{x}). \tag{3.11}$$

Sofern $\mathcal{D}$ bereits bekannt und mit m geeigneten Elementen angefüllt ist (Methoden dazu diskutieren wir im nächsten Abschnitt), beträgt der Rechenaufwand $\mathcal{O}(nm^2)$ Operationen um Gewichtsvektor $\boldsymbol{\alpha}$ aus den n Daten mit (3.10) zu bestimmen und $\mathcal{O}(m)$ Operationen, um mit (3.11) die gelernte Funktion auszuwerten. Der notwendige Speicherbedarf ist $\mathcal{O}(m^2)$ während der Berechnung der $\boldsymbol{\alpha}$, am Ende des Lernens reichen aber wieder $\mathcal{O}(m)$ für die Speicherung der Gewichte $\boldsymbol{\alpha}$ und den Elementen von $\mathcal{D}$ aus. Weiterhin ist anzumerken, daß wir die Elemente von $\mathcal{D}$ auch als herkömmliche Basisfunktionen betrachten können. Durch die SR-Approximation (3.9), (3.10), (3.11) reduzieren sich a priori nicht-parametrisierte Modelle (2.18), (2.19), (2.20) auf den Fall gewöhnlicher, linear parametrisierter Modelle und dürfen daher wie die in Kapitel 1 und Kapitel 2 beschriebenen herkömmlichen Basisfunktionsnetze mit festen Knoten behandelt werden (mit einer speziellen Form des Strafterms $\mathbf{K}_{mm}$).

3.3 Selektion der relevanten Basisfunktionen in $\mathcal{D}$

Als nächstes stellt sich natürlich die Frage, welche Daten wir in die Teilmenge $\mathcal{D}$ aufnehmen wollen. Die sicherlich einfachste Möglichkeit ist randomisierte Auswahl: die m Daten für $\mathcal{D}$ werden einfach zufällig aus der Menge aller n Trainingsdaten herausgegriffen. Als zugrundeliegende Verteilung kann dafür, wie in Williams und Seeger (2001), die Gleichverteilung genommen werden, oder man versieht, wie in Drineas und Mahoney (2005), gewisse Daten aufgrund einer Heuristik mit einer höheren Auswahlwahrscheinlichkeit. Die Vorteile der randomisierten Auswahl liegen auf der Hand: sie sind algorithmisch leicht umzusetzen und mit keinem oder sehr wenigem zusätzlichen Rechenaufwand verbunden. Möglicherweise lassen sich aber, wenn wir etwas mehr Arbeit in die Auswahl investieren, noch bessere Ergebnisse erzielen.

Die diesbezüglichen Auswahlverfahren lassen sich grob in zwei Kategorien einteilen:

1. unüberwacht
2. überwacht

Die *unüberwachten* Verfahren verwenden nur die reinen Eingaben $\{\mathbf{x}_i\}_{i=1}^n$ der vorliegenden Trainingsdaten $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ und zielen darauf ab, den Fehler $\|\mathbf{K} - \tilde{\mathbf{K}}\|_F$ durch Approximation der Kernmatrix zu minimieren. Leider kann das Minimum nicht direkt bestimmt werden, weil es sich bei der Auswahl um ein kombinatorisches Problem handelt[8] und es schlichtweg viel zu aufwendig ist, unter allen möglichen m-elementigen Teilmengen der Trainingsdaten diejenige zu ermitteln, die das Minimum bezüglich $\|\mathbf{K}-\tilde{\mathbf{K}}\|_F$ realisiert. Stattdessen kommen nur nichtoptimale, sequentielle Auswahlprozeduren in Frage.

Primär geht es uns aber auch gar nicht um eine gute Approximation des Kerns (und damit einer guten Repräsentation der Inputs im Merkmalsraum), sondern vielmehr um eine gute Approximation des Ausgabesignals $\mathbf{y}$ (und damit einer guten Rekonstruktion der zugrundeliegenden Funktion). Die *überwachten* Verfahren berücksichtigen daher bei der Selektion zusätzlich oder alternativ das Residuum $J_\lambda(\cdot)$ der ursprünglichen Approximation aus (3.9). Ihr Hauptvorteil ist, daß sie tendenziell in einer deutlich kleineren Auswahl resultieren, weil nur solche Elemente selektiert werden, die zum Darstellen des konkreten Ausgabesignals unbedingt benötigt werden, während unüberwachte Methoden vor allem durch irrelevante Ausreißerdaten unnötig belastet werden. Eine möglichst kompakte Darstellung ist für uns natürlich wieder deswegen von Interesse, weil die Komplexität des gesamten Lösungsverfahren quadratisch von ihr (von m) abhängt; wir werden diesen Umstand später in Kapitel 4 zur weiteren Steigerung der Effizienz unseres Online-Verfahren noch gehörig ausnutzen.

[8] Achtung, die optimale Rang-m Approximation bzgl. $\|\mathbf{K} - \tilde{\mathbf{K}}\|_F$ kann nach (3.2) durch PCA bzw. abgeschnittene Singulärwertzerlegung konstruiert werden. Das ist aber etwas völlig anderes als die hier betrachtete Form der Approximation, weil bei PCA *alle* Daten eingehen und die neuen (reduzierten) Richtungen aus der Linearkombination aller Spalten aufgebaut werden. Hier versuchen wir dagegen eine Approximation aus einer Teilmenge der Daten (d.h. den Zeilen) zu konstruieren.

Gierige Auswahlstrategien

Gierige Selektionsstrategien sind inkrementelle Auswahlverfahren, die in jedem Auswahlschritt das bezüglich eines vorgegebenen Kriteriums optimale Element hinzufügen und sowohl unüberwacht als auch überwacht sein können. Zur Durchführung werden zwei Indexmengen verwaltet: $\mathcal{A}$, die Indizes der bisher ausgewählten (aktiven) Daten und $\mathcal{I}$, die Indizes der nicht-ausgewählten (inaktiven) Daten. Zu Beginn ist $\mathcal{A}$ leer und $\mathcal{I}$ erhält als mögliche Kandidaten alle Daten. In jedem Schritt wird nun dasjenige Element von $\mathcal{I}$ nach $\mathcal{A}$ verschoben, das unter allen verbleibenden Elementen von $\mathcal{I}$ bezüglich des vorgegebenen Selektionskriteriums optimal ist. Die Auswahlprozedur endet, wenn eine zuvor festgelegte Anzahl zu selektierender Elemente erreicht ist, oder wird vorzeitig abgebrochen, wenn ein Abbruchkriterium erfüllt ist (z.B. der Restfehler eine vorgegebene Schranke unterschreitet). Hier stellen wir mehrere Möglichkeiten vor:

In Smola und Schölkopf (2000); Schölkopf und Smola (2002) wird das unüberwachte Verfahren SGMA beschrieben, das auf diese Weise das Kriterium $\|\mathbf{K} - \tilde{\mathbf{K}}\|_F$ schrittweise minimiert (wobei $\tilde{\mathbf{K}}$ aus der jeweiligen aktiven Menge $\mathcal{A}$ nach (3.8) aufgebaut ist). Der benötigte Aufwand ist allerdings sehr hoch: jede Evaluation eines Kandidaten in $\mathcal{I}$ kostet $\mathcal{O}(|\mathcal{A}|\cdot n)$, jeder einzelne Iterationsschritt somit $\mathcal{O}(|\mathcal{I}|\cdot|\mathcal{A}|\cdot n)$ Operationen. Für Aufgabenstellungen mit vielen Daten (und genau vor diesem Hintergrund betrachten wir die SR-Approximation) ist das natürlich nicht praktikabel.

Ähnlich teuer aber überwacht sind die vor allem im Zusammenhang mit Signalapproximation angewendeten Techniken Matching Pursuit (Mallat und Zhang, 1993) und Optimized Matching Pursuit (Pati et al., 1993; Davis et al., 1994; Rebollo-Neira und Lowe, 2002), im Bereich Neuronaler Netze auch als Orthogonal Least-Squares (Chen et al., 1991) bekannt. Hierbei wird in jedem Iterationsschritt dasjenige Element ausgewählt, das den verbleibenden Fehler der Approximation in (3.9)[9] am meisten reduziert. Als mögliche Basisfunktionskandidaten in $\mathcal{I}$ kann man sich speziell natürlich auch Kernfunktionen zentriert auf den Daten vorstellen und erhält so beispielsweise Kernel Matching Pursuit (Vincent und Bengio, 2002). Hoch ist dagegen in jedem Fall wieder der benötigte Aufwand: aufgrund der Orthogonalität kosten einzelne Evaluationen eines Kandidaten aus $\mathcal{I}$ zwar nur $\mathcal{O}(n)$ Operationen und die Suche nach dem besten Element somit $\mathcal{O}(|\mathcal{I}|\cdot n)$ Bei jedem Iterationsschritt wird aber zusätzlich noch eine Nach-Orthogonalisierung für $\mathcal{O}(|\mathcal{I}|\cdot n + |\mathcal{A}|\cdot n) = \mathcal{O}(n^2)$ Operationen fällig, so daß die Gesamtkosten für die Selektion von m Elementen ebenfalls $\mathcal{O}(mn^2)$ betragen. Auch scheinen insgesamt alle diese Verfahren allein für den Zweck Selektion geeigneter Basisfunktionen in der SR-Approximation überdimensioniert, weil sie ja schon für sich genommen das Funktionsapproximationsproblem lösen.

Im Rahmen von GP-Regression wurden noch weitere überwachte, Matching Pursuit-artige Varianten zur Selektion von $\mathcal{D}$ vorgeschlagen, die den notwendigen Rechenaufwand durch diverse vereinfachende Zusatzmaßnahmen reduzieren (Smola und Bartlett, 2001; Seeger et al., 2003; Keerthi und Chu, 2005).

Die unvollständige Cholesky-Zerlegung (Dualität von Gram-Schmidt und Cholesky)

Ein unüberwachtes, als Ausgangspunkt für die in dieser Arbeit in Kapitel 4 entwickelte Selektionsprozedur relevantes Verfahren ist die unvollständige Cholesky-Zerlegung (ICD) der Kernmatrix (Fine und Scheinberg, 2001; Bach und Jordan, 2002), die wir hier wie in Shawe-Taylor und Cristianini (2004) über partielle Gram-Schmidt Orthogonalisierung der Daten im Merkmalsraum motivieren wollen.

Beginnen wir wieder mit der primalen Sicht: die G.-S. Orthogonalisierung ist eine iterative Prozedur, die aus einer Menge von Elementen eines Hilbertraums, in unserem Fall die Repräsentation der Daten im Merkmalsraum $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_n)$, eine orthonormale Basis $\mathbf{u}_1, \ldots, \mathbf{u}_n$ konstruiert. Die Daten werden dabei in der Reihenfolge $1, 2, \ldots, n$ abgearbeitet. Der erste Schritt der Prozedur besteht aus den drei Teilschritten:

$$\begin{aligned} \mathbf{u}_1 &= \boldsymbol{\phi}(\mathbf{x}_1) \\ \delta_1 &= \|\mathbf{u}_1\| \\ \mathbf{u}_1 &:= \mathbf{u}_1/\delta_1 \end{aligned}$$

[9] Meist als gewöhnliches Least-Squares Problem ohne zusätzlichen Strafterm, weil die Komplexität durch die Anzahl der selektierten Basisfunktionen kontrolliert wird.

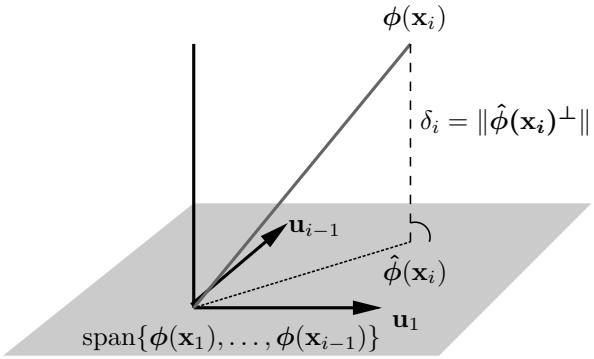


Abbildung 3.2: Projektion von $\phi(\mathbf{x}_i)$ auf den Unterraum der zuvor gesehenen Daten $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_{i-1})$ mit ONB $\mathbf{u}_1, \ldots, \mathbf{u}_{i-1}$. Die Länge des Residuums der Projektion δ_i kann als Kriterium für unüberwachte Selektion geeigneter Basisfunktionen herangezogen werden.

Für jeden weiteren Schritt i nehmen wir das nächste Element $\phi(\mathbf{x}_i)$, ziehen den Anteil der $i-1$ zuvor bearbeiteten Daten $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_{i-1})$ mit orthonormaler Basis $\mathbf{u}_1, \ldots, \mathbf{u}_{i-1}$ ab und normalisieren

$$\begin{aligned} \mathbf{u}_i &= \phi(\mathbf{x}_i) - \sum_{\nu=1}^{i-1} \langle \phi(\mathbf{x}_i), \mathbf{u}_\nu \rangle \mathbf{u}_\nu \\ \delta_i &= \|\mathbf{u}_i\| \\ \mathbf{u}_i &:= \mathbf{u}_i / \delta_i \end{aligned} \tag{3.12}$$

Von besonderer Bedeutung ist der Wert δ_i; er mißt den Abstand, den das aktuelle Beispiel i von den $i-1$ zuvor gesehenen Beispielen hat und ist ein Indikator dafür, wieviel 'neue' Informationen $\phi(\mathbf{x}_i)$ gegenüber den bisherigen Beispielen $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_{i-1})$ enthält (vgl. Abbildung 3.2). Der Wert δ_i ist auch identisch mit $\sqrt{\delta_{\mathbf{x}}}$ aus (3.6), wenn $\mathbf{x} = \mathbf{x}_i$ und $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}\}$ ist. Aus ihm wird später das entscheidende Selektionskriterium konstruiert werden.

Betrachten wir nun die $d_{\mathcal{H}} \times (i-1)$ Matrix $\mathbf{U}_{i-1} := \begin{bmatrix} \mathbf{u}_1 & \ldots & \mathbf{u}_{i-1} \end{bmatrix}$. Dann erhalten wir

$$\phi(\mathbf{x}_i) = \mathbf{U}_{i-1} \mathbf{U}_{i-1}^\mathsf{T} \phi(\mathbf{x}_i) + \mathbf{u}_i \cdot \delta_i$$

und weiter mit

$$\mathbf{U}_i := \begin{bmatrix} \mathbf{U}_{i-1} & \mathbf{u}_i \end{bmatrix}, \qquad \mathbf{r}_i := \begin{bmatrix} \mathbf{U}_{i-1}^\mathsf{T} \phi(\mathbf{x}_i) \\ \delta_i \end{bmatrix}$$

auch

$$\phi(\mathbf{x}_i) = \mathbf{U}_i \mathbf{r}_i.$$

Führen wir das G.-S. Verfahren für alle n Daten $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)$ durch, so erhalten wir insgesamt für die Datenmatrix $\mathbf{\Phi}^\mathsf{T} = \begin{bmatrix} \phi(\mathbf{x}_1) & \ldots & \phi(\mathbf{x}_n) \end{bmatrix}$ die Zerlegung $\mathbf{\Phi}^\mathsf{T} = \mathbf{QR}$ mit $\mathbf{Q} := \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \ldots & \mathbf{u}_n \end{bmatrix}$ und $\mathbf{R} := \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \ldots & \mathbf{r}_n \end{bmatrix}$, wobei genauer

$$\mathbf{R} = \begin{bmatrix} \delta_1 & \mathbf{U}_1^\mathsf{T}\phi(\mathbf{x}_2) & & \mathbf{U}_{n-1}^\mathsf{T}\phi(\mathbf{x}_n) \\ & \delta_2 & & \\ & & \ldots & \\ & & & \delta_n \end{bmatrix} = \begin{bmatrix} \delta_1 & \langle \mathbf{u}_1, \phi(\mathbf{x}_2) \rangle & & \langle \mathbf{u}_1, \phi(\mathbf{x}_n) \rangle \\ & \delta_2 & & \langle \mathbf{u}_2, \phi(\mathbf{x}_n) \rangle \\ & & \ldots & \vdots \\ & & & \langle \mathbf{u}_{n-1}, \phi(\mathbf{x}_n) \rangle \\ & & & \delta_n \end{bmatrix} \tag{3.13}$$

gilt.

Wie zuvor können wir aber das alles aufgrund der hohen Dimension des Merkmalsraums nicht direkt ausrechnen.

Allerdings sind die Einträge von $\mathbf{R}$ ausschließlich Skalarprodukte im Merkmalsraum. Gehen wir daher jetzt wieder auf die duale Sichtweise über, bei der wir nicht mehr direkt mit den Elementen des Merkmalsraums arbeiten, sondern das Skalarprodukt $\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y}) \rangle$ zweier Elemente mit Hilfe des Kerns $k(\mathbf{x}, \mathbf{y})$ ausdrücken. Für δ_i erhalten wir mit $r_{i\nu} := \langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{u}_\nu \rangle$ aus (3.12) den Ausdruck

$$\begin{aligned} \delta_i^2 &= \|\mathbf{u}_i\|^2 = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_i) \rangle - 2 \sum_{\nu=1}^{i-1} r_{i\nu} \underbrace{\langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{u}_\nu \rangle}_{=r_{i\nu}} + \sum_{\nu=1}^{i-1} \sum_{\mu=1}^{i-1} r_{i\nu} r_{i\mu} \underbrace{\langle \mathbf{u}_\nu, \mathbf{u}_\mu \rangle}_{=\delta_{\nu,\mu}} \\ &= \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_i) \rangle - \sum_{\nu=1}^{i-1} r_{i\nu}^2 \end{aligned} \tag{3.14}$$

Und weiter folgt für $i = 1, \ldots, n$ und $j = 1, \ldots, i-1$ wegen $\mathbf{u}_j = (\boldsymbol{\phi}(\mathbf{x}_j) - \sum_\nu^{j-1} r_{j\nu} \mathbf{u}_\nu)/\delta_j$ der Ausdruck

$$\langle \mathbf{u}_j, \boldsymbol{\phi}(x_i) \rangle = \frac{1}{\delta_j} \left(\langle \boldsymbol{\phi}(\mathbf{x}_j), \boldsymbol{\phi}(\mathbf{x}_i) \rangle - \sum_{\nu=1}^{j-1} r_{j\nu} \underbrace{\langle \mathbf{u}_\nu, \boldsymbol{\phi}(\mathbf{x}_i) \rangle}_{=r_{\nu i}} \right).$$

Bezeichnen wir die Einträge von $\mathbf{R}$ in (3.13) mit r_{ij}, dann folgt daraus insgesamt für die Konstruktion von $\mathbf{R}$

1. Für $i = 1 \ldots n$
2. (a) Für $j = 1 \ldots i-1$
   (b) i. $r_{ij} := \frac{1}{r_{jj}} \left(k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{\nu=1}^{j-1} r_{i\nu} r_{\nu j} \right)$
   (c) $r_{ii} := \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - \sum_{\nu=1}^{i-1} r_{i\nu}^2}$ (r_{ii} ist gleich δ_i aus (3.14))

was aber nichts anderes ist, als der Algorithmus für die Cholesky-Zerlegung der symmetrischen positiv definiten Kernmatrix $\mathbf{K}$ in $\mathbf{K} = \mathbf{R}^\mathsf{T}\mathbf{R}$ mit oberer Dreiecksmatrix $\mathbf{R}$. Anders gesagt, die Berechnung der Cholesky-Zerlegung von $\mathbf{K}$ ist das duale Äquivalent zur Berechnung der QR-Zerlegung von der primalen Darstellung $\boldsymbol{\Phi}^\mathsf{T}$[10]. Besonders interesssant sind die diagonalen Einträge r_{ii} des Cholesky-Faktors von $\mathbf{K}$, denn hier steht das Residuum der Projektion vom Merkmalsvektor $\boldsymbol{\phi}(\mathbf{x}_i)$ auf den Unterraum der zuvor gesehenen Daten $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_{i-1})$; eine Quantität, die für die Auswahl einer relevanten Teilmenge von Daten noch sehr nützlich sein wird (vgl. Kapitel 4).

Die Berechnung der vollständigen Cholesky-Zerlegung von $\mathbf{K}$ kostet $\mathcal{O}(n^3)$ Operationen. Bis hierhin haben wir also noch nichts wirklich erreicht.

Wir können uns nun aber leicht eine modifizierte Version der G.-S. Orthogonalisierung vorstellen, bei der die Daten $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_n)$ nicht in der Reihenfolge $1, 2, \ldots, n$ durchlaufen werden, sondern bei der wie in gieriger Selektion in jedem Iterationsschritt dasjenige Element als nächstes der Basis hinzugefügt wird, das unter allen verbleibenden Elementen den größten Abstand δ_i zum bisherigen Erzeugnis hat. Die Iteration bricht ab, wenn eine vorher festgelegte Anzahl m von Elementen selektiert wurde, oder wenn kein geeigneter Kandidat mehr vorhanden ist, d.h. der Abstand aller verbleibenden Elemente eine Toleranz nicht mehr überschreitet. Auf diese Weise stellen wir sicher, daß nur noch Daten hinzugenommen werden, die gegenüber den bisher ausgewählten Daten hinreichend neue Informationen mitbringen. Dieses Verfahren wird in Shawe-Taylor und Cristianini (2004) partielle Gram-Schmidt Orthogonalisierung genannt und wird aufgrund der Dualität durch die unvollständige Cholesky-Zerlegung von $\mathbf{K}$ implementiert (Fine und Scheinberg, 2001; Bach und Jordan, 2002).

[10] Hier hätte man auch direkt argumentieren können, und von der Eindeutigkeit der QR-Zerlegung von $\boldsymbol{\Phi}^\mathsf{T} = \mathbf{QR}$ auf die Dualität mit der Cholesky-Zerlegung von $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} = \mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{QR} = \mathbf{R}^\mathsf{T}\mathbf{R}$ schließen können. Wir haben das aber ganz bewußt so ausführlich beschrieben, um den Zusammenhang von der Diagonalen des Cholesky-Faktors mit dem Abstand der zuvor 'gesehenen' Daten klarzumachen. Dieser Abstand kann als 'Neuheit' eines Datenbeispiels interpretiert werden und wird für unseren Online-Algorithmus in Kapitel 4 eine entscheidende Rolle spielen.

Die unvollständige Cholesky-Zerlegung (ICD) findet darüberhinaus auch Anwendung als Vorkonditionierer beim iterativen Lösen großer Gleichungssysteme (vgl. Golub und Van Loan, 1996). Im Zusammenhang mit der SR-Approximation ist sie gegenwärtig das Standardverfahren, um die m relevanten Elemente der Teilmenge zu identifizieren; für die Selektion von m Komponenten beträgt ihr Rechenaufwand $\mathcal{O}(m^2n)$ Operationen, so daß der Gesamtaufwand zusammen mit dem Bestimmen des Gewichtsvektors aus (3.10) ebenfalls $\mathcal{O}(m^2n)$ erfordert. Gegenüber SGMA (Smola und Schölkopf, 2000) ist sie daher auch in der Praxis durchführbar. Der zur Durchführung notwendige Speicherbedarf ist $\mathcal{O}(mn)$. Als inkrementelle Selektionsprozedur hat sie darüberhinaus noch den Vorteil eines flexiblen Abbruchkriteriums: entweder eine feste Anzahl oder vorzeitiger Abbruch, falls kein geeignetes Element mehr vorhanden ist.

Allerdings ist unvollständige Cholesky-Zerlegung auch ein unüberwachtes Verfahren, das das zu lernende Ausgabesignal nicht mitberücksichtigt. In Bach und Jordan (2005) wird daher eine überwachte Variante vorgeschlagen, die ohne nennenswerte Einbußen bei der Approximation des Ausgabesignals zu einer deutlich kompakteren Selektion $\mathcal{D}$ für SR führt (experimentell). Ihre Komplexität ist ebenfalls nur $\mathcal{O}(m^2n)$.

Beispiele

Wir illustrieren das soweit Gesagte anhand dreier konkreter Benchmark-Datensätze:

1. der synthetische Datensatz 'Sinc' aus Abschnitt 2.4.2 (Seite 39), bestehend aus 200 Trainingsdaten und 100 Testdaten. Die Inputs $\mathbf{x}$ sind eindimensional.
2. der reale Datensatz 'Abalone' zur automatischen Bestimmung des Alters von Abalone-Fischen (was manuell eine aufwendige und komplizierte Prozedur ist) anhand einer Reihe leicht feststellbarer äußerer Merkmale. Die 4177 Daten wurden zu Beginn zufällig in 3000 Trainingsbeispiele und 1177 Testbeispiele partitioniert. Die Inputs $\mathbf{x}$ sind 10-dimensional, das diskrete Merkmal Geschlecht (male/female/infant) wurde als (1,0,0), (0,1,0), (0,0,1) kodiert.
3. der reale Datensatz 'Boston housing data' zur Bestimmung des Grundstückswerts anhand leicht ermittelbarer externer Indikatoren. Die 506 Daten wurden zu Beginn zufällig in 400 Trainingsbeispiele und 106 Testbeispiele partitioniert. Die Inputs $\mathbf{x}$ sind in diesem Fall 13-dimensional.

In allen drei Fällen wurden die reellwertigen Inputs standardisiert, d.h. attributweise auf Mittelwert 0 und Varianz 1 skaliert. Die vorherzusagende Ausgabegröße ist jeweils reellwertig (Regression). Die beiden letztgenannten Datensätze sind frei verfügbar und wurden dem UCI Machine Learning Repository entnommen (Newman et al., 1998).

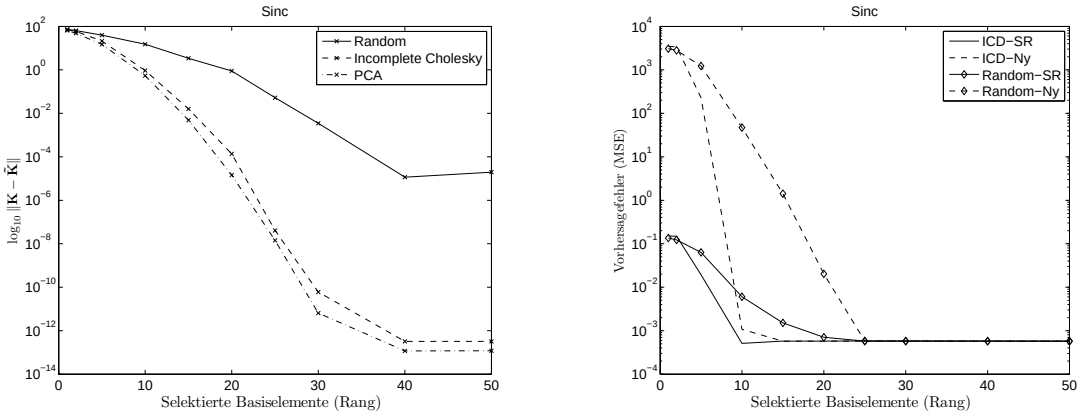
In den Experimenten wollen wir drei Fragen nachgehen: erstens, wie wirkt sich die Auswahlstrategie der Teilmenge auf den Fehler bei der resultierenden Rang-m Approximation der Kernmatrix aus? Zu diesem Zweck vergleichen wir in Abbildung 3.3 (linke Spalte) die Auswahlstrategien 'Random' und 'ICD' sowie (außer Konkurrenz)[11] das optimale PCA für verschiedene Werte von m. Zweitens, wie wirken sich 'Random' und 'ICD' auf den Vorhersagefehler in den Testdaten aus, wenn das Modell in den jeweils selektierten Basisfunktionen gelernt wurde? Und schließlich drittens, wie wirken sich die beiden Ansätze 'Subset of Regressors' (SR) aus (3.9) und 'Nyström' (Ny)[12] auf den jeweils resultierenden Vorhersagefehler in Abbildung 3.3 (rechte Spalte) aus?

Als Kern verwenden wir Gauß-RBF mit den Parametern $h = 0.02$ (Sinc), $h = 20$ (Abalone) und $h = 15.56$ (Boston); als Regularisierung wählen wir $\lambda = 0.1$.

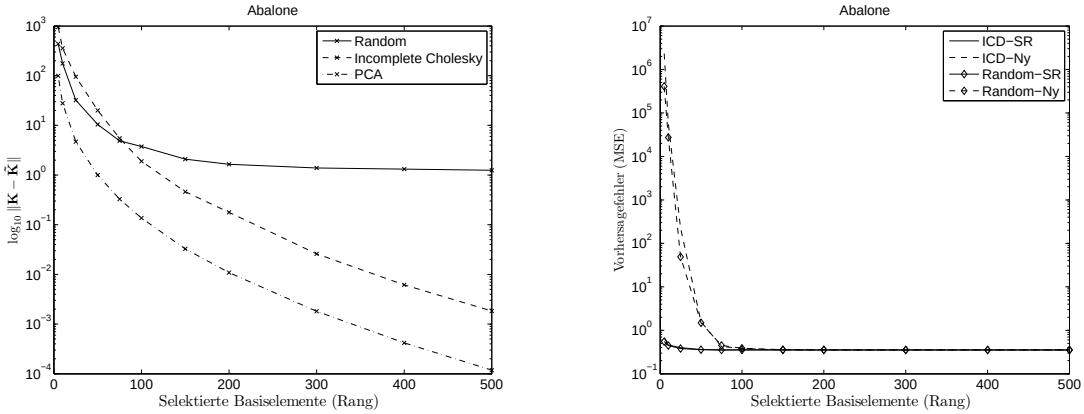
Den Ergebnissen in Abbildung 3.3 können wir folgendes entnehmen: erstens, 'ICD' liefert für den gleichen Preis (d.h. Anzahl m selektierter Basiselemente) eine merklich bessere Approximation des Kerns als 'Random' und ein besseres Ergebnis bei den Vorhersagen. Zweitens, anders als 'Ny' ist 'SR' robust und liefert auch bei wenigen Basiselementen (d.h. grober Approximation des Kerns) gute Vorhersagen; erst ab einer relativ großen Anzahl von Basiselementen (d.h. einer sehr genauen Approximation des Kerns) ist kein Unterschied mehr feststellbar. Fazit: SR zusammen mit ICD-Auswahl der Basisfunktionen erzielt für alle Approximationsgenauigkeiten m die besten Ergebnisse, was sich mit den Beobachtungen anderer Autoren deckt, vgl. Rifkin (2002); Williams et al. (2002); Rasmussen und Williams (2006).

[11] Vgl. Fußnote auf Seite 47

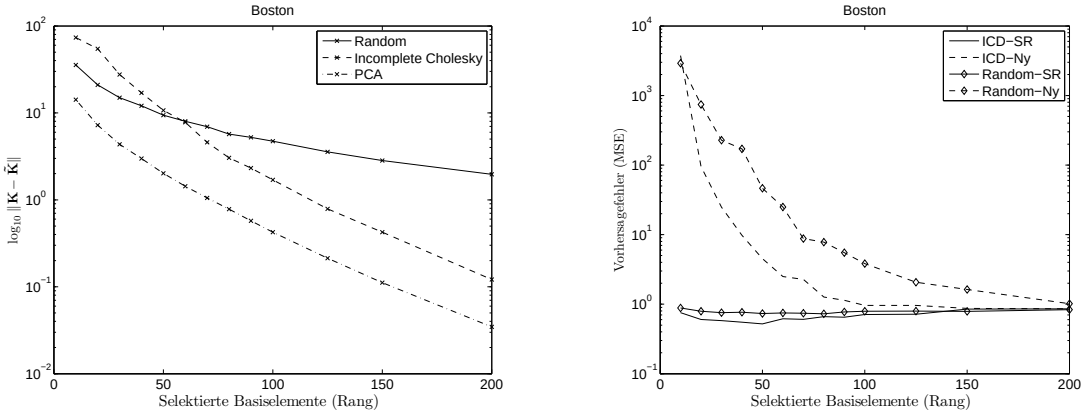
[12] Ausnutzen der Faktorisierung (3.1) zum effizienten Lösen von $(\mathbf{K}+\lambda\mathbf{I})\boldsymbol{\alpha} = \mathbf{y}$, ohne Reduktion der Anzahl der Gewichte.



(a) Datensatz 'Sinc' (200 Daten)



(b) Datensatz 'Abalone' (3000 Daten)



(c) Datensatz 'Boston' (400 Daten)

Abbildung 3.3: Auswirkungen Rang-reduzierter Approximation abhängig von m für die Selektionsstrategien 'Random', 'ICD' und der optimalen Rang-m Approximation durch PCA. Gezeigt werden jeweils der Approximationsfehler in der Kernmatrix (linke Spalte) sowie der Vorhersagefehler der damit gelernten Funktion in unabhängigen Testdaten (rechte Spalte). Zusätzlich vergleichen wir in jedem der Fälle die Varianten „Nyström“ und „Subset of Regressors“ zum Lernen der Funktion. Man sieht zweieierlei: erstens, vor allem bei wenigen Basiselementen ist 'SR' weitaus besser als 'Ny'. Zweitens, 'ICD' erzielt erwartungsgemäß eine bessere Approximation des Kerns als 'Random'.

Kapitel 4

Online Lernen mit Regularisierungsnetzen

Worin ein neuer Algorithmus entwickelt wird, der RN+SR auch für Online-Lernen möglich macht

Die im vorangegangenen Kapitel vorgestellte SR-Approximation erlaubt es uns, nichtparametrisierte RN mit linearem Zeitaufwand zu lösen ($\mathcal{O}(nm^2)$ statt $\mathcal{O}(n^3)$). Allerdings ist sie bisher nur in der Situation von Batch-Lernen anwendbar, wo die gesamten Trainingsdaten fest und zu Beginn des Lernens verfügbar sind. In diesem Kapitel wollen wir einen Schritt weitergehen und darauf aufbauend ein neues Verfahren formulieren, das speziell auf die Anforderungen von Online-Lernen zugeschnitten ist. Der wesentliche Unterschied gegenüber Batch-Lernen ist, daß beim Online-Lernen die Daten nur noch sequentiell in festvorgegebener und nichtbeeinflußbarer Reihenfolge zugreifbar sind, etwa $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$ und in *jedem* Schritt t jeweils eine Lösung für alle bisher gesehenen Daten $1, 2, \ldots, t$ berechnet werden soll. Damit dies effizient durchführbar ist, benötigen wir eine rekursive Implementation, die aus der Lösung zum Zeitpunkt t und einer neuen Beobachtung $(\mathbf{x}_{t+1}, y_{t+1})$ eine neue Lösung für den Zeitpunkt $t+1$ berechnet. Der dazu notwendige Aufwand muß dabei konstant und unabhängig von der Anzahl t der zuvor gesehenen Daten sein.

Online-Lernen selbst ist keine besonders exotische Anforderung, sondern bei vielen praktischen Problemen notwendig oder sinnvoll: so stammen bei Anwendungen wie der Signalverarbeitung oder Zeitreihenvorhersage die Daten aus einem dynamischen Prozeß, wodurch in regelmäßigen Zeitabständen immer neue Daten hinzukommen. Darüberhinaus kann Online-Lernen auch bei statischen Problemen mit sehr großen Datenmengen oder nur beschränkt zur Verfügung stehender Rechenkapazität (Echtzeitanwendung) sinnvoll sein. Hier betrachten wir Online-Lernen natürlich auch deswegen, weil unser eigentliches Ziel – Reinforcement Lernen – selbst auch ein dynamischer Prozeß und damit primär ein Online-Problem ist: die Daten werden zur Laufzeit von einem Agenten (Roboter) durch tatsächliche Interaktion mit der Umwelt gesammelt.

Ausgehend von der SR-Approximation des vorangehenden Kapitels entwickeln wir in diesem Kapitel einen Algorithmus, mit dem Online-Lernen für RN möglich wird. Was diesen Algorithmus in ganz besonderer Weise auszeichnet, ist, daß er nicht nur effizient (mit konstantem Zeitaufwand) neue Daten, sondern ebenso effizient auch neue Basisfunktionen hinzunehmen kann: die Menge der relevanten Basisfunktionen für SR wird *zur Laufzeit* aus dem aktuellen Datenstrom generiert. Möglich wird das wieder durch eine spezielle Approximation, die von Csató und Opper (2001) im Rahmen von GP-Regression und von Engel et al. (2004) für eine Recursive Least-Squares Variante vorgeschlagen wurde. Dabei dient, wie beim partiellen G.-S. in Kapitel 3, jeweils der Abstand zum Erzeugnis der bisher selektierten Daten als Auswahlkriterium für die Hinzunahme neuer Basisfunktionen; hier allerdings als eine Online-Prozedur implementiert, die diese Entscheidung für jedes Trainingsbeispiel einmalig und sofort treffen muß. Wir formulieren eine überwachte Erweiterung dieses Kriteriums, die zusätzlich zum Approximationsfehler (hervorgerufen durch die Approximation des Kerns) noch den Fehler der eigentlichen Regressionsaufgabe berücksichtigt, und nur solche Basisfunktionen auswählt, die auch für das Lernen der konkreten Funktion/Daten relevant

sind. Indem die verfügbaren Ressourcen dorthin verteilt werden, wo sie am meisten gebraucht werden, kann die Anzahl der notwendigen Basisfunktionen (und damit der Rechenaufwand) bei gleichbleibender Qualität der Approximation noch weiter reduziert werden, was uns insgesamt einen sehr leistungsfähigen Echtzeitalgorithmus liefert.

4.1 Inkrementelle Regularisierungsnetze (wie es nicht funktioniert)

Für den Fall, daß die Trainingsdaten nur sequentiell als Abfolge $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t), \ldots$ vorliegen, betrachten wir die SR-Aufgabenstellung aus (3.9) zum Zeitpunkt t in der Form

$$\min_{\mathbf{w}\in\mathbb{R}^m} \; J_{tm}(\mathbf{w}) := \|\mathbf{y}_t - \mathbf{K}_{tm}\mathbf{w}\|^2 + \lambda \mathbf{w}^\mathsf{T}\mathbf{K}_{mm}\mathbf{w} \tag{4.1}$$

mit Lösung

$$\mathbf{w}_{tm} = (\mathbf{K}_{tm}^\mathsf{T}\mathbf{K}_{tm} + \lambda \mathbf{K}_{mm})^{-1}\mathbf{K}_{tm}^\mathsf{T}\mathbf{y}_t \tag{4.2}$$

und Kosten

$$\xi_{tm} := J_{tm}(\mathbf{w}_{tm}), \tag{4.3}$$

wobei $\mathbf{y}_t = \big(y_1, \ldots, y_t\big)^\mathsf{T}$ die Outputs, $\mathcal{D}_t = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_m\}$ die Daten zu den gegenwärtig selektierten Basisfunktionen $k(\tilde{\mathbf{x}}_1, \cdot), \ldots, k(\tilde{\mathbf{x}}_m, \cdot)$ für SR und Datenmatrix $\mathbf{K}_{tm}$ gleich

$$\mathbf{K}_{tm} = \begin{bmatrix} \text{—} \; \mathbf{k}_m(\mathbf{x}_1)^\mathsf{T} \; \text{—} \\ \vdots \\ \text{—} \; \mathbf{k}_m(\mathbf{x}_t)^\mathsf{T} \; \text{—} \end{bmatrix}$$

mit $\mathbf{k}_m(\cdot) = \big(k(\tilde{\mathbf{x}}_1, \cdot), \ldots, k(\tilde{\mathbf{x}}_m, \cdot)\big)^\mathsf{T}$ ist. Größen, die sowohl von der Anzahl der Daten t als auch von der Anzahl der Basisfunktionen m abhängen, werden mit einem doppelten Index gekennzeichnet. Die Menge $\mathcal{D}_t$ der Basisfunktionen im Modell ist ebenfalls variabel: unser Ziel ist es, *zur Laufzeit* sowohl neue relevante Basisfunktionen hinzuzunehmen, als auch redundant gewordene alte wieder zu entfernen.

Der Aufbau des Algorithmus sieht nun wie folgt aus. In jedem Schritt beobachten wir ein neues Trainingsbeispiel $(\mathbf{x}_{t+1}, y_{t+1})$ und führen dann folgende Teilschritte aus (im folgenden sei $y^* := y_{t+1}$):

1. **Normaler Schritt:** ($\{t, m\} \to \{t+1, m\}$) Zuerst fügen wir das neue Trainingsbeispiel in das bestehende Modell mit m Basisfunktionen ein und passen den Gewichtsvektor $\mathbf{w}_{tm}$ an:

$$\mathbf{w}_{t+1,m} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^m} \left\{ \left\| \begin{bmatrix} \mathbf{y}_t \\ y^* \end{bmatrix} - \begin{bmatrix} \mathbf{K}_{tm} \\ \mathbf{k}_{t+1}^\mathsf{T} \end{bmatrix} \mathbf{w} \right\|^2 + \lambda \mathbf{w}^\mathsf{T}\mathbf{K}_{mm}\mathbf{w} \right\}$$

2. **Basiserweiterungstest:** Dann wird überprüft, ob $\mathbf{x}_{t+1}$ einen geeigneten Kandidaten für eine neue Basisfunktion abgeben würde, oder ob es bereits hinreichend gut von den bisher in $\mathcal{D}_t$ ausgewählten Daten repräsentiert wird.

3. **Basiserweiterungsschritt:** ($\{t+1, m\} \to \{t+1, m+1\}$) Falls eine Basiserweiterung notwendig ist, wird $\mathbf{x}_{t+1}$ der Menge $\mathcal{D}_t$ hinzugefügt und der Gewichtsvektor $\mathbf{w}_{t+1,m}$ für das erweiterte Modell angepaßt:

$$\mathbf{w}_{t+1,m+1} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{m+1}} \left\{ \big\|\mathbf{y}_{t+1} - \begin{bmatrix} \mathbf{K}_{t+1,m} & \mathbf{q} \end{bmatrix} \mathbf{w}\big\|^2 + \lambda \mathbf{w}^\mathsf{T} \begin{bmatrix} \mathbf{K}_{mm} & \mathbf{k}_{t+1} \\ \mathbf{k}_{t+1}^\mathsf{T} & k^* \end{bmatrix} \mathbf{w} \right\}$$

4. **Löschen redundanter Basisfunktionen:** ($\{t, m\} \to \{t, m\backslash i\}$) Übersteigt zu einem gegebenen Zeitpunkt die Anzahl der selektierten Basisfunktionen in $\mathcal{D}_t$ eine vorher festgelegte maximale Anzahl, so wird die am wenigsten zur Gesamtlösung beitragende Basisfunktion $i \in \{1, \ldots, m\}$ aus dem Modell entfernt und die Lösung angepaßt:

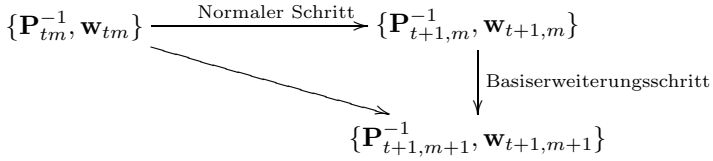
$$\mathbf{w}_{t,m\backslash i} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{m-1}} \left\{ \big\|\mathbf{y}_t - \mathbf{K}_{t,m\backslash i}\mathbf{w}\big\|^2 + \lambda \mathbf{w}^\mathsf{T}\mathbf{K}_{m\backslash i,m\backslash i}\mathbf{w} \right\}$$

wobei $\mathbf{k}_{t+1} := \mathbf{k}_m(\mathbf{x}_{t+1})$ die zur Datenmatrix neu hinzugefügte Zeile (entspricht dem Merkmalsvektor des neuen Beispiels), $\mathbf{q} := \big(k(\mathbf{x}_1, \mathbf{x}_{t+1}), \ldots, k(\mathbf{x}_t, \mathbf{x}_{t+1}), k^*\big)^\mathsf{T}$ die neu hinzugefügte Spalte (entspricht einer neuen Basisfunktion zentriert auf $\mathbf{x}_{t+1}$) und $k^* := k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1})$ ist.

Der gesamte Algorithmus wird analog zu Recursive Least Squares (vgl. Sayed, 2003) um das Vorwärtspropagieren der Inversen[1] von $\mathbf{P}_{tm} := (\mathbf{K}_{tm}^\mathsf{T}\mathbf{K}_{tm} + \lambda\mathbf{K}_{mm})$ herum aufgebaut, so daß für die Lösung stets

$$\mathbf{w}_{tm} = \mathbf{P}_{tm}^{-1}\mathbf{K}_{tm}^\mathsf{T}\mathbf{y}_t \tag{4.4}$$

gilt. Insgesamt reicht es aus, nur die Größen $\mathbf{P}_{tm}^{-1}$ (eine $m \times m$ Matrix), $\mathbf{w}_{tm}$ (ein $m \times 1$ Vektor), ξ_{tm} (ein Skalar) und $\mathcal{D}_t$ (eine Liste mit Einträgen $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_m$) zu speichern und bei jedem Schritt $t \to t+1$ anzupassen:



Die Hauptarbeit wird dabei beim Aktualisieren der Inversen $\mathbf{P}_{tm}^{-1} \to \mathbf{P}_{t+1,m}^{-1}$ bzw. $\mathbf{P}_{t+1,m}^{-1} \to \mathbf{P}_{t+1,m+1}^{-1}$ verrichtet, wofür wir folgende allgemeine Formeln ausnutzen (für $\mathbf{A}_t$ symmetrisch, $\mathbf{a}, a^*$ geeignet):

- Gilt $\mathbf{A}_{t+1} = \mathbf{A}_t + \mathbf{a}\mathbf{a}^\mathsf{T}$, so folgt aus (B.2) für die Inverse

$$\mathbf{A}_{t+1}^{-1} = \mathbf{A}_t^{-1} - \frac{\mathbf{A}_t^{-1}\mathbf{a}\mathbf{a}^T\mathbf{A}_t^{-1}}{1 + \mathbf{a}^T\mathbf{A}_t^{-1}\mathbf{a}}, \tag{4.5}$$

  was beim Hinzufügen einer neuen Zeile, also Teilschritt 1, notwendig ist.

- Gilt

$$\mathbf{A}_{t+1} = \begin{bmatrix} \mathbf{A}_t & \mathbf{a} \\ \mathbf{a}^\mathsf{T} & a^* \end{bmatrix},$$

  so folgt aus (B.4)

$$\mathbf{A}_{t+1}^{-1} = \begin{bmatrix} \mathbf{A}_t^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \frac{1}{a^* - \mathbf{a}^\mathsf{T}\mathbf{A}_t^{-1}\mathbf{a}} \begin{bmatrix} -\mathbf{A}_t^{-1}\mathbf{a} \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{A}_t^{-1}\mathbf{a} \\ 1 \end{bmatrix}^\mathsf{T}, \tag{4.6}$$

  was beim Hinzufügen einer neuen Spalte, also Teilschritt 3, notwendig ist.

Für die Kosten $\xi_{tm} := J(\mathbf{w}_{tm})$ erhalten wir den Ausdruck

$$\begin{aligned}
\xi_{tm} &= (\mathbf{y}_t - \mathbf{K}_{tm}\mathbf{w}_{tm})^\mathsf{T}(\mathbf{y}_t - \mathbf{K}_{tm}\mathbf{w}_{tm}) + \lambda\mathbf{w}_{tm}^\mathsf{T}\mathbf{K}_{mm}\mathbf{w}_{tm} \\
&= \mathbf{y}_t^\mathsf{T}\mathbf{y}_t - 2\mathbf{y}_t^\mathsf{T}\mathbf{K}_{tm}\mathbf{w}_{tm} + \mathbf{w}_{tm}^\mathsf{T}\mathbf{K}_{tm}^\mathsf{T}\mathbf{K}_{tm}\mathbf{w}_{tm} + \lambda\mathbf{w}_{tm}^\mathsf{T}\mathbf{K}_{mm}\mathbf{w}_{tm} \\
&= \mathbf{y}_t^\mathsf{T}\mathbf{y}_t - 2\mathbf{y}_t^\mathsf{T}\mathbf{K}_{tm}\mathbf{w}_{tm} + \mathbf{w}_{tm}^\mathsf{T}\underbrace{(\mathbf{K}_{tm}^\mathsf{T}\mathbf{K}_{tm} + \lambda\mathbf{K}_{mm})\mathbf{w}_{tm}}_{\mathbf{K}_{tm}^\mathsf{T}\mathbf{y}_t} \\
&= \mathbf{y}_t^\mathsf{T}(\mathbf{y}_t - \mathbf{K}_{tm}\mathbf{w}_{tm}) \\
&= \mathbf{y}_t^\mathsf{T}\mathbf{y}_{tm}^{\text{res}}
\end{aligned} \tag{4.7}$$

wobei wir $\mathbf{y}_{tm}^{\text{in}} := \mathbf{K}_{tm}\mathbf{w}_{tm}$ und $\mathbf{y}_{tm}^{\text{res}} := \mathbf{y}_t - \mathbf{y}_{tm}^{\text{in}}$ definiert haben. Aufgrund der zusätzlichen Regularisierung sind $\mathbf{y}_{tm}^{\text{in}}$ und $\mathbf{y}_{tm}^{\text{res}}$ nicht orthogonal (vgl. gewöhnliches Least-Squares in Abbildung 2.3).

Betrachten wir nun die Aktualisierungen, die durch die einzelnen Teilschritte vorgenommen werden müssen, im Detail. (Der Basiserweiterungstest wird in Abschnitt 4.2 beschrieben.)

[1] Numerisch stabiler wäre eine Variante, die stattdessen den Cholesky-Faktor vorwärtspropagiert, wie es in diesem Zusammenhang in Jung und Polani (2006a) beschrieben wird. An dieser Stelle wählen wir die Formulierung über Inversenbildung aber bewußt, um eine konsistente Herleitung mit der Anwendung auf Reinforcement Lernen zu gewährleisten, wo die fragliche Matrix $\mathbf{P}_{tm}$ nicht mehr länger symmetrisch ist (vgl. LSTD Gleichung (5.19') auf Seite 82).

4.1.1 Normaler Schritt

Aus dem vorangegangenen Zeitschritt t seien $\mathbf{P}_{tm}^{-1} := (\mathbf{K}_{tm}^\mathsf{T}\mathbf{K}_{tm} + \lambda\mathbf{K}_{mm})^{-1}$, die optimalen Koeffizienten $\mathbf{w}_{tm} = \mathbf{P}_{tm}^{-1}\mathbf{K}_{tm}^\mathsf{T}\mathbf{y}_t$, die Kosten ξ_{tm} sowie die Liste der Basisvektoren $\mathcal{D}_t$ bekannt. Jetzt beobachten wir das neue Datenpaar $(\mathbf{x}_{t+1}, y^*)$ und wollen damit rekursiv die Aktualisierung auf $\{\mathbf{P}_{t+1,m}^{-1}, \mathbf{w}_{t+1,m}, \xi_{t+1,m}\}$ vornehmen; die Basisvektoren bleiben in diesem Schritt unberührt. Zunächst stellen wir fest, wie sich die relevanten Matrizen verändern, wenn ein neues Beispiel hinzukommt: für $\mathbf{x}_{t+1}$ berechnen wir das Bild unter den m Basisfunktionen $\mathbf{k}_{t+1} := \mathbf{k}_m(\mathbf{x}_{t+1})$ und fügen $\mathbf{k}_{t+1}^\mathsf{T}$ als neue Zeile in der Datenmatrix und y^* als neuen Eintrag in $\mathbf{y}_{t+1}$ hinzu:

$$\mathbf{K}_{t+1,m} = \begin{bmatrix}\mathbf{K}_{tm}\\ \mathbf{k}_{t+1}^\mathsf{T}\end{bmatrix} \quad \text{und} \quad \mathbf{y}_{t+1} = \begin{bmatrix}\mathbf{y}_t\\ y^*\end{bmatrix}.$$

Damit ist

$$\mathbf{P}_{t+1,m} = \begin{bmatrix}\mathbf{K}_{tm}\\ \mathbf{k}_{t+1}^\mathsf{T}\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{K}_{tm}\\ \mathbf{k}_{t+1}^\mathsf{T}\end{bmatrix} + \lambda\mathbf{K}_{mm} = \mathbf{K}_{tm}^\mathsf{T}\mathbf{K}_{tm} + \mathbf{k}_{t+1}\mathbf{k}_{t+1}^\mathsf{T} + \lambda\mathbf{K}_{mm} = \mathbf{P}_{tm} + \mathbf{k}_{t+1}\mathbf{k}_{t+1}^\mathsf{T}$$

und mit der Formel (4.5) folgt für dessen Inverse

$$\mathbf{P}_{t+1,m}^{-1} = \mathbf{P}_{tm}^{-1} - \frac{\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^\mathsf{T}\mathbf{P}_{tm}^{-1}}{\boldsymbol{\Delta}}$$

mit $\boldsymbol{\Delta} := 1 + \mathbf{k}_{t+1}^\mathsf{T}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}$.

Aus der Rekursion für $\mathbf{P}_{t+1,m}^{-1}$ können wir nun auch eine für $\mathbf{w}_{t+1,m}$ herleiten, denn multiplizieren wir diesen Ausdruck von rechts mit $\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{y}_{t+1} = \mathbf{K}_{tm}^\mathsf{T}\mathbf{y}_t + y^*\mathbf{k}_{t+1}$, so erhalten wir

$$\begin{aligned}
\mathbf{w}_{t+1,m} &= \mathbf{P}_{t+1,m}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{y}_{t+1} = \left(\mathbf{P}_{tm}^{-1} - \Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^\mathsf{T}\mathbf{P}_{tm}^{-1}\right)\left(\mathbf{K}_{tm}^\mathsf{T}\mathbf{y}_t + y^*\mathbf{k}_{t+1}\right)\\
&= \mathbf{w}_{tm} + \underbrace{y^*\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}}_{\text{mit }\Delta\text{ erweitern}} - \Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^\mathsf{T}\mathbf{w}_{tm} - \Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^\mathsf{T}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}y^*\\
&= \mathbf{w}_{tm} + \frac{(y^* - \mathbf{k}_{t+1}^\mathsf{T}\mathbf{w}_{tm})}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}.
\end{aligned}$$

Zur weiteren Abkürzung definieren wir $\varrho := y^* - \mathbf{k}_{t+1}^\mathsf{T}\mathbf{w}_{tm}$.

Auch die Kosten $\xi_{t+1,m} = \mathbf{y}_{t+1}^\mathsf{T}\mathbf{y}_{t+1,m}^{\text{res}}$ lassen sich rekursiv aus ξ_{tm} nach (4.7) erzeugen. Zunächst betrachten wir die Aktualisierung des Residuums zu $\mathbf{y}_{t+1,m}^{\text{res}} = \mathbf{y}_{t+1} - \mathbf{K}_{t+1,m}\mathbf{w}_{t+1,m}$:

$$\begin{aligned}
\mathbf{y}_{t+1,m}^{\text{res}} &= \begin{bmatrix}\mathbf{y}_t\\ y^*\end{bmatrix} - \begin{bmatrix}\mathbf{K}_{tm}\\ \mathbf{k}_{t+1}^\mathsf{T}\end{bmatrix}\left[\mathbf{w}_{tm} + \tfrac{\varrho}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\right] = \begin{bmatrix}\mathbf{y}_{tm}^{\text{res}} - \frac{\varrho}{\Delta}\mathbf{K}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\\ \underbrace{y^* - \mathbf{k}_{t+1}^\mathsf{T}\mathbf{w}_{tm}}_{\varrho} - \underbrace{\frac{\varrho}{\Delta}\mathbf{k}_{t+1}^\mathsf{T}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}}_{=\frac{\varrho}{\Delta}(\Delta-1)=\varrho-\frac{\varrho}{\Delta}}\end{bmatrix}\\
&= \begin{bmatrix}\mathbf{y}_{tm}^{\text{res}} - \frac{\varrho}{\Delta}\mathbf{K}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\\ \frac{\varrho}{\Delta}\end{bmatrix}
\end{aligned}$$

Somit erhalten wir schließlich für $\xi_{t+1,m}$:

$$\begin{aligned}
\xi_{t+1,m} &= \begin{bmatrix}\mathbf{y}_t\\ y^*\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{y}_{tm}^{\text{res}} - \frac{\varrho}{\Delta}\mathbf{K}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\\ \frac{\varrho}{\Delta}\end{bmatrix} = \overbrace{\mathbf{y}_t^\mathsf{T}\mathbf{y}_{tm}^{\text{res}}}^{\xi_{tm}} - \frac{\varrho}{\Delta}\mathbf{k}_{t+1}^\mathsf{T}\overbrace{\mathbf{P}_{tm}^{-1}\mathbf{K}_{tm}^\mathsf{T}\mathbf{y}_t}^{\mathbf{w}_{tm}} + y^*\frac{\varrho}{\Delta}\\
&= \xi_{tm} + \frac{\varrho}{\Delta}(y^* - \mathbf{k}_{t+1}^\mathsf{T}\mathbf{w}_{tm}) = \xi_{tm} + \frac{\varrho^2}{\Delta}
\end{aligned}$$

Zusammengenommen erhalten wir die im Bereich des Adaptiven Filterns als Recursive Least-Squares (RLS) bekannten Aktualisierungen (Sayed, 2003):

Lemma 1 (Recursive Least-Squares). *Gegeben das regularisierte Least-Squares Problem für t Daten*

$$\min_{\mathbf{w}\in\mathbb{R}^m} J_{tm}(\mathbf{w}) = \|\mathbf{y}_t - \mathbf{K}_{tm}\mathbf{w}\|^2 + \lambda \mathbf{w}^T \mathbf{K}_{mm}\mathbf{w}$$

mit Lösung $\mathbf{w}_{tm}$, der inversen Kreuzproduktmatrix $\mathbf{P}_{tm}^{-1} = (\mathbf{K}_{tm}^T\mathbf{K}_{tm} + \lambda\mathbf{K}_{mm})^{-1}$ und den Kosten $\xi_{tm} = J_{tm}(\mathbf{w}_{tm})$. Für das um das neue Beispiel $(\mathbf{x}_{t+1}, y^*)$ erweiterte Problem

$$\min_{\mathbf{w}\in\mathbb{R}^m} J_{t+1,m}(\mathbf{w}) = \left\| \begin{bmatrix}\mathbf{y}_t \\ y^*\end{bmatrix} - \begin{bmatrix}\mathbf{K}_{tm} \\ \mathbf{k}_{t+1}^T\end{bmatrix}\mathbf{w}\right\|^2 + \lambda \mathbf{w}^T\mathbf{K}_{mm}\mathbf{w}$$

lassen sich die entsprechenden Größen $\{\mathbf{w}_{t+1,m}, \mathbf{P}_{t+1,m}^{-1}, \xi_{t+1,m}\}$ rekursiv aus den vorhergehenden berechnen. Mit der Abkürzung $\Delta := 1 + \mathbf{k}_{t+1}^T\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}$ und $\varrho := y^* - \mathbf{k}_{t+1}^T\mathbf{w}_{tm}$ gilt:

- $\mathbf{w}_{t+1,m} = \mathbf{w}_{tm} + \frac{\varrho}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}$
- $\mathbf{P}_{t+1,m}^{-1} = \mathbf{P}_{tm}^{-1} - \frac{\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^T\mathbf{P}_{tm}^{-1}}{\Delta}$
- $\xi_{t+1,m} = \xi_{tm} + \frac{\varrho^2}{\Delta}$.

Der Aufwand der Aktualisierung ist $\mathcal{O}(m^2)$.

4.1.2 Basiserweiterungsschritt

Seien nach dem Hinzufügen eines Beispiels in Teilschritt 1 die Größen $\{\mathbf{P}_{t+1,m}^{-1}, \mathbf{w}_{t+1,m}, \xi_{t+1,m}\}$ bekannt, so betrachten wir jetzt den Fall, daß der Basiserweiterungstest in Teilschritt 2 erfolgreich ist, d.h. die gegenwärtige Basis um $\mathbf{x}_{t+1}$ erweitert werden soll und $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{\mathbf{x}_{t+1}\}$ gesetzt wird.

Betrachten wir wieder, wie sich in diesem Fall die involvierten Matrizen verändern. Zunächst fügen wir den $(t+1)\times 1$ Spaltenvektor $\mathbf{q}$ als Bild der $t+1$ Beispiele unter der neuen Basisfunktion der Datenmatrix $\mathbf{K}_{t+1,m}$ hinzu:

$$\mathbf{K}_{t+1,m+1} = \begin{bmatrix}\mathbf{K}_{t+1,m} & \mathbf{q}\end{bmatrix}.$$

Damit ist

$$\begin{aligned}\mathbf{P}_{t+1,m+1} &= \begin{bmatrix}\mathbf{K}_{t+1,m} & \mathbf{q}\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{K}_{t+1,m} & \mathbf{q}\end{bmatrix} + \lambda\begin{bmatrix}\mathbf{K}_{mm} & \mathbf{k}_{t+1} \\ \mathbf{k}_{t+1}^\mathsf{T} & k^*\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{P}_{t+1,m} & \mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{q} + \lambda\mathbf{k}_{t+1} \\ \mathbf{q}^\mathsf{T}\mathbf{K}_{t+1,m} + \lambda\mathbf{k}_{t+1}^\mathsf{T} & \mathbf{q}^\mathsf{T}\mathbf{q} + \lambda k^*\end{bmatrix}\end{aligned}$$

und mit der Formel für die partitionierte Matrixinverse (4.6) folgt für dessen Inverse

$$\mathbf{P}_{t+1,m+1}^{-1} = \begin{bmatrix}\mathbf{P}_{t+1,m}^{-1} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0\end{bmatrix} + \frac{1}{\Delta_b}\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix}\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix}^\mathsf{T} \tag{4.8}$$

wobei wir die Abkürzungen

$$\mathbf{w}_b := \mathbf{P}_{t+1,m}^{-1}(\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{q} + \lambda\mathbf{k}_{t+1}) \tag{4.9}$$

$$\begin{aligned}\Delta_b &:= \mathbf{q}^\mathsf{T}\mathbf{q} + \lambda k^* - (\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{q} + \lambda\mathbf{k}_{t+1})^\mathsf{T}\overbrace{\mathbf{P}_{t+1,m}^{-1}(\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{q} + \lambda\mathbf{k}_{t+1})}^{\mathbf{w}_b} \\ &= \mathbf{q}^\mathsf{T}\underbrace{(\mathbf{q} - \mathbf{K}_{t+1,m}\mathbf{w}_b)}_{=:\mathbf{q}_{t+1,m}^{\mathrm{res}}} + \lambda(k^* - \mathbf{k}_{t+1}^\mathsf{T}\mathbf{w}_b)\end{aligned} \tag{4.10}$$

definiert haben.

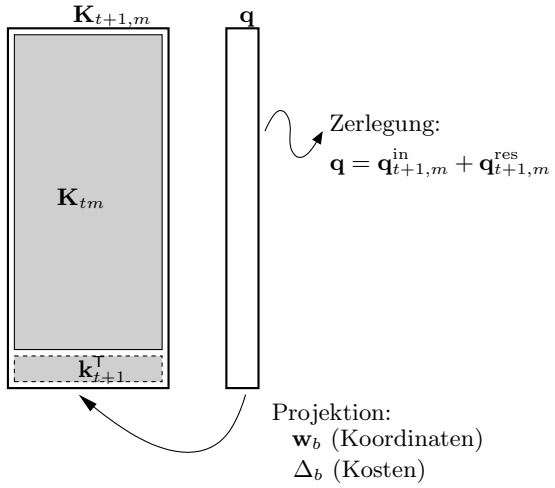


Abbildung 4.1: Für den Fall $\lambda = 0$ werden $\mathbf{w}_b, \Delta_b$ aus der Rückprojektion von der neuen Spalte $\mathbf{q}$ auf das Erzeugnis der bisherigen Spalten von $\mathbf{K}_{t+1,m}$ berechnet. Nur das (orthogonale) Residuum $\mathbf{q}_{t+1,m}^{\text{res}}$ trägt zur Reduktion des Fehlers im ursprünglichen Problem bei.

Anschaulich haben die Größen $\mathbf{w}_b, \Delta_b$ folgende Bedeutung (siehe auch Abbildung 4.1): für den speziellen Fall $\lambda = 0$ (gewöhnliches Least-Squares) ist $\mathbf{w}_b$ gerade der Vektor der Projektionskoordinaten der Projektion von $\mathbf{q}$ auf das Spaltenerzeugnis von $\mathbf{K}_{t+1,m}$ (das Erzeugnis der bisherigen Basisfunktionen), also selbst wieder Lösung eines Least-Squares Problems $\mathbf{w}_b = \text{argmin}_{\mathbf{z}} \ \|\mathbf{q} - \mathbf{K}_{t+1,m}\mathbf{z}\|^2$. Und Skalar $\Delta_b = \mathbf{q}^{\mathsf{T}}(\mathbf{q} - \mathbf{K}_{t+1,m}\mathbf{w}_b) =: \mathbf{q}^{\mathsf{T}}\mathbf{q}_{t+1,m}^{\text{res}}$ ist gerade die quadrierte Norm des Residuums, also die Kosten dieser Approximation. Die neu hinzugefügte Spalte $\mathbf{q}$ wird demnach wieder in einen Anteil $\mathbf{q}_{t+1,m}^{\text{in}} := \mathbf{K}_{t+1,m}\mathbf{w}_b$ und

$$\mathbf{q}_{t+1,m}^{\text{res}} := \mathbf{q} - \mathbf{K}_{t+1,m}\mathbf{w}_b \tag{4.11}$$

zerlegt. Für $\lambda = 0$ ist $\Delta_b = \|\mathbf{q}_{t+1,m}^{\text{res}}\|^2$. Es wird sich später zeigen, daß nur der Anteil $\mathbf{q}_{t+1,m}^{\text{res}}$ zur Reduktion des Fehlers ξ_{tm} aus (4.3) beitragen wird. Sollten wir also im weiteren Verlauf einmal die Möglichkeit haben, das $\mathbf{q}$ aus mehreren Alternativen auszuwählen, so kann es eine gute Idee sein, dasjenige zu nehmen, welches den verbleibenden Fehler am meisten reduziert. Doch dazu in Abschnitt 4.2 mehr.

Aus der Rekursion für $\mathbf{P}_{t+1,m+1}^{-1}$ kann nun leicht wieder eine Aktualisierung für $\mathbf{w}_{t+1,m+1}$ hergeleitet werden. Multiplizieren wir (4.8) mit $\mathbf{K}_{t+1,m+1}^{\mathsf{T}}\mathbf{y}_{t+1}$, dann folgt

$$\begin{aligned}
\mathbf{w}_{t+1,m+1} &= \mathbf{P}_{t+1,m+1}^{-1}\begin{bmatrix}\mathbf{K}_{t+1,m} & \mathbf{q}\end{bmatrix}^{\mathsf{T}}\mathbf{y}_{t+1} \\
&= \left(\begin{bmatrix}\mathbf{P}_{t+1,m}^{-1} & \mathbf{0} \\ \mathbf{0}^{\mathsf{T}} & 0\end{bmatrix} + \frac{1}{\Delta_b}\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix}\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix}^{\mathsf{T}}\right)\begin{bmatrix}\mathbf{K}_{t+1,m}^{\mathsf{T}}\mathbf{y}_{t+1} \\ \mathbf{q}^{\mathsf{T}}\mathbf{y}_{t+1}\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{w}_{t+1,m} \\ 0\end{bmatrix} + \frac{1}{\Delta_b}(-\mathbf{w}_b^{\mathsf{T}}\mathbf{K}_{t+1,m}^{\mathsf{T}}\mathbf{y}_{t+1} + \mathbf{q}^{\mathsf{T}}\mathbf{y}_{t+1})\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{w}_{t+1,m} \\ 0\end{bmatrix} + \frac{\mathbf{y}_{t+1}^{\mathsf{T}}\overbrace{(\mathbf{q} - \mathbf{K}_{t+1,m}\mathbf{w}_b)}^{\mathbf{q}_{t+1,m}^{\text{res}}}}{\Delta_b}\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{w}_{t+1,m} \\ 0\end{bmatrix} + \kappa\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix},
\end{aligned}$$

wobei wir zur Abkürzung

$$\kappa := \frac{\mathbf{y}_{t+1}^{\mathsf{T}}\mathbf{q}_{t+1,m}^{\text{res}}}{\Delta_b} \tag{4.12}$$

definiert haben (im Falle von gewöhnlichem Least-squares ist $\kappa = \mathbf{y}_{t+1}^{\mathsf{T}}\mathbf{q}_{t+1,m}^{\text{res}}/\|\mathbf{q}_{t+1,m}^{\text{res}}\|^2$).

Bei Hinzunahme einer neuen Spalte $\mathbf{q}$ verringert sich der Fehler der Approximation, auch $\xi_{t+1,m+1} = \mathbf{y}_{t+1}^{\mathsf{T}}\mathbf{y}_{t+1,m+1}^{\text{res}}$ läßt sich rekursiv aus $\xi_{t+1,m}$ erzeugen. Zunächst betrachten wir wieder die Aktualisierung des Residuums $\mathbf{y}_{t+1,m+1}^{\text{res}}$:

$$\begin{aligned}\mathbf{y}_{t+1,m+1}^{\text{res}} &= \mathbf{y}_{t+1} - \begin{bmatrix}\mathbf{K}_{t+1,m} & \mathbf{q}\end{bmatrix}\begin{bmatrix}\mathbf{w}_{t+1,m} - \kappa\mathbf{w}_b \\ \kappa\end{bmatrix} \\ &= \overbrace{\mathbf{y}_{t+1} - \mathbf{K}_{t+1,m}\mathbf{w}_{t+1,m}}^{\mathbf{y}_{t+1,m}^{\text{res}}} + \kappa\mathbf{K}_{t+1,m}\mathbf{w}_b - \kappa\mathbf{q} \\ &= \mathbf{y}_{t+1,m}^{\text{res}} - \kappa\mathbf{q}_{t+1,m}^{\text{res}}\end{aligned}$$

Somit erhalten wir schließlich für $\xi_{t+1,m+1}$ die Aktualisierung

$$\begin{aligned}\xi_{t+1,m+1} &= \mathbf{y}_{t+1}^{\mathsf{T}}(\mathbf{y}_{t+1,m}^{\text{res}} - \kappa\mathbf{q}_{t+1,m}^{\text{res}}) = \xi_{t+1,m} - \kappa\overbrace{\mathbf{y}_{t+1}^{\mathsf{T}}\mathbf{q}_{t+1,m}^{\text{res}}}^{\kappa\cdot\Delta_b} \\ &= \xi_{t+1,m} - \kappa^2\Delta_b.\end{aligned}$$

Zusammenfassend gelten die folgenden Aktualisierungen:

Lemma 2 (Hinzufügen einer Spalte). *Gegeben das regularisierte Least-Squares Problem mit $t \times m$ Datenmatrix $\mathbf{K}_{t+1,m}$*

$$\min_{\mathbf{w}\in\mathbb{R}^m} J_{t+1,m}(\mathbf{w}) = \|\mathbf{y}_{t+1} - \mathbf{K}_{t+1,m}\mathbf{w}\|^2 + \lambda\mathbf{w}^T\mathbf{K}_{mm}\mathbf{w}$$

und dessen Lösung $\mathbf{w}_{t+1,m}$, der inversen Kreuzproduktmatrix $\mathbf{P}_{t+1,m}^{-1} = (\mathbf{K}_{t+1,m}^T\mathbf{K}_{t+1,m} + \lambda\mathbf{K}_{mm})^{-1}$ und den Kosten $\xi_{t+1,m} = J_{t+1,m}(\mathbf{w}_{t+1,m})$. Für das um eine neue Basisfunktion, d.h. $(t+1)\times 1$ Vektor $\mathbf{q}$, erweiterte Problem

$$\min_{\mathbf{w}\in\mathbb{R}^{m+1}} J_{t+1,m+1}(\mathbf{w}) = \left\|\mathbf{y}_{t+1} - \begin{bmatrix}\mathbf{K}_{t+1,m} & \mathbf{q}\end{bmatrix}\mathbf{w}\right\|^2 + \lambda\mathbf{w}^T\begin{bmatrix}\mathbf{K}_{mm} & \mathbf{k}_{t+1} \\ \mathbf{k}_{t+1}^T & k^*\end{bmatrix}\mathbf{w}$$

lassen sich die entsprechenden Größen $\mathbf{w}_{t+1,m+1}, \mathbf{P}_{t+1,m+1}^{-1}, \xi_{t+1,m+1}$ rekursiv aus den vorhergehenden berechnen. Mit den Abkürzungen

- $\mathbf{w}_b = \mathbf{P}_{t+1,m}^{-1}(\mathbf{K}_{t+1,m}^T\mathbf{q} + \lambda\mathbf{k}_{t+1})$
- $\mathbf{q}_{t+1,m}^{\text{res}} = \mathbf{q} - \mathbf{K}_{t+1,m}\mathbf{w}_b$
- $\Delta_b = \mathbf{q}^T\mathbf{q}_{t+1,m}^{\text{res}} + \lambda(k^* - \mathbf{k}_{t+1}^T\mathbf{w}_b)$
- $\kappa = \frac{\mathbf{y}_{t+1}^T\mathbf{q}_{t+1,m}^{\text{res}}}{\Delta_b}$

aus der Projektion von $\mathbf{q}$ auf den Spaltenraum von $\mathbf{K}_{t+1,m}$ (der 'Rückprojektion') gilt:

- $\mathbf{w}_{t+1,m+1} = \begin{bmatrix}\mathbf{w}_{t+1,m} \\ 0\end{bmatrix} + \kappa\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix}$
- $\mathbf{P}_{t+1,m+1}^{-1} = \begin{bmatrix}\mathbf{P}_{t+1,m}^{-1} & \mathbf{0} \\ \mathbf{0} & 0\end{bmatrix} + \frac{1}{\Delta_b}\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix}\begin{bmatrix}-\mathbf{w}_b \\ 1\end{bmatrix}^T$
- $\xi_{t+1,m+1} = \xi_{t+1,m} - \kappa^2\Delta_b$

Der Aufwand der Aktualisierung ist $\mathcal{O}(tm)$.

Das Hauptproblem dieser Aktualisierung ist der Rechenaufwand zum Bestimmen von $\mathbf{w}_b, \Delta_b, \kappa$: dieser beträgt $\mathcal{O}(tm)$ und ist für einen Online-Algorithmus natürlich völlig inakzeptabel. Vernünftigerweise

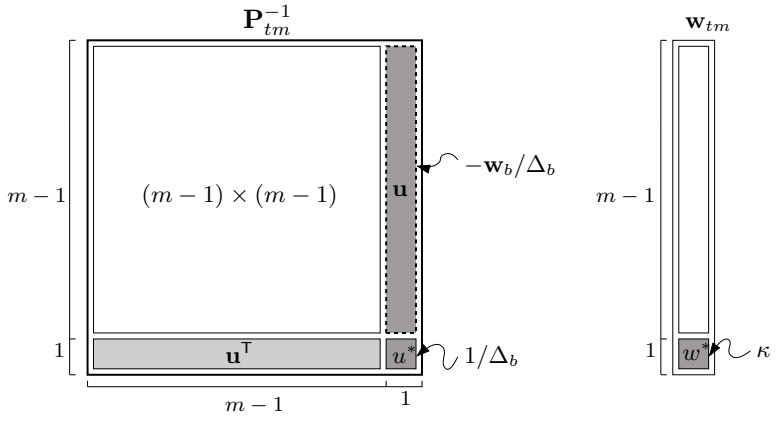


Abbildung 4.2: Die zum Löschen einer Basisfunktion benötigten Größen $\Delta_b, \mathbf{w}_b, \kappa$ lassen sich direkt aus dem aktuellen Modell mit m Basisfunktionen ablesen.

sollte der pro-Schritt Aufwand unabhängig von der Anzahl t der bisher durchgeführten Schritte sein. In Abschnitt 4.2 werden wir deswegen eine Variante vorstellen, wodurch dieses Manko behoben wird, indem wir die Approximation des Kerns aus Kapitel 3 ausnutzen, um die meisten Einträge von $\mathbf{q}$ durch eine Näherung zu ersetzen. Das Resultat wird dann nicht mehr von t abhängen, der Rechenaufwand pro Schritt wird konstant sein.

Zunächst betrachten wir aber noch die Operation zum Löschen einer Basisfunktion für Teilschritt 4.

4.1.3 Löschen einer beliebigen Basisfunktion

Zur Notation: wir betrachten zunächst nur den Fall, daß wir die zuletzt hinzugefügte Basisfunktion aus einem Modell mit m Basisfunktionen und t Beispielen entfernen wollen, d.h. wir betrachten den Übergang von $\{t, m\}$ nach $\{t, m-1\}$ für die zugehörigen Größen $\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}$.

Um $\mathbf{P}_{t,m-1}^{-1}$ aus $\mathbf{P}_{tm}^{-1}$ zu gewinnen, betrachten wir die Aktualisierung aus Lemma 2 von $\mathbf{P}_{t+1,m}^{-1}$ zu $\mathbf{P}_{t+1,m+1}^{-1}$ und drehen sie einfach um:

$$\begin{bmatrix} \mathbf{P}_{t,m-1}^{-1} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix} = \mathbf{P}_{tm}^{-1} - \frac{1}{\Delta_b} \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix}^\mathsf{T}. \tag{4.13}$$

Um die letzte Basisfunktion aus $\mathbf{P}_{tm}^{-1}$ zu löschen, benötigen wir also Kenntnis von Δ_b und $\mathbf{w}_b$. Woher bekommen wir diese? Die Lösung ist sehr einfach, denn beide Größen lassen sich direkt aus $\mathbf{P}_{tm}^{-1}$ ablesen: nach der Aktualisierung durch Lemma 2 gilt

$$\mathbf{P}_{tm}^{-1} = \left[\begin{array}{c|c} \mathbf{P}_{t,m-1}^{-1} + \Delta_b^{-1}\mathbf{w}_b\mathbf{w}_b^\mathsf{T} & -\Delta_b^{-1}\mathbf{w}_b \\ \hline -\Delta_b^{-1}\mathbf{w}_b^\mathsf{T} & \Delta_b^{-1} \end{array}\right],$$

so daß Δ_b gerade aus dem letzen Diagonalelement und $\mathbf{w}_b$ aus der letzten Spalte gewonnen werden kann (Abbildung 4.2). Mit dem $(m-1)\times 1$ Vektor $\mathbf{u} := \mathbf{P}_{tm}^{-1}(1:m-1, m)$, d.h. den ersten $m-1$ Einträgen der m-ten Spalte von $\mathbf{P}_{tm}^{-1}$, und mit $u^* := \mathbf{P}_{tm}^{-1}(m,m)$, d.h. dem m-ten Diagonalelement von $\mathbf{P}_{tm}^{-1}$, ist

$$\begin{aligned} \Delta_b &= 1/u^* \\ \mathbf{w}_b &= -\mathbf{u}\cdot\Delta_b. \end{aligned}$$

Eingesetzt in (4.13) ergibt sich

$$\begin{bmatrix} \mathbf{P}_{t,m-1}^{-1} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix} = \mathbf{P}_{tm}^{-1} - u^* \begin{bmatrix} -\mathbf{u}/u^* \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{u}/u^* \\ 1 \end{bmatrix}^\mathsf{T}.$$

Wenn nun *Trunc* eine Operation bezeichnet, bei der die letzte Zeile und Spalte einer Matrix (bzw. eines Vektors) gelöscht wird (aus einer $m \times m$ Matrix also eine $(m-1) \times (m-1)$ Matrix wird), können wir das auch kurz schreiben als

$$\mathbf{P}_{t,m-1}^{-1} = \text{Trunc}(\mathbf{P}_{tm}^{-1}) - \frac{\mathbf{u}\mathbf{u}^T}{u^*}.$$

Auf ähnliche Weise können wir auch die neue Lösung $\mathbf{w}_{t,m-1}$ aus $\mathbf{w}_{tm}$ erhalten. Wie zuvor drehen wir dazu einfach die entsprechende Aktualisierung aus Lemma 2 von $\mathbf{w}_{t+1,m}$ nach $\mathbf{w}_{t+1,m+1}$ um:

$$\begin{bmatrix} \mathbf{w}_{t,m-1} \\ 0 \end{bmatrix} = \mathbf{w}_{tm} + \ \kappa \begin{bmatrix} \mathbf{w}_b \\ -1 \end{bmatrix}.$$

Das benötigte κ können wir daher ebenfalls wieder direkt aus $\mathbf{w}_{tm}$ ablesen, es ist gerade die letzte Komponente $w^* := \mathbf{w}_{tm}(m)$ von $\mathbf{w}_{tm}$ (Abbildung 4.2). Somit ist

$$\mathbf{w}_{t,m-1} = \text{Trunc}(\mathbf{w}_{tm}) - w^* \cdot \mathbf{u}/u^*.$$

Um nun auch noch die neuen Kosten zu gewinnen, drehen wir schließlich auch die entsprechende Aktualisierung aus Lemma 2 von $\xi_{t+1,m}$ nach $\xi_{t+1,m+1}$ um:

$$\xi_{t,m-1} = \xi_{tm} + \kappa^2 \cdot \Delta_b$$

bzw.

$$\xi_{t,m-1} = \xi_{tm} + (w^*)^2/u^*.$$

Alle notwendigen Änderungen um $\mathbf{P}_{t,m-1}^{-1}, \mathbf{w}_{t,m-1}, \xi_{t,m-1}$ aus $\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}$ zu gewinnen, sind nun beisammen. Um jetzt aber nicht nur die letzte, sondern eine beliebige Basisfunktion $i \in \{1, \ldots, m\}$ aus dem Modell zu löschen, tauschen wir zunächst die Position der Basisfunktionen i und m. Für $\mathbf{P}_{tm}^{-1}$ ist das gleichbedeutend mit einem Zeilen-und Spaltentausch (Multiplikation von links und rechts mit einer Transpositionsmatrix $\mathbf{T}_{i \leftrightarrow m}$) und für $\mathbf{w}_{tm}$ mit einem Zeilentausch (Multiplikation von links mit $\mathbf{T}_{i \leftrightarrow m}$). Damit haben wir das Problem auf ein bereits bekanntes zurückgeführt und erhalten $\mathbf{P}_{t,m\setminus i}^{-1}, \mathbf{w}_{t,m\setminus i}, \xi_{t,m\setminus i}$ einfach wieder durch Löschen der letzten Basisfunktion.

Da wir nun beliebige Basisfunktionen aus einem bestehenden Modell entfernen können, stellt sich als nächstes natürlich die Frage, welche wir dazu am besten nehmen. Eine Möglichkeit ist es, diejenige zu entfernen, die am wenigsten zur Lösung des Problems beiträgt. Dazu gehen wir wie folgt vor: wir rechnen für jeden Index i aus, um wieviel sich die Kosten ξ_{tm} vergrößern würden, wenn die i-te Basisfunktion entfernt werden würde. Dazu müssen wir einfach die Änderung an den Kosten, also $\kappa^2\Delta_b$, für jeden möglichen Kandidaten ausrechnen. Da wir durch Zeilen-und Spaltentausch jeden Index i an das Ende schieben können (und $\kappa^2\Delta_b$ in diesem Fall $[\mathbf{w}_{tm}(m)]^2/\mathbf{P}_{tm}^{-1}(m,m)$ ist), folgt, daß der Anstieg der Kosten für jedes i durch

$$\frac{[\mathbf{w}_{tm}(i)]^2}{\mathbf{P}_{tm}^{-1}(i,i)}, \quad i = 1, \ldots, m \tag{4.14}$$

ausgerechnet werden kann. Somit lautet das Auswahlkriterium: lösche diejenige Basisfunktion i, die in (4.14) das Minimum erzielt, also diejenige Basisfunktion, die die Kosten am wenigsten ansteigen läßt, wenn sie entfernt wird. Der Rechenaufwand zum Finden dieses Kandidaten ist sehr gering, weil für jeden der m möglichen Kandidaten lediglich (4.14) ausgewertet werden muß.

Zusammengefaßt gilt für das Löschen einer Basisfunktion:

Lemma 3 (Löschen einer Spalte). *Gegeben sei das regularisierte Least-Squares Problem in m Basisfunktionen mit $t \times m$ Datenmatrix $\mathbf{K}_{tm}$*

$$\min_{\mathbf{w}\in\mathbb{R}^m} \; J_{tm}(\mathbf{w}) = \|\mathbf{y}_t - \mathbf{K}_{tm}\mathbf{w}\|^2 + \lambda \mathbf{w}^T \mathbf{K}_{mm}\mathbf{w}$$

und dessen Lösung $\mathbf{w}_{tm}$, der inversen Kreuzproduktmatrix $\mathbf{P}_{tm}^{-1} = (\mathbf{K}_{tm}^T\mathbf{K}_{tm} + \lambda\mathbf{K}_{mm})^{-1}$ und den Kosten $\xi_{tm} = J_{tm}(\mathbf{w}_{tm})$. Sei $\mathbf{K}_{t,m\setminus i}$ die $t \times (m-1)$ Datenmatrix, in der die i-te Basisfunktion entfernt wurde. Die Lösung für die Aufgabe mit $m\setminus i$

$$\min_{\mathbf{w}\in\mathbb{R}^{m-1}} J_{t,m\setminus i}(\mathbf{w}) = \left\|\mathbf{y}_t - \mathbf{K}_{t,m\setminus i}\mathbf{w}\right\|^2 + \lambda \mathbf{w}^T \mathbf{K}_{m\setminus i,m\setminus i}\mathbf{w}$$

kann rekursiv aus der vorhergehenden Lösung gewonnen werden. Mit der $m \times m$ Transpositionsmatrix $\mathbf{T}_{i\leftrightarrow m}$ und den Größen

- $\mathbf{u} = [\mathbf{T}_{i\leftrightarrow m}\mathbf{P}_{tm}^{-1}\mathbf{T}_{i\leftrightarrow m}](1:m-1,m)$ *(die $m-1$ Zeilen der m-ten Spalte)*
- $u^* = [\mathbf{T}_{i\leftrightarrow m}\mathbf{P}_{tm}^{-1}\mathbf{T}_{i\leftrightarrow m}](m,m)$ *(das m-te Diagonalelement)*
- $w^* = [\mathbf{T}_{i\leftrightarrow m}\mathbf{w}_{tm}](m)$ *(das m-te Element)*

gilt:

- $\mathbf{P}_{t,m\setminus i}^{-1} = \text{Trunc}(\mathbf{T}_{i\leftrightarrow m}\mathbf{P}_{tm}^{-1}\mathbf{T}_{i\leftrightarrow m}) - \frac{\mathbf{u}\mathbf{u}^T}{u^*}$
- $\mathbf{w}_{t,m\setminus i} = \text{Trunc}(T_{i\leftrightarrow m}\mathbf{w}_{tm}) - w^* \cdot \mathbf{u}/u^*$
- $\xi_{t,m\setminus i} = \xi_{tm} + (w^*)^2/u^*$

Der Aufwand der Aktualisierung ist $\mathcal{O}(m^2)$.

4.2 Inkrementelle Regularisierungsnetze (wie es funktioniert)

Das Hauptproblem der bisher vorgeschlagenen Lösungsprozedur ist, daß einerseits der Teilschritt Basiserweiterung eine $\mathcal{O}(tm)$ Operation (und damit nicht unabhängig von der Anzahl bisher gesehener Daten t) ist, andererseits die m Basisfunktionen für SR aufgrund der Online-Natur des Lernszenarios aber auch nur zur Laufzeit ausgewählt werden können. In diesem Abschnitt kümmern wir uns daher darum, dieses Problem zu beheben.

4.2.1 Auswahl geeigneter Basisfunktionen durch gierige Online-Selektion

Betrachten wir zuerst die sequentielle Auswahl relevanter Basisfunktionen im Fall von Online-Lernen. Klar ist, daß in dieser Situation die in Kapitel 3 beschriebenen Verfahren partielles G.-S./ICD nicht mehr direkt anwendbar sind, weil diese die Kenntnis aller Trainingsdaten voraussetzen: in jedem Auswahlschritt wird aus *allen* bisher nicht-selektierten Daten dasjenige Element als nächstes hinzugenommen, das am schlechtesten durch die bisher ausgewählten repräsentiert wird. Mit einer leichten Modifikation läßt sich dieses Prinzip aber auch auf ein Online-Szenario übertragen. Die von Csató und Opper (2001) im Rahmen von GP-Regression vorgeschlagene *spärliche gierige Online-Selektion*, die sich in gleicher Form auch bei den Arbeiten von Engel et al. (2003, 2004, 2005a) wiederfindet, überprüft in jedem Zeitschritt t nur, ob das *aktuelle* Beispiel von den bisher ausgewählten Basiselementen gut repräsentiert wird. Immer dann wenn das nicht der Fall ist, d.h. die Norm des Residuums der Projektion auf den Unterraum der bisher ausgewählten Elemente eine vorgegebene Schranke überschreitet, wird das aktuelle Beispiel als neue Basisfunktion aufgenommen.

Spärliche gierige Online-Selektion verwendet demnach das gleiche Kriterium wie partielles G.-S. (nämlich den Abstand des zu untersuchenden Kandidaten von den bisher ausgewählten) mit dem Unterschied, daß

- jedes Beispiel nur *ein einziges mal* untersucht wird, während bei partiellem G.-S. in jedem Auswahlschritt jeweils *alle verbleibenden* Kandidaten überprüft werden
- die resultierende Approximation des Kerns nur die *zum jeweiligen Zeitpunkt* ausgewählten Basisfunktionen berücksichtigt. Später hinzugefügte Basisfunktionen verbessern also nicht rückwirkend frühere Approximationen (und aus genau diesem Grund können wir die Basiserweiterung auch so effizient mit konstantem Zeitaufwand durchführen).

Online-Selektion

Die Online-Selektion läuft folgendermaßen ab: sei $\mathcal{D}_t = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_m\}$ die Menge der m ausgewählten Basisfunktionen zum Zeitpunkt t und $(\mathbf{x}_{t+1}, y^*)$ das nächste Trainingsbeispiel. Wir rechnen zunächst den Abstand $\delta_{t+1} := \delta_{\mathbf{x}_{t+1}}$ von $\boldsymbol{\phi}(\mathbf{x}_{t+1})$ zu $\text{span}\{\boldsymbol{\phi}(\tilde{\mathbf{x}}_1), \ldots, \boldsymbol{\phi}(\tilde{\mathbf{x}}_m)\}$ aus, was nach (3.5),(3.6) gerade gleich

$$\delta_{t+1} = k^* - \mathbf{k}_{t+1}^\mathsf{T} \mathbf{a}_{t+1} \tag{4.15}$$

mit

$$\mathbf{a}_{t+1} = \mathbf{K}_{mm}^{-1} \mathbf{k}_{t+1} \tag{4.16}$$

ist. Gilt nun $\delta_{t+1} \leq \texttt{TOL}$,wobei TOL ein vorher festgelegter Schwellenwert ist (z.B. $\texttt{TOL} = 10^{-1}$ oder $\texttt{TOL} = 10^{-2}$), so wird $\mathbf{x}_{t+1}$ im Merkmalsraum hinreichend gut durch die bisher ausgewählten Elemente in $\mathcal{D}_t$ repräsentiert:

$$\boldsymbol{\phi}(\mathbf{x}_{t+1}) = \overbrace{\sum_{i=1}^{m} a_{t+1}^{(i)} \boldsymbol{\phi}(\tilde{\mathbf{x}}_i)}^{=:\hat{\boldsymbol{\phi}}(\mathbf{x}_{t+1})} + \hat{\boldsymbol{\phi}}(\mathbf{x}_{t+1})^\perp, \tag{4.17}$$

wobei für den Fehler der Approximation $\|\hat{\boldsymbol{\phi}}(\mathbf{x}_{t+1})^\perp\|^2 = \delta_{t+1} \leq \texttt{TOL}$ gilt (vgl. Abbildung 3.2). In diesem Fall ist es nicht notwendig, die Basis zusätzlich zu erweitern, sondern es reicht aus, lediglich mit der projizierten Darstellung $\hat{\boldsymbol{\phi}}(\mathbf{x}_{t+1})$ weiterzurechnen und nur den Teilschritt 'Normal' (Abschnitt 4.1.1) auszuführen. Ist dagegen $\delta_{t+1} > \texttt{TOL}$, so heißt das, daß das neue Beispiel (im Merkmalsraum) nicht gut genug durch die bisher ausgewählten Basiselemente repräsentiert wird. In diesem Fall führen wir zusätzlich den Teilschritt 'Basiserweiterung' (aus Abschnitt 4.1.2) aus, der eine neue Basisfunktion, zentriert auf $\mathbf{x}_{t+1}$ hinzufügt.

Um diese Online-Selektion algorithmisch durchzuführen, müssen wir in jedem Zeitschritt t lediglich die Koeffizienten $\mathbf{a}_{t+1}$ aus (4.16) ausrechnen. Dazu speichern und verwalten wir die Inverse $\mathbf{K}_{mm}^{-1}$ (oder alternativ den Cholesky-Faktor) und aktualisieren sie nach jeder Basiserweiterung. Mit (4.6) folgt dafür

$$\mathbf{K}_{m+1,m+1}^{-1} = \begin{bmatrix} \mathbf{K}_{mm} & \mathbf{k}_{t+1} \\ \mathbf{k}_{t+1}^\mathsf{T} & k^* \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{K}_{mm}^{-1} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix} + \frac{1}{\delta_{t+1}} \begin{bmatrix} -\mathbf{a}_{t+1} \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{a}_{t+1} \\ 1 \end{bmatrix}^\mathsf{T}. \tag{4.18}$$

Insgesamt ist die Menge der selektierten Daten beschränkt. In Engel et al. (2004) wird gezeigt, daß selbst für einen unendlichen Datenstrom für jeden Wert von $\texttt{TOL} > 0$ immer nur endlich viele Elemente selektiert werden[2] und daß diese Anzahl durch $const \cdot \texttt{TOL}^{-d}$ (wobei d die Dimension der Inputs $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$) beschränkt ist.

Online-Approximation

Das Problem beim Teilschritt Basiserweiterung besteht darin, daß eine neue Spalte $\mathbf{q}$ mit t neuen Einträgen an $\mathbf{K}_{tm}$ angehängt wird, und daher der Rechenaufwand zur Aktualisierung von $\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}$ ebenfalls von t abhängt (Lemma 2). Die Idee von Csató und Opper (2001) ist es, die Einträge der neu dazugefügten

[2] Beweisskizze: Für Eingaberaum $\mathcal{X} \subset \mathbb{R}^d$ kompakt ist auch $\phi(\mathcal{X}) := \{\phi(\mathbf{x}) \,|\, \mathbf{x} \in \mathcal{X}\}$ kompakt. Daher gibt es eine endliche ε-Überdeckung von $\phi(\mathcal{X})$, was bedeutet, daß die Anzahl der ε-separierten Elemente (d.h. Punkte, die untereinander jeweils Abstand $\geq \varepsilon$ haben) ebenfalls endlich ist. Da nur solche Elemente als neue Basiselemente selektiert werden, die mindesten $\sqrt{\texttt{TOL}}$ von allen bisher gewählten entfernt sind, folgt für $\varepsilon := \sqrt{\texttt{TOL}}$, daß die Anzahl dieser Elemente endlich sein muß.

Spalte zu approximieren. Betrachten wir Abbildung 4.3, worin diese Situation für $\mathbf{K}_{t+1,m+1}$ skizziert wird: wir approximieren alle zur Vergangenheit $i = 1, \ldots, t$ gehörenden Einträge $\bar{\mathbf{q}}$ wie in (3.7) durch die gegenwärtig selektierten Basiselemente:

$$k(\mathbf{x}_i, \mathbf{x}_{t+1}) \overset{(3.7)}{\approx} \mathbf{k}_m(\mathbf{x}_i)^\mathsf{T} \underbrace{\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}_{t+1})}_{=\mathbf{a}_{t+1}\ (4.16)} \qquad i = 1, \ldots, t.$$

Somit ist $\bar{\mathbf{q}} \approx \mathbf{K}_{tm}\mathbf{a}_{t+1}$ und

$$\mathbf{q} = \begin{bmatrix} \bar{\mathbf{q}} \\ k^* \end{bmatrix} \approx \begin{bmatrix} \mathbf{K}_{tm}\mathbf{a}_{t+1} \\ k^* \end{bmatrix}, \tag{4.19}$$

wobei die Approximation für solche Indizes $i \in \{1 \ldots t\}$ exakt ist, die zu einem der m Elemente in $\mathcal{D}_t$ korrespondieren. Statt die Aktualisierung der Datenmatrix $\mathbf{K}_{tm}$ exakt, d.h. mittels

$$\mathbf{K}_{t+1,m} = \begin{bmatrix} \mathbf{K}_{tm} \\ \mathbf{k}_{t+1}^\mathsf{T} \end{bmatrix} \quad \text{(Beispiel dazu)} \qquad \mathbf{K}_{t+1,m+1} = \begin{bmatrix} \mathbf{K}_{tm} & \bar{\mathbf{q}} \\ \mathbf{k}_{t+1}^\mathsf{T} & k^* \end{bmatrix} \quad \text{(Basisfunktion dazu)}$$

auszuführen, definieren wir rekursiv eine Approximation $\tilde{\mathbf{K}}_{tm}$ für $\mathbf{K}_{tm}$ durch

$$\tilde{\mathbf{K}}_{t+1,m} = \begin{bmatrix} \tilde{\mathbf{K}}_{tm} \\ \mathbf{k}_{t+1}^\mathsf{T} \end{bmatrix} \quad \text{(Beispiel dazu)} \qquad \tilde{\mathbf{K}}_{t+1,m+1} = \begin{bmatrix} \tilde{\mathbf{K}}_{tm} & \tilde{\mathbf{K}}_{tm}\mathbf{a}_{t+1} \\ \mathbf{k}_{t+1}^\mathsf{T} & k^* \end{bmatrix} \quad \text{(Basisfunktion dazu)}$$

mit der Anfangsinitialisierung $\tilde{\mathbf{K}}_{11} := \mathbf{K}_{11} = [k(\mathbf{x}_1, \tilde{\mathbf{x}}_1)]$. Ersetzen wir nun $\mathbf{K}_{tm}$ in (3.9), (3.10) durch $\tilde{\mathbf{K}}_{tm}$, dann hat das den Effekt, daß bei jeder Basiserweiterung anstatt t neuen Informationen (entspricht der Vergangenheit aller bisher gesehenen Daten) nur m neue Informationen (entspricht dem aktuell beobachteten Trainingsbeispiel und den Basiselementen in $\mathcal{D}_t$) in das Modell injiziert wird. Damit wird es uns möglich, den Basiserweiterungsschritt (Abschnitt 4.1.2) effizient und mit konstantem Zeitaufwand unabhängig von t durchzuführen:

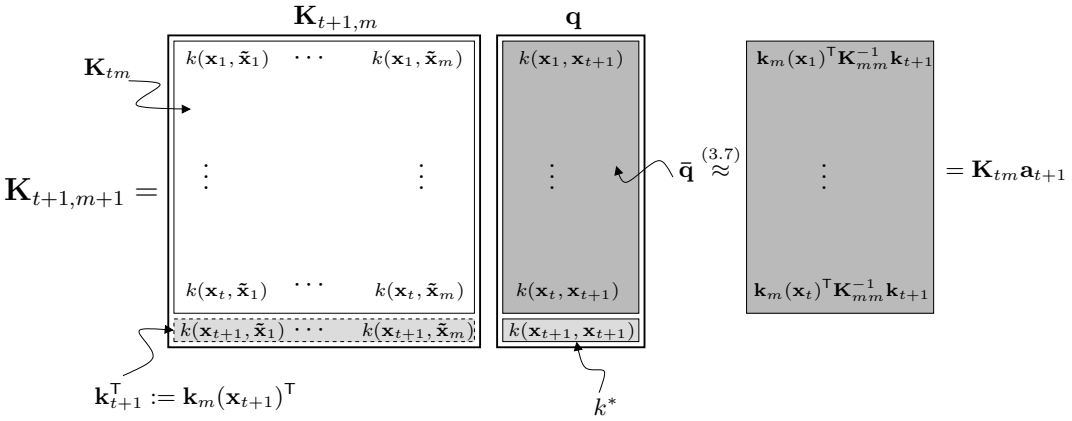


Abbildung 4.3: Aktualisierung von $\mathbf{K}_{tm}$, wenn ein neues Beispiel (d.h. Zeilenvektor $\mathbf{k}_{t+1}^\mathsf{T}$) oder eine neue Basisfunktion (d.h. Spaltenvektor $\mathbf{q}$) hinzugefügt wird. Die Einträge des neuen Spaltenvektors können durch die bisher selektierten Basiselemente mittels (3.7) approximiert werden, wodurch sich bei der Operation Hinzufügen einer Basisfunktion (Lemma 2) erhebliche Einsparungen ergeben.

4.2.2 Der modifizierte Basiserweiterungsschritt

Wir ersetzen nun $\mathbf{K}_{tm}$ in (3.9), (3.10) durch die Online-Approximation $\tilde{\mathbf{K}}_{tm}$ und überladen die Notation aus Abschnitt 4.1, indem wir

$$\mathbf{P}_{tm} := (\tilde{\mathbf{K}}_{tm}^\mathsf{T}\tilde{\mathbf{K}}_{tm} + \lambda\mathbf{K}_{mm}) \quad \text{und} \quad \mathbf{w}_{tm} := \mathbf{P}_{tm}^{-1}\tilde{\mathbf{K}}_{tm}\mathbf{y}_t$$

setzen. Alle übrigen Bezeichner bleiben unverändert und die beschriebenen Teilschritte können identisch übernommen werden. Lediglich die Basiserweiterung wird von der Online-Approximation betroffen.

Effiziente Basiserweiterung

Ersetzen wir $\mathbf{K}_{tm}$ durch die Online-Approximation $\tilde{\mathbf{K}}_{tm}$, so können wir die vormals nur mit von t abhängigem Zeitaufwand bestimmbaren Hilfsgrößen $\mathbf{w}_b$ aus (4.9), Δ_b aus (4.10) und κ aus (4.12) wesentlich effizienter ausrechnen.

Nach (4.9) gilt für $\mathbf{w}_b$:

$$\mathbf{w}_b = \mathbf{P}_{t+1,m}^{-1}(\tilde{\mathbf{K}}_{t+1,m}^{\mathsf{T}}\mathbf{q} + \lambda\mathbf{k}_{t+1}).$$

Nun ist

$$\begin{aligned}
\tilde{\mathbf{K}}_{t+1,m}^{\mathsf{T}}\mathbf{q} + \lambda\mathbf{k}_{t+1} &\overset{(4.19)}{=} \begin{bmatrix}\tilde{\mathbf{K}}_{tm}\\ \mathbf{k}_{t+1}^{\mathsf{T}}\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}\tilde{\mathbf{K}}_{tm}\mathbf{a}_{t+1}\\ k^*\end{bmatrix} + \lambda \overbrace{\mathbf{k}_{t+1}}^{\mathbf{K}_{mm}\mathbf{a}_{t+1}}\\
&= \tilde{\mathbf{K}}_{tm}^{\mathsf{T}}\tilde{\mathbf{K}}_{tm}\mathbf{a}_{t+1} + k^*\mathbf{k}_{t+1} + \lambda\mathbf{K}_{mm}\mathbf{a}_{t+1}\\
&= (\tilde{\mathbf{K}}_{tm}^{\mathsf{T}}\tilde{\mathbf{K}}_{tm} + \lambda\mathbf{K}_{mm})\mathbf{a}_{t+1} + k^*\mathbf{k}_{t+1}\\
&= \mathbf{P}_{tm}\mathbf{a}_{t+1} + k^*\mathbf{k}_{t+1}
\end{aligned}$$

so daß folgt

$$\begin{aligned}
\mathbf{w}_b &= \mathbf{P}_{t+1,m}^{-1}(\mathbf{P}_{tm}\mathbf{a}_{t+1} + k^*\mathbf{k}_{t+1})\\
&\overset{\text{Lemma 1}}{=} (\mathbf{P}_{tm}^{-1} - \Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{P}_{tm}^{-1})(\mathbf{P}_{tm}\mathbf{a}_{t+1} + k^*\mathbf{k}_{t+1})\\
&= \mathbf{a}_{t+1} + \underbrace{k^*\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}}_{\text{mit }\Delta\text{ erweitern}} - \Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{a}_{t+1} - \Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}k^*\\
&= \mathbf{a}_{t+1} + \Delta^{-1}k^*\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1} + \Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}k^*\\
&\qquad -\Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{a}_{t+1} - \Delta^{-1}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}k^*\\
&= \mathbf{a}_{t+1} + \Delta^{-1}\underbrace{(k^* - \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{a}_{t+1})}_{\delta_{t+1}}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}
\end{aligned}$$

und damit insgesamt gilt

$$\mathbf{w}_b = \mathbf{a}_{t+1} + \frac{\delta_{t+1}}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}. \tag{4.20}$$

Alle in (4.20) auftretenden Größen wurden bereits in einem der der Basiserweiterung vorausgehenden Teilschritte berechnet und können wiederverwendet werden.

Zum Bestimmen von $\Delta_b = \mathbf{q}^{\mathsf{T}}\mathbf{q}_{t+1,m}^{\text{res}} + \lambda(k^* - \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{w}_b)$ aus (4.10) betrachten wir zunächst die Berechnung von $\mathbf{q}_{t+1,m}^{\text{res}} := (\mathbf{q} - \tilde{\mathbf{K}}_{t+1,m}\mathbf{w}_b)$:

$$\begin{aligned}
\mathbf{q}_{t+1,m}^{\text{res}} &= \begin{bmatrix}\tilde{\mathbf{K}}_{tm}\mathbf{a}_{t+1}\\ k^*\end{bmatrix} - \begin{bmatrix}\tilde{\mathbf{K}}_{tm}\mathbf{w}_b\\ \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{w}_b\end{bmatrix}\\
&\overset{(4.20)}{=} \begin{bmatrix}\tilde{\mathbf{K}}_{tm}\mathbf{a}_{t+1}\\ k^*\end{bmatrix} - \begin{bmatrix}\tilde{\mathbf{K}}_{tm}(\mathbf{a}_{t+1} + \frac{\delta_{t+1}}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1})\\ \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{w}_b\end{bmatrix}\\
&= \frac{\delta_{t+1}}{\Delta}\begin{bmatrix}-\tilde{\mathbf{K}}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\\ (k^* - \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{w}_b)\cdot\Delta/\delta_{t+1}\end{bmatrix}
\end{aligned}$$

Eine kleine Nebenrechnung zeigt:

$$\begin{aligned}
k^* - \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{w}_b &\overset{(4.20)}{=} k^* - \mathbf{k}_{t+1}^{\mathsf{T}}(\mathbf{a}_{t+1} + \frac{\delta_{t+1}}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1})\\
&= \underbrace{k^* - \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{a}_{t+1}}_{\delta_{t+1}} - \frac{\delta_{t+1}}{\Delta}\underbrace{\mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}}_{\Delta-1}\\
&= \frac{\delta_{t+1}}{\Delta}
\end{aligned}$$

so daß

$$\mathbf{q}_{t+1,m}^{\text{res}} = \frac{\delta_{t+1}}{\Delta}\begin{bmatrix}-\tilde{\mathbf{K}}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\\ 1\end{bmatrix} \tag{4.21}$$

ist. Damit folgt für Δ_b zunächst

$$\begin{aligned}\Delta_b &\overset{(4.21)}{=} \frac{\delta_{t+1}}{\Delta}\begin{bmatrix}\tilde{\mathbf{K}}_{tm}\mathbf{a}_{t+1}\\ k^*\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}-\tilde{\mathbf{K}}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\\ 1\end{bmatrix} + \lambda\frac{\delta_{t+1}}{\Delta}\\ &= (-\mathbf{a}_{t+1}^{\mathsf{T}}\tilde{\mathbf{K}}_{tm}^{\mathsf{T}}\tilde{\mathbf{K}}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1} + k^*)\frac{\delta_{t+1}}{\Delta} + \lambda\frac{\delta_{t+1}}{\Delta}\end{aligned}$$

In einer Zwischenrechnung schieben wir $\lambda\mathbf{K}_{mm}$ in den linken Term ein:

$$\begin{aligned}\mathbf{a}_{t+1}^{\mathsf{T}}\tilde{\mathbf{K}}_{tm}^{\mathsf{T}}\tilde{\mathbf{K}}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1} &= \mathbf{a}_{t+1}^{\mathsf{T}}\underbrace{(\tilde{\mathbf{K}}_{tm}^{\mathsf{T}}\tilde{\mathbf{K}}_{tm} + \lambda\mathbf{K}_{mm})}_{\mathbf{P}_{tm}}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1} - \lambda\underbrace{\overbrace{\mathbf{a}_{t+1}^{\mathsf{T}}\mathbf{K}_{mm}}^{\mathbf{k}_{t+1}^{\mathsf{T}}}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}}_{\Delta-1}\\ &= \mathbf{a}_{t+1}^{\mathsf{T}}\mathbf{k}_{t+1} - \lambda\Delta + \lambda\end{aligned}$$

Eingesetzt in Δ_b ergibt sich somit

$$\begin{aligned}\Delta_b &= \frac{\delta_{t+1}}{\Delta}(\underbrace{k^* - \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{a}_{t+1}}_{\delta_{t+1}} + \lambda\Delta - \lambda) + \lambda\frac{\delta_{t+1}}{\Delta}\\ &= \frac{\delta_{t+1}^2}{\Delta} + \lambda\delta_{t+1}\end{aligned} \tag{4.22}$$

Zuletzt bestimmen wir noch κ aus (4.12). Hierfür folgt

$$\begin{aligned}\kappa &\overset{(4.12)}{=} \frac{\mathbf{y}_{t+1}^{\mathsf{T}}\mathbf{q}_{t+1,m}^{\text{res}}}{\Delta_b}\\ &\overset{(4.21)}{=} \frac{\delta_{t+1}}{\Delta\Delta_b}\begin{bmatrix}\mathbf{y}_t\\ y^*\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}-\tilde{\mathbf{K}}_{tm}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\\ 1\end{bmatrix}\\ &= \frac{\delta_{t+1}}{\Delta\Delta_b}\underbrace{(-\mathbf{k}_{t+1}^{\mathsf{T}}\overbrace{\mathbf{P}_{tm}^{-1}\tilde{\mathbf{K}}_{tm}^{\mathsf{T}}\mathbf{y}_t}^{\mathbf{w}_{tm}} + y^*)}_{\varrho}\\ &= \frac{\delta_{t+1}\varrho}{\Delta\Delta_b}\end{aligned} \tag{4.23}$$

Jetzt sind wir auch schon fertig, weil alle Aktualisierungen aus Lemma 2 von $\{\mathbf{P}_{t+1,m}^{-1}, \mathbf{w}_{t+1,m}, \xi_{t+1,m}\}$ auf $\{\mathbf{P}_{t+1,m+1}^{-1}, \mathbf{w}_{t+1,m+1}, \xi_{t+1,m+1}\}$ nur von den Hilfsgrößen $\mathbf{w}_b, \Delta_b$ und κ abhängen. Zusammenfassend gilt:

Lemma 4. *Unter Verwendung der Online-Approximation $\tilde{\mathbf{K}}_{tm}$ kann die Basiserweiterung aus Lemma 2 mit vom Zeitindex t unabhängigem Aufwand vorgenommen werden. Die zur Aktualisierung auf $\{\mathbf{P}_{t+1,m+1}^{-1}, \mathbf{w}_{t+1,m+1}, \xi_{t+1,m+1}\}$ relevanten Größen $\mathbf{w}_b, \Delta_b$ und κ werden berechnet durch:*

- $\mathbf{w}_b = \mathbf{a}_{t+1} + \frac{\delta_{t+1}}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}$
- $\Delta_b = \frac{\delta_{t+1}^2}{\Delta} + \lambda\delta_{t+1}$
- $\kappa = \frac{\delta_{t+1}}{\Delta} \cdot \frac{\varrho}{\Delta_b}$

Der Aufwand dieser Berechnung ist $\mathcal{O}(m)$, sofern wir die zuvor (Lemma 1) berechneten Größen $\{\mathbf{a}_{t+1}, \mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}, \Delta, \delta_{t+1}\}$ wiederverwenden.

Ein modifiziertes Kriterium zur Basisselektion

Das von Csató und Opper (2001) vorgeschlagene Kriterium zur Hinzunahme einer Basisfunktion (4.15) ist, wie beschrieben, ein unüberwachtes Kriterium, das nur den Fehler bei der Approximation durch die bereits selektierten Basiselemente berücksichtigt. Mit Hilfe von Lemma 4 können wir daraus aber auch ein effizientes überwachtes Kriterium konstruieren.

Betrachten wir die regularisierten Kosten $\xi_{t+1,m}$ der zur Regression gehörenden Optimierungsaufgabe. Nach Lemma 2 verringern sich diese durch Hinzunahme einer weiteren Basisfunktion mittels

$$\xi_{t+1,m+1} = \xi_{t+1,m} - \kappa^2 \Delta_b.$$

Somit ist die Quantität $\kappa^2 \Delta_b$ ein Indikator dafür, wie relevant eine Basisfunktion $k(\mathbf{x}_{t+1}, \cdot\,)$ zum Lösen des Problems ist und kann dazu eingesetzt werden, die Anzahl der selektierten Basisfunktionen schon während der Online-Selektion möglichst klein zu halten. (Am Ende des Lernens können wir sich als insgesamt irrelevant herausstellende Basisfunktionen leicht mit Lemma 3 entfernen).

In dieser Arbeit stellen wir daher ein neues, zweigeteiltes Selektionskriterium vor, das sowohl den Fehler bei der Approximation des Kerns als auch den Beitrag zum Lösen der Regressionsaufgabe mitberücksichtigt:

- Neuheit (unüberwachter Anteil):

$$\text{FALLS } \delta_{t+1} = k^* - \mathbf{k}_{t+1}^\mathsf{T} \mathbf{a}_{t+1} > \texttt{TOL1} \text{ DANN Basiserweiterung}$$

- Relevanz (überwachter Anteil):

$$\text{FALLS } \kappa^2 \Delta_b > \texttt{TOL2} \text{ DANN Basiserweiterung}$$

Unsere Experimente (siehe kommenden Abschnitt 4.3) haben gezeigt, daß sich mit einer Kombination beider Kriterien der Form

$$\text{FALLS } \delta_{t+1} > \texttt{TOL1} \text{ UND } \kappa^2 \Delta_b / t > \texttt{TOL2} \text{ DANN Basiserweiterung} \tag{4.24}$$

vernünftige Ergebnisse erzielen lassen. Die zusätzliche Division durch t führen wir deswegen ein (heuristisch), weil die Gesamtkosten ξ_{tm} die Summe der Fehlerquadrate mißt und von der Anzahl der Daten abhängt. Der zuätzliche Aufwand zur Berechnung des überwachten Kriteriums (4.24) gegenüber dem unüberwachten (4.15) ist vernachlässigbare $\mathcal{O}(1)$, weil alle involvierten Terme κ, Δ_b nach Lemma 4 schon bekannt sind.

4.2.3 Zusammenfassung: der Algorithmus

In Abbildung 4.4 fassen wir das soweit Gesagte zu einem kompletten Lernalgorithmus für den Gewichtsvektor $\mathbf{w}$ zusammen. Die vom Anwender vor dem Starten festzulegenden Parameter sind (durch Ausprobieren und/oder Parameterwahlstrategien):

- die Kernfunktion $k(\mathbf{x}, \mathbf{y})$, z.B. Gauß-RBF $k(\mathbf{x}, \mathbf{y}) = \exp\{-h^{-1}\|\mathbf{x} - \mathbf{y}\|^2\}$ mit Längenskala $h = d$ für d-dimensionale standardisierte Inputs
- der Regularisierungsparameter λ; typische Werte sind $\lambda = 10^{-1}$ oder $\lambda = 10^{-2}$
- die Schranke `TOL1` der unüberwachten Basisselektion; typische Werte sind $\texttt{TOL1} = 10^{-1}$ oder $\texttt{TOL1} = 10^{-2}$
- die Schranke `TOL2` der überwachten Basisselektion; typische Werte sind $\texttt{TOL2} = 10^{-2}$ oder $\texttt{TOL2} = 10^{-3}$ (oder $\texttt{TOL2} = 0$ um unüberwachte Selektion zu erhalten).

- eine maximale Anzahl `MAX_m` zu selektierender Basiselemente, um den worst-case Aufwand beim Online-Lernen zu beschränken[3], z.B. `MAX_m` = 5000 (bei zeitkritischen Echtzeitanwendungen können wir uns nur kleinere Werte leisten, etwa `MAX_m` = 1000 oder `MAX_m` = 1500).

Zur Durchführung müssen über alle Zeitschritte t hinweg nur die Größen $\mathbf{P}_{tm}^{-1}$ ($m \times m$ Matrix), $\mathbf{K}_{mm}^{-1}$ ($m \times m$ Matrix), $\mathbf{w}_{tm}$ ($m \times 1$ Vektor) und $\mathcal{D}_t$ ($m \times d$ Matrix) gespeichert und aktualisiert werden, wobei die aktuelle Anzahl selektierter Basiselemente m durch `MAX_m` beschränkt ist. Alle restlichen Größen sind temporäre Variablen und $m \times 1$ Vektoren, so daß der Speicherbedarf des Algorithmus insgesamt $\mathcal{O}(\texttt{MAX\_m}^2)$ ist. Der Rechenaufwand beträgt für jedes beobachtete Datenbeispiel $\mathcal{O}(m^2)$.

4.3 Experimente

Wir untersuchen und demonstrieren die Leistungsfähigkeit unseres Algorithmus anhand einiger konkreter Beispiele aus den Anwendungsgebieten des überwachten Lernens:

Einfache Funktionsapproximation

Hier untersuchen wir anhand der Datensätze 'Sinc', 'Boston' und 'Abalone' (aus Abschnitt 3.3, Seite 51), wie sich unsere Online-Variante von Regularisierungsnetzen gegenüber herkömmlichen Varianten verhält. Die Datensätze selbst sind statisch und bestehen aus einer relativ geringen Anzahl von Trainingsdaten, so daß wir problemlos aufwendige aber leistungsfähige Offline-Verfahren zum Einsatz bringen können, um so unseren Algorithmus fundiert zu evaluieren. Als Evaluationskriterium dient uns (wie zuvor) einmal der Vorhersagefehler in den unabhängigen Testdaten, gemessen als mittlerer quadratischer Fehler (MSE). Zweitens wollen wir aber auch die Effizienz der vorgeschlagenen überwachten Online-Basisselektion untersuchen, wozu wir zum Vergleich einerseits unüberwachte Online-Basisselektion und andererseits überwachte Offline-Basisselektion (mittels des Standardverfahren Optimized Matching Pursuit/Orthogonal Least Squares der adaptiven Signalapproximation) heranziehen. Insgesamt betrachten und vergleichen wir die folgenden Verfahren, die allesamt die Eigenschaft haben, die Basis adaptiv und automatisch aus den Trainingsdaten zu generieren:

1. (Online und überwacht) unseren Algorithmus aus Abbildung 4.4.
2. (Online und unüberwacht) Kernel Recursive Least Squares (KRLS) aus Engel et al. (2004); entspricht unserem Algorithmus ohne Regularisierung und mit unüberwachter Basisselektion.
3. (Offline und unüberwacht) RN mit SR-Approximation und ICD-Auswahl der Basis aus Kapitel 3. Als Abbruchkriterium verwenden wir dieselbe Schranke `TOL1` wie in 1. und 2., d.h. ICD bricht ab, wenn das Residuum aller verbleibenden, nicht-selektierten Basiskandidaten `TOL1` nicht mehr überschreitet.
4. (Offline und überwacht) OMP/OLS aus Rebollo-Neira und Lowe (2002); Andrle et al. (2004), wobei die Menge der Basiskandidaten durch die Kernfunktion, plaziert auf allen Trainingsdaten, erzeugt wird. Als Abbruchkriterium verwenden wir den GCV, d.h. es werden solange Basiselemente hinzugefügt, bis der GCV wieder ansteigt.[4]

Die Parameter des Lernverfahrens (Kern und Regularisierung) wählen wir für alle Verfahren identisch wie in Abschnitt 3.3 beschrieben; alle Verfahren können die Auswahl ihrer Basisfunktionen aus exakt

[3] Für eine effiziente Implementation muß der notwendige Arbeitsspeicher vor dem Starten alloziiert werden.

[4] Gewöhnlich würde überwachte gierige Basisselektion bei Offline-Lernen in zwei Phasen stattfinden. Die erste Phase, die Vorwärtsselektion, ist genau die beschriebene gierige Auswahlprozedur, bei der schrittweise diejenige Basisfunktion als nächstes hinzugefügt wird, die das verbleibende Residuum am meisten reduziert. Häufig sind dadurch aber am Ende zuviele und redundante Basisfunktionen selektiert worden. Daher werden in einer zweiten Phase, der Rückwärtselimination, schrittweise diejenigen Basisfunktionen wieder entfernt, die das Residuum am wenigsten ansteigen lassen. Auf diese zweite Phase verzichten wir hier in den Experimenten (obwohl das mit unserem Algorithmus in Abbildung 4.4 technisch ebenfalls möglich wäre).

Parameter:

// $k(\cdot,\cdot)$, h: Kernfunktion und Parameter h der Kernfunktion
// λ: Regularisierungsparameter
// TOL1: Schranke für unüberwachte Basisselektion
// TOL2: Schranke für überwachte Basisselektion
// MAX_m: maximale Anzahl zu selektierender Basiselemente

Initialisierung:

Beobachte erstes Beispiel $(\mathbf{x}_1, y^*)$. Initialisiere Basis $\mathcal{D}_1 = \{\mathbf{x}_1\}$ und setze $m = 1$ und $t = 1$.
Bestimme $\mathbf{P}_{11}^{-1}, \mathbf{w}_{11}, \mathbf{K}_{11}$.

Hauptschleife:

Beobachte $(\mathbf{x}_{t+1}, y^*)$ und berechne $\mathbf{k}_{t+1} = \mathbf{k}_m(\mathbf{x}_{t+1})$ sowie $k^* = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1})$. $\mathcal{O}(m)$

$\mathbf{a}_{t+1} = \mathbf{K}_{mm}^{-1}\mathbf{k}_{t+1}$ $\mathcal{O}(m^2)$

$\delta_{t+1} = k^* - \mathbf{k}_{t+1}^\mathsf{T}\mathbf{a}_{t+1}$ $\mathcal{O}(m)$

$\mathbf{z} := \mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}$ // Hilfsvariable $\mathcal{O}(m^2)$

$\Delta = 1 + \mathbf{k}_{t+1}^\mathsf{T}\mathbf{z}$ $\mathcal{O}(m)$

$\Delta_b = \frac{\delta_{t+1}^2}{\Delta} + \lambda\delta_{t+1}$ $\mathcal{O}(1)$

$\varrho = y^* - \mathbf{w}_{tm}^\mathsf{T}\mathbf{k}_{t+1}$ $\mathcal{O}(m)$

$\kappa = \frac{\delta_{t+1}}{\Delta} \cdot \frac{\varrho}{\Delta_b}$ $\mathcal{O}(1)$

1. Normaler Schritt

$\mathbf{P}_{t+1,m} = \mathbf{P}_{tm} - \frac{\mathbf{z}\mathbf{z}^\mathsf{T}}{\Delta}$ $\mathcal{O}(m^2)$

$\mathbf{w}_{t+1,m} = \mathbf{w}_{tm} + \frac{\varrho}{\Delta}\mathbf{z}$ $\mathcal{O}(m)$

2. Basiserweiterungstest

FALLS $\delta_{t+1} >$ TOL1 UND $\Delta_b\kappa^2/t >$ TOL2 DANN gehe zu 3.
SONST überspringe 3. und 4.

3. Basiserweiterungsschritt

$\mathbf{w}_b = \mathbf{a}_{t+1} + \frac{\delta_{t+1}}{\Delta}\mathbf{z}$ $\mathcal{O}(m)$

$\mathbf{P}_{t+1,m+1}^{-1} = \begin{bmatrix}\mathbf{P}_{t+1,m}^{-1} & \mathbf{0}\\ \mathbf{0}^\mathsf{T} & 0\end{bmatrix} + \frac{1}{\Delta_b}\begin{bmatrix}-\mathbf{w}_b\\ 1\end{bmatrix}\begin{bmatrix}-\mathbf{w}_b\\ 1\end{bmatrix}^\mathsf{T}$ $\mathcal{O}(m^2)$

$\mathbf{w}_{t+1,m+1} = \begin{bmatrix}\mathbf{w}_{t+1,m}\\ 0\end{bmatrix} + \kappa\begin{bmatrix}-\mathbf{w}_b\\ 1\end{bmatrix}$ $\mathcal{O}(m)$

$\mathbf{K}_{m+1,m+1}^{-1} = \begin{bmatrix}\mathbf{K}_{mm}^{-1} & \mathbf{0}\\ \mathbf{0}^\mathsf{T} & 0\end{bmatrix} + \frac{1}{\delta_{t+1}}\begin{bmatrix}-\mathbf{a}_{t+1}\\ 1\end{bmatrix}\begin{bmatrix}-\mathbf{a}_{t+1}\\ 1\end{bmatrix}^\mathsf{T}$ $\mathcal{O}(m^2)$

$\mathcal{D} = \mathcal{D} \cup \{\mathbf{x}_{t+1}\}$

$m = m + 1$

4. Löschen (Falls $m >$ MAX_m)

$i^* := \text{argmin}_{i=1\ldots m}\ \frac{\mathbf{w}_{tm}[i]}{\mathbf{P}_{tm}^{-1}[ii]}$ $\mathcal{O}(m)$

$\mathbf{P}_{tm}^{-1} = \mathbf{T}_{i^*\leftrightarrow m}\mathbf{P}_{tm}^{-1}\mathbf{T}_{i^*\leftrightarrow m}$ // Zeilen-und Spaltentausch $\mathcal{O}(m)$

$\mathbf{w}_{tm} = \mathbf{T}_{i^*\leftrightarrow m}\mathbf{w}_{tm}$ // Zeilentausch $\mathcal{O}(1)$

$\mathbf{u} := \mathbf{P}_{tm}^{-1}[1 : m-1, m]$ // (die m-1 der Zeilen der m.ten Spalte) $\mathcal{O}(m)$

$u^* := \mathbf{P}_{tm}^{-1}[m, m]$ // (das m.te Diagonalelement) $\mathcal{O}(1)$

$w^* := \mathbf{w}_{tm}[m]$ // (das m.te Element) $\mathcal{O}(1)$

$\mathbf{P}_{t,m\setminus i}^{-1} = \text{Trunc}(\mathbf{P}_{tm}^{-1}) - \frac{\mathbf{u}\mathbf{u}^\mathsf{T}}{u^*}$ $\mathcal{O}(m^2)$

$\mathbf{w}_{t,m\setminus i} = \text{Trunc}(\mathbf{w}_{tm}) + \frac{w^*}{u^*}\mathbf{u}$ $\mathcal{O}(m)$

// Mit $\mathbf{K}_{m\setminus i,m\setminus i}^{-1}$ verfahren wir analog $\mathcal{O}(m^2)$

$\mathcal{D} = \mathcal{D}\setminus\{\mathbf{x}_{i^*}\}$

$t = t + 1$

Abbildung 4.4: Algorithmus zum Online-Lernen mit Regularisierungsnetzen. Die Basisfunktionen werden dabei automatisch aus dem Datenstrom selektiert. Der Aufwand pro beobachteten Datenbeispiel beträgt $\mathcal{O}(m^2)$.

derselben Menge von Kandidaten treffen. Bei den beiden Online-Verfahren hängt das Ergebnis von der Reihenfolge ab, in der die Trainingsdaten präsentiert werden. Daher führen wir hier 100 unabhängige Lernläufe mit zufällig permutierter Anordnung der Trainingsdaten durch. Die einzelnen Resultate werden in Tabelle 4.1 zusammengefaßt. Die Ergebnisse zeigen allgemein, daß erstens die Online-Verfahren (beinahe) ebensogute Vorhersagen liefern können wie die Offline-Verfahren, und daß zweitens überwachte Basisselektion erwartungsgemäß in einer deutlich kompakteren Auswahl von Basisfunktionen resultiert als die unüberwachte. Speziell im Vergleich mit KRLS zeigt sich weiter, daß unser weiterentwickelter Ansatz ebenso leistungsfähig[5] ist, dieses Ergebnis aber mit einer deutlich geringeren Anzahl von Basisfunktionen realisiert, weil das zusätzliche überwachte Kriterium redundante und irrelevante Basisfunktionen von vorneherein außen vor läßt. Das wiederum ist natürlich von entscheidender Bedeutung bei sehr großen Datenmengen und/oder zeitkritischen Echtzeitanwendungen, weil bei allen diesen Varianten die Anzahl der Basisfunktionen quadratisch in den Rechen- und Speicheraufwand eingeht.

Tabelle 4.1: Vergleich verschiedener Lernverfahren mit automatischer Basisselektion auf den drei Datensätzen 'Sinc', 'Abalone' und 'Boston'. Die Einträge bestehen jeweils aus dem erzielten Vorhersagefehler (gemessen als MSE) und der Anzahl vom Verfahren selektierter Basisfunktionen. Bei unserem Verfahren aus Abb. 4.4 und bei KRLS hängt das Ergebnis von der Reihenfolge der präsentierten Daten ab. Daher wird hier der Mittelwert aus 100 unabhängigen Läufen mit zufällig permutierter Reihenfolge angezeigt.

	Sinc	**Abalone**	**Boston**
OLS+GCV	0.00056 (10)	0.35 (62)	0.88 (44)
KRLS ($\texttt{TOL1} = 0.1$)	0.00073±0.00006 (10.94)	0.37±0.0006 (46.3)	0.61±0.002 (101.6)
Alg. 4.4 ($\texttt{TOL1} = 0.1, \texttt{TOL2} = 10^{-3}$)	0.00212±0.00024 (7.92)	0.40±0.0013 (15.65)	0.66±0.007 (33.05)
SR-RN-ICD ($\texttt{TOL1} = 0.1$)	0.00055 (11)	0.37 (43)	0.66 (88)
KRLS ($\texttt{TOL1} = 0.01$)	0.00091±0.0001 (14.3)	0.36±0.01 (124.3)	0.65±0.039 (220.6)
Alg. 4.4 ($\texttt{TOL1} = 0.01, \texttt{TOL2} = 10^{-4}$)	0.00075±0.0003 (11.1)	0.37±0.0006 (31.5)	0.63±0.2 (59.2)
SR-RN-ICD ($\texttt{TOL1} = 0.01$)	0.00058 (13)	0.36 (105)	0.86 (196)

Online-Lernen bei sehr großen Datenmengen: der Datensatz 'Forest'

Als nächstes untersuchen wir, wie sich unser Verfahren beim Einsatz auf sehr umfangreiche Datensätze, einem Anwendungsgebiet von Online-Algorithmen, schlägt. Dazu betrachten wir den größten der im UCI Repository frei verfügbaren Benchmarks: den Datensatz 'Forest'. Dieser besteht aus rund 580.000 einzelnen Datensätzen, mit jeweils 54 reellwertigen Attributen (diskrete Attribute wurden binär kodiert).

Wir vergleichen unseren Ansatz erneut mit OLS sowie dem verwandten Ansatz Kernel Matching Pursuit wie in Popovici et al. (2005); beide wurden aber nicht mit dem vollen Datensatz, sondern lediglich mit einer zufälligen, stark reduzierten Auswahl von Trainingsdaten konfrontiert. Die genaueren Details zu diesen Experimenten sowie die Ergebnisse im einzelnen wollen wir hier nicht näher ausführen, sondern verweisen dazu auf Jung und Polani (2006b). Es zeigt sich aber auch bei diesem Beispiel, daß unser Online-Ansatz (bei einer identischen Anzahl von Basisfunktionen) zwar eine etwas geringere Genauigkeit bei den Vorhersagen erzielt als die beiden Offline-Verfahren, zum Erreichen dieses Resultats aber weitaus

[5] Bis auf 'Sinc', wo die wir mit der großen Schranke zu aggressiv Basisfunktionen eliminieren.

weniger Aufwand (Rechenaufwand und Speicheraufwand) nötig ist: unser Ansatz ist um (mindestens) eine ganze Größenordnung schneller.

AR-Zeitreihenvorhersage einer Glasschmelze

In einem dritten Beispiel demonstrieren wir anhand eines realen Problems, wie das Verfahren zur autoregressiven Zeitreihenvorhersage eingesetzt werden kann. Der Datensatz 'CWT-2004' von der Firma Schott enthält umskalierte Messdaten eines realen Glasschmelzeprozeß über einen Zeitraum von rund 5 Monaten (ein Meßwert pro 15 Minuten). Ziel ist es, aus den Daten ein Modell zu lernen, mit dem die zeitliche Entwicklung für eine Spanne von weiteren 2 Wochen möglichst exakt vorhergesagt werden kann[6].

Die Daten enthalten sowohl die zeitliche Entwicklung der vorherzusagenden Ausgabegröße ('Qualität' des produzierten Glases) als auch eine Reihe zusätzlicher Einflußgrößen (Steuerungsinformationen wie Gasmenge im Brenner, Umwelteinflüsse wie Außentemperatur). Bezogen auf Funktionsapproximation gestaltet sich das Problem vor allem wegen der Trägheit des zugrundeliegenden physikalischen Prozesses als recht schwierig: zwischen Änderungen an den Eingaben und Auswirkungen auf die Ausgabe können oft Stunden oder Tage liegen, so daß man für eine präzise Vorhersage eine große Zahl vergangener Daten hinzunehmen muß und so leicht ein Regressionsproblem in mehreren hundert Dimensionen erhält. Ein Standardansatz (Feedforward Neuronales Netz mit vorgeschalteter PCA zur Reduktion der Dimension der Eingaben) brachte hier nur wenig zufriedenstellende Ergebnisse.

Unser Kernbasierter Ansatz dagegen expandiert die Lösung in den Daten selbst und kann daher besser mit hochdimensionalen Inputs umgehen. Insbesondere konnten wir die Online-Fähigkeit unseres Algorithmus ausnutzen, um die Trainingsdaten durch Vorhersagen des gelernten Modells dynamisch zu erweitern und sukzessive zu ersetzen, und so am Ende eine deutlich präzisere Vorhersage erreichen. Eine detailliertere Beschreibung dessen wollen wir an dieser Stelle nicht geben, siehe Jung et al. (2005).

[6] Nahezu derselbe Datensatz wurde als 'Challenge' beim Modellierungswettbewerb des Symposiums EUNITE 2003 eingesetzt.

Kapitel 5

Reinforcement Lernen mit Regularisierungsnetzen

Worin der Algorithmus aus Kapitel 4 schließlich auf Reinforcement Lernen übertragen wird

Im dritten Teil der Arbeit kehren wir von der allgemeinen nichtparametrisierten Funktionsapproximation mit Regularisierungsnetzen (RN) wieder zum eigentlichen Thema Reinforcement Lernen (RL) zurück. Unser Ziel wird es jetzt sein, den in Kapitel 4 entwickelten Algorithmus zum effizienten Online-Lernen mit RN für das Problem RL einzusetzen, indem wir ihn als Funktionsapproximationsmodul in der APE-Komponente einer auf API basierenden Architektur einbetten.

Anders als im vorangegangenen Teil handelt es sich bei APE aber nicht mehr direkt um ein klassisches Regressionsproblem, bei dem die Trainingsdaten aus Ein-und Ausgabepaaren $\{\mathbf{x}_i, y_i\}_{i=1}^n$ bestehen. Vielmehr werden hier die gewünschten Outputs y_i nur noch indirekt, in Form von Belohnungen (Rewards) oder 1-Schritt-Auszahlungen beobachtet. Um APE daher wieder auf klassische Regression zurückzuführen, verwenden wir die in Kapitel 1 beschriebenen Least-Squares-basierten APE Varianten BRM, LSTD sowie das weniger bekannte und noch zu erläuternde LSPE (Nedić und Bertsekas, 2003), welche wir mit Hilfe von RN zusammen mit der SR-Approximation (aus Kapitel 3) formulieren werden. Auf diese Weise umgehen wir den 'Curse of dimensionality', der für gewöhnlich den a priori parametrisierten (gitterbasierten) Methoden anhaftet, da die Basisfunktionen erst zur Laufzeit aus den tatsächlichen Trainingsdaten generiert werden. Gleichzeitig können wir uns aber auch bei der APE die hohen Dateneffizienz zunutze machen, die mit der geschlossenen Lösung des zugrundeliegenden Least-Squares einhergeht.

Für alle drei Formulierungen BRM, LSTD und LSPE werden wir nun, analog zu Kapitel 4, einen effizienten, rekursiven Lösungsalgorithmus formulieren. Eingebettet in API (und dank LSPE auch mit der Option auf OPI versehen), erhalten wir so ein leistungsfähiges RL-Verfahren, das insbesondere für Lernprobleme mit kontinuierlichen und hochdimensionalen Zustandsräumen, mit stochastischen Übergängen und lange hinausgezögerten Belohnungen eine gute (und hochperformante) Politik lernt. Dabei sind wir nicht auf ein Modell der Umwelt angewiesen, arbeiten dank RN (weitestgehend) unabhängig von der Dimension des Zustandsraums, konvergieren dank Least-Squares-basierter APE wesentlich schneller und mit deutlich weniger Agent-Umwelt Interaktionen, und können dank der effizienten Online-Approximation auch im Kontext zeitkritischer Echtzeitanwendungen operieren.

Zum Abschluß demonstrieren wir die Leistungsfähigkeit unseres Verfahrens anhand einer Auswahl typischer Benchmarkprobleme.

5.1 Herleitung und Algorithmus: RN + RL

Um die optimale Entscheidungsstrategie (Politik) in einem Markov-Entscheidungsproblem (MDP) näherungsweise durch Simulation und Funktionsapproximation zu bestimmen, betrachten wir die allgemeine

Architektur modellfreier approximativer Politik-Iteration:

- Wähle Anfangspolitik π_0 (zufällig oder problemspezifisch)
- Iteriere für $k = 0, 1, 2 \ldots$
  - Berechne (approximativ) die Wertefunktion $\tilde{Q}^{\pi_k}$ für π_k (Approximative Politik-Evaluation oder APE)
  - Bestimme eine verbesserte neue Politik π_{k+1} aus $\tilde{Q}^{\pi_k}$ (Approximative Politik-Verbesserung)

In Kapitel 1 hatten wir dies ausführlicher diskutiert, und argumentiert, daß die Hauptarbeit dabei in dem Schritt APE verrichtet wird und die Schwierigkeit in erster Linie darin besteht, den 'richtigen' Ansatzraum und die 'richtige' Parametrisierung für die Wertefunktion zu finden (die über dem hochdimensionalen Zustands-Aktionsraum definiert ist). Hierfür wollen wir nun die RN mit der SR-Approximation aus Kapitel 3 zusammen mit dem in Kapitel 4 entwickelten Online-Algorithmus einsetzen.

5.1.1 Least-Squares APE mit RN

Gegeben eine zu evaluierende Politik π und eine Folge von (unter π) beobachteten Zustandsübergängen $s_0, s_1, \ldots, s_t$ mit beobachteten Belohnungen $r_1, r_2, \ldots, r_t$. Im modellfreien Lernen betrachten wir als Wertefunktion die über Zustands-Aktionspaaren definierte Q-Funktion; zur Abkürzung schreiben wir daher für die Inputs $\mathbf{x}_i := (s_i, a_i)$, wobei hier Aktionen a_i gemäß π ausgewählt werden, d.h. $a_i = \pi(s_i)$.

Jetzt wollen wir (wie in Abschnitt 2.4.1 und 2.4.2 für RR gezeigt) die Least-Squares basierten APE Varianten BRM, LSTD und LSPE 'kernelisieren'. Wir starten zunächst wieder mit der primalen Form der zu lernenden Q-Funktion:

$$\tilde{Q}(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{d_{\mathcal{H}}} w_i \phi_i(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T} \mathbf{w} \tag{5.1}$$

wobei $\boldsymbol{\phi}(\mathbf{x}) = \big(\phi_1(\mathbf{x}), \ldots, \phi_{d_{\mathcal{H}}}(\mathbf{x})\big)^\mathsf{T}$ der Merkmalsvektor und $\mathbf{w}$ der gesuchte (primale) Gewichtsvektor ist.

BRM (primal)

Nach (1.27) erhalten wir im Fall BRM den gesuchten Gewichtsvektor in (5.1) als Lösung $\hat{\mathbf{w}}$ der (hier nun regularisierten) Least-Squares Aufgabe

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left\{ \sum_{i=0}^{t-1} \big[\boldsymbol{\phi}(\mathbf{x}_i)^\mathsf{T} \mathbf{w} - \gamma \boldsymbol{\phi}(\mathbf{x}_{i+1})^\mathsf{T} \mathbf{w} - r_{i+1} \big]^2 + \sigma \left\| \mathbf{w} \right\|^2 \right\}$$

wobei σ der Regularisierungsparameter[1] und γ der Diskontfaktor ist. Zur kürzeren Schreibweise drücken wir das äquivalent in Matrizenform aus:

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left\{ \left\| \mathbf{D}\boldsymbol{\Phi}\mathbf{w} - \mathbf{r} \right\|^2 + \sigma \left\| \mathbf{w} \right\|^2 \right\} \tag{5.2}$$

wobei wir $\mathbf{D}$ (als $t \times (t+1)$ Matrix), $\boldsymbol{\Phi}$ (als $(t+1) \times d_{\mathcal{H}}$ Matrix) und $\mathbf{r}$ (als $t \times 1$ Vektor) folgendermaßen definiert haben:

$$\mathbf{D} = \begin{bmatrix} 1 & -\gamma & & \\ & \ddots & \ddots & \\ & & 1 & -\gamma \end{bmatrix}, \quad \boldsymbol{\Phi} = \begin{bmatrix} \text{—} \; \boldsymbol{\phi}(\mathbf{x}_0)^\mathsf{T} \; \text{—} \\ \vdots \\ \text{—} \; \boldsymbol{\phi}(\mathbf{x}_t)^\mathsf{T} \; \text{—} \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} r_1 \\ \vdots \\ r_t \end{bmatrix} \tag{5.3}$$

Die Lösung von (5.2) ist nun

$$\hat{\mathbf{w}} = \Big[(\mathbf{D}\boldsymbol{\Phi})^\mathsf{T} (\mathbf{D}\boldsymbol{\Phi}) + \sigma \mathbf{I} \Big]^{-1} (\mathbf{D}\boldsymbol{\Phi})^\mathsf{T} \mathbf{r}$$

[1] Zu beachten ist, daß der Regularisierungsparameter hier mit σ und nicht mehr mit λ bezeichnet wird, um Verwechselung mit dem λ aus TD(λ) auszuschließen.

BRM (dual)

Zum Dualisieren verfahren wir nun wie in Abschnitt 2.4.2. Mit dem Matrixinversionslemma (B.3)[2] folgt

$$\hat{\mathbf{w}} = (\mathbf{D}\boldsymbol{\Phi})^\mathsf{T}\Big[\mathbf{D}\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}\mathbf{D}^\mathsf{T} + \sigma\mathbf{I}\Big]^{-1}\mathbf{r},$$

so daß der gesuchte Gewichtsvektor $\hat{\mathbf{w}}$ auch durch duale Variablen $\hat{\boldsymbol{\alpha}}$ ausgedrückt werden kann:

$$\hat{\mathbf{w}} = (\mathbf{D}\boldsymbol{\Phi})^\mathsf{T}\hat{\boldsymbol{\alpha}} \text{ mit } \hat{\boldsymbol{\alpha}} = \Big[\mathbf{D}\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}\mathbf{D}^\mathsf{T} + \sigma\mathbf{I}\Big]^{-1}\mathbf{r} \tag{5.4}$$

Die auftretenden Skalarprodukte zwischen den Merkmalsvektoren werden wieder durch die Kernfunktion $k(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\phi}(\mathbf{y})$ ersetzt, so daß $\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} = \mathbf{K}$ mit $(t+1)\times(t+1)$ Kernmatrix $[\mathbf{K}]_{ij} = k(\mathbf{x}_i,\mathbf{x}_j)$ gilt.

BRM (SR-Approximation)

Jetzt reduzieren wir das $t \times t$ Problem in (5.4) mit Hilfe der SR-Approximation (wie in Abschnitt 3.2) wieder auf ein $m \times m$ Problem. Sei $\{\tilde{\mathbf{x}}_i\}_{i=1}^m$ eine m-elementige Teilmenge ausgewählter Trainingsdaten. Dann ersetzen wir den Kern (wie in (3.7)) durch die Approximation

$$k(\mathbf{x},\mathbf{y}) \approx \mathbf{k}_m(\mathbf{x})^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{y}) \tag{5.5}$$

und damit Kernmatrix $\mathbf{K}$ durch die Approximation $\mathbf{K} \approx \mathbf{K}_{t+1,m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}$, wobei

$$\mathbf{k}_m(\cdot) := \begin{bmatrix} k(\tilde{\mathbf{x}}_1,\cdot) \\ \vdots \\ k(\tilde{\mathbf{x}}_m,\cdot) \end{bmatrix}, \mathbf{K}_{t+1,m} := \begin{bmatrix} \text{—}\ \mathbf{k}_m(\mathbf{x}_0)^\mathsf{T}\ \text{—} \\ \vdots \\ \text{—}\ \mathbf{k}_m(\mathbf{x}_t)^\mathsf{T}\ \text{—} \end{bmatrix}, \mathbf{K}_{mm} := \begin{bmatrix} k(\tilde{\mathbf{x}}_1,\tilde{\mathbf{x}}_1) & \cdots & k(\tilde{\mathbf{x}}_1,\tilde{\mathbf{x}}_m) \\ \vdots & & \\ k(\tilde{\mathbf{x}}_m,\tilde{\mathbf{x}}_1) & \cdots & k(\tilde{\mathbf{x}}_m,\tilde{\mathbf{x}}_m) \end{bmatrix} \tag{5.6}$$

ist. Für den $(t+1)\times 1$ Vektor $\mathbf{k}(\mathbf{x}) := \big(k(\mathbf{x}_0,\mathbf{x}),\ldots,k(\mathbf{x}_t,\mathbf{x})\big)^\mathsf{T}$ erhalten wir so die Approximation $\mathbf{k}(\mathbf{x}) \approx \mathbf{K}_{t+1,m}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x})$.

Setzen wir die SR-Approximation (5.5) nun in Vorhersagen (5.1) in beliebigen Argumenten $\mathbf{x}$ ein:

$$\begin{aligned}
\tilde{Q}(\mathbf{x}) &\overset{(5.1)}{=} \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\hat{\mathbf{w}} \\
&\overset{(5.4)}{=} \underbrace{\boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\Phi}^\mathsf{T}}_{\mathbf{k}(\mathbf{x})}\mathbf{D}^\mathsf{T}\Big[\mathbf{D}\underbrace{\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}}_{\mathbf{K}}\mathbf{D}^\mathsf{T} + \sigma\mathbf{I}\Big]^{-1}\mathbf{r} \\
&\overset{(5.5)}{\approx} \mathbf{k}_m(\mathbf{x})^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{D}^\mathsf{T}\Big[\mathbf{D}\mathbf{K}_{t+1,m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{D}^\mathsf{T} + \sigma\mathbf{I}\Big]^{-1}\mathbf{r} \\
&\overset{(B.3)}{=} \mathbf{k}_m(\mathbf{x})^\mathsf{T}\Big[(\mathbf{D}\mathbf{K}_{t+1,m})^\mathsf{T}(\mathbf{D}\mathbf{K}_{t+1,m}) + \sigma\mathbf{K}_{mm}\Big]^{-1}(\mathbf{D}\mathbf{K}_{t+1,m})^\mathsf{T}\mathbf{r}
\end{aligned}$$

so erhalten wir die durch SR reduzierten, dualen Koeffizienten $\hat{\boldsymbol{\alpha}}$ durch

$$\hat{\boldsymbol{\alpha}} = \Big[(\mathbf{D}\mathbf{K}_{t+1,m})^\mathsf{T}(\mathbf{D}\mathbf{K}_{t+1,m}) + \sigma\mathbf{K}_{mm}\Big]^{-1}(\mathbf{D}\mathbf{K}_{t+1,m})^\mathsf{T}\mathbf{r}$$

bzw. als Lösung des regularisierten Least-Squares Problems

$$\hat{\boldsymbol{\alpha}} = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^m}\Big\{\|\mathbf{H}_{tm}\boldsymbol{\alpha} - \mathbf{r}\|^2 + \sigma\boldsymbol{\alpha}^\mathsf{T}\mathbf{K}_{mm}\boldsymbol{\alpha}\Big\} \tag{5.7}$$

$$= (\mathbf{H}_{tm}^\mathsf{T}\mathbf{H}_{tm} + \sigma\mathbf{K}_{mm})^{-1}\mathbf{H}_{tm}^\mathsf{T}\mathbf{r} \tag{5.8}$$

wobei zur Abkürzung $t \times m$ Matrix $\mathbf{H}_{tm}$ folgendermaßen definiert wurde:

$$\mathbf{H}_{tm} := \mathbf{D}\mathbf{K}_{t+1,m} = \begin{bmatrix} \text{—}\ \mathbf{k}_m(\mathbf{x}_0)^\mathsf{T} - \gamma\mathbf{k}_m(\mathbf{x}_1)^\mathsf{T}\ \text{—} \\ \vdots \\ \text{—}\ \mathbf{k}_m(\mathbf{x}_{t-1})^\mathsf{T} - \gamma\mathbf{k}_m(\mathbf{x}_t)^\mathsf{T}\ \text{—} \end{bmatrix}. \tag{5.9}$$

[2] Es gilt: $(\mathbf{A}+\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1} = \mathbf{A}^{-1}\mathbf{B}(\mathbf{D}+\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}$. Mit $\mathbf{A} := \sigma\mathbf{I}$, $\mathbf{B} := (\mathbf{D}\boldsymbol{\Phi})^\mathsf{T}$, $\mathbf{D} := \mathbf{I}_t$ und $C := (\mathbf{D}\boldsymbol{\Phi})$ folgt Behauptung.

LSTD-0 (primal)

Ebenso verfahren wir mit LSTD, zunächst für den Fall $\lambda = 0$: nach (1.24) erhalten wir den Gewichtsvektor in (5.1) als Lösung der folgenden Fixpunktgleichung in $\hat{\mathbf{w}}$ (wobei wir das der Projektion zugrundeliegende Least-Squares Problem regularisiert[3] haben):

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left\{ \sum_{i=0}^{t-1} \left[\boldsymbol{\phi}(\mathbf{x}_i)^\mathsf{T}\mathbf{w} - \gamma \boldsymbol{\phi}(\mathbf{x}_{i+1})^\mathsf{T}\hat{\mathbf{w}} - r_{i+1} \right]^2 + \sigma \left\|\mathbf{w}\right\|^2 \right\} \tag{5.10}$$

Ausgedrückt in Matrizen heißt das

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left\{ \left\|\mathbf{G}\boldsymbol{\Phi}\mathbf{w} - (\mathbf{G}-\mathbf{D})\boldsymbol{\Phi}\hat{\mathbf{w}} - \mathbf{r}\right\|^2 + \sigma \left\|\mathbf{w}\right\|^2 \right\} \tag{5.11}$$

wobei $\mathbf{D}, \boldsymbol{\Phi}, \mathbf{r}$ wie in (5.3) und $\mathbf{G} := \begin{bmatrix}\mathbf{I}_t & \mathbf{0}\end{bmatrix}$, d.h.

$$\mathbf{G} := \begin{bmatrix} 1 & & & 0 \\ & \ddots & & \vdots \\ & & 1 & 0 \end{bmatrix} \tag{5.12}$$

definiert werden. Damit erhalten wir für (5.11)

$$\hat{\mathbf{w}} = \Big[(\mathbf{G}\boldsymbol{\Phi})^\mathsf{T}(\mathbf{G}\boldsymbol{\Phi}) + \sigma\mathbf{I} \Big]^{-1} (\mathbf{G}\boldsymbol{\Phi})^\mathsf{T} \Big(\mathbf{r} + (\mathbf{G}-\mathbf{D})\boldsymbol{\Phi}\hat{\mathbf{w}} \Big)$$

und so

$$\begin{aligned} 0 &= (\mathbf{G}\boldsymbol{\Phi})^\mathsf{T} \Big(\mathbf{r} + (\mathbf{G}-\mathbf{D})\boldsymbol{\Phi}\hat{\mathbf{w}} \Big) - \Big[(\mathbf{G}\boldsymbol{\Phi})^\mathsf{T}(\mathbf{G}\boldsymbol{\Phi}) + \sigma\mathbf{I} \Big] \hat{\mathbf{w}} \\ &= (\mathbf{G}\boldsymbol{\Phi})^\mathsf{T} \big(\mathbf{r} + \mathbf{G}\boldsymbol{\Phi}\hat{\mathbf{w}} - \mathbf{D}\boldsymbol{\Phi}\hat{\mathbf{w}} - \mathbf{G}\boldsymbol{\Phi}\hat{\mathbf{w}} \big) - \sigma\hat{\mathbf{w}} \\ &= (\mathbf{G}\boldsymbol{\Phi})^\mathsf{T}\mathbf{r} - \Big[(\mathbf{G}\boldsymbol{\Phi})^\mathsf{T}\mathbf{D}\boldsymbol{\Phi} + \sigma\mathbf{I} \Big] \hat{\mathbf{w}} \end{aligned}$$

und daher

$$\hat{\mathbf{w}} = \Big[(\mathbf{G}\boldsymbol{\Phi})^\mathsf{T}\mathbf{D}\boldsymbol{\Phi} + \sigma\mathbf{I} \Big]^{-1} (\mathbf{G}\boldsymbol{\Phi})^\mathsf{T}\mathbf{r}. \tag{5.13}$$

LSTD-0 (dual)

Jetzt wenden wir wieder die Matrixinversionsformel (B.3) auf (5.13) an und erhalten

$$\hat{\mathbf{w}} = (\mathbf{G}\boldsymbol{\Phi})^\mathsf{T} \Big[\mathbf{D}\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}\mathbf{G}^\mathsf{T} + \sigma\mathbf{I} \Big]^{-1} \mathbf{r}.$$

so daß der gesuchte Gewichtsvektor $\hat{\mathbf{w}}$ durch duale Variablen ausgedrückt werden kann:

$$\hat{\mathbf{w}} = (\mathbf{G}\boldsymbol{\Phi})^\mathsf{T}\hat{\boldsymbol{\alpha}} \text{ mit } \hat{\boldsymbol{\alpha}} = \Big[\mathbf{D}\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}\mathbf{G}^\mathsf{T} + \sigma\mathbf{I} \Big]^{-1} \mathbf{r}. \tag{5.14}$$

[3] Die Konvergenzeigenschaften von LSTD (für wachsende Anzahl beobachteter Zustandsübergänge t) werden davon nicht berührt, weil für eine feste Parametrisierung in m Basisfunktionen (und genau darauf läuft die SR-Approximation hinaus) die Stichprobenmatrix $\mathbf{A}_t$ in (1.22) mit Regularisierung genauso gegen $\mathbf{A}$ konvergiert, wie ohne: für

$$\mathbf{A}_t/(t+1) \overset{t\to\infty}{\longrightarrow} \mathbf{A} \quad \text{f.s.}$$

ist auch

$$(\mathbf{A}_t + \sigma\mathbf{K}_{mm})/(t+1) \overset{t\to\infty}{\longrightarrow} \mathbf{A} \quad \text{f.s.}$$

erfüllt.

LSTD-0 (SR-Approximation)

Wir setzen die SR-Approximation (5.5) in Vorhersagen (5.1) in einem beliebigem Punkt $\mathbf{x}$ ein

$$\begin{aligned}
\tilde{Q}(\mathbf{x}) &\overset{(5.1)}{=} \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\hat{\mathbf{w}} \\
&\overset{(5.14)}{=} \underbrace{\boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\Phi}^\mathsf{T}}_{\mathbf{k}(\mathbf{x})}\mathbf{G}^\mathsf{T}\Big[\mathbf{D}\underbrace{\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}}_{\mathbf{K}}\mathbf{G}^\mathsf{T}+\sigma\mathbf{I}\Big]^{-1}\mathbf{r} \\
&\overset{(5.5)}{\approx} \mathbf{k}_m(\mathbf{x})^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{G}^\mathsf{T}\Big[\mathbf{D}\mathbf{K}_{t+1,m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{G}^\mathsf{T}+\sigma\mathbf{I}\Big]^{-1}\mathbf{r} \\
&\overset{(B.3)}{=} \mathbf{k}_m(\mathbf{x})^\mathsf{T}\Big[(\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}(\mathbf{D}\mathbf{K}_{t+1,m})+\sigma\mathbf{K}_{mm}\Big]^{-1}(\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}\mathbf{r}
\end{aligned}$$

so daß wir die durch SR reduzierten, dualen Koeffizienten $\hat{\boldsymbol{\alpha}}$ durch

$$\hat{\boldsymbol{\alpha}} = \Big[(\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}(\mathbf{D}\mathbf{K}_{t+1,m})+\sigma\mathbf{K}_{mm}\Big]^{-1}(\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}\mathbf{r}$$

erhalten. Mit der Abkürzung $\mathbf{H}_{tm} := \mathbf{D}\mathbf{K}_{t+1,m}$ aus (5.9) und

$$\mathbf{K}_{tm} = \mathbf{G}\mathbf{K}_{t+1,m} = \begin{bmatrix} — & \mathbf{k}_m(\mathbf{x}_0)^\mathsf{T} & — \\ & \vdots & \\ — & \mathbf{k}_m(\mathbf{x}_{t-1})^\mathsf{T} & — \end{bmatrix} \tag{5.15}$$

folgt

$$\hat{\boldsymbol{\alpha}} = \big[\mathbf{K}_{tm}^\mathsf{T}\mathbf{H}_{tm}+\sigma\mathbf{K}_{mm}\big]^{-1}\mathbf{K}_{tm}^\mathsf{T}\mathbf{r} \tag{5.16}$$

Anders als im Fall BRM in (5.7) ist $\hat{\boldsymbol{\alpha}}$ aber *keine* direkte Lösung eines Least-Squares Problems.

LSTD-λ

Betrachten wir LSTD allgemeiner für $\lambda \in [0,1]$, dann erhalten wir statt (5.10) nach (1.26) den primalen Gewichtsvektor als Lösung der folgenden Fixpunktgleichung in $\hat{\mathbf{w}}$:

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left\{ \sum_{i=0}^{t-1}\Big[\boldsymbol{\phi}(\mathbf{x}_i)^\mathsf{T}\mathbf{w} - \boldsymbol{\phi}(\mathbf{x}_i)^\mathsf{T}\hat{\mathbf{w}} - \sum_{k=i}^{t-1}(\lambda\gamma)^{k-i}d(\mathbf{x}_k,\mathbf{x}_{k+1};\hat{\mathbf{w}})\Big]^2 + \sigma\left\|\mathbf{w}\right\|^2 \right\}$$

mit

$$d(\mathbf{x}_k,\mathbf{x}_{k+1};\hat{\mathbf{w}}) = r_{k+1} + \gamma\boldsymbol{\phi}(\mathbf{x}_{k+1})^\mathsf{T}\hat{\mathbf{w}} - \boldsymbol{\phi}(\mathbf{x}_k)^\mathsf{T}\hat{\mathbf{w}}.$$

Ausgedrückt in Matrizen heißt das

$$\begin{aligned}
\hat{\mathbf{w}} &= \operatorname*{argmin}_{\mathbf{w}} \Big\{ \left\|\mathbf{G}\boldsymbol{\Phi}\mathbf{w} - \mathbf{G}\boldsymbol{\Phi}\hat{\mathbf{w}} - \boldsymbol{\Lambda}(\mathbf{r} - \mathbf{D}\boldsymbol{\Phi}\hat{\mathbf{w}})\right\|^2 + \sigma\left\|\mathbf{w}\right\|^2 \Big\} \\
&= \operatorname*{argmin}_{\mathbf{w}} \Big\{ \left\|\mathbf{G}\boldsymbol{\Phi}\mathbf{w} - (\mathbf{G} - \boldsymbol{\Lambda}\mathbf{D})\boldsymbol{\Phi}\hat{\mathbf{w}} - \boldsymbol{\Lambda}\mathbf{r}\right\|^2 + \sigma\left\|\mathbf{w}\right\|^2 \Big\}
\end{aligned} \tag{5.17}$$

wobei $\mathbf{G}, \mathbf{D}, \boldsymbol{\Phi}, \mathbf{r}$ wie zuvor in (5.12),(5.3) und $(t+1)\times(t+1)$ Matrix $\boldsymbol{\Lambda}$ durch

$$\boldsymbol{\Lambda} := \begin{bmatrix} 1 & (\lambda\gamma)^1 & \cdots & (\lambda\gamma)^t \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & 1 & (\lambda\gamma)^1 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \tag{5.18}$$

definiert wird. Vergleichen wir (5.17) mit (5.11), dann stellen wir fest, daß für den allgemeineren Fall $\lambda > 0$ in (5.11) lediglich $\mathbf{D}$ durch $\boldsymbol{\Lambda}\mathbf{D}$ und $\mathbf{r}$ durch $\boldsymbol{\Lambda}\mathbf{r}$ ersetzt werden muß. Mit der gleichen Argumentation wie für LSTD-0 erhalten wir für LSTD-λ statt (5.16) den Ausdruck

$$\hat{\boldsymbol{\alpha}} = \big[\mathbf{K}_{tm}^\mathsf{T}\boldsymbol{\Lambda}\mathbf{H}_{tm}+\sigma\mathbf{K}_{mm}\big]^{-1}\mathbf{K}_{tm}^\mathsf{T}\boldsymbol{\Lambda}\mathbf{r}.$$

Zur weiteren Abkürzung führen wir noch $\mathbf{Z}_{tm} := \mathbf{\Lambda}^\mathsf{T}\mathbf{K}_{tm}$ ein, so daß der Gewichtsvektor bei LSTD-λ insgesamt durch

$$\hat{\boldsymbol{\alpha}} = \left[\mathbf{Z}_{tm}^\mathsf{T}\mathbf{H}_{tm} + \sigma\mathbf{K}_{mm}\right]^{-1}\mathbf{Z}_{tm}^\mathsf{T}\mathbf{r} \tag{5.19}$$

berechnet wird.

5.1.2 Least-Squares OPI mit RN

Die beiden APE Varianten BRM aus (5.8) und LSTD aus (5.19) sind prinzipiell Offline-Verfahren, die aus einer festen Menge von t beobachteten Zustandsübergängen (unter einer festen Politik π) den Gewichtsvektor für die Approximation der Wertefunktion als geschlossene Lösung eines linearen Gleichungssystems bestimmen. Eine Online-Implementation, bei der in jedem Zeitschritt ein neubeobachteter Zustandsübergang hinzugenommen wird, ist zunächst nicht zwingend erforderlich; genausogut könnten (5.8) und (5.19) offline gelöst werden, etwa mittels SR-Approximation und ICD-Auswahl der Basiselemente (wie in Kapitel 3).

Ein drittes Verfahren: LSPE(λ)

Allgemein können BRM und LSTD auch nur im Rahmen von API eingesetzt werden, bei der die zu evaluierende Politik für einen hinreichend langen Zeitraum festgehalten wird (oder mit einem exponentiell abfallenden 'Vergessens-Faktor' wie in Xu et al. (2002) gearbeitet wird). An dieser Stelle wollen wir nun mit der Least-Squares Policy Evaluation (LSPE) eine dritte, weniger bekannte Variante von APE betrachten, die das beste aus zwei Welten vereint: auf der einen Seite basiert sie, ebenso wie BRM und LSTD, auf der geschlossenen Lösung eines (direkten) Least-Squares Problem, das alle beobachteten Zustandsübergänge berücksichtigt und daher eine schnellere Konvergenz erzielt. Auf der anderen Seite ist sie aber auch ein iteratives Verfahren und produziert, genau wie TD(λ), eine Folge von Gewichtsvektoren $\mathbf{w}_1, \mathbf{w}_2, \ldots$, die (mit wachsender Anzahl von beobachteten Übergängen) gegen $\mathbf{w}^*$ konvergieren. Damit kann LSTD(λ), genau wie TD(λ), zur optimistischen Politik-Iteration (OPI) eingesetzt werden, bei der Evaluation und Verbesserung einer Politik in viel kürzeren Abständen erfolgen. Das Verfahren LSPE(λ) wird in Nedić und Bertsekas (2003) vorgeschlagen und basiert auf der λ-Politik-Iteration aus Bertsekas und Ioffe (1996).

In der primalen Darstellung von $\tilde{Q}(\mathbf{x}; \mathbf{w})$ in (5.1) läuft das Verfahren folgendermaßen ab:

LSPE-0 (primal)

LSPE ist ein iteratives Online-Verfahren, das nach jedem beobachteten Zustandsübergang einen neuen, aktualisierten Gewichtsvektor produziert. Sei $s_0, s_1, \ldots, s_{t-1}$ die Folge bisher beobachteter Zustandsübergänge, $r_1, \ldots, r_{t-1}$ die zugehörigen Belohnungen und $\mathbf{w}_{t-1}$ die bisherige Lösung. Wie zuvor bezeichne $\mathbf{x}_i := (s_i, a_i)$ mit $a_i = \pi(s_i)$ die Inputs des Funktionsapproximators. Im aktuellen Zeitschritt t beobachten wir einen neuen Zustand s_t unter Belohnung r_t. Der Gewichtsvektor $\mathbf{w}_{t-1}$ der Approximation aus dem vergangenen Zeitschritt wird nun wie folgt aktualisiert: zunächst lösen wir das (hier regularisierte[4]) Least-Squares Problem

$$\hat{\mathbf{w}}_t = \operatorname*{argmin}_{\mathbf{w}} \left\{ \sum_{i=0}^{t-1} \left[\boldsymbol{\phi}(\mathbf{x}_i)^\mathsf{T}\mathbf{w} - \gamma\boldsymbol{\phi}(\mathbf{x}_{i+1})^\mathsf{T}\mathbf{w}_{t-1} - r_{i+1}\right]^2 + \sigma \left\|\mathbf{w} - \mathbf{w}_{t-1}\right\|^2 \right\} \tag{5.20}$$

woraus wir $\hat{\mathbf{w}}_t$ erhalten. Der neue Gewichtsvektor $\mathbf{w}_t$ wird dann berechnet durch die Akualisierungsvorschrift

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \eta_t(\hat{\mathbf{w}}_t - \mathbf{w}_{t-1}) \tag{5.21}$$

wobei Startwert $\mathbf{w}_0$ ein beliebiger initialer Gewichtsvektor (z.B. $\mathbf{w}_0 = \mathbf{0}$) und $0 < \eta_t \leq 1$ eine (abfallende) Lernrate[5]. Die Folge der Iterierten $\mathbf{w}_1, \mathbf{w}_2, \ldots$ konvergiert (mit wachsender Anzahl beobachteter

[4] Wie im Fall LSTD so wird auch hier die Konvergenz nicht durch Regularisierung beeinträchtigt.

[5] Bei einer *festen* Politik ist abfallend nicht notwendig, Konvergenz ist auch für $\eta_t \equiv 1$ sichergestellt. Siehe Satz 3.1 in Bertsekas et al. (2003).

Zustandsübergänge) gegen $\mathbf{w}^*$; die gleiche Lösung, die im Grenzfall auch durch LSTD und TD erhalten wird (Bertsekas et al., 2003).

Jetzt drücken wir (5.20) wieder in Matrizenschreibweise aus:

$$\hat{\mathbf{w}}_t = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \|\mathbf{G}\mathbf{\Phi}\mathbf{w} - (\mathbf{G}-\mathbf{D})\mathbf{\Phi}\mathbf{w}_{t-1} - \mathbf{r}\|^2 + \sigma \|\mathbf{w} - \mathbf{w}_{t-1}\|^2 \right\} \tag{5.22}$$

wobei $\mathbf{G}, \mathbf{D}, \mathbf{\Phi}, \mathbf{r}$ wie in (5.12),(5.3) und σ der Regularisierungsparameter ist. Wir berechnen die Ableitung der in (5.22) zu minimierenden Funktion und setzen sie gleich Null:

$$0 = (\mathbf{G}\mathbf{\Phi})^\mathsf{T}(\mathbf{G}\mathbf{\Phi})\mathbf{w} - (\mathbf{G}\mathbf{\Phi})^\mathsf{T}\big[(\mathbf{G}-\mathbf{D})\mathbf{\Phi}\mathbf{w}_{t-1} + \mathbf{r}\big] + \sigma(\mathbf{w} - \mathbf{w}_{t-1}).$$

Damit folgt für die Lösung $\hat{\mathbf{w}}_t$ von (5.22)

$$\Big[(\mathbf{G}\mathbf{\Phi})^\mathsf{T}(\mathbf{G}\mathbf{\Phi}) + \sigma\mathbf{I}\Big]\hat{\mathbf{w}}_t = (\mathbf{G}\mathbf{\Phi})^\mathsf{T}\Big[\mathbf{G}\mathbf{\Phi}\mathbf{w}_{t-1} + \mathbf{r} - \mathbf{D}\mathbf{\Phi}\mathbf{w}_{t-1}\Big] + \sigma\mathbf{w}_{t-1}$$

und so

$$\begin{aligned}
\hat{\mathbf{w}}_t &= \Big[(\mathbf{G}\mathbf{\Phi})^\mathsf{T}(\mathbf{G}\mathbf{\Phi}) + \sigma\mathbf{I}\Big]^{-1}\Big[(\mathbf{G}\mathbf{\Phi})^\mathsf{T}\mathbf{G}\mathbf{\Phi}\mathbf{w}_{t-1} + \sigma\mathbf{w}_{t-1} + (\mathbf{G}\mathbf{\Phi})^\mathsf{T}\big(\mathbf{r} - \mathbf{D}\mathbf{\Phi}\mathbf{w}_{t-1}\big)\Big] \\
&= \mathbf{w}_{t-1} + \Big[(\mathbf{G}\mathbf{\Phi})^\mathsf{T}(\mathbf{G}\mathbf{\Phi}) + \sigma\mathbf{I}\Big]^{-1}(\mathbf{G}\mathbf{\Phi})^\mathsf{T}\big(\mathbf{r} - \mathbf{D}\mathbf{\Phi}\mathbf{w}_{t-1}\big)
\end{aligned} \tag{5.23}$$

Eingesetzt in (5.21) erhalten wir so insgesamt für den neuen Gewichtsvektor $\mathbf{w}_t$

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \eta_t\Big[(\mathbf{G}\mathbf{\Phi})^\mathsf{T}(\mathbf{G}\mathbf{\Phi}) + \sigma\mathbf{I}\Big]^{-1}(\mathbf{G}\mathbf{\Phi})^\mathsf{T}\big(\mathbf{r} - \mathbf{D}\mathbf{\Phi}\mathbf{w}_{t-1}\big). \tag{5.24}$$

LSPE-0 (dual)

Wir wenden die Matrixinversionsformel (B.3) auf (5.23) an, so daß (5.24) geschrieben werden kann als

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \eta_t(\mathbf{G}\mathbf{\Phi})^\mathsf{T}\Big[\mathbf{G}\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T}\mathbf{G}^\mathsf{T} + \sigma\mathbf{I}\Big]^{-1}\big(\mathbf{r} - \mathbf{D}\mathbf{\Phi}\mathbf{w}_{t-1}\big).$$

Alle iterierten Gewichtsvektoren $\mathbf{w}_i$ liegen (für $\mathbf{w}_0 = \mathbf{0}$) im Spaltenerzeugnis von $(\mathbf{G}\mathbf{\Phi})^\mathsf{T}$ und können demnach wieder durch duale Variablen $\boldsymbol{\alpha}_i$ mit $(\mathbf{G}\mathbf{\Phi})^\mathsf{T}\boldsymbol{\alpha}_i = \mathbf{w}_i$ ausgedrückt werden. Damit erhalten wir für (5.24)

$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1} + \eta_t\mathbf{\Phi}^\mathsf{T}\mathbf{G}^\mathsf{T}\hat{\boldsymbol{\alpha}}_t \quad \text{mit} \quad \hat{\boldsymbol{\alpha}}_t = \Big[\mathbf{G}\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T}\mathbf{G}^\mathsf{T} + \sigma\mathbf{I}\Big]^{-1}\big(\mathbf{r} - \mathbf{D}\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T}\mathbf{G}^\mathsf{T}\boldsymbol{\alpha}_{t-1}\big) \tag{5.25}$$

LSPE-0 (SR-Approximation)

Jetzt ist wieder zu zeigen, wie sich mit der SR-Approximation (5.5) für eine m-elementige Teilmenge der Trainingsdaten aus dem dualen $t \times t$ Problem in (5.25) ein reduziertes $m \times m$ Problem ergibt. Nehmen wir an, daß gerade die ersten m Daten die selektierten Daten der Teilmenge sind (ansonsten ordnen wir sie entsprechend um). Dann ist $\mathbf{K}_{tm}$ von der Form $\begin{bmatrix}\mathbf{K}_{mm} & * \end{bmatrix}^\mathsf{T}$. Die SR-Approximation ist nun äquivalent damit, daß wir in (5.25) alle diejenigen Koeffizienten in den dualen Variablen, also den $t \times 1$ Vektoren $\boldsymbol{\alpha}_i$ bzw. $\hat{\boldsymbol{\alpha}}_i$, gleich Null setzen, die zu den $t - m$ Trainingsdaten gehören, die *nicht* selektiert sind:

$$\boldsymbol{\alpha}_t \triangleq \begin{bmatrix} \boldsymbol{\alpha}_t^m \\ \mathbf{0} \end{bmatrix} \tag{5.26}$$

nur die ersten m Einträge $\boldsymbol{\alpha}_t^m$ müssen bestimmt werden.

Setzen wir damit (5.5) in Vorhersagen (5.1) für beliebige Argumente $\mathbf{x}$ ein:

$$
\begin{aligned}
\tilde{Q}(\mathbf{x};\mathbf{w}_t) &\overset{(5.1)}{=} \boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\mathbf{w}_t \\
&\overset{(5.25)}{=} \underbrace{\boldsymbol{\phi}(\mathbf{x})^\mathsf{T}\boldsymbol{\Phi}^\mathsf{T}}_{\mathbf{k}(\mathbf{x})^\mathsf{T}}\mathbf{G}^\mathsf{T}\Big[\boldsymbol{\alpha}_{t-1} + \eta_t\big(\mathbf{G}\underbrace{\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}}_{\mathbf{K}}\mathbf{G}^\mathsf{T} + \sigma\mathbf{I}\big)^{-1}\big(\mathbf{r} - \mathbf{D}\underbrace{\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}}_{\mathbf{K}}\mathbf{G}^\mathsf{T}\boldsymbol{\alpha}_{t-1}\big)\Big] \\
&\overset{(5.5)}{\approx} \mathbf{k}_m(\mathbf{x})^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{G}^\mathsf{T}\Big[\boldsymbol{\alpha}_{t-1} + \eta_t\big(\mathbf{G}\mathbf{K}_{t+1,m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{G}^\mathsf{T} + \sigma\mathbf{I}\big)^{-1}\cdot \\
&\qquad\qquad \cdot\big(\mathbf{r} - \mathbf{D}\mathbf{K}_{t+1,m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{G}^\mathsf{T}\boldsymbol{\alpha}_{t-1}\big)\Big] \\
&\overset{(B.3)}{=} \mathbf{k}_m(\mathbf{x})^\mathsf{T}\Big[\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{G}^\mathsf{T}\boldsymbol{\alpha}_{t-1} + \eta_t\big((\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}(\mathbf{G}\mathbf{K}_{t+1,m}) + \sigma\mathbf{K}_{mm}\big)^{-1}\cdot \\
&\qquad\qquad \cdot(\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}\big(\mathbf{r} - \mathbf{D}\mathbf{K}_{t+1,m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{G}^\mathsf{T}\boldsymbol{\alpha}_{t-1}\big)\Big] \\
&\overset{(5.26)\,(5.12)}{=} \mathbf{k}_m(\mathbf{x})^\mathsf{T}\Big[\underbrace{\mathbf{K}_{mm}^{-1}\begin{bmatrix}\mathbf{K}_{mm} & *\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}_{t-1}^m \\ \mathbf{0}\end{bmatrix}}_{\boldsymbol{\alpha}_{t-1}^m} + \eta_t\big((\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}(\mathbf{G}\mathbf{K}_{t+1,m}) + \sigma\mathbf{K}_{mm}\big)^{-1}\cdot \\
&\qquad\qquad \cdot(\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}\big(\mathbf{r} - \mathbf{D}\mathbf{K}_{t+1,m}\underbrace{\mathbf{K}_{mm}^{-1}\begin{bmatrix}\mathbf{K}_{mm} & *\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}_{t-1}^m \\ \mathbf{0}\end{bmatrix}}_{\boldsymbol{\alpha}_{t-1}^m}\big)\Big] \\
&= \mathbf{k}_m(\mathbf{x})^\mathsf{T}\Big[\boldsymbol{\alpha}_{t-1}^m + \eta_t\big[(\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}(\mathbf{G}\mathbf{K}_{t+1,m}) + \sigma\mathbf{K}_{mm}\big]^{-1}\cdot \\
&\qquad\qquad \cdot(\mathbf{G}\mathbf{K}_{t+1,m})^\mathsf{T}\big(\mathbf{r} - \mathbf{D}\mathbf{K}_{t+1,m}\boldsymbol{\alpha}_{t-1}^m\big)\Big]
\end{aligned}
$$

so daß es auch hier ausreicht, nur den reduzierten $m \times 1$ Gewichtsvektor $\boldsymbol{\alpha}_t^m$ zu bestimmen. Im folgenden schreiben wir aus Konsistenzgründen wieder $\boldsymbol{\alpha}_t$ anstatt $\boldsymbol{\alpha}_t^m$, meinen damit aber wieder den reduzierten Gewichtsvektor.

Aus (5.22) erhalten wir nun in den reduzierten Variablen das Least-Squares Problem

$$\hat{\boldsymbol{\alpha}}_t = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^m}\Big\{\|\mathbf{K}_{tm}\boldsymbol{\alpha} - \mathbf{K}_{tm}\boldsymbol{\alpha}_{t-1} - (\mathbf{r} + \mathbf{H}_{tm}\boldsymbol{\alpha}_{t-1})\|^2 + \sigma(\boldsymbol{\alpha} - \boldsymbol{\alpha}_{t-1})^\mathsf{T}\mathbf{K}_{mm}(\boldsymbol{\alpha} - \boldsymbol{\alpha}_{t-1})\Big\} \tag{5.27}$$

wobei wieder $\mathbf{G}\mathbf{K}_{t+1,m} = \mathbf{K}_{tm}$ aus (5.15) und $\mathbf{D}\mathbf{K}_{t+1,m} =: \mathbf{H}_{tm}$ aus (5.9) benutzt wurde. Für den neuen (dualen) Gewichtsvektor $\boldsymbol{\alpha}_t$ korrespondierend zu (5.21) erhalten wir so insgesamt

$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1} + \eta_t\Big[\mathbf{K}_{tm}^\mathsf{T}\mathbf{K}_{tm} + \sigma\mathbf{K}_{mm}\Big]^{-1}\Big[\mathbf{K}_{tm}^\mathsf{T}\mathbf{r} - \mathbf{K}_{tm}^\mathsf{T}\mathbf{H}_{tm}\boldsymbol{\alpha}_{t-1}\Big] \tag{5.28}$$

LSPE-λ

Für den allgemeineren Fall $\lambda > 0$ betrachten wir, korrespondierend zu LSTD-λ in (5.17), anstatt (5.22) das folgende (primale) Least-Squares Problem:

$$\hat{\mathbf{w}}_t = \operatorname*{argmin}_{\mathbf{w}}\Big\{\|\mathbf{G}\boldsymbol{\Phi}\mathbf{w} - (\mathbf{G} - \boldsymbol{\Lambda}\mathbf{D})\boldsymbol{\Phi}\mathbf{w}_{t-1} - \boldsymbol{\Lambda}\mathbf{r}\|^2 + \sigma\,\|\mathbf{w} - \mathbf{w}_{t-1}\|^2\Big\}$$

mit $\boldsymbol{\Lambda}$ aus (5.18). Wie beim verwandten LSTD-λ so besteht auch hier die Änderung gegenüber dem Fall $\lambda = 0$ darin, Matrix $\mathbf{D}$ durch $\boldsymbol{\Lambda}\mathbf{D}$ und $\mathbf{r}$ durch $\boldsymbol{\Lambda}\mathbf{r}$ zu ersetzen. Mit der gleichen Argumentation wie für LSPE-0 betrachten wir dann insgesamt statt (5.27) das Least-Squares Problem

$$\hat{\boldsymbol{\alpha}}_t = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^m}\Big\{\|\mathbf{K}_{tm}\boldsymbol{\alpha} - \mathbf{K}_{tm}\boldsymbol{\alpha}_{t-1} - \boldsymbol{\Lambda}(\mathbf{r} + \mathbf{H}_{tm}\boldsymbol{\alpha}_{t-1})\|^2 + \sigma(\boldsymbol{\alpha} - \boldsymbol{\alpha}_{t-1})^\mathsf{T}\mathbf{K}_{mm}(\boldsymbol{\alpha} - \boldsymbol{\alpha}_{t-1})\Big\} \tag{5.29}$$

Den neuen Gewichtsvektor $\boldsymbol{\alpha}_t$ erhalten wir dann insgesamt statt durch (5.28) durch

$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1} + \eta_t\Big[\mathbf{K}_{tm}^\mathsf{T}\mathbf{K}_{tm} + \sigma\mathbf{K}_{mm}\Big]^{-1}\Big[\mathbf{Z}_{tm}^\mathsf{T}\mathbf{r} - \mathbf{Z}_{tm}^\mathsf{T}\mathbf{H}_{tm}\boldsymbol{\alpha}_{t-1}\Big] \tag{5.30}$$

wobei wir wieder die Abkürzung $\mathbf{Z}_{tm} := \boldsymbol{\Lambda}^\mathsf{T}\mathbf{K}_{tm}$ verwenden.

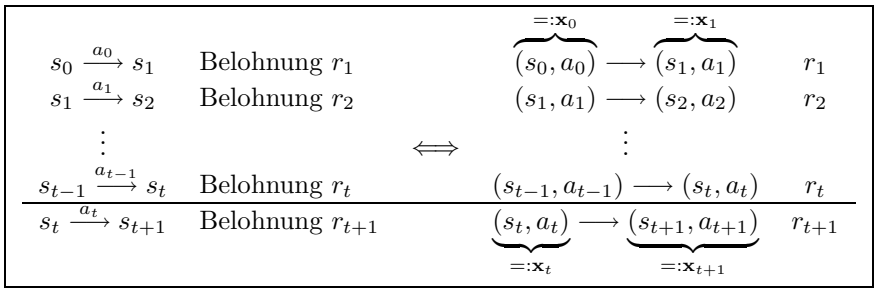


Abbildung 5.1: Bei APE sind die 'Trainingsdaten' für den Funktionsapproximator die beobachteten Zustandsübergänge unter Politik π, d.h. $a_i = \pi(s_i)$. Linke Seite: gewöhnliche Notation in Zustand-Aktion-Folgezustand. Rechte Seite: erweiterte Notation für die Q-Funktion als Prozeß zwischen Zustands-Aktions-Paaren $\mathbf{x}_i := (s_i, a_i)$.

5.1.3 Algorithmus: Online Politik-Evaluation mit APE

Jetzt wollen wir den Online-Algorithmus aus Kapitel 4 zur rekursiven Lösung der Probleme BRM in (5.8), LSTD in (5.19) und LSPE in (5.30) anwenden, indem in jedem Zeitschritt ein neuer Zustandsübergang beobachtet (oder abgearbeitet) wird. Unter Politik π, d.h. $a_i = \pi(s_i)$, sehen die 'Trainingsdaten' wie in Abbildung 5.1 gezeigt aus.

Gegeben die jeweilige Lösung zum Zeitpunkt t, so sind wir an der Aktualisierung des Gewichtsvektors $\mathbf{w}_{tm}$ in der Approximation (mit $\mathbf{w} = \mathbf{w}_{tm}$)

$$\tilde{Q}(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{m} k(\tilde{\mathbf{x}}_i \mathbf{x}) w_i = \mathbf{k}_m(\mathbf{x})^\mathsf{T} \mathbf{w} \tag{5.31}$$

für den Zeitpunkt $t+1$ interessiert, wenn wir als folgenden Zustand s_{t+1} unter r_{t+1} beobachten. Gegenüber Abschnitt 5.1.1 betrachten wir jetzt von vorneherein das dualisierte reduzierte Problem in der Repräsentation mit der Kernfunktion und bezeichnen auch den dualen Gewichtsvektor mit dem Symbol '$\mathbf{w}$' anstatt '$\boldsymbol{\alpha}$' (aus Konsistenzgründen mit Kapitel 4). Zudem behandeln wir jetzt auch wieder den Übergang von t nach $t+1$. Alle auftretenden Größen werden mit zwei Indizes versehen: der erste Index t bzw. $t+1$ markiert den Zeitschritt, der zweite Index m bzw. $m+1$ die Anzahl der Basisfunktionen.

Wir stellen fest, daß alle drei Methoden BRM in (5.8), LSTD in (5.19) und LSPE in (5.30) auf ein sehr ähnlich aussehendes lineares Gleichungssystem hinauslaufen. Um diese Gemeinsamkeit noch stärker herauszustellen, überladen wir die Notation und fassen unsere Ergebnisse aus Abschnitt 5.1.1 kompakt zusammen zu:

- **BRM:** Der Gewichtsvektor $\mathbf{w}_{t+1,m}$ ergibt sich als Lösung des regularisierten LS-Problems

$$\min_{\mathbf{w}} \left\{ \|\mathbf{H}_{t+1,m}\mathbf{w} - \mathbf{r}_{t+1}\|^2 + \sigma \mathbf{w}^\mathsf{T} \mathbf{K}_{mm} \mathbf{w} \right\} \tag{5.7'}$$

  als

$$\mathbf{w}_{t+1,m} = \mathbf{P}_{t+1,m}^{-1} \mathbf{b}_{t+1,m} \tag{5.8'}$$

  wobei

  - $\mathbf{P}_{t+1,m}^{-1} := (\mathbf{H}_{t+1,m}^\mathsf{T} \mathbf{H}_{t+1,m} + \sigma \mathbf{K}_{mm})^{-1}$
  - $\mathbf{b}_{t+1,m} := \mathbf{H}_{t+1,m}^\mathsf{T} \mathbf{r}_{t+1}$

- **LSTD:** Der Gewichtsvektor $\mathbf{w}_{t+1,m}$ ergibt sich als Lösung der folgenden Fixpunktgleichung in $\hat{\mathbf{w}}$

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^m} \left\{ \|\mathbf{K}_{t+1,m}\mathbf{w} - \mathbf{K}_{t+1,m}\hat{\mathbf{w}} - \boldsymbol{\Lambda}(\mathbf{r}_{t+1} - \mathbf{H}_{t+1,m}\hat{\mathbf{w}})\|^2 + \sigma \mathbf{w}^\mathsf{T} \mathbf{K}_{mm} \mathbf{w} \right\} \tag{5.17'}$$

als

$$\mathbf{w}_{t+1,m} = \mathbf{P}_{t+1,m}^{-1}\mathbf{b}_{t+1,m} \tag{5.19'}$$

wobei

- $\mathbf{P}_{t+1,m}^{-1} := (\mathbf{Z}_{t+1,m}^{\mathsf{T}}\mathbf{H}_{t+1,m} + \sigma\mathbf{K}_{mm})^{-1}$
- $\mathbf{b}_{t+1,m} := \mathbf{Z}_{t+1,m}^{\mathsf{T}}\mathbf{r}_{t+1}$

- **LSPE:** Der Gewichtsvektor $\mathbf{w}_{t+1,m}$ ergibt sich mit der Lösung des regularisierten LS-Problems

$$\min_{\mathbf{w}\in\mathbb{R}^m} \left\{ \|\mathbf{K}_{t+1,m}\mathbf{w} - \mathbf{K}_{t+1,m}\mathbf{w}_{tm} - \mathbf{\Lambda}(\mathbf{r}_{t+1} - \mathbf{H}_{t+1,m}\mathbf{w}_{tm})\|^2 + \sigma(\mathbf{w} - \mathbf{w}_{tm})^{\mathsf{T}}\mathbf{K}_{mm}(\mathbf{w} - \mathbf{w}_{tm}) \right\} \tag{5.29'}$$

als

$$\mathbf{w}_{t+1,m} = \mathbf{w}_{tm} + \eta_{t+1}\mathbf{P}_{t+1,m}^{-1}(\mathbf{b}_{t+1,m} - \mathbf{A}_{t+1,m}\mathbf{w}_{tm}) \tag{5.30'}$$

wobei

- $\mathbf{P}_{t+1,m}^{-1} := (\mathbf{K}_{t+1,m}^{\mathsf{T}}\mathbf{K}_{t+1,m} + \sigma\mathbf{K}_{mm})^{-1}$
- $\mathbf{b}_{t+1,m} := \mathbf{Z}_{t+1,m}^{\mathsf{T}}\mathbf{r}_{t+1}$
- $\mathbf{A}_{t+1,m} := \mathbf{Z}_{t+1,m}^{\mathsf{T}}\mathbf{H}_{t+1,m}$

Für einen zeitrekursiven Lösungsalgorithmus laufen demnach alle drei Verfahren auf eine rekursive Anpassung der inversen Matrix $\mathbf{P}_{tm}^{-1}$ und des Gewichtsvektors $\mathbf{w}_{tm}$ für jeden Übergang von t nach $t+1$ hinaus. Genau das hatten wir aber in Kapitel 4 diskutiert: vergleichen wir (5.8'),(5.19'),(5.30') mit (4.4), dann stellen wir fest, daß der Algorithmus in Abbildung 4.4 zur rekursiven Aktualisierung der Matrizen für Problem (4.1) auch zur rekursiven Aktualisierung der Matrizen in (5.8'),(5.19'),(5.30') herangezogen werden kann. Wir verzichten daher an dieser Stelle auf eine erneute Herleitung der zugehörigen Rekursionsgleichungen und präsentieren gleich den Algorithmus in Abbildung 5.2. Dessen Struktur ist für alle drei Methoden BRM, LSTD und LSPE identisch; nur die konkreten Gleichungen sind unterschiedlich (siehe Anhang C). Zu beachten ist allerdings, daß wir im Fall von LSTD kein direktes Least-Squares Problem lösen und wir daher eine überwachte Basisselektion wie in (4.24) beschrieben nicht durchführen können. Zu beachten ist auch, daß im Fall von LSTD die der Inversen $\mathbf{P}_{tm}^{-1}$ entsprechende Matrix nicht mehr symmetrisch ist.

5.1.4 Lernen der optimalen Politik mit API/OPI (online)

Haben wir den Teilschritt Politik-Evaluation gelöst, so müssen wir uns jetzt Gedanken zur Integration in den gesamten API-Zyklus (vgl. Abbildung 1.7, Seite 12) zum näherungsweisen Lernen der optimalen Politik machen. Folgende Vorgehensweisen bieten sich dabei an:

Klassische API (Actor/Critic)

Wir iterieren APE und Politikverbesserung, wobei wir für APE Zustandsübergänge durch Simulation der zu evaluierenden festen Politik generieren (d.h. der Agent interagiert mit der Umwelt und wählt dabei Aktionen gemäß der aktuellen Politik aus). Das Verfahren ist ein sogenanntes *on-policy* Verfahren.

Zur Implementation sind zwei unabhängige Instanzen des RN nötig: eines (das Actor-Netzwerk) ist die gelernte Wertefunktion aus dem letzten Iterationsschritt und repräsentiert die gegenwärtige Politik π_k (mittels gieriger Ableitung der Aktionen). Das zweite (das Critic-Netzwerk) ist dasjenige, das die Wertefunktion von π_k repräsentiert, und dessen Gewichtsvektor/Basis mit dem in Abbildung 5.2 beschriebenen Algorithmus gelernt wird. Die Politik-Iteration besteht dann darin, nach einer hinreichend großen Anzahl von Evaluationsschritten das Critic-Netzwerk in das Actor-Netzwerk umzukopieren, und mit einem frisch initialisierten Critic dann dessen Evaluation zu beginnen.

Relevante Symbole:

// π: Politik, dessen Wertefunktion Q^π approximiert werden soll
// t: Anzahl bisher beobachteter Zustandsübergänge
// m: Anzahl bisher hinzugefügter Basisfunktionen in $\mathcal{BV}$
// $\mathbf{P}_{tm}^{-1}$: Inverse Kreuzproduktmatrix zur Berechnung von $\mathbf{w}_{tm}$
// $\mathbf{w}_{tm}$: Gewichtsvektor in $\tilde{Q}(\cdot\,;\mathbf{w}_{tm})$, der aktuellen Approximation für Q^π
// $\mathbf{K}_{mm}^{-1}$: Wird gebraucht, um $\mathbf{a}_{t+1}$ für Approximation des Kerns auszurechnen

Initialisierung:

Generiere ersten Zustand s_0. Wähle Aktion $a_0 = \pi(s_0)$. Führe a_0 aus und beobachte s_1 und r_1. Wähle $a_1 = \pi(s_1)$. Setze $\mathbf{x}_0 := (s_0, a_0)$ und $\mathbf{x}_1 := (s_1, a_1)$. Initialisiere Menge von Basisfunktionen $\mathcal{BV} := \{\mathbf{x}_0, \mathbf{x}_1\}$ und $\mathbf{K}_{2,2}^{-1}$. Initialisiere $\mathbf{P}_{1,2}^{-1}$, $\mathbf{w}_{1,2}$ je nach der Methode LSPE, LSTD oder BRM. Setze $t := 1$ and $m := 2$.

Hautpschleife:

Führe Aktion a_t aus (simuliere Zustandsübergang).
Beobachte nächsten Zustand s_{t+1} und Belohnung r_{t+1}.
Wähle nächste Aktion $a_{t+1} = \pi(s_{t+1})$. Setze $\mathbf{x}_{t+1} := (s_{t+1}, a_{t+1})$.

1. Basiserweiterungstest Teste, ob $\mathbf{x}_{t+1}$ zu Basisfunktionen hinzugefügt werden soll.
Unüberwacht: TRUE, FALLS (C.1)> `TOL1` (der Fall LSTD ist nur unüberwacht möglich).
Überwacht: TRUE, FALLS (C.1)> `TOL1`
und zusätzlich entweder (C.7) oder (C.7")> `TOL2`.

2. Normaler Schritt
Berechne $\mathbf{P}_{t+1,m}^{-1}$ aus (C.3),(C.3'), oder (C.3").
Berechne $\mathbf{w}_{t+1,m}$ aus (C.4),(C.4'), oder (C.4").

3. Basiserweiterungsschritt (nur wenn Basiserweiterungstest erfolgreich)
Berechne $\mathbf{P}_{t+1,m+1}^{-1}$ aus (C.5),(C.5'), oder (C.5").
Berechne $\mathbf{w}_{t+1,m+1}$ aus (C.6),(C.6'), oder (C.6").
Füge $\mathbf{x}_{t+1}$ der Menge $\mathcal{BV}$ hinzu und aktualisiere $\mathbf{K}_{m+1,m+1}$ mit (C.2).
$m := m + 1$

$t := t + 1$, $s_t := s_{t+1}$, $a_t := a_{t+1}$

Abbildung 5.2: Online APE mit Regularisierungsnetzen für die drei Verfahren BRM, LSTD und LSPE. Die grundlegende Struktur ist wie in Abbildung 4.4. Die genauen Rekursionsgleichungen sind in Anhang C angegeben. Der Rechenaufwand pro Zustandsübergang beträgt $\mathcal{O}(m^2)$.

Off-policy API

Ein Problem der klassischen On-policy API ist, daß sie selbst nicht effizient mit den Trainingsdaten (beobachteten Zustandsübergängen) umgeht; unter Umständen ist eine sehr große Zahl Zustandsübergänge notwendig, um die aktuelle Politik hinreichend genau auszuwerten. Weil das Verfahren ein On-Policy Verfahren ist, können einmal beobachtete Zustandsübergänge auch nicht zwischen den Iterationen wiederverwendet werden, sondern müssen für jede (und unter jeder) betrachteten Politik erneut generiert werden.

Lagoudakis und Parr (2003) schlagen deswegen eine *Off-policy*[6] Variante vor, bei der für alle Politik-Iterationen *dieselbe* Menge von Zustandsübergängen verwendet werden kann. Sie funktioniert folgendermaßen:

1. Generiere (einmalig) Menge von n Übergängen $\mathcal{M} := \{(s_i, a_i, s_i', r_i)\}_{i=1}^n$, wobei die Aktionen a_i nach einer beliebigen explorativen Politik ausgewählt werden, z.B. zufällig

2. Wähle Startpolitik π_0 repräsentiert durch Gewichtsvektor $\mathbf{w}_{-1}$

3. Iteriere $k = 0, 1, 2, \ldots$

[6] D.h. die Politik, unter der die Stichprobenübergänge erzeugt werden (d.h. die Politik, die der Agent ausführt), ist nicht identisch mit der Politik, deren Wertefunktion mit APE ausgerechnet werden soll.

(a) Assembliere aus $\mathcal{M}$ die Trainingsdaten $\mathcal{M}_k$ zur Evaluation von π_k:

$$\mathbf{x}_i := (s_i, a_i) \longrightarrow (s'_i, \pi_k(s'_i)) =: \mathbf{x}'_i \text{ unter Belohnung } r_i, \quad i = 1, \ldots, n$$

Dabei ist π_k gierige Politik bzgl. $\tilde{Q}^{\pi_{k-1}}$, repräsentiert durch Gewichtsvektor $\mathbf{w}_{k-1}$, d.h.

$$\pi_k(s'_i) = \underset{a'}{\text{argmax}}\ \tilde{Q}([s'_i, a_i]; \mathbf{w}_{k-1}), \quad i = 1, \ldots, n$$

(b) Führe APE mit Trainingsdaten aus $\mathcal{M}_k$ durch und erhalte Gewichtsvektor $\mathbf{w}_k$ für $\tilde{Q}^{\pi_k}$. (Zu beachten ist, daß diese Variante, anders als die zuvor betrachtete, off-policy ist und die Daten daher nicht mehr aus einer zusammenhängenden Trajektorie stammen, sondern i.a. $\mathbf{x}'_i \neq \mathbf{x}_{i+1}$ ist, vgl. auch Abbildung 5.1).

Der entscheidende Vorteil dieser Variante ist, daß sie in hohem Maße dateneffizient ist, weil die Stichprobenübergänge nur einmal (außerhalb der Hauptschleife) gesammelt werden müssen und dann zur Evaluation *aller* Politik verwendet werden können.

Ein (möglicherweise hauptsächlich die Theorie betreffendes) Problem dabei ist, daß die verwendeten Stichprobenübergänge nicht mehr länger unter der jeweils zu evaluierenden Politik beobachtet werden, das Verfahren also *off-policy* ist. Damit ist die Konvergenz in (1.22) nicht mehr gegeben, bzw. müßte der Unterschied in den Verteilungen durch eine andere Gewichtung der Stichprobenübergänge ausgeglichen werden. In Lagoudakis und Parr (2003) wird dieser Aspekt in den praktischen Experimenten ignoriert; dennoch wurden dort gute Ergebnisse erzielt. Auch wir werden in unseren Experimenten im kommenden Abschnitt diesen Punkt ausblenden.

Ein zweites potentielles Problem dieser Vorgehensweise ist, daß die anfänglich (unter einer zufälligen Politik) erzeugten Daten nicht repräsentativ sind, in dem Sinne, daß gerade solche 'Erfahrungen' (Stichprobenübergänge) fehlen, die man nur beim Folgen einer guten oder beinahe-optimalen Politik erhält. Dieses Problem kann umgangen werden, wenn zu einem späteren Zeitpunkt neue Übergänge hinzugenommen werden.

OPI mit LSPE

Als spezielle Variante der klassischen On-policy API betrachten wir nicht zuletzt auch die ebenfalls dateneffiziente optimistische Politik-Iteration (OPI), bei der bereits nach jedem einzelnen beobachteten Zustandsübergang eine (implizite) Politikverbesserung erfolgt: die Aktionen werden nicht gemäß einer festen Politik, sondern als gierige Aktion bezüglich der aktuellen Approximation der Wertefunktion ausgewählt. OPI ist daher nur möglich zusammen mit Online-Lernen, weil nach jedem Schritt der Gewichtsvektor der Approximation aktualisiert werden muß. Von den hier betrachteten APE-Varianten BRM, LSTD und LSPE ist OPI, wie in Bertsekas und Tsitsiklis (1996) beschrieben, nur zusammen mit LSPE möglich. LSPE mit OPI ist von der Funktionsweise her am ehesten mit dem Standardverfahren Sarsa(λ) zu vergleichen (siehe Kapitel 1, Seite 15).

5.2 Einfache Beispiele

In diesem Abschnitt untersuchen und evaluieren wir unser Verfahren anhand der einfachen Testbeispiele 'Kettenwelt', 'Mountain-Car' und 'Acrobot'. Das Verfahren wurde anfänglich in MATLAB entwickelt und getestet, später aus Gründen der Laufzeiteffizienz nach C/C++ übertragen, womit auch die Anbindung an die verschiedenen Simulationen und Benchmarks möglich wird. Zur Steigerung der Effizienz greifen wir auf eine Prozessor-optimierte BLAS-Implementation[7] zurück, wobei im wesentlichen nur Level-1 und Level-2 Operationen benötigt werden.

[7] Frei verfügbar: ATLAS (`http://math-atlas.sourceforge.net`)

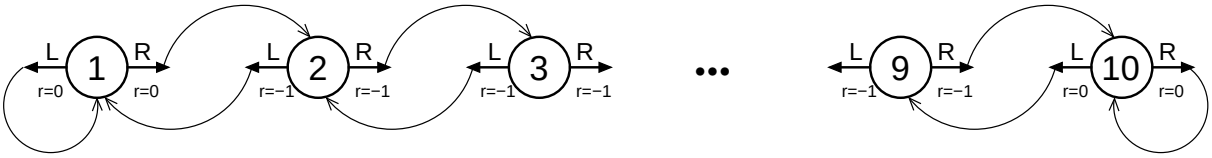


Abbildung 5.3: Deterministische Kettenwelt mit 10 Zuständen und Aktionen 'Links' und 'Rechts'. Ziel ist das Erreichen der Zustände '1' oder '10', weil nur hier die Kosten für einen Übergang $r = 0$ sind. Alle anderen Übergänge kosten $r = -1$.

5.2.1 Die Kettenwelt

Wir betrachten das in Abbildung 5.3 gezeigte Beispiel eines Markov-Entscheidungsprozeß mit 10 Zuständen (als Perlen einer Kette aufgereiht) und den zwei möglichen Aktionen 'Links' und 'Rechts'. Die Belohnung der einzelnen Übergänge ist 0 an den beiden Enden und -1 sonst. Dieses offensichtlich sehr simple Problem hat natürlich weniger praktische Relevanz, ist aber vortrefflich geeignet, um die Arbeitsweise unseres Verfahrens zu illustrieren und wichtige Aspekte auch visuell klarzumachen, denn hier können wir die optimale Lösung exakt ausrechnen und mit unserer gelernten Approximation vergleichen.

Deterministische Variante

Zunächst betrachten wir die in Abbildung 5.3 dargestellte Variante mit deterministischen Zustandsübergängen. Das Ziel ist es, die Arbeitsweise von Politik-Iteration zu verdeutlichen, wir verwenden dazu die von Lagoudakis und Parr (2003) vorgeschlagene Off-Policy Variante; zur APE vergleichen wir BRM und LSTD.

Zu Beginn der Iteration erzeugen wir uns einmalig 1000 Stichprobenübergänge (in der Form von Abbildung 5.1), indem wir jeweils einen zufälligen Startzustand auswählen, und dann für 5 Schritte einer explorativen Politik folgen, die mit der gleichen Wahrscheinlichkeit die Aktion L oder R auswählt. Alle Politiken werden dann mit diesem einem Trainingsdatensatz ausgewertet, entweder mit dem Verfahren BRM oder LSTD.

Für beide APE Varianten verwenden wir identische Metaparameter. Betrachten wir den wichtigsten zuerst, die Kernfunktion: eine Besonderheit von modellfreiem RL ist, daß die Wertefunktion über Zustands-Aktionspaaren $\mathbf{x} := (s, a) \in \mathbb{R} \times \{'L', 'R'\}$ definiert ist und wir daher nicht einfach den gewöhnlich nur für vektorwertige Argumente definierten Kern nehmen können. Die Aktionen sind hier Teil der Objekte, über denen der Kern definiert ist. Ein erster Versuch, bei dem wir Aktionen binärkodiert als (1,0),(0,1) an den Zustandsvektor angehängt hatten, brachte keine zufriedenstellende Ergebnisse. Wir verwenden daher, wie in Engel et al. (2005a) vorgeschlagen, einen Produktkern der Form

$$k([s, a], [s', a']) := k_{\mathcal{S}}(s, s')k_{\mathcal{A}}(a, a'), \tag{5.32}$$

indem wir den gesamten Kern k in einen Zustandskern $k_{\mathcal{S}}$ und Aktionskern $k_{\mathcal{A}}$ aufspalten (das Produkt zweier Kerne ist wieder ein Kern). Für den über reellwertigen Zustandsvektoren definierten Kern $k_{\mathcal{S}}$ wählen wir wieder Gauß-RBF, für den über diskreten Elementen definierten Kern $k_{\mathcal{A}}$ das Kroneckerdelta[8]:

$$k_{\mathcal{S}}(s, s') := \exp\left\{-\frac{\|s - s'\|^2}{h}\right\}, \qquad k_{\mathcal{A}}(a, a') := \begin{cases} 1, & \text{falls } a = a' \\ 0, & \text{sonst} \end{cases} \tag{5.33}$$

Die Metaparameter im einzelnen sind nun: Breite des Kerns $h^{-1} = 0.5$, Toleranz bei der Basisselektion $\texttt{TOL1} = 0.01$, $\texttt{TOL2} = 0$, Regularisierung $\sigma = 0.01$, n-Schritt-Mittelung $\lambda = 0$ und Diskontfaktor $\gamma = 0.95$.

Abbildung 5.4 zeigt die Entwicklung bei 6 Iterationen mit LSTD-APE und Abbildung 5.5 die mit BRM-APE, wenn wir in beiden Fällen mit der (nicht-optimalen) Startpolitik links (LLLLLLLLLL) starten. In

[8] Die Idee ist, daß der Kern die Ähnlichkeit zwischen den Argumenten mißt. Bei diskreten Aktionen gibt es keine direkte Ähnlichkeit.

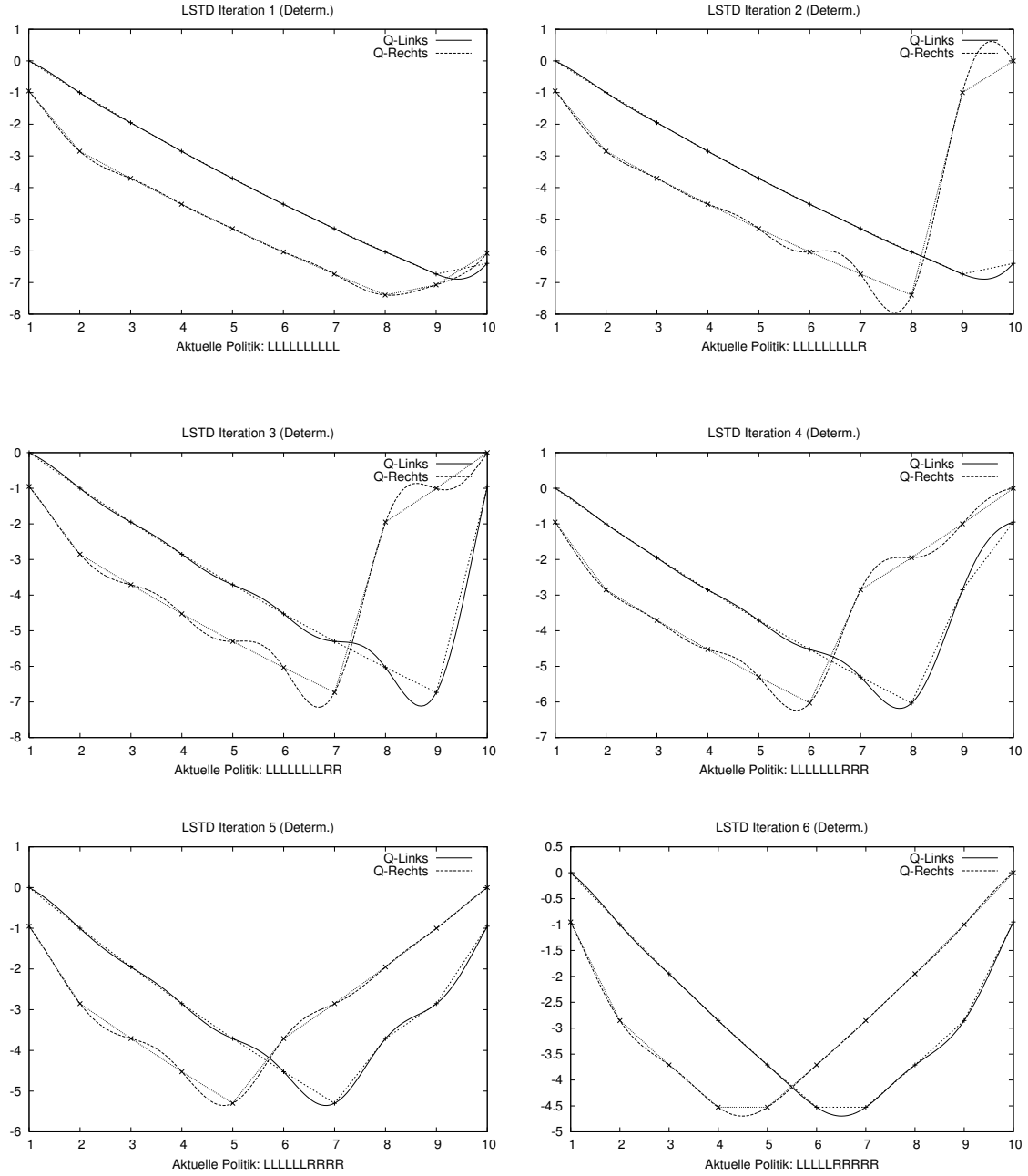


Abbildung 5.4: Approximative Politik-Iteration bei der deterministischen Kettenwelt: gezeigt werden die Zustände $\{1, \ldots, 10\}$ (x-Achse), die mit LSTD und RN gelernten Q-Funktionen $\tilde{Q}^{\pi_k}(\cdot, \text{'L'})$ und $\tilde{Q}^{\pi_k}(\cdot, \text{'R'})$, sowie die zugehörigen exakten Q-Funktionen (gepunktete Linien). Die jeweils evaluierte Politik π_k wird unten angegeben. Man sieht: LSTD findet die optimale Politik LLLLLRRRRR nach 6 Politik-Iterationen.

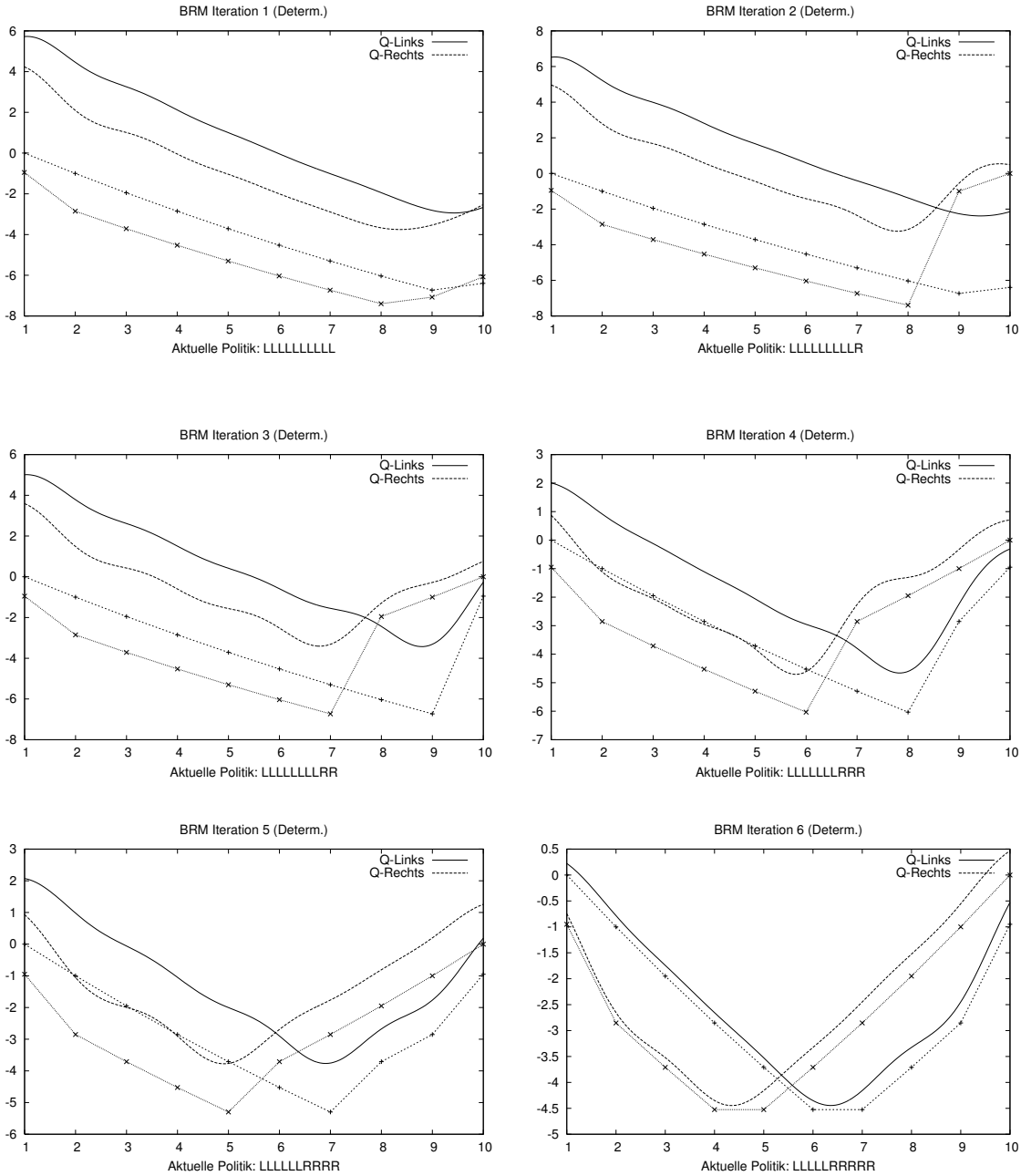


Abbildung 5.5: Approximative Politik-Iteration bei der deterministischen Kettenwelt: das gleiche Experiment wie in Abbildung 5.4, nur wird dieses mal BRM und RN zur APE verwendet. Man sieht: auch BRM findet die optimale Politik nach 6 Politik-Iterationen, die gelernten Approximationen der einzelnen Q-Funktionen sind allerdings schlechter als zuvor bei LSTD.

jedem Graph plotten wir für die aktuelle Politik π_k die beiden gelernten Q-Funktionen $\tilde{Q}^{\pi_k}(\cdot\,, \text{'links'})$ und $\tilde{Q}^{\pi_k}(\cdot\,, \text{'rechts'})$. Auf der x-Achse sind die Zustände, auf der y-Achse die zugehörigen Q-Werte eingetragen. Die Verbesserung der Politik zwischen den Iterationen besteht darin, in jedem Zustand die beste Aktion zu wählen, also das punktweise Maximum der beiden Zustands-Aktionsbewertungen. Zum Vergleich sind neben den Approximationen auch noch die exakten Q-Funktionen eingezeichnet (als 'Linespoints'-Kurven).

Die Ergebnisse zeigen, daß sowohl LSTD als auch BRM bereits nach 5 Iterationen die optimale Politik LLLLLRRRRR gefunden haben[9]. Mit dem verwendeten Regularisierungsnetz können wir dank der nichtparametrisierten Natur Architektur eine recht gute Approximation der exakten Werte erzielen – viel präziser als mit a priori parametrisierten RBF-Netzen oder Polynomen in einem vergleichbaren Experiment in Lagoudakis und Parr (2003). Obwohl schließlich beide Varianten LSTD und BRM konvergieren, zeigen sich in der Qualität der Approximation Unterschiede; die mit BRM gelernten erfassen zwar den ungefähren Verlauf der exakten Funktion, die gelernten Werte können aber mit relativ großen Abweichungen behaftet sein. Solange dies die Wertefunktionen beider Aktionen 'links' und 'rechts' in gleichem Maß betrifft, hat das keine Auswirkungen auf Konvergenz, weil für die Aktionswahl nur die relativen Werte eine Rolle spielen.

In Abbildung 5.6 zeigen wir aber einen zweiten unabhängigen Durchlauf für BRM mit leicht veränderten Metaparametern und neuem Datensatz. Hier produziert BRM wieder relativ große Abweichungen in der Approximation, die diesmal jedoch dazu führen, daß die Iteration bei der suboptimalen Politik LLLLLLLRRR hängenbleibt. Für LSTD (Ergebnisse nicht gezeigt) ist das nicht der Fall, hier konnte die optimale Politik gelernt werden.

Stochastische Variante

Um die Aufgabe etwas schwieriger zu gestalten, führen wir jetzt stochastische Zustandsübergänge ein: jede Aktion wird nur noch mit einer Wahrscheinlichkeit von 80% korrekt ausgeführt, ansonsten bewirkt mit 20% ein beabsichtigter Schritt in die eine Richtung genau einen Schritt in die entgegengesetzte Richtung (siehe Abbildung 5.7). In Abbildung 5.8 und 5.9 führen wir dann dasselbe Experiment (mit neu erzeugtem Datensatz) noch einmal durch.

Mit LSTD können wir auch bei stochastischen Zustandsänderungen die noch immer optimale Politik LLLLLRRRRR erlernen, die Approximation ist nun aber mit einer größeren Abweichung behaftet, als es zuvor der Fall war. Der näherungsweise Verlauf der Funktion wird aber immer noch richtig wiedergegeben, so daß Politikableitung erfolgreich zu besseren Politiken führt. Mit BRM ist das nicht mehr der Fall, hier konnten wir die optimale Politik bei keiner Konfiguration der Parameter erfolgreich lernen. Abbildung 5.9 zeigt ein Beispiel, wo die Iteration bereits nach 2 Schritten hängenbleibt. Das schlechte Abschneiden von BRM bei stochastischen Übergängen kommt aber nicht völlig unerwartet; schon in Kapitel 1 hatten wir darauf hingewiesen, daß das Annähern der Kreuzproduktmatrix mit nur einer Stichprobe keinen erwartungstreuen Schätzer liefert und das Ergebnis verzerren kann (siehe auch Seite 23).

5.2.2 Mountain-Car

Als nächstes Beispiel betrachten wir das schon in Kapitel 1 erläuterte Mountain-Car (siehe Seite 2 und Seite 16) mit 2-dimensionalem Zustandsraum. Dieses Szenario ist ein Standardbenchmark in der RL-Literatur (Sutton und Barto, 1998) und wird häufig zur Evaluation neuer Ansätze und Algorithmen eingesetzt.

Es handelt sich dabei um ein episodisches Problem mit ausgezeicheneten Terminalzuständen.[10] Mit einer Belohnung von $r = -1$ pro Schritt ist es die Aufgabe des Agenten, von jeder beliebigen Startposition aus

[9] Für alle weiteren Iterationen werden dann keine Änderungen mehr eintreten, wie man sich leicht durch Betrachten des punktweisen Maximums klarmachen kann

[10] Ein Terminalzustand wird formal so modelliert, daß er unter allen Aktionen mit Wahrscheinlichkeit 1 und Belohnung 0 in sich selbst übergeht und daher den Wert 0 hat. Beim Erreichen eines Terminalzustands wird (für diesen Übergang) $\gamma = 0$ gesetzt.

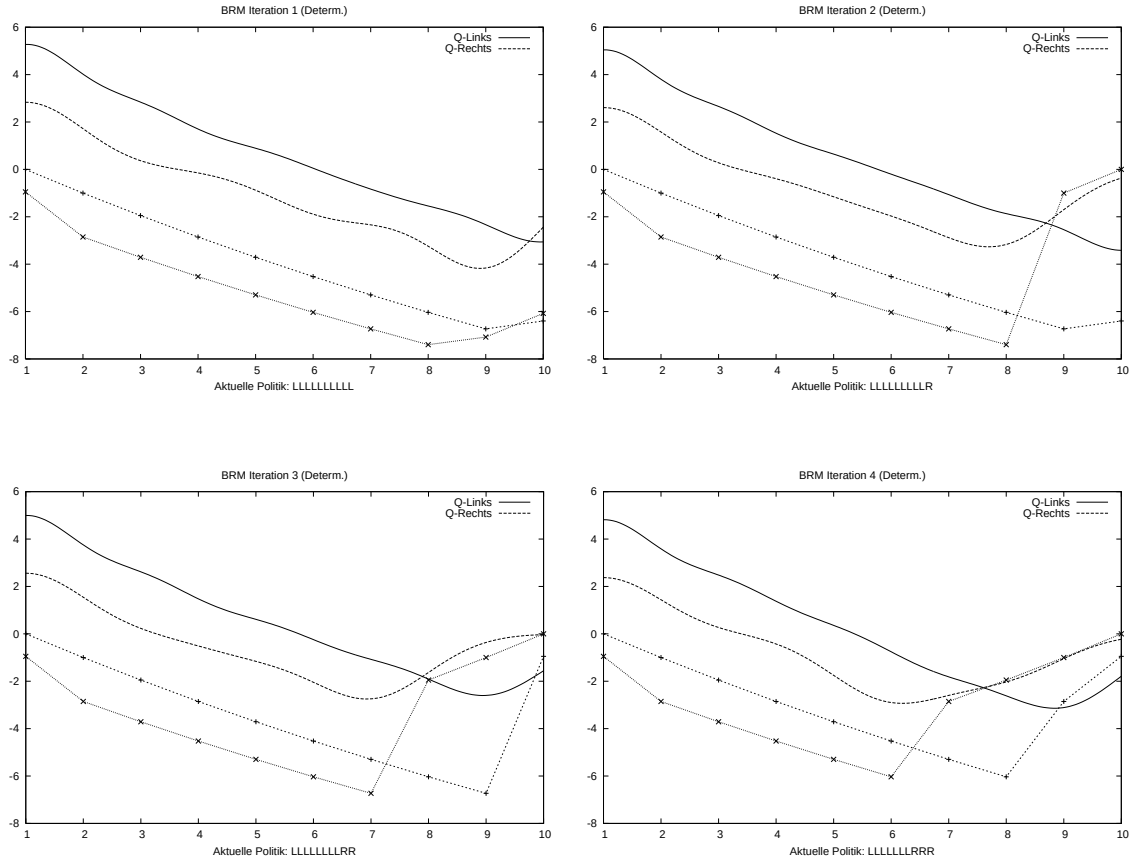


Abbildung 5.6: Approximative Politik-Iteration bei der deterministischen Kettenwelt: ein Beispiel dafür, daß mit BRM nicht immer die optimale Politik gefunden wird. Bei leicht veränderter Einstellung der Metaparameter und einem neuen Datensatz bleibt in diesem Beispiel die Iteration bei der suboptimalen Politik LLLLLLLRRR hängen.

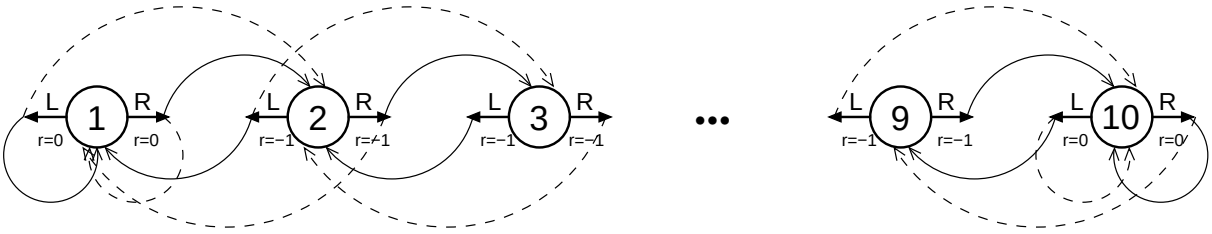


Abbildung 5.7: Stochastische Variante der Kettenwelt. Wie Abbildung 5.3, nur daß jetzt der Ausgang jeder Aktion ungewiß ist: nur mit 80% wird die Aktion wie beabsichtigt durchgeführt, mit 20% landet man dagegen in der entgegengesetzten Richtung.

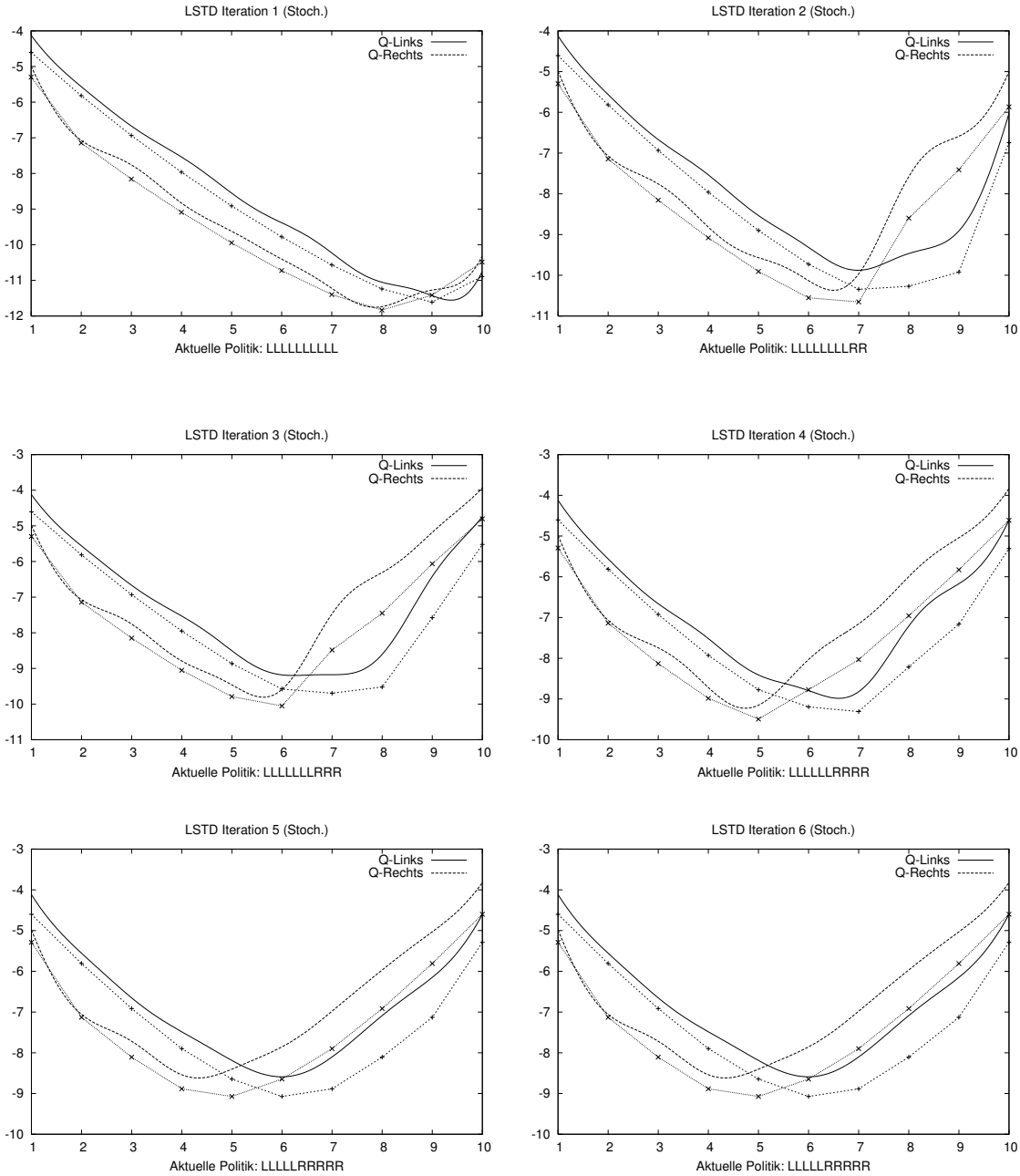


Abbildung 5.8: Approximative Politik-Evaluation für die stochastische Kettenwelt aus Abbildung 5.7. Auch hier findet LSTD mit RN die optimale Politik nach 6 Iterationen; die Approximationen der einzelnen Q-Funktionen sind nun aber mit einer größeren Abweichung behaftet, als es zuvor bei der deterministischen Variante der Fall war.

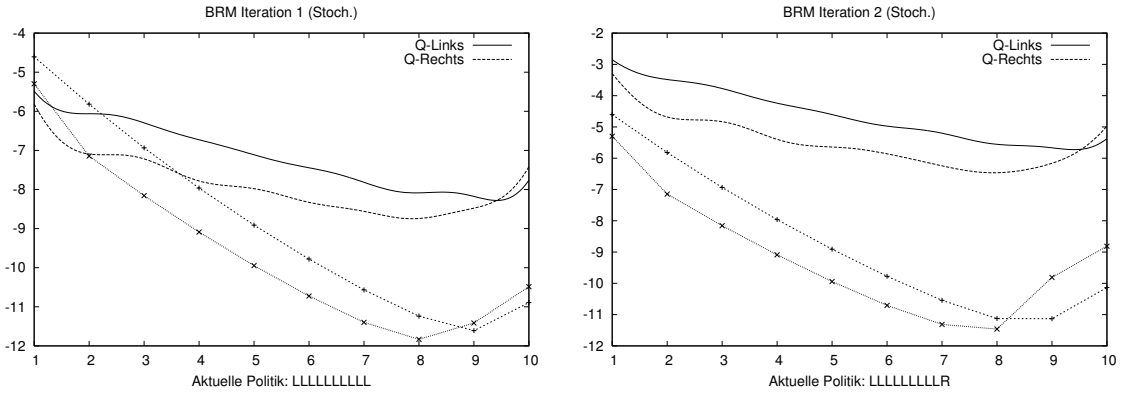


Abbildung 5.9: Approximative Politik-Evaluation für die stochastische Kettenwelt: das gleiche Experiment wie in Abbildung 5.8, nur wird dieses mal BRM und RN zur APE verwendet. Man sieht: BRM findet nicht die optimale Politik, sondern die API bleibt aufgrund einer mit großen Fehlern behafteten Approximation bei der suboptimalen Politik LLLLLLLLLR stecken. So ist in Zustand '9' für die wahre Q-Funktionen die Aktion 'R' besser als 'L'; bei den Approximationen ist dagegen genau das Gegenteil der Fall, so daß Politik-Ableitung hier keine Verbesserung bewirken kann.

den kürzesten Weg zum Ziel zu finden, d.h. die Episodenlänge zu minimieren. Auf der anderen Seite ist es aber auch möglich, daß unter einer ungünstigen Politik der Terminalzustand gar nicht erreicht wird; in diesem Fall brechen wir eine Episode nach 500 Schritten ab.

Off-Policy API mit LSTD

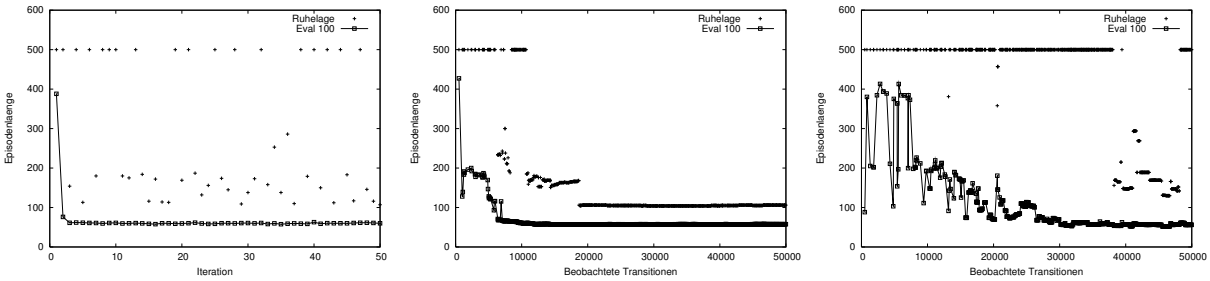
Zunächst betrachten wir als Lösungsverfahren wieder LSTD mit RN für Off-policy API wie im Kettenwelt-Szenario zuvor. Wie gehabt erzeugen wir zunächst eine Menge von 20000 Stichprobenübergängen[11] unter einer explorativen Politik, indem wir jeweils in einem zufälligen Anfangszustand starten und dann für 20 Zeitschritte eine Aktion zufällig (mit derselben Wahrscheinlichkeit) auswählen. Mit dieser Menge Trainingsdaten für LSTD führen wir dann Politik-Iteration durch. Nach jeder Iteration messen wir die Performanz der aktuellen Politik durch Simulation, indem wir erstens die mittlere Episodenlänge über einer Menge von 100 zufällig ausgewählten Startzuständen ermitteln (diese Menge wurde nur einmal generiert und ist für alle Lernläufe und Experimente dieselbe), und zweitens die Episodenlänge für den besonderen Startzustand $s = \left(-\frac{\pi}{6}, 0\right)^\mathsf{T}$, der der Ruhelage im Tal entspricht, heranziehen (vgl. Seite 16).

Als Kern verwenden wir wieder (5.33) mit $h^{-1} = 50$, die übrigen Metaparameter sind $\texttt{TOL1} = 0.1, \texttt{TOL2} = 0, \sigma = 0.1, \lambda = 0$ und $\gamma = 0.99$. Abbildung 5.10 (links) zeigt das Ergebnis: bereits nach 5 Iterationen wurde eine Politik erlernt, die eine durchschnittliche Weglänge von 59 Schritten (im Mittel) und 110 Schritten (aus der Ruhelage) erzielt und beinahe die bestmögliche Performanz unter dieser Architektur ist. Alle weiteren Iterationen bringen keine weiteren Verbesserungen mehr. Man sieht vielmehr anhand der Kurve zur Startposition 'Ruhelage', daß approximative Politikverbesserung nicht notwendigerweise auch zu einer besseren Politik führen muß, sondern auch zu Oszillationen im Raum der Politiken führen kann (Bertsekas und Tsitsiklis, 1996).

Online-Lernen: LSPE(λ) vs. Sarsa(λ)

Als zweites Experiment in diesem Szenario untersuchen wir Online-Lernen mit optimistischer Politik-Iteration (OPI) und vergleichen unseren Ansatz LSPE(λ) mit RN gegenüber Sarsa(λ) mit Tilecoding,

[11] Entspricht ungefähr der Anzahl Übergänge, die ein Agent im Fall von Online-Lernen bei rund 200 Lernepisoden beobachtet.



(a) LSTD-API (mit RN) (b) LSPE-OPI (mit RN) (c) Sarsa(λ)-OPI (mit Tilecoding)

Abbildung 5.10: Vergleich der gelernten gierigen Politik für das Szenario Mountain-Car (gesucht ist der kürzeste Weg zum Ziel). Jeder Plot zeigt die Performanz der gelernten Lösung anhand zwei verschiedener Kenngrößen: der mittleren Episodenlänge bei einer Menge von 100 zufälligen (festen) Startzuständen, und der Episondelänge beim Starten aus der Ruhelage. Links: Offline-Lernen mit Off-Policy API, LSTD-APE und RN. Mitte: Online-Lernen mit On-Policy OPI, LSPE-APE und RN. Rechts: Online-Lernen mit Sarsa(λ) und Tilecoding.

dem Standardverfahren in RL (Sutton und Barto, 1998). Die Trainingsdaten für APE (beobachtete Zustandsübergänge und Belohnungen) werden nun direkt aus einer laufenden Simulation mit einem Agenten entnommen; die aktuelle Politik wird nach jedem beobachteten Zustandsübergang ε-gierig verbessert. Jeder Simulationslauf besteht aus mehreren einzelnen Episoden; nur die erste Episode wird mit einem frischen Agenten (initialer Gewichtsvektor $\mathbf{w} = 0$) gestartet. In jedem Schritt einer Episode führt der Agent die bezüglich der aktuellen Wertefunktion beste Aktion (ε-gierig) aus und aktualisiert gleichzeitig den Gewichtsvektor aufgrund des letzten beobachteten Zustandsübergang. Erreicht der Agent das Ziel oder wird die Episode vorzeitig abgebrochen (mehr als 500 Schritte), so wird der Agent in einen neuen zufälligen Startzustand versetzt und eine neue Episode beginnt (der bereits gelernte Gewichtsvektor wird beibehalten).

Für das RN in LSPE verwenden wir dieselben Parameter wie für LSTD zuvor, für das Tilecoding in Sarsa wie auf Seite 16 beschrieben ein $9 \times 9 \times 9$ Gitter. Die übrigen Parameter sind in beiden Fällen gleich: $\varepsilon = 0.01, \lambda = 0.4$ und Lernrate η bzw. $\alpha = 0.2$.

Abbildung 5.10 zeigt das Ergebnis eines typischen Simulationslauf mit LSPE (mitte) und Sarsa (rechts); nach jeder Episode führen wir zur Vergleichbarkeit dieselbe Evaluation wie bei LSTD durch. Geplottet wird die Performanz abhängig von der Anzahl bisher beobachteter Zustandsübergänge; nach den gezeigten 50000 Zustandsübergängen hat der Agent rund 750 Episoden hinter sich gebracht. Die Kurven zeigen deutlich, daß Least-Squares-basierte Politik-Evaluation sehr dateneffizient ist: LSPE (und auch LSTD) konvergieren nach weitaus weniger beobachteten Zustandsübergängen zu einer guten Politik (nach ungefähr 200 Episoden oder 18000 Zustandsübergängen) als das auf stochastischem Gradientenabstieg basierende Sarsa, was dafür rund 5-mal länger braucht (Konvergenz nach rund 1600 Episoden oder 100000 Zustandsübergängen, vgl. Abbildung 1.8, Seite 17). Auf der anderen Seite können RN beim Mountain-Car ihren wirklichen Vorteil gegenüber Tilecoding noch nicht unter Beweis stellen, weil die Dimension der Zustände hier (noch) kein Problem für das gitterbasierte Tilecoding darstellt.

5.2.3 Acrobot

Als nächstes Szenario betrachten wir den einem an einer Stange schwingenden Turner nachempfundenen 'Acrobot', der in Abbildung 5.11 skizziert wird. Der Acrobot ist ein herausforderndes 4-dimensionales Steuerproblem: Ziel ist es, aus der Ruhelage heraus den Roboter invers (auf dem Kopf stehend) zu balancieren. Hier betrachten wir nur das viel einfachere 'Swing-Up' Problem aus Sutton und Barto (1998), bei dem es lediglich das Ziel ist, den Endpunkt (die Füße) in Höhe ℓ_1 über den Aufhängepunkt zu schwingen.

Für RL werden die kontinuierlichen Steuerbefehle auf drei diskrete Aktionen ('vor','zurück','neutral')

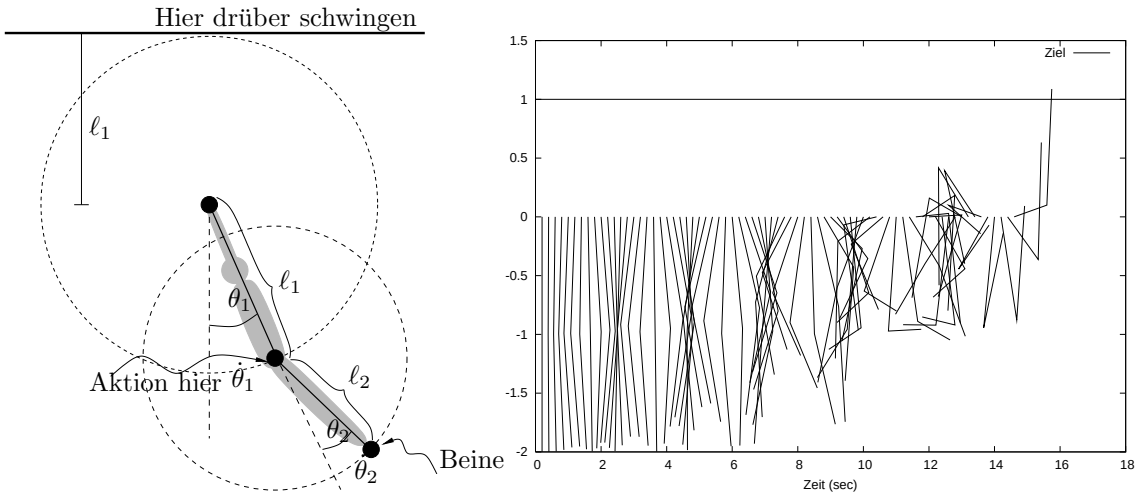


Abbildung 5.11: Links: Der Acrobot. Rechts: Illustration eines erfolgreich gelernten Hochschwingens aus der Ruhelage heraus in 71 Schritten, was $71 * 0.2 = 14.2$ sec dauert. In jedem Zeitschritt wird die Position des Acrobots geplottet.

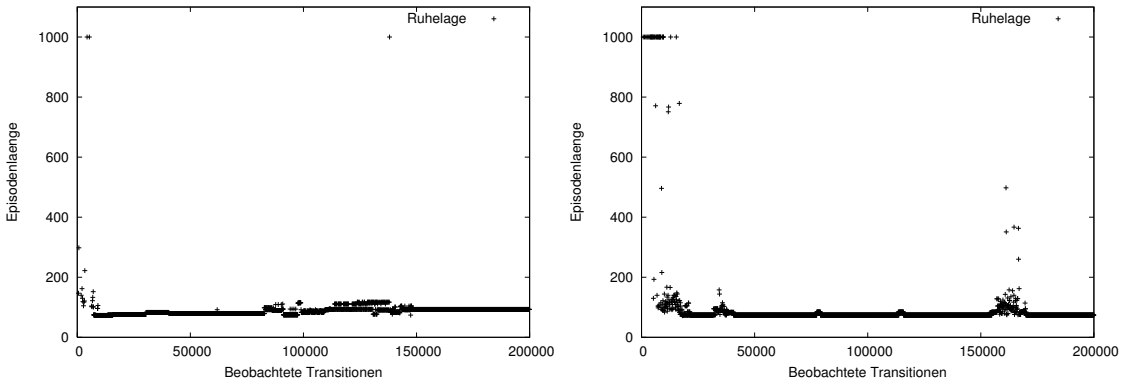
reduziert, wobei das Drehmoment ± 1 Nm beträgt (in der Literatur findet man manchmal den Acrobot auch mit einem stärkeren Motor ± 2 Nm oder ± 3 Nm modelliert, wodurch die zu lösende Aufgabe deutlich einfacher wird). Pro Schritt wird eine Belohnung von $r = -1$ vergeben. Eine Lernepisode startet in der Ruhelage $s_0 = (0, 0, 0, 0)$ und endet entweder erfolgreich mit Erreichen des Ziels oder wird nach 1000 erfolglosen Schritten abgebrochen. Der Zustand des Systems wird durch den Vektor $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ beschrieben. Für die genauen Bewegungsgleichungen und weitere technische Details verweisen wir auf die Beschreibung in Sutton und Barto (1998).

Online-Lernen: LSPE(λ) vs. Sarsa(λ)

Wir wiederholen die generelle Vorgehensweise vom Mountain-Car und testen LSPE(λ) mit RN gegen Sarsa(λ) mit Tilecoding für agentenbasiertes Online-Lernen. Für RN verwenden wir die folgenden Parameter: $h^{-1} = 2, \texttt{TOL1} = 0.1, \texttt{TOL2} = 0, \sigma = 0.1$. Für Tilecoding können wir den 4-dimensionalen Zustandsraum nicht mehr vollständig uniform in höherer Auflösung diskretisieren, wir verwenden daher die in Sutton und Barto (1998) beschriebene Teilzerlegung (siehe auch Abbildung 1.6). Die Winkel θ_1, θ_2 werden dabei zunächst in 7 und die Geschwindigkeiten $\dot{\theta}_1, \dot{\theta}_2$ in jeweils 6 Abschnitte diskretisiert. Daraus werden folgende, den Zustandsraum überdeckende Gitterschichten (Tilings) gebildet:

- 12 Tilings für alle 4 Dimensionen des Zustands
- je 3 Tilings für alle Dreier-Kombinationen der Dimensionen (4 solcher Kombinationen möglich)
- je 2 Tilings für alle Zweier-Kombinationen der Dimensionen (6 solcher Kombinationen möglich)
- je 3 Tilings für jede einzelne Dimension des Zustands (4 solcher Kombinationen möglich)

Die übrigen Lernparameter sind $\lambda = 0.9, \varepsilon = 0, \gamma = 1.0, \eta = 1.0, \alpha = 0.2$. Abbildung 5.11 (rechts) zeigt die Trajektorie eines erfolgreich gelernten Aufschwing-Versuchs aus der Ruhelage heraus und Abbildung 5.12 zeigt den Lernerfolg bei den beiden Vorgehensweisen LSPE mit RN und Sarsa mit Tilecoding; verglichen wird hier nach jeder Episode die Anzahl der notwendigen Schritte, die die jeweils gierige Politik zum Erreichen des Ziels benötigt. Interessant, wenn auch nicht besonders aussagekräftig, ist ein Vergleich der Anzahl der Basisfunktionen in der Parametrisierung der Wertefunktion: bei Tilecoding sind es ungefähr 20000, bei unserem RN lediglich 300.



(a) LSPE-OPI (mit RN) (b) Sarsa(λ)-OPI (mit Tilecoding)

Abbildung 5.12: Vergleich der gelernten Politik für den Acrobot (Ziel ist Minimierung der Episodenlänge). Nach jeder einzelnen Lernepisode ermitteln wir die Performanz der bezüglich der aktuellen Approximation gierigen Politik, d.h. die Anzahl der notwendigen Schritte bis zum erfolgreichen Aufschwingen (oder vorzeitiger Abbruch nach 1000 erfolglosen Schritten). Links: Online-Lernen mit On-Policy OPI, LSPE-APE und RN. Rechts: Online-Lernen mit Sarsa(λ) und Tilecoding.

Bei beiden Methoden lernt der Agent nach einiger Zeit erfolgreich die Aufgabe zu lösen, und benötigt dafür eine vergleichbare Anzahl beobachteter Zustandsübergänge. Anders als beim Mountain-Car hat LSPE hier gegenüber Sarsa keinen merklichen Vorsprung mehr (siehe Diskussion zur Exploration am Ende des Kapitels). Nach hinreichend vielen beobachteten Daten beträgt die Performanz am Ende bei beiden Methoden 69-85 Schritte zum Ziel. Zur Erläuterung dieser Zahlen ist anzumerken, daß die Aufgabe des Aufschwingens es erforderlich macht, eine lange Kette von 'richtigen' Aktionen auszuwählen; eine einzige 'falsche' Aktion kurz vor Erreichen des Ziels kann alles wieder zunichte machen. Weil demnach kleine Änderungen an der Politik große Änderungen an der Episodenlänge bewirken können, sind die angegebenen Resultate mit hoher Varianz versehen und selbst bei einem fertig-gelernten Agenten kann es später noch sporadisch zu einem kurzzeitigen Leistungseinbruch kommen (deutlich wird das bei den plötzlichen Performanz-Sprüngen auch nach fortgeschrittenem Lernen in Abbildung 5.12).

Off-Policy API mit LSTD

Off-Policy API mit einer festen Menge von Trainingsdaten analog zu Lagoudakis und Parr (2003) erweist sich beim Acrobot als schwieriger. Das Problem ist, daß wir beim Folgen einer rein explorativen Politik den Zustandsraum nicht repräsentativ abtasten und möglicherweise keine Daten in der Nähe der Terminalzustände erzeugen (eine rein explorative Politik schwingt beim Acrobot nur um die Ruhelage herum, erzielt aber keinen Fortschritt). Zu einem gewissen Grad bestand dieses Problem auch beim Mountain-Car, hier tritt es aber wegen der Komplexität der Aufgabe noch viel deutlicher zutage. Wir behelfen uns, indem wir die Trainingsdaten unter einer zuvor mit LSPE gelernten Politik erzeugen (mit großzügiger Explorationsrate). Diese sind dann in den 'interessanten' Regionen des Zustandsraums konzentriert und wir stellen sicher, daß die Daten auch mindestens einen Übergang in einen Terminalzustand enthalten. Ohne die Plots im einzelnen anzugeben, wollen wir hier nur das Ergebnis erwähnen: LSTD mit RN konnte zwar erfolgreich lernen, die jeweils gierige Politik in den einzelnen Iterationsschritten waren aber nicht stabil und insgesamt schlechter als die zuvor durch Online-Lernen gefundenen.

5.3 Komplexe und realistische Anwendungsbeispiele

Zum Abschluß betrachten wir noch zwei komplexere Szenarien, die mit zum Grenzbereich dessen gehören, was mit den gegenwärtigen Techniken in RL noch möglich ist:

5.3.1 RoboCup-Keepaway

Das *Keepaway*-Problem ist ein Multiagenten-Lernproblem im Kontext von Roboterfußball und der Simulationsliga in RoboCup[12]. Gegeben sind zwei Teams fußballspielender autonomer Agenten: das Team der 3 Keeper und das der 2 Taker. Die Aufgabe der ballführenden Keeper ist es, durch geschicktes Zupassen des Balls untereinander ihre zahlenmäßige Überlegenheit auszunutzen und so lange wie möglich in Ballbesitz zu bleiben. Die Aufgabe der gegnerischen Taker ist es, das zu verhindern und die Kontrolle über den Ball zu erlangen.

Keepaway basiert auf dem offiziellen Soccerserver (Noda et al., 1998), der auch für die Ausrichtung der internationalen Wettbewerbe zu Einsatz kommt und wurde von Stone et al. (2005) als Benchmarkproblem speziell für RL formuliert[13]. Ziel ist es dabei, eine komplette Steuerung für die Keeper zu *lernen*, die den Ballbesitz maximiert, wenn diese gegen ein Team mit stationärer (fest-kodierter) Verhaltenssteuerung antritt. Dank der folgenden Umstände gestaltet sich das als eine ziemliche Herausforderung:

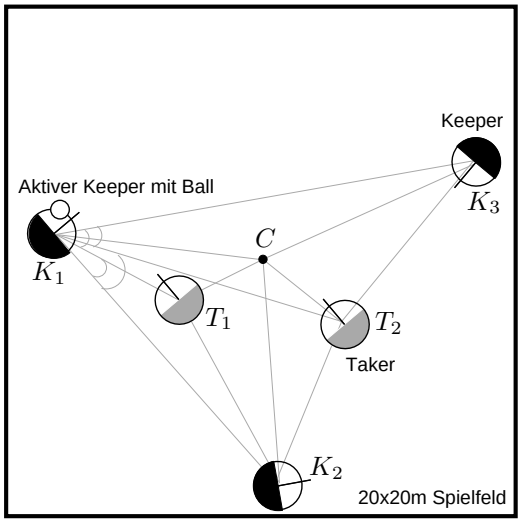
- **Dimensionalität:** die beobachteten Zustände bestehen wie in Abbildung 5.13 skizziert, aus den Abständen und Winkeln der Agenten untereinander: im Fall von 3 gegen 2 ein 13-dimensionaler Vektor. Die im Gegensatz zu den zuvor betrachteten Beispielen relativ hohe Dimension erschwert den Einsatz gitterbasierter Funktionsapproximation wie Tilecoding.
- **Stochastizität:** ein Agent kann seine Umwelt nicht exakt wahrnehmen und auch nicht präzise mit ihr interagieren: sowohl die beobachteten Zustandsvektoren als auch die ausgeführten Aktionen werden mit einem zufälligen Rauschen gestört. Die einzelnen Agenten sind zudem als autonome Entitäten modelliert und lernen unabhängig voneinander nur aus den Erfahrungen, die sie selbst tatsächlich auch gemacht haben. Erfolg ist aber ein Produkt der Teamleistung, woran alle befreundeten Agenten (und nicht nur der lernende) beteiligt sind. Aufgrund des Rauschens sind die Zustandsübergänge stochastisch und aufgrund der Multiagenten-Natur ist die Dynamik des Systems nichtstationär. Beides hat u.a. zur Folge, daß wir nur mit modellfreien Methoden, der Q-Funktion, arbeiten können.
- **Effizienz:** der Soccerserver ist eine Echtzeitanwendung, bei der die Agenten in Zeitintervallen von je 100 msec (Echtzeit) mit der Umwelt interagieren. Soll das Agentenverhalten aus der laufenden Simulation heraus gelernt werden, dann können wir das nur mit sehr effizienten Methoden schaffen (was Rechenzeit und Speicherbedarf angeht). Bedingt durch die aufwendige Simulation mit dem Soccerserver ist es zudem auch recht 'teuer', eine hinreichend große Anzahl Zustandsübergänge zu erzeugen: 50.000 Übergänge entsprechen ungefähr der Dauer eines ganzen Tags (simuliert). Deswegen spielt hier auch die Dateneffizienz des verwendeten RL-Verfahrens eine wichtige Rolle, denn es macht durchaus einen Unterschied, ob performantes Verhalten bereits nach 2 Stunden oder erst nach einem ganzen Tag (24 Stunden) erworben wird.

Modellierung als RL-Problem

Für RL stellt sich *Keepaway* folgendermaßen dar: nur der jeweils ballführende Keeper ist aktiv und hat bei jedem Aufruf die Wahl zwischen den drei alternativen Aktionen `Ball-halten, Paß-zu-Mitspieler-1, Paß-zu-Mitspieler-2`. Bezogen auf den Soccerserver sind das nur Meta-Aktionen, die automatisch in elementare Steuerbefehle umgesetzt werden. Die beiden übrigen Keeper folgen einer automatischen (fest-kodierten) Steuerung, um sich möglichst günstig für den Empfang eines Passes zu positionieren. Die Taker folgen ebenfalls einer automatischen Steuerung um den Ball zu erobern. Die Ausführung der Meta-Aktionen wird i.a. einen variablen Zeitraum in Anspruch nehmen. Als unmittelbare Belohnung dient uns die Zeit (in 100 ms), die zwischen den einzelnen Aufrufen des Agenten verstreicht. Ziel der lernenden

[12] RoboCup ist eine internationale Initiative mit dem ohne Frage sehr ehrgeizigen Ziel, in nicht allzu ferner Zukunft reale und humanoide Roboter zu konstruieren, die in der Lage sind, motorisch anspruchsvolle Tätigkeiten wie Fußballspielen perfekt auszuführen. Ein Ziel, das sehr weit über das hinausgeht, was gegenwärtig technisch in Robotik und KI machbar ist.

[13] Der Quellcode ist frei verfügbar unter: `http://www.cs.utexas.edu/users/AustinVilla/sim/keepaway/`



	Zusammensetzung des Zustandsvektors
1.	$dist(K_1, C)$
2.	$dist(K_2, C)$
3.	$dist(K_3, C)$
4.	$dist(T_1, C)$
5.	$dist(T_2, C)$
6.	$dist(K_1, K_2)$
7.	$dist(K_1, K_3)$
8.	$dist(K_1, T_1)$
9.	$dist(K_1, T_2)$
10.	$\min\{dist(K_2, T_1), dist(K_2, T_2)\}$
11.	$\min\{dist(K_3, T_1), dist(K_3, T_2)\}$
12.	$\min\{ang(K_2, K_1, T_1), ang(K_2, K_1, T_2)\}$
13.	$\min\{ang(K_3, K_1, T_1), ang(K_3, K_1, T_2)\}$

Abbildung 5.13: Illustration von *Keepaway* und Aufbau der Zustandsvektoren nach Stone et al. (2005).

Agenten insgesamt ist es, möglichst lange im Ballbesitz zu bleiben und zu verhindern, daß der Ball das 20×20 m große Spielfeld verläßt oder die Taker erfolgreich Kontrolle über den Ball erlangen. Für weitere Details siehe Stone et al. (2005).

Experiment

Als RL-Verfahren untersuchen wir, wie im Abschnitt zuvor, zwei verschiedene Ansätze: erstens, unseren eigenen Ansatz LSTD mit RN und off-policy API wie in Lagoudakis und Parr (2003) und zweitens, Sarsa(λ) mit Tilecoding. Sarsa(λ) wurde bereits mit Erfolg in dem Originalartikel von Stone et al. (2005) verwendet; wie haben ihn lediglich zu Vergleichszwecken mit den dort beschriebenen Parametereinstellungen re-implementiert[14]. Das Hauptproblem von *Keepaway*, die hohe Dimensionalität des Zustandsraums, wird dabei von den Autoren dadurch umgangen, daß sie mit Tilecoding keine vollständige Diskretisierung des Produktsraums (das kartesische Produkt aller 13 Dimensionen) vornehmen, sondern jede Dimension nur einzeln und unabhängig mit einem Gitter überdecken. Das ist eine nicht unerhebliche Vereinfachung, wird dadurch doch die Interaktion und Abhängigkeit der Zustandsvariablen untereinander ignoriert. Trotz dieser Einschränkung konnten die Autoren eine sehr performante Politik erlernen.

Für unser Regularisierungsnetzwerk stellt die Dimension der Eingaben natürlich weniger ein Problem dar; wir werden demonstrieren, daß sich auch aus den reinen Zustandsvektoren (ohne Vereinfachung oder problemspezifische Vorverarbeitung) erfolgreich ein performantes Verhalten erlernen läßt. Wir wiederholen unser generelles Vorgehen aus Abschnitt 5.2, verändern aber die Art und Weise, wie Trainingsdaten erzeugt werden. Hatten wir dort API als Offline-Verfahren angewendet, so werden wir es hier wie ein Online-Verfahren einsetzen, indem der Agent einerseits mit der Umwelt für einen längeren Zeitraum mit einer festen Politik interagiert und neue Erfahrungen sammelt, und zur gleichen Zeit die APE mit den bereits gespeicherten Daten durchführt. Zur Initialisierung starten wir jeden Lernlauf mit einer festen Menge von 100 Übergängen, nehmen dann aber aus der laufenden Simulation heraus fortwährend die neubeobachteten Zustandsübergänge hinzu. Aufgrund der Echtzeitanforderung können wir die APE nicht in einem Schritt komplett durchführen. Stattdessen nutzen wir das zur Verfügung stehende Zeitfenster (100 msec), um die Trainingsmenge in kleinen Blöcken von je 25 Daten abzuarbeiten, was mit Hilfe des speziellen Online-Algorithmus aus Abbildung 5.2 geschieht (den wir für genau diesen Einsatzzweck entwickelt hatten). Ist die Trainingsmenge komplett durchlaufen worden und die Evaluation der aktuellen Politik abgeschlossen, so führen den Schritt Politikverbesserung durch gierige Politikableitung aus und starten einen erneuten Durchlauf APE in den Trainingsdaten mit der neuen Politik.

[14] Die von uns mit Sarsa erzeugten Resultate stimmen quantitativ mit den nach der Originalarbeit zu erwartenden überein.

Resultate

Abbildung 5.14 zeigt die mit unserem Ansatz erzielten Ergebnisse mit einem Kern der Form (5.33) und den Parametern $h^{-1} = 0.2, \sigma = 0.1, \texttt{TOL1} = 0.1, \lambda = 0, \gamma = 0.99, \varepsilon = 0.01$. Jeder Plot zeigt 5 Kurven, die das Ergebnis von 5 unabhängigen Lernläufen sind und bei denen wir jeweils mit einem frisch-initialisierten Team von Keepern starten (die initiale Politik wählt Aktionen zufällig aus). Jeder Lernlauf besteht, wie bei den zuvor betrachteten Szenarien auch, aus einzelnen Episoden, wobei jede Episode mit einer zufälligen Anfangspositionierung startet und bis zum Ballverlust der Keeper andauert. Je nach Lernfortschritt ist eine Episode durchschnittlich 2-40 sec lang (aufgrund der Stochastizität ist die Varianz sehr hoch). Jede Kurve plottet die Performanz (die über einen größeren Zeitraum gemittelte Episodenlänge) gegen die vergangene Zeit, die die Agenten bisher mit der Simulation interagiert und gelernt haben. Die horizontalen Linien markieren zum Vergleich die Performanz von zwei besonderen Benchmarkstrategien: einmal die Strategie 'Random', die in jedem Schritt eine zufällige Aktion auswählt, und einmal die Strategie 'Handcoded', die der aufwendig von Hand konstruierten und optimierten Steuerung des CMUnited-99 Agenten entspricht und nicht gelernt wurde (Stone et al., 2005).

In den verschiedenen Abbildungen (a)-(c) untersuchen wir die Effizienz unseres Basisselektionskriteriums und gehen von der unüberwachten Basisselektion ($\texttt{TOL2} = 0$) zur überwachten ($\texttt{TOL2} = 0.001$) über[15]. In Abbildung (d) plotten wir zum Vergleich auch das Ergebnis von Lernläufen mit Sarsa(λ). Alle Ergebnisse zeigen allgemein, daß mit RL eine Steuerung automatisch gelernt werden kann, die mindestens ebenso gut ist, wie eine aufwendig von Hand optimierte. Betrachten wir die Plots im einzelnen, so stellen wir fest, daß unser Ansatz LSTD mit RN den Ansatz Sarsa(λ) mit Tilecoding um rund 25% übetrifft: bei Sarsa ist die beste Performanz ungefähr 15 sec, mit LSTD kommen wir auf rund 20 sec. Es ist aber nicht nur die Qualität der gelernten Steuerung, worin wir Sarsa übertreffen. Noch eindrucksvoller wird der Leistungsunterschied, wenn wir die Zeitskala betrachten: nach nur 2 Stunden Lernen übertrifft die mit unserem Ansatz gelernte Steuerung bereits die von Hand optimierte; Sarsa benötigt dazu immerhin 15-20 Stunden. Ein dritter Punkt ist die Effizienz unserer überwachten Basisselektion gegenüber der unüberwachten: bei $\texttt{TOL2} = 0.01$ ist die erzielte Performanz genauso hoch wie für $\texttt{TOL2} = 0$, es werden aber wesentlich weniger Basisfunktionen benötigt: 700 Basisfunktionen bei $\texttt{TOL2} = 0.01$ gegen 1400 Basisfunktionen bei $\texttt{TOL2} = 0$. Eine Halbierung der notwendigen Basisfunktionen bedeutet wegen des quadratischen Aufwands eine Einsparung von 75% Rechenleistung.

Neben dem Ansatz LSTD mit RN untersuchten wir alternativ auch LSPE mit RN für *Keepaway*. Die detaillierten Resultate sollen an dieser Stelle nicht wiedergegeben werden, siehe dazu Jung und Polani (2007). Dort hatte sich aber gezeigt, daß die Ergebnisse von LSPE (mit identischen Parametern für das RN) zwar gleichauf mit Sarsa, aber weniger gut als die durch LSTD erzielten waren. Es ist zu vermuten, daß der Grund dafür in den unterschiedlichen Architekturen liegt (API gegenüber OPI), was sich auch mit unseren ähnlichen Beobachtungen in den anderen Experimenten deckt. Genauer haben wir diesen Punkt nicht untersucht (siehe auch Diskussion am Ende des Kapitels).

5.3.2 Oktopus-Arm

Als zweites komplexes Anwendungbeispiel betrachten wir die Modellierung und Steuerung eines simulierten Oktopus-Tentakels nach Engel et al. (2005b). Die Tentakel eines realen Oktopus sind ein äußerst flexibles Organ; anders als die Gliedmaßen von skelett-basierten Kreaturen (Vertebraten) besitzen Tentakel kein starres Skelett sondern können sich dehnen und kontrahieren und an nahezu jedem Punkt entlang jeder Achse beliebig rotieren. Für die Robotik könnte das im Zusammenhang mit der Entwicklung künstlicher Muskeln basierend auf elektroaktiven Polymeren interessante und vielfältige Anknüpfungspunkte bieten. Das Problem ist es aber nicht allein, ein solches Ding physikalisch zu bauen, sondern dieses auch zu kontrollieren, wofür die klassischen Ansätze in der Robotik und Steuerungstheorie nicht auszureichen scheinen.

[15] Weil wir bei LSTD keinen Ausdruck für die Fehlerreduktion eines Basisfunktionskandidaten haben, verwenden wir zu diesem Zweck den Ansatz BRM. Der algorithmische Aufwand steigt dadurch zunächst um knapp ein Drittel an. Insgesamt kommen wir aber dann am Ende mit deutlich weniger Basisfunktionen aus.

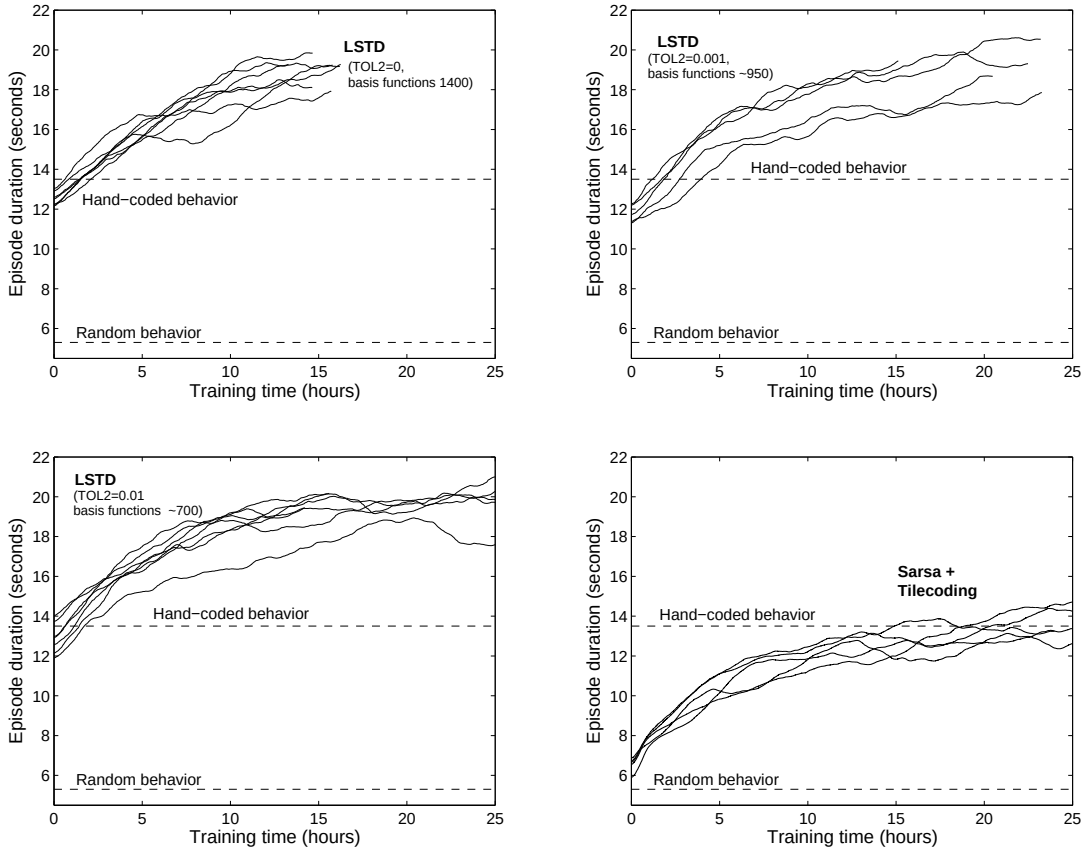


Abbildung 5.14: Resultate bei 3vs2 *Keepaway*. Von links nach rechts: Lernkurven für unseren Ansatz LSTD mit RN (`TOL2`=0), LSTD mit RN (`TOL2`=0.001), und LSTD mit RN (`TOL2`=0.01). Zum Vergleich zeigen wir auch die Ergebnisse von Sarsa(λ) und Tilecoding, korrespondierend zu Stone et al. (2005). Geplottet wird jeweils die durchschnittliche Dauer, die die Keeper in Ballbesitz bleiben konnten (Performanz des gelernten Verhaltens) gegen die vergangene Zeit in der Simulation (proportional zur Anzahl beobachteter Zustandsübergänge). Nach ungefähr 15 Stunden trat i.a. keine weitere Verbesserung der Performanz mehr ein. Das Ausführen einer Aktion kann unterschiedlich viel Simulationstakte in Anspruch nehmen, so daß den 15 Stunden Lernen ca. 30.000 beobachteten Zustandsübergängen (für jeden Keeper-Agenten) entsprechen. Zu beachten ist, daß alle Kurven stark geglättet wurden um den Anstieg deutlich zu machen (die Rohdaten sind mit hoher Varianz behaftet; eine Episode variiert typischerweise zwischen 3 und 40 sec). Die Ausgangsperformanz beim Start liegt in allen Fällen bei ca. 5 sec (wie Random). Allerdings lernt LSTD mit RN bedeutend schneller hochperformantes Verhalten als Sarsa mit Tilecoding, so daß die entsprechenden Kurven nach Mittelung bei 12 sec starten.

Modellierung als RL-Problem

In Engel et al. (2005b) wird das Problem der Steuerung eines Oktopus-Tentakels als Reinforcement Learning Problem formuliert[16]: der Tentakel wird in N Segmente diskretisiert und mittels Punktmassen verbunden durch gedämpfte Federn modelliert. Bei N Segmenten ergeben sich $2(N+1)$ Punktmassen, jede einzelne charakterisiert durch Position und Geschwindigkeit. Zur Vereinfachung wird nur ein 2-dimensionaler Tentakel betrachtet; nichtsdestotrotz wird in diesem Fall der komplette Zustand eines in nur 7 Segmente unterteilten Tentakels durch $4 \times 2 \times (7+1) = 64$-dimensionale Vektoren beschrieben. Das Tentakel wird kontrolliert durch Kontraktion der einzelnen Muskelstränge (eine jeweils reellwertige Steuergröße); bei 7 Segmenten hat ein Steuervektor die Dimension 26. Abbildung 5.15 zeigt diese Modellierung graphisch.

Die Steuerung des Tentakels ist ein deterministisches, episodisches Lernproblem: bei dem von Engel et al. (2005b) betrachteten Szenario geht es darum zu lernen, mit dem Tentakel einen vorgegebenen Zielpunkt im Raum zu berühren (siehe Abbildung 5.16). Anstatt kontinuierlicher Aktionensvektoren werden eine Reihe von Aktivierungsprofilen (26-dimensionale Steuervektoren) vordefiniert, wodurch sich das Problem wieder auf eine kleine Anzahl diskreter Aktionen reduziert (7 Stück). Nichtsdestrotrotz ist allein aufgrund der sehr hohen Dimensionalität des Zustandsraums das Problem ein herausforderndes und kann (ohne problemspezifische Vorverarbeitung) unmöglich mit traditionellen gitterbasierten Funktionsapproximatoren gelöst werden.

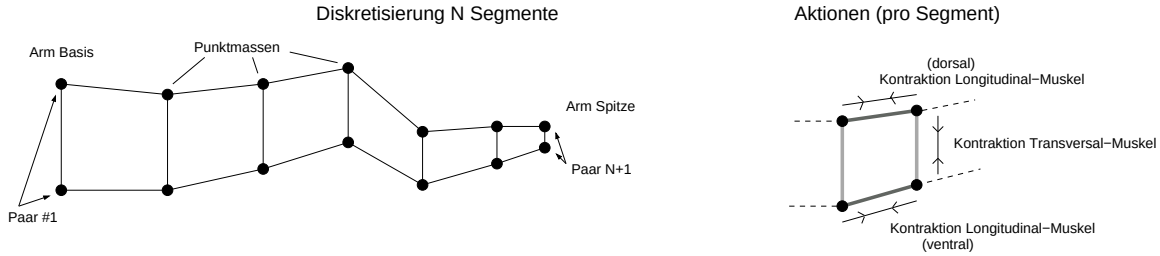


Abbildung 5.15: In N Partitionen diskretisierter 2-dimensionaler Oktopus-Arm. Abbildung nach Engel et al. (2005b).

Resultate

Hier wenden wir unseren Ansatz LSPE mit RN und OPI für Online-Lernen an, wobei wir anders als in Engel et al. (2005b) kein Modell für die (deterministischen) Zustandsübergänge verwenden, sondern stattdessen die modellfreie Q-Funktion lernen. Wie zuvor führen wir unabhängige Lernläufe durch, bei der jeder Lernlauf mit einem frisch-initialisierten Agenten startet und aus der Abfolge einzelner Episoden besteht. Jede Episode startet aus einer zufälligen Anfangspositionierung des Tentakels (diese Startzustände sind zufällig aber fest und werden aus einer Datei eingelesen, so daß sie für alle Lernläufe identisch sind). Pro Schritt wird eine Belohnung von $r = -1$ vergeben; die Belohnung beinhaltet somit keine zielführenden Zusatzinformationen. Eine Episode endet, wenn das Tentakel erfolgreich das Ziel berührt ($r = +1000$), oder wird nach 1000 erfolglosen Schritten vorzeitig abgebrochen.

Die Parameter sind $h^{-1} = 0.2, \sigma = 0.1, \texttt{TOL1} = 0.1, \lambda = 0.6, \gamma = 0.999, \varepsilon = 0.05, \eta = 0.5$. Das Ergebnis einiger Lernläufe wird in Abbildung 5.16 gezeigt und Abbildung 5.17 stellt dazu die Trajektorie eines erfolgreichen Greifvorgangs graphisch dar. Aufgrund der relativ langen Laufzeit des Simulators (ein Lernlauf mit 200 Episoden benötigt ungefähr 2 Stunden, wobei die meiste Rechenzeit von dem in Java implementierten Simulator beansprucht wird) konnten insgesamt nur wenige Parametereinstellungen und Lernläufe durchgeführt werden. Die Plots zeigen, daß unser Ansatz im Mittel nach 20-50 Episoden erfolgreich gelernt hat, das Ziel aus jeder beliebigen Anfangsposition heraus mit 100% Erfolgsrate zu berühren. Das ist schon allein deswegen bemerkenswert, weil hier der Agent trotz der hohen Dimensionalität und

[16] Wir basieren unsere Experimente auf dem begleitend zur ICML 2006 stattgefundenen Workshop 'Reinforcement Learning benchmarking' verfügbar gemachten Simulator: `http://www.cs.mcgill.ca/dprecup/workshops/ICML06/octopus.html`

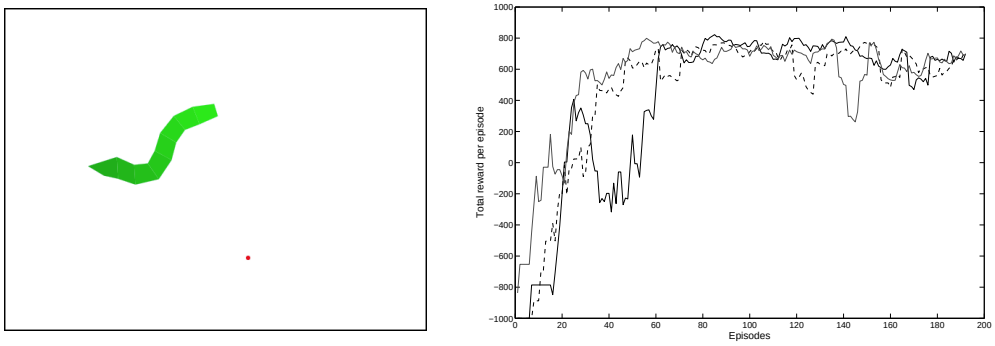
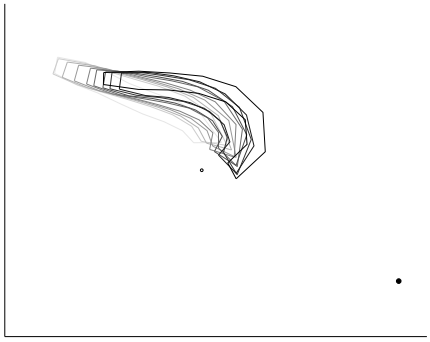
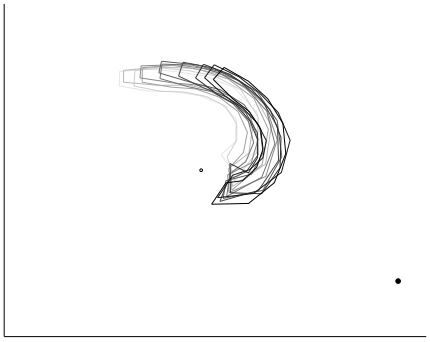


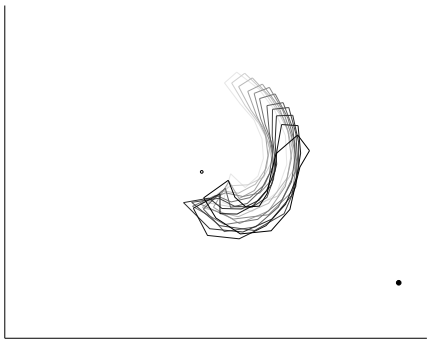
Abbildung 5.16: Links: der Tentakel muß sich strecken und rotieren, um den Zielpunkt zu berühren (Screenshot des Simulators). Rechts: Online-Lernen mit unserem Ansatz LSPE und RN. Gezeigt wird das Resultat für die ersten 200 Episoden, jede Kurve ist ein unabhängiger Lernlauf. Die Ergebnisse sind geglättet. Eine negative Gesamtbelohnung bedeutet, daß eine Episode erfolglos nach 1000 Schritten abgebrochen wurde (Gesamtbelohnung -1000). Eine positive Gesamtbelohnung bedeutet eine erfolgreich abgeschlossene Episode und mißt in diesem Fall die Anzahl der dafür notwendigen Schritte.



Schritte 1-11



Schritte 12-22



Schritte 23-33

Schritte 34-46

Abbildung 5.17: Illustration eines gelernten, erfolgreichen Greifvorgangs mit dem Tentakel in 46 Schritten: Ziel ist, den Punkt unten rechts zu berühren. Der Tentakel ist an dem Punkt in der Mitte befestigt (die Position des erste Segments wurde vom Simulator nicht zurückgeliefert). Der letzte Schritt (erfolgreiches Berühren) fehlt in der rechten unteren Abbildung, weil der letzte Zustand einer Episode vom Simulator ebenfalls nicht zurückgeliefert wurde.

Komplexität der Aufgabe innerhalb sehr kurzer Zeit nur durch Ausprobieren eine erfolgreiche Steuerung lernen kann. Zweitens, haben wir in einer weiteren Reihe von Lernläufen auch wieder die Effektivität der überwachten Basisselektion untersucht: bei unüberwachter Basisselektion (`TOL2` = 0) benötigen wir ungefähr 2000 Basisfunktionen, bei überwachter Basisselektion (`TOL2` = 0.1) reichen uns lediglich 1200 Basisfunktionen um die gleiche Leistung zu erzielen.

Eine präzisere quantitative Evaluation der gelernten Steuerung (Untersuchung der Optimalität bezüglich der benötigten Schritte zum Ziel – diejenige Quantität, die minimiert werden soll) konnten wir mangels Vergleichsdaten in der Literatur[17] nicht vornehmen. Der Oktopus-Benchmark ist aber vergleichsweise neu: möglicherweise werden erst in der Zukunft weitere Ergebnisse von anderen Arbeitsgruppen veröffentlicht.

5.4 Diskussion

In diesem Kapitel haben wir die Anwendung von nichtparametrisierten Regularisierungsnetzen zur Funktionsapproximation im Kontext von Reinforcement Lernen beschrieben und einen Lernalgorithmus für approximative Politik-Evaluation angegeben.

5.4.1 Zusammenfassung der Ergebnisse

Die von uns durchgeführten Experimente zeigen, daß Reinforcement Lernen mit Regularisierungsnetzen sowohl erfolgreich als auch effizient möglich ist. Bei niedrigdimensionalen „Spielzeug“-problemen ist unser Ansatz von der Performanz her vergleichbar mit dem Standardansatz Sarsa mit Tilecoding als gitterbasiertem Funktionsapproximator (Sutton und Barto, 1998), erzielt aber unter Umständen eine deutlich schnellere Konvergenz. Seine wirkliche Stärke spielt unser Ansatz dann aber bei den hochdimensionalen Problemen aus: beim Keepaway-Problem konnten wir eine um 25% bessere Steuerung lernen, benötigten dazu aber gleichzeitig weniger als ein Fünftel der notwendigen Trainingsdaten. Da die Trainingsdaten aufwendig aus der Simulation von den Agenten erzeugt werden müssen, ist das zumindest erwähnenswert; beim Einsatz auf reale Roboter kann das aber entscheidend sein. Zudem konnte bei Keepaway das gitterbasierte Tilecoding nur durch eine stark vereinfachende Zusatzannahme (die einzelnen Komponenten des Zustandsvektors wirken sich unabhängig voneinander auf die Wertefunktion aus) realisiert werden; bei unserem kernbasierten Ansatz war so etwas nicht nötig, wir können direkt und unabhängig von der Dimension der Zustandsvektoren lernen. Die Steuerung des Oktopus-Tentakels nach Engel et al. (2005b) ist gegenwärtig das nach unserem Wissen schwierigste (bezogen auf die Dimensionalität des Zustands-Aktionsraums) frei verfügbare RL-Benchmarkproblem: während andere in der Literatur typische Benchmarks wie Mountain-Car oder Acrobot mit einem zwei oder vierdimensionalen Zustandsraum ringen, haben wir es beim Oktopus mit 60-100 dimensionalen Zustandsvektoren zu tun (bei einer feineren Diskretisierung des Oktopus entsprechend mehr).[18] Dennoch ließ sich auch hier mit unserem Ansatz mit überraschend wenigen Lernepisoden eine erfolgreiche Steuerung für das (zugegebenermaßen nicht überwältigend schwierige) Greifen-Problem lernen.

Ein noch näher zu untersuchendes Problem sind die Auswirkungen der unterschiedlichen Politikiterations-Architekturen Actor/Critic (mit Off-Policy API oder On-Policy API) einerseits und optimistischer Politikiteration OPI andererseits. Bei unseren Experimenten konnten wir beobachten, daß bei gleicher Einstellung der Meta-paramenter für das RN der Ansatz Actor/Critic (mit Off-Policy API) besonders gut bei solchen Lernaufgaben funktioniert hat, die – salopp gesagt – implizit darauf abzielen, die Episodenlänge zu maximieren (beispielsweise Pole-balancing oder Keepaway). Andererseits erwies sich Actor/Critic als weniger erfolgreich bei Kürzeste-Wege Lernaufgaben wie Acrobot oder Oktopus, die implizit darauf abzielen, die Episodenlänge zu minimieren. Hier stellte sich OPI als klar bessere Alternative heraus. Wir vermuten, daß dieser Punkt mit der „Kontrollierbarkeit“ des Systems zusammenhängt, wobei wir unter Kontrollierbarkeit verstehen, inwiefern das Eintreten eines Folgezustands das Resultat von der Aktion des Agenten oder doch eher von der Dynamik des Systems ist. Bei einer niedrigen Kontrollierbarkeit wird

[17] Auch bei Engel et al. (2005b) wird das Kriterium Erfolg/oder nicht zur Evaluation verwendet.

[18] Diese Größenordnung würde ungefähr auch dem entsprechen, was bei der Steuerung eines (humanoiden) Roboters oder anderen komplexen Aufgabenstellungen aus der realen Welt anfällt.

die beobachtete Trajektorie mehr vom System als vom Agieren des Agenten bestimmt. Bei einer hohen Kontrollierbarkeit ist hauptsächlich der Agent selbst dafür verantwortlich: hier kann das Folgen einer ungünstigen Politik dazu führen, daß der Agent in einem kleinen Teil des Zustandsraum steckenbleibt und keinen Fortschritt erzielt. OPI kann in diesem Fall sofort und punktweise die ausgeführte Politik abändern.

Möglicherweise steht dieses Problem aber auch in Verbindung mit einem anderen von uns nur unzureichend bearbeitetem Punkt: der Exploration. Im Grunde genommen basieren die RL-Verfahren auf Trial & Error; wenn das Lernziel im Erreichen eines bestimmten Zustands (Stochastische Kürzeste-Wege) besteht, muß der Agent also wenigstens einmal ,,blindlings“ in das Ziel stolpern. Bei Sarsa mit Tilecoding wird Exploration in erster Linie durch eine optimistische Anfangsinitialisierung erreicht: die Gewichte der Approximation werden anfänglich mit Null initialisiert. Sind dann die Belohnungen alle negativ (z.B. $r = -1$), so wählt der Agent als gierige Aktion immer eine solche, die er im gegenwärtigen Zustand noch nicht ausprobiert hat. Bei den RN ist das in dieser Form nicht möglich, weil die Basisfunktionen i.a. keinen kompakten Träger haben und die Gewichte daher nicht nur lokal verändert werden. In unseren Experimenten hatten wir daher versucht, die notwendige Exploration durch einfache ε-greedy Aktionswahl zu gewährleisten. Hier wäre eine etwas zielführendere Exploration angebrachter gewesen, z.B. in dem man sicherstellt, daß vor allem die bislang noch nicht ausgeführten Handlungsoptionen ausprobiert werden. Ein Ansatz, mit dem das mit RN möglich ist, wäre es, die Varianz der geschätzten Werte zu berücksichtigen (im Zusammenhang mit Bayes'scher Regression ist die RN-Lösung auch der Erwartungswert der Aposteriori-Verteilung über den Gewichten): die Varianz kann dabei als ,,Unsicherheit“ eines von der Approximation vorhergesagten Wertes interpretiert werden, und ist dann hoch, wenn der Vorhersagepunkt weit von den Trainingsdaten entfernt ist. In Engel et al. (2005a) wird dies im Zusammenhang mit RL und GP-Regression vorgeschlagen, bei der Beschreibung des Oktopus-Tentakels in Engel et al. (2005b) aber praktisch nicht verwendet. Eine Schwierigkeit ist, daß bei GP-Regression zusammen mit der SR-Approximation diese Varianz normalerweise auch dann gegen Null geht, wenn der Vorhersagepunkt von den Basiselementen weit entfernt ist – genau das Gegenteil dessen, wozu wir die Varianz praktisch benötigen. Daher wird in Rasmussen und Quiñonero Candela (2005) ein Vorgehen beschrieben, mit dem dieser Makel beseitigt werden kann: bei jeder Vorhersage wird temporär ein neues Basiselement zentriert auf den aktuellen Vorhersagepunkt der Menge der bisherigen Basiselemente hinzugefügt. Weiter wird in Rasmussen und Quiñonero Candela (2005) bemerkt, daß dieses Vorgehen praktisch nicht durchführbar ist, weil das ständige Hinzufügen einer Basisfunktionen (siehe Abschnitt 4.1.2) rechnerisch viel zu aufwendig ist. Mit unserem speziellem Ansatz und der effizienten Online-Approximation (siehe Abschnitt 4.2.2) wäre genau das allerdings wieder kein Problem mehr.

Technisch betrachtet entspricht das RN mit der beschriebenen automatischen Online-Basisselektion einem linear-parametrisierten Modell, und kann (da die Anzahl der selektierten Basiselemente immer beschränkt ist) wie ein gewöhnliches Basisfunktionsnetzwerk mit festen Basisfunktionen behandelt werden. Insbesondere bleiben daher für unseren Ansatz auch alle Konvergenzaussagen für approximative Politik-Evaluation mit linearen Funktionsapproximatoren bestehen (Tsitsiklis und Van Roy, 1997). Bei anderen nichtlinearen Funktionsapproximatoren, wie den Feed-Forward Neuronalen Netzen, ist das nicht der Fall; andererseits läßt sich aus dem Fehlen einer Konvergenzaussage auch nicht ableiten, daß das Verfahren praktisch nicht doch funktionieren kann: im Gegenteil, es gibt eine Reihe sehr erfolgreicher Anwendungen von Neuronalen Netzen im Reinforcement Lernen (Tesauro, 1994; Coloum, 2002).

5.4.2 Relevante Literatur

Der Ansatz kernbasierte Methoden zur Approximation der Wertefunktion einzusetzen ist kein Novum dieser Arbeit: an erster Stelle muß hier der Beitrag von Yaakov Engel genannt werden. Dieser hatte zuerst in Engel et al. (2003) und den nachfolgenden Arbeiten (Engel et al., 2005a; Engel, 2005; Engel et al., 2005b) einen Ansatz im Rahmen Bayes'scher Regression präsentiert, worin das Bellman-Residuum mittels GP-Regression modelliert wird. Weil ein Regularisierungsnetz das Nicht-Bayes'sche Gegenstück zu GP-Regression ist, und auf ein identisches Gleichungssystem (und damit derselben Lösung) hinausläuft, sind beide Ansätze rein technisch betrachtet äquivalent. Zur effizienten Online-Lösung greift der Autor – genau wie wir – auf die allgemeine SR-Approximation mit der speziellen Online-Basisselektion aus

Csató und Opper (2001) zurück. Der in Engel et al. (2003) präsentierte Algorithmus ist demnach identisch mit unserem Ansatz BRM. Allerdings ist er, wie wir auch in den Experimenten demonstriert haben, nur für Probleme mit rein deterministischen Zustandsübergängen geeignet[19]. In dieser Arbeit haben wir dagegen mit Regularisierungsnetzwerken von vorneherein einen anderen Ansatz gewählt, der allgemein das Lösen von gewöhnlichen Least-Squares Aufgaben mit Tikhonov-Regularisierung behandelt. Dadurch ist es uns möglich geworden, die auf einer Least-Squares Aufgabe basierenden Algorithmen zur approximativen Politik-Evaluation LSTD und LSPE direkt und einheitlich mit kernbasierter Funktionsapproximation zu formulieren. Anders als mit BRM sind mit LSTD und LSPE auch stochastische Zustandsübergänge möglich; zudem zeigten unsere Experimente, daß LSTD grundsätzlich auch bei deterministischen Problemen bessere Lösungen liefert, was sich mit ähnlichen Beobachtungen in Lagoudakis und Parr (2003) deckt. Eine zweite Neuerung gegenüber Engel et al. (2003) ist, daß wir, ähnlich wie in der Signalapproximation, ein überwachtes Basisselektionskriterium verwenden, wodurch die Anzahl der notwendigen Basiselemente bereits während der Laufzeit kleingehalten wird (25%-50% weniger Basisfunktionen). Eine Verringerung des Rechenaufwands ist natürlich vor allem deswegen erstrebenswert, weil wir bei einem Online-Lernproblem (wie RoboCup-Keepaway) nur sehr beschränkte Rechenkapazität zur Verfügung haben. Eine dritte Neuerung ist das „Kernelisieren“ von LSPE, wodurch wir uns auch die Möglichkeit zur optimistischen Politik-Iteration erschließen. Auf OPI basierende Verfahren können in manchen Lernaufgaben viel bessere Ergebnisse liefern; in Engel et al. (2005b) wurde dagegen für den Oktopus-Arm, um OPI wenigstens nachzuahmen, ein exponentiell abfallender Vergessensfaktor verwendet (allein mit Actor/Critic konnten die Autoren keinen Erfolg erzielen). Viertens, schließlich belegen wir anhand einer Reihe weiterer praktischer Resultate (und insbesondere der neuen und eindrucksvollen Anwendung bei Keepaway), daß kernbasierte Funktionsapproximation in RL nicht nur technisch effizient realisierbar ist, sondern möglicherweise auch das Potential hat, bei zukünftigen hoch-dimensionalen Lern-und Steuerproblemen die entscheidende Rolle zu spielen.

Ein zu RN ähnlicher kernbasierter Ansatz wird in Loth et al. (2007) beschrieben: hier wird das zu BRM gehörende Least-Squares Problem mittels Kernel-LASSO[20] (basierend auf LARS) gelöst und ein zu TD ähnlicher Online-Algorithmus präsentiert. Ein Vorteil von LASSO gegenüber RN ist, daß LASSO (wie Vorwärtsselektion mit Optimized Matching Pursuit) durch den L_1-Strafterm von vorneherein eine Entwicklung in möglichst wenigen Basiselementen erzwingt. Das Problem der künstlichen Selektion von relevanten Basiselementen bei den RN (zur Reduktion des Rechenaufwands) entfällt also. Ein zweiter Vorteil ist, daß LASSO kein auf positiv definiten Kernen basierender Algorithmus ist, sondern mit beliebigen Basisfunktionen zurecht kommt. Insbesondere darf die Menge der Basisfunktionskandidaten „gemischte“ Basisfunktionen enthalten. In Loth et al. (2007) wird das zum Beispiel ausgenutzt, um RBF-Basisfunktionen (wie bei Kernen zentriert auf den Trainingsdaten) mit unterschiedlicher Auflösung (Längenskala h) und unter Berücksichtigung der Symmetrie der Zustände als Basisfunktionskandidaten zu verwenden. Es bleibt allerdings abzuwarten, wie dieser Ansatz auf hochdimensionale Anwendungsbeispiele mit einer großen Zahl von Trainingsdaten (und damit einer noch größeren Zahl von Basisfunktionskandidaten) skaliert.

In Rasmussen und Kuss (2004) wird GP-Regression dagegen in anderer Weise mit RL verbunden: ein GP approximiert die Übergangswahrscheinlichkeiten, ein zweiter GP approximiert dann die Wertefunktion. Das Verfahren ist ein Offline-Verfahren und entspricht gefitteter Werteiteration (mit dem approximierten Modell) über einer Menge von zu Beginn speziell ausgewählten repräsentativen Zuständen. Eine Reihe weiterer Zusatzannahmen sowie experimentelle Evaluation durch ein lediglich sehr einfaches Spielzeugproblem machen eine erfolgreiche Anwendung auf hochdimensionale, realistische Lernprobleme fraglich.

[19] In der Dissertation von Engel (2005) wird allerdings gezeigt, daß GPTD für eine bestimmte Wahl der Störung (die Kovarianzmatrix muß von speziellen, nicht-diagonalen Form sein) LSTD emulieren kann.

[20] Bei RN betrachten wir ein Minimierungsproblem mit L_2-Strafterm

$$\min_{\mathbf{w}} \ \|\mathbf{A}\mathbf{w} - \mathbf{y}\|^2 + \lambda \sum w_i^2$$

bei LASSO ein Problem mit L_1-Strafterm

$$\min_{\mathbf{w}} \ \|\mathbf{A}\mathbf{w} - \mathbf{y}\|^2 + \lambda \sum |w_i|$$

Kapitel 6

Schlußwort

In dieser Arbeit haben wir die Anwendung kernbasierter Methoden zur Funktionsapproximation im Kontext von Reinforcement Lernen beschrieben und einen Lernalgorithmus für approximative Politik-Evaluation basierend auf Regularisierungsnetzen und Temporal-Difference Lernen angegeben.

Was kernbasierte Methoden vor allem aus praktischer Sicht für RL so reizvoll macht, ist, daß sie die Lösung in den Trainingsdaten entwickeln und sich daher an die Komplexität der zu lernenden Funktion anpassen können. Das ist vor allem deswegen interessant, weil RL ein Online-Problem ist, bei dem die Daten von vorneherein nicht bekannt sind, sondern erst sequentiell verfügbar gemacht werden (aus der Interaktion eines Agenten mir der Umwelt) und die jeweils zu lernende Funktion von der Strategie abhängt, die der Agent dabei verfolgt. Bei konventionellen linear parametrisierten Ansätzen (wie Gittern) werden dagegen die Basisfunktionen vor dem Lernen gleichmäßig im Eingaberaum verteilt, so daß mit wachsender Dimension des Eingaberaums deren Anzahl exponentiell anwächst (der Fluch der Dimensionen).

Der Preis, den wir für die Flexibilität der kernbasierten Methoden zu zahlen haben, ist der notwendige Rechenaufwand. Gerade weil die kernbasierten Methoden die zu lernende Approximation generell in allen gegebenen Trainingsdaten expandieren, stößt der Aufwand sehr schnell in nicht mehr durchführbare Regionen vor. Unser besonderes Augenmerk gilt deswegen der Effizienz: der Rechenaufwand kann durch Ansätze wie der SR-Approximation erheblich reduziert werden, indem tatsächlich nur ein sehr viel kleinerer Teil der Daten zur Expansion der Lösung verwendet wird.

Der erste Beitrag dieser Arbeit ist ein rekursiver Lösungsalgorithmus für RN basierend auf der SR-Approximation, der die dazu notwendigen Basiselemente automatisch und zur Laufzeit aus dem Datenstrom selektiert und damit speziell für Online-Lernen geeignet ist. Der Lösungsaufwand (pro Schritt) ist asymptotisch unabhängig von der Anzahl der Trainingsbeispiele und konstant, so daß auch sehr große Datenmengen effizient (mit linearem Aufwand) verarbeitet werden können.

Im zweiten Teil übertrugen wir dann diesen Algorithmus auf approximative Politik-Evaluation mittels Least-Squares-basiertem Temporal-Difference Lernen. Insgesamt erhalten wir so ein RL-System zum autonomen Lernen von optimalem Verhalten, das in Echtzeit operiert und insbesondere für Lernprobleme mit kontinuierlichen und hochdimensionalen Zustandsräumen sowie stochastischen Zustandsübergängen geeignet ist, kein Modell der Umwelt benötigt und Konvergenz bereits mit relativ wenigen Agent-Umwelt Interaktionen erzielt. Die Anwendung auf – für konventionelle Ansätze sehr herausfordernde – Lernprobleme (wie RoboCup-Keepaway) hat gezeigt, daß sich mit unserem Verfahren sehr gute Ergebnisse erzielen lassen, und daß dieser oder ähnliche kernbasierte Ansätze (wie Engel's GPTD) das Potential haben, das bislang nicht befriedigend gelöste Problem, eine geeignete Basis für die Repräsentation der Wertefunktion auch in höherdimensionalen Zustandsräumen (wie sie etwa in der Robotik üblich sind) automatisch zu finden, angehen zu können.

Zu den Dingen, die wir nicht mehr ausführlicher untersuchen konnten, gehört zuallererst die Entwicklung einer intelligenteren Explorationsstrategie, beispielsweise indem die Unsicherheit über die Vorhersagen genutzt wird, um zielgerichtet explorative Handlungen auszuwählen. Ein zweiter Punkt ist die Erweiterung auf kontinuierliche Aktionen, die mit diesem Ansatz zwar technisch möglich ist (über die Wahl

des Kerns), bislang aber nicht erfolgreich bei realistischen Anwendungsbeispielen demonstriert wurde. Drittens kann die Mächtigkeit der nichtparametrisierten Regularisierungsnetze auch zur Anwendung bei anderen Varianten von approximativen Dynamischen Programmieren eingesetzt werden, etwa solcher, die auf gefitteter Werteiteration beruhen. Für solche Anwendungen ist Online-Lernen keine Notwendigkeit mehr: in diesem Fall könnten wir von machtvolleren Ansätzen zur überwachten Auswahl der Basiselemente und disziplinierteren Parameterwahlstrategien (etwa GCV oder Automatic Relevance Determination) profitieren.

Ein letzter Punkt betrifft noch einmal die Dimensionalität des Zustandsraums: obwohl kernbasierte Methoden davon zunächst nicht berührt werden, kann sich (abhängig vom verwendeten Kern) wegen der SR-Approximation eine sehr hohe Dimensionalität der Zustandsvektoren ungünstig auswirken, in dem Sinne, daß sehr viele Basiselemente zu einer näherungsweisen Repräsentation der Daten im Merkmalsraum benötigt werden. In der Praxis werden hochdimensionale Zustandsbeschreibungen aber häufig redundant sein; wird bei einem Roboter beispielsweise der Zustand durch die Position jedes einzelnen Gelenks beschrieben, so kann die Position eines bestimmten Gelenks nicht unabhängig von allen anderen variieren (z.B. wird die Position eines Fußgelenks nicht völlig unabhängig vom Kniegelenk sein). Die Zustände leben daher auf einem Unterraum, der effektiv von viel kleinerer Dimension ist, als die naive Repräsentation zunächst nahelegt. Identifizieren wir solche Unterräume (beispielsweise mit PCA oder PLS), so können wir die Dimension der Zustände vor dem eigentlichen Lernvorgang mit RL reduzieren, wodurch sich das anschließende Lernen der gesuchten Wertefunktion deutlich vereinfachen könnte.

Anhang A

Was ist ein RKHS?

In diesem Anhang sammeln und fassen wir die Kernaussagen zum Lernen im RKHS zusammen. Weitergehende Informationen finden sich in den Texten zu kernbasierten Lernen (Wahba, 1990; Shawe-Taylor und Cristianini, 2004; Schölkopf und Smola, 2002) und Standardliteratur der Funktionalanalysis. Die folgende Darstellung orientiert sich vor allem an Evgeniou (2000) und Rifkin (2002).

Hilberträume mit reproduzierendem Kern (RKHS)

Ein Hilbertraum mit reproduzierendem Kern (RKHS) ist ein Hilbertraum reellwertiger Funktionen über einem beschränkten Gebiet $\mathcal{X} \subset \mathbb{R}^d$ mit der Eigenschaft, daß alle Punktauswertungen beschränkte lineare Funktionale sind. Zu jedem RKHS $\mathcal{H}$ korrespondiert eine eindeutig bestimmte, symmetrische, positiv definite Funktion $k(\mathbf{x}, \mathbf{x}')$ in den beiden Variablen $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, mit der folgenden Eigenschaft:

$$f(\mathbf{x}) = \langle f(\cdot), k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} \quad \forall f \in \mathcal{H} \tag{A.1}$$

wobei auch die umgekehrte Richtung der Aussage gültig ist. Die Funktion $k(\mathbf{x}, \mathbf{x}')$ wird auch als reproduzierender Kern von $\mathcal{H}$ bezeichnet. Die Funktion $k(\mathbf{x}, \cdot) \in \mathcal{H}$ ist nach dem Riesz'schen Darstellungssatz der Repräsenter $k_{\mathbf{x}} := k(\mathbf{x}, \cdot)$ der Auswertung an der Stelle $\mathbf{x}$. Somit wirkt sie in ähnlicher Weise wie die Deltafunktion in L_2, wobei hervorzuheben ist, daß der L_2 kein RKHS ist, weil dort nicht alle Elemente punktweise definiert sind. Ein RKHS ist somit ein Raum von speziellen Funktionen, die sich in gewisser Hinsicht ordentlich verhalten und daher für Interpolation und Approximation besonders geeignet sind.

Mercer's Theorem

Um diesen Sachverhalt etwas anschaulicher darzustellen, skizzieren wir nun einen Weg, wie ein solcher RKHS explizit konstruiert werden kann. Mercer's Theorem besagt, daß jeder symmetrische Kern $k(\mathbf{x}, \mathbf{y})$ eines positiv definiten Integraloperators entwickelt werden kann als

$$k(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d_{\mathcal{H}}} \lambda_i \varphi_i(\mathbf{x}) \varphi_i(\mathbf{y}) \tag{A.2}$$

wobei $\varphi_i \in L_2(\mathcal{X})$ die orthonormalen Eigenfunktionen und $\lambda_i > 0$ die zugehörigen Eigenwerte des Integraloperators sind, d.h.

$$\int_{\mathcal{X}} k(\mathbf{x}, \mathbf{y}) \varphi_i(\mathbf{x}) d\mathbf{x} = \lambda_i \varphi_i(\mathbf{y})$$

gilt. Für $d_{\mathcal{H}}$ gilt entweder $d_{\mathcal{H}} \in \mathbb{N}$ oder $d_{\mathcal{H}} = \infty$. Hat die Eigenwertgleichung unendlich viele Lösungen, dann liegt der zugehörige RKHS dicht in L_2. Wir betrachten nun als unseren Hilbertraum $\mathcal{H}$ die Menge aller Funktionen f von der Form

$$f(\mathbf{x}) = \sum_{i=1}^{d_{\mathcal{H}}} a_i \varphi_i(\mathbf{x}) \tag{A.3}$$

für $a_i \in \mathbb{R}$. Das Skalarprodukt zweier Funktionen in $\mathcal{H}$ definieren wir dann durch

$$\langle \sum_{i=1}^{d_\mathcal{H}} a_i \varphi_i(\mathbf{x}), \sum_{i=1}^{d_\mathcal{H}} b_i \varphi_i(\mathbf{x}) \rangle_\mathcal{H} := \sum_{i=1}^{d_\mathcal{H}} \frac{a_i b_i}{\lambda_i}. \tag{A.4}$$

Für ein beliebiges $f \in \mathcal{H}$ folgt nun:

$$\langle f(\cdot), k(\mathbf{x}, \cdot) \rangle_\mathcal{H} = \sum_{i=1}^{d_\mathcal{H}} \frac{a_i \lambda_i \varphi_i(\mathbf{x})}{\lambda_i} = \sum_{i=1}^{d_\mathcal{H}} a_i \varphi_i(\mathbf{x}) = f(\mathbf{x})$$

weil $k(\mathbf{x}, \cdot) = \sum [\lambda_i \varphi_i(\mathbf{x})] \varphi_i(\cdot)$ natürlich auch ein Element von $\mathcal{H}$ ist. Somit ist Aussage (A.1) erfüllt und $\mathcal{H}$ ein RKHS. Darüberhinaus zeigt Gleichung (A.4), daß die Norm im RKHS die folgende Gestalt hat:

$$\|f\|_\mathcal{H}^2 = \langle f(\cdot), f(\cdot) \rangle_\mathcal{H} = \sum_{i=1}^{d_\mathcal{H}} \frac{a_i^2}{\lambda_i} \ < \infty, \tag{A.5}$$

d.h. anschaulich die a_i^2 schneller gegen Null gehen müssen als die λ_i (bei $d_\mathcal{H}$ unendlich).

Merkmalsraum

Die Darstellung von Kernen mit Hilfe von Mercer's Theorem erlaubt es uns, die den RKHS aufspannenden Eigenfunktionen φ_i als gewöhnliche Basisfunktionen und damit den RKHS als hoch-oder unendlichdimensionalen Merkmalsraum aufzufassen. Betrachten wir Abbildung $\boldsymbol{\phi}$ mit

$$\begin{aligned} \boldsymbol{\phi} : \mathcal{X} &\to \ell_2^{d_\mathcal{H}} \\ \mathbf{x} &\mapsto \boldsymbol{\phi}(\mathbf{x}) := \left(\sqrt{\lambda_1} \varphi_1(\mathbf{x}), \ldots, \sqrt{\lambda_{d_\mathcal{H}}} \varphi_{d_\mathcal{H}}(\mathbf{x}) \right) \end{aligned} \tag{A.6}$$

wobei $d_\mathcal{H}$ die Dimension des Merkmalsraums (und ℓ_p^N der normierte Raum der Folgen von Länge N mit p-Norm) ist, als die Abbildung, die die Daten in den Merkmalsraum abbildet. Dann gilt für das Skalarprodukt der Bilder von Daten $\boldsymbol{\phi}(\mathbf{x})$ und $\boldsymbol{\phi}(\mathbf{y})$ im Merkmalsraum

$$\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y}) \rangle = \sum_{i=1}^{d_\mathcal{H}} \lambda_i \varphi_i(\mathbf{x}) \varphi_i(\mathbf{y}) = k(\mathbf{x}, \mathbf{y}). \tag{A.7}$$

Somit kann jede Kernfunktion als Skalarprodukt in einem hochdimensionalen Merkmalsraum interpretiert werden. Das ist besonders deswegen erwähnenswert, weil wir es einerseits bei der expliziten Repräsentation im hochdimensionalen Merkmalsraum mit sehr großen Vektoren $\boldsymbol{\phi}(\mathbf{x})$ zu tun haben und das Skalarprodukt zweier solcher Vektoren sehr viele Rechenoperationen benötigen würde. Andererseits ist aber der Kern eine geschlossene Funktion, die häufig mit sehr wenigen Rechenoperationen ausgewertet werden kann.

Zusammenfassend sind RKHS also Hilberträume, in denen das Skalarprodukt besonders einfach durch eine positiv definite Funktion ausgewertet werden kann. Ein bestimmter RKHS $\mathcal{H}$ kann auf zweierlei Weisen ausgewählt werden:

1. Primal: durch Auswahl von geeigneten λ_i, φ_i in (A.2), oder
2. Dual: durch Auswahl des reproduzierenden Kerns k.

In den allermeisten Fällen ist die zweite Alternative die übliche Vorgehensweise.

Das Repräsenter-Theorem

Grundlegend für das in dieser Arbeit beschriebene Lernen einer Funktion aus Daten $\{\mathbf{x}_i, y_i\}_{i=1}^n$ ist das Repräsenter-Theorem von Kimeldorf und Wahba (1971). Es besagt, daß für Ansatzräume $\mathcal{H}$, wobei $\mathcal{H}$ RKHS mit Kern k, die Lösung des Variationsproblems

$$\min_{f\in\mathcal{H}} \sum_{i=1}^{n} V(y_i - f(\mathbf{x}_i)) + \lambda \|f\|_{\mathcal{H}}^2$$

immer von der Form

$$f(\mathbf{x}) = \sum_{i=1}^{n} c_i k(\mathbf{x}_i, \mathbf{x})$$

ist. Die Lösung wird demnach durch die Daten parametrisiert. Die Funktion V ist eine beliebige monotone Kostenfunktion, die die Abweichungen von den Sollwerten bestraft; verwendet man quadratische Kosten, dann erhalten wir die beschriebenen Regularisierungsnetzwerke und die Koeffizienten c_i können durch Lösen des linearen Gleichungssystems $(\mathbf{K}+\lambda\mathbf{I})\mathbf{c} = \mathbf{y}$ bestimmt werden (vgl. Kapitel 2). Eine andere Wahl der Kostenfunktion, beispielsweise die ε-tolerante $|\cdot|_\varepsilon := \max(0, |\cdot| - \varepsilon)$, führt zu den Support Vektor Maschinen (Vapnik, 1998). Die Koeffizienten c_i können dort aber nicht mehr länger durch ein lineares Gleichungssystem bestimmt werden. Siehe speziell Evgeniou et al. (2000), wo verschiedene bekannte Lernverfahren im gemeinsamen Kontext von Lernen im RKHS beschrieben werden.

Die Wahl des Kerns

Die folgende Tabelle aus Evgeniou (2000) gibt eine Übersicht über die gebräuchlichsten Kerne:

Kernfunktion	**Parameter**	**Name**
$k(\mathbf{x}-\mathbf{y}) = \exp\{-h^{-1}\|\mathbf{x}-\mathbf{y}\|^2\}$	h	Gauß'sche RBF
$k(\mathbf{x}-\mathbf{y}) = (\|\mathbf{x}-\mathbf{y}\|^2 + c^2)^{-1/2}$	c	Inverse Multiquadric
$k(\mathbf{x}-\mathbf{y}) = (\|\mathbf{x}-\mathbf{y}\|^2 + c^2)^{1/2}$	c	Multiquadric[1]
$k(\mathbf{x}-\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|^{2n+1}$ $k(\mathbf{x}-\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|^{2n}\ln\|\mathbf{x}-\mathbf{y}\|$	n n	Thin Plate Splines[1]
$k(\mathbf{x},\mathbf{y}) = \tanh(\langle\mathbf{x},\mathbf{y}\rangle - \theta)$	θ	Multi Layer Perceptron[2]
$k(\mathbf{x},\mathbf{y}) = (1+\langle\mathbf{x},\mathbf{y}\rangle)^d$	d	Polynom vom Grad d
$k(x,y) = B_{2n+1}(x-y)$	n	B-Splines[3]
$k(x,y) = \frac{\sin(d+1/2)(x-y)}{\sin\left(\frac{x-y}{2}\right)}$	d	Trigon. Polyn.[3]

Die ersten vier Einträge sind translationsinvariante, radialsymmetrische Kerne, die letzten drei wurden von Vapnik (1998) im Zusammenhang mit SVM vorgeschlagen. Von größerer Bedeutung sind die translationsinvarianten Kerne, d.h. $k(\mathbf{x},\mathbf{y}) = k(\mathbf{x}-\mathbf{y})$ und hier speziell die radialen Kerne $k(\mathbf{x},\mathbf{y}) = k(\|\mathbf{x}-\mathbf{y}\|)$. Diese definieren einen RKHS, in dem die Merkmale φ_i Fourierkomponenten sind.

Die Regularisierungseigenschaften der Kerne können mit Hilfe der Fouriertransformation veranschaulicht werden, siehe Schölkopf und Smola (2002, Kapitel 4). Es zeigt sich, daß die (kontinuierliche) Fouriertransformierte des Gauß-Kerns $k(\|\mathbf{x}-\mathbf{y}\|) = \exp\{-h^{-1}\|\mathbf{x}-\mathbf{y}\|^2\}$ ebenfalls wieder eine Gaußfunktion $\mathcal{F}[k](\omega) = \text{const}\cdot\exp(-h\omega^2)$ ist und daß das Produkt zwischen Breite des Kerns und Breite seiner Fouriertransformierten konstant ist. Mit anderen Worten, je größer h und je 'breiter' die RBF, desto enger und konzentrierter verläuft ihre Fouriertransformierte. Die Fouriertransformierte wiederum steht in engem Zusammenhang damit, was als hochfrequenter und 'komplizierter' Anteil interpretiert und aufgrund der

[1] Positiv semidefinit, brauchen eine Erweiterung der hier vorgestellten einfachen RKHS Theorie.
[2] Nur für manche θ.
[3] 1-dim. Kerne. Multivariate Versionen können aus dem Tensorprodukt der univariaten erzeugt werden (vgl. Kapitel 1, Abbildung 1.5).

Regulärisierung gedämpft und herausgefiltert wird (Poggio und Girosi, 1989; Girosi et al., 1995). Breitere RBFs liefern daher einfachere und glattere Lösungen, schmälere RBFs kompliziertere und kurvigere Lösungen (abhängig von der Wahl des Regularisierungsparameters).

Tatsächlich ist es auch so, daß *alle* translationsinvarianten Kerne einen RKHS mit Fourier-Eigenfunktionen φ_i als Merkmale erzeugen und sich nur in der Form des Eigenwertspektrums unterscheiden. Es findet sich daher in der Literatur oft auch die Beobachtung, daß die Wahl des konkreten Kerns keinen sehr großen Einfluß auf das Ergebnis hat, weil alle diese Kerne ohnehin die gleiche Art von Merkmalsraum induzieren (vgl. Evgeniou, 2000).

Anhang B

Verschiedene Formeln

Transformation der Varianz

Es gilt für lineare Transformation eines Zufallsvektors $\mathbf{z}$

$$\mathrm{Var}(\mathbf{Az}) = \mathbf{A}\,\mathrm{Var}(\mathbf{z})\mathbf{A}^\mathsf{T} \tag{B.1}$$

Formeln für Matrix-Inversion

Man kann zeigen (oder durch Multiplizieren nachrechnen), daß für die Inverse einer in Blöcken partitionierten Matrix einerseits gilt (sofern die angegebenen inversen Blöcke existieren):

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1} = \begin{bmatrix}\mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}\\ -(\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1} & (\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}\end{bmatrix}.$$

Mit der gleichen Argumentation kann andererseits aber auch

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1} = \begin{bmatrix}(\mathbf{A}-\mathbf{BD}^{-1}\mathbf{C})^{-1} & -(\mathbf{A}-\mathbf{BD}^{-1}\mathbf{C})^{-1}\mathbf{BD}^{-1}\\ -\mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{BD}^{-1}\mathbf{C})^{-1} & \mathbf{D}^{-1}+\mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{BD}^{-1}\mathbf{C})^{-1}\mathbf{BD}^{-1}\end{bmatrix}$$

gezeigt werden. Aufgrund der Eindeutigkeit der Inversen müssen die Einträge auf den jeweils rechten Seiten übereinstimmen. Wir erhalten:

$$\begin{aligned}(\mathbf{A}-\mathbf{BD}^{-1}\mathbf{C})^{-1} &= \mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1}\\ (\mathbf{A}-\mathbf{BD}^{-1}\mathbf{C})^{-1}\mathbf{BD}^{-1} &= \mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}\\ \mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{BD}^{-1}\mathbf{C})^{-1} &= (\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1}\\ \mathbf{D}^{-1}+\mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{BD}^{-1}\mathbf{C})^{-1}\mathbf{BD}^{-1} &= (\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}.\end{aligned}$$

Mit Umdrehen des Vorzeichens ($\mathbf{C} := -\mathbf{C}$) erhalten wir aus den ersten beiden Identitäten die Formel von Sherman-Morrison-Woodbury:

$$(\mathbf{A}+\mathbf{BD}^{-1}\mathbf{C})^{-1} = \mathbf{A}^{-1}-\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}+\mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1} \tag{B.2}$$

sowie

$$(\mathbf{A}+\mathbf{BD}^{-1}\mathbf{C})^{-1}\mathbf{BD}^{-1} = \mathbf{A}^{-1}\mathbf{B}(\mathbf{D}+\mathbf{CA}^{-1}\mathbf{B})^{-1}. \tag{B.3}$$

Die Inverse einer partitionierten Matrix kann somit kurz auch als

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1} = \begin{bmatrix}\mathbf{A}^{-1} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix} + \begin{bmatrix}-\mathbf{A}^{-1}\mathbf{B}\\ \mathbf{I}\end{bmatrix}\boldsymbol{\Delta}\begin{bmatrix}-\mathbf{CA}^{-1} & \mathbf{I}\end{bmatrix} \tag{B.4}$$

mit $\boldsymbol{\Delta} = (\mathbf{D}-\mathbf{CA}^{-1}\mathbf{B})^{-1}$ geschrieben werden.

Anhang C

Zusammenfassung der Rekursionsgleichungen

Sei $\mathbf{x}_{t+1} = (s_{t+1}, a_{t+1})$ das aktuelle beobachtete Zustandsaktionspaar und r_{t+1} die beim Übergang vom letzten Zustand s_t nach s_{t+1} unter a_t beobachtete Belohnung. Wir definieren folgende Abkürzungen:

$$\mathbf{k}_t := \mathbf{k}_m(\mathbf{x}_t) \qquad \mathbf{k}_{t+1} := \mathbf{k}_m(\mathbf{x}_{t+1}) \qquad \mathbf{h}_{t+1} := \mathbf{k}_t - \gamma \mathbf{k}_{t+1}$$
$$k_t^* := k(\mathbf{x}_t, \mathbf{x}_{t+1}) \qquad k_{t+1}^* := k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \qquad h_{t+1}^* := k_t^* - \gamma k_{t+1}^*$$

und $\mathbf{a}_{t+1} := \mathbf{K}_{mm}^{-1} \mathbf{k}_{t+1}$.

Basiserweiterungstest (unüberwacht)

Wir testen, ob $\mathbf{x}_{t+1}$ von den gegenwärtig selektierten m Basisfunktionen hinreichend gut repräsentiert wird (im Merkmalsraum), oder ob eine neue Basisfunktion (zentriert auf $\mathbf{x}_{t+1}$) hinzugefügt werden muß. Berechne

$$\delta = k_{t+1}^* - \mathbf{k}_{t+1}^\mathsf{T} \mathbf{a}_{t+1}. \tag{C.1}$$

Falls $\delta >$ `TOL1`, führe Basiserweiterungsschritt (siehe unten) aus und aktualisiere $\mathbf{K}_{mm}^{-1}$:

$$\mathbf{K}_{m+1,m+1}^{-1} = \begin{bmatrix} \mathbf{K}_{mm}^{-1} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix} + \frac{1}{\delta} \begin{bmatrix} -\mathbf{a}_{t+1} \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{a}_{t+1} \\ 1 \end{bmatrix}^\mathsf{T}. \tag{C.2}$$

Rekursionsgleichungen für BRM

- Normaler Schritt $\{t, m\} \mapsto \{t+1, m\}$:

  1. $$\mathbf{P}_{t+1,m}^{-1} = \mathbf{P}_{tm}^{-1} - \frac{\mathbf{P}_{tm}^{-1} \mathbf{h}_{t+1} \mathbf{h}_{t+1}^\mathsf{T} \mathbf{P}_{tm}^{-1}}{\Delta} \tag{C.3}$$

     mit $\Delta = 1 + \mathbf{h}_{t+1}^\mathsf{T} \mathbf{P}_{tm}^{-1} \mathbf{h}_{t+1}$.

  2. $$\mathbf{w}_{t+1,m} = \mathbf{w}_{tm} + \frac{\varrho}{\Delta} \mathbf{P}_{tm}^{-1} \mathbf{h}_{t+1} \tag{C.4}$$

     mit $\varrho = r_{t+1} - \mathbf{h}_{t+1}^\mathsf{T} \mathbf{w}_{tm}$.

- Basiserweiterungsschritt $\{t+1, m\} \mapsto \{t+1, m+1\}$

1.
$$\mathbf{P}_{t+1,m}^{-1} = \begin{bmatrix} \mathbf{P}_{t+1,m}^{-1} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} + \frac{1}{\Delta_b} \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix}^\mathsf{T} \tag{C.5}$$

wobei

$$\mathbf{w}_b = \mathbf{a}_{t+1} + \frac{\delta_h}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{h}_{t+1}, \qquad \Delta_b = \frac{\delta_h^2}{\Delta} + \sigma\delta_h, \qquad \delta_h = h_{t+1}^* - \mathbf{h}_{t+1}^\mathsf{T}\mathbf{a}_{t+1}$$

2.
$$\mathbf{w}_{t+1,m+1} = \begin{bmatrix} \mathbf{w}_{t+1,m} \\ 0 \end{bmatrix} + \kappa \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix} \tag{C.6}$$

wobei $\kappa = -\frac{\delta_h \varrho}{\Delta_b \Delta}$.

- Fehlerreduktion durch Hinzufügen von $\mathbf{x}_{t+1}$ (überwachte Basisselektion):
$$\xi_{t+1,m+1} = \xi_{t+1,m} - \kappa^2 \Delta_b \tag{C.7}$$
Bei der überwachten Basisselektion fügen wir nur dann eine Basisfunktion (zentriert auf $\mathbf{x}_{t+1}$) hinzu, wenn zusätzlich zu $\delta >$ `TOL1` aus (C.1) auch $\kappa^2\Delta_b >$ `TOL2` gilt.

Rekursionsgleichungen für LSTD(λ)

- Normaler Schritt $\{t, m\} \mapsto \{t+1, m\}$:

1.
$$\mathbf{z}_{t+1,m} = (\gamma\lambda)\mathbf{z}_{tm} + \mathbf{k}_t$$

2.
$$\mathbf{P}_{t+1,m}^{-1} = \mathbf{P}_{tm}^{-1} - \frac{\mathbf{P}_{tm}^{-1}\mathbf{z}_{t+1,m}\mathbf{h}_{t+1}^\mathsf{T}\mathbf{P}_{tm}^{-1}}{\Delta} \tag{C.3'}$$

mit $\Delta = 1 + \mathbf{h}_{t+1}^\mathsf{T}\mathbf{P}_{tm}^{-1}\mathbf{z}_{t+1,m}$.

3.
$$\mathbf{w}_{t+1,m} = \mathbf{w}_{tm} + \frac{\varrho}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{z}_{t+1,m} \tag{C.4'}$$

mit $\varrho = r_{t+1} - \mathbf{h}_{t+1}^\mathsf{T}\mathbf{w}_{tm}$.

- Basiserweiterungsschritt $\{t+1, m\} \mapsto \{t+1, m+1\}$

1.
$$\mathbf{z}_{t+1,m+1} = \begin{bmatrix} \mathbf{z}_{t+1,m}^\mathsf{T} & z_{t+1,m}^* \end{bmatrix}^\mathsf{T}$$

mit $z_{t+1,m}^* = (\gamma\lambda)\mathbf{z}_{tm}^\mathsf{T}\mathbf{a}_{t+1} + k_t^*$.

2.
$$\mathbf{P}_{t+1,m}^{-1} = \begin{bmatrix} \mathbf{P}_{t+1,m}^{-1} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} + \frac{1}{\Delta_b} \begin{bmatrix} -\mathbf{w}_b^{(1)} \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{w}_b^{(2)} & 1 \end{bmatrix} \tag{C.5'}$$

mit

$$\mathbf{w}_b^{(1)} = \mathbf{a}_{t+1} + \frac{\delta^{(1)}}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{z}_{t+1,m} \qquad \delta^{(1)} = h_{t+1}^* - \mathbf{a}_{t+1}^\mathsf{T}\mathbf{h}_{t+1}$$

$$\mathbf{w}_b^{(2)} = \mathbf{a}_{t+1}^\mathsf{T} + \frac{\delta^{(2)}}{\Delta}\mathbf{h}_{t+1}^\mathsf{T}\mathbf{P}_{tm}^{-1} \qquad \delta^{(2)} = z_{t+1,m}^* - \mathbf{a}_{t+1}^\mathsf{T}\mathbf{z}_{t+1,m}$$

und $\Delta_b = \frac{\delta^{(1)}\delta^{(2)}}{\Delta} + \sigma(k_{t+1}^* - \mathbf{k}_{t+1}^\mathsf{T}\mathbf{a}_{t+1})$.

3.
$$\mathbf{w}_{t+1,m+1} = \begin{bmatrix} \mathbf{w}_{t+1,m} \\ 0 \end{bmatrix} + \kappa \begin{bmatrix} -\mathbf{w}_b^{(1)} \\ 1 \end{bmatrix} \tag{C.6'}$$

mit $\kappa = -\frac{\delta^{(2)}\varrho}{\Delta_b\Delta}$.

Rekursionsgleichungen für LSPE(λ)

- Normaler Schritt $\{t, m\} \mapsto \{t+1, m\}$:

  1.
  $$\begin{aligned} \mathbf{z}_{t+1,m} &= (\gamma\lambda)\mathbf{z}_{tm} + \mathbf{k}_{t+1} \\ \mathbf{A}_{t+1,m} &= \mathbf{A}_{tm} + \mathbf{z}_{t+1,m}\mathbf{h}_{t+1}^{\mathsf{T}} \\ \mathbf{b}_{t+1,m} &= \mathbf{b}_{t,m} + \mathbf{z}_{t+1,m} r_{t+1} \end{aligned}$$

  2.
  $$\mathbf{P}_{t+1,m}^{-1} = \mathbf{P}_{tm}^{-1} - \frac{\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{P}_{tm}^{-1}}{\Delta} \tag{C.3"}$$

  mit $\Delta = 1 + \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}$.

  3.
  $$\mathbf{w}_{t+1,m} = \mathbf{w}_{tm} + \eta\mathbf{P}_{t+1,m}^{-1}(\mathbf{b}_{t+1,m} - \mathbf{A}_{t+1,m}\mathbf{w}_{tm}) \tag{C.4"}$$

- Basiserweiterungsschritt $\{t+1, m\} \mapsto \{t+1, m+1\}$

  1.
  $$\mathbf{z}_{t+1,m+1} = \begin{bmatrix} \mathbf{z}_{t+1,m} \\ z_{t+1,m}^* \end{bmatrix} \qquad \mathbf{b}_{t+1,m+1} = \begin{bmatrix} \mathbf{b}_{t+1,m} \\ \mathbf{a}_{t+1}^{\mathsf{T}}\mathbf{b}_{t,m} + z_{t+1,m}^* r_{t+1} \end{bmatrix}$$
  $$\mathbf{A}_{t+1,m+1} = \begin{bmatrix} \mathbf{A}_{t+1,m} & \mathbf{A}_{tm}\mathbf{a}_{t+1} + \mathbf{z}_{t+1,m}h^* \\ \mathbf{a}_{t+1}^{\mathsf{T}}\mathbf{A}_{tm} + z_{t+1,m}^*\mathbf{h}_{t+1}^{\mathsf{T}} & \mathbf{a}_{t+1}^{\mathsf{T}}\mathbf{A}_{tm}\mathbf{a}_{t+1} + z_{t+1,m}^* h^* \end{bmatrix}$$

  wobei $z_{t+1,m}^* = (\gamma\lambda)\mathbf{z}_{tm}^{\mathsf{T}}\mathbf{a}_{t+1} + k_t^*$.

  2.
  $$\mathbf{P}_{t+1,m}^{-1} = \begin{bmatrix} \mathbf{P}_{t+1,m}^{-1} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} + \frac{1}{\Delta_b}\begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix}\begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix}^{\mathsf{T}} \tag{C.5"}$$

  wobei
  $$\mathbf{w}_b = \mathbf{a}_{t+1} + \frac{\delta}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}, \qquad \Delta_b = \frac{\delta^2}{\Delta} + \sigma\delta, \qquad \delta = k_t^* - \mathbf{k}_t^{\mathsf{T}}\mathbf{a}_{t+1}$$

  und $\Delta_b = \frac{\delta^{(1)}\delta^{(2)}}{\Delta} + \sigma(k_{t+1}^* - \mathbf{k}_{t+1}^{\mathsf{T}}\mathbf{a}_{t+1})$.

  3.
  $$\mathbf{w}_{t+1,m+1} = \begin{bmatrix} \mathbf{w}_{t+1,m} \\ 0 \end{bmatrix} + \kappa\begin{bmatrix} -\mathbf{w}_b^{(1)} \\ 1 \end{bmatrix} \tag{C.6"}$$

  wobei $\kappa = -\frac{\delta^{(2)}\varrho}{\Delta_b\Delta}$.

- Fehlerreduktion durch Hinzufügen von $\mathbf{x}_{t+1}$ (überwachte Basisselektion):
  $$\xi_{t+1,m+1} = \xi_{t+1,m} - \Delta_b^{-1}(c - \mathbf{w}_b^{\mathsf{T}}\mathbf{d})^2 \tag{C.7"}$$

  wobei $c = \mathbf{a}_{t+1}^{\mathsf{T}}(\mathbf{b}_{t,m} - \mathbf{A}_{tm}\mathbf{w}_{tm}) + z_{t+1,m}^*(r_{t+1} - \mathbf{h}_{t+1}^{\mathsf{T}}\mathbf{w}_{tm})$ und $\mathbf{d} = \mathbf{b}_{t+1,m} - \mathbf{A}_{t+1,m}\mathbf{w}_{tm}$. Bei der überwachten Basisselektion fügen wir nur dann eine Basisfunktion (zentriert auf $\mathbf{x}_{t+1}$) hinzu, wenn zusätzlich zu $\delta >$ `TOL1` aus (C.1) auch $\Delta_b^{-1}(c - \mathbf{w}_b^{\mathsf{T}}\mathbf{d})^2 >$ `TOL2` gilt.

Literaturverzeichnis

J. S. Albus. A new approach to manipulator control: the cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97(3):220–227, 1975.

M. Andrle, L. Rebollo-Neira und E. Sagiones. Backward-optimized orthogonal matching pursuit approach. *IEEE Signal Processing Letters*, 11(9):705–708, 2004.

F. R. Bach und M. I. Jordan. Kernel independent component analysis. *JMLR*, 3:1–48, 2002.

F. R. Bach und M. I. Jordan. Predictive low-rank decomposition for kernel methods. In *Proc. of ICML 22*, 2005.

L. C. Baird. Residual Algorithms: Reinforcement Learning with Function Approximation. In *Proc. of ICML 12*, Seiten 30–37, 1995.

D. Bertsekas und J. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, 1996.

D. P. Bertsekas und S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technischer Report, LIDS Tech. Report LIDS-P-2349, MIT, 1996.

D. P. Bertsekas, V. S. Borkar und A. Nedić. Improved temporal difference methods with linear function approximation. Technischer Report, LIDS Tech. Report 2573, MIT, 2003, (also appears in Learning and Approximate Dynamic Programming, by A. Barto, W. Powell, J. Si, (Eds.), IEEE Press, 2004)., 2003.

C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

J. A. Boyan. Least-Squares Temporal Difference Learning. In *Proc. of ICML 16*, 1999.

S. J. Bradtke und A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

M. Campbell, A. J. Hoane Jr. und F.-h. Hsu. Deep Blue. *Artificial Intelligence*, 134:57–83, 2002.

S. Chen, F. Cowen und P. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Trans. on Neural Networks*, 2(2):302–309, 1991.

R. Coloum. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control.* Doktorarbeit, INP Grenoble, 2002.

P. Craven und G. Wahba. Smoothing noisy data with spline functions: estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik*, 31:377–403, 1979.

L. Csató und M. Opper. Sparse representation for Gaussian process models. In *Advances in NIPS 13*, Seiten 444–450, 2001.

G. Davis, S. Mallat und Z. Zhang. Adaptive time-frequency decompositions. *Optical Engineering*, 33(7): 2183–2191, 1994.

P. Drineas und M. W. Mahoney. On the Nyström Method for approximating a Gram matrix for improved kernel-based learning. *JMLR*, 6:2153–2175, 2005.

Y. Engel. *Algorithms and Representations for Reinforcement Learning.* Doktorarbeit, The Hebrew University of Jerusalem, 2005.

Y. Engel, S. Mannor und R. Meir. Bayes meets Bellman: The Gaussian Process Approach to Temporal Difference Learning. In *Proc. of ICML 20*, Seiten 154–161, 2003.

Y. Engel, S. Mannor und R. Meir. The Kernel Recursive Least Squares Algorithm. *IEEE Trans. on Sig. Proc.*, 52(8):2275–2285, 2004.

Y. Engel, S. Mannor und R. Meir. Reinforcement Learning with Gaussian Processes. In *Proc. of ICML 22*, 2005a.

Y. Engel, P. Szabo und D. Volkinshtein. Learning to Control an Octopus Arm with Gaussian Process Temporal Difference Methods. In *Advances in NIPS 17*, 2005b.

T. Evgeniou. *Learning with kernel machine architectures.* Doktorarbeit, MIT, 2000.

T. Evgeniou, M. Pontil und T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13:1–50, 2000.

S. Fine und K. Scheinberg. Efficient SVM training using low-rank kernel representation. *JMLR*, 2: 243–264, 2001.

C. Fowkles, S. Belongie, F. Chun und J. Malik. Spectral grouping using the Nyströem method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):214–225, 2004.

I. Frank und J. Friedman. A statistical view of some chemometrics regression tools (with discussion). *Technometrics*, 35(2):109–148, 1993.

J. Garcke. *Maschinelles Lernen durch Funktionsrekonstruktion mit verallgemeinerten dünnen Gittern.* Doktorarbeit, Universität Bonn, 2004.

F. Girosi, M. Jones und T. Poggio. Regularization theory and Neural Network architectures. *Neural Computation*, 7(2):219–269, 1995.

G. Golub und C. Van Loan. *Matrix Computations (3rd Edition).* John Hopkins University Press, Baltimore, 1996.

G. J. Gordon. Stable function approximation in dynamic programming. Technischer Report, CMU-CS-95-103, Carnegie Mellon University, 1995.

P. Green und B. Silverman. *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach.* Chapman and Hall, London, 1994.

M. Hanke und P. Hansen. Regularization methods for large-scale problems. *Surveys on Mathematics for Industry*, 3:253–315, 1993.

P. C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion.* Society for Industrial and Applied Mathematics, 1998.

T. Hastie, R. Tibshirani und J. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

L. Hoegarts, J. A. Suykens, J. Vandervalle und B. De Moor. Subset Based Least Squares Subspace Regression in RKHS. *Neurocomputing*, 63:293–323, 2005.

A. E. Hoerl und R. Kennard. Ridge Regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(3):55–67, 1970.

I. T. Jolliffe. *Principal Component Analysis.* Springer, New York, 1986.

T. Jung und D. Polani. Least-squares SVM for Least-squares TD learning. In *Proc. of ECAI 17*, 2006a.

T. Jung und D. Polani. Sequential learning with LS-SVM for Large-Scale Data Sets. In *Proc. of ICANN 16*, 2006b.

T. Jung und D. Polani. Learning RoboCup-Keepaway with Kernels. *JMLR: Workshop and Conference Proceedings (Gaussian Processes in Practice)*, 1:33–57, 2007.

T. Jung, L. Herrera und B. Schölkopf. Long Term Prediction of Product Quality in a Glass Manufacturing Process Using a Kernel Based Approach. In *Proc. of 8th IWANN*, Seiten 960–967. Springer, 2005.

S. Sathiya Keerthi und W. Chu. A matching pursuit approach to sparse Gaussian process regression. In *Advances in NIPS 15*, 2005.

G. Kimeldorf und G. Wahba. Tchebycheffian spline functions. *J. Math. Ana. Applic.*, 33:82–95, 1971.

M. G. Lagoudakis und R. Parr. Least-squares Policy Iteration. *JMLR*, 4:1107–1149, 2003.

M. Loth, M. Davy und P. Preux. Sparse Temporal Difference Learning using LASSO. In *Proc. of 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, Seiten 352–359, 2007.

A. K. Louis. *Inverse und schlecht gestellte Probleme*. Teubner, 1989.

Z. Luo und G. Wahba. Hybrid Adaptive Splines. *J. Amer. Statist. Assoc.*, 92:107–116, 1997.

S. Mallat und Z. Zhang. Matching Pursuit with time-frequency dictionaries. *IEEE Trans. Signal Proc.*, 41(12):3397–3415, 1993.

R. Munos. Error bounds for approximate policy iteration. In *Proc. of ICML 20*, Seiten 560–567, 2003.

A. Nedić und D. P. Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications*, 13:79–110, 2003.

D.J. Newman, S. Hettich, C.L. Blake und C.J. Merz. UCI Repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

I. Noda, H. Matsubara, K. Hiraki und I. Frank. Soccer Server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.

Y. Pati, R. Rezaiifar und P. Krishnaprasad. Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition. In *Proc. of the 27th Annual Asimolar Conf. on Signals, Systems and Computers*, Seiten 40–44, 1993.

T. Poggio und F. Girosi. A theory of networks for approximation and learning. Technischer Report, A.I. Memo No. 1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1989.

T. Poggio und F. Girosi. Networks for Approximation and Learning. *Proceedings of IEEE*, 78:1481–1497, 1990.

V. Popovici, S. Bengio und J. P. Thiran. Kernel matching pursuit for large datasets. *Pattern Recognition*, 38(12):2385–2390, 2005.

C. Rasmussen und M. Kuss. Gaussian processes in reinforcement learning. In *Advances in NIPS 16*, 2004.

C. E. Rasmussen und J. Quiñonero Candela. Healing the Relevance Vector Machine through Augmentation. In *Proc. of ICML 22*, 2005.

Carl Edward Rasmussen und Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

L. Rebollo-Neira und D. Lowe. Optimized orthogonal matching pursuit approach. *IEEE Signal Processing Letters*, 9(4):137–140, 2002.

R. Rifkin. *Everything Old is New again: A Fresh Look at Historical Approaches in Machine Learning.* Doktorarbeit, MIT, 2002.

C. Saunders, A. Gammerman und V. Vork. Ridge Regression Learning Algorithm in Dual Variables. In *Proc. of the 15th ICML*, Seiten 515–521, 1998.

A. Sayed. *Fundamentals of Adaptive Filtering.* Wiley Interscience, 2003.

B. Schölkopf und A. Smola. *Learning with Kernels.* Cambridge, MA: MIT Press, 2002.

M. Seeger, C. K. Williams und N. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Workshop on AI and Statistics 9*, 2003.

J. Shawe-Taylor und N. Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, Cambridge, England, 2004.

J. Shawe-Taylor, C. K. I. Williams, N. Cristianini und J. Kandola. On the eigenspectrum of the Gram matrix and the generalisation error of kernel PCA. *IEEE Transactions on Information Theory*, 51(7): 2510–2522, 2005.

A. J. Smola und P. L. Bartlett. Sparse greedy Gaussian process regression. In *Advances in NIPS 13*, Seiten 619–625, 2001.

A. J. Smola und B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proc. of ICML 17*, Seiten 911–918, 2000.

P. Stone, R. S. Sutton und G. Kuhlmann. Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

R. Sutton und A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor und J. Vandewalle. *Least Squares Support Vector Machines.* World Scientific, Singapore, 2002.

G. J. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc. B.*, 58:267–288, 1996.

A. N. Tikhonov und V. Y. Arsenin. *Solutions of Ill-posed Problems.* W. H. Winston, Washington, D.C., 1977.

J. N. Tsitsiklis und B. Van Roy. An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

V. N. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

P. Vincent und Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48:165–187, 2002.

G. Wahba. *Splines Models for Observational Data. Series in Applied Mathematics, Vol. 59.* SIAM, Philadelphia, 1990.

C. J. C. H. Watkins. *Learning from Delayed Rewards.* Doktorarbeit, Cambridge, England, 1989.

C. Williams und M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in NIPS 13*, Seiten 682–688, 2001.

C. Williams, C. Rasmussen, A. Schwaighofer und V. Tresp. Observations on the Nyström method for Gaussian processes. Technischer Report, Tech.rep., Institute for Adaptive and Neural Computation, Division of Informatics, University of Edinburgh, 2002.

H. Wold. Soft Modeling by Latent Variables; the Nonlinear Iterative Partial Least Squares Approach. In editor J. Gani, Herausgeber, *Perspectives in Probability and Statistics, Papers in Honour of M.S. Bartlett*, Seiten 520–540. Academic Press, London, 1975.

X. Xu, H. He und D. Hu. Efficient Reinforcement Learning Using Recursive Least-Squares Methods. *Journal of Artificial Intelligence Research*, 16:259–292, 2002.