Überblick

1 Motivation

Wenn wir komplexere Aufgabenstellungen mit Methoden des maschinellen Lernens bearbeiten wollen, stehen wir vor mehreren Fragen:

- Wie kann man aus einzelnen Lernmodulen mit begrenzten Fähigkeiten leistungsfähigere Systeme zusammenfügen?
- Wie kann man die Lernschritte der zusammengefügten Module koordinieren?
- Wie kann man die Interaktion mit der Umgebung aktiv strukturieren, um aus möglichst wenigen Beispielen möglichst viel zu lernen?
- Wie kann man Aktionsfolgen lernen, deren einzelne Schritte aufeinander aufbauen müssen?
- Wie kann man lernen, wenn keine Lösungsbeispiele, sondern lediglich Erfolgs/Mißerfolgsrückmeldungen gegeben werden?

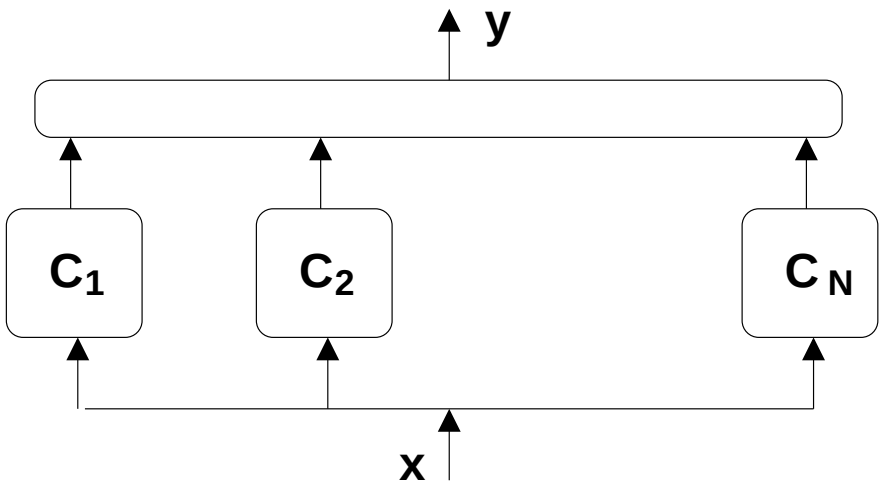
Lernarchitekturen bieten den allgemeinen Rahmen, um diese Fragen näher zu beleuchten und zu beantworten.

Definition Lernarchitektur: Unter einer Lernarchitektur verstehen wir ein Schema, das eine komplexe Lernaufgabe durch ein Zusammenwirken mehrerer einfacherer Teile löst. Die Zerlegung kann dabei auf der Systemebene, der Funktionsebene, oder beiden zusammen erfolgen. Darüberhinaus lassen sich parallele, hierarchische und serielle Zerlegungsstrategien unterscheiden.

2 Beispiele von Lernarchitekturen

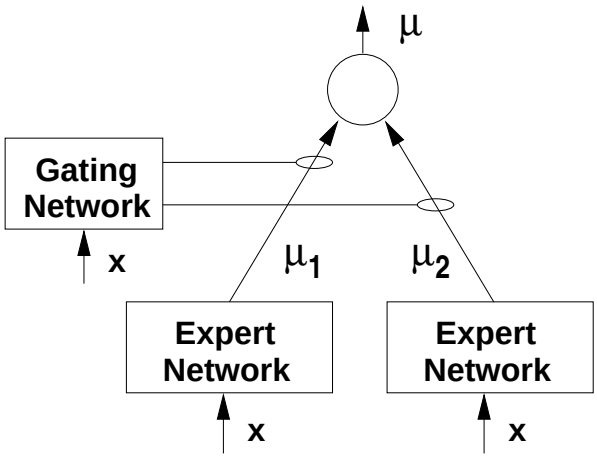
2.1 Ensemblelernen

Hier gelangt man durch "Parallelschalten" mehrerer Klassifikatoren zu einem Klassifikator, der die Klassifikationsleistung der Einzelklassifikatoren oft erheblich übertrifft.



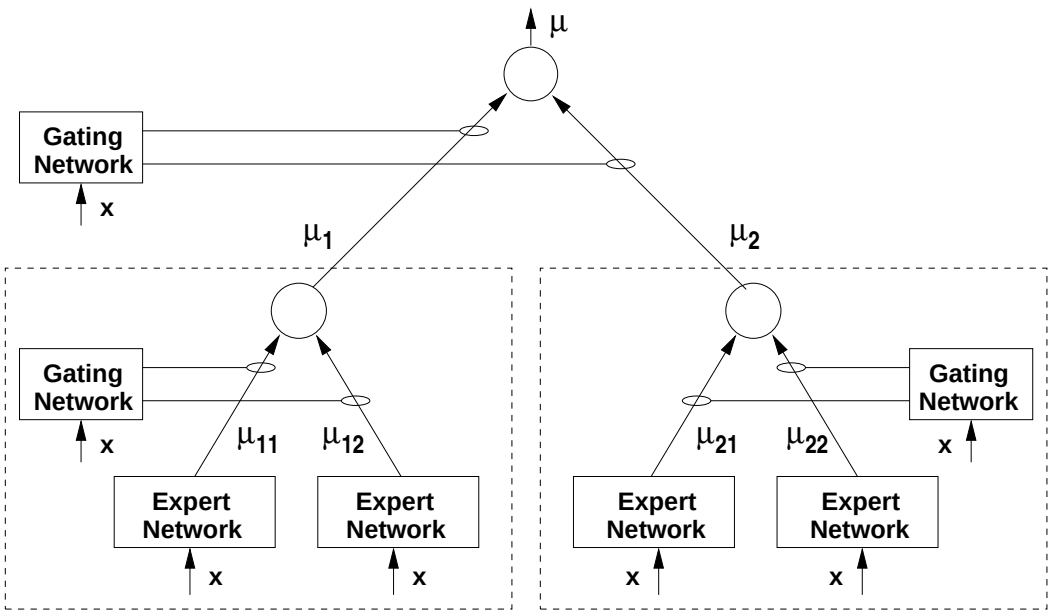
2.2 Expertenmischungs-Architektur

Verwendet eine Anzahl unterschiedlich spezialisierter "Experten-Module", die wiederum parallelgeschaltet werden. Im Unterschied zum Ensemblelernen wird der wird der Beitrag jedes einzelnen Moduls jedoch jetzt in Abhängigkeit vom momentanen Input durch ein übergeordnetes "Gating-Modul" dynamisch gewichtet. Dadurch ergibt sich ein ähnlicher Effekt wie bei einem Aufmerksamkeitssteuerungsmechanismus.



2.3 Hierarchische Architekturen

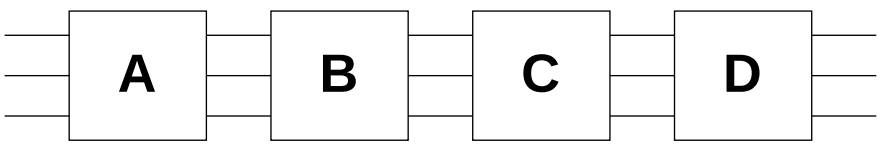
Hier bilden die einzelnen Module eine hierarchische Baumstruktur. Die Eingabe erfolgt an der Wurzel. Dort erfolgt in einem ersten eine (u.U. gewichtete) Weiterleitung in die Zweige der nächsten Ebene, bis die Module in den Blättern erreicht sind. Diese liefern schließlich die Ausgabe.



Mit diesem Aufbau ermöglichen hierarchische Architekturen eine schrittweise Verarbeitung der Eingabe, etwa voranschreitend vom Groben zum Feinen.

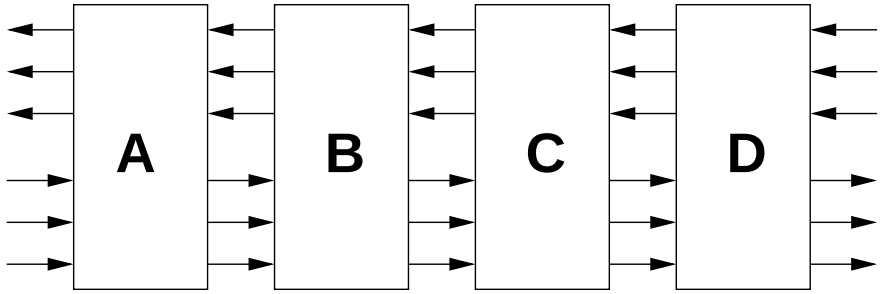
2.4 Serielle Feedforward-Architekturen

Hier liegen mehrere Verarbeitungsmodule in Reihe. Damit wird die Gesamtaufgabe in mehrere, sequentielle Teilschritte zerlegt. Läßt sich als Spezialfall einer verzweigungslosen Baumstruktur auffassen. Wie bereits dort, läuft auch hier der Informationsfluß streng in eine Richtung, d.h., es ist keine Rückkopplung von "höheren" zu "tieferen" Ebenen möglich. Wichtige Spezialfälle sind Markovketten und Hidden-Markov-Systeme.



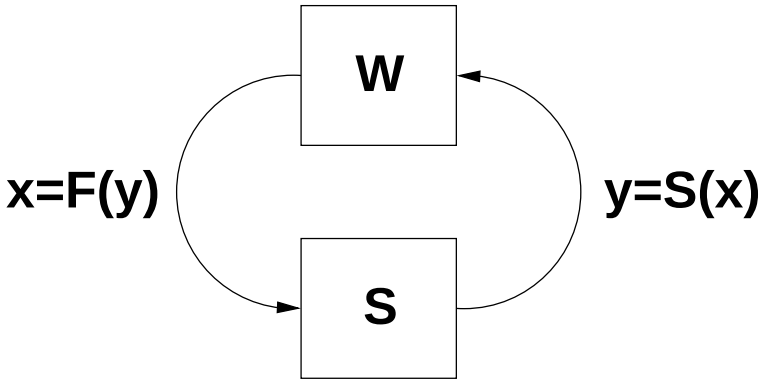
2.5 Bidirektionale Architekturen

Wie serielle Feedforward-Architektur, aber mit der Möglichkeit eines hin- ("Bottom-Up") und eines rücklaufenden ("Top-Down") Verarbeitungswegs. Über den letzteren kann ein bereits gewonnenes (Teil-)ergebnis die weitere Verarbeitung der früheren Stufen verändern.



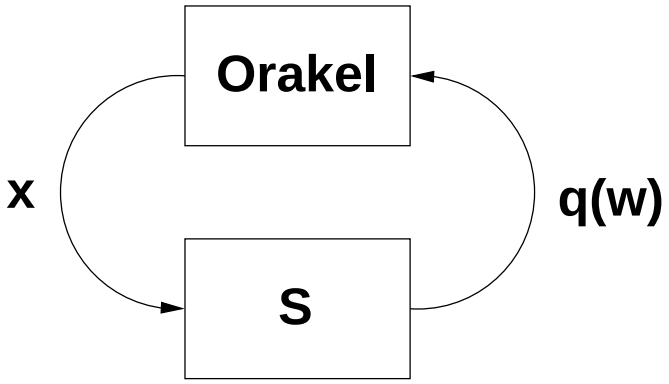
2.6 Closed-Loop-Lernarchitekturen

Hier wird als neues Element die Rückkopplungsschleife "über die Welt" geschlossen: Das System verfügt über "Aktoren", die auf die Umgebung einwirken und damit die künftigen Eingaben des Lerners verändern. Mit der dadurch hergestellten Perzeptions-Aktions-Schleife wird Lernen zu einem *aktiven Prozeß*, bei dem der Lerner seine Trainingsdaten zielgerichtet beeinflussen kann. Daher spricht man auch von "Wahrnehmungs-Handlungs-Architekturen".



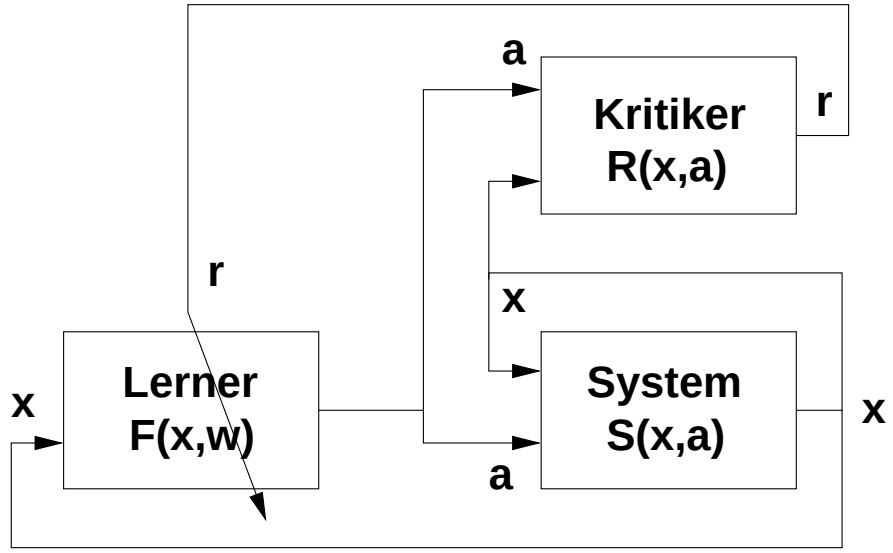
2.7 Query-Architekturen

Im einfachsten Fall spezifizieren die "Aktoren" eine "Frage" an eine "Orakel", aus dessen Antwort der Lerner einen Lernfortschritt erzielen möchte. Query-Architekturen zielen darauf ab, möglichst "informative" Fragen zu generieren, die ein schnelles Lernen erlauben.



2.8 Reinforcement-Lernarchitekturen

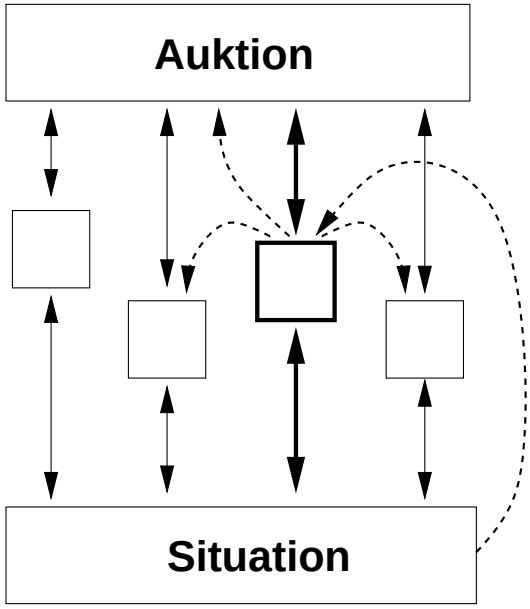
liefern einen Rahmen, wie ein Lerner eine Sequenz aufeinander aufbauender Aktionen ohne Lösungsbeispiele, sondern allein gestützt auf selbstgesteuerter Exploration lernen kann.



Einzige Informationsquelle dabei ist die Rückmeldung von Erfolg oder Mißerfolg am Ende (gelegentlich auch zwischendurch) einer Aktionsfolge. Wesentliches Element derartiger Architekturen ist häufig ein internes "Kritiker-Modul", das im Laufe der Zeit versucht, eine Vorhersage des längerfristig zu erwartenden Erfolgs oder Mißerfolgs zu lernen.

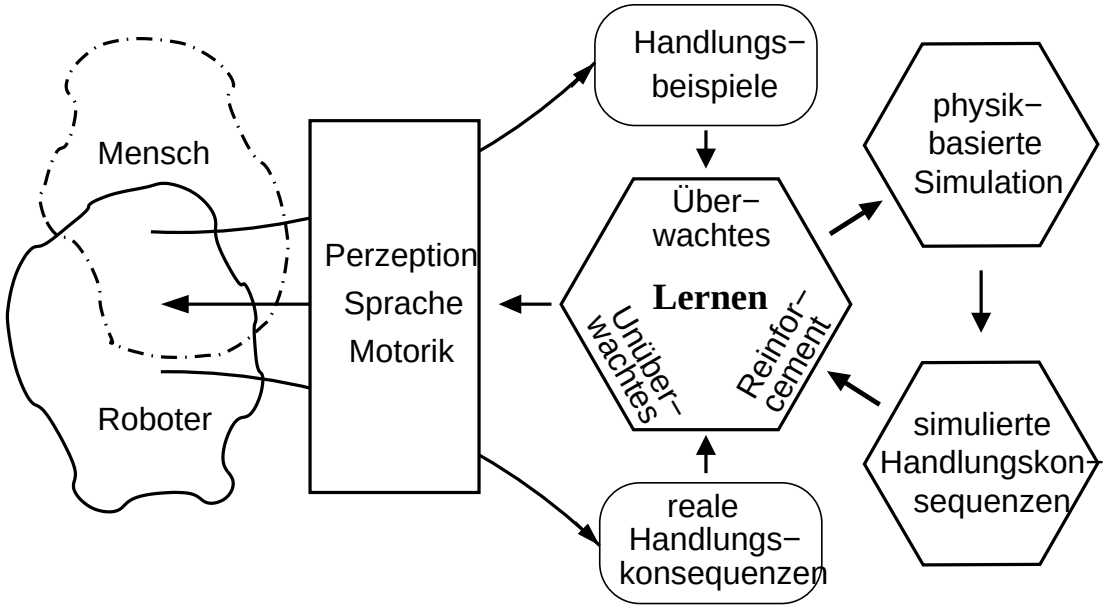
2.9 Agenten-Architekturen

organisieren Lernen als Wettbewerb/Kooperation innerhalb einer „Gemeinschaft" von Agenten. Ein interessanter Ansatz verwendet *Marktprinzipien* (Agenten „verdienen" mit erfolgreichen Aktionen „Geld" und können als „Bieter" neue Aufträge „ersteigern") als Triebfeder für die Herausbildung von Spezialisierung und Kooperation.



2.10 Architekturen für Imitationslernen

sind am komplexesten und beeinhalten komplette Systeme, die mehrere der vorstehenden Architekturen verbinden um neue Fähigkeiten aus der Beobachtung erfolgreicher Aktionen zu lernen.



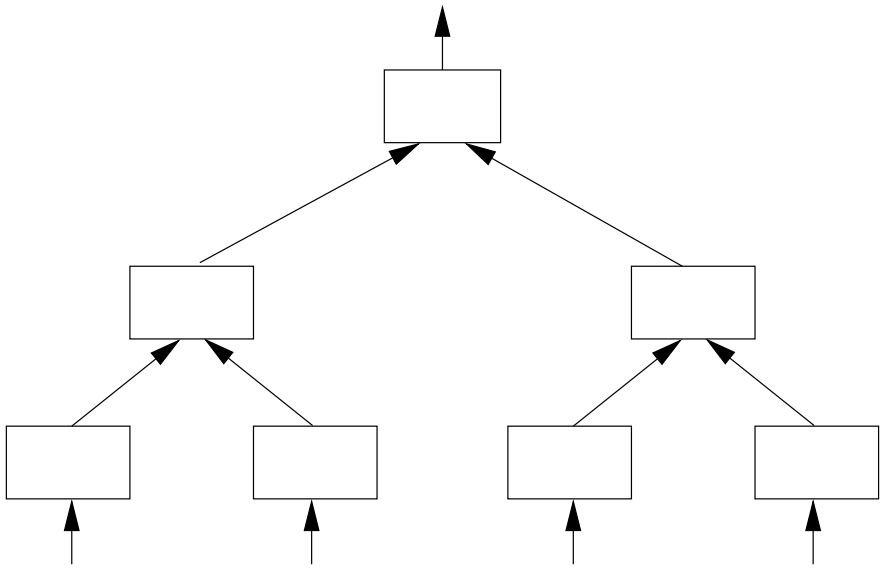
Mit den meisten der dargestellten Architekturen werden wir uns in den folgenden Kapiteln näher auseinandersetzen.

3 Granularität

Die Module einer Architektur können von unterschiedlicher Komplexität ("Granularität") sein. Solange wir nicht auf spezielle Eigenschaften der von einem Modul repräsentierten Eingabe-Ausgabe-Abbildung zurückgreifen, bleiben unsere Überlegungen gleichermaßen gültig für Architekturen aus ganzen (intern wiederum strukturierten) Subsystemen bis hinab zu Modulen aus einzelnen Neuronen eines KNN.

4 Das Credit-Assignment Problem

tritt typischerweise auf, wenn mehrere Lernmodule in einer Lernarchitektur ihre Lernschritte koordinieren müssen. Es betrifft die Frage nach der geeignetsten Stelle im System, um eine adaptive Veränderung zwecks Verbesserung des Antwortverhaltens vorzunehmen.



Beispiel: Baumarchitektur Abb. **??**. Bei zu kleinem Antwortwert:

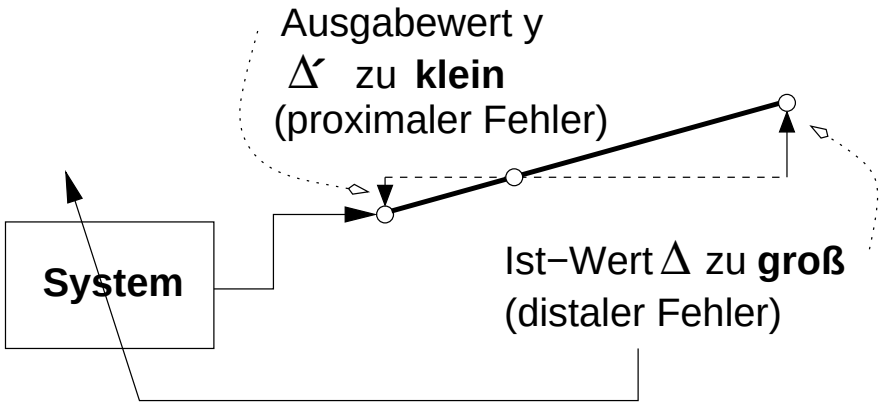
- welches der Module soll seine Ausgabe verringern?
- oder ist es besser, die notwendige Änderung gleichmäßig auf alle Module zu verteilen?
- oder kann eine geeignete, ungleichmäßige Verteilung zu einem langfristig besseren Resultat führen?
- wie wirkt sich eine Ausgabeveränderung früherer Module im Verarbeitungsweg auf das Endergebnis aus?

Das Beispiel illustriert einige der folgenden Schwierigkeiten und daraus aufgeworfener Fragen:

Redundanz in einem komplexen System gibt es i.d.R. viele unterschiedliche Veränderungsmöglichkeiten, die alle zur selben Veränderung in der momentanen Einzelantwort führen. Ein einzelner Trainingsschritt bietet nur einen kleinen Teil der Information, die zur Auflösung dieser Redundanz nötig wäre. Wie kann man diese Redundanz durch geeignete Zusatzannahmen auflösen?

fehlende Zielwerte selbst wenn das "verantwortliche" Modul bekannt wäre, ist für die Ausgabewerte "innerer" Module nicht ohne weiteres ein Zielwert bekannt, auf den hin ihre Ausgabe zu verändern wäre. Wie kann man geeignete "innere Werte" konstruieren?

distale Fehler dies kann selbst für Ausgabemodule gelten, wenn der Fehler nicht direkt, sondern erst nach Vermittlung weiterer Prozesse in der Welt entsteht. Wie kann man geeignete proximale Fehler ableiten?



verzögerte Rückmeldung in einer zusammenhängenden Aktionskette kann die Verantwortung für einen Fehler u.U. bei schon länger zurückliegenden Antworten liegen. Wie kann man dieses "zeitliche" Credit-Assignment Problem lösen?

5 Weitere Herausforderungen

"kostspielige" Trainingsbeispiele. Als Folge können häufig nur vergleichsweise wenige Trainingsschritte durchgeführt werden. Wie kann man den Informationsgehalt jedes Trainingsbeispiels maximieren?

Fluch der Dimensionalität Wie kann man eine hochdimensionale Lernsituation in leichter lernbare, niedrigerdimensionale Teile aufspalten?

Unvollständige Beobachtbarkeit Wichtige Interna eines zu steuernden Systems können für den Lerner unzugänglich sein (z.B. sog. "Partially Observable Markov Decision Problems – POMDPs). Dies kann an der Kapselung relevanter Variabler, mangelnder sensorischer Auflösung ("perceptual aliasing") oder der Überlagerung von statistischen Störeinflüssen liegen. Wie kann man in diesem Falle den internen Zustand des zu steuernden Systems *rekonstruieren* oder zumindest *schätzen*?

Diese und die vorstehenden Fragen wollen wir in den nächsten Kapiteln näher betrachten.

6 Themen der Vorlesung

1. Überblick (dieses Kapitel)
2. Ensemblelernen
3. Mischungsarchitekturen
4. Closed-Loop-Lernarchitekturen
5. Aktives Lernen
6. Reinforcement-Lernen
7. Lernen in Stochastischen Systemen
8. Agentenarchitekturen

Literatur

[Braun] Braun. Neuronale Netze

Ensemble-Lernen

1 Motivation

Ensemble-Lernen ermöglicht die Konstruktion eines "starken" Klassifikators durch geeignete (feste) Kombination eine Anzahl „schwacher" Klassifikatoren. Dabei werden zwei Hauptideen verwendet: (i) Erzeugung eines möglichst diversen Ensembles von Klassifikatoren zur selben Aufgabenstellung (meist durch Training auf geeignet erzeugten Varianten der gegebenen Datenmenge) und (ii) geeignete Kombination dieser zu dem gesuchten "starken" Klassifikator.

2 Grundansatz

Gemeinsamer Ausgangspunkt aller Verfahren ist ein Algorithmus $A : D \mapsto h$, der aus einer gegebenen Trainingsmenge $D = \{(x^\alpha, y^\alpha) \in X \times Y \| \alpha = 1 \dots M\}$ einen Klassifikator $h : X \mapsto Y$ generiert. Dabei sind X, Y Definitions- und Wertebereich der Klassifikationsaufgabe (z.B. $X = \mathbb{R}^d, Y = \{-1, 1\}$), M bezeichnet die Anzahl der Trainingsbeispiele (x, y). A kann beispielsweise ein Trainingsalgorithmus für eine neuronales Netz, eine Support-Vektormaschine oder ein Konstruktionsverfahren für einen Klassifikationsbaum sein.

Grundidee (i) wird umgesetzt, indem aus der gegebenen Trainingsmenge D eine Anzahl voneinander verschiedener Varianten D_t, $t = 1, 2 \dots ..K$ erzeugt wird. Daraus wird durch Anwendung von A ein Ensemble von Klassifikatoren $h_t = A(D_t)$ gewonnen. Die einzelnen Verfahren unterscheiden sich darin, wie die D_t generiert werden.

Grundidee (ii) wird meist als gewichtete Summation verwirklicht:

$$H(x) = \sum_{k=1}^{K} \alpha_k h_k(x).$$

Ensemblemethoden realisieren damit ein Architekturschema mit separatem, überwachtem Training seiner Module:

3 Elementare Verfahren

Im einfachsten Falle werden die einzelnen D_t durch voneinander statistisch unabhängige Variationen von D gewonnen und die resultierenden Klassifikatoren h_t zu einem Ergebnisklassifikator H gemittelt:

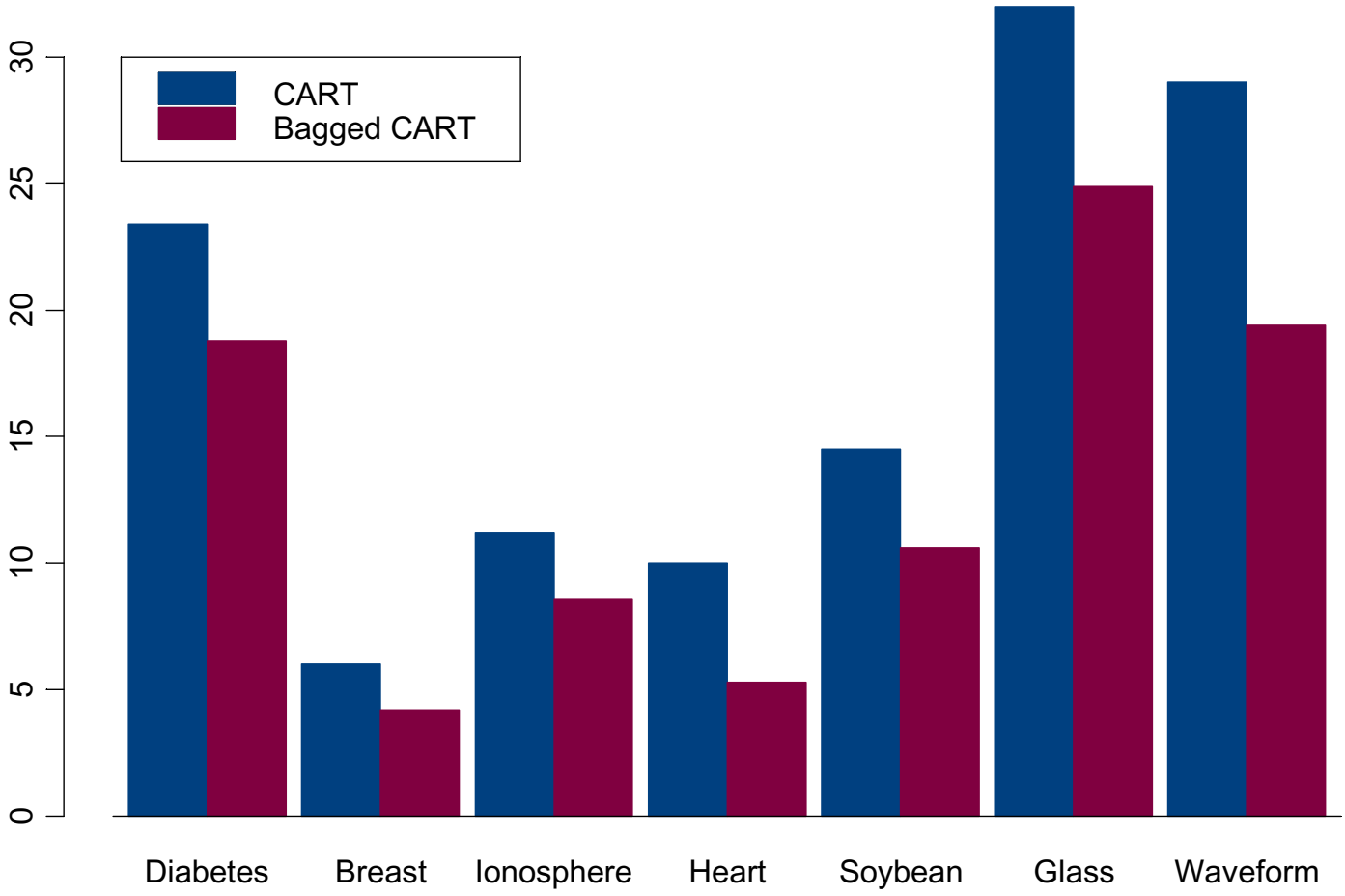
$$H(x) = \frac{1}{K} \sum_{k=1}^{K} h_k(x)$$

Grundidee dabei ist, daß sich dabei unkorrelierte Fehler der Einzelklassifikatoren herausmitteln, so daß im Ergebnis eine bessere Klassifikationsleistung entsteht.

Datenrauschen. Hier wird jedes D_t durch geringfügiges Verrauschen von D (ggf. mit mehrfacher Wiederholung unterschiedlich verrauschter Varianten jedes Datenpunkts) gewonnen. Während die Hinzufügung von Rauschen zu den Ausgabewerten eher einen Informationsverlust bewirkt, kann additives normalverteiltes Rauschen $N(0, \sigma)$ im Eingabeteil x jedes Datenpunkts die Generalisierungsfähigkeit verbessern, da es ähnlich einer (weichen) "Schutzzone" vom Radius σ um jeden Eingabepunkt wirkt, die die Klassifikationsgrenze auf Abstand zu halten versucht.

Bagging. Hier wird jedes D_t durch m-maliges zufälliges Auswählen eines Trainingspaars (x^α, y^α) (mit erlaubten Wiederholungen desselben Paars) aus D erzeugt. D.h., jedes D_t geht durch relative statistische Wichtung der ursprünglichen Datenpunkte α beim Training mit ganzzahligen Gewichten $d_\alpha \in 0, 1, 2, \ldots$ einer Bernoulli-Verteilung hervor.

Beispiel: Datenklassifikation:


CART
Bagged CART
0
5
10
15
20
25
30
Diabetes
Breast
Ionosphere
Heart
Soybean
Glass
Waveform

In manchen Fällen kann ein geeignetes Klassifikatorensemble auch bereits durch alleiniges Variieren der Anfangsbedingungen des Trainingsalgorithmus A (z.B. zufällig gewürfelte Startwerte der Gewichte eines KNN) aus ein und demselben Trainingsdatensatz D generiert werden.

4 AdaBoost

Hier wird als entscheidende Verbesserung die Sequenz h_t der Einzelklassifikatoren iterativ erzeugt, wobei in jedem Iterationsschritt jedes Trainingsbeispiel in Abhängigkeit von seinem aktuellen Beitrag zum Klassifikationsfehler gewichtet wird. „Schwierige" („einfache") Beispiele werden dabei allmählich höher (niedriger) gewichtet. Dadurch wird das Training der sukzessive später erzeugten Klassifikatoren allmählich immer entschiedener auf die noch verbleibenden „Problemfälle" fokussiert.

Ausgangssituation.

Trainingsdaten: $(x_1, y_1), \ldots (x_M, y_M)$, wobei $y_i \in \{-1, 1\}$ korrekte Klassifikation von Beispiel $x_{\in} X$ angibt.

„schwache" Klassifikatoren: es existiere ein Algorithmus, der zu jeder vorgegebenen Datenpunkt-Wichtung $d(1), \ldots d(M)$ $(\forall d(i) > 0, \sum_i d(i) = 1)$ einen „schwachen Klassifikator" $h : X \mapsto \{-1, 1\}$ derart erzeugen kann, daß dessen „$d-$gewichteter Fehler"

$$\epsilon = \sum_{i:y_i \neq h(x_i)} d_i < \frac{1}{2} - \gamma$$

einen festen „Margin" $\gamma > 0$ gegenüber einer zufälligen Klassifikation nicht unterschreitet.

Dann lautet der AdaBoost-Algorithmus im Detail:

1. Initialisierung Setze $t = 1$ und gewichte alle Datenpunkte gleichmäßig:

$$d_t(i) = 1/M$$

2. Trainingsschritt Trainiere einen neuen „schwachen" Klassifikator h_t. Trainingsziel für h_t ist die Minimierung des mit den Gewichten $d_t(i)$ des zuletzt ausgeführten Iterationsschritts gewichteten Trainingsfehlers

$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} d_i$$

3. Neue Gewichte Bestimme aus dem im vorigen Schritt erreichten Trainingsfehler ϵ_t ein „Klassifikatorgewicht"

$$\alpha_t = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$$

und bestimme neue Datengewichte

$$d_{t+1}(i) = d_t(i) \frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

wobei

$$Z_t = \sum_{i=1}^{M} \exp(-\alpha_t y_i h_t(x_i))$$

die Summe aller $d_{t+1}(i)$ zu 1 normiert.

4. Terminationstest Setze

$$H(x) = \operatorname{sgn} \sum_{t'=1}^{t} \alpha_t h_t(x)$$

und ermittle den *ungewichteten* Klassifikationsfehler

$$\epsilon^* = \sum_{i:H(x_i) \neq y_i} \frac{1}{M}$$

Falls ϵ^* klein genug oder die maximale Iterationszahl erreicht ist, ende mit Ergebnisklassifikator $H(\cdot)$. Andernfalls: $t = t + 1$ und weiter bei Schritt 2.

5 Theorie des Adaboost-Verfahrens

Wir wollen zeigen: das Adaboost-Verfahren liefert nach T Schritten einen Klassifikator, dessen Trainingsfehler durch

$$\epsilon_t \leq \exp(-2 \sum_{t=1}^{T} \gamma_t^2 T) \leq \exp(-2T\gamma^2)$$

nach oben beschränkt ist. Dabei sind $\gamma_t \geq \gamma > 0$ (vgl. Gl.(4) die „Margins" der zwischendurch konstruierten „schwachen" Klassifikatoren gegenüber zufälliger Klassifikation.

Beweis: Im folgenden setzen wir $f(x) = \sum_t \alpha_t h_t(x)$. Dann ist

$$d_T(i) = d_0(i) \cdot \prod_t \frac{e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \tag{1}$$

$$= \frac{1}{M} \frac{e^{-y_i f(x_i)}}{\prod_t Z_t} \tag{2}$$

Zur Abschätzung des Fehlers benutzen wir zunächst die offensichtlich stets erfüllte Ungleichung

$$1 - \delta_{H(x_i),y_i} \leq \exp(-y_i f(x_i))$$

(wegen $y_i f(x_i) < 0$, falls $H(x_i) \neq y_i$). Daher können wir den *ungewichteten* Endfehler ϵ^* folgendermaßen abschätzen:

$$\epsilon^* = \frac{1}{M} \sum_{i=1}^{M} (1 - \delta_{H(x_i),y_i}) \tag{3}$$

$$\leq \frac{1}{M} \sum_{i=1}^{M} e^{-y_i f(x_i)} \tag{4}$$

$$= \sum_{i=1}^{M} d_T(i) \prod_t Z_t = \prod_t Z_t \tag{5}$$

Für jedes einzelne Z_t erhalten wir die Beziehung

$$Z_t = \sum_i d_t(i) e^{-\alpha_t y_i h_t(x_i)} \tag{6}$$

$$= \sum_{i:y_i \neq h_t(x_i)} d_t(i) e_t^{\alpha} + \sum_{i:y_i = h_t(x_i)} d_t(i) e^{-\alpha_t} \tag{7}$$

$$= \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t} \tag{8}$$

(wobei wir uns an Gl. 4 und $\sum_i d_t(i) = 1$ erinnert haben). Dies legt nahe, α_t so zu bestimmen, dass Z_t minimiert wird:

$$0 = \frac{\partial Z_t}{\partial \alpha_t} = \epsilon_t e^{\alpha_t} - (1 - \epsilon_t) e^{-\alpha_t}$$

Daraus folgt unschwer die in Gl. 4 getroffene Wahl für α_t, sowie

$$Z_t = 2\sqrt{(1-\epsilon_t)\epsilon_t} = \sqrt{1-4\gamma_t^2} \leq e^{-2\gamma_t^2}$$

oder

$$\epsilon^* \leq \prod_t Z_t \leq= \exp(-2\sum_t \gamma_t^2)$$

und damit die Behauptung.

Attraktiv an diesem Ergebnis ist

- solange jeder „schwache" Klassifikator einen festen „Margin" $\gamma > 0$ gegenüber einer Zufallsklassifikation nicht unterschreitet, gelingt es Adaboost einen „starken" Klassifikator zu konstruieren.
- große $\gamma_t > 0$ einzelner Klassifikatoren schlagen sich als Vorteil direkt im Endresultat nieder
- weder γ noch T müssen vorher bekannt sein.

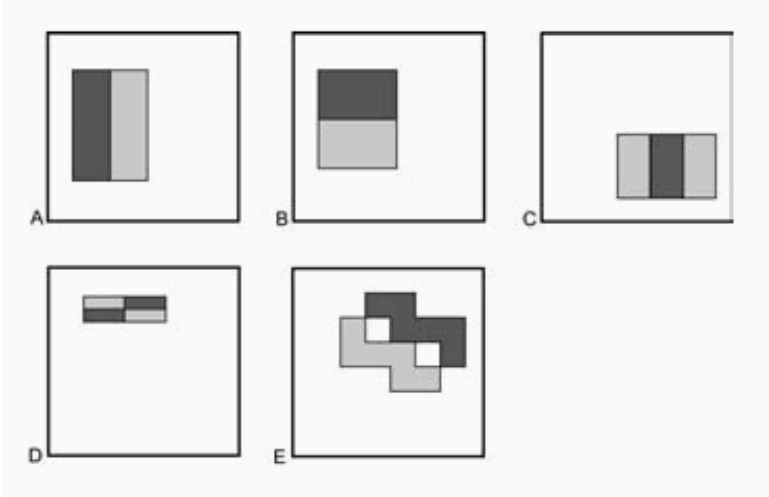
Aber beachte: über den *Generalisierungsfehler* wird durch dieses Ergebnis noch nichts ausgesagt!

6 Beispiel: Gesichtserkennung ([?])

Anwendungsdaten: Grauwert-Bildpaare $x_i = (I_i^A, I_i^B)$ von Gesichtern. $y_i = +1/-1$ falls Gesichter A,B gleich/verschieden.

Gesucht: Klassifikator $C(x) = C(I^A, I^B)$ zur Entscheidung, ob Gesichter A,B übereinstimmen oder nicht.

Einzelklassifikatoren: $C_t(x)$ wird mittels effizient berechenbarer Rechteck-Faltungsmasken $\phi(I)$ konstruiert (Abb.**??**):



$$C_t(I^A, I^B) = \begin{cases} \alpha_t & \text{falls} \quad |\phi_t(I^A) - \phi_t(I^B)| > \theta_t \\ \beta_t & \text{sonst.} \end{cases}$$

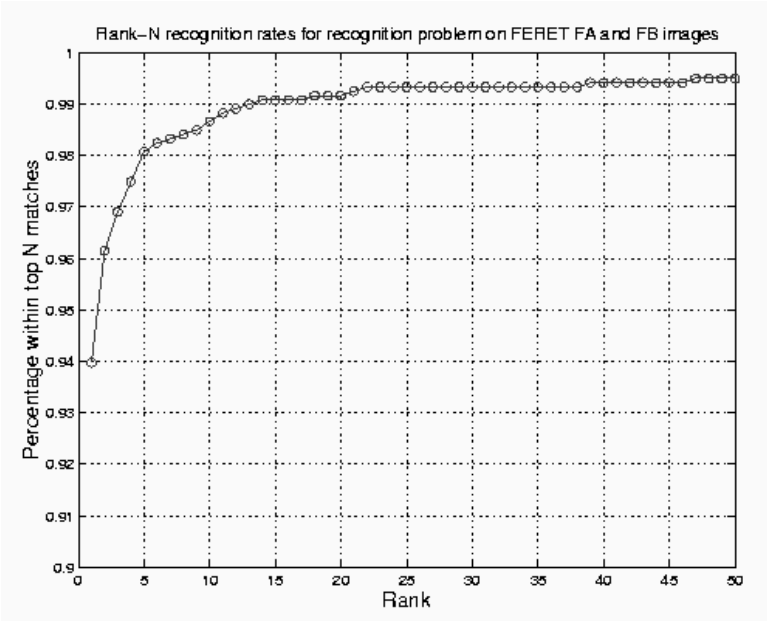
(hierbei sind $\phi_t(), \alpha_t, \beta_t, \theta_t$ Optimierungsparameter des $t-$ten Einzelklassifikators.

Trainingsschritt: besteht (i) in Wahl der besten Kombination Maske-Schwellwert (ϕ_t, θ_t) für die Klassifikationsfehler auf Trainingsdaten minimiert wird. Anschließend werden (ii) optimale Gewichtsparameter α_t, β_t durch Minimierung von Z_t bestimmt (Verallgemeinerung von Schritt 3. oben: hier zweiter Parameter β_t anstelle von α_t allein).

Datenbasis: FERET-Gesichter-Database: ca. 1200 Bildpaare, auf 45×36 Pixel normiert. Generierung von $y_i = 1$ und $y_i = -1$ Bildpaaren im Verhältnis von etwa 1:8.

weitere Details: Ausgangsensemble von ca. 52000 Rechteckfaltungsmasken. 400 Adaboost-Runden mit ebensovielen Ergebnismerkmalen.

Ergebnisse:



- 94% Klassifikationsgenauigkeit (bestes veröffentlichtes Konkurrenzresultat: 96%).
- sehr schnelles System (wenige ms pro Entscheidung)

7 Zusammenfassung

Literatur

[JonVio] M. Jones, P. Viola (2000) Face Recognition Using Boosted Local Features.

[EldRid99] J.F. Elder, G. Ridgeway (1999) Combining Estimators to Improve Performance. KDD-99 Tutorial

Mixture-of-Experts-Architektur

1 Motivation

Modularisierung ist ein wichtiges Grundprinzip der Informatik: komplexe Aufgaben werden in kleinere Teilaufgaben zerlegt, für Lösungen leichter gefunden werden können. Anschließend werden die Teillösungen zur Gesamtlösung zusammengefügt.

Die Mixtures-of-Experts-Architektur (MOE) bietet einen Rahmen, für die datengetriebene Konstruktion einer komplexen Eingabe-Ausgabe-Abbildung Teile einer solche Zerlegungs- und Lösungsstrategie durch ein geschlossenes Lernverfahren zu automatisieren.

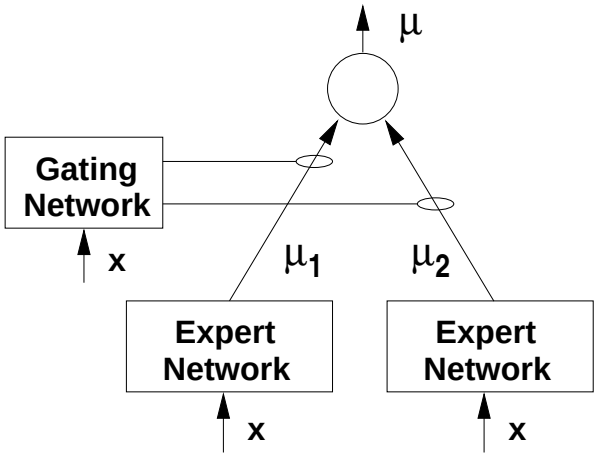
Die Betrachtung der MOE-Lernarchitektur befaßt uns mit folgenden drei wichtige Fragen:

- Wie kann man in einem System mehrerer parallel arbeitender adaptiver Module diese durch Lernen so spezialisieren, daß sie sich für eine vorgebene (feste) Aufgabe möglichst gut ergänzen?
- Wie kann man zugleich mit der Herausbildung der Spezialisierung die notwendige Koordination der Antworten der Teilmodule durch ein weiteres „Gating-Modul" mitlernen?
- Wie sieht ein direkt auf die Identifikation verborgener Variablen (in MOE: Beteiligung der Teilmodule) gerichtetes Lernverfahren aus?

2 Konzept

Die Grundarchitektur ist in Abb. **??** dargestellt. Im Vergleich zur Kommittee-Architektur des Ensemble-Lernens (**??**) gibt es ein zusätzliches *Gating-Modul*, das

den Beitrag der (auch hier wieder) additiv überlagerten Modulausgaben nunmehr *dynamisch*, d.h., in Abhängigkeit vom momentanen Input, wichtet. Ein zweiter Unterschied betrifft den Lernalgorithmus selbst: im Gegensatz zum Ensemble-Lernen wird die gesamte Lernarchitektur „in einem Stück" und mit dem Originaldatensatz trainiert, d.h., es ist keine „künstliche Auftrennung" in separat zu trainierende Komponenten mehr erforderlich.



Die von MOE geleistete Eingabe-Ausgabe-Transformation ist

$$F(x;\theta) = \sum_{s=1}^{M} g_s(x, v_s) f_k(x; w_s)$$

Dabei ist $g_m(x, v_s)$ die Ausgabe des Gating-Moduls für „Kanal" s, $f_s(x; w_s)$ die Antwortfunktion des s-ten Expertenmoduls, und $\theta = (v_1 \dots v_M, w_1 \dots w_M)$ fasst alle adaptiven Parameter zusammen.

3 "Naives" Trainingsverfahren

Lernen geschieht durch iterative Minimierung eines geeigneten Fehlermaßes $E_D(\theta)$, das auf einer Trainingsdatenmenge D definiert ist.

Eine "naive" Fehlerfunktion (Datenpaare (x_i, y_i)) wäre:

$$\tilde{E}_D(\theta) = \frac{1}{2} \sum_i \left(y_i - F(x_i; \theta)\right)^2$$

Minimierung von E_D kann mit Gradientenabstieg bezüglich der Parameter $\theta = (v, w)$ erfolgen. Dies kann als Online-Verfahren datenpunktweise (stochastische Approximation) oder als Batch-Verfahren (vor jedem Schritt Summation über alle Datenpunkte) durchgeführt werden.

4 Statistische Interpretation

Eine besser begründete Fehlerfunktion ergibt sich, wenn man die Lernaufgabe als *statistische Modellierung eines unbekannten Datenerzeugungsprozesses* auffaßt. Angestrebtes Ergebnis ist dann ein sog. *Generatives Modell* für die zu lernenden Daten.

Definition „Generatives Modell" Ein Generatives Modell modelliert eine vorgegebene Datenquelle durch einen *stochastischen Prozess,* der Datenwerte hervorbringt, die nach Struktur und Verteilung die vorgegebene Datenquelle approximieren.

Gl. (2) modelliert eine Datenquelle, die folgendermaßen charakterisiert ist:

- Aufbau aus M Teilkomponenten
- jede Eingabe x führt zur zufälligen Auswahl genau einer Teilkomponente s, die die Antwort übernimmt.
- die a-priori[1] Auswahlwahrscheinlichkeit für s ist dabei

$$p_s = \frac{g_s(x, v_s)}{\sum_{s'} g_{s'}(x, v_{s'})}$$

- die Antwort der gewählten Komponente k ist

$$y = f_s(x; w_s) + \eta_s$$

  wobei $\eta_s \propto N(0, \sigma_s)$ eine normalverteilte Zufallsvariable ist (additives Rauschen mit Varianz σ_s^2).

Im folgenden verlangen wir o.b.d.A. die Normierung $\sum_s g_s(x, v_s) = 1$ (die sich z.B. mit dem Ansatz $g(x, v_s) = e^{-x/cdotv_s} / \sum_{s'} e^{-x/cdotv_{s'}}$ sicherstellen läßt).

[1] da y noch nicht bekannt!

Dann ist die Wahrscheinlichkeit $P(z|\theta)$ eines Datenpunkts $z = (x, y)$ einfach

$$P(z|\theta) = \sum_s g_s P(s, z|\theta) \tag{1}$$

$$= \sum_s g_s(x, v_s) N_s \exp\left[-\frac{1}{2\sigma^2}(y - f_s(x; w_s))^2\right] \tag{2}$$

$$= \sum_s q_s(x, v_s) \exp\left[-\frac{1}{2\sigma^2}(y - f_s(x; w_s))^2\right] \tag{3}$$

(wobei $q_s = g_s \cdot N_s$). Bei statistisch unabhängig vorausgesetzten Datenpunkten wird die dazugehörige log-Likelihood-Funktion daher

$$L_D(\theta) = \log \prod_i P(z_i|\theta) \tag{4}$$

$$= \sum_i \log\left(\sum_{s=1}^{M} q_s(x; v_s) \exp\left[-\frac{(y_i - f_s(x; w_s))^2}{2\sigma_s^2}\right]\right) \tag{5}$$

Die Interpretation als generatives Modell in Verbindung mit der Maximum-Likelihood-Methode führt daher zu der gegenüber der „naiven" Variante abweichenden Kostenfunktion

$$E_D(\theta) = -L_D(\theta)$$

die wiederum mit Gradientenabstieg iterativ minimiert werden kann.

5 EM-Verfahren

Der MOE-Ansatz (als generatives Modell) ist ein wichtiges Beispiel einer oft anzutreffenden, allgemeineren Situation: Gesucht sind die Maximum-Likelihood-Parameter

$$\theta^* = arg\max \sum_i \log P(z_i; \theta)$$

für eine „komplizierte" Wahrscheinlichkeitsdichte $P(z_i; \theta)$, die additiv aus einer Anzahl *einfacherer* Wahrscheinlichkeiten $P_s(z_i; \theta)$ aufgebaut ist:

$$P(z_i; \theta) = \sum_s P_s(z_i; \theta)$$

Jedes $P_s(z_i;\theta) \equiv P(s, z_i;\theta)$ erhält dabei die Interpretation einer Verbundwahrscheinlichkeit für das Auftreten eines um eine „verborgene" Hilfsvariable s „vervollständigten" Wertepaars (s, z_i) (Die Summation über s bedeutet dann gerade, daß der hilfsweise eingeführte Wert von s am Ende wieder ignoriert wird).

Für diese Art von Situation liefert das EM-Verfahren eine elegante Methode, die Likelihoodfunktion durch wiederholtes Abwechseln zweier Schritte zu maximieren:

1. **(„E-Schritt")** hier werden die a-posteriori[2] Wahrscheinlichkeiten („Erwartungen") q_{si} geschätzt, mit denen ein Datenpunkt z_i einer der „Quellen" $s = 1, 2 \ldots$ entstammt:

$$\hat{q}_{si} = P(s|z_i;\theta_t) = \frac{P_s(z_i;\theta_t)}{P(z_i;\theta_t)}$$

   Dabei sind θ_t die bis zum t-ten Schritt des Verfahrens gewonnenen Schätzwerte der wahren Parameter θ.

2. **(„M-Schritt")** Anstelle von (5) wird nun die q_{si}-gewichtete log-Likelihood-Summe der „einfachen" Wahrscheinlichkeiten bezüglich der Parameter θ maximiert:

$$\theta_{t+1} = arg\max_{\theta} \sum_{s,i} \hat{q}_{si} \log P_s(z_i;\theta)$$

   (wobei die $\hat{q}_{si}$ als *Konstanten* behandelt werden!).

Diese Maximierungsaufgabe ist häufig *viel einfacher,* als das Ausgangsproblem: so sind in unserem Mischungsmodell-Fall die $P_s(z_i;\theta)$ beispielsweise Gauß-Verteilungen

$$P_s(z_i;\theta) = g_s \cdot N_s \exp -\frac{1}{2\sigma_s^2}(x_i - \mu_s)^2$$

und die Summe in (5) daher eine gewichtete Überlagerung *einfacher quadratischer Funktionen*: das neue Maximum θ_{t+1} läßt sich daher durch Lösen *einer linearen Gleichung* (mit der Nebenbedingung $\sum_s g_s(x, v_s) = 1$) bestimmen!

Im folgenden werden wir zeigen:

- jeder EM-Schritt (1./2.) erhöht die Log-Likelihood des Ausgangsproblems:

$$L_D(\theta_{t+1}) \geq L_D(\theta_t)$$

[2] da jetzt sowohl x_i als auch y_i genutzt werden!

- daher konvergiert
$$\lim_{t\to\infty} L_D(\theta_t)$$
gegen ein (möglicherweise lokales) Maximum der Likelihood.

Beweis: Seien $\hat{q}_{si} \geq 0$, $\sum_s \hat{q}_{si} = 1$ ansonsten beliebig gewählte Gewichtungsfaktoren. Dann gilt die Abschätzung

$$\log P(z_i;\theta) = \log \sum_s P_s(z_i;\theta) \tag{6}$$
$$= \log \sum_s \frac{P_s(z_i;\theta)}{\hat{q}_{si}} \cdot \hat{q}_{si} \tag{7}$$
$$\geq \sum_s \hat{q}_{si} \log \left(\frac{P_s(z_i;\theta)}{\hat{q}_{si}} \right) \tag{8}$$
$$= \sum_s \hat{q}_{si} \log P_s(z_i;\theta) - \sum_s \hat{q}_{si} \log \hat{q}_{si} \tag{9}$$

Beim mittleren Schritt haben wir dabei die *Jensensche Ungleichung*

$$\log \left(\sum_s c_s x_s \right) \geq \sum_s c_s \log(x_s)$$

verwendet, die für jede konvexe Funktion (hier: $\log(x)$) gilt, wenn $\sum_s \hat{c}_s = 1$ und alle $c_s \geq 0$ sind.

Die Funktion

$$-F(\hat{q};\theta) = \sum_{s,i} \hat{q}_{si} \log P_s(z_i;\theta) - \sum_{s,i} \hat{q}_{si} \log \hat{q}_{si}$$

liefert daher eine untere Schranke für die gesuchte log-Likelihood. Die EM-Schritte (1./2.) stellen sich nun als die *abwechselnde Maximierung* dieser Schranke in den beiden Argumentparametern $\hat{q}$ und θ heraus:

- Da der 2. Term in Gl. (5) nicht von θ abhängt, ist Maximierung von $-F(\hat{q};\theta)$ bezüglich θ äquivalent zum „M"-Schritt Gl. (5).
- Die Äquivalenz des „E"-Schritts Gl.(5) zur Maximierung von $-F(q;\theta)$ bezüglich $\hat{q}$ sehen wir durch Umformung von F unter Benutzung der in (5) definierten

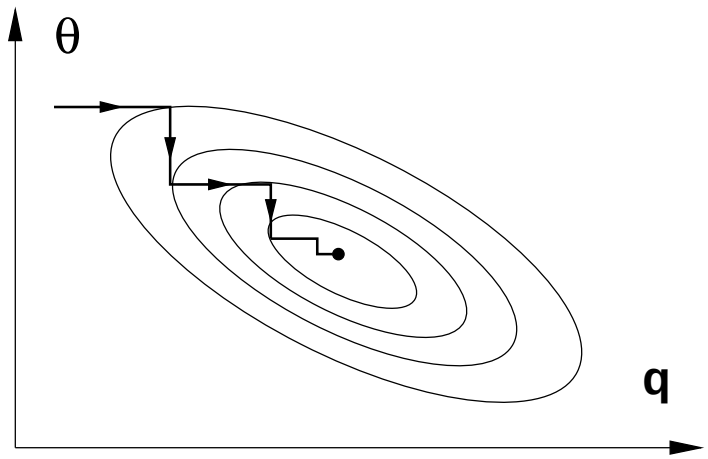
Beteiligungswahrscheinlichkeiten q_{si}:

$$-F(\hat{q};\theta_t) = \sum_{s,i} \hat{q}_{si} \log\left(\frac{q_{si} P(z_i;\theta)}{\hat{q}_{si}}\right) \tag{10}$$

$$= \sum_i \left(\sum_s \hat{q}_{si}\right) \cdot \log P(z_i;\theta_t) - \sum_i \sum_s \hat{q}_{si} \log\left(\frac{\hat{q}_{si}}{q_{si}}\right) \tag{11}$$

$$= \sum_i \log P(z_i;\theta_t) - \sum_i D_{KB}(\hat{q}_i \| q_i) \tag{12}$$

Hierbei ist $D_{KB}(\hat{q}_i \| q_i) \geq 0$ die *Kullback-Leibler*-Distanz zwischen $\hat{q}_i$ und q_i (hier bezeichnet q_i die i-te Spalte der Matrix mit Elementen q_{si}). Diese verschwindet genau für $\hat{q}_i = q_i$, d.h., die im „E"-Schritt geforderten Gewichtsfaktoren.



- Zusätzlich sehen wir, daß dann die untere Schranke $-F(q;\theta)$ die Log-Likelihood-Funktion (an der Auswerte-Stelle θ_t) „berührt".
- Die Bedeutung des EM-Verfahrens liegt darin, einen recht allgemeinen Ansatz für die (iterative) „Vervollständigung" eines nur teilweise in den beobachtbaren Daten z erfaßten Systemzustands um „verborgene innere" Zustandsvariable s liefern zu können.
- berechnet wird dabei die unter den gegebenen „äußeren" Daten z „wahrscheinlichste" (im Maximum-Likelihood-Sinne) Verteilung der „inneren" Variablen

- Die Abhängigkeit zwischen äußeren und inneren Variablen steckt dabei in der gewählten Aufspaltung

$$P(z_i;\theta) = \sum_s P_s(z_i;\theta)$$

der Wahrscheinlichkeitsdichte $P(z_i;\theta)$ der beobachtbaren Daten.

- weitere Anwendungssituationen sind
  - Vervollständigung fragmentarischer Daten
  - Hidden Markov Modelle: *Baum-Welch*-Algorithmus
  - Latente Semantische Indizierung: Auffinden Semantischer Kategorien

Products of Experts

Alternative zu additiven Mischungen: *Produkte* von Verteilungen [Hinton2002]:

$$P(z|\theta) = \frac{\prod_s P_s(z|\theta_s)}{\sum_{z'} \prod_s P_s(z'|\theta_s)}$$

- kann (im Unterschied zu additiven Mischungen) auch (gegenüber Mischungskomponenten) *schärfer lokalisierte* Verteilungen erzeugen
- erfordert nicht-gaußsche Faktoren (Produkte von Gaußverteilungen liefern nur wieder Gaußverteilungen)

Training: "naiver" Ansatz wäre Gradientenaufstieg für die Log-Likelihood:

$$\begin{aligned}\frac{\boldsymbol{\partial} \log P(z|\theta)}{\boldsymbol{\partial}\theta_s} =& \frac{\boldsymbol{\partial} \log P_s(z|\theta_s)}{\boldsymbol{\partial}\theta_s} - \sum_{z'} P(z'|\theta) \frac{\boldsymbol{\partial}}{\boldsymbol{\partial}\theta_s} \log P_s(z'|\theta_s) \\ =& \langle \frac{\boldsymbol{\partial} \log P_s(z|\theta_s)}{\boldsymbol{\partial}\theta_s} \rangle_{P_0} - \langle \frac{\boldsymbol{\partial}}{\boldsymbol{\partial}\theta_s} \log P_s(z'|\theta_s) \rangle_P\end{aligned}$$

D.h., die Richtung des steilsten Anstieg ist die Differenz zweier Erwartungswerte desselben Gradienten, einmal bezüglich einer bei den gegebenen Daten z nadelförmig konzentrierten „Verteilung" P_0, und einmal bezüglich der vom Expertenprodukt repräsentierten Verteilung $P = P(\cdot|\theta))$ („halluzinierte Daten").

Algorithmikprobleme:

- Erschöpfende Summation über alle z' scheitert an immenser Größe des Datenraums
- Näherung durch Monte-Carlo-Verfahren: Gibbs-Sampling erzeugt Markov-Sequenz von Daten, die Verteilungen $P_1, P_2, ..$ entstammen, die immer näher an der wahren Verteilung $P_\infty = P(\cdot|\theta)$ liegen.
- Jedoch entsteht weitgehende Auslöschung der Gradientenbeiträge infolge hoher Varianz der Datenvektoren.

Ausweg (Hinton2002): Minimierung der „kontrastiven Divergenz": Für die hintere Mittelung wird die wahre Verteilung $P_\infty = P(\cdot|\theta)$ durch die Verteilung P_1 *des ersten Gibbs-Schrittes* ersetzt:

$$\frac{\boldsymbol{\partial} \log P(z|\theta)}{\boldsymbol{\partial} \theta_s} \approx \langle \frac{\boldsymbol{\partial}}{\boldsymbol{\partial} \theta_s} \log P_s(z'|\theta_s) \rangle_{P_0} - \langle \frac{\boldsymbol{\partial}}{\boldsymbol{\partial} \theta_s} \log P_s(z'|\theta_s) \rangle_{P_1}$$

- da P_1 in der „richtigen Richtung" nach P hin liegt, ist eine Parameterverbesserung zu erwarten
- dadurch wird im nächsten Lernschritt P_1 wiederum näher an P liegen usf.: P_1 wirkt wie ein im Laufe des Gradientenlernens zum Ziel hin zurückweichender „Lockvogel", der die Parameter zum Ziel „zieht".
- im Gegensatz zu P ist P_1 noch eng um den Punkt z konzentriert: dadurch verschwindet das Auslöschungsproblem.

Beispiel (aus Hinton): 5x5 Clustergitter, 15 Experten (jeder eine Summe aus einer Gaußfunktion und einer uniformen Verteilung).

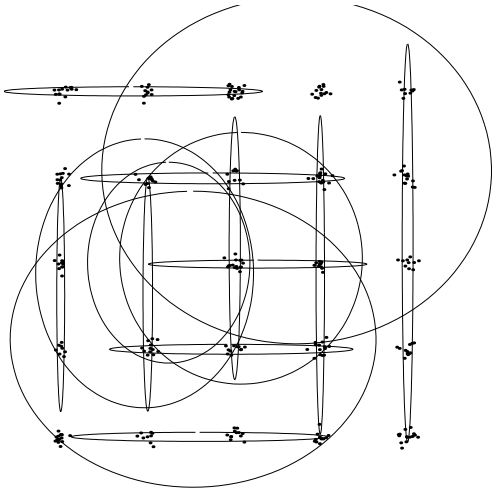


Abb.1 : PoE Repräsentation eines 5x5 „Clustergitters" durch 15 Experten. Fünf Experten enden diffus, die übrigen bilden günstige Faktoren zur Darstellung der Cluster (aus Hinton et al. 2001)

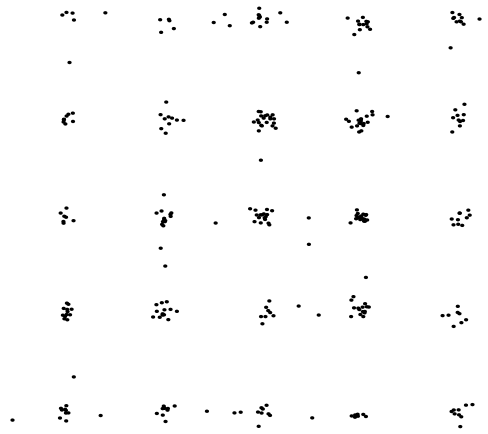


Abb.2 : Aus dem PoE Modell generierte Daten. Ein fehlender Cluster wird generalisiert. (aus Hinton)

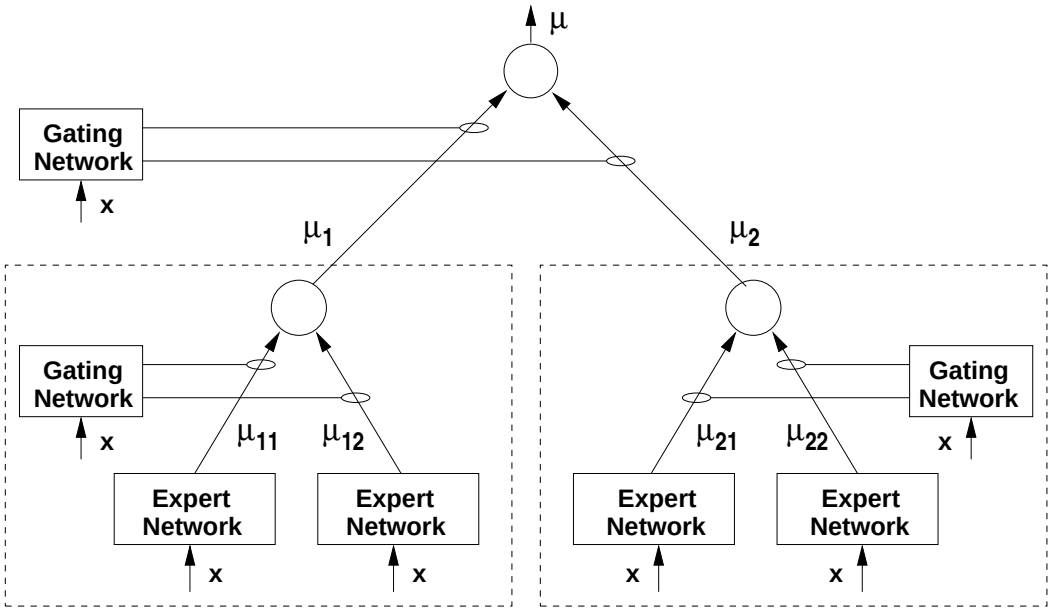
6 Hierarchisches MOE

Komplexere generative Modelle können *hierarchisch* aufgebaut sein. Zur Illustration betrachten wir den Fall einer weiteren Hierarchieebene:

obere Ebene: Auswahl eines „Zweigs" s mit Wahrscheinlichkeit $g_s(x;v)$ ($\sum_s g_s = 1$).

untere Ebene: Auswahl eines „Subzweigs" r von s mit Wahrscheinlichkeit $g_{r|s}(x;u)$.

Ausgabewert: Erzeugung Ausgabe y als Normalverteilung $P((y-f_{rs}(x;w))/\sigma_{rs})$ (wobei $P(\cdot)$ die Normalverteilungsdichte zu Varianz 1 und Mittelwert 0 bezeichnet).



Dem entspricht ein Datenerzeugungsprozeß der in Abb.6 gezeigten Struktur mit einem generativen Modell ($\theta = (u, v, w)$):

$$P(y|x;\theta) = \sum_{r,s} g_{r|s}(x;u) g_s(x;v) P\left(\frac{y - f_{rs}(x;w)}{\sigma_{rs}}\right)$$

Damit ist $P(y|x;\theta)$ wieder eine Summe über „verborgene Variable", diesmal Paare r, s. Wie zuvor, benötigt der E-Schritt die mit den a-posteriori-Schätzwahrscheinlichkeiten $\hat{q}_{rsi}$ für das Auftreten von r, s (für Datenpunkt i) gewichteten „r,s"-log-Likelihoods:

$$Q(\theta) = \sum_{r,s,i} \hat{q}_{rsi} \log\left[g_r^i g_{r|s}^i P\left(\frac{y_i - f_{rs}(x_i;w)}{\sigma_{rs}}\right)\right] \tag{13}$$

$$= \sum_{r,s,i} \hat{q}_{rsi}\left[\log g_r^i + \log g_{r|s}^i - \frac{(y_i - f_{rs}(x_i;w))^2}{2\sigma_{rs}^2} + c_{ij}\right] \tag{14}$$

wobei die a-posteriori-Wahrscheinlichkeiten

$$\hat{q}_{rsi} = P(r, s|x_i, y_i) = \frac{g_{r|s}^i g_s^i P((y_i - f_{rs}(x_i;w))/\sigma_{rs})}{P(y_i|x_i;\theta)}$$

für den E-Schritt wieder als konstante Koeffizienten gelten.

Daher zerfällt der E-Schritt in folgende drei *entkoppelte* Extremalprobleme:

$$\max_u \sum_{rsi} \hat{q}_{rsi} \log g_{r|s}(x_i; u) \tag{15}$$

$$\max_v \sum_{rsi} \hat{q}_{rsi} \log g_s(x_i; v) \tag{16}$$

$$\min_w \sum_{rsi} \hat{q}_{rsi} \frac{(y_i - f_{rs}(x_i; w))^2}{2\sigma_{rs}^2} \tag{17}$$

Beispiel: für ein Beispiel siehe etwa [JorJac94]

7 Diskussion

Man beachte, daß MOE zwar die Module selbst vorgibt, deren funktionale Spezialisierung sich jedoch unter dem Lernprozeß "emergent" herausbildet. D.h., ausgehend von vorgegebenen Strukturkomponenten dekomponiert MOE die zu lernende Transformation in eine eine gleichgroße (oder kleinere, falls einige Module ungenutzt bleiben) Anzahl von einfacherer, additiv-parallel zu kombinierender Teiltransformationen.

Literatur

[JorJac94] Jordan, MI, and Jacobs RA (1994) Hierarchical Mixture-of-Experts and the EM-algorithm. Neural Computation 6:181-214.

[Hinton2002] Training products of experts by minimizing contrastive divergence. *Neural Computation* 14(8) 1771-1800

Closed-Loop Lernarchitekturen

1 Motivation

Lernen erhält eine neue Qualität, wenn nachfolgende Trainingsbeispiele von früheren Antworten des lernenden Systems abhängen (etwa beim Lernen von Bewegungen). An die Stelle einer festen Trainingsmenge tritt dann ein *dynamisches System*[1]. Die Lernaufgabe besteht dann darin, die *Freiheitsgrade* (z.B. Neigungswinkel eines zu balancierenden Stabs; Kippwinkel und Ort eines Fahrrads) dieses externen dynamischen Systems möglichst weitgehend beherrschen zu lernen.

Wir wollen uns daher Lernarchitekturen zuwenden, die folgende wesentliche Situationsmerkmale berücksichtigen:

- Vorliegen einer „geschlossenen Schleife“ (auch als Perzeptions-Aktions-, Feedback- oder Regelungs-Schleife bezeichnet).
- Zusammenhang zwischen Aktionen und ihren Rückwirkungen über die Welt kommt ins Blickfeld. Auftreten eines zeitlichen „Credit-Assignment”-Problems.
- Möglichkeit zur aktiven Exploration.

2 Dynamische Systeme

Zur Beschreibung solcher Situationen benötigen wir das Konzept eines “Dynamischen Systems”:

Definition Dynamisches System Ein (zeitdiskretes) Dynamisches System ist gekennzeichnet durch eine *innere Zustandsgröße* $x \in X$ und eine Funktion S :

[1] dabei handelt es sich *nicht* um den Lerner, sondern um den Ausschnitt der Welt, über den die Aktionen des Lerners auf seine Eingaben rückwirken.

$X \times \mathbb{R} \mapsto X$, die eine zeitliche Entwicklung von x

$$x(t+1) = S(x(t), t)$$

festlegt.

Für zeitkontinuierliche dynamische Systeme tritt an die Stelle von Gl() eine Differentialgleichung:

$$\dot{x}(t) = S(x(t), t)$$

In beiden Fällen beschreibt die Funktion $S()$ [2] eine unendliche Schar möglicher Zeittrajektoren $x(t)$. Erst die zusätzliche Angabe eines Anfangswerts $x(0) = x_0$ wählt daraus eine bestimmte Trajektorie aus.

Anmerkung: häufig liegt eine Differentialgleichung *höherer als erster* Ordnung vor. Z.B. (Bewegung einer Masse $m = 1$ unter einer äußeren Kraft $u(t)$, geschwindigkeitsproportionale Reibung)

$$\ddot{x}(t) - \gamma \dot{x}(t) = u(t)$$

Hier hilft folgender „Trick" (Reduktion der Ordnung durch Einführung neuer Variabler):

$$\vec{x} = \begin{pmatrix} x \\ \dot{x} \end{pmatrix} =: \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Dann ist

$$\frac{d\vec{x}}{dt} = \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} x_2 \\ \gamma x_2 + u(t) \end{pmatrix}$$

Analog kann im zeitdiskreten Fall vorgegegangen werden (Einführung von $\vec{x}(t) = (x(t), x(t-1))^T$ zur „Absorption" von Termen $x(t-2)$ in Ausdrücken für $x(t)$).

Meist werden wir im folgenden mit der (oft etwas bequemer zu handhabenden) zeitdiskreten Formulierung arbeiten.

Anmerkung: der Übergang von einer Differentialgleichung zu einer diskreten Gleichung geschieht durch Diskretisierung. Im einfachsten Fall:

$$\dot{x}(t) \approx \frac{1}{\Delta t} \left(x(t + \Delta t) - x(t) \right)$$

und nachfolgende Umbenennung $x(k \cdot \Delta t) \mapsto x(k)$. Höhere Ableitungen (oder genauere Diskretisierung) führen auf zeitdiskrete Systeme höherer Ordnung, die

[2] Achtung: $S()$ ist für jeden Fall eine eigene Funktion!

durch Einführung entsprechender Zusatzvariablen wieder auf erste Ordnung gebracht werden.

Wir sind dabei i.d.R. an dynamischen Systemen interessiert, deren Verhalten durch eine zusätzliche *Stellgröße* $u \in U$ gesteuert werden kann. U taucht daher als zusätzliches Argument in $S()$ auf:

$$x(t+1) = S(x(t), u(t), t)$$

Ein einfaches Beispiel hatten wir schon in Gl() kennengelernt.

Anmerkung: ein wichtiger Spezialfall ergibt sich, wenn $S()$ *linear* von u abhängt. Das ist z.B. für alle Roboterbewegungen der Fall:

$$\mathbf{H}(\mathbf{x})\ddot{\mathbf{x}} + C(\dot{\mathbf{x}}, \mathbf{x}) + G(\mathbf{x}) = \mathbf{u}$$

($\mathbf{x} = (\theta_1 \dots \theta_n)^T$).

Manchmal ist anstelle des Zustands $x(t)$ nur eine abgeleitete Beobachtungsgröße $y = C(x)$ "von außen" sichtbar, aus der dann x erst rekonstruiert werden muß. Wir lassen die daraus resultierenden Komplikationen vorerst außer acht.

3 Das Closed-Loop Lernszenario

können wir jetzt als Rückkopplung eines (stationären[3]) dynamischen Systems formalisieren;

$$u(t) = F(x(t); w) \tag{1}$$
$$x(t+1) = S(x(t), u(t)) \tag{2}$$
$$\tag{3}$$

$F : x \mapsto u$ beschreibt dabei die Perzeptions-Aktions-Abbildung des Lerners (mit w als adaptiven Parametern), und S das dynamische Verhalten des zu steuernden Systems. Lernziel im weitesten Sinne ist das Auffinden einer Abbildung F (bzw. geeigneter adaptiver Parameter) so daß $x(t)$ „ein gewünschtes Verhalten zeigt".

[3] d.h., wir nehmen an, daß die Systemdynamik sich zeitlich nicht verändert. Ein wichtiger Fall, in dem diese Annahme nicht zutrifft, sind z.B. Drifts.

4 Typen von Lernaufgaben

Ausgehend von Gl.() können wir nun verschiedenen Closed-Loop-Lernaufgaben näher charakterisieren. Fünf große Haupttypen sind:

4.1 Stabilisierung

Hier möchte man einen unstabilen Systemzustand x^* durch Schließen der Rückkopplungsschleife stabilisieren (z.B. vertikales Balancieren eines Stabs). Detaillierter kann diese Aufgabe als Minimierung eines geeigneten Gütefunktionals $E[e, u]$ der zeitlichen Abweichungen $e(t) = x^* - x(t)$ und der ausgelösten Stellgrößenverlaufs u spezifiziert werden (im Falle eines linearen Systems $S()$ und eines quadratischen Gütefunktionals E ergibt sich für F eine lineare Abbildung als Optimallösung: *linearer Regler*).

4.2 Tracking

Läßt sich als Verallgemeinerung der Stabilisierung auffassen: der Zielpunkt $x^*(t)$ ist folgt jetzt einer vorgegebenen Bahn und man möchte $x(t)$ so steuern, daß der Trackingfehler $e(t) = x^*(t) - x(t)$ (oder wieder ein geeignetes Funktional davon und von u) möglichst gering ausfällt.

4.3 Optimale Zielpunktansteuerung

Hier möchte man innerhalb kürzestmöglicher Zeit T einen Zielpunkt x^* ansteuern. Häufig ist dabei die maximale „Größe" von u begrenzt.

4.4 Sequentielles Entscheidungsproblem

Hier steht der Aufbau einer Entscheidungskette im Vordergrund. Die einzelnen Schritte der Kette führen dabei zu „Belohnungen" oder „Bestrafungen". Ist zu Tracking verwandt: anstelle einer vorgegebenen Bahn treten (mehr oder weniger indirekte) Erfolgs/Mißerfolgsrückmeldungen.

4.5 Aktives Lernen

Hier besteht jede Entscheidung in der Generierung einer *neuen Frage* x, auf die ein „Orakel" eine Antwort y liefern soll. Die „Belohnung" besteht daher im „Informationsgehalt" der gelieferten Antwort (s. nächstes Kapitel).

4.6 Optimale Steuerung

stellt den allgemeinsten Aufgabentyp dar. Hier ist der Zustandsraum mit einer (dem Lerner bekannten oder unbekannten) „Belohnungsfunktion" $R(x)$ „belegt". Ziel ist die Steuerung einer Bahn (bzw. einer diskreten Zustandsfolge) die eine maximale Belohnung erzeugt. Hierbei sind mehrere wichtige Varianten möglich: graduelle Belohnung nach jeder Aktion, Belohnung erst nach der letzten Aktion, mit unterschiedlichem "Gedächtnis" akkumulierte Belohnungen, etc. Zusätzlich können die Aktionen u selbst Kosten erzeugen.

Alle vorigen Aufgaben lassen sich in die Form dieses Aufgabentyps bringen.

5 Fehlerbasierte Lernstrategien

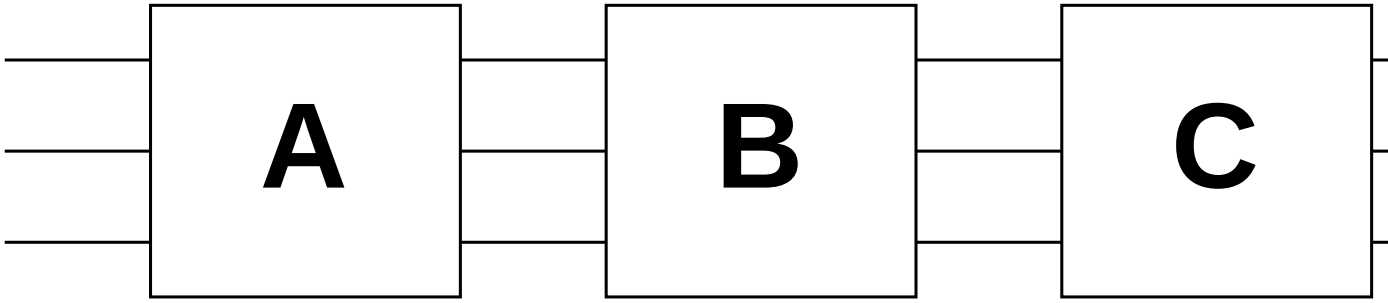
Die Vielfalt der skizzierten Lernaufgaben erfordert unterschiedliche Lernstrategien, die jeweils für unterschiedliche Aspekte einer Aufgabe geeignet sind.

Im nachfolgenden betrachten wir *fehlerbasierte Closed-Loop-Lernarchitekturen*, deren Lernschritte auf der expliziten Verfügbarkeit eines Zielfehlers basieren.

5.1 Systemidentifikation

Ziel ist hier die Gewinnung eines approximativen Modells $\hat{S}(x,u)$ für die "wahre" Dynamikfunktion $S(x,u)$ des zu kontrollierenden Systems. Faßt man die Tupel $z(t) = ((x(t),u(t)),x(t+1))$ als Trainingsbeispiele auf, so ähnelt die Aufgabe formal "normalem" überwachten Lernen. Allerdings sind aufeinanderfolgende Trainingsbeispiele z nunmehr stark korreliert (d.h., es gilt keine Unabhängigkeitsannahme mehr); außerdem kann der Lerner die Produktion neuer Beispiele über die Wahl von u *aktiv beeinflussen*.

Ein naiver Trainingsansatz ignoriert obige Unterschiede und verwendet überwachtes Lernen mit den durch Steuerungsaktionen u hervorgebrachten $z(t)$ Werten:



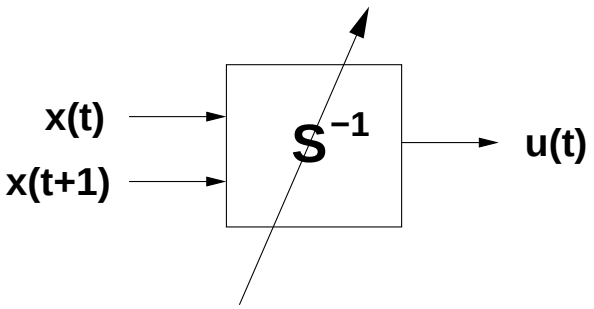
Kenntnis des approximativen Modells $\hat{S}(x,u)$ (oder auch $S(x,u)$ selbst) schafft eine *Vorhersagemöglichkeit* für das Systemverhalten und kann damit in einer Lernarchitektur andere Lernkomponenten unterstützen.

5.2 Inverse Systemidentifikation

Unmittelbar nützlicher ist in vielen Fällen die *inverse Dynamik*

$$u(t) = G(x(t), x(t+1))$$

Im Existenzfalle (F braucht nicht invertierbar sein) beantwortet sie direkt die Frage, welches u das System aus einem gegebenen Zustand $x(t)$ in einen gewünschten, neuen Zustand $x(t+1)$ führt. Ein naives Schema zum Erlernen einer Schätzung $\hat{G}(x,x')$ ergibt sich aus dem vorigen Ansatz durch Vertauschung der "Anschlüsse":



Manko dieses Ansatzes ist, daß er die Frage unbeantwortet läßt, woher man *vor Kenntnis* von G die Steuerungswerte u gewinnt, die solche Trainingszustände

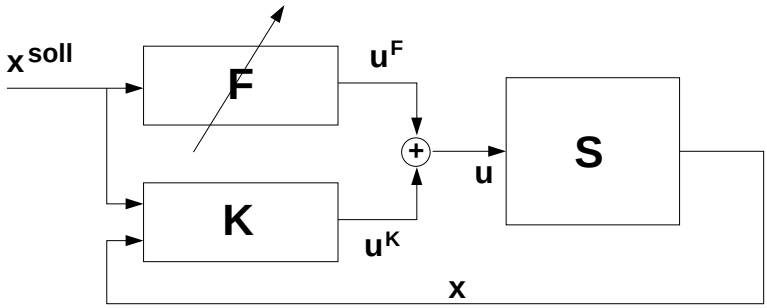
$x(t)$ hervorbringen (z.B. vertikal balancierter Stab), für die man G gerne lernen möchte.

Anmerkung: Im Falle einer *linearen* Abhängigkeit von u läßt sich das inverse Modell aus dem Vorwärtsmodell analytisch gewinnen:

$$x(t+1) = A(x(t)) + Bu(t) \Leftrightarrow u(t) = B^{-1}\left(x(t+1) - A(x(t))\right)$$

5.3 Error-Feedback-Lernen

liefert eine Teilantwort auf diese Frage: hier geht man davon aus, im Besitz eines "groben" Reglers $u = K(x)$ zu sein, der das System in die Nähe der gewünschten Zustände steuern kann ($K()$ kann z.B. ein linearer Regler sein). „Parallelgeschaltet" ist das zu trainierende System F, dessen Lernziel in der „Übernahme" der von $K()$ produzierten Steuerungswerte besteht. Ein hinreichend "repräsentationsmächtiges" F kann auf diese Weise den einfachen Regler $K()$ gewissermaßen "als Vorhut" verwenden, um allmählich ein wesentlich komplexeres Regelverhalten als $K()$ selbst zu realisieren. Der gelernte Regler liefert dann ein *inverses Modell*, das für einen gewünschten Zielzustand die benötigte Stellgröße erzeugt.

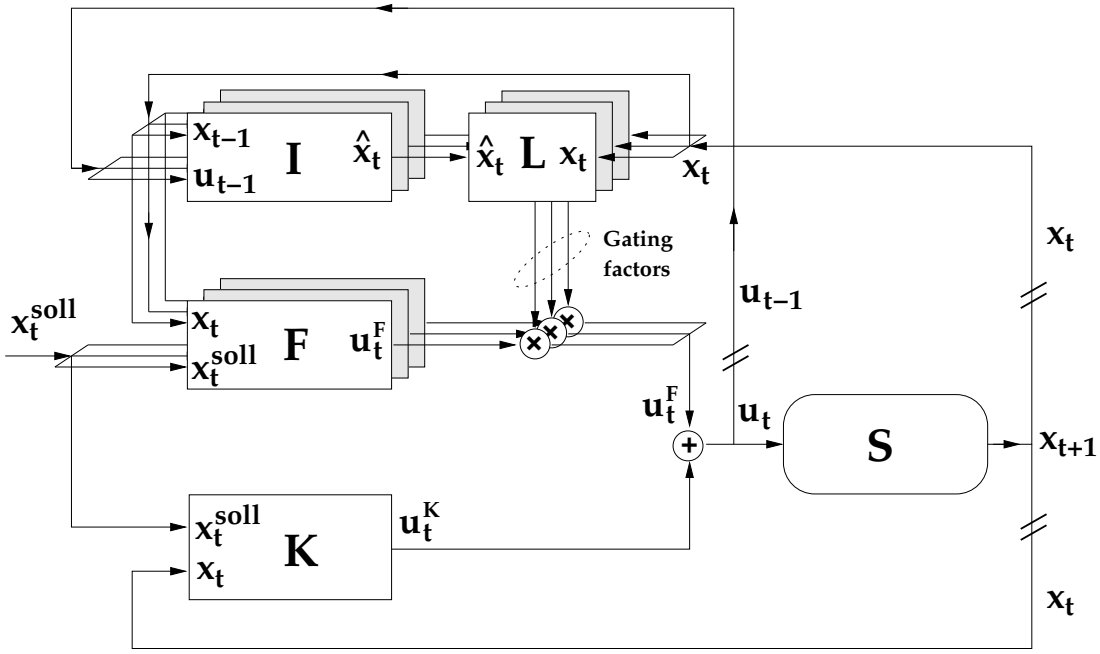


5.4 Vorwärts- und Inverses Modell kombiniert

5.5 Mixture-of-Controllers

Error-Feedback-Lernen läßt sich auch innerhalb einer Mixture-of-Experts-analogen Architektur anwenden. Dies ermöglicht das simultane Trainieren mehrerer inverser Modelle, die sich auf unterschiedliche Bereiche des Zustandsraums spezialisieren. Dazu ist jedem inversen Modell ein *Prädiktor-Modul* zugeordnet, das

eine Schätzung $\hat{x}_t$ von x_t aus dem vorangehenden $(x_{t-1}, u_t t-1)$ vorhersagt. Die Qualität dieser Vorhersage steuert die „Beteiligung" des zugeordneten inversen Modells sowohl an der nächsten Aktion u_t als auch an einem nachfolgend ausgeführtem Lernschritt.



Formal:

$$u_t^{(k)} = F^{(k)}(x_t, x_t^{(soll),w}) \tag{4}$$

$$u_t^K = K(x_t, x_t^{(soll)}) \tag{5}$$

$$\hat{x}_t^{(k)} = I^{(k)}(x_{t-1}, u_{t-1}, v) \tag{6}$$

$$\lambda_t^{(k)} = \exp(-\frac{1}{2\sigma^2}(\hat{x}_t^{(k)} - x_t)^2) \tag{7}$$

$$Z_t = \sum_k \lambda_t^{(k)} \tag{8}$$

$$u_t = u_t^K + Z_t^{-1} \sum_k \lambda_t^{(k)} u_t^{(k)} \tag{9}$$

Überwachtes Lernen kann als zu minimierende Fehlergröße die vom parallelgeschalteten Regler $K()$ erzeugten Stellkraft-Korrektur u^K benutzen. Die einzelnen

Lernschritte werden dabei mit den zuvor berechneten Beteiligungsfaktoren λ_k skaliert:

$$\delta v^{(k)} = \epsilon \lambda_t^{(k)} \frac{\partial I}{\partial v} (\hat{x}_t^{(k)} - x_t) \tag{10}$$

$$\delta w^{(k)} = \epsilon \lambda_t^{(k)} \frac{\partial F}{\partial w} \cdot u_t^K \tag{11}$$

5.6 Mosaic-Ansatz

erweitert die Mixture-of-Controllers-Architektur um ein zusätzliches Kontext-Modul je Zweig. Dieses lernt aus zusätzlichem sensorischen Kontext einen weiteren Satz multiplikativ wirkender Beteiligungsfaktoren π_k, die die Beteiligungsfaktoren λ_k gemäß der Bayes-Regel modifizieren:

$$\lambda_k \mapsto \frac{\pi_k \cdot \lambda_k}{\sum_n \pi_n \cdot \lambda_n}$$

Dadurch kann die Mixture-of-Controllers-Architektur auch ohne Information aus einem bereits begonnenen Bewegungsvorgang arbeiten. Zusätzliche Erweiterungen gestatten eine Mitlernen einer etwaigen systematischen Struktur des zeitlichen Aktivierungsmusters der einzelnen Experten-Zweige. Für nähere Einzelheiten siehe etwa [Mosaic01].

6 Zusammenfassung

Literatur

[Mosaic01] M. Haruno, D. Wolpert and M. Kawato (2001): MOSAIC-Model for Sensorimotor Learning and Control. *Neural Computation Vol. 13 Nr. 10 pp.2201-2220.*.

Aktives Lernen

1 Motivation

Aktives Lernen ist der erste Schritt auf Lernen-in-the-Loop: anstelle einer passiven Auswertung von Trainingsbeispielen fordert ein aktiver Lerner gezielt „Antworten" y auf selbsterzeugte Fragen x. Die Fragen sind dabei Gegenstand einer Optimierung in Hinblick auf Informationsgewinn.

2 Situationsmodell

Wir betrachten ein stochastisches Modell

$$P(y|w;x)$$

für die Antwort y in Abhängigkeit von einer gewählten Frage x und (zu lernenden!) Systemparametern w.

Beispiel (additives Rauschen)

$$y = F(x;w) + \eta$$

wobei $\eta \propto N(0,\sigma)$ normalverteilte Zufallsvariable ist.

Ziel ist, durch geschickte Wahl von x möglichst viel Information über den wahren Wert von w zu gewinnen. Unsicherheit über den wahren Wert von w läßt sich als eine (fiktive) Verteilungsdichte $P(w)$ beschreiben.

Die zugehörige Entropie

$$S_w = -\int P(w) \log P(w) dw$$

ist ein quantitatives Maß für die Unsicherheit unserer Kenntnis von w.

Kenntnis der Antwort y auf eine Frage x führt zu einer aktualisierten (a-posteriori) Verteilung $P(w|y)$, die nach der Bayesregel durch

$$P(w|y;x) = \frac{P(y|w;x)P(w)}{P(y;x)}$$

gegeben ist. Hier taucht überall x in der Rolle eines frei wählbaren, zusätzlichen „Kontrollparameters" auf (nur die a-priori Verteilung $P(w)$, die unseren Kenntnisstand beschreibt, ist von x unabhängig).

Unseren aufgrund der Antwort y geänderter Kenntnisstand über w wird beschrieben durch die neue Entropie

$$S_{w|y} = -\int P(w|y)\log(P(w|y))$$

die zu der a-posteri-Verteilung $P(w|y)$ gehört.

3 Maximierung des Informationsgewinns

Der damit verbundene Informationsgewinn ΔI ist dann quantifizierbar als

$$\Delta I(x) = S_w - \langle S_{w|y;x}\rangle_y$$

wobei wir durch Übergang zum Erwartungswert $\langle S_{w|y;x}\rangle_y$ berücksichtigt haben, daß y (selbst für festgelegtes x) stochastisch und damit nicht vorhersagbar ist.

Gl.() formalisiert die Suche nach einer maximal informativen Frage als Maximierungsproblem von $I(x)$.

Zur weiteren Bearbeitung notieren wir einen nützlichen Hilfssatz:

Satz Bayesregel für Entropie: Sind w, y zwei Zufallsvariable, S_y,S_w die Entropien ihrer Verteilungen, und $S_{w|y}$ bzw. $S_{y|w}$ die Entropien der entsprechend konditionierten Verteilungen, so gilt

$$\langle S_{w|y}\rangle_y + S_y = \langle S_{y|w}\rangle_w + S_w$$

Beweis: durch Hinschreiben und Anwendung der Bayesregel.

Daraus folgt:

$$\begin{aligned} I(x) &= S_w - \langle S_{w|y;x}\rangle_y && (1)\\ &= S_w - (\langle S_{y|w}\rangle_w + S_w - S_y) && (2)\\ &= S_y - \langle S_{y|w}\rangle_w && (3) \end{aligned}$$

Anschaulich interpretiert: der Informationsgewinn ist die Antwortentropie S_y, verringert um die mittlere Antwortentropie bei Kenntnis des Modells.

Eine besonders übersichtliche Situation entsteht, wenn die mittlere Antwortentropie bei Kenntnis des Modells von den Modellparametern unabhängig ist.

Dieser Fall tritt z.B. für additives gausssches Rauschen ein: Hier ist $P(y|w) = N_\sigma \exp[-(y - F(x;w)^2)/2\sigma^2]$ und daher

$$S_{y|w} = -\int P(y|w) \log P(y|w) dy \tag{4}$$

$$= \int P(y|w) \left[\frac{1}{2\sigma^2}(y - F(x;w))^2 - \log N_\sigma \right] dy \tag{5}$$

$$= \frac{\sigma^2}{2\sigma^2} - \log N_\sigma \tag{6}$$

$$= \frac{1}{2} - \log N_\sigma \tag{7}$$

Für konstantes σ hängt damit $S_{y|w}$ weder von x noch von w ab, d.h. maximaler Informationsgewinn $I(x)$ geht einher mit Maximierung der (i.d.R. x-abhängigen) Antwortentropie S_y

Falls $\sigma = \sigma(x)$ von der Frage x abhängt, ergibt sich die zu maximierende Funktion als

$$S_y + \log N_\sigma(x)$$

4 Komittee zur Schätzung der Antwortentropie

Eine geschlossene Berechnung der Antwortentropie $S_y(x)$ ist i.d.R. nicht möglich. Ein Ausweg bietet folgendes Schätzverfahren:

Die „wahre Verteilung" $P(w)$ wird durch eine endliche Anzahl $i = 1 \ldots K$ von „Samples" $w_i \sim P(w)$ approximiert. Jedes w_i läßt sich als eine infragekommende Version des gesuchten Modells (genauer: seiner Parameter) interpretieren. Der Parameterraum W wird daher auch als *Versionsraum* bezeichnet.

Die K Modelle $P(y|x;w_i)$ bilden ein „Komitee", das zu jeder potenziellen Eingabe x („Frage") K Antworten $y_i \sim P(y|x;w_i)$ liefert.

Die Entropie („Uneinheitlichkeit") $\hat{S}_y(x)$ der Kommitteantworten y_i „simuliert" dann den zu erwartenden Informationsgewinn für Frage x.

Im Falle von N diskreten Antwortmöglichkeiten ist

$$\hat{S}_y(x) = -\frac{1}{N}\sum_{i=1}^{N} \frac{n_i(x)}{\log}\frac{n_i(x)}{N}$$

wobei $n_i(x)$ die Anzahl der Komitteevoten für die $i-$te Antwort bezeichnet.

Die Optimierung der Frage kann daher in einer internen, das Kommitte nutzenden Auswerteschleife erfolgen:

Ein geeigneter Kompromiß zwischen Maximierung und Rechenaufwand muß im Auge behalten, daß $\hat{S}_y$ lediglich eine (u.U.) grobe Schätzung der wahren Antwortentropie S_y bietet.

5 Komiteebasiertes Lernen

Nach Auswahl einer geeigneten Frage x kann die eingeholte, „wahre" Antwort y zur Aktualisierung des Komitees verwendet werden. Die exakte a-posteriori-Verteilung nach der Bayesregel (Gl.**??**) wird i.d.R. nicht praktisch berechenbar sein. Mögliche Workarounds sind:

- A-posteriori-Wahrscheinlichkeit aus einem Näherungsverfahren + "Würfeln" eines neuen Komittees.
- Anwendung eines geeigneten Lernverfahrens das die Antworten der „falschliegenden" Komitteemitglieder in bessere Übereinstimmung mit y bringt und Weiterarbeit mit dem so aktualisierten Komittee.

Potenzielles Problem bei der zweiten Alternative ist eine Neigung zur Konzentration der Komitteemitglieder in der Nähe einer „Oberflächenschicht" des wahren Lösungsgebiets im Versionsraum. In einem hochdimensionalen Raum ist der dadurch erzeugte „Bias" allerdings gering, da in hohen Dimensionen die Oberfläche ohnehin stark dominiert.

6 Illustrationsbeispiel: High-Low-Game

Das High-Low-Game illustriert Arbeitsweise und mögliche Konvergenzbeschleunigung komitteebasierten aktiven Lernens im Vergleich zu passivem Lernen.

Aufgabe ist die Lokalisierung des (dem Lerner unbekannten) Sprungorts w einer Stufenfunktion $y(x;w) = \theta(x-w)$, wobei für die Sprungstelle w jeder Ort in $[0,1]$ gleichwahrscheinlich sei.

In diesem Falle ist jede „Frage" ein Wert $x \in [0,1]$, die Antworten $y \in \{0,1\}$ sind binär. Nach m Trainingsbeispielen (x^α, y^α), $\alpha = 1 \ldots m$:

$$x_L = \max\{x^\alpha \| y^\alpha = 0\} \quad (8)$$

$$x_R = \min\{x^\alpha \| y^\alpha = 1\} \quad (9)$$

$$(10)$$

Dann

- Versionsraum $W = [x_L, x_R]$.
- „ideales" Komittee: k Samples aus uniformer Verteilung in W
- „ideale Frage": maximale Antwortentropie: ergibt sich für $50-50$-Verteilung, also $x^{\alpha+1} = \frac{x_L + x_R}{2}$
- $\mapsto$ *Versionsraumhalbierung* pro Frage $\mapsto$ *exponentielle Konvergenz*
- wird approximiert für großes „ideales" Komittee.
- zum Vergleich: passives Lernen: nach N Beispielen „Restlücke" $O(1/N) \mapsto$ lediglich *lineare Konvergenz*.

Praktische Durchführung: ideales Komittee nur approximierbar: hängt von Lernregel ab. Z.B.

Fehlerkorrektur-Regel $\Delta w = \eta[y^\alpha - y]x$

Gibbs-Regel zufällige Auswahl eines Parameterwerts w aus dem Versionsraum solange, bis $y(x;w)$ *alle bisherigen* Trainingsbeispiele korrekt identifiziert.

Simulationsbeispiel: Komittees mit 2, 16, 128 Mitgliedern

7 Weitere Varianten Aktiven Lernens

Grundidee: Konzentration der Fragen in der Nähe der Klassifikationsgrenze!

- „Inversion" von Multilagen-Perzeptrons zur Lokalisierung der Klassifikationsgrenze [Hwang]
- Verwendung lokaler Modelle
- direkte Maximierung der erwarteten Verringerung des Generalisierungsfehlers

Literatur

[M. Hasenjäger and Helge Ritter] Active Learning in Neural Networks

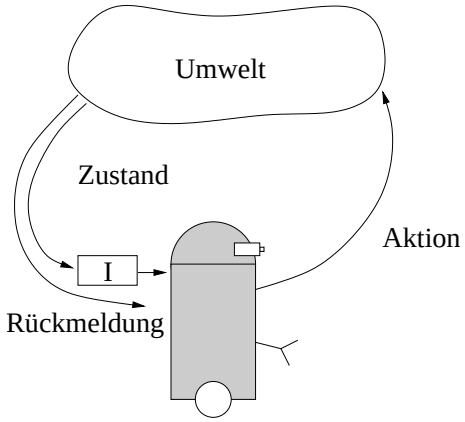
[Y. Freund, H.S. Seung, E. Shamir, N. Tishby] Selective Sampling using the Query-by-Committee algorithm. Machine Learning 28:133-168 (1997)

1 Reinforcement Learning (Verstärkungslernen)

Das Reinforcement Learning widmet sich Lernaufgaben, bei denen

- keine „Sollausgabe“ vorgegeben werden kann, sondern der Lerner selbständig eine geeignete Handlungsstrategie entwickeln muss.
- ein unmittelbarer zeitlicher Zusammenhang zwischen Aktion und Reaktion nicht besteht, sondern die Folgen einer Aktion möglicherweise erst viel später sichtbar werden. Typisches Beispiel: Labyrinth.

In diesem Fall sind klassische überwachte und gradientenbasisierte Lernverfahren nicht einsetzbar. Ein Reinforcement-Lerner ist ein Agent, der mit seiner Umwelt interagiert und durch trial-and-error versucht, seine Handlungsstrategie zu optimieren, um möglichst viel positives Feedback zu ernten.



Das zugrundeliegende Modell enthält folgende Komponenten:

- Der Zustand des Agenten in seiner Umwelt wird mittels einer *diskreten und endlichen Menge von Umgebungszuständen* $\mathcal{S}$ modelliert, wobei der Agent eventuell nur einen Ausschnitt $I(s)$ davon wahrnehmen kann. (Wir beschränken uns auf den Fall $I(s) = s$.)

- Ausgehend von seinem wahrgenommenen Zustand $s \in \mathcal{S}$ führt der Agent eine Aktion a aus einer *diskreten und endlichen Menge von Aktionen* $\mathcal{A}$ aus, z.B. $\mathcal{A}$ ={linkes/rechtes Rad vor/zurück bewegen}.

- Die Auswahl der Aktion wird durch die Policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ gesteuert, also einer Funktion die zu einem geg. Zustand s_t die nächste auszuführende Aktion $a_t = \pi(s_t)$ liefert. Die Policy ist die Handlungsstrategie des Agenten.

- Die Reaktion der Umwelt auf die Aktion wird durch einen endlichen Zustandsautomat modelliert. Der Folgezustand $s_{t+1} = \delta(s_t, a_t)$ ist also letztlich durch eine Zustandsübergangsfunktion $\delta : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ bestimmt.

- Damit der Agent auch lernen kann, erhält er ein Reward-Signal $r_t = r(s_t, a_t)$ oder $r_t = r(s_{t+1})$, dass den Erfolg oder den Misserfolg seiner bisherigen „Mission“ bewertet.
- Das Reward-Signal ist (leider) meistens neutral, also wertlos, und erst am Ende einer Aktionssequenz erhält der Agent eine positive oder negative Rückmeldung.
- Das Ziel des Agenten ist es, seine Strategie oder Policy π so anzupassen, dass er maximalen Reward von der Umgebung erhält.

Die Zustandsübergänge werden von den Gesetzmäßigkeiten der Umwelt diktiert, während die Reward-Funktion $r_t(s_t, a_t)$ die Lernaufgabe bestimmt.

Definition 1.1 Ein so modellierter Prozess wird auch *Markovscher Entscheidungsprozess (MDP)* genannt. Entscheidend dabei ist die Markov-Eigenschaft, die fordert, dass die Zustandsübergänge δ nur vom aktuellen Zustand s_t und der ausgeführten Aktion a_t abhängen, nicht jedoch von der gesamten Handlungsgeschichte $(s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0)$. Ein MDP-Iterationsschritt aus Sicht des Agenten enthält folgende Punkte:

1. Bestimme aktuellen Zustand s_t.
2. Wähle eine Handlung $a_t = \pi(s_t)$ und führe sie aus.
3. Erhalte Reward $r_t(s_t, a_t)$.
4. Umgebung geht als Reaktion auf a_t in neuen Zustand $s_{t+1} = \delta(s_t, a_t)$ über.

Beispiel 1.2 Ein Agent soll lernen, Fahrrad zu fahren.

Zustände: Neigung des Fahrrads, Geschwindigkeit

Aktionen: Drehung des Lenkers nach rechts, links

Rewards: z.B. abhängig von der Neigung des Rades,
streng negativ (Bestrafung), falls das Rad umstürzt.

Der Agent soll lernen, den kumulierten Reward zu maximieren, d.h. möglichst lange aufrecht zu fahren.

Beispiel 1.3 Ein Agent soll lernen, Schach zu spielen.

Zustände: Alle möglichen Konstellation auf dem Schachbrett (sehr sehr viele)

Aktionen: Alle möglichen Züge (auch sehr viele)

Rewards: Am Ende des Spiels erfährt der Agent ob er gewonnen (+1) oder verloren (-1) hat. Ein unerlaubter Zug wird sehr stark bestraft (-10).

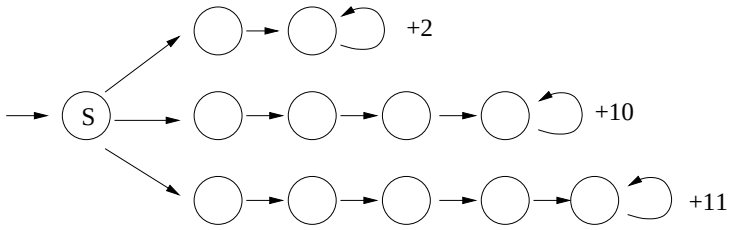
In diesem Fall hat der Agent kaum eine Chance zu lernen, wie man gewinnt, denn den positiven Reward wird er nur sehr selten erhalten... Schon eher könnte er die Schachregeln lernen, um damit die starke Bestrafung zu vermeiden...

Definition 1.4 Der *kumulierte Reward* oder auch *value function* gibt den zu erwartenden Reward an, wenn der Agent ausgehend von einem Zustand s_t seine Policy π verfolgt: $V^\pi : \mathcal{S} \to \mathbb{R}$.

Unterschiedliche Definitionen von V^π beziehen zukünftige Rewards in unterschiedlicher Weise ein:

discounted reward:	$V^\pi = \sum_{t=0}^{\infty} \gamma^t r_t$	exponentiell abnehmende Gewichtung zukünftiger Rewards
finite-horizon reward:	$V^\pi = \sum_{t=0}^{h} r_t$	gleiche Gewichtung zukünftiger Rewards, endlicher Zeithorizont
average reward:	$V^\pi = \lim_{h\to\infty} \frac{1}{h} \sum_{t=0}^{h} r_t$	gleiche Gewichtung aller zukünftigen Rewards

Beispiel 1.5 Wir betrachten folgenden Zustandsübergangsgraphen:



$\gamma = 0.9, h = 4$

Eine Aktionsauswahl besteht eigentlich nur am Anfang. Hat der Agent sich einmal für einen Zweig entschieden, kann er immer nur eine Aktion ausführen, die ihn vorwärts bringt. Erst am Ende jeden Zweiges gibt es einen Reward – dies sind sozusagen die Zielzustände (mit unterschiedlicher Attraktivität).

discounted reward: $$V^\pi = \sum_{t=0}^{\infty} \gamma^t r_t = \gamma^n \sum_{t=0}^{\infty} \gamma^t R = \frac{R\gamma^n}{1-\gamma} \approx \{16; 66; 65\}$$

finite-horizon reward: $$V^\pi = \sum_{t=0}^{4} r_t = \{4; 0; 0\}$$

average reward: $$V^\pi = \lim_{h\to\infty} \frac{1}{h} \sum_{t=0}^{h} r_t = \{2; 10; 11\}$$

Je nach Wahl der Funktion für den kumulierten Reward V^π ergibt sich eine andere optimale Strategie π^*!

Im folgenden werden wir nur den discounted Reward betrachten, weil damit am einfachsten gerechnet werden kann. Der Faktor $\gamma \in (0,1)$ bestimmt die exponentiell abnehmende Gewichtung zukünftiger Rewards, d.h. man möchte lieber heute positive Rewards sammeln als irgendwann in der Zukunft. Wir können den discounted Reward auch rekursiv berechnen:

$$\begin{aligned} V^\pi(s_0) &:= \sum_{t=0}^{\infty} \gamma^t \, r_t = r_0(s_0, \pi(s_0)) + \gamma\Big(r_1(s_1, \pi(s_1)) + \gamma\big(r_2(s_2, \pi(s_2)) + \dots\big)\Big) \\ &= r_0(s_0, \pi(s_0)) + \gamma V^\pi(s_1) \\ &= r_0(s_0, \pi(s_0)) + \gamma V^\pi\big(\delta(s_0, \pi(s_0))\big) \end{aligned} \tag{1.1}$$

Der discounted reward im aktuellen Zustand s wird dabei ausgedrückt als Summe des Rewards $r(s, \pi(s))$ aufgrund der Aktion $a = \pi(s)$ im aktuellen Zeitschritt und des zu erwartenden Rewards $V^\pi(s')$ im Folgezustand.

Gleichung (1.1) gilt natürlich für alle möglichen Zustände $s_0 \in \mathcal{S}$, so dass wir es eigentlich mit einem Gleichungssystem von $|\mathcal{S}|$ Gleichungen zu tun haben. Dieses System kann nach den Unbekannten $V^\pi(s), s \in \mathcal{S}$ aufgelöst und damit die value-function V^π zu gegebener Policy π bestimmt werden.

1.1 Nicht-Deterministische Prozesse

Sowohl die Zustandsübergangsfunktion der Weltmodellierung als auch die Policy des Agenten können (unabhängig voneinander) stochastisch sein, d.h. einem Zufallsgesetz folgen. Wir betrachten beide Möglichkeiten zunächst separat.

Stochastische Policy Anstatt deterministisch immer dieselbe Aktion $a = \pi(s)$ auszuwählen, ordnet die Policy π im Zustand s nun jeder Aktion $a \in \mathcal{A}$ eine Wahrscheinlichkeit $P(a|s)$ zu. Entsprechend dieser Wahrscheinlichkeitsverteilung $P(\cdot|s)$ wird die tatsächlich auszuführende Aktion ausgewählt. Die value-function muss nun den Erwartungswert der Rewards betrachten:

$$\begin{aligned} V^\pi(s) &:= \sum_{t=0}^{\infty} \gamma^t \, E(r_t) = E\big(\sum_{t=0}^{\infty} \gamma^t \, r_t\big) \overset{(1.1)}{=} E(r(s, \pi(s))) + \gamma E\big(V^\pi(\delta(s, \pi(s)))\big) \\ &= \sum_a P(a|s) \cdot r(s,a) \; + \; \gamma \sum_a P(a|s) \cdot V^\pi(\delta(s,a)) \end{aligned} \tag{1.1'}$$

Stochastische Zustandsübergänge Die Reaktion der Umwelt auf Aktionen des Agenten wird nun durch einen nicht-deterministischen Zustandsautomat beschrieben. Auch hier ordnet die Zustandsübergangsfunktion δ jedem Zustands-Aktions-Paar (s,a) jetzt eine Wahrscheinlichkeitsverteilung $P(s'|s,a)$ zu, die angibt mit welcher Wahrscheinlichkeit der Folgezustand s' erreicht wird. Die value-function muss

wieder den Erwartungswert der Rewards betrachten, diesmal gewichtet über alle möglichen Folgezustände s':

$$V^\pi(s) := E\Big(\sum_{t=0}^{\infty} \gamma^t\, r_t\Big) \overset{(1.1)}{=} r(s,a) + \gamma \sum_{s'} P(s'|s,a) \cdot V^\pi(s') \qquad a = \pi(s) \qquad (1.1")$$

Wir werden i.d.R. nur stochastische Übergangsfunktionen δ, nicht jedoch stochastische Policies betrachten. Man beachte, dass sich die deterministischen Gleichungen als Grenzfall der stochastischen ergeben, wenn man die Wahrscheinlichkeitsverteilungen $P(\cdot|s,a)$ bzw. $P(\cdot|s)$ für genau einen Folgezustand s' bzw. für genau eine Aktion a Eins sind.

Für die optimale value-function $V^* \equiv V^{\pi^*}$ gelten die *Bellman'schen Optimalitätsgleichungen:*

$$V^*(s) = \max_a \Big(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \cdot V^*(s')\Big) \quad \forall s \in \mathcal{S}$$

Diese Gleichungen sind wieder rekursiv definiert und beschreiben lediglich, dass die optimale value-function in jedem Zustand s den maximal erreichbaren Reward angibt.

1.2 Policy Iteration

Das Lernproblem besteht nun aber darin, die optimale Policy zu lernen. Die folgende Gleichung liefert eine Iterationsvorschrift um eine bisherige Policy π mit zugehöriger value-function V^π (mittels einer greedy-Strategie) zu verbessern:

$$\pi'(s) := \arg\max_a \Big(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \cdot V^\pi(s')\Big)\,. \qquad (1.2)$$

Die neue Policy π' hat für jeden Zustand s eine höhere Bewertung $V^{\pi'}(s)$. *Wir erhielten genau die alte Policy $\pi' = \pi$ und die alte value-function V^π, wenn wir $a = \pi(s)$ wählten. Da aber über alle möglichen Aktionen a maximiert wird, verbessert sich die Policy π' bzw. die zugehörige value-function $V^{\pi'}$.* Wir erhalten damit den folgenden Algorithmus, um die optimale Policy π^* zu lernen:

```
Initialisiere die Policy π' zufällig
loop
     π := π'
     Berechne V^π durch Lösung von (1.1-1.1'')
     Verbessere die Policy entsprechend (1.2)
until π = π'
```

Die Berechnung der value-function V^π über das Gleichungssystem (1.1') *ist u.U. sehr aufwendig.* Das folgende iterative Verfahren ermöglicht eine Approximation von V^π:

`Initialisiere` $k = 0,\ V_k^\pi = 0$
`loop`
$$V_{k+1}^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) \cdot V_k^\pi(s')$$
$k \to k + 1$
`until` $V_{k+1}^\pi \approx V_k^\pi$

1.3 Value Iteration

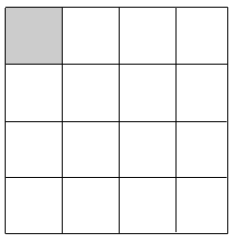
Alternativ kann man direkt die value-function optimieren:

`Initialisiere` $k = 0, V_k(s) = 0$
`loop`
`Berechne` $Q_k(s, a) := r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \cdot V_k(s')$
`Update` $V_{k+1}(s) := \max_a Q_k(s, a)$
$k \to k + 1$
`until` Änderungen von V sind klein genug

Es kann gezeigt werden, dass dieser Algorithmus tatsächlich zur optimalen value-function V^* konvergiert. Die optimale Policy kann von dieser einfach abgeleitet werden:

$$\pi^*(s) := \arg\max_a \Big(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \cdot V^*(s') \Big) . \tag{1.3}$$

Beispiel 1.6
Zum Vergleich der beiden Verfahren Policy- und Value-Iteration schauen wir uns das typische Reinforcement-Problem an: Ein Agent kann sich auf einem endlichen Gitter fortbewegen, indem er eine der Aktionen *hoch, runter, rechts, links* auswählt. Am Rand des Gitters sollen „verbotene“ Aktionen unberücksichtigt bleiben, d.h. der Agent erhält keinen negativen Reward, bewegt sich aber auch nicht fort. Ziel des Agenten soll es sein, eines der grauen Felder zu erreichen (und dort zu bleiben), so dass nur bei Erreichen dieser Felder ein positiver Reward von Eins erteilt wird.



Policy Iteration

$\xrightarrow{\pi_0}$

←	←	←	←
→	→	→	↑
↓	←	←	←
→	→	→	↑

$\xrightarrow{V^{\pi_0}}$

100	100	90	81
53.14	59.05	65.61	72.9
0	0	0	0
0	0	0	0

$\xrightarrow{\pi_1}$

←	←	←	←
↑	↑	↑	↑
↑	↑	↑	↑
→	→	→	↑

$\xrightarrow{V^{\pi_1}}$

100	100	90	81
100	90	81	72.9
90	81	72.9	65.61
43.05	47.83	53.14	59.05

$\xrightarrow{\pi_2}$

←	←	←	←
↑	↑	↑	↑
↑	↑	↑	↑
↑	↑	↑	↑

$\xrightarrow{V^{\pi_2}}$

100	100	90	81
100	90	81	72.9
90	81	72.9	65.61
81	72.9	65.61	59.05

$\xrightarrow{\pi_3=\pi_2} \dots$

Nach jeder Verbesserung der Policy muss hier die auf der neuen Strategie basierende Wertefunktion exakt berechnet werden. Dafür ist jeweils eine größere Menge an Iterationen erforderlich.

Value Iteration Im Gegensatz dazu, benötigt Value Iteration pro Iterationsschritt nur wenige Rechenschritte, um die neue Wertefunktion zu bestimmen:

$\xrightarrow{V_0}$

10	10	0	0
10	0	0	0
0	0	0	0
0	0	0	0

$\xrightarrow{V_1}$

19	19	9	0
19	9	0	0
9	0	0	0
0	0	0	0

$\xrightarrow{V_2}$

27.1	27.1	17.1	8.1
27.1	17.1	8.1	0
17.1	8.1	0	0
8.1	0	0	0

$\xrightarrow{V_3}$

34.39	34.39	24.39	15.39
34.39	24.39	15.39	7.29
24.39	15.39	7.29	0
15.39	7.29	0	0

$\xrightarrow{V_4}$

40.95	40.95	30.95	21.95
40.95	30.95	21.95	13.85
30.95	21.95	13.85	6.56
21.95	13.85	6.56	0

$\xrightarrow{V_5}$

46.86	46.86	36.86	27.86
46.86	36.86	27.86	19.76
36.86	27.86	19.76	12.47
27.86	19.76	12.47	5.9

$\xrightarrow{V_6}$

52.17	52.17	42.17	33.17
52.17	42.17	33.17	25.07
42.17	33.17	25.07	17.78
33.17	25.07	17.78	11.22

$\xrightarrow{V_7}$

56.95	56.95	46.95	37.95
56.95	46.95	37.95	29.85
46.95	37.95	29.85	22.56
37.95	29.85	22.56	16

$\rightarrow \dots \xrightarrow{V_\infty}$

100	100	90	81
100	90	81	72.9
90	81	72.9	65.61
81	72.9	65.61	59.05

Policy Iteration benötigt zwar weniger Iterationen, ist aber pro Iteration langsamer als Value Iteration. Es ist daher nicht klar, welcher Algorithmus der bessere ist, und es gibt Beispiele, für die der eine oder andere Algorithmus schneller konvergiert. Policy Iteration und Value Iteration sind Beispiele von *dynamischer Programmierung. Sie benutzen die Zustandsübergangsfunktion $\delta(s, a)$ und die Reward-Funktion $r(s, a)$.*

1.4 Q-Lernen

Policy und Value Iteration benutzen sowohl die Reward-Funktion $r(s,a)$ als auch die Wahrscheinlichkeiten für Zustandsübergänge $P(s'|s,a)$, um die optimale Policy zu finden. Diese Daten sind i.d.R. jedoch nicht ad hoc verfügbar und der Agent muss sie durch Exploration der Umwelt erst lernen. Das Q-Lernen umgeht dieses Modell-Wissen durch Nutzung der Q-Funktion (oder auch action-value function genannt):

$$Q^\pi(s,a) := E\big(V^\pi(s) \mid a\big) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \cdot V^\pi(s')$$

Sie gibt den erwarteten Reward an, wenn man in Zustand s **die Aktion a auswählt** sowie **im weiteren** der Policy π folgt. *Die ursprüngliche value function $V^\pi(s)$ wird also zu einer zwei-parametrigen Funktion $Q^\pi(s,a)$ „aufgebläht", mit dem Erfolg, dass die auszuführende Aktion a im aktuellen Zustand s von der ansonsten verfolgten Policy (und der damit verbundenen value-function V^π) entkoppelt wird. Man kann also im aktuellen Zustand s nun **alle** möglichen Aktionen betrachten. Die „alte", von a unabhängige value function erhält man aus Q^π mittels der Beziehung $V^\pi(s) = Q^\pi(s,\pi(s))$, also durch Verwendung der Aktion $a = \pi(s)$.* Analog zu Gleichung (1.3) kann die zu Q gehörige Policy π bestimmt werden:

$$\pi(s) = \arg\max_a Q(s,a)\,. \tag{1.4}$$

Mit der Beziehung $V^*(s) = \max_a Q^*(s,a)$ kann man die Bellman'schen Optimalitätsgleichungen für die optimale Q-Funktion Q^* aufstellen:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \cdot \underbrace{\max_{a'} Q^*(s',a')}_{V^*(s')}$$

1.4.1 Basisalgorithmus des Q-Lernen

Diese rekursiven Definition(en) von Q bilden im Prinzip die Grundlage für den Q-Lernalgorithmus. Wir müssen jedoch noch die Weltmodellierung in Form der Übergangswahrscheinlichkeiten $P(s'|s,a)$ loswerden. Aber wir können den Folgezustand s' aufgrund einer ausgewählten Aktion a ja wieder beobachten! Der Lernalgorithmus wird dadurch zu einem (inter)aktiven Lernprozess in der Umwelt:

`Initialisiere` $k = 0$, $Q_k(s,a)$ `zufällig, aber klein`
`loop`
    `Bestimme aktuellen Zustand` s
    `Führe eine Aktion` a `aus (zufällig oder entsprechend` Q_k (1.4)`)`
    `Erhalte Reward` $r(s,a)$ `bzw.` $r(s,a')$
    `Bestimme den Folgezustand` s'
    `Update` $Q_{k+1}(s,a) := r(s,a) + \gamma \max_{a'} Q(s',a')$
    $k \to k+1$
`until` Änderungen von Q sind klein genug

Das Tupel $\langle s, a, r, s' \rangle$ beschreibt die *Erfahrung*, im Zustand s bei ausgeführter Aktion a im Zustand s' zu landen und den Reward r erhalten zu haben. Das Update wird so gemacht, dass man den höchstmöglichen Q-Wert $\max_{a'} Q(s', a')$ im Folgezustand verwendet (greedy).

neo: QLearn-Demo

1.4.2 Temporal Difference Learning: TD(0)

Eine zentrale Idee des Reinforcement Lernens ist das *temporal difference*-Lernen, kurz TD-Lernen. Es kombiniert zwei wichtige Aspekte:

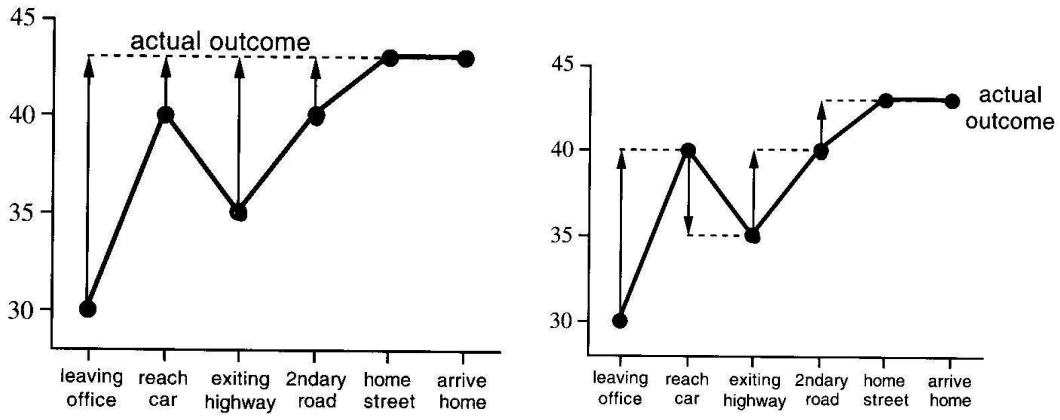
- das direkte Lernen aus den eigenen Erfahrungen in Abwesenheit eines Modells des zu lernenden Problems und
- die Fähigkeit, noch vor Erreichen des Ziels die aktuellen Werte auf der Basis anderer, ebenfalls noch nicht perfekt gelernter Werte, zu verbessern (das sog. *bootstrapping*).

Die Updateformel für TD-Lernen ist nur mithilfe der Value-Funktion $V(s_t)$ definiert:

$$V(s) := V(s) + \alpha\Big(r + \gamma V(s') - V(s)\Big)\,, \tag{1.5}$$

$\alpha \in [0, 1]$ ist dabei die Lernschrittweite.

Beispiel 1.7 Ein Dozent fährt aus der Uni mit dem Auto nach Hause. Beim Verlassen seines Büros registriert er die Uhrzeit, den Wochentag und sonstige relevante Daten. An einem Freitag verläßt er sein Büro um 18 Uhr und schätzt, daß er 30 Minuten bis zu seinem Haus brauchen wird. Beim Betreten des Parkplatzes um 18:05 Uhr beginnt es aber zu regnen, so daß er seine Schätzung auf eine Gesamtreisezeit von 40 Minuten erhöht. Eine Viertelstunde später (also um 18:20 Uhr) hat er die Autobahn verlassen und ist dabei sehr glatt durchgekommen – er revidiert die geschätzte Reisezeit auf 35 Minuten. Dummerweise setzt sich auf der Landstraße ein nicht überholbarer Trecker vor ihn, so daß er erst um 18:40 Uhr in seine Straße einbiegt und um 18:43 Uhr dann glücklich zu Hause ankommt.



Wie aus Gleichung 1.5 ersichtlich, ist die Wahl der Aktionen bei TD-Lernen nicht festgelegt. Hier gibt es zwei prinzipielle Möglichkeiten:

- on-policy-Lernen: Die Policy π wird sowohl zur Verhaltensgenerierung als auch zur Bestimmung der Wertefunktion Q^π im Folgezustand verwendet:

  $$Q(s,a) = Q(s,a) + \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$$

  Die Aktion a', die im Folgezustand s' ausgewählt wird, wird also mittels der Policy bestimmt: $a' = \pi(s')$. Da hierbei das Tupel (s,a,r,s',a') verwendet wird, heißt dieser Algorithmus *Sarsa*.

- off-policy-Lernen: Die Policy π wird nur zur Verhaltensgenerierung im aktuellen Zustand verwendet. Der Wert des Folgezustandes wird aber mittels einer anderen Stratgie (typischerweise greedy) bestimmt:

  $$Q(s,a) := Q(s,a) + \alpha\Big(\underbrace{r + \gamma \max_{a'} Q(s',a')}_{Q'(s,a)} - Q(s,a)\Big) \tag{1.6}$$

  Hier erkennt man den Basisalgorithmus des Q-Lernen wieder, der über alle möglichen Aktionen im Folgezustand maximiert. Im Prinzip wird in der Praxis nur dieser Algorithmus verwendet.

Anders als in Kapitel 1.4 wird der aktuelle Wert hier jeweils nur in Richtung des neuen Wertes verändert - man springt nicht mehr sofort zum neuen Wert, sondern erhält einen fließenden Übergang zu diesem Wert. Dies ist insbesondere dann von Vorteil, wenn der Algorithmus momentan einer suboptimalen Strategie folgt. Die Lernschrittweite $\alpha \in [0,1]$ bestimmt, wie groß der Lernschritt in Richtung des neuen Q-Wertes $Q'(s,a)$ sein soll. Für $\alpha = 1$ erhält man wieder die obige Q-Lernregel, für $\alpha = 0$ lernt man gar nichts.

Damit lautet der Algorithmus:

`Initialisiere` $k = 0, Q_k \equiv 0$
`loop`
    `Beobachten aktuellen Zustand` s
    `Wähle Handlung` $a = \pi(s)$
    `Erhalte Reward` r `und beobachte Folgezustand` s'
    `Update` $Q_{k+1}(s,a) := Q_k(s,a) + \alpha\big(r + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\big)$
`until` Änderungen von Q sind klein genug

Satz 1.8 Unter folgenden Voraussetzungen konvergiert Q gegen die optimale Q-Funktion Q^*:

- $|r(s,a)| < \infty \qquad \forall s,a,$
- $0 \leq \gamma < 1,$
- Jedes (s,a)-Paar wird unendlich oft besucht.

1.4.3 Temporal Difference Learning: TD(λ)

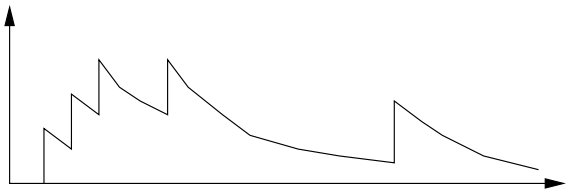
Die Lernregel TD(0) berücksichtigt nur eine einzige Erfahrung (s_t, a_t, r_t) des Agenten:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\Big(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\Big) .$$

Damit führt jeder Iterationsschritt des Agenten nur zu einem einzigen Update der Q-Funktion. Falls der Reward stark verzögert kommt, baut sich die richtige Q-Funktion nur sehr langsam auf. Sinnvoller ist es daher bei einem Update auch vorherige Schritte – also den gesamten Weg zum Ziel – zu berücksichtigen. Dazu führen wir die *Eignung (eligibility)* eines (s, a)-Paares ein:

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a) + 1 & \text{falls } (s, a) = (s_t, a_t) \\ \gamma\lambda e_{t-1}(s, a) & \text{sonst} \end{cases}$$

Dabei bestimmt der Faktor λ, wie stark die vorherigen Zustände gewichtet werden sollen. Falls ein Paar (s, a) im Zeitschritt t besucht wird, erhöht sich die Eignung um 1, sonst nimmt sie exponentiell ab:

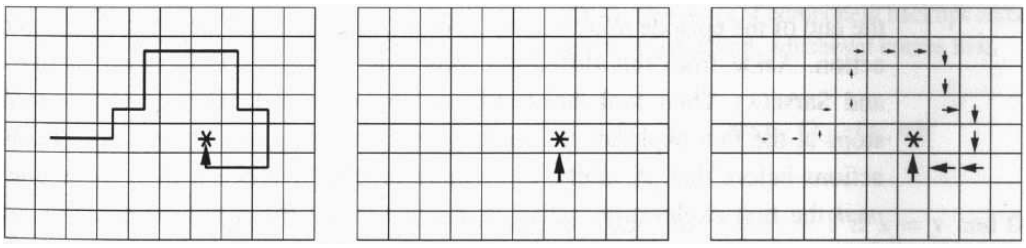


Je höher der Wert $e(s, a)$, desto öfter wurde das Paar besucht, desto weniger in der Vergangenheit liegen diese Besuche und desto wichtiger ist offenbar sein Anteil am erzielten Erfolg (Reward). Dementsprechend nutzen wir folgende Updateregel:

$$\begin{aligned} Q(s, a) &:= Q(s, a) + \alpha \cdot e(s, a) \cdot \Delta(s, a, s') \qquad \text{für alle } (s, a) \in \mathcal{S} \times \mathcal{A} \\ \Delta(s, a, s') &= r + \gamma \max_{a'} Q(s', a') - Q(s, a) \end{aligned}$$

Die alte Lernregel erhalten wir mit $e(s, a) = \delta_{ss_t}\delta_{aa_t}$, also für $\lambda = 0$.

Beispiel 1.9



durchlaufener Pfad | 1-Step-Update: TD(0) | TD(λ)

1.4.4 Exploration vs. Exploitation

Exploration: meist zufällige Erforschung des Zustandsraumes
Exploitation: optimale Ausnutzung des bisher gewonnenen Wissens (Approximation von V oder Q) zur Bestimmung der Policy.

- *Zu ausgiebiges Erforschen bedeutet, dass der Agent auch nach langem Lernen noch ziellos im meist sehr großen Zustandsraum umherwandert. Dadurch werden auch Bereiche intensiv untersucht, die für die Lösung der Aufgabe gar nicht relevant sind.*
- *Zu frühes Ausbeuten der gelernten Approximation der Q-Funktion bewirkt möglicherweise, dass sich ein suboptimaler Pfad durch den Zustandsraum etabliert, weil dieser zufällig zuerst gefunden wurde und gute Ergebnisse lieferte. Die optimale Lösung wird jedoch nicht mehr gesucht.*

Beispiel 1.10 Mehrere einarmige Banditen, deren Ausschüttung und Erfolgsquote zufällig sind. Wie lange soll man bei anderen Automaten nach höheren Ausschüttungsquoten suchen und ab wann soll man sich auf einen Automaten konzentrieren?

Lösung: teilweise zufällige Auswahl der Aktionen.

ε-greedy: Wähle typischerweise die optimale Aktion a^* aus, aber mit Wahrscheinlichkeit $\frac{\varepsilon}{|\mathcal{A}|}$ eine zufällige andere Aktion a.

Boltzmann-Gewichtung: Wähle die Aktionen a zufällig entsprechend der Boltzmann-Verteilung aus:

$$\pi(s,a) = \frac{\mathrm{e}^{\beta(Q(s,a)-Q(s,a^*))}}{\sum_{a'} \mathrm{e}^{\beta(Q(s,a')-Q(s,a^*))}} \quad \text{mit } a^* = \arg\max_a Q(s,a)$$

Die Lernparameter $\varepsilon \in [0,1]$ und $\beta \in [0,\infty)$ sollten langsam abnehmen.

- $\varepsilon = 1$ bzw. $\beta = 0 \Rightarrow$ Gleichverteilung
- $\varepsilon = 0$ bzw. $\beta = \infty \Rightarrow$ greedy

Der wichtigste Unterschied zwischen ε-greedy und Boltzmann-Exploration ist die Auswahl der auszuführenden Aktion im Falle der Exploration: während bei ε-greedy jede suboptimale Aktion, also auch die bisher am schlechtesten bewertete, die gleiche Chance hat, ausgesucht zu werden, werden suboptimale Aktionen bei der Boltzmann-Exploration mit einer Wahrscheinlichkeit ausgewählt, die mit ihrem momentanen Wert korreliert.

1.5 Generalisierung: Kontinuierliche Zustands- und Aktionsräume

Alle Algorithmen sind von diskreten Zustands- und Aktionsräumen ausgegangen und speichern die Funktionen $V(s)$ und $Q(s,a)$ in einer Tabelle. Dies ist sehr ineffizient und im Falle von kontinuierlichen Zustands- und Aktionsräumen nicht mehr praktikabel.

Lösung: Funktionsapproximation für $V(s)$ bzw. $Q(s,a)$.
allg. Ansatz eines Approximators: $\hat{Q}_w(s,a)$.
Die Gewichte w haben je nach Wahl des Approximators unterschiedliche Bedeutung:

- Splines/Polynome: Stützstellen
- Multilagen-Perzeptron (MLP): Gewichte im Netz
- SOM, LLM, PSOM: Stützstellen

Ein guter Funktionsapproximator sollte folgende Eigenschaften haben:

- Gute Generalisierung, d.h. Interpolation und Extrapolation bei unbekannten Zuständen und Aktionen.
- Gute Approximation der tatsächlichen Q-Funktion. Dazu muss die Auflösung der Q-Funktion genügend hoch sein, kann aber in unterschiedlichen Bereichen des Raumes $\mathcal{S} \times \mathcal{A}$ variieren.
- Geringer Speicherbedarf
- Effiziente Berechnung der optimalen Aktion: $\arg\max_{a \in \mathcal{A}} \hat{Q}_w(s,a)$.
  Bei kontinuierlichem Aktionsraum $\mathcal{A}$ stellt dies ein nichtlineares Optimierungsproblem dar, mit all seinen Problemen (lokale Minima = suboptimale Aktionsauswahl)
- Effiziente Updates der Q-Funktion.
- Lokale Lernverfahren.

Einige dieser Eigenschaften widersprechen einander und man muss einen geeigneten Mittelweg finden, der alle Eigenschaften ausreichend berücksichtigt.

6. Partially Observable Markov Decision Processes

Bisher: Zustand war voll zugänglich

Aber: tatsächliche Zustand oft nicht direkt zugänglich, sondern nur die Messg von Beobachtgsgrößen indirekt „abschätzbar". Beispiel Labyrinth-Navigation

Lerner erhält in Zustand s eine Observation o mit W.keit $P(o|s,a)$

	Einfluß auf Zustandsübergang (Aktionen)	
	✗	✓
Markov Modelle ✓	Markov-Ketten chain	Markov Entscheidgsprozeß MDP
Zustand observierbar ✗	HMM	POMDP

POMDP-Modell

Lerner muss (theoretisch) gesamte Historie von Aktionen und Beobachtungen speichern, um sinnvolle Entscheidgen treffen zu können. Praktisch trifft aber schon W.keitsverteilg über Zustand: $b(s) = P(s)$ (Belief State)

Update des Belief-State:

$$b'(s') = P(s'|o,a,b) = \frac{P(o|s',a,b)\cdot P(s'|a,b)}{z = P(o|a,b)}$$

$$= \frac{1}{z} P(o|s',a) \sum_{s\in S} P(s'|a,s,b)\cdot P(s|a,b)$$

$$= \frac{1}{z} P(o|s',a) \sum_{s} \underbrace{P(s'|a,s)}_{\text{state-transition prob.}} \cdot \underbrace{b(s)}_{\text{old belief}} \qquad \tau(b,a,o) \equiv b'$$

Hängt tatsächlich nur von $b(s)$ ab. POMDP kann als MDP auf dem nun kontinuierlichen Belief-Raum (Raum der W.verteilgen über S) aufgefasst werden, mit τ als Zustandsübergangsfkt und

$$r(b,a) = \sum_s b(s)\cdot R(s,a) \quad \text{als Reward-Fkt.}$$

Bellmann-Gleichung:

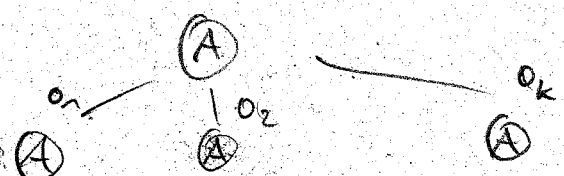
$$V^*(b) = \max_{a \in A} \left(r(b,a) + \gamma \sum_{o \in O} P(o|b,a)\, V^*(\underbrace{\tau(b,a,o)}_{b'_o}) \right)$$

Lösung per Value Iteration auf nun unendliche Belief-Raum

Endlicher Horizont: Policy ist zeitabhängig. Wieviel Zeit bleibt oder kostet?

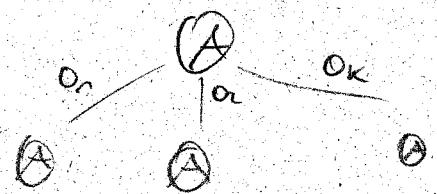
Unendlich: Policy ist stationär: beliebig viel Zeit bleibt

Zeitabhängige Policy in Policy-Tree



2 steps to go

$t-1$ steps to go



2 steps

1 step to go

$$V_p(s) = R(s, a(p)) + \gamma \sum_{s' \in S} P(s'|s, a(p)) \sum_{o} P(o|s', a(p)) \cdot V_{o(p)}(s')$$

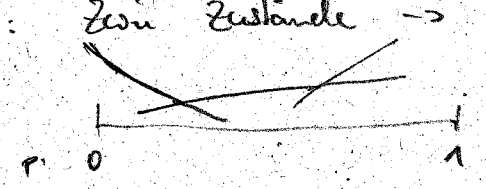
Agent muss diese Values für echte Zustände mit seinem Belief gewichten:

$$V_p(b) = \sum_s b(s) \cdot V_p(s) = \vec{b} \cdot \vec{V}_p$$

Optimale Policy maximiert über dem Raum aller Policies:

$$V_t^*(b) = \max_{p \in ?} \vec{b} \cdot \vec{V}_p$$

Bsp: Zwei Zustände → 1-dim Belief-Space



p- W.keit für State 1
1-p W.keit für State 2

stückweise lineare Funktion, konvex

Unendlicher Horizont: Näherung durch Value-Fkt für end. Horizont

Policy-Raum wächst beliebig an, feinere Unterteilung in Intervalle

Value Iteration

V_1 - Menge der 1-Schritt Policy Trees $\left(V_1^a(s) = R(s,a)\right)$
(einer pro Aktion)

berechne V_t basierend auf V_{t-1}
bis Änderungen klein

Bsp. Tiger mit W.keit 50% hinter einer von zwei Türen

Aktionen: rechte / linke Tür öffnen Tiger → -100 - r
Tiger → +10 = r

listen: r = -1, 85% Korrektheit

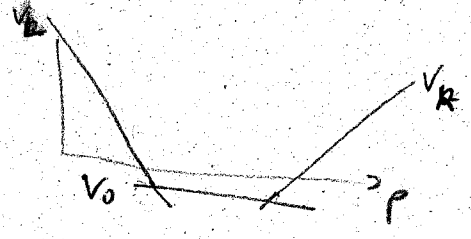
Übung:

States sl, sr, $b(sl) = p$ $b(sr) = 1-p$ p - Belief das Tiger links

$T=1$: $V_R(b) = \sum_s b(s) \cdot V_L(s) = p \cdot 10 + (1-p)\cdot(-100) = -100 + 110p$

$V_L(b) = p\cdot(-100) + (1-p)\cdot 10 = 10 - 110p$

$V_0(b) = p\cdot(-1) + (1-p)\cdot(-1) = -1$



$V_0 > V_R$ / V_L,

V_R $-1 > -100 + 110p$ → $p < 0.9$

V_L $-1 > 10 - 110p$ → $p > 0.1$

$T=2$: $V_{R[RL]} = p\cdot 10 + (1-p)\cdot(-100) + 0.5\cdot 10 + 0.5\cdot(-100) = -145 + 110p$ (next belief = 0.5)

$V_{L[RL]} = V_L(b) + 0.5(10-100) = -35 - 110p$

$V_{[LR]0} = V_{LR}(b) + (-1) = \begin{cases} 9 - 110p & V_{LO} \\ -101 + 110p & V_{RO} \end{cases}$

$V_{0[LR]} = -1 + V_{LR}(b'|o=TL/TR))$

$$b'(s') = \frac{1}{Z} P(o|s',a) \sum_s P(s'|a,s) \cdot b(s)$$

$o = 0.85$
$1-o = 0.85$

$b'(sl, o=TL) \sim o \cdot (1-p) + o \cdot (1-p)) = o \cdot p =$
$b'(sl, o=TR) \sim (1-o) \cdot p$
$b'(sr, o=TL) \sim (1-o) \cdot (1-p)$
$b'(sr, o=TR) \sim o \cdot (1-p)$

Z: $P(o=TL|b) = o \cdot p + (1-o) \cdot (1-p) = 1 + 2op - o - p$
$P(o=TR|b) = (1-o)p + o \cdot (1-p) = p - op + o - op = p - o - 2op$

$$p'_{TL} = b'(sl, o=TL) = \frac{op}{op + (1-o)(1-p)}$$

$$p'_{TR} = b'(sl, o=TR) = \frac{(1-o)p}{(1-o)p + o(1-p)}$$

$V_{0L}(o=TL) = -1 + 10 - 10\, p'_{TL}$
$V_{0L}(o=TR) = -1 + 10 - 10\, p'_{TR}$

7. Intrinsisches Reinforcement-Lernen

- Entwicklg des Kindes ist
  - progressiv : Aktivitäten entsprechen in ihrer Komplexität immer den Fähigkeiten
  - inkrementell : des Kindes. Aufbauend auf Gelerntem, wird neues Fähigkeit erworben
  - autonom : Die Kinder entscheiden, was sie lernen wollen / spielen
  - aktiv : Sie entwickeln eigene Lernaufgaben. Exploration
  - Entwicklgsschritte Piaget [41 The Origins of Intelligence in Children, 1952]

=> Motivation: Vorhandensein von intrinsic Rewards / intrinsic motivation (Interesse)

- Neuheit — novelty
- Überraschung — surprise
- Unstimmigkeit — incongruity
- Nichtvorhersagbarkeit — complexity

Aufgabe darf nicht zu einfach, aber auch nicht zu schwer sein.
geeignete Herausforderung

Existierende Belohnungssysteme
- Beruhen auf Vorhersage von Handlungskonsequenzen
- Wähle Aktionen, die am schlechtesten vorhergesagt werden können
  mit größtem Vorhersagefehler

=> Häufig Fokus auf lösbare (nicht vorhersagbare) Probleme

- Maximiere Lernfortschritt : Änderungsrate des Vorhersagefehlers ∂E

=> Differenz von $\left(\partial E(t+n) - \partial E(t)\right) = -r$

$r > 0$ Fehlerrate sinkt
$r < 0$ -"- steigt
$r = 0$ -"- konstant

=> Problem: Bei mehreren Tasks: instabiles Wechseln zwischen Tasks
mit kleiner und großer Fehlerrate

=> nur Vergleich der Fehlerraten in ähnlichen Situationen / Task
nicht zeitabhängig

Intelligent Adaptive Curiosity (IAC), (Oudeyer, Kaplan, Hafner 2007)

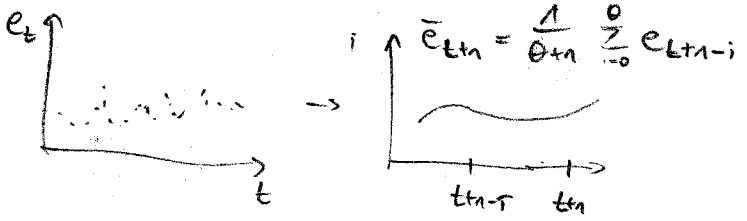
Vereine Mixture of Experts mit RL:

Menge von Experten (wachsend) mit Predictor - für Folgezustand

$s_t, a_t \rightarrow [\ \ ] \rightarrow \hat{s}_{t+1}$

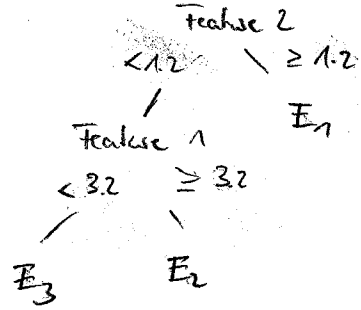
Speichere Vorhersagefehler $e_{t+1} = \|\hat{s}_{t+1} - s_{t+1}\|^2$ pro Experte

Lernfortschritt $FL = -(\bar{e}_{t+1} - \bar{e}_{t+1-\tau})$ Differenz von Prediction Error jetzt (Ableitung) und vor Zeit τ



Generierung von Experten

- Begrenzte Zahl von (Situation, Action)-Samples pro Region
- Teile Raum entlang Koordinatenachsen und Threshold-Value
  => Binärer Baum



Vereinfachung von RL: endlicher Zeithorizont $T=1$: Maximiere Lernfortschritt im nächsten Schritt

- Im Zustand $s(t)$ generiere Liste von Handlungsoptionen $a_1(t) \dots a_n(t)$
- Zuordnung von (s, a_i) zu Experten
- Bewerte erwarteten Lernfortschritt mit (*)
- ε-greedy: Zufallsauswahl mit W.keit ε

8. Gaussian Processes

8.1. Bayesian inference revisited

Bayes-Ansatz basiert auf zwei einfachen Regeln:

- Produktregel: $p(x,y) = p(y|x) \cdot p(x) = p(x|y) \cdot p(y)$
- Summenregel: $p(x) = \int p(x,y)\, dy$

Ziel: Funktionsapproximation

geg.: Datenpaare $(x_1, y_1), \dots (x_N, y_N)$ ges.: $y(x)$ bzw $p(y|x,\mathcal{D})$

$X = [x_1, \dots, x_N]^T \quad Y = [y_1, \dots, y_N]^T$

Ansatz: parametrisierte Funktionenklasse: $f(x,w)$

Stochastisches Modell: $p(y|x,w) = \mathcal{N}(y \mid f(x,w), \beta^{-1})$

einzelne Datenpaare identisch, unabhängig voneinander verteilt

$$\Rightarrow p(Y|X,w) = \prod_{n=1}^{N} p(y_n|x_n,w)$$

$$\ln p(Y|X,w) = -\frac{\beta}{2}\sum_{n=1}^{N}(y(x_n,w)-y_n)^2 + \underbrace{\frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi)}_{C(\beta)}$$

maximum likelihood: $w_{ML} = \underset{w}{\operatorname{argmax}} \ln p(Y|X,w)$

$= \underset{w}{\operatorname{argmin}} \sum (y(x_n,w)-y_n)^2$ - quadr. Fehler

$\beta_{ML}^{-1} = \frac{1}{N}\sum (y(x_n,w_{ML})-y_n)^2$ - mittlere Datenvarianz

ML-Schätzung: $p(y|x,w_{ML},\beta_{ML}) = \mathcal{N}(y \mid f(x,w_{ML}), \beta_{ML}^{-1})$

Problem: Overfitting, Unsicherheit (β_{ML}^{-1}) unabhängig von x

Bayes-Ansatz berücksichtigt a-priori W.keit für Parameter:

$$p(w) = \mathcal{N}(w \mid 0, \alpha^{-1}\mathbb{1}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left(-\frac{\alpha}{2} w^T w\right)$$

Bayes-Regel: $p(w|Y,X) \sim p(Y|X,w) \cdot p(w)$ - a-posteriori W.keit für w

$$w_{MAP} = \underset{w}{\operatorname{argmax}} \ln p(w|Y,X) = \underset{w}{\operatorname{argmin}} \frac{\beta}{2}\sum (y(x_n,w)-y_n)^2 + \frac{\alpha}{2} w^T w$$

MAP-Schätzung: $p(y|x, w_{MAP}, \beta)$ $\Rightarrow$ Overfitting reduziert durch regularisierenden $\|w\|^2$-Term

Festlegung auf ein w erhöht W.keit für Fehler.
Ensemble Averaging / Boosting hat gezeigt, dass Mittelung über versch. Modelle sinnvoll ist und das Bias-Varianz-Dilemma löst

Bayes-Regel für Normalverteilungen

ges. $p(\omega) = \mathcal{N}(\omega \mid \mu, \Sigma)$

und $p(y \mid \omega) = \mathcal{N}(y \mid A\omega + b, S)$ Mittelwert = affine Funktion von ω

$\Rightarrow p(\omega \mid y) = \mathcal{N}(\omega \mid \mu_{\omega|y}, \Sigma_{\omega|y})$

$$\mu_{\omega|y} = \Sigma_{\omega|y}\left(A^T S^{-1}(y-b) + \Sigma^{-1}\mu\right) \quad (*)$$

$$\Sigma_{\omega|y} = \left(\Sigma^{-1} + A^T S^{-1} A\right)^{-1}$$

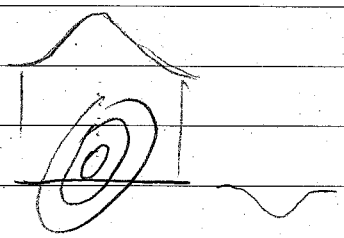
$\Rightarrow p(y) = \mathcal{N}(y \mid \underbrace{A\mu + b}, \underbrace{S + A\Sigma A^T}) \quad (**)$

Varianz summiert sich

voller Bayes-Ansatz: Bestimme Verteilung für neues Datenbsp.:

$$p(y|x, X, Y) = \int \underbrace{p(y|x,\omega)}_{\text{Verteilung für } y \text{ bei best. } \omega} \cdot \underbrace{p(\omega|X,Y)}_{\text{a-posteriori von } \omega} d\omega \qquad p(y,\omega|x,X,Y)$$

$$\hat{=} E\left(p(y|x,\omega)\right)_{\omega|X,Y}$$



Einschub: Eigenschaften der Normalverteilung
→ separate Tafel

Beh. $p(x_a, x_b) = \mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{(\det\Sigma)^{1/2}} \cdot \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

mit $x = (x_a, x_b) \quad \mu = (\mu_a, \mu_b)$

$$\Sigma = \begin{pmatrix}\Sigma_{aa} & \Sigma_{ab}\\ \Sigma_{ab}^T & \Sigma_{bb}\end{pmatrix} = \begin{pmatrix}A & C\\ C^T & B\end{pmatrix}$$

$$p(x_a|x_b) = \mathcal{N}(x_a|\mu_{a|b}, \Sigma_{a|b})$$

mit $\mu_{a|b} = \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(x_b-\mu_b) = \mu_a + CB^{-1}(x_b-\mu_b)$

$$\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ab}^T = A - CB^{-1}C^T$$

$$p(x_a) = \int p(x_a, x_b)\, dx_b = \mathcal{N}(x_a|\mu_a, \Sigma_{aa})$$

*** Bayes for Gaussian

weitere Annahme: lineare Abhängigkeit von den Parametern:

$$\vec{y} = f_M(x,\omega) = W^T\cdot\varphi(\vec{x}) \qquad W\in\mathbb{R}^{M\times K} \quad (i\text{-ter Zeilenvektor von } W^T \hat{=} \text{Parameter für } i\text{-te Komponente})$$

$$y_i = \sum_{j=1}^{\cdot} w_{ji}\,\varphi_j(\vec{x}) \qquad i = 1\dots K \qquad \text{Bsp: Polynome, Gaußglocken, ...}$$

$$\Rightarrow \vec{y}^t = \vec{\varphi}^t\cdot W \Rightarrow Y = \Phi\cdot W \qquad Y\in\mathbb{R}^{N\times K}\ \Phi\in\mathbb{R}^{N\times M}\ W\in\mathbb{R}^{M\times K}$$

$$\Rightarrow W_{ML} = (\Phi^T\cdot\Phi)^{-1}\Phi^T\cdot Y = \Phi^{\#}\cdot Y$$

$$\Phi = \begin{pmatrix}\varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_M(x_1)\\ \vdots & & & \vdots\\ \varphi_1(x_N) & \varphi_2(x_N) & \cdots & \varphi_M(x_N)\end{pmatrix} \qquad Y = (y_1, y_2, \dots y_N)^T$$

$$Y = W^T\cdot\Phi \qquad A = \Phi$$

Bayes a-priori: $p(\omega) = \mathcal{N}(\omega|\mu_0,\Sigma_0)$

a-posteriori: $p(\omega|Y,X) = \mathcal{N}(\mu_N,\Sigma_N)$

$$\mu_N = \Sigma_N(\Sigma_0^{-1}\mu_0 + \beta\cdot\Phi^T\cdot Y) \qquad (*)$$

$$\Sigma_N = (\Sigma_0^{-1} + \beta\Phi^T\cdot\Phi)^{-1}$$

oder für $\varphi(\omega) = \mathcal{N}(\omega | 0, \alpha^{-1} \mathbb{1})$ Mittelwert Null

$$\Sigma_N = \left(\alpha \mathbb{1} + \beta \Phi^T \Phi\right)^{-1}$$

$$\mu_N = \beta \Sigma_N \Phi^T Y = \underbrace{\left(\frac{\alpha}{\beta}\mathbb{1} + \Phi^T\Phi\right)^{-1}\Phi^T}_{\text{regularisierte Pseudoinverse}} Y \quad \in \mathbb{R}^{M \times K}$$ (W-Matrix)

Neu! : $\mathcal{N}(\varphi^T(x) \cdot W, \beta^{-1}\mathbb{1})$ $\mathcal{N}(\mu_N, \Sigma_N)$

$$p(y|x,(X,Y)) = \int p(y|x,\omega,X,Y) \cdot p(\omega|X,Y)\, d\omega$$

Korrektur $\Phi \to \varphi$ $= \mathcal{N}(y | \mu_N^T \cdot \varphi(x), \sigma^2(x)\mathbb{1})$ (** mit $A = \varphi^t$)

$$\sigma^2(x) = \frac{1}{\beta} + \varphi(x)^t \Sigma_N \varphi(x)$$

§2. Gaussian Processes Perspective - W.keitsverteilg über Funktionen

$f_i(x_i) = f(x_i, \omega) = \varphi^T(x_i) \cdot \vec{w}$ $\varphi(\omega) = \mathcal{N}(\omega | 0, \alpha^{-1}\mathbb{1})$

W.keitsverteilg über ω induziert W.verteilg über Funktionen $f(x,\omega)$

Die Trainingsdaten sind gegeben durch

$F = (f_1, \dots, f_N)^T = \Phi \vec{w}$ - Funktionswerte ohne Rauschen

Das ist wiederum eine Zufallsvariable, die - als Summe von Normalverteilgen (über ω) - wiederum eine Normalverteilg ist. Für Mittelwert und Kovarianz gilt:

$$\mu = E(F) = \Phi \cdot E(w) = 0$$

$$\Sigma = \mathrm{cov}(F) = E(F \cdot F^T) = \Phi E(w w^T) \Phi^T = \frac{1}{\alpha}\Phi\Phi^T =: K$$

$$K_{nm} = k(x_n, x_m) = \frac{1}{\alpha}\varphi(x_n)^T \cdot \varphi(x_m)$$ — Kernelfunktion

Der Übergang zur Kernelfunktion $k(x, x')$ macht den Ansatz unabhängig von der Anzahl M von Basisfunktionen. Prinzipiell können in K also auch unendlich viele Basisfkt. eingehen.

⇒ Def
⇒ Illustration

Für Regression berücksichtigen wir wieder Rauschen in den Daten

$$p(Y|F) = \mathcal{N}(Y | F, \beta^{-1}\mathbb{1}) \qquad y_n = f_n + \eta_n = \vec{\varphi}(x_n) \cdot \vec{w} + \eta_n$$

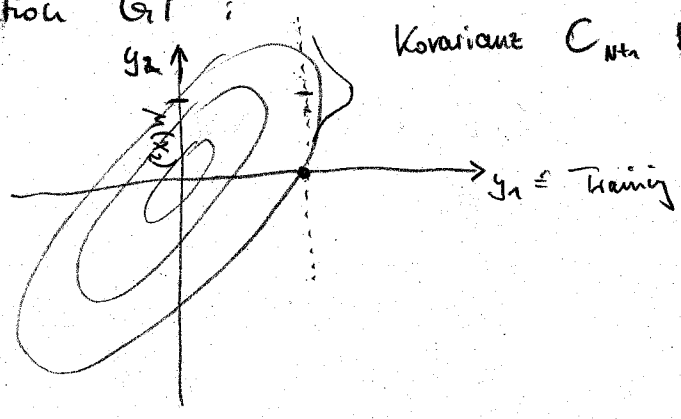
$$p(F) = \mathcal{N}(F | 0, K) \quad \text{- GP mit } \mu = 0 \triangleq E(w) = 0$$

$$K_{nm} = k(x_n, x_m) \quad n,m = 1 \dots N$$

$$p(Y) = \int p(Y|F) \cdot p(F)\, dF = \mathcal{N}(Y | 0, C) \qquad (*)$$

$$C = K + \beta^{-1}\mathbb{1} \qquad C_{nm} = k(x_n, x_m) + \underbrace{\frac{\delta_{nm}}{\beta}}_{\sim \frac{1}{\alpha}\varphi(x_n)\cdot\varphi(x_m)} + \frac{\delta_{nm}}{\beta}$$

Illustration GP:



Kovarianz C_{N+n} ges. durch Kernel $k(x, x')$

Def. Gaussian Process: ist Generalisierung einer multivariaten Gaußverteilung auf unendlich viele Variablen.

GP ist eine Menge von Zufallsvariablen, so daß jede endliche Teilmenge normalverteilt ist.

Informell: GP $\triangleq$ Verteilung über Funktionen

Normalverteilung:	$f = (f_1, \dots f_N)^T \sim \mathcal{N}(\mu, \Sigma)$	index $i = 1 \dots N$
GP	$f(x) \sim GP(m(x), k(x, x'))$	index: x

Für Vorhersage eines neuen Datenpunktes y_{N+1} betrachten wir die bedingte Verteilung $p(y_{N+1} | Y_N)$ wobei $p(y_{N+1}, Y_N)$ wie (*) verteilt ist mit Übergang $N \to N+1$. (Wir können bel. viele Datenpkte verwenden)

$$p(y_{N+1}, Y_N) = \mathcal{N}(Y_{N+1} | 0, C_{N+1}) \qquad C_{N+1} = \begin{pmatrix} C_N & k \\ k^T & c \end{pmatrix}$$

C_N - geg. durch Trainingsdaten, $K_i(x) = k(x, x_i)$ $i = 1 \dots N$

$c = K(x, x) + \beta^{-1}$

$$p(y_{N+1} | Y_N) = \mathcal{N}(y_{N+1} | \mu(x), \sigma^2(x))$$

$$\mu(x) = k^T C_N^{-1} Y_N \qquad \sigma^2(x) = c - k^T C_N^{-1} k$$

D.h. geg. die Kovarianz C_N der Trainingsdaten und die Korrelation von x zu den Trainingsdaten ($k(x)$) kann die Verteilung $p(y)$ bestimmt werden.

$$\mu(x) = \sum_{n=1}^{N} a_n \, k(x_n, x) \qquad a_n = (C_N^{-1} Y_N)_n$$

Falls $K(\cdot, \cdot)$ Funktion von $\|x - x'\|$, dann liefert GP eine Entwicklung in Radialen Basisfunktionen.

8.2.1. Kernelfunktionen

$$C_{nm} = \underbrace{\theta_0 \exp\left(-\frac{1}{2\theta_1}(x_n - x_m)^2\right)}_{\text{Abstandsbasierter Kernel}} + \underbrace{\theta_2 \delta_{nm}}_{\substack{\text{Sanß'sches} \\ \text{Rauschen} \\ \text{mit Varianz } \theta_2}} + \underbrace{\theta_3 \vec{x}_n \cdot \vec{x}_m}_{\substack{\text{lineares Funktionenmodell} \\ y \propto w \cdot \vec{x}}}$$

Bedingungen an Kernelfunktionen:

- Symmetrie: $k(x, x') = k(x', x) \qquad C = C^T$
- positiv definit: $C \geq 0 \hookrightarrow \forall x \neq 0 : x^t C x > 0$

Kombination von Kernelfunktionen:

$c\,k_1(x,x')$, $k_1(x,x') + k_2(x,x')$ — c Konstante

$f(x)\cdot k_1(x,x')\,f(x')$ — f bel. Funktion

$q(k_1(x,x'))$, $\exp(k_1(x,x'))$ — q Polynom mit positiven Koeffizienten

$k_1(x,x') \cdot k_2(x,x')$

8.2.2. Schätzung der Hyperparameter

Maximiere log-Likelihood $\ln p(Y|\theta)$:

$$\ln p(Y|\theta) = -\frac{1}{2}\ln|C_N| - \frac{1}{2} Y_N C_N^{-1} Y_N - \frac{N}{2}\ln(2\pi)$$

$\Rightarrow$ nichtlineare Optimierung: z.B. Gradientenabstieg

UKR

9. Sampling-Verfahren

Bayes-Ansatz liefert: $p(y|x) = \int \underbrace{p(y|x,w)}_{\text{Verl. } y \text{ für bek. } w} \cdot \underbrace{p(w|X,Y)}_{\text{a-posteriori von } w} dw$

Problem: Verteilungen und Integral häufig nicht analytisch berechenbar.
(Bislang: Annahme von Gaußverteilungen, keine multi-modalen Verl.)
⇒ Approximationsverfahren für $E[f(z)]_z = \int f(z)\, p(z)\, dz$

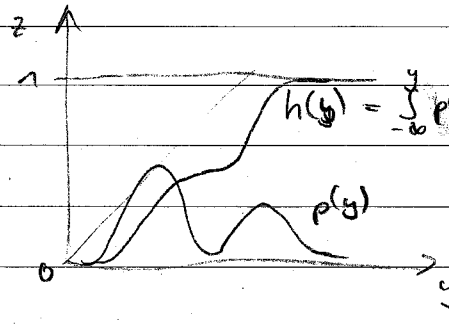
"Monte Carlo" / "empirisches Mittel"
$\hat{f} = \frac{1}{L} \sum_{l=1}^{L} f(z_l)$ z_l – zufällig entsprechend $p(z)$ gezogen

Vorteil: Genauigkeit der Schätzung hängt nicht von Dimensionalität von z ab ??

9.1 Sampling eines bel. Verteilg $p(y)$

Bsp.: Pseudo-Zufallszahlen-Generator mit gleichverteilg in $[0,1]$
⇒ z gleichverteilt in $[0,1]$

Wie muss z transformiert werden, um eine bel. Verteilg $p(y)$ zu erhalten?



$h(y) = \int_{-\infty}^{y} p(y')\, dy' \doteq \int_{0}^{z} p(z')\, dz' = \underbrace{p(z)}_{1} \cdot z = z$

$\frac{dh}{dy} = p(y) = p(z) \cdot \left|\frac{dz}{dy}\right|$

Transformation: $y = h^{-1}(z)$

Bsp.: Exponentialverteilg: $p(y) = \lambda e^{-\lambda y}$ $y \in [0, \infty)$

$h(y) = -e^{-\lambda y'}\Big|_0^y = -e^{-\lambda y} + 1$

$\Rightarrow y = -\frac{1}{\lambda} \ln(1-z)$

Bsp.: Multivariate Verteilg: $p(y_1, \dots, y_M) = p(z_1, \dots, z_M) \left|\underbrace{\frac{\partial(z_1, \dots, z_M)}{\partial(y_1, \dots, y_M)}}_{\text{Jacobi-Matrix der Transformation}}\right|$

Box-Mullers-Verfahren für 2-d Fall:
- (z_1, z_2) uniform in $[-1,1]$ $(z \to 2z-1)$.
- Lösche alle Samples, falls $z_1^2 + z_2^2 > 1$

⇒ gleichverteilg im Einheitskreis mit $p(z_1, z_2) = \frac{1}{\pi}$

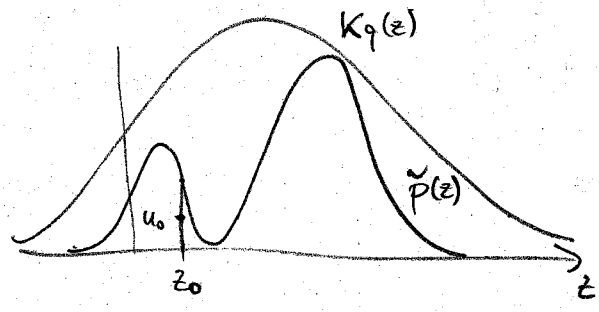
$$y_1 = z_1 \sqrt{\frac{-2\ln r^2}{r^2}} \qquad y_2 = z_2 \sqrt{\frac{-2\ln r^2}{r^2}} \qquad r^2 = z_1^2 + z_2^2$$

$$p(y_1, y_2) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y_1^2}{2}\right) \cdot \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y_2^2}{2}\right) \quad - \mathcal{N}(0,1)$$

bel. Verteilg $\mathcal{N}(\mu, \Sigma)$ $\quad x = \mu + Ly \quad \Sigma = LL^T$ Cholesky-Zerlegg für symm. Matrix

9.2. Rejection Sampling

analytische Bestimmung von $h^{-1}(z)$ oft nicht möglich:



$$p(z) = \frac{1}{Z}\tilde{p}(z)$$

$z_0 \sim q(z)$

$u_0 \sim$ gleichverteilt in $[0, kq(z)]$

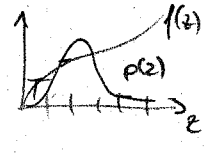
→ (z_0, u_0) gleichverteilt unter Kurve $kq(z)$
→ Ignoriere Sample mit $u_0 > \tilde{p}(z)$
→ Gleichverteilung von (z_0', u_0') unter $\tilde{p}(z)$
⇒ z_0 verteilt entsprechend $p(z)$
→ Akzeptanzrate: $p(\text{accept}) = \int \frac{\tilde{p}(z)}{kq(z)} q(z)\, dz = \frac{\text{Fläche unter } \tilde{p}}{\text{Fläche unter } kq}$

Nachteil: ungeeignet in hochdimensionalen Räumen
Die Akzeptanzrate sinkt exponentiell mit Dimension: p^M

9.3. Importance Sampling

$$E[f] \simeq \sum_{l=1}^{L} \frac{\tilde{p}(z_l)}{z_p} f(z_l)$$

z-Raum in L-Bins unterteilt



$$E[f] = \int f(z)\, p(z)\, dz$$
$$= \int f(z) \frac{\tilde{p}(z)}{z_p} \cdot \left(\frac{\tilde{q}(z)}{z_q}\right)^{-1} q(z)\, dz$$

↓ L samples aus $q(z)$

$$\simeq \frac{1}{L} \frac{z_q}{z_p} \sum_{l=1}^{L} \frac{\tilde{p}(z_l)}{\tilde{q}(z_l)} f(z_l) = \sum w_l f(z_l)$$

$$w_l = \frac{\tilde{p}(z_l)}{\tilde{q}(z_l)} \cdot \frac{1}{\sum_l \frac{\tilde{p}}{\tilde{q}}} \quad - \text{Wichtigkeit}$$

Wichtig $q(z)$ soll Verteilung $p(z)$ grob wiedergeben ✓

3.4 Markov-Chain Monte-Carlo

Markov-Kette

Idee: Einzige Sequenz $z_1 \dots z_t$ mit random walk

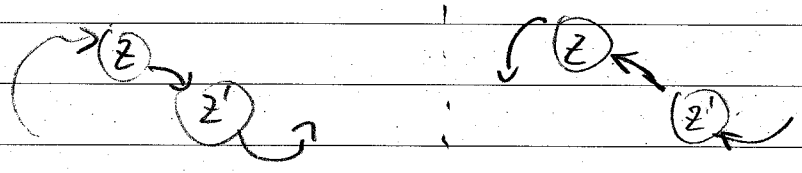
Markov-Eigenschaft: $p(z_{t+1} | z_t, \dots z_1) = p(z_{t+1} | z_t)$

Formal $z_{t+1} \sim T(z_{t+1} \leftarrow z_t)$ Transition-W.keit ($p(z_{t+1} | z_t)$

Ziel: stationäre Verteilung $p^*(z)$: $\int T(z' \leftarrow z)\, p^*(z)\, dz \overset{!}{=} p^*(z')$

Detailierte Gleichheit impliziert Stationarität:

$$T(z' \leftarrow z)\, p^*(z) \overset{!}{=} T(z \leftarrow z')\, p^*(z') \Rightarrow \int T(z' \leftarrow z) p^*(z) = p^*(z')$$



Metropolis-Hasting-Algorithmus

Transition:

- Vorgeschlagene Transition: $q(z' | z_t) = \mathcal{N}(z' | z_t, \sigma^2)$
- akzeptiere mit W.keit $a = \min\left(1, \frac{p(z') \cdot q(z_t | z')}{p(z_t) \cdot q(z' | z_t)}\right)$
- $z_{t+1} = \begin{cases} z' & \text{falls akzeptiert} \\ z_t & \text{sonst} \end{cases}$

Bemerkung:

- akzeptiere samples mit höherer W.keit auf jeden Fall
- Normalisierung von $\tilde{p}$ irrelevant $p(z) = \frac{\tilde{p}(z)}{Z_p}$ kürzt sich im Quotient raus
- Detailierte Gleichheit gegeben: $T(z' \leftarrow z) = q(z' | z) \cdot a(z', z)$
- Schrittweite σ^2 für Sampling-Schritt entsprechend Ausdehnung der Zielverteilung $p(z)$ wählen