Grundlagen der Künstlichen Intelligenz

Prof. Dr. Sibylle Schwarz
HTWK Leipzig, Fakultät IM
Gustav-Freytag-Str. 42a, 04277 Leipzig
Zimmer Z 411 (Zuse-Bau)
https://informatik.htwk-leipzig.de/schwarz
sibylle.schwarz@htwk-leipzig.de

Wintersemester 2022/23

Was ist Künstliche Intelligenz?

EU-Factsheet on Artificial Intelligence
(`https://digital-strategy.ec.europa.eu/en/library/factsheet-artificial-intelligence-europe`)

> *Artificial intelligence (AI) refers to systems that show intelligent behaviour: by analysing their environment they can perform various tasks with some degree of autonomy to achieve specific goals.*
> *Mobile phones, e-commerce tools, navigation systems and many other different sensors constantly gather data or images. AI, particularly machine-learning technologies, can learn from this torrent of data to make predictions and create useful insights.*

Aussage über das derzeitige (eingeschränkte) Verständnis von KI

Können Maschinen denken?

Alan Turing 1950

Konkretisierung der Frage:
Können Maschinen denken?

zur überprüfbaren Frage:
Können Maschinen konstruiert werden, die einen speziellen Test bestehen?

Imitation Game

Imitation Game (Alan Turing 1950):

- zwei verschlossene Räume,
  in einem befindet sich Herr A, im anderen Frau B
- eine Person C (Frager) stellt Fragen, A und B antworten
- Kommunikation über neutrales Medium,
  an welchem das Geschlecht nicht erkennbar ist,
- C soll herausfinden, in welchem der Räume Frau B ist
- Herr A versucht, C irrezuführen
- Frau B kooperiert mit C

Herr A besteht den Test, wenn ihn C für Frau B hält.

Wie erkennt man Intelligenz: Turing-Test

Turing-Test 1950: verschiedene Versionen des Imitation Game

- A ist Machine statt Mann (B Person beliebigen Geschlechts)
- verschiedene Kooperationsverhalten von A und B

Vorschlag zur Bewertung natürlichsprachlicher Kommunikationsfähigkeiten

Beginn koordinierter Forschung zur Künstlichen Intelligenz

John McCarthy	Marvin Minsky	Claude Shannon
Programmiersprachen	Kognitionswissenschaft	Informationstheorie

stellten 1955 die Vermutung auf, dass

„jeder Aspekt des Lernens oder jedes anderen Ausdrucks von Intelligenz prinzipiell so präzise beschrieben werden kann, dass sich eine Maschine konstruieren lässt, die ihn simuliert. “

Begriff Künstliche Intelligenz

McCarthy formulierte das Ziel,

„herauszufinden, wie man Machinen konstruiert, die

- natürliche Sprache benutzen,
- Abstraktionen und Begriffe entwickeln,
- Aufgaben lösen, die (bis dahin) nur Menschen lösen konnten,
- sich selbst verbessern.“

und prägte dafür den Begriff Künstliche Intelligenz.

Beginn koordinierter Forschung zur Künstlichen Intelligenz

1956: erste Konferenz zur Künstlichen Intelligenz
Dartmouth Summer Research Project on Artificial Intelligence

Themen:

- Berechnungsmodelle in Computern
- Kommunikation mit Computern in natürlicher Sprache
- Berechenbarkeitstheorie
- Neuronale Netzwerke
- Selbst-Verbesserung
- Abstraktionen
- Zufälligkeit und Kreativität

Forschung zur Künstlichen Intelligenz

Momentaufnahme 2006:
Dartmouth Artificial Intelligence Conference: The Next Fifty Years

Themen:

- Modelle des (menschlichen) Denkens
- Neuronale Netzwerke
- (Maschinelles) Lernen und Suchen
- Maschinelles Sehen
- Logisches Schließen
- Sprache und Kognition
- KI und Spiele
- Interaktion mit intelligenten Maschinen
- Ethische Fragen und zukünftige Möglichkeiten der KI

Ansätze intelligenter Systeme

- Simulation menschlichen Verhaltens
  (Verständnis und eigenes Denken nicht notwendig)
  Modellierung von Kognition,
  statistische Verfahren, Training mit vielen Fällen
  Getroffene Entscheidungen werden nicht begründet.
  schwache künstliche Intelligenz

- Verstehen und Nachbilden des menschlichen Denkens
  (Verständnis und eigenes Denken notwendig)
  Denkmodelle, mentale Modelle als Grundlage
  logisches Schließen, Abstraktion
  Jede Entscheidungen kann nachvollziehbar begründet werden.
  starke künstliche Intelligenz

Kritik am Turing-Test

Kritik:

schwache KI genügt, um den Turing-Test zu bestehen

1966: Maschinelle Psychotherapeutin Eliza besteht Turing-Test

Searle (1980) Chinese-Room-Argument:
eine (nicht chinesisch verstehende) Person B in einem verschlossenen Raum mit einem (riesigen) Regelbuch mit chinesischen Fragen und passenden Antworten.

- A stellt Fragen, B antwortet.
- B antwortet mit Hilfe des Buches immer passend, ohne die Frage verstanden zu haben.

These: (anscheinend) intelligentes Verhalten ist noch keine Intelligenz, wenn Verständnis fehlt (Ansatz der starken KI)

außerdem: praktisch nicht umsetzbar (Datenmenge)

Symbolische KI-Methoden

Wissensrepräsentation: formale Beschreibung von Umwelt (Randbedingungen) und Problem

Problemlöseverfahren: zur Lösung vieler Probleme anwendbares Standardverfahren (z.B. logisches Schließen)

Beispiele:

- Entscheidungsbäume und -tabellen
- Regelsysteme, Logiken, logisches Schließen
- Constraint-Systeme und -Löser
- deklarative Programmierung (logisch, funktional)
- fallbasiertes Schließen (durch Analogien)
- Simulation

typische Anwendungen klassischer KI-Methoden:

- Entscheidungsunterstützung (z.B. Finanzwirtschaft)
- Diagnosesysteme (z.B. in Medizin, Technik)
- Bewegungs- und Ablaufplanung

Statistische KI-Methoden

„Soft-Computing“ oft besser geeignet für Probleme

- die unvollständig beschrieben sind,
- die keine eindeutige Lösung haben,
- für die keine effizienten Lösungsverfahren bekannt sind, usw.

einige Ansätze:

- künstliche neuronale Netze
- evolutionäre Algorithmen
- Schwarm-Intelligenz, Ameisen-Algorithmen
- Fuzzy-Logiken, probabilistische Logiken

Aktuelle Entwicklung

starker Fortschritt einiger KI-Methoden („Deep Learning“)
in den letzten 15 Jahren aufgrund der Entwicklung bei

- Computertechnik: Parallelrechner, GPU (70% Einfluss)
- Speichermöglichkeit großer Datenmengen, Verfügbarkeit großer strukturierter und annotierter Datenmengen (20%)
- neue Typen künstlicher neuronaler Netze, bessere Algorithmen (10%)

sowie starkem Medieninteresse an bestimmten Erfolgen, z.B.

- 1997 Deep Blue gewinnt gegen amtierenden Weltmeister
- 2011 Watson schlägt zwei Meister in Quizshow Jeopardy!
- 2012 erste Zulassung eines autonomen Fahrzeugs für den Test auf öffentlichen Straßen
- 2016 AlphaGo schlägt Go-Meister
- Unterhaltsames in Bilderkennung, -interpretation, -erzeugung
- ...

führt zum wiederholten Aufflammen der KI-Euphorie

Leistung aktueller (statistischer) KI-Systeme

nahe und teilweise über den menschlichen Fähigkeiten z.B. bei

- Erkennung von Objekten in Bildern
- Einordnung / Klassifikation von Objekten und Situationen
- Reaktion auf klar erkannte Situationen
- strategischen Spielen mit endlichem Zustandsraum
  z.B. Schach, Go

prinzipielle Herausforderungen:

- Zuverlässigkeit, Sicherheit
- Begründung, Erklärung
- Kontext menschlichen Denkens (z.B. Ethik, Recht)
- individuelle / kulturelle Bezugssysteme
- mentale Modelle

Schwächen aktueller (statistischer) KI-Systeme

KI derzeit noch weit von menschlichen Fähigkeiten entfernt bzgl.

- Erkennung der eigenen Grenzen
- Lernen ohne vorheriges Training mit großen Mengen (manuell) annotierter Daten
- Übertragen von Wissen zwischen verschiedenen Anwendungsbereichen
- Intuition
- Aufstellen und Überprüfen sinnvoller Annahmen bei unvollständig vorhandener Information
- Kombination verschiedener Methoden
- Schließen im Kontext menschlicher Bezugssysteme

Einordnung in die Informatik

Informatik Wissenschaft von der Darstellung und Verarbeitung symbolischer Information durch Algorithmen

Einordnung in die Teilgebiete der Informatik:

theoretisch

- Sprachen zur Formulierung von Information und Algorithmen,
- Berechenbarkeit durch Algorithmen,
- Grundlagen für technische und praktische (und angewandte) Informatik

Grundlagen, z.B. Logik, formale Sprachen

technisch

- maschinelle Darstellung von Information
- Mittel zur Ausführung von Algorithmen

Anwendung, z.B. technische Diagnose

praktisch Entwurf und Implementierung von Algorithmen
Grundlagen, z.B. Graph-Suchverfahren, Inferenzalgorithmen, Algorithmen zum Constaint-Lösen

angewandt Anwendung von Algorithmen, z.B.
Anwendung, z.B. KI, Spracherkennung, Bilderkennung, Suchmaschinen, autonome Agenten

Inhalt der Lehrveranstaltung

Motivation und Grundlagen:

- Daten, Information, Wissen
- Wissenbasierte Systeme (Aufgaben, Aufbau)

Wissensrepräsentations- und -verarbeitungsprinzipien:

- symbolische Verfahren:
  - Suche in Zustandsübergangssystemen (vollständig, heuristisch)
  - Klassifikation
  - Regelsysteme
  - Logiken (klassische und nichtklassische)
  - Logische Programme
  - Unvollständiges Wissen (nichtmonotones Schließen)
  - Entscheidungstabellen, -bäume und -diagramme
- statistische Verfahren:
  - mehrwertiges / fuzzy / unscharfes Schließen
  - maschinelles Lernen
  - künstliche neuronale Netze

Literatur

Folien, Aufgaben, ... zur aktuellen Vorlesung unter
`https://informatik.htwk-leipzig.de/schwarz/lehre/ws22/kib`

Bücher:

- Ingo Boersch, Jochen Heinsohn, Rolf Socher:
  Wissensverarbeitung (Spektrum, 2007)
- Wolfgang Ertel:
  Grundkurs Künstliche Intelligenz (Springer, 2016)
  (elektronische Version in HTWK-Bibliothek)
- Ronald Brachman, Hector Levesque:
  Knowledge Representation and Reasoning
  (Morgan Kaufmann 2004)
- Stuart Russell, Peter Norvig:
  Künstliche Intelligenz (Pearson 2004)
- George Luger: Künstliche Intelligenz (Pearson 2001)

Organisation

5 ECTS (Präsenzzeit 56 h, Vor- und Nachbereitungszeit 94 h)

- wöchentlich
  - eine Vorlesung
  - Übungsaufgaben (ÜS + gelegentlich Autotool)
  - Übung je Gruppe
- Prüfungsvorleistung: Beleg
  - Präsentation der Lösung der Übungsaufgaben (MLV)
  - Autotool
- Prüfung: Klausur 90 min

Daten, Wissen, Intelligenz

Umwelt		Reize, Eindrücke
Agent	Wahrnehmen, Beobachten	Daten
	Erkennen, Verstehen	Information
	Anwenden, Können	Wissen
	Lernen	Wissenserwerb (Intelligenz?)
	Reflektieren, Begründen, Erkennen der Grenzen, Verstehen	Intelligenz

Beispiel: Daten, Information, Wissen, Intelligenz

Daten	Darstellungsform (Syntax) Zeichenketten, Bilder, Ton, . . . (z.B 39.7)
Information	Bedeutung der Daten (Semantik) in einem bestimmten Kontext im Beispiel: Körpertemperatur= 39.7
Wissen	Information mit einem Nutzen, trägt zur Lösung eines Problemes bei, Nutzen abhängig von vorhandenem Kontextwissen im Beispiel: Kontext Körpertemperatur $>$ 39.0 ist Fieber, Fieber ist Symptom von Erkältungen oder Nebenwirkung einer Impfung oder ... mögliche Behandlung, z.B. Wadenwickel, Medikamente
Wissenserwerb	selbständige Informationsgewinnung (auch zum Kontext) im Beispiel über (derzeit typische) Auslöser, Nebensymptome, neue Behandlungsmethoden
Intelligenz	Diagnose und Auswahl aus Therapie-Alternativen speziell für die zu behandelnde Person durch Abwägung der zu erwartenden Wirkungen, ggf. Überweisung zu Spezialisten

Explizites und implizites Wissen

explizites Wissen

z.B. Fakten, Aussagen, Zusammenhänge, Verfahren
symbolische Repräsentation ermöglicht
Anwendung symbolischer / logischer Verfahren

implizites Wissen

z.B. Fähigkeiten wie Laufen, Autofahren,
Schachspielen
wird durch Training erworben,
(ggf. mit Hilfe expliziten Wissens, z.B. Spielregeln)
Nachbildung durch statistische Verfahren

Kommuniziert werden kann nur explizites Wissen.

Zur Kommunikation (in Raum und Zeit) ist
Transformation von implizitem in explizites Wissen notwendig.

Arten von Wissen

deklarativ über Zustände (der Welt) Fakten, Aussagen, Zusammenhänge, z.B.

- Fliegenpilze sind ungenießbar.
- Es existieren gerade Primzahlen.
- Eine Liste $(x_1, \ldots, x_n)$ ist genau dann aufsteigend sortiert, wenn sie leer ist oder ($x_1 \leq x_2$ und $(x_2, \ldots, x_n)$ aufsteigend sortiert ist).

prozedural über Zustandsübergänge Regeln, Algorithmen, Funktionen, z.B.

- Euklidischer Algorithmus
- aussagenlogisches Resolutionsverfahren
- Sortierverfahren
- Kochrezept

Ist die folgende Aussage Fakten- oder prozedurales Wissen?
Jedes Kind eines Kindes einer Person X ist ein Enkel von X.

Also: Repräsentationen von Regeln, Algorithmen und Funktionen lassen sich auch als Faktenwissen auffassen.

Was bisher geschah

- Definition KI (Versuche früher und jetzt)
- KI-Geschichte
- KI-Tests (Turing, Chinese Room)
- Daten, Information, Wissen, Intelligenz
- symbolische / statistische KI
- explizites und implizites Wissen

Wissen

- Was ist Wissen?
- Wie lässt es sich darstellen?
- Wie lässt es sich nutzen, um Probleme zu lösen ?
- Wie lässt es sich erweitern / ändern ?

Analogie zu Wissen von Experten auf einem Fachgebiet

Anforderungen an die Repräsentation von Wissen

Qualitätskriterien bei der Modellierung von (Kontext-)Wissen:

- für Problembereich geeignete Abstraktion
- effektiv, redundanzfrei
- vollständig
- (einfach) erweiterbar und modifizierbar
- verständlich

- verständliche Methode zum Lösen von Aufgaben (Beantworten von Fragen) in diesem Bereich / Kontext
- Begründungen / Erklärungen für gefundene Lösung

Verarbeitung von Wissen – Aufgaben

Beispiele:

- Problemlösen
  - algorithmische Suche in Zustandsräumen
  - logisches Schließen

  Beispiel: n-Damen-Problem, kürzeste Wege in Graphen
- Planen

  Finden einer Folge von Aktionen zum Erreichen eines Zieles

  Beispiel: morgens Anziehen, Fertigungsroboter
- Klassifikation

  Finden von Klassen (Diagnosen) anhand der Merkmalswerte (Symptome)

  Beispiel: Tiere, Fahrzeuge, Krankheiten, Fehlfunktionen

Lösungsverfahren teilweise schon bekannt aus den Modulen

- Modellierung
- Algorithmen und Datenstrukturen

Verarbeitung von Wissen – Methoden (Beispiele)

Suche / Planen:

- Kontext: Zustandsübergangssystem
- Aufgabe: Startzustand und Anforderungen an Zielzustände
- Lösung: Zielzustand (und evtl. Pfad dorthin)

Lösungsverfahren: Suche (vollständig oder heuristisch)

Inferenz, logisches Schließen:

- Kontext: Menge logischer Formeln
- Aufgabe: Gilt die Behauptung (logische Formel) im Kontext?
- Lösung: ja / nein (evtl. mit Begründung)

Lösungsverfahren: logisches Folgern oder Schließen

Statistische Klassifikation:

- Kontext: klassifizierte Datenmenge (bekannte Fälle)
- Aufgabe: neuer Fall
- Lösung: Klassifikation (Zuordung zu einer Klasse)

Lösungsverfahren: statistische Verfahren (z.B. trainiertes KNN)

Problemlösung durch Suche in Graphen – Beispiele

- Finden von Wegen in einem Graphen
  - Aufgabe:
    - gegeben: Graph G (Tafel)
    - gesucht: Weg (Pfad) in G von Knoten u zu Knoten v
  - Lösungsidee: Suche im Graphen
- Münzenstapelspiel (für eine Person)
  - Aufgabe:
    - gegeben: Stapel von n Münzen
    - gesucht: Zugfolge durch erlaubte Züge (zwei Münzen von einem Stapel nehmen und auf beide Nachbarn verteilen) bis zu einer Situation, in der kein Zug möglich ist
  - Lösungsidee:
    - Modellierung als Zustandsübergangssystem (Graph mit Markierungen)
    - Suche im Graphen
- 3 Krüge
  - Aufgabe:
    - gegeben: 3 volle Krüge mit Volumen 9l, 7l, 4l,
    - gesucht: genau 6l in einem der 3 Krüge
  - Lösungsidee: Zustände als Knoten eines Suchbaumes

Suchprobleme – Wissen (Kontext), Aufgabe und Lösung

Darstellung von

Wissen (Kontext) (Zustandsübergangssystem)
als Graph $G = (V, E)$ mit

- Menge V von Zuständen (evtl. unendlich)
  oft beschrieben durch Eigenschaften
- (gerichtete) Kanten:
  mögliche Übergänge zwischen Zuständen
  Übergangsrelation $E \subseteq V \times V$

Aufgabe definiert durch

- Startzustand $s \in V$
- Menge $Z \subseteq V$ von Zielzuständen
  (oder Eigenschaften der Zielzustände)

Lösung: Folge von Zuständen
(Weg von einem Start- zu einem Zielzustand)
(Mitunter interessiert nur der erreichte Zielzustand.)

Entfaltung des Graphen zu einem Baum:

Pfade im Graphen = Knoten im Baum

Problemlösen durch Suchen

- formale Darstellung des Problemes
  als Graph (z.B. Baum, DAG)
- formale Beschreibung der Lösung als Eigenschaft von
  - Pfaden im Graphen
  - Knoten im Baum

Möglichkeiten zum Problemlösen:

- Pfadsuche im Graphen
- Knotensuche im Baum

Suche in Graphen

(aus dem Modul Algorithmen und Datenstrukturen schon bekannte) Verfahren zur Suche in Graphen (und Bäumen):

Tiefensuche (depth-first search):
Suche zuerst in Teilbäumen eines noch nicht besuchten Nachbarn des aktuellen Knotens

Breitensuche (breadth-first search):
Suche zuerst in Teilbäumen eines noch nicht besuchten Knotens mit der geringsten Tiefe

Allgemeines Suchverfahren

Daten:
- L_a Menge der noch zu expandierenden Knoten (Agenda)
- L_x Menge der expandierten Knoten
- s Startknoten
- φ Anforderungen an Lösung (Eigenschaft der Zielknoten)

Allgemeiner Suchalgorithmus:

1. $L_a = \{s\}, L_x = \emptyset$
2. solange $L_a \neq \emptyset$:
   - 2.1 Wähle auf festgelegte Art einen Knoten u in L_a und verschiebe u aus L_a in L_x.
   - 2.2 Füge alle Nachbarn v von u mit $v \notin L_a \cup L_x$ und $v \notin \text{Mod}(\varphi)$ auf eine festgelegte Art in L_a ein.
     (falls für ein v gilt $v \in \text{Mod}(\varphi)$: Abbruch mit Lösung v)

prominente Spezialfälle:

Tiefensuche
- Wähle Konten, der zuletzt in L_a eingefügt wurde
- Einfügen der Nachbarn an den Anfang der Liste L_a
- Verwaltung von L_a als Stack

Breitensuche
- Wähle Konten, der zuerst in L_a eingefügt wurde
- Einfügen der Nachbarn an das Ende der Liste L_a
- Verwaltung von L_a als Queue

Eigenschaften von Suchverfahren

Fairness: jeder Knoten wird expandiert

Vollständigkeit: jede Lösung wird gefunden

Optimalität: eine beste Lösung wird zuerst gefunden

Zeitaufwand: größtmögliche Anzahl der Schritte von einem Start- zu einem Endzustand

(Speicher-)Platzbedarf: größtmögliche Länge der Anzahl der noch nicht expandierten Knoten in der Agenda L_a

Laufzeit für Baum mit maximaler Tiefe t und maximalem Verzweigungsgrad d:

- Breitensuche: vollständig,
  (optimal, falls beste Lösung = geringe Tiefe)
  großer Zeitaufwand: $O(d^t)$
  großer Platzbedarf: $O(d^t)$
- Tiefensuche:
  unvollständig bei unendlichen Bäumen, nicht optimal
  großer Zeitaufwand: $O(d^t)$
  geringerer Platzbedarf: $O(td)$

Suche mit schrittweiser Vertiefung

beschränkte Tiefensuche:

1. festgelegte Tiefenbeschränkung $m \in \mathbb{N}$
2. Tiefensuche auf allen Pfaden bis zur Tiefe m

nicht vollständig, weiter entfernte Lösungen werden nicht gefunden

Schrittweise Vertiefung(iterative deepening)
Kombination aus Breiten- und Tiefensuche durch Nacheinanderausführung der beschränkten Tiefensuche für alle $m \in \mathbb{N}$, solange keine Lösung gefunden wurde

vollständig, optimal
(asymptotischer) Zeit- und Platzbedarf wie Tiefensuche

Gleiche-Kosten-Suche (kleinste bisherige Kosten)

(uniform-cost-search)

bei Zustandsübergängen mit verschiedenen Kosten

Ziel: Lösung (Pfad vom Start- zu einem Lösungsknoten) mit möglichst geringen Pfadkosten
(Pfadkosten = Summe der Kosten aller Übergänge auf dem Pfad)

Kostenfunktion für Knoten $k : V \to \mathbb{R}_{\geq 0}$

$k(u)$ = minimale (bisher entdeckte) Pfadkosten vom Startknoten zu u

Datenstruktur zur Verwaltung von L_a: Priority Queue
Priorität eines Knotens u: $k(u)$

prominente Spezialfälle:

- Breitensuche (Kosten = Tiefe des aktuellen Knotens u)
- Dijkstra-Algorithmus zur Bestimmung kürzester Wege (bekannt aus Algorithmen und Datenstrukturen)
  Kosten = minimale bisher bekannte Kosten vom Startknoten zum aktuellen Knoten u

Was bisher geschah

- Daten, Information, Wissen, Lernen, Intelligenz
- Repräsentation und Verarbeitung von Wissen

Repräsentation: Beschreibung von

Kontext: Zustandsübergangssystem:
gerichteter Graph $G = (V, E)$ mit

- Knotenmarkierungen $l_v : V \to L_V$ mit
  L_V: Eigenschaften der Zustände
- Kantenmarkierungen $l_E : E \to L_E$ mit
  L_E: mögliche / zulässige Aktionen, Gewicht, Kosten

Aufgabe / Frage:

- Startzustand $s \in V$
- (Eigenschaften der) Zielzustände
  (z.B. Einschränkung von Variablenwerten)

Lösung: zulässiger Weg (Zustandsfolge $p \in V^*$)
vom Start- zu einem Zielzustand

Verarbeitung: Pfadsuche im Graphen

- uninformiert (blind): Tiefen-, Breitensuche, schrittweise Vertiefung
- Gleiche-Kosten-Suche bzgl. Kostenfunktion $k : V \to \mathbb{R}_{\geq 0}$

WH: Allgemeines Suchverfahren

Daten:
- L_a Menge der noch zu expandierenden Knoten (Agenda)
- L_x Menge der expandierten Knoten
- s Startknoten
- φ Anforderungen an Lösung (Eigenschaft der Zielknoten)

Allgemeiner Suchalgorithmus:

1. $L_a = \{s\}, L_x = \emptyset$
2. solange $L_a \neq \emptyset$:
   - 2.1 Wähle auf festgelegte Art einen Knoten u in L_a und verschiebe u aus L_a in L_x.
   - 2.2 Füge alle Nachbarn v von u mit $v \notin L_a \cup L_x$ und $v \notin \text{Mod}(\varphi)$ auf eine festgelegte Art in L_a ein.
     (falls für ein v gilt $v \in \text{Mod}(\varphi)$: Abbruch mit Lösung v)

Verschiedene Suchverfahren unterscheiden sich nur in der Auswahl des expandierten (festgelegten) Knotens aus L_a.
nach Festlegung durch Datenstruktur zur Verwaltung von L_a:

- Stack: Tiefensuche
- Queue: Breitensuche
- Priority Queue: Gleiche-Kosten-Suche (z.B. Dijkstra)

Beispiel Missionare und Kannibalen

informale Problembeschreibung:

- Zu Beginn: 3 Missionare und 3 Kannibalen am linken Flussufer
- Ziel ist das Übersetzen aller Personen an das rechte Ufer.
- Links liegt ein Ruderboot, welches höchstens zwei Personen fasst.
- Sobald an einem Ufer mehr Kannibalen als Missonare sind, werden alle Missionare dort gefressen.

(eine mögliche) formale Darstellung der Aufgabe:

- Zustände: $((m_l, k_l), (m_r, k_r), b) \in \left(\{0, \ldots, 3\}^2\right)^2 \times \{0, 1\}$ mit Anz. Missionare/Kannibalen links/rechts $m_l, m_r, k_l, k_r \in \{0, \ldots, 3\}$, wobei $m_l + m_r = 3$ (Missionare) und $k_l + k_r = 3$ (Kannibalen) Boot liegt links $b \in \{0, 1\}$
- Startzustand: $((3, 3), (0, 0), 1)$
- Zielzustand: $((0, 0), (3, 3), x)$, wobei x egal
- Zustandsübergänge $((m, k), d) \in \{(0, 1), (0, 2), (1, 1), (1, 0), (2, 0)\} \times \{\rightarrow, \leftarrow\}$ $((m_l, k_l), (m_r, k_r), b) \overset{((m,k),d)}{\rightarrow} ((m'_l, k'_l), (m'_r, k'_r), 1 - b)$, wobei ...

Bidirektionale Suche (nach Lösungswegen, Plänen)

- simultane Suche ab Startknoten und ab Zielknoten
  Vorwärtssuche mit L_{xs}, L_{as}, Rückwärtssuche mit L_{xg}, L_{ag}
- Lösung (Pfad $p(s, g)$ von Start s zu Ziel g) gefunden, wenn ein Zustand u von s und g erreichbar ist (also bei beiden Suchen entdeckt wurde)
  Lösung $p(s, g) = p(s, u) \circ p(g, u)^{-1}$
- Bidirektionale Suche endet, wenn sich die „Grenzen“ der durch die Suche bisher entdeckten Mengen überschneiden
  $((L_{xs} \cup L_{as}) \cap (L_{xg} \cup L_{ag} \neq \emptyset)$

Speicherbedarf geringer als bei Breitensuche

- nur sinnvoll, wenn eindeutiger (gesuchter) Zielzustand bekannt ist, z.B. bei Kannibalen-Missionare-Rätsel, Navigation
- Erweiterung auf endliche Mengen explizit gegebener Zustände möglich (Betrachtung von Zustandsmengen in Suchknoten)
- meist ungeeignet, wenn Zielzustände durch zu erfüllende Bedingung definiert sind
  (z.B. Spiele mit Zielbedingung wie Schach-Matt, kein Zug möglich), mehreren Zielzuständen (gleicher oder verschiedener Güte)

Beispiel Schiebefax

- Zustände $u \in \{0, \ldots, 8\}^{3\times 3}$, 3×3-Matrix mit Einträgen $\{0, \ldots, 8\}$ (jede Zahl genau einmal, 0 bezeichnet Lücke / leeres Feld)
- Zulässige Züge: Verschieben des leeren Feldes auf ein Nachbarfeld d. h. Vertauschen von 0 und einem Wert in einem Nachbarfeld (gleicher Zeilen- oder Spaltenindex)
- Zielkonfiguration

1	2	3
8	0	4
7	6	5

- Aufgabeninstanz: gegebene Ausgangskonfiguration (Matrix), z.B.

8	0	3
2	1	4
7	6	5

- Lösung: Folge von zulässigen Zügen (Bewegung der Lücke 0) von der Ausgangs- zur Zielkonfiguration
- Bewertung der Lösung: Anzahl der Züge (Länge der Lösungsfolge)

Heuristische Suche – Motivation

Heuristik: Effizienzsteigerung durch Zusatzinformationen
(z.B. Erfahrungswerte)

Anwendung bei

- falls vollständige Suche zu aufwendig
- Aufgaben mit mehreren Lösungen (z.B. Wege in Graphen)
- unterschiedliche Qualität der Lösungen
  (z.B. Länge des Weges)
- Suche nach optimalen Lösungen (z.B. kürzester Weg)

Ziele:

- Wahl einer geeigneten Such-Reihenfolge, unter welcher gute Lösungen zuerst gefunden werden
- Verwerfen von Knoten, die wahrscheinlich nicht zu einer Lösung führen
  (beabsichtigte Verletzung der Fairness-Eigenschaft)

Schätzfunktionen

Ziel: sinnvolle Auswahl der in jedem Schritt zu expandierenden Knoten unter Verwendung von Zusatzinformationen

Schätzfunktion (heuristische Funktion) $h : V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$
(oder in eine andere geordnete Menge)
Schätzung der erwartete Restkosten vom Knoten u bis zum Ziel

h repräsentiert die Zusatzinformation

ohne Zusatzinformation: uninformierte (blinde) Suchverfahren, z.B. Tiefensuche, Breitensuche, Dijkstra

mit Zusatzinformation: informierte Suchverfahren
heuristische Suche

Eigenschaften von Heuristiken

Schätzfunktion $h : V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ heißt

perfekt (Schätzfunktion $H(u)$), gdw. $\forall u \in V : H(u) =$ genau die Kosten einer optimalen Lösung durch u ($H(u) = \infty$, falls keine Lösung über u existiert)

zielerkennend gdw. für jeden Lösungsknoten $u \in V$ gilt $h(u) = 0$

sicher gdw. aus jedem Knoten $u \in V$ mit $h(u) = \infty$ ist kein Lösungsknoten erreichbar
d.h. $\forall u : (h(u) = \infty \to H(u) = \infty)$

konsistent gdw. für jeden Knoten $u \in V$ und alle Folgeknoten v von u gilt $h(u) \leq w(u, v) + h(v)$
($w(u, v)$ Kosten des Übergangs von u nach v)

nicht-überschätzend gdw. für jeden Knoten $u \in V$ gilt $h(u) \leq H(u)$

Aus nicht-überschätzend folgt sicher und zielerkennend. (ÜA)
Aus zielerkennend und konsistent folgt nicht-überschätzend. (ÜA)

Besten-Suche

(best-first-search)

Allgemeines Suchverfahren mit Bewertungsfunktion
$f : V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$
mit folgender Strategie zur Auswahl der in jedem Schritt zu expandierenden Knoten:

- Knoten werden aufsteigend nach Bewertung $f(u)$ expandiert,
- Expansion des Knotens u mit dem geringsten Wert $f(u)$ zuerst
- Verwaltung von L_a als priority queue

Beispiel: Suche eines kürzesten Weges zwischen Orten A und B

- Bewertungsfunktion $f(u)$: bisherige Kosten bis zum Ort u (ohne Schätzfunktion, uniforme Kostensuche, Dijkstra)
- Bewertungsfunktion $f(u)$:
  Luftlinienentfernung des Ortes u von B (nur Schätzfunktion)

Besten-Suche – Eigenschaften

zwei Methoden:

1. Knoten mit großen Werten möglichst spät expandieren
2. Knoten mit großen Werten nicht expandieren

- Bestensuche mit einer beliebigen Bewertungsfunktion ist nicht immer optimal.
- Bestensuche nach Methode 1 (fair) ist vollständig.
- Bestensuche nach Methode 2 ist nicht immer vollständig.

Greedy-Suche (kleinste geschätzte Restkosten)

Idee: Suche zuerst in Teilbäumen der noch nicht besuchten Knoten mit den geringsten (geschätzten) noch aufzuwendenden Kosten

Heuristische Funktion $h : V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$

$h(v)$ ist Abschätzung des von Knoten v aus den noch notwendigen Kosten zum Erreichen eines Zielzustandes

Greedy-Suche:
Besten-Suche mit Bewertungsfunktion $f : V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$,
wobei für jeden Knoten $v \in V$ gilt

$$f(v) = h(v)$$

Eigenschaften der Greedy-Suche:

- optimal?
- vollständig?

Schiebefax – Heuristische Funktionen

Heuristische Funktionen $h_i : \{0, \ldots, 8\}^{3\times 3} \to \mathbb{N}$ mit

- h_1 Anzahl der Zahlen, die sich nicht an ihrer Zielposition befinden
- h_2 weitester Abstand einer Zahl zu seiner Zielposition
- h_3 Summe der Manhattan-Abstände jeder Zahl zu seiner Zielposition

Tafel: Bestensuche mit Bewertungsfunktionen $f(u) = h_i(u)$

Qualität der Schätzfunktionen:

- gute Trennung verschiedener Zustände
- fair: zu jedem $n \geq 0$ existieren nur endlich viele $u \in V$ mit $h(u) \leq n$

Bisherige Kosten

Kostenfunktion $k : V \to \mathbb{R}_{\geq 0}$

$k(u)$ Kosten des besten (bisher bekannten) Pfades vom Startzustand zum Zustand u

Kostenfunktion $k : V \to \mathbb{R}_{\geq 0}$ heißt

streng monoton wachsend , falls für alle Knoten u und alle Nachfolger v von u gilt $k(u) < k(v)$

Beispiele für Kostenfunktionen:

- Tiefe des Knotens im Suchbaum,
- maximale Entfernung vom Startknoten

A*-Suche (kleinste Gesamtkosten)

Idee: Suche (zuerst / nur) in Teilbäumen der noch nicht besuchten Knoten mit dem geringsten Wert der Schätzfunktion
(Summe von bisherigen und geschätzen zukünftigen Kosten)
Funktionen

- $k : V \to \mathbb{R}_{\geq 0}$ – geringste bisher bekannte Kosten von einem Startzustand zu v
- $h : V \to \mathbb{R}_{\geq 0}$ – geschätzte Kosten von v zu einem Endzustandr Lösung

A*-Suche: Besten-Suche mit Schätzfunktion $f : V \to \mathbb{R}_{\geq 0}$, wobei für jeden Knoten $v \in V$ gilt

$$f(v) = k(v) + h(v)$$

Eigenschaften der A^*-Suche: Wann optimal? Wann vollständig?

IDA*-Suche: Kombination von

- schrittweiser Vertiefung (iterative deepening)
- A*-Suche

Anwendungen

Planungsprobleme und kombinatorische Suchprobleme, z.B.

- Routenplanung
- Navigation (z.B. autonomer Fahrzeuge)
- TSP in Graphen
- Verlegen von Leitungen
- Schaltkreis-Layout
- Scheduling
- Produktionsplanung

Was bisher geschah

- Daten, Information, Wissen
- Wissensrepräsentation und -verarbeitung
- Wissensbasierte Systeme

Wissensrepräsentation:

- Zustandsübergangssystem:
  Graph mit markierten Knoten
  (Zustände und deren Eigenschaften)
- Startzustand
- Eigenschaften der Zielzustände

Lösung: Pfad vom Start- zu einem Zielzustand

Wissensverarbeitung: Suche im Graphen

uninformiert: Breiten-, Tiefen-, Gleiche-Kosten-Suche

informiert: Heuristik, Greedy-, A*-Suche

Zwei-Personen-Spiele

Zwei-Personen-(Brett)spiel

- aktueller Spielzustand immer für beide Spieler sichtbar (vollständige Information)
- einer gewinnt, der andere verliert (Nullsummenspiel)

Wissensrepräsentation (Spielbaum):

- Menge von Zuständen (Min- und Max-Zustände)
- Startzustand
- Endzustände (ohne Fortsetzung)
- Nachfolgermenge $S(v)$ = Menge von Zuständen (nach zulässigen Zügen)
- Bewertungsfunktion: Menge der Endzustände $\rightarrow \mathbb{Z}$
  - positiv: Spieler (1, Max, beginnt) gewinnt
  - negativ: Gegner (0, Min) gewinnt

Beispiel Nim (Variante)

- n Münzen auf einem Stapel
- Spielzug: Teilen eines Stapels in zwei nichtleere Stapel ungleicher Größe
- Sobald ein Spieler keinen Zug mehr ausführen kann, hat er verloren (und der andere gewonnen).

neutrales Spiel: für jeden Spieler sind dieselben Spielzüge möglich

(eine mögliche) Modellierung als Zustandsübergangssystem:

Zustände: $S : \mathbb{N} \to \mathbb{N}$ (Multimenge)
Münzanzahl $\mapsto$ Anzahl der Stapel mit dieser Zahl an Münzen

Startzustand: $S(n) = 1 \wedge \forall i \neq n : S(i) = 0$

Endzustände: kein Zug möglich

Übergänge: (erlaubte Züge) für $x = x_1 + x_2 \wedge x_1 \neq x_2 \wedge x_1 x_2 \neq 0$:
$S \to S'$ mit
$S'(x) = S(x) - 1$
$\wedge S'(x_1) = S(x_1) + 1 \wedge S'(x_2) = S(x_2) + 1$
$\wedge \forall i \in \mathbb{N} \setminus \{x, x_1, x_2\} : S'(i) = S(i)$

Minimax-Werte in vollständigen Spielbäumen

- vollständiger Spielbaum $B = (V, E)$
- Bewertung der Endzustände (Blätter im Spielbaum) bekannt
- Fortsetzung der Bewertungsfunktion von den Blättern auf alle Knoten im Spielbaum $b : V \to \mathbb{Z}$

rekursive Berechnung (Minimax-Algorithmus) des Wertes eines Knotens v im Spielbaum:

$$m(v) = \begin{cases} b(v) & \text{falls } v \text{ Endzustand} \\ \max\{m(u) \mid u \in S(v)\} & \text{falls } v \text{ Max-Knoten} \\ \min\{m(u) \mid u \in S(v)\} & \text{falls } v \text{ Min-Knoten} \end{cases}$$

Beispiele (Tafel):

- Spielbaum,
- Nim mit $n = 7$

Spielstrategie für Spieler 1 (Max):
Zug wählen, der zum Zustand mit höchstem Minimax-Wert führt

Minimax-Werte mit Heuristik

bei unvollständigem Spielbaum: Kombination von

- heuristischer Knotenbewertung
- Berechnung der Minimax-Werte

Beispiele (Tafel): Tic-Tac-Toe
kein neutrales Spiel: jeder Spieler darf nur sein Symbol setzen
(wie z.B. auch Gomoku, Schach, Go)
mit Schätzfunktion für den Spieler am Zug:
Differenz der Anzahlen der noch nicht blockierten Gewinntripel

auch dabei Spielstrategie für Spieler 1 (Max):
Zug wählen, der zum Zustand mit höchstem Minimax-Wert führt

α-β-Suche

Berechnung der Minimax-Werte in Teilbaum beschränkter Tiefe durch

- Tiefensuche mit zusätzlicher Verwaltung eines Intervalles $[\alpha(u), \beta(u)]$ für jeden Knoten u
- Minimax-Wert $w(u)$ liegt immer im Intervall $[\alpha(u), \beta(u)]$
- zu Beginn für alle Knoten u: $[-\infty, \infty]$ ($\alpha(u) = -\infty$, $\beta(u) = \infty$)
- schrittweise Annäherung der Intervallgrenzen an $w(u)$

α-β-Pruning:
Bei Berechnung des Minimax-Wertes des Knotens u Berechnungen für alle Enkel von u auslassen (abtrennen), wenn diese die Intervallgrenzen $\alpha(u)$ und $\beta(u)$ nicht verbessern können
Abtrennen jedes Kindes v eines

Min-Knotens u, falls $\beta(u) \leq \alpha(v)$
(Min kann in u durch Wahl eines zuvor untersuchten Kindes v' den geringeren Minimax-Wert $\beta(u)$ erreichen als durch Wahl von v)

Max-Knotens u, falls $\alpha(u) \geq \beta(v)$
(Max kann in u durch Wahl eines zuvor untersuchten Kindes v' den höheren Minimax-Wert $\alpha(u)$ erreichen als durch Wahl von v)

Automatische Berechnung heuristischer Funktionen

Ziel: automatische Bewertung von Spielzügen, d.h.
Bewertung von Knoten v (Spielsituationen) im Spielbaum
Beispiel: Monte-Carlo-Baum-Suche MCTS
Idee: Berechnung des Wertes für v aus simulierten Spielen

- simuliertes Spiel i: Folge von (zufälligen) Zügen bis Spielende
- Bewertung des Knotens v im Spielbaum durch
  - n simulierte Spiele, beginnend in v
  - $\forall i \in \{1, \ldots, n\}$: R_i = Ergebnis des Spieles i
  - Berechnung des Wertes von v aus Ergebnissen $\{R_1, \ldots, R_n\}$ z.B. Mittel, Gewinnwahrscheinlichkeit

übliche Modifikationen:

- Integration von spezifischem Wissen (z.B. Standard-Antworten, Eröffnungsbibliotheken) statt ausschließlich zufälliger Züge
- Spiele nur bis festgelegter Anzahl von Zügen
- Backpropagation: Update der Werte auf Pfad zur Wurzel
- Speichern und laufendes Anpassen schon berechneter Bewertungen von Spielsituationen über mehrere Spiele (Lernen)

Beispiele: Monte-Carlo Go (1993), AlphaGo (erfolgreich 2015/16)

Was bisher geschah

Abgrenzung der Begriffe: Daten, Information, Wissen, Intelligenz
Symbolische KI: Suchprobleme mit / ohne Gegenspieler

Wissensrepräsentation (Darstellung des Kontextes):
Zustandsübergangssystem, Zielbedingung,
(Verfahren zur Bestimmung der) Werte der Zielknoten

Lösung: Pfad zu einem Zielzustand im Zustandsübergangssystem
Spielstrategie

Wissensverarbeitung (Lösungsverfahren): Pfadsuche (informiert, uninformiert), MiniMax-Werte, α-β-Suche

Heuristische Funktionen:

- notwendig für informierte Suche
- Eigenschaften
- mitunter automatische Berechnung möglich

Beispiel für Kombination symbolischer und statistischer Verfahren

- heuristische Spielbaum-Suche (symbolisch)
- automatische Bewertung der Knoten durch Simulation mehrerer Spiele (statistisch)

Monte-Carlo-Baum-Suche (MCTS)

Entscheidungsunterstützung

Ziel der KI: intelligente Entscheidungen treffen oder vorschlagen (analog menschlichen Experten)

Entscheidung:
Auswahl einer aus mehreren Optionen abhängig von der (aktuellen) Situation, z.B.

- nächster zu expandierender Knoten im Suchbaum
- nächster Spielzug
- Einordnung von Objekten
- Diagnosen
- Kreditwürdigkeit
- Therapieansätze

Bewertung von Objekten / Situationen

Ziel: Bewertung von Objekten (Fällen) anhand bestimmter Merkmale
für Mengen O aller Objekte, W aller möglichen Werte
Funktion $f : O \to W$, z.B.

- O: Knoten in Agenda, W: Priorität $\in \mathbb{R}_{\geq 0}$
- O: mögliche Spielzüge, W: Minimax-Wert $\in \mathbb{R}$
- O: Personen, W: Alter $\in \mathbb{N}$
- O: Personen, W: Geschlecht $\in \{m, w, d\}$
- O: Belegungen $\beta : P \to \{0, 1\}$, W: Wahrheitswert $\in \{0, 1\}$
- O: digitale Bilder, W: $\in 2^{\{\text{Katze,Hund,Maus}\}}$

Bewertung der Objekte anhand ihrer Merkmale, z.B.

- Position, bisherige und geschätzte zukünftige Kosten
- Eigenschaften des Spielzustandes, z.B. noch nicht blockierte Tripel
- Wahrheitswert $[\![\varphi]\!]_\beta \in \{0, 1\}$ für gegebene Formel $\varphi \in \mathrm{AL}(P)$
- Anordnung der Pixel / Farbwerte im Bild (Matrix)

Klassifikation

Ziel: Einteilung von Objekten (Fällen) anhand bestimmter Merkmale in (eine oder mehrere) Klassen

Beispiel: Auswahl der Lösung für einen Fall aus einer Menge gegebener Alternativen

Klassifikation: Bewertung / Funktion $f : O \to W$ (oder $f : O \to 2^W$) mit diskreter (meist endlicher) Menge W von Klassen

M Menge von Merkmalen m (Symptome, Attribute)
jedes Merkmal m mit Menge V_m möglicher Werte (Ausprägungen),
z.B. $M = \{a, g\}$ mit Alter a, $V_a = \mathbb{N}$, Geschlecht g,
$V_g = \{m, w, d\}$
Merkmalsraum: $\bigtimes_{m \in M} V_m$

O : $m \to V_m$ Menge aller Objekte (Fälle)
Jedem Objekt $o \in O$ sind seine Merkmalswerte $m_o \in \bigtimes_{m \in M} V_m$ (Punkt im Merkmalsraum) zugeordnet.
z.B Tom $\in O$ mit $m_{\text{Tom}} = [a \mapsto 5, g \mapsto m]$ ($m_{\text{Tom}} = (5, m)$)

K ($= W$) Menge aller Klassen (Diagnosen, Lösungen),
Klasse repräsentiert Teilmenge des Merkmalsraumes $\bigtimes_{m \in M} V_m$
z.B. Baby, Kind, Mädchen, Rentner, Hörer dieser Vorlesung

oft hilfreich: geometrische Interpretation

Beipiel

Merkmale (Attribute) mit Menge möglicher Werte (Ausprägungen)
Beispiele:

- Name: Tom, Tina, Anna, Paul
- Geschlecht, Werte: w, m, d
- Alter: natürliche Zahl

Objekte definiert durch Zuordnung:
Merkmal $\rightarrow$ Wert
Beispiele:

- Name $\mapsto$ Tina, Alter $\mapsto$ 7, Geschlecht $\mapsto$ w
- Name $\mapsto$ Paul, Alter $\mapsto$ 87, Geschlecht $\mapsto$ m

Klassen Kombination von Merkmalswerten
definiert Menge von Objekten
Beispiele:

- Rentner: Geschlecht $=$ m und Alter ≥ 65
- Mädchen: Geschlecht $=$ w und Alter ≤ 14
- Test vor Röntgen:
  Geschlecht $=$ w und Alter $\in [15, 50]$

Klassifikationsprobleme

gegeben: Objekt $o \in O$ mit Merkmalswerten m_o
Zuordnung $K \to X_K$ mit $X_K \subseteq M$ (Extension)
jede Klasse definiert durch Merkmalswerte
(z.B. Produkt von Intervallen)

gesucht: Zuordnung Objekt $o \in O$ zu Klasse K mit $m_o \in X_K$

Beispiel: Klassifikation von

- Objekte: Adler, Biber, Elefant, Fledermaus, Frosch, Gorilla, Hecht, Pinguin, Specht
- Merkmale: Flügel, Flossen, Federn, Schuppen, Fell, kann fliegen, kann schwimmen, legt Eier
- Klassen: Säugetier, Fisch, Vogel

Diagnose-Probleme

Spezialfall von Klassifikationsproblemen:

- Objekte: Fälle
- Merkmale: Fragen, Tests
- Merkmalswerte: Antworten
- Klassen: Diagnosen

Anwendungen, z.B. in

- Medizin:
  - Krankheitserkennung
  - Entscheidung für eine Therapie
  - Auswertung von Studien
- Technische Systeme:
  - Konfiguration
  - Feststellung von Störungen
  - Entscheidung für Vorgehen bei Behebung

Klassifikationsarten und -verfahren

sichere Klassifikation: klar begrenzte Klassen,
Zuordnung Objekt $\rightarrow$ Klasse immer wahr oder falsch
Methoden: z.B. Regelsysteme, Entscheidungsbäume, Entscheidungstabellen

fallbasierte Klassifikation:
Suche nach Lösungen für ähnliche Fälle
Grundfrage: Wann sind Fälle ähnlich zueinander?
verschiedene Ansätze, z.B. Hamming-Abstand der Merkmalsvektoren
Verwaltung großer Datenmengen (vorangegangener Fälle)

unsichere / heuristische / statistische Klassifikation
Zuordnung Objekt $\rightarrow$ Klasse nicht immer klar, graduell
Methoden: z.B. neuronale Netze, heuristische Entscheidungsbäume und -tabellen, Regelsysteme mit gewichteten Regeln
statistische, unscharfe Klassifikation

modellbasiert: Merkmale = Interpretation von Beobachtungen bzgl. eines Modells des Verhaltens des Agenten / Systems

Sichere Klassifikation

typische Verfahren:

- Mengen von Regeln ohne / mit Alternative
- Entscheidungsbäume, -tabellen, BDDs
- logische Programme (Prolog)
  logisches Programm: Menge von Fakten und Regeln
- deduktive Datenbanken (erlauben rekursive Abfragen):
  extensionale Datenbasis: Datenbank (Menge von Relationen)
  repräsentiert die Faktenmenge
  Regelsysteme als Datalog-Programm (Anfragen)
  intensionale Datenbasis: Menge aller ableitbaren Fakten

Heuristische / statistische Klassifikation

bei unvollständigem, unsicherem, unzuverlässigem Wissen

Lösung (Klasse, Diagnose) ist Verdacht
Lösungen werden bewertet und verglichen, um die / eine beste Lösung zu finden

häufig auch numerische Bewertung von

- Plausibilität
- Zuverlässigkeit, Glaubwürdigkeit
- Häufigkeit

Klassifikation durch Regeln

Wissensbasis: Menge (aussagenlogischer) Regeln
verschiedene Repräsentationsformen möglich, z.B.

- Menge von Regeln
- Formelmenge $\subseteq \mathrm{AL}(P)$
- Entscheidungsbaum
- Entscheidungsdiagramm
- Entscheidungstabelle

Anfrage (Fall): Menge von Fakten als

- Formel
- Formelmenge
- Zuordnung: Merkmal $\rightarrow$ Wert

Lösung: Klasse (z.B. ja / nein) ,
evtl. mit Modell / Begründung

Problemlöseverfahren: abhängig von der Art der Regeldarstellung

Beispiel Rabatt

1. Wenn Kundenkarte vorhanden, bisheriges Zahlungsverhalten ok und und Bestellwert hoch, wird Rabatt gewährt.
2. Wenn bisheriges Zahlungsverhalten nicht ok oder Bestellwert gering, wird kein Rabatt gewährt.
3. Wenn keine Kundenkarte vorhanden ist, wird kein Rabatt gewährt, aber eine Kundenkarte angeboten.

Im Folgenden (Tafel) repräsentiert durch:
Regelmenge, Entscheidungstabelle, Entscheidungsbaum, BDD

Entscheidungstabellen

kompakte Darstellung einer endlichen Menge von Regeln

Aufbau einer Entscheidungstabelle:

- Bedingungsteil
  je Bedingung eine Zeile, je Regel eine Spalte
- Folgerungs- bzw. Aktionsteil
  je Folgerung bzw. Aktion eine Zeile
  in jeder Spalte alle Folgerungen / Aktionen der entsprechenden Regel markiert.
- Regeln werden in der angegebenen Reihenfolge nacheinander ausgewertet (von links nach rechts).

Klassifikation durch Entscheidungstabellen

Wissensbasis (Kontextwissen):
Entscheidungstabelle (repräsentiert Menge von Regeln)

Problem (konkreter Fall):
Menge von Fakten (Aussagen, Merkmalswerten)
Menge möglicher Aussagen (oder Aktionen)

Lösung: zum konkreten Fall passende Aussagen (oder Aktionen)

Wissensverarbeitung (Problemlösung):
Suche der gegebenen Aussagenkombination im Bedingungsteil der Entscheidungstabelle,
Ablesen der Aussagen (oder Aktionen) im Folgerungsteil

Eigenschaften von Entscheidungstabellen

Entscheidung zwischen mehreren Alternativen
im Beispiel: Annahme / Ablehnung / Rückruf

- vollständig:
  je Kombination von Bedingungen eine Spalte
  (Wahrheitswerttabelle)
- konsolidiert (kompakte Darstellung):
  Zusammenfassung von Spalten mit denselben Aktionen
  (Wildcards in Bedingungen)

Übersetzung: Regelmenge $\leftrightarrow$ Entscheidungstabelle

- Regelmenge $\rightarrow$ Entscheidungstabelle
  (i.A. keine vollständige ET)
- Entscheidungstabelle $\rightarrow$ Regelmenge (einfach):
  jede Spalte repräsentiert eine oder mehrere Regeln
  (nach Anzahl der markierten Folgerungen, Aktionen)

Verknüpfung mehrerer Entscheidungstabellen

Menge $\{T_1, \ldots, T_n\}$ von Entscheidungstabellen

spezielle Aktion: Übergang zu einer Tabelle T_i

Ablaufsteuerung durch mehrere Entscheidungstabellen:

- Sequenz: Übergang zur Folge-ET unter allen Bedingungen
- Verzweigung: von Bedingungen abhängiger Übergang zu verschiedenen Folge-ET
- Rekursion: Übergang zur selben ET

Regeln ohne / mit Alternative

Regeln ohne Alternative: Falls A, dann B

als logische Formel: $(A \to B) \equiv (\neg A \vee B)$

Regeln mit Alternative: Falls A, dann B sonst C

als logische Formel:
$(A \to B) \wedge (\neg A \to C) \equiv (A \wedge B) \vee (\neg A \wedge C) = \mathrm{ite}(A, B, C)$
mit dreistelligem Junktor ite

$\{\mathrm{ite}, \mathbb{t}, \mathbb{f}\}$ ist eine Junktorbasis. (ÜA)

Zu jeder aussagenlogischen Formel $\varphi \in \mathrm{AL}(P)$ existiert sogar eine äquivalente Formel ψ, die nur Junktoren aus $\{\mathrm{ite}, \mathbb{t}, \mathbb{f}\}$ enthält und jede Teilformeln mit Wurzel ite die Form $\mathrm{ite}(p, \varphi_1, \varphi_2)$ mit $p \in P$ hat (ite-Formel).

Regeln mit Alternative

(Induktive) Definition der Menge aller ite-Formeln

IA: Jede Aussagenvariable, $\mathbb{t}$ und $\mathbb{f}$ sind ite-Formeln.

IS: Sind $p \in P$ eine Aussagenvariable und φ und ψ ite-Formeln, dann ist auch $\mathsf{ite}(p, \varphi, \psi)$ eine ite-Formel

Beispiele (Tafel):

- $\neg p \equiv \mathsf{ite}(p, \mathbb{f}, \mathbb{t})$
- $p \wedge q \equiv \mathsf{ite}(p, \mathsf{ite}(q, \mathbb{t}, \mathbb{f}), \mathbb{f})$
- $p \rightarrow q \equiv \mathsf{ite}(p, q, \mathbb{t})$
- $b \wedge (a \vee c) \equiv \mathsf{ite}(a, \mathsf{ite}(b, \mathbb{t}, \mathbb{f}), \mathsf{ite}(c, \mathsf{ite}(b, \mathbb{t}, \mathbb{f}), \mathbb{f}))$

Klassifikation durch Entscheidungsbäume

z.B. `https://www.rki.de/SharedDocs/FAQ/COVID-Impfen/Flowchart_Allergieanamnese.pdf?__blob=publicationFile`

- Wissensbasis (Kontext): Entscheidungsbaum mit
  - inneren Knoten: Merkmale
  - Kanten: Merkmalsausprägungen (Werte)
  - Blättern: Klassen
- Repräsentation des Problems (Objekt, Fall):
  fallspezifische Merkmalswerte
- Repräsentation der Lösung:
  Zuordnung des Objektes (Falles) zu einer Klasse
  z.B. Symptome zu einer Diagnose
- Problemlöseverfahren:
  Beginn in der Wurzel,
  schrittweises Verfolgen der Kanten im Entscheidungsbaum
  entsprechend der Merkmalswerte in jedem Knoten,
  Ablesen der Klasse (Aussage, Aktion) im erreichten Blatt

Binäre Entscheidungsbäume

Binärer Entscheidungsbaum:
Graph $T = (V, E)$ mit den folgenden Eigenschaften:

- Baum T hat eine Wurzel $r \in V$,
- jeder innere Knoten hat Grad 2,
  je einen mit 0 und einen mit 1 markierten Nachfolger
- jedes Blatt hat Grad 0
- jeder Knoten hat genau einen Vorgänger

und Knotenmarkierungen

- innere Knoten: Variablen (Entscheidungskriterien, Merkmale)
- Blätter: Werte oder Aktionen

Beispiel: $b \wedge (a \vee c)$

Jeder binäre Entscheidungsbaum repräsentiert eine Boolesche Funktion.

Übersetzung: Regelmenge $\leftrightarrow$ Entscheidungsbaum

- Entscheidungsbaum $\rightarrow$ Regelmenge (einfach):
  Jeder Pfad von der Wurzel zu einem Blatt repräsentiert eine Regel.
  Innere Knoten und die dazu ausgewählten Kanten bestimmen die Voraussetzungen, das Blatt die Folgerung.
- Regelmenge $\rightarrow$ Entscheidungsbaum (weniger einfach), z.B. über ite-Formel oder Entscheidungstabelle

Allgemeine Entscheidungsbäume

Verallgemeinerung binärer Entscheidungsbäume:

- Knotenmarkierung: Variable mit mehreren möglichen Werten,
- innere Knoten mit beliebig vielen Nachfolgern
  (mehrere Alternativen möglich)

Allgemeiner Entscheidungsbaum:
Baum (gerichteter azyklischer Graph) mit den Eigenschaften

- eine Wurzel,
- jeder Knoten hat genau einen Vorgänger,
- Blätter, Knotenmarkierung: Klassen
- innere Knoten mit Knotenmarkierung: Merkmal,
  (Frage nach Merkmalswert)
  ausgehende Kanten markiert mit möglichen Werten oder
  Wertebereichen dieses Merkmals (Antworten)
  Jeder innere Knoten hat so viele Nachfolger wie mögliche
  Alternativen (Werte oder Wertebereiche)

Beispiel: Warten im Restaurant (Tafel)

Effektivere Darstellung binärer Entscheidungsbäume

Beispiel: $\varphi \in \mathsf{AL}(\{x, y, z, u\})$ mit
$\mathsf{Mod}(\varphi) = \{W_{0000}, W_{0011}, W_{1100}, W_{1111}\}$

Schrittweise Anwendung (ebenenweise von Blättern zur Wurzel) der folgenden Operationen auf einen Entscheidungsbaum T (oder ein BDD):

Verschmelzung von Knoten , die Wurzeln gleicher Teilbäume sind
(Vereinigung der Kantenmarkierungen)

Kontraktion von Kanten zwischen einem Knoten und seinem Kind, falls dieses sein einziges Kind ist
(Vereinigung der Knotenmarkierungen, gleichzeitige Entscheidung nach mehreren Kriterien)

Ergebnis: markierter gerichteter azyklischer Graph (DAG), hat meist weniger Knoten als T

Binary Decision Diagrams

kompakte Darstellungsform für Boolesche Funktionen
(und endliche Mengen)

Binary Decision Diagram (BDD):
markierter gerichteter azyklischer Graph (DAG) mit

Markierungen:

- innere Knoten: Variablen
- Kanten: 0 oder 1

und den folgenden Eigenschaften:

- BDD hat eine Wurzel,
- jeder innere Knoten hat Grad 2,
  je einen mit 0 und einen mit 1 markierten Nachfolger
- jedes Blatt hat Grad 0

ROBDDs

Variablenordnung: totale Ordnung $<$ auf Variablenmenge

BDD heißt genau dann bzgl. der Variablenordnung $<$ geordnet, falls auf jedem Pfad von der Wurzel zu einem Blatt die Variablen in aufsteigender Reihenfolge der Variablenordnung vorkommen.

Beispiel (Tafel): φ mit $\mathrm{Mod}(\varphi) = \{W_{0000}, W_{0011}, W_{1100}, W_{1111}\}$ mit Variablenordnungen $[x, y, z, u]$ und $[x, z, y, u]$

BDD heißt genau dann OBDD (ordered binary decision diagram), wenn es bzgl. einer Variablenordnung geordnet ist

OBDD heißt genau dann ROBDD (reduced ordered binary decision diagram), wenn keine der Reduktionsoperationen (Knotenverschmelzung, Kantenkontraktion) möglich ist.

ROBDD bilden für jede Variablenordnung kanonische Repräsentation (bis auf Isomorphie eindeutig)
Größe abhängig von Variablenordnung

Was bisher geschah

Daten, Information, Wissen, Lernen, Intelligenz

Ziel künstlicher Intelligenz: Unterstützen / Treffen von Entscheidungen

Verfahren:

- Kontext: Zustandsübergangssystem
- Lösung: Pfad im Zustandsübergangssystem, Strategie
- Lösungsverfahren: Pfadsuche (informiert, uninformiert)
  Spielstrategien, Minimax-Werte, α-β-Suche

(sichere) Klassifikation

- Kontext: Regelmenge, Fall (spezielle Merkmalswerte)
  Lösung: Modell(e)
  Lösungsverfahren: Bestimmung der Modellmenge, evtl. Auswahl
- Kontext: Entscheidungstabelle, Fall (spezielle Merkmalswerte)
  Lösung: Aktionsteil einer Spalte
  Lösungsverfahren: Finden der ersten Spalte (von links) mit zum Fall passendem Bedingungsteil
- Kontext: Entscheidungsbaum / (RO)BDD, Fall
  Lösung: Blatt im Baum
  Lösungsverfahren: Verfolgung des Pfades von der Wurzel aus anhand der fallspezifischen Merkmalswerte

Wissensrepräsentation durch Logiken

Klassische Logiken:

- Aussagenlogik:
  Repräsentation einfacher und zusammengesetzter Aussagen
- Prädikatenlogik:
  Wissen über Eigenschaften von und Beziehungen zwischen Objekten eines Bereiches

Prinzipien der klassischen Logik:

Zweiwertigkeit Jede Aussage ist wahr oder falsch.

ausgeschlossener Widerspruch Keine Aussage ist sowohl wahr als auch falsch.

Nichtklassische Logiken

- Nichtmonotone Logiken: unvollständiges Wissen
- mehrwertige Logiken:
  unsicheres und unscharfes Wissen
  fuzzy Logiken, probabilistische Logiken
- Modallogiken:
  Wissen über verschiedene mögliche Welten (Zustände)
  - Temporallogiken: Wissen über zeitliche Zusammenhänge (Zustandsübergangssysteme)
  - Beschreibungslogiken:
    Wissen über Begriffe und Zusammenhänge zwischen diesen
  - Epistemische Logiken:
    Wissen über Wissen (z.B. verschiedener Agenten)

Wissensverarbeitung in Logiken

Ziele:

- Beantwortung von Anfragen der Form: Gilt $\Phi \models \psi$?
  (Für welche Individuen) Gilt die Aussage ... unter den bekannten Voraussetzungen?
- Herleitung neuen Wissens: Für welche φ gilt $\Phi \models \psi$?
- Konsistenztests vorhandenen Wissens: Ist Φ erfüllbar?
- Konsistentes Zusammenfügen verschiedener Wissensquellen
  Ist $\Phi_1 \cup \Phi_2$ erfüllbar? falls nicht, maximale erfüllbare Teilmengen

Lösungsverfahren:

- Suche nach (intendierten) Modellen
- semantische Methoden:
  semantisches Folgern (z.B. für AL: Wahrheitswerttabellen, Entscheidungstabellen, Entscheidungsbäume, BDD, SAT-Solver)
- syntaktische Methoden:
  Schließen, Ableiten in logischen Kalkülen, Beweisen

Wiederholung Aussagenlogik: Syntax

Atome Syntax: elementare Formeln, Aussagenvariablen
Semantik: Wahrheitswert

Formeln Syntax (induktive Definition): AL(P)

- IA: Alle Atome sind Formeln. $P \subset \mathrm{AL}(P)$
- IS: $\varphi_1, \ldots, \varphi_n \in \mathrm{AL}(P) \rightarrow j(\varphi_1, \ldots, \varphi_n) \in \mathrm{AL}(P)$

Semantik: Modellmenge (Boolesche Funktion)

Literal Atom oder negiertes Atom

Klausel Disjunktion von Literalen
(entsteht beim Umformen von Implikationen, Regeln)

CNF Konjunktion von Klauseln

Wiederholung Aussagenlogik: Semantik

Belegung $W : P \to \{0, 1\}$

Wert von $\varphi \in \mathrm{AL}(P)$ unter Belegung W:

- $W(\varphi) = W(p)$ für $\varphi = p \in P$,
- induktive Berechnung von $W(\varphi)$ für zusammengesetzte Formeln φ

Modell (erfüllende Belegung) für $\varphi \in \mathrm{AL}(P)$:
$W : P \to \{0, 1\}$ mit $W(\varphi) = 1$

Modellmenge von $\varphi \in \mathrm{AL}(P)$:
$\mathrm{Mod}(\varphi) = \{W : P \to \{0, 1\} \mid W(\varphi) = 1\}$
(Boolesche Funktion, Wahrheitswerttabelle)

Formel $\varphi \in \mathrm{AL}(P)$ heißt

erfüllbar gdw. $\mathrm{Mod}(\varphi) \neq \emptyset$

unerfüllbar gdw. $\mathrm{Mod}(\varphi) = \emptyset$

allgemeingültig gdw. $\mathrm{Mod}(\neg\varphi) = \emptyset$

Wiederholung Semantisches Folgern

$\psi \in \mathsf{AL}(P)$ heißt genau dann (semantische) Folgerung aus $\Phi \subseteq \mathsf{AL}(P)$, wenn jedes Modell für Φ auch ein Modell für ψ ist.

Kurzform:

$$\Phi \models \psi \quad \text{gdw.} \quad \mathsf{Mod}(\Phi) \subseteq \mathsf{Mod}(\psi)$$

Beispiele:

- $\{p, p \to q\} \models q$
- $\{p, \neg(q \wedge p)\} \models \neg q$
- $\{p\} \models q \to p$
- $\{q \to p\} \not\models p$
- $\emptyset \models p \vee \neg p$
- $p \wedge \neg p \models q$

WH: Folgerungsrelation

Folgerungsrelation:
zweistellige Relation $\models \subseteq 2^{\mathsf{AL}(P)} \times \mathsf{AL}(P)$

Spezialfälle der Notation:

für $\Phi = \{\varphi\}$: $\varphi \models \psi$ (statt $\{\varphi\} \models \psi$)
für $\Phi = \emptyset$: $\models \psi$ (statt $\emptyset \models \psi$)

Fakt
Für jede Formel $\varphi \in \mathsf{AL}(P)$ gilt $\models \varphi$ genau dann, wenn φ allgemeingültig ist.

WH: Sätze über das Folgern

- Für endliche Formelmengen $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ gilt

$$\Phi \models \psi \quad \text{genau dann, wenn} \quad \bigwedge_{i=1}^{n} \varphi_i \models \psi$$

- $\varphi \equiv \psi$ gdw. $\varphi \models \psi$ und $\psi \models \varphi$

Für jede Formelmenge $\Phi \subseteq \mathsf{AL}(P)$ und jede Formel $\psi \in \mathsf{AL}(P)$ gilt:

- falls $\psi \in \Phi$ gilt $\Phi \models \psi$.
- $\Phi \models \psi$ gdw. $\mathsf{Mod}(\Phi) = \mathsf{Mod}(\Phi \cup \{\psi\})$
- $\Phi \models \psi$ gdw. $\Phi \cup \{\neg\psi\}$ unerfüllbar
- $\Phi \models \psi$ gdw. $\Phi \cup \{\neg\psi\} \models \mathbb{f}$

WH: Beispiel

Kontext (Wissensbasis): Wenn der Zug zu spät kommt und kein Taxi am Bahnhof steht, ist Tom nicht pünktlich.
Der Zug kam zu spät und Tom ist pünktlich.
Modellierung:
z – Zugverspätung, t – Taxi da, p – pünktlich
$\Phi = \{\underbrace{z \wedge \neg t \rightarrow \neg p}_{\text{Regel}}, \underbrace{z \wedge p}_{\text{Fakt(en)}}\}$

Frage Stand ein Taxi am Bahnhof?
Behauptung $\psi = t$
Gilt $\Phi \models \psi$?

Lösung
- semantisch, z.B. Modellmengen (WW-Tabellen)
- syntaktisch, z.B. aussagenlogische Resolution
- maschinell (Kombination), z.B. SAT-Solver

Regelbasierte Systeme

Wissensrepräsentation durch
Mengen von Regeln (ohne Alternative)

Regel: Falls φ, dann ψ
φ Rumpf (Voraussetzung), ψ Kopf (Folgerung)

als logische Formel $\varphi \rightarrow \psi$

aussagenlogische Regeln mit $\varphi, \psi \in \mathsf{AL}(P)$

prädikatenlogische Regeln mit $\varphi, \psi \in \mathsf{FOL}(\Sigma, \mathbb{X})$

mögliche Bedeutungen der Regel: Falls φ, dann ψ

Implikationsregel: Interpretation des Regelkopfes als Aussage
(φ, ψ Formeln)

Aktionsregel: Interpretation des Regelkopfes als Aktion
(φ Formel, ψ als Aktion)

Wissensrepräsentation durch Regeln

Regel (ohne Alternative): Implikation $r = (\varphi \to \psi)$

Kopf $H(r) = \psi$ (Folgerung)

Rumpf $B(r) = \varphi$ (Voraussetzung)

Aufbau und häufige Formen von Regeln:

Literal : einfaches oder negiertes (aussagen- oder prädikatenlogisches) Atom

Klausel : Disjunktion $l_1 \vee \cdots \vee l_n$ von Literalen $l_1, \ldots, l_n$

Regel : Implikation $r = (\varphi \to \psi)$ der Form

$$\begin{aligned} r &= ((b_1 \wedge \ldots \wedge b_n \wedge \neg c_1 \wedge \ldots \wedge \neg c_m) \to h) \\ &\equiv (\neg b_1 \vee \cdots \vee \neg b_n \vee c_1 \vee \cdots \vee c_m \vee h) \end{aligned}$$

mit Atomen $b_1, \ldots, b_n$ (positive Voraussetzungen) und $c_1, \ldots, c_m$ (negative Voraussetzungen) und h (Kopf)

definite Regel (ohne negative Voraussetzungen): Implikation $r = (\varphi \to \psi)$ mit

$$r = ((b_1 \wedge \ldots \wedge b_n) \to h) \equiv (\neg b_1 \vee \cdots \vee \neg b_n \vee h)$$

Hornklausel: enthält genau ein positives Literal

Interpretation von Regeln

Regel

$$r = ((b_1 \wedge \ldots \wedge b_n \wedge \neg c_1 \wedge \ldots \wedge \neg c_m) \rightarrow h)$$

mögliche Interpretation des Regelkopfes ψ als

Aussage , z.B. Kälte $\wedge$ Niederschlag $\rightarrow$ Schnee
Deduktionsregeln

Aktion , z.B. heiße Stirn $\rightarrow$ Fieber messen
Aktionsregeln, Produktionsregeln

Regelwissen

Regelbasiertes System (Produktionssystem) besteht aus

- Wissensbasis (F, R) mit
  - Regelmenge (meist Kontext)
    $R = \{(\varphi_i \to \psi_i) \mid i \in \{1, \ldots, n\}\}$ und
  - Faktenmenge (meist Fall)
    $F = \{f_1, \ldots, f_m\}$

  (meist alle φ_i Konjunktionen von Literalen und f_i, ψ_i Atome)
- evtl. Anfrage ψ
- Regelinterpreter (Inferenzverfahren):
  definiert das Ableiten neuer Fakten durch Anwendung der Regeln

Regelbasiertes Problemlösen

gegeben:

- Wissensbasis $P = F \cup R$ (z.B. logisches Programm)
  Jede Regel mit Variablen repräsentiert die Mengen aller Instanzen.
- Anfrage (Hypothese) h
  evtl. mit Variablen

gesucht:

- Nachweis, dass die Hypothese h aus der Wissensbasis folgt
  Lösung: erfüllende Belegung der Variablen
  (falls in Anfrage enthalten)

übliche Verfahren:

Rückwärtsverkettung: Suche nach Antworten auf eine gezielte Anfrage (zielorientiert), z.B. Prolog

Vorwärtsverkettung: Ableitung neuen Wissens als Folgerungen aus Faktenmengen (datenorientiert), z.B. Datalog

Prinzip Rückwärtsverkettung

gegeben:

- Wissensbasis $P = F \cup R$ mit
  - endliche Regelmenge R (Klauseln)
  - endliche Faktenmenge F (Literale)
- Anfrage (Hypothese) h (Literal)

rekursive Definition:
Hypothese h ist aus der Wissensbasis (R, F) ableitbar, falls

IA: $h \in F$ (evtl. Nutzer fragen) oder

IS: es existiert eine Regel $(l_1 \wedge \cdots \wedge l_n \rightarrow h) \in R$ und alle $l_1, \ldots, l_n$ aus (R, F) ableitbar.
(Rekursion: alle $l_1, \ldots, l_n$ werden neue Hypothesen)

Beispiel Rückwärtsverkettung

Beispiel:

Kontext (Wissensbasis, Regelmenge) $R = \{R1, R2, R3\}$ mit

R1 Schollen sind Fische. $s \rightarrow f$

R2 Zu Steaks serviert sie Bier. $m \rightarrow b$

R3 Zu Fisch serviert sie Wein. $f \rightarrow w$

Fall (Faktenmenge) $F = \{F1\}$:

F1 Tina kocht heute Scholle. s

Frage: Serviert Tina heute Wein? (Hypothese, zielorientiert)
wird positiv beantwortet.

Schlussschritte aus R und F:

1. w wird gezeigt durch (neue Hypothese) f (R3)
2. f wird gezeigt durch s (R1)
3. s ist ein Fakt in der Wissensbasis (F)

Verfahren: schrittweise Generierung neuer Hypothesen
(welche die Wahrheit der Anfrage belegen könnten)

Rückwärtsverkettung

Anwendung der Regeln (Regelinstanzen)
von rechts (Kopf) nach links (Rumpf)

+ zielgerichtete Suche,
  keine Abfrage unnötiger Fakten
- Hypothese (Anfrage) benötigt

Idee der Programmiersprache Prolog (später mehr dazu)

Prinzip Vorwärtsverkettung

gegeben:

- Wissensbasis (R, F) mit
  - Kontext: endliche Regelmenge R
  - Fall: endliche Faktenmenge F

Anwendbarkeit von Regeln:
Regel(instanz) feuert (ist anwendbar) in einer Faktenmenge gdw. Voraussetzung (Regelrumpf) in dieser Faktenmenge erfüllt.
Verfahren: Schrittweise Erweiterung der Faktenmenge um gültige Fakten (Köpfe feuernder Regeln)
rekursive Definition:
Menge aller Folgerungen (Grundatome) aus Wissensbasis (R, F):
(bzgl. $\subseteq$) kleinste Menge $T(R, F)$ mit

IA: $F \subseteq T(R, F)$

IS: Falls $p_1, \ldots, p_n \in T(R, F)$
und $(p_1 \wedge \cdots \wedge p_n \rightarrow h) \in R$
dann ist auch $h \in T(R, F)$
(Rekursion: alle h werden neue Fakten)

Beispiel Vorwärtsverkettung

Beispiel: aus der Wissensbasis

R1	Babies sind Kinder.	$\forall x : b(x) \rightarrow k(x)$
R2	Männliche Kinder sind Jungen.	$\forall x : (m(x) \wedge k(x)) \rightarrow j(x)$
R3	Weibliche Kinder sind Mädchen.	$\forall x : (w(x) \wedge k(x)) \rightarrow g(x)$
F1	Tom ist ein Baby.	$b(T)$
F2	Tom ist männlich.	$m(T)$

folgt (ohne gezielte Anfrage): Tom ist ein Kind. Tom ist ein Junge.
Instanz $b(T) \rightarrow k(T)$ der Regel R1 feuert in der Faktenmenge
$F = \{b(T), m(T)\}$,
weil $b(T) \in F$, also Voraussetzung (Regelrumpf) in F erfüllt.
schrittweise Erweiterung der Faktenmenge um gültige Fakten:

$$
\begin{aligned}
F_0 &= \{b(T), m(T)\} \\
F_1 &= \{b(T), m(T), k(T)\} \quad \text{wegen R1} \\
F_2 &= \{b(T), m(T), k(T), j(T)\} \quad \text{wegen R2} \\
F_3 &= F_2 \qquad \text{(Fixpunkt)}
\end{aligned}
$$

Typische regelbasierte Systeme

Regelbasiertes System (Produktionssystem): (endliche) Mengen

- R der Regeln, oft einer bestimmten Form (z.B. definit)
- F der Fakten

häufige Formen der Bedingungen:

- $a = b$, $a \neq b$ (Gleichheit von Objekten)
- $p(a), r(a, b, c)$ (Eigenschaft / Zusammenhang von Objekten)
- $w(a) = v$ (Attributwerte von Objekten)

übliche Aktionen:

- direkte Änderung der Wissensbasis
  (z.B. Erweiterung, Korrektur)
- Rückfragen (Antwort führt zur Änderung der Wissensbasis)
- Handlungsanweisungen (z.B. Reglereinstellung)
  führt zu Zustandsänderungen (Änderung der Wissensbasis)

Wissensverarbeitung häufig durch
Vorwärtsverkettung (Produktion) mit Konfliktlösestrategien

Konflike bei Aktionsregeln

gegeben: Faktenmenge F, Regelmenge R (z.B. definit)

Regel $b_1 \wedge \cdots \wedge b_n \rightarrow h$ feuert in F genau dann , wenn ihr Rumpf $b_1 \wedge \cdots \wedge b_n$ in F erfüllt ist (also wenn $\{b_1, \ldots, b_i\} \in F$)

Problem:

- Faktenmenge (Zustand) F' mit mehreren anwendbaren Regeln aus R, die gegenteilige Aktionen auslösen
- Aktionsteil welcher Regel soll ausgeführt werden?

Konfliktmenge zu einer Faktenmenge F':
Menge aller in F' anwendbaren Regeln aus R

Beispiele

Regeln: R1 Wenn Besuch zum Essen kommt,
gibt es Weiß- oder Rotwein, aber nie beides.
R2 Zum Fisch gibt es Weißwein.
R3 Wenn Bob da ist, gibt es Rotwein.

Fakten: F Bob kommt zum Fischessen.

Frage: Welches Getränk soll serviert werden?

Konfliktmenge in $\{F\}$: $\{R1, R2, R3\}$

Regeln: R1 Vögel können fliegen.
R2 Pinguine sind Vögel.
R3 Pinguine können nicht fliegen.

Fakten: F Tweety ist ein Pinguin.

Frage: Kann Tweety fliegen?

nach einer Anwendung von R2: Faktenmenge $\{p(t), v(t)\}$
Konfliktmenge in $\{p(t), v(t)\}$: $\{R1, R3\}$

Konfliktlösestrategien

Konfliktlösung in Faktenmenge F durch Auswahl einer Teilmenge der Konfliktmenge , deren Anwendung nicht widersprüchlich ist.

oft durch (schrittweises) Entfernen Ignorieren ausgewählter Regeln

(einige) Konfliktlösestrategien:

Refraktionsprinzip: Keine Regelinstanz feuert zweimal direkt nacheinander

Präferenzen: Bei der Wissensrepräsentation mit definierte Ordnung auf Regeln
Regeln niederer Präferenzen feuern nicht, wenn eine Konflikt-Regel höherer Präferenz angewendet wurde.

Spezifität: spezifischere Regelinstanzen werden bevorzugt

Aktualität: erstmals feuernde Regeln werden bevorzugt

zufällige Auswahl einer Regel

also auch hier: Lösung anhand
Bewertung / Klassifikation der Optionen / Aktionen

Wiederholung Prädikatenlogik: Syntax

Ziel: Modellierung von Aussagen über Eigenschaften und Beziehungen von Objekten eines bestimmten Bereiches

Signatur $\Sigma = (\Sigma_F, \Sigma_R)$ Funktions- und Relationssymbole

(Individuen-)Variablen $\mathbb{X}$

Terme $\mathrm{Term}(\Sigma_F, \mathbb{X})$, induktive Definition:

- **IA:** $\mathbb{X} \subseteq \mathrm{Term}(\Sigma_F, \mathbb{X})$
- **IS:** Aus $(f, n) \in \Sigma_F$ und $t_1, \ldots, t_n \in \mathrm{Term}(\Sigma_F, \mathbb{X})$ folgt $f(t_1, \ldots, t_n) \in \mathrm{Term}(\Sigma_F, \mathbb{X})$.

Atome $\mathrm{Atom}(\Sigma, \mathbb{X})$:

Aus $(p, n) \in \Sigma_R$ und $t_1, \ldots, t_n \in \mathrm{Term}(\Sigma_F, \mathbb{X})$ folgt $p(t_1, \ldots, t_n) \in \mathrm{Atom}(\Sigma_F, \mathbb{X})$

Formeln $\mathrm{FOL}(\Sigma, \mathbb{X})$ induktive Definition:

- **IA:** $\mathrm{Atom}(\Sigma, \mathbb{X}) \subseteq \mathrm{FOL}(\Sigma, \mathbb{X})$
- **IS:** Falls j ein n-stelliger Junktor ist, $x \in \mathbb{X}$ und $\varphi_1, \ldots, \varphi_n \in \mathrm{FOL}(\Sigma, \mathbb{X})$, dann gilt $j(\varphi_1, \ldots, \varphi_n) \in \mathrm{FOL}(\Sigma, \mathbb{X})$, $\forall x \varphi \in \mathrm{FOL}(\Sigma, \mathbb{X})$ und $\exists x \varphi \in \mathrm{FOL}(\Sigma, \mathbb{X})$,

Wiederholung Prädikatenlogik: Semantik

Σ-Struktur $\mathcal{A} = (A, [\![\cdot]\!]_{\mathcal{A}})$ mit

- nichtleerer Menge A (Trägermenge)
- Interpretation $[\![\cdot]\!]_{\mathcal{A}}$ der Funktions- und Relationssysmbole aus Σ
  - für jedes $(f, n) \in \Sigma_F$ eine Funktion $[\![f]\!]_{\mathcal{A}} : A^n \to A$
  - für jedes $(p, n) \in \Sigma_R$ eine Relation $[\![p]\!]_{\mathcal{A}} \subseteq A^n$

Belegung $\beta : X \to A$ der Individuenvariablen

Eine Interpretation $(\mathcal{A}, \beta)$ für Term $t \in \mathrm{Term}(\Sigma_F, X)$ oder Formel $\varphi \in \mathrm{FOL}(\Sigma, X)$

- einer Σ-Struktur $\mathcal{A} = (A, [\![\cdot]\!]_{\mathcal{A}})$ und
- einer Variablenbelegung $\beta : X \to A$.

Menge aller Modelle der Formel $\varphi \in \mathrm{FOL}(\Sigma, X)$

$$\mathrm{Mod}(\varphi) = \left\{ (\mathcal{S}, \beta) \;\middle|\; \begin{array}{c} (\mathcal{S}, \beta) \text{ ist } \Sigma\text{-Interpretation und} \\ [\![\varphi]\!]_{(\mathcal{S},\beta)} = 1 \end{array} \right\}$$

(Eingeschränkte) Übersetzung in Aussagenlogik

Grundinstanziierung $G : \mathrm{FOL}(\Sigma, \mathbb{X}) \to 2^{\mathrm{AL}(P)}$ in einer Σ-Struktur $\mathcal{A} = (A, [\![\cdot]\!]_{\mathcal{A}})$ (nur die endliche Trägermenge A relevant):
Grundinstanziierung der Formel $\varphi \in \mathrm{FOL}(\Sigma, \mathbb{X})$:

1. Signaturerweiterung $\Sigma' = \Sigma \cup \{(d, 0) \mid d \in A\}$:
   neues Konstantensymbol $(d, 0) \in \Sigma_F$ für jedes $d \in A$
2. Ersetzung aller Quantoren und gebundenen Variablen:
   $\forall x \varphi \mapsto \bigwedge_{d \in A} \varphi\{x \mapsto d\}$, $\exists x \varphi \mapsto \bigvee_{d \in A} \varphi\{x \mapsto d\}$,
3. Ersetzung $\beta : \mathrm{FOL}(\Sigma, \mathbb{X}) \to 2^{\mathrm{FOL}(\Sigma', \emptyset)}$ aller freien Variablen:
   Formelmenge $\beta(\varphi) =$
   $\{\varphi\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\} \mid \mathrm{fvar}(\varphi) = \{x_1, \ldots, x_n\}, a_1, \ldots, a_n \in A\}$
4. Ersetzung (Umbenennung) jedes Grundatoms $a \in \mathrm{Atom}(\beta(\varphi))$
   durch eine Aussagenvariable $p \in P$
   (z.B. $p(a_1, \ldots, a_n) \mapsto p a_1 \cdots a_n$)

Beispiele (Tafel): Grundinstanzen von

- $p(x, x) \wedge \exists x\ p(y, x)$ in $A = \{1, 2\}$,
- $\forall x(P(x) \to \exists y N(y, x))$ in $A' = \{$ Anton, Berta, Conny $\}$

Grundinstanziierung zur Wissensverarbeitung

1. Formulierung der Wissensbasis als Formelmenge
2. Formulierung der Frage als Formel
   (Behauptung, evtl. mit Variablen)
3. Grundinstanziierung (sofern möglich)
4. Transformation in z.B. Klauselmenge
   (Konjunktion von Regeln, CNF)
5. Lösung durch aussagenlogisches Verfahren

Vorteile:
- Entscheidbarkeit,
- aussagenlogische Methoden anwendbar,
- Standard-Werkzeuge einsetzbar,
  z.B. SAT-Solver

Nachteile:
- ergibt evtl. große unübersichtliche Formelmengen,
- nur möglich, falls beide folgende Bedingungen erfüllt sind:
  1. Struktur hat endliche Trägermenge
  2. Signatur enthält keine > 0-stelligen Funktionen
     (nur Konstanten)

Idee: benötigte Instanzen nach Bedarf erzeugen

Was bisher geschah

Grundlagen der (Logik-basierten) künstlichen Intelligenz

- klassische Aussagenlogik
- klassische Prädikatenlogik

- jeweils WH Syntax und Semantik
- Grundinstanziierung: (mitunter mögliche) Übersetzung von Prädikaten- in Aussagenlogik

Wissensrepräsentation und -verarbeitung in Logiken:

- Entscheidungbäume, -diagramme
- Entscheidungtabellen
- Mengen von Regeln ohne / mit Alternative
  Schließen durch Vorwärts- oder Rückwärtsverkettung

WH: Sätze in FOL

- gebundene (durch Quantoren) und freie Variablenvorkommen in Formel $\varphi \in \mathrm{FOL}(\Sigma, X)$
- $\mathrm{bvar}(\varphi)$ Menge der in $\varphi \in \mathrm{FOL}(\Sigma, X)$ gebunden vorkommenden Variablen
- $\mathrm{fvar}(\varphi)$ Menge der in $\varphi \in \mathrm{FOL}(\Sigma, X)$ frei vorkommenden Variablen

Formel $\varphi \in \mathrm{FOL}(\Sigma, X)$ mit $\mathrm{fvar}(\varphi) = \emptyset$ heißt Satz.

Beispiele:

- $\forall x \forall y (R(x, y) \to R(y, x))$ ist ein Satz.
- $R(x, y) \to R(y, x), \forall x (R(x, y) \to R(y, x)),$ $\exists y (R(x, y) \to R(y, x)), \forall x (R(x, y) \to \forall y R(y, x))$ sind keine Sätze.

für $\varphi \in \mathrm{FOL}(\Sigma, X)$ mit $\mathrm{fvar}(\varphi) = \{x_1, \ldots, x_n\}$:

universeller Abschluss $\forall \varphi := \forall x_1 \cdots \forall x_n \varphi$

existentieller Abschluss $\exists \varphi := \exists x_1 \cdots \exists x_n \varphi$

Fakt

Für jede Formel $\varphi \in \mathrm{FOL}(\Sigma, X)$ sind $\forall \varphi$ und $\exists \varphi$ Sätze.

WH: Wichtige Äquivalenzen mit Quantoren

$$
\begin{aligned}
\neg\forall x\varphi &\equiv \exists x\neg\varphi \\
\neg\exists x\varphi &\equiv \forall x\neg\varphi \\
\forall x\varphi \wedge \forall x\psi &\equiv \forall x(\varphi \wedge \psi) \\
\exists x\varphi \vee \exists x\psi &\equiv \exists x(\varphi \vee \psi) \\
\forall x\forall y\varphi &= \forall y\forall x\varphi \\
\exists x\exists y\varphi &= \exists y\exists x\varphi
\end{aligned}
$$

für $x \notin \mathrm{fvar}(\psi)$ gilt außerdem

$$
\begin{aligned}
\forall x\varphi \vee \psi &\equiv \forall x(\varphi \vee \psi) \\
\exists x\varphi \vee \psi &\equiv \exists x(\varphi \vee \psi) \\
\forall x\varphi \wedge \psi &\equiv \forall x(\varphi \wedge \psi) \\
\exists x\varphi \wedge \psi &\equiv \exists x(\varphi \wedge \psi)
\end{aligned}
$$

Prädikatenlogische Normalformen

Eine Formel $\varphi \in \mathrm{FOL}(\Sigma, X)$ heißt in

bereinigter Form , wenn $\mathrm{bvar}(\varphi) \cap \mathrm{fvar}(\varphi) = \emptyset$ und jeder Quantor eine andere Variable bindet
z.B. $\forall x(p(x) \to \exists y(r(y,z) \vee r(z,y)))$

Pränexform , wenn $\varphi = Q_1 x_1 \cdots Q_n x_n \psi$, wobei $\forall i : Q_i \in \{\forall, \exists\}$ und in ψ keine Quantoren vorkommen.
z.B. $\forall x \exists y(p(x) \to (r(y,z) \vee r(z,y)))$

Skolemform , wenn $\varphi = \forall x_1 \cdots \forall x_m \psi$, wobei $\mathrm{fvar}(\varphi) = \emptyset$ (Satz), alle x_i verschiedene Variablen sind und in ψ keine Quantoren vorkommen.
z.B. $\forall x \forall z(p(x) \to (r(a,z) \vee r(z,a)))$

Klauselform , wenn $\varphi = \forall x_1 \cdots \forall x_m \psi$, wobei alle x_i verschiedene Variablen und ψ eine CNF sind.
z.B. $\forall x \forall z((\neg p(x) \vee r(a,z)) \wedge (\neg p(x) \vee r(z,a)))$
Darstellung als Regelmenge möglich:
$\{p(x) \to r(a,z), p(x) \to r(z,a)\}$

Erfüllbarkeits-Äquivalenz

Definition

Zwei Formeln φ, ψ heißen Erfüllbarkeits-äquivalent, wenn gilt:
φ ist genau dann erfüllbar, wenn ψ erfüllbar ist.

($\mathrm{Mod}(\varphi) \neq \emptyset$ gdw. $\mathrm{Mod}(\psi) \neq \emptyset$)

Achtung:

- Erfüllbarkeits-Äquivalenz ist schwächer als semantische Äquivalenz.

  Beispiel: $p \vee q$ und r sind erfüllbarkeitsäquivalent,
  $p \vee q \equiv r$ gilt aber nicht. (Warum?)

- Erfüllbarkeits-Äquivalenz bleibt beim Einsetzen in Formeln i.A. nicht erhalten.

  Beispiel: p und q sind Erfüllbarkeits-äquivalent,
  aber $p \wedge \neg p$ und $p \wedge \neg q$ nicht.

Konstruktion prädikatenlogischer NF

gebundene Umbenennung $\theta : X \to X$ einer Formel $\varphi \in \mathrm{FOL}(\Sigma, X)$
ersetzt jedes gebundene Variablen-Vorkommen (auch beim bindenden Quantor) durch eine Variable, die in φ noch nicht vorkommt.

Zu jeder Formel $\varphi \in \mathrm{FOL}(\Sigma, X)$ lassen sich konstruieren:

- eine äquivalente Formel in bereinigter Form,
  (Konstruktion durch gebundene Umbenennungen)
- eine äquivalente Formel in Pränexform,
  (Konstruktion durch äquivalente Umformungen)
- eine Erfüllbarkeits-äquivalente Formel (Satz) in Skolemform,
  aus deren Modellen sich Modelle für φ konstruieren lassen
  (existenzieller Abschluss und schrittweise Eliminierung der $\exists$ von links nach rechts durch Einführung neuer Funktionssymbole)
- eine Erfüllbarkeits-äquivalente Formel (Satz) in Klauselform.

Beispiele:

- $(\neg\exists x(P(x,z) \vee \forall y Q(x, f(y))) \vee \forall y P(g(z,y), z))$
- $\exists x \forall y \exists z \forall u \exists v P(x, y, z, u, v)$

Syntaktisches Schließen in der Prädikatenlogik

Modellmengen prädikatenlogischer Formeln (und Formelmengen) sind i.A. unendlich.

Aussagen wie
In jeder Äquivalenzrelation gilt φ, d.h.

$$\left\{ \begin{array}{l} \forall x\ R(x,x) \\ \forall x \forall y\ (R(x,y) \rightarrow R(y,x)) \\ \forall x \forall y \forall z\ (R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \end{array} \right\} \models \varphi$$

lassen sich i.A. nicht durch Nachprüfen aller Interpretationen beweisen.

Syntaktisches Schließen in Kalkülen ist daher in der Prädikatenlogik noch wichtiger als in der Aussagenlogik.

WH: Modellierungsbeispiel Prädikatenlogik

Wissensbasis:

F1	Tom ist ein Baby.	$b(T)$
F2	Tom ist männlich.	$m(T)$
F3	Anna ist weiblich.	$m(A)$
R1	Babies sind Kinder.	$\forall x : b(x) \rightarrow k(x)$
R2	Männliche Kinder sind Jungen.	$\forall x : (m(x) \wedge k(x)) \rightarrow j(x)$
R3	Weibliche Kinder sind Mädchen.	$\forall x : (w(x) \wedge k(x)) \rightarrow g(x)$

Fragen:

- Q1 Ist Tom ein Junge?
- Q2 Ist Anna ein Kind?

- schon bekannt:
  Lösung durch Vorwärtsverkettung (datenorientiert)
- jetzt:
  Lösung durch Rückwärtsverkettung (zielorientiert)

Lösungsidee Resolution

1. Formulierung der Wissensbasis als Formelmenge (Regelmenge, logisches Programm)
2. Formulierung der Frage als Formel (Behauptung, evtl. mit Variablen)
3. Transformation in Klauselmenge
4. Grundinstanziierung (sofern möglich)
5. Lösung durch aussagenlogische Resolution

Nachteil: Grundinstanziierung aufwendig

besser: benötigte Instanzen nach Bedarf erzeugen

Substitutionen (WH aus FOP)

Substitution: partielle Funktion $\theta : X \to \text{Term}(\Sigma, X)$
Notation $\{x \mapsto t_1, y \mapsto t_2, \ldots\}$
Anwendung einer Substitution:

- $s\{x \mapsto t\}$ ist der Term, welcher aus dem Term s durch Ersetzung jedes Vorkommens der Variable x durch t entsteht
- $\varphi\{x \mapsto t\}$ ist die Formel, die aus der Formel φ durch Ersetzung jedes freien Vorkommens der Variable x durch t entsteht

Beispiele:

- $g(x, f(a))\{x \mapsto b\} = g(b, f(a))$
- $P(y, x, f(g(y, a)))\{x \mapsto g(a, z), y \mapsto a\} = P(a, g(a, z), f(g(a, a)))$
- $g(x, f(a))\{x \mapsto b, y \mapsto a\} = g(b, f(a))$
- $g(b, f(y))\{x \mapsto b, y \mapsto a\} = g(b, f(a))$
- $(P(b, f(y)) \to Q(x))\{x \mapsto b, y \mapsto f(a)\} = (P(b, f(f(a))) \to Q(b))$
- für $\theta = \{x \mapsto b\}, \sigma = \{y \mapsto f(a)\}$ (auch $\theta(x) = b, \sigma(y) = f(a)$) gilt
  $(P(b, f(y)) \to Q(x))\theta\sigma = \sigma(\theta(P(b, f(y)) \to Q(x))) =$
  $= P(b, f(f(a))) \to Q(b)$

Unifikator

Substitution θ heißt genau dann Unifikator der Terme (Atome, Literale) t_1 und t_2 (θ unifiziert t_1 und t_2), wenn $\theta(t_1) = \theta(t_2)$ gilt.
Beispiele:

1. $\theta = \{x \mapsto b, y \mapsto a\}$ unifiziert $t_1 = g(x, f(a))$ und $t_2 = g(b, f(y))$
2. $\{x \mapsto g(g(y)), z \mapsto g(y)\}$ unifiziert $f(x, g(y))$ und $f(g(z), z)$ (und $f(g(z), g(y))$.
3. $\{x \mapsto g(g(a)), y \mapsto a, z \mapsto g(a)\}$
   unifiziert $f(x, g(y))$ und $f(g(z), z)$.
4. $\{x \mapsto g(g(y)), z \mapsto g(y), v \mapsto f(a)\}$
   unifiziert $f(x, g(y))$ und $f(g(z), z)$.

Terme (Atome, Literale) t_1, t_2 heißen genau dann unifizierbar, wenn ein Unifikator für t_1 und t_2 existiert.
Beispiele:

1. $g(x, f(a))$ und $g(b, f(y))$ sind unifizierbar,
   $f(g(a, x))$ und $f(g(f(x), a))$ nicht. Warum?
2. $\neg P(a, f(x), g(a, y))$ und $\neg P(x, f(y), z)$ sind unifizierbar,
   $R(f(a), x)$ und $R(x, a)$ nicht.

Ordung auf Unifikatoren

Für zwei Unifikatoren σ, θ der Terme t_1, t_2 gilt:

σ heißt genau dann allgemeiner als θ, wenn eine Substitution ρ (die nicht nur Umbenennung ist) existiert, so dass
$t_1\theta = t_1\sigma\rho = t_2\sigma\rho = t_2\theta$

(analog Teilerrelation)

Beispiele: Unifikatoren für $f(x, g(y)), f(g(z), z)$

1. Unifikator $\{x \mapsto g(g(y)), z \mapsto g(y)\}$ ist allgemeiner als $\{x \mapsto g(g(a)), z \mapsto g(a)\}$
   $\rho = \{y \mapsto a\}$
2. Unifikator $\{x \mapsto g(g(y)), z \mapsto g(y)\}$ ist allgemeiner als $\{x \mapsto g(g(y)), z \mapsto g(y), v \mapsto g(b)\}$
   $\rho = \{v \mapsto g(b)\}$

Allgemeinster Unifikator

Zu unifizierbaren Termen (Atomen, Literalen) s, t existiert (bis auf Umbenennung der Variablen) genau ein Unifikator θ mit der folgenden Eigenschaft:
Für jeden Unifikator σ für s, t ist θ allgemeiner als σ.
Dieser heißt allgemeinster Unifikator $\theta = \text{mgu}(s, t)$ von s und t.

(analog ggT)

Beispiele:

- $\text{mgu}(f(x, a), f(g(b), y)) = \{x \mapsto g(b), y \mapsto a\}$
- $\text{mgu}(f(x, g(y)), f(g(z), z)) = \{x \mapsto g(g(y)), z \mapsto g(y)\}$

Unifizierbarkeit

- t ist mit t unifizierbar (mit Unifikator $\theta = \emptyset$)
- t ist mit $x \in \mathbb{X}$ unifizierbar (Unifikator $\theta = \{x \mapsto t\}$) gdw. x in t nicht vorkommt
- $f(t_1, \ldots, t_n), g(s_1, \ldots, s_m)$ sind nicht unifizierbar, falls $n \neq m$ oder $f \neq g$
- θ ist Unifikator für $f(t_1, \ldots, t_n), f(s_1, \ldots, s_n)$ gdw. $\forall i \in \{1, \ldots, n\} : \theta$ unifiziert t_i und s_i

Unifikationsalgorithmus

zur Bestimmung des $\mathrm{mgu}(s,t)$ für $s,t \in \mathrm{Term}(\Sigma_F, \mathbb{X})$ oder $s,t \in \mathrm{Atom}(\Sigma, \mathbb{X})$ (sofern dieser existiert)

Algorithmus: Unifikationsalgorithmus

Eingabe: $s,t \in \mathrm{Term}(\Sigma_F, \mathbb{X})$ (oder $\in \mathrm{Atom}(\Sigma, \mathbb{X})$)
Ausgabe: $\theta = \mathrm{mgu}(s,t)$ oder „nicht unifizierbar“

wenn $s = t$ **dann**
| $\theta \leftarrow \emptyset$
sonst wenn $s \in \mathbb{X}$ **dann**
| **wenn** $s \in \mathrm{var}(t)$ **dann** „nicht unifizierbar“
| **sonst** $\theta \leftarrow \{s \mapsto t\}$
sonst wenn $t \in \mathbb{X}$ **dann**
| $\theta \leftarrow \mathrm{mgu}(t,s)$
sonst wenn $s = f(s_1,\ldots,s_m), t = g(t_1,\ldots,t_n)$ **dann**
| **wenn** $f = g$ *(und* $m = n$*)* **dann**
| $\theta \leftarrow \mathrm{mgu}(s_1,t_1) \circ \cdots \circ \mathrm{mgu}(s_m,t_m)$
| **sonst** „nicht unifizierbar“

Dabei gilt für jede Substitution θ:

$$\theta \circ \text{„nicht unifizierbar“} = \text{„nicht unifizierbar“} \circ \theta = \text{„nicht unifizierbar“}$$

Unifikationsalgorithmus – Beispiele

- $\mathrm{mgu}(f(x, h(y), y), f(g(z), z, a)) =$
  $\{x \mapsto g(h(a)), z \mapsto h(a), y \mapsto a\}$
- $\mathrm{mgu}\,(P(f(x), g(y, h(a, z))), P(f(g(a, b)), g(g(u, v), w))) =$
  $\{x \mapsto g(a, b), y \mapsto g(u, v), w \mapsto h(a, z)\}$
- $\mathrm{mgu}(P(f(a), g(x)), P(y, y))$ existiert nicht
- $\mathrm{mgu}(f(x, g(a, z)), f(f(y), f(x))$ existiert nicht
- $\mathrm{mgu}(f(x, x), f(y, g(y))$ existiert nicht
- $\mathrm{mgu}(f(x, g(y)), f(y, x)$ existiert nicht

Was bisher geschah

Wissensrepräsentation und -verarbeitung in Logiken

klassische Aussagenlogik:

- Syntax (Wiederholung)
- Semantik (Wiederholung)
- Resolution

klassische Prädikatenlogik:

- Syntax (Wiederholung)
- Semantik (Wiederholung)
- Normalformen:
  - bereinigt
  - Pränex
  - Skolem ($\exists$-Eliminierung)
  - Klausel (Menge von Klauseln, Notation ohne Quantoren)
- Substitutionen,Unifikatoren
- allgemeinster Unifikator
- Unifikationsalgorithmus

Prädikatenlogik: Modellierungsbeispiel

Wissen
- Paul liest Kriminalromane.
- Tina liest Arztromane.
- Bob mag sich selbst.
- Tina mag alle Krimileser.
- Krimileser mögen Zeitungsleser.
- Alle lesen Zeitungen.

Darstellung als prädikatenlogische Formelmenge

Fragen
- Mag Bob Tina?
- Wen mag Bob?
- Wer mag Bob?
- Wer mag Zeitungsleser?
- Wer mag wen?
- Wer liest was?

Typ der Lösung ja / nein oder Substitution (und evtl. Begründung)ls ls ..

Lösungen ...

Lösungsidee Rückwärtsverkettung (Wiederholung)

1. Formulierung der Wissensbasis als Formelmenge (Regelmenge, logisches Programm)
2. Formulierung der Frage als Formel (Behauptung, evtl. mit Variablen)
3. Transformation in Klauselmenge
4. Grundinstanziierung (sofern möglich)
5. Lösung durch aussagenlogische Resolution

Nachteil:
Grundinstanziierung ergibt i.A. sehr große (evtl. unendliche) Formelmengen

Idee: nur Erzeugung der zur Lösung notwendigen Instanzen

Prädikatenlogische Resolution

Wiederholung: Resolutionsregel (Aussagenlogik)

$$\frac{l_1 \vee \cdots \vee l_n \vee l \qquad l'_1 \vee \cdots \vee l'_m \vee \neg l}{l_1 \vee \cdots \vee l_n \vee l'_1 \vee \cdots \vee l'_m}$$

Ziel: Erweiterung auf Prädikatenlogik
Beispiel: Für jeden Papagei gilt
Sein Vater hat eine rote Feder und er selbst trägt keine rote Feder.
$\varphi = \forall x(R(v(x)) \wedge \neg R(x))$ ist unerfüllbar.
Interpretiert in einer Struktur $\mathcal{A} = (A, [\![\cdot]\!]_{\mathcal{A}})$, repräsentiert φ die Formelmenge (Grundinstanziierung)

$$\bigcup_{a \in A} \{R(v(a)) \wedge \neg R(a), R(v(v(a))) \wedge \neg R(v(a)), \ldots\}$$

φ repräsentiert die (grundinstanziierte) Klauselmenge

$$\bigcup_{a \in A} \{\underline{R(v(a))}, \neg R(a), R(v(v(a))), \underline{\neg R(v(a))}, \ldots\}$$

unerfüllbar nach (aussagenlogischer) Resolution
Ziel: Resolution geeigneter Instanzen der zu resolvierenden Klauseln.

Prädikatenlogische Resolution

(Einfache) Resolutionsregel für Klauselform (implizit $\forall$-abgeschlossen): falls $\theta = \mathrm{mgu}(l, l')$ existiert:

$$\frac{l_1 \vee \cdots \vee l_n \vee l \qquad l'_1 \vee \cdots \vee l'_m \vee \neg l'}{(l_1 \vee \cdots \vee l_n \vee l'_1 \vee \cdots \vee l'_m)\theta}$$

Beispiel:

$$\{P(f(x)) \vee \underline{R(g(x), x)}, \underline{\neg R(u, v)} \vee \neg S(u, v)\} \models P(f(x)) \vee \neg S(g(x), x)$$

mit Unifikator $\theta = [u \mapsto g(x), v \mapsto x]$

Volle Resolutionsregel erlaubt gleichzeitige Eliminierung aller unifizierbaren Instanzen eines Literals.

Beispiel:

$$\{\underline{P(x, b)} \vee \underline{P(a, y)} \vee Q(x, f(y)), \underline{\neg P(z, w)} \vee \neg Q(w, z)\} \models Q(a, f(b)) \vee \neg Q(b, a)$$

mit Unifikator
$\theta(P(x, b), P(a, y), P(z, w)) = [x \mapsto a, y \mapsto b, z \mapsto a, w \mapsto b]$

Prädikatenlogische Resolution

Wiederholung:
In einer Logik L (definiert durch Syntax, Semantik)
ist der Kalkül (Schlussverfahren) $\vdash \subseteq 2^L \times L$
gegenüber der Folgerungsrelation $\models \subseteq 2^L \times L$

korrekt gdw. für jede Formelmenge $\Phi \subseteq L$ und jede Formel $\psi \in L$:
aus $\Phi \vdash \psi$ folgt $\Phi \models \psi$

vollständig gdw. für jede Formelmenge $\Phi \subseteq L$ und jede Formel $\psi \in L$:
aus $\Phi \models \psi$ folgt $\Phi \vdash \psi$

(aussagen- und prädikatenlogischer) Resolutionskalkül:

$\Phi \vdash_R \psi$ gdw. $\mathbb{f}$ durch Resolution aus $\Phi \cup \{\neg\psi\}$ ableitbar

Satz

Der Resolutionskalkül ist für AL(P) *und* FOL($\Sigma, \mathbb{X}$) *(bzgl.* $\models$*) korrekt und vollständig.*

Prädikatenlogische Resolution – Beispiele

- Gilt

$$\left\{ \begin{array}{l} E(a,b), E(a,c), E(b,c), E(c,d), \\ E(x,y) \rightarrow P(x,y), \\ (E(u,v) \wedge P(v,w)) \rightarrow P(u,w) \end{array} \right\} \models P(a,d)?$$

- $\{\neg b(x) \vee r(y,y)) \vee r(x,y), \neg b(z) \vee \neg r(u,u)) \vee \neg r(z,u), b(a)\}$ ist unerfüllbar

Beispiel: Barbier (Russellsches Paradox)

informal: Der Barbier im Dorf rasiert (genau) diejenigen, die sich nicht selbt rasieren.

Problem: Frage: Rasiert der Barbier (b) sich selbst? 2 Fälle:

1. Ann.: b rasiert sich selbst $\rightarrow$ b rasiert sich nicht
2. Ann.: b rasiert sich nicht selbst $\rightarrow$ b rasiert sich

Widerspruch in beiden Fällen

Folgerung: Also existiert kein solcher Barbier.

(analog existiert keine Menge aller Mengen)

formal: Kontext: $\forall x(r(b,x) \leftrightarrow \neg r(x,x))$

in Klauselform: $\Phi =$

$\{\neg r(b,x) \vee \neg r(x,x), r(x,x) \vee r(b,x)\}$

Resolution: 2 Fälle: Frage (als Behauptung)

$\psi \in \{r(b,b), \neg r(b,b)\}$

1. $\{\neg r(b,x) \vee \neg r(x,x), r(x,x) \vee r(b,x), \neg r(b,b)\}$
2. $\{\neg r(b,x) \vee \neg r(x,x), r(x,x) \vee r(b,x), r(b,b)\}$

in beiden Fällen Resolution zu $\mathbb{f}$ mit $\sigma = \{x \mapsto b\}$

Horn-Logik (definite Regeln)

Fragment der klassischen Prädikatenlogik (der ersten Stufe):

Horn-Klausel (endliche) Disjunktion von Literalen, die höchstens ein positives Literal enthalten.

Horn-Formel (endliche) Konjunktion von Horn-Klauseln

(definites) logisches Programm (endliche) Menge von Hornklauseln mit je einem positiven Literal

Darstellung von Horn-Klauseln:

- Hornklausel mit einem positiven Literal:
  $\neg b_1 \vee \ldots \vee \neg b_n \vee h \equiv (b_1 \wedge \ldots \wedge b_n) \rightarrow h$
  Regeln
  Spezialfall Fakt h (Regel ohne Rumpf)
- Hornklausel ohne positive Literale:
  $\neg b_1 \vee \ldots \vee \neg b_n \equiv \neg b_1 \vee \ldots \vee \neg b_n \vee \mathbb{f} \equiv (b_1 \wedge \ldots \wedge b_n) \rightarrow \mathbb{f}$
  Anfrage (Query, Zielklausel, herzuleitende Formel)

Achtung:
Nicht zu jeder Formel existiert eine äquivalente Formel in Horn-Logik (z.B. $p \vee q$, Formalisierung Barbier).

Horn-Logik ist also ein echtes Fragment der klassischen Prädikatenlogik

Prolog: Syntax

Signatur $\Sigma = (\Sigma_F, \Sigma_R)$

Syntax (ASCII) analog Datalog:

Prolog-Term: $t \in \text{Term}(\Sigma_F, \mathbb{X})$ (mehr als Datalog)

Prolog-Atom: $a = p(t_1, \ldots, t_n) \in \text{Atom}(\Sigma, \mathbb{X})$
mit n-stelligem Relationssymbol $p \in \Sigma_R$
Prolog-Syntax: `p( t1, ... , tn )`

Prolog-Klausel: Regel `h :- b1 , ... , bn.`
mit Prolog-Atomen `b1 , ... , bn , h`

Prolog-Fakt: Regel ohne Rumpf, d.h. Prolog-Klausel mit $n = 0$

Prolog-Programm (Wissensbasis):
endliche Menge von Prolog-Klauseln

Prolog-Anfrage (Zielklausel, Query): Regel ohne Kopf, d.h.
Formel `?- b1 , ... , bn.`
mit Prolog-Atomen `b1 , ... , bn`

Prolog: Semantik

Wissensbasis: Prolog-Programm (Programmklauseln)

$$P = \{b_{i1} \wedge \cdots \wedge b_{in_i} \rightarrow h_i \mid i \in \{1, \ldots n\}\}$$

Zielklausel: $b_1 \wedge \cdots \wedge b_m$

Antwort: nein

ja und evtl. Variablenbelegung β

Was ist die Bedeutung der drei Komponenten Wissensbasis, Zielklausel und Antwort? Wie hängen sie zusammen?

deklarative Semantik: logische Bedeutung der Komponenten und der Antwort

operationale Semantik: Verfahren zum maschinellen Finden der Antwort (durch den Prolog-Interpreter) aus den Komponenten

Prolog – deklarative Semantik

1. Programm $P = \{$ `h_i :- b_i1 , ... , b_ini.` $\mid i \in \{1, \ldots m\}\}$
   repräsentiert die Menge $\Phi \subseteq \mathrm{FOL}(\Sigma, \mathbb{X})$ von Sätzen

   $$\Phi = \{\forall\,(b_{i1} \wedge \cdots \wedge b_{in_i} \rightarrow h_i) \mid i \in \{1, \ldots m\}\}$$

   ($\forall$-Abschlüsse aller Regeln)
   Σ enthält genau alle in P vorkommenen Relations- und Funktionssymbole (Konstruktoren)
   (und vordefinierte Symbole, z.B. $+$, $=$, `is` , ...)

2. Zielklausel $\psi = (b_1 \wedge \cdots \wedge b_m) \in \mathrm{FOL}(\Sigma, \mathbb{X})$

3. Herbrand-Modell von Φ:
   Modell für Φ über der Trägermenge $\mathrm{Term}(\Sigma_F, \emptyset)$ (Grundterme)

4. Mod_H: Menge aller Herbrand-Modelle von Φ

5. Antwort:

   ja mit Substitution $\theta : \mathrm{fvar}(\psi) \rightarrow \mathrm{Term}(\Sigma_F, \emptyset)$, falls
   $\mathrm{Mod}_H(\Phi) \subseteq \mathrm{Mod}_H(\psi\theta)$ $(\mathrm{Mod}_H(\Phi \cup \{\neg\psi\theta\}) = \emptyset)$

   nein , falls für jede Substitution θ gilt
   $\mathrm{Mod}_H(\Phi) \not\subseteq \mathrm{Mod}_H(\psi\theta)$ $(\mathrm{Mod}_H(\Phi \cup \{\neg\psi\theta) \neq \emptyset)$

Prolog – deklarative Semantik – Beispiel in AL

Programm `a :- b,c. a :- d. b. b :- d. c. d :- e. d.`

interpretiert als $\Phi = \{b \wedge c \to a, d \to a, b, d \to b, c, e \to d, d\}$

(rein aussagenlogisch, $\Sigma_F = \emptyset$, $\mathrm{Term}(\Sigma_F, \emptyset) = \emptyset$, $\mathrm{Mod}_H = \mathrm{Mod}$)

$\mathrm{Mod}_H(\Phi) = \{W_{11110}, W_{11111}\} = \{\underline{\{a, b, c, d\}}, \{a, b, c, d, e\}\}$

Anfrage 1: `?- a.` , $\psi = a$,
$\mathrm{Mod}_H(\Phi \cup \{\neg a\}) = \emptyset$,
also Antwort: ja (ohne Substitution)

Anfrage 2: `?- e.` , $\psi' = e$,
$\mathrm{Mod}_H(\Phi \cup \{\neg e\}) = \{W_{11110}\} \neq \emptyset$,
also Antwort: nein

Prolog – deklarative Semantik – Beispiel in FOL

Programm `p(a). q(X) :- p(X). q(b).`

interpretiert als $\Phi = \{p(a), \forall x(p(x) \rightarrow q(x)), q(b)\}$

Herbrand-Modelle von Φ mit Trägermenge $\{a, b\}$ (Konstanten):
$\mathrm{Mod}_H(\Phi) = \{\mathcal{A}, \mathcal{B}\}$ mit $\mathcal{A} = (\{a, b\}, [\![\cdot]\!]_\mathcal{A})$, $\mathcal{B} = (\{a, b\}, [\![\cdot]\!]_\mathcal{B})$,
wobei $[\![p]\!]_\mathcal{A} = \{a\}$, $[\![p]\!]_\mathcal{B} = \{a, b\}$ und $[\![q]\!]_\mathcal{A} = [\![q]\!]_\mathcal{B} = \{a, b\}$
alternativ notiert als Mengen von Grundatomen:
$\mathrm{Mod}_H(\Phi) = \{\underbrace{\{p(a), q(a), q(b)\}}_{\mathcal{A}}, \underbrace{\{p(a), p(b), q(a), q(b)\}}_{\mathcal{B}}\}$

Anfrage 1: `?- q(a).` , $\psi = q(a)$ $\mathrm{Mod}_H(\Phi \cup \{\neg q(a)\}) = \emptyset$,
also Antwort: ja (ohne Substitution)

Anfrage 2: `?- q(X).` , $\psi = q(x)$ $\mathrm{Mod}_H(\Phi \cup \{\neg q(x)\}) = \emptyset$,
also Antwort: ja, mit Substitutionen $\alpha = \{x \mapsto a\}$ und $\beta = \{x \mapsto b\}$

Anfrage 3: `?- p(a).` , $\psi = p(a)$ $\mathrm{Mod}_H(\Phi \cup \{\neg p(a)\}) = \emptyset$,
also Antwort: ja (ohne Substitution)

Anfrage 4: `?- p(b).` , $\psi = p(b)$ $\mathrm{Mod}_H(\Phi \cup \{\neg p(b)\}) = \{\mathcal{A}\}$,
also Antwort: nein

Anfrage 5: `?- p(X).` , $\psi = p(x)$ $\mathrm{Mod}_H(\Phi \cup \{\neg p(x)\}) = \emptyset$,
also Antwort: ja, mit Substitution $\beta = \{x \mapsto a\}$

Programmierung in Prolog – Beispiel

Wissensbasis: logisches Programm P (Prolog, Datalog)

```
liest(paul,krimi). liest(tina,arztroman).
mag(bob,bob).
mag(tina,X) :- liest(X,krimi).
mag(X,Y) :- liest(X,krimi), liest(Y,zeitung).
liest(X,zeitung).
```

repräsentiert die Formelmenge

$$\Phi = \left\{ \begin{array}{l} l(p,k), l(t,a), m(b,b), \forall x\,(l(x,k) \rightarrow m(t,x))\,, \\ \forall x\ \forall y\ \ ((l(x,k) \wedge l(y,z)) \rightarrow m(x,y))\,, \forall x\ l(x,z) \end{array} \right\}$$

Aufgabenbeschreibung: Zielklausel (Anfrage, Query)

```
?- mag (bob,tina).
```

repräsentiert die Formel $\psi = m(b,t)$

Paar (Programm, Zielklausel) repräsentiert die Frage:

(Für welche Belegungen) Gilt $\Phi \models \psi$?

Lösung ja und Belegung β mit $\Phi \models \psi$

oder nein

Lösungsverfahren Resolution auf $\Phi \cup \{\neg\psi\}$

Logische Programmierung – Antworten

Variablen in Zielklauseln:

```
?- mag (tina,X).
```

repräsentiert die Formel $\psi = \exists x : m(t, x)$
Für welche x gilt $\Phi \models \psi$?
Antwort: erfüllende Belegung für x
Lösung durch Resolution auf

$$\Phi \cup \{\neg\psi\} = \Phi \cup \{\neg\exists x : m(t, x)\} = \Phi \cup \{\neg\exists y : m(t, y)\} = \Phi \cup \{\forall y : \neg m(t, y)\}$$

Beispiel als Klauselmenge:

$$\left\{ \begin{array}{l} l(p, k), l(t, a), m(b, b), \\ \neg l(x, k) \vee m(t, x), \neg l(x, k) \vee \neg l(y, z) \vee m(x, y), l(x, z), \neg m(t, y) \end{array} \right\}$$

mögliche Antworten:
während der Resolution berechnete Substitutionen für x
Was bedeutet `?- mag (X,Y).`
Welche Antworten?

Prolog – operationale Semantik

SLD-Resolution:

linear resolution with selection function for definite clauses

- Definite clauses:
  Horn-Programme (höchstens ein positives Literal je Klausel)
- Linear resolution:
  Folge von Resolutionsschritten, wobei in jedem Schritt mit einer Klausel aus dem Programm resolviert wird
- Input-Resolution
  Folge von Resolutionsschritten, wobei in jedem Schritt nach einem Literal in der im vorigen Schritt erzeugten Klausel (zu Beginn also der Zielklausel) resolviert wird
- Selection function: Auswahl
  - der Klausel im Programm (unter mehreren passenden): von oben nach unten (von links nach rechts)
  - des Literals in der Klausel: von links nach rechts

  Tiefensuche mit Backtracking

Beispiel (Tafel): Anfrage $p(y)$, $P =$
$\{\neg s(x) \vee p(x), q(a,a), q(b,a), t(b,b), \neg q(x,a) \vee \neg t(x,b) \vee p(x)\}$

Was bisher geschah

Wissensrepräsentation und -verarbeitung in Logiken
WH klassische Aussagenlogik: Syntax, Semantik, Resolution
klassische Prädikatenlogik:

- Wiederholung Syntax, Semantik
- Normalformen: bereinigt, Pränex-, Skolem-, Klausel-
- Substitutionen, Unifikatoren
- allgemeinster Unifikator (mgu(s, t)), Unifikationsalgorithmus
- Prädikatenlogisches Resolutionsverfahren

logische Programmierung in Prolog:

- Syntax: Horn-Klauseln in Regel-Form
- Wissensbasis: logisches Programm aus Fakten und (definiten) Regeln
- Anfrage: Konjunktion von Literalen
- Semantik:

  deklarativ: analog FOL, jedoch nur kleinstes Termmodell
  operational: SLD-Resolution

Datalog

Datalog: Anfragesprache für relationale Datenbanken
(Tabellen repräsentieren Relationen, definieren die Signatur)

FOL($\Sigma, \mathbb{X}$)-Fragment mit den folgenden Eigenschaften:

Syntax Σ_F enthält nur Konstantensymbole
(nullstellige Funktionssymbole)

Semantik Interpretation über einer festen Struktur (definiert durch Datenbank) mit endlicher Trägermenge
(Menge aller in der DB vorkommenden Konstanten)

extensionale Prädikate:
in Datenbank definiert
(Tabellen-, Relationenschemata)

intensionale Prädikate:
im Datalog-Programm (Regelköpfen) definiert

Einschränkung (Syntax):
Kein Prädikat darf extensional und intensional vorkommen.

Datalog – Syntax

Datalog-Term: Konstantensymbol oder Variable

Datalog-Atom: `p( t1 , ... , tn )` mit n-stelligem Relationssymbol $p \in \Sigma_R$ und Termen `t1 , ... , tn` (nur Variablen oder Konstanten)

Datalog-Klausel: Hornklausel, d.h. Klausel $\neg b_1 \vee \ldots \vee \neg b_n \vee h$ mit genau einem positiven Literal
äquivalent: $(b_1 \wedge \ldots \wedge b_n) \rightarrow h$
Rumpf $b_1 \wedge \ldots \wedge b_n$, Kopf h
Datalog-Syntax: Regel `h :- b1 , ... , bn.`
mit Datalog-Atomen `b1 , ... , bn , h`
intensionales Prädikat im Kopf `h`,
intensionale oder extensionale Prädikate in `bi`

Datalog-Fakt: Datalog-Klausel mit $n = 0$
(positives Literal, Regel ohne Rumpf)

Datalog-Wissensbasis: endliche Menge von Datalog-Klauseln

Datalog-Anfrage: Formel `?- b1, ... , bn.` mit Datalog-Atomen `b1, ... , bn`

WH: Datalog – Vorwärtsverkettung

Aus der Wissensbasis

F1 Tom ist ein Baby. `b(t).` (`b` extensional)

F2 Tom ist männlich. `m(t).` (`m` extensional)

F3 Anna ist weiblich. `w(a).` (`w` extensional)

R1 Babies sind Kinder. `k(X) :- b(X).` (`k` intensional)

R2 Männliche Kinder sind Jungen. `j(X) :- k(X),m(X).` (`j` intens.)

R3 Weibliche Kinder sind Mädchen. `g(X) :- k(X),w(X).` (`g` intens.)

folgt (ohne gezielte Anfrage): Tom ist ein Kind. Tom ist ein Junge.

Instanz `k(t) :- b(t).` der Regel `k(X) :- b(X).`
„feuert“ in der Faktenmenge $F = \{b(t), m(t)\}$.

WH: schrittweise Erweiterung der Faktenmenge um gültige Fakten

$$
\begin{aligned}
F_0 &= \{b(t), m(t), w(a)\} \\
F_1 &= \{b(t), m(t), w(a), k(t)\} \\
F_2 &= \{b(t), m(t), w(a), k(t), j(t)\} = F_3 \quad \text{(Fixpunkt)}
\end{aligned}
$$

Idee: Aussage folgt genau dann aus der Wissensbasis, wenn sie im (iterativ bestimmten) Fixpunkt erfüllt ist.

Erweiterung der Faktenmenge

gegeben: logisches Programm $P = F \cup R$ mit

- Faktenmenge $F \subseteq \text{Atom}(P)$ (interpretiert als Zustand)
  repräsentiert Menge aller Instanzen der Fakten,
  Menge von Grundatomen (Herbrand-Interpretation)
  (aus Datenbank oder Grundinstanziierung)
- Regelmenge R

Operationale Semantik (Vorwärtsverkettung) definiert durch den
Ein-Schritt-Konsequenzoperator $T_P : 2^{\text{Atom}(P)} \to 2^{\text{Atom}(P)}$ mit

$$\begin{aligned} \forall M \subseteq \text{Atom}(P) : T_P(M) &= \{h \mid (b \to h) \in P \text{ und } M \models_H b\} \\ &= \{h \mid ((b_1 \wedge \cdots \wedge b_n) \to h) \in P \text{ und } \{b_1, \ldots, b_n\} \subseteq M\} \end{aligned}$$

T_P definiert die Folge

$$\begin{aligned} T_P^0(\emptyset) &= \emptyset, \quad T_P^1(\emptyset) = T_P(\emptyset) = F \\ \forall i \in \mathbb{N} : F_{i+1} &= \{h \mid ((b \to h) \in P) \wedge (F_i \models b)\} \\ T_P^*(\emptyset) &= \bigcup_{i \in \mathbb{N}} T_P^i(\emptyset) \quad \text{(kleinster Fixpunkt von } T_P\text{)} \end{aligned}$$

Satz: Für definite Programme (nur Horn-Klauseln) gilt
$T_P^*(\emptyset) =$ kleinstes Herbrand-Modell für P.

Hülleneigenschaften

Ein Hüllenoperator ist eine Funktion $f : 2^M \to 2^M$ mit den folgenden Eigenschaften (Hülleneigenschaften)

- Für alle Mengen $m, n \in 2^M$ folgt aus $m \subseteq n$, dass $f(m) \subseteq f(n)$ gilt.
  f ist monoton
- Für jede Menge $m \in 2^M$ gilt $m \subseteq f(m)$
  f ist extensiv
- Für jede Menge $m \in 2^M$ gilt $f(f(m)) = f(m)$
  f ist idempotent

Für definite Programme ist die Funktion T_P^* (Konsequenzoperator) ein Hüllenoperator. (ÜA)

Fixpunkt-Semantik für Horn-Klausel-Programme

(operationale Semantik von Datalog)
$T_P^*(\emptyset)$ ist der kleinste Fixpunkt des

- Ein-Schritt-Konsequenzoperators T_P
- Konsequenzoperators T_P^*

Für definite Programme P:

- gilt $T_P^*(\emptyset) = \bigcap \mathrm{Mod}(P)$
- ist $T_P^*(\emptyset)$ das eindeutige kleinste Herbrand-Modell für P.
- Falls $T_P^n(\emptyset) = T_P(T_P^n(\emptyset))$ gilt, dann ist $T_P^n(\emptyset) = T_P^*(\emptyset)$.
- Für endliche (grundinstanziierte) Programme P wird $T_P^*(\emptyset)$ nach endlich vielen Anwendungen von T_P erreicht.

Folgern / Schließen aus definitem Programm P

- Ein Atom a folgt genau dann aus P ($P \models_H a$), wenn $a \in T_P^*(\emptyset)$.
- Ein negiertes Atom $\neg a$ folgt genau dann aus P ($P \models_H \neg a$), wenn $a \notin T_P^*(\emptyset)$.
- Vorsicht bei Fortsetzung auf Formeln, übliche Semantik der Junktoren ist dafür i.A. nicht erfüllt.

Unvollständiges Wissen

kommt praktisch überall vor

einige mögliche Quellen der Unvollständigkeit:

- Aussagen mit unbekanntem Wahrheitswert
- Unvollständige Beschreibung der Situation
- Abstraktion von unwichtig erscheinenden Details
- Falsche Wahrnehmung
- Kein sicheres Wissen über zukünftige Aussagen
- natürlichsprachliche ungenaue Formulierungen

Schließen und Treffen sinnvoller Entscheidungen oft trotzdem möglich.

Regeln (Wiederholung)

Formen von Regeln:

allgemeine Implikation: $\varphi \rightarrow \psi$
mit beliebigen (aussagen- oder prädikatenlogischen) Formeln

definite Regel: $l_1 \wedge \ldots \wedge l_n \rightarrow h$
mit Atomen l_i, h (Horn-Klausel)

normale Regel: $l_1 \wedge \ldots \wedge l_n \rightarrow h$
mit Literalen l_i und Atom h

allgemeine Regel: $l_1 \wedge \ldots \wedge l_n \rightarrow h$
mit Literalen l_i, h

Allgemeines (normales, definites) logisches Programm:
endliche Menge von allgemeinen (normalen, definiten) Regeln.

Beispiel

Wissensbasis:

F1 mann(Paul).

F2 mann(Otto).

F3 verheiratet(Paul).

R junggeselle(X) :– mann(X), ¬ verheiratet(X).

(R ist keine definite, aber normale Regel)

Anfrage: ?- junggeselle(X).

intuitive (gewünschte) Antworten:

- ja, mit Substitution X $\mapsto$ Otto
  Begründung: Otto-Instanz der Regel R
  mann(Otto) $\wedge\neg$ verheiratet(Otto).
  feuert in der Faktenbasis $\{F1, F2, F3\}$
- aber nicht mit Substitution X $\mapsto$ Paul
  Begründung: Paul-Instanz der Regel R
  mann(Paul) $\wedge\neg$ verheiratet(Paul).
  feuert in der Faktenbasis $\{F1, F2, F3\}$ wegen F3 nicht

Negative Voraussetzungen

Frage: Wann gilt $\neg p$ in einer Faktenbasis F?

Ansätze:

starke Negation : $\neg p$ gilt genau dann, wenn $\neg p \in F$

Vorteil: positive Antwort immer korrekt

Probleme:

- erfordert Verwaltung negativer Fakten in Faktenbasis
- Was gilt, falls weder p noch $\neg p$ in F? (Unbestimmtheit)
- Was gilt, falls sowohl p als auch $\neg p$ in F? (Inkonsistenz)

schwache Negation :

Nicht aus der Wissensbasis ableitbare Aussagen werden als unwahr angenommen.
(analog Freispruch aus Mangel an Beweisen)
Vorteil: ergibt immer eine Antwort (zweiwertig)
Problem: nach Erweiterung der Wissensbasis evtl. ungültig
(z.B. nach Hinzufügen des Faktes F4: verheiratet(Otto))

Closed World Assumption

CWA: Der Anwendungsbereich ist durch die Wissensbasis vollständig beschrieben.

Damit gilt insbesondere

- Jede im Anwendungsbereich gültige Aussage ist aus der Wissensbasis ableitbar.
- Jede nicht aus der Wissensbasis ableitbare Aussage gilt im Anwendungsbereich nicht.
  (also gilt ihre Negation)

entspricht der Idee der schwachen Negation

Normal logische Programme

(negative Voraussetzungen erlaubt)

normal logisches Programm P (Wissensbasis) enthält:

- Menge R von Regeln der Form

$$p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge \neg q_{m_i} \rightarrow h$$

  mit Atomen p_i, q_i, h

spezielle Regeln:

- Regeln mit leerem Rumpf: h (Fakten)
- Regeln mit leerem Kopf: $p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge \neg q_{m_i}$ (Constraints)

Beispiel: $\{p \rightarrow q, \neg q \rightarrow r, \neg r \rightarrow q, p\}$

Modelle normal logischer Programme

für normal logisch Programme P:

Herbrand-Universum U_P von P:
: Menge aller Grundatome über der Signatur von P, betrachtet als Aussagevariablen (Grundinstanziierung)

Herbrand-Interpretation I für P:
: Menge von Grundatomen $I \subseteq U_P$ aus dem Herbrand-Universum von P
Belegung der Aussagevariablen ist charakteristische Funktion von I

Herbrand-Modell für P:
: Herbrand-Interpretation $I \subseteq U_P$ mit $I \in \mathrm{Mod}(P)$
(Belegung = charakteristische Funktion)

Beispiel: $P = \{p \to q, \neg q \to r, \neg r \to q, p\}$

- Herbrand-Universum $U_P = \{p, q, r\}$
- (eine mögliche) Herbrand-Interpretation: $I = \{p, r\}$
- $\{p, q, r\}, \{p, q\}$ sind Herbrand-Modelle für P
- $\{q, r\}, \emptyset$ sind keine Herbrand-Modelle für P
- $\mathrm{Mod}_H(P) = \{\{p, q, r\}, \{p, q\}\}$, darunter ist $\{p, q\}$ minimal

Schließen aus Regeln mit negativen Bedingungen

Regel der Form $p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge q_{m_i} \rightarrow h$ mit

- positiven Bedingungen $p_1, \ldots, p_n$
- negativen Bedingungen $q_1, \ldots, q_m$

ist in der Faktenmenge M genau dann anwendbar, wenn

$$M \models_H (p_1 \wedge \cdots \wedge p_n \wedge \neg q_1 \wedge \cdots \wedge q_m)$$

also

- $\{p_1, \ldots, p_n\} \subseteq M$ und
- $\{q_1, \ldots, q_m\} \cap M = \emptyset$

Vorwärtsverkettung ist auch für Wissensbasen mit Regeln mit (schwacher) Negation möglich.
Ein-Schritt-Konsequenzoperator $T_P : 2^{\text{Atom}(P)} \rightarrow 2^{\text{Atom}(P)}$ definiert durch

$$\begin{aligned} T_P(M) &= \{h \mid b \rightarrow h \in P \text{ und } M \models_H b\} \\ &= \left\{ h \mid \begin{array}{l} ((p_1 \wedge \cdots \wedge p_n \wedge \neg q_1 \wedge \cdots \wedge \neg q_m) \rightarrow h) \in P \\ \text{und } \{p_1, \ldots, p_n\} \subseteq M \text{ und } \{q_1, \ldots, q_m\} \cap M = \emptyset \end{array} \right\} \end{aligned}$$

(Definition von T_P für Hornklausel-Programme P ist Spezialfall)

Nichtmonotones Schließen

Problem beim Schließen mit Regeln mit negativen Bedingungen:

- Als falsch angenommene Voraussetzungen können sich später im Laufe der Fixpunkt-Berechnung als wahr herausstellen.
- Voraussetzungen früher angewendeter Regeln gelten damit evtl. nicht mehr.

Für Programme mit Regeln mit negativen Voraussetzungen sind daher

- der Ein-Schritt-Konsequenzoperator T_P und
- der Konsequenzoperator T_P^*

i.A. nicht monoton.
Beispiele:

- $P = \{\neg p \to q, \neg q \to p\}$
  $T_P^0(\emptyset) = \emptyset$, $T_P^1(\emptyset) = \{p, q\}$, $T_P^2(\emptyset) = T_P(\{p, q\}) = \emptyset, \ldots$
  T_P^0 hat keinen Fixpunkt (Oszillation zwischen $\emptyset$ und $\{p, q\}$)
  $T_P^*(\emptyset) = \{p, q\}$ ist kein Fixpunkt für T_P (aber für T_P^*)
- $P' = P \cup \{p\}$
  $T_{P'}^0(\emptyset) = \emptyset$, $T_{P'}^1(\emptyset) = \{p, q\}$, $T_{P'}^2(\emptyset) = T_{P'}(\{p, q\}) = \{p\} = T_{P'}^3(\emptyset)$
  $T_{P'}$ hat Fixpunkt $\{p\} \neq \{p, q\} = T_{P'}^*(\emptyset)$

Auswahl intuitiver Modelle

WH: für definite logische Programme (Horn-Klausel-Programme):

- existiert (eindeutiges) kleinstes H-Modell $\min \mathrm{Mod}(P)$ für P
- gilt: $\min \mathrm{Mod}(P) = \bigcap \mathrm{Mod}(P)$
  $= T_P^*(\emptyset) =$ kleinster Fixpunkt von T_P

Beispiele: $P = \{p \to q\}$, $P' = \{p \to q, p\}$

$\min \mathrm{Mod}(P)$ ist das eindeutige intuitive Modell

intuitiver Folgerungsbegriff:
Aussage a folgt aus P gdw. $a \in \min \mathrm{Mod}(P)$

für normal logische Programme:

- existiert i.A. kein kleinstes Modell für P
- hat T_P i.A. keinen kleinsten Fixpunkt
- sind die Fixpunkte von T_P gute Kandidaten für intuitive Modelle

Beispiel: $P = \{\neg p \to q\}$
Was soll $P \models \varphi$ bei mehreren (intuitiven) Modellen bedeuten?

Eigenschaften intuitiver Modelle

Eigenschaften von Interpretationen I des logischen Programmes P:

abgeschlossen unter P:
für jede Regelinstanz $B \rightarrow h$ aus P gilt:
falls $I \models B$, dann $h \in I$

begründet: für jedes $p \in I$ existiert eine Ableitung (Begründung) für p in I

Eigenschaften von Modellen I eines logischen Programmes P:

minimal: falls $J \subseteq I$ und $J \in \mathrm{Mod}(P)$, dann gilt $J = I$

Intuitive Modelle: (minimale) Modelle für P, die begründet und unter P abgeschlossen sind.

Intuitive Folgerungen aus normal logischen Programmen

WH: Für definite Programme P existiert ein eindeutiges intuitives Modell, nämlich das (eindeutige) kleinste Herbrand-Modell M von P.
damit definiert: Atom a folgt genau dann aus P ($P \models_H a$), wenn $a \in M$.

Normal logische Programme P haben i.A. kein kleinstes Herbrand-Modell, sondern eine (evtl. leere) Menge intuitiver Modelle.
Beispiel: $P = \{\neg p \to q, \neg q \to p\}$ mit $\mathrm{Mod}(P) = \{\{p\}, \{q\}, \{p, q\}\}$,
darunter intuitive (minimale) Modelle $\mathrm{Mod}_i(P) = \{\{p\}, \{q\}\}$

Zwei prominente intuitive Folgerungsbegriffe aus Menge $\mathrm{Mod}_i(P)$ mehrerer intuitiver Modelle von P: Atom / Formel φ folgt aus P

leichtgläubig (credulous, $P \models_c \varphi$)
gdw. $\exists M \in \mathrm{Mod}_i(P) : M \models \varphi$
mit P aus Beispiel oben:
$P \models_c p$, $P \models_c q$, $P \models_c p \vee q$, aber nicht $P \models_c p \wedge q$

skeptisch ($P \models_s \varphi$)
gdw. $\forall M \in \mathrm{Mod}_i(P) : M \models \varphi$
mit P aus Beispiel oben:
$P \not\models_s p$, $P \not\models_s q$, aber $P \models_s p \vee q$

Gelfond-Lifschitz-Transformation

gegeben:

- normal logisches Programm P
- Herbrand-Interpretation (Modell) I für P

I-Redukt von P:

$$P^I = \left\{ p_1 \wedge \ldots \wedge p_m \rightarrow h \mid \begin{array}{l} p_1 \wedge \ldots \wedge p_m \wedge \neg q_1 \wedge \ldots \wedge \neg q_n \rightarrow h \in R \\ \text{und} \{q_1, \ldots, q_n\} \cap I = \emptyset \end{array} \right\}$$

Beispiel: $P = \{p \rightarrow q, \neg q \rightarrow r, \neg r \rightarrow q, p\}$, $I = \{p, r\}$
Programmtransformation von P abhängig von I:

1. Alle Regeln mit negativen Bedingungen $\neg q_i$ mit $q_i \in I$ aus P entfernen.
2. Alle negativen Bedingungen aus allen in P verbliebenen Regeln entfernen.

Für jedes normale logische Programm P und jede Interpretation I ist das I-Redukt P^I ein definites Programm.

Der Konsequenzoperator T_{P_I} von P_I ist also monoton.

Stabile Modelle normaler logischer Programme

Bestimmung der intuitiven Modelle für ein normal logisches Programm P: Auswahl einer Teilmenge von $\mathrm{Mod}(P)$
Modell I für P heißt stabiles Modell für P, falls

$$T^*_{(P^I)}(\emptyset) = I$$

Beispiele:

- $P_1 = \{p \to q, \neg q \to r, \neg r \to q, p\}$
  - $I_1 = \{p, q\}$ ist stabiles Modell für P_1, weil
    ${P_1}^{\{p,q\}} = \{p \to q, q, p\}$ und $T^*_{\left(P_1^{\{p,q\}}\right)}(\emptyset) = \{p, q\} = I_1$
  - $I_2 = \{p, q, r\}$ ist kein stabiles Modell für P_1, weil
    ${P_1}^{\{p,q,r\}} = \{p \to q, p\}$ und $T^*_{\left(P_1^{\{p,q,r\}}\right)}(\emptyset) = \{p, q\} \neq I_2$
- $P_2 = \{\neg p \to q, \neg q \to p\}$
- $P_3 = \{\neg p \to q, p \to q, \neg q \to p\}$
- $P_4 = \{\neg p \to p\}$

Beispiel: gefärbte Graphen

Wissensbasis (Beschreibung des Problems):

- Knotenmenge $V = \{v_1, \ldots, v_n\}$
  `ecke(v1), ..., ecke(vn)`
- Kantenmenge $E = \{(v_i, v_j), \ldots\}$
  `kante(vi, vj), ...`
- Menge $C = \{r, g, b\}$ von Farben

Erzeugung der Kandidaten (jede Ecke genau eine Farbe):

```
farbe(X,r) :- ecke(X), not farbe(X,b), not farbe(X, g).
farbe(X,b) :- ecke(X), not farbe(X,r), not farbe(X, g).
farbe(X,g) :- ecke(X), not farbe(X,r), not farbe(X, b).
```

Bedingung für korrekte Färbung (Ausschlusskriterium):

```
:-  kante(X,Y), farbe(X,C), farbe(Y,C).
```

Jedes stabile Modell repräsentiert eine Lösung (korrekte Färbung)

Was bisher geschah

Einordnung von KI-Verfahren: symbolisch, statistisch

- Wissensrepräsentation und -verarbeitung in Logiken
  - Regelsysteme, Entscheidungsbäume, -tabellen, BDD
  - klassische Prädikatenlogik:
    Normalformen: bereinigt, Pränex-, Skolem-, Klauselform
    allgemeinster Unifikator (mgu(s, t)), Unifikationsalgorithmus
    prädikatenlogisches Resolutionsverfahren
    logische Programme (Syntax, deklarative Sematik)
    operationale Sematik Prolog: SLD-Resolution
    operationale Sematik Datalog: Konsequenzoperator
    (Einführung in) Nichtmonotones Schließen, stabile Modelle
- Zustandsübergangssysteme:
  - (heuristische) Suchverfahren
  - Spiele, Minimax-Werte, α-β-Pruning
  - oft Kombination mit statistischen Verfahren zur Bestimmung der Heuristiken

Statistische Verfahren in der KI

(soft computing)

Einsatz zum Lösen von Problemen,

- die unvollständig beschrieben sind
- die keine eindeutige Lösung haben
- für die keine effizienten exakten Algorithmen bekannt sind

einige Ansätze:

- Fuzzy-Logik, probabilistisches Schließen
- Maschinelles Lernen
- Künstliche neuronale Netze
- Evolutionäre Algorithmen
- Schwarm-Intelligenz

(Natürliches und) Maschinelles Lernen

(Schrittweise) Änderung eines Systems (Verfahrens zur Problemlösung), so dass es bei der zukünftigen Anwendung dasselbe oder ähnliche Probleme besser löst.

- Aufgaben (Problem): Menge von Eingaben
- Aufgabeninstanz: Eingabe
- Lösung der Instanz: Ausgabe
- Bewertung der Lösung: Zuordnung Lösung $\rightarrow$ Güte

Schritte bei der Lösung von Aufgabeninstanzen mit Lerneffekt:
Schüler (System) führt wiederholt aus:

1. verwendet ein Lösungsverfahren V für diese Aufgabe
2. bestimmt eine Lösung l der gegebenen Aufgabeninstanz
3. bestimmt (oder erfährt) eine Bewertung dieser Lösung l
4. modifiziert das Lösungsverfahren V zu V', um (in Zukunft) Lösungen mit besseren Bewertungen zu finden
5. wendet im nächsten Schritt zur Lösung dieser Aufgabe das Lösungsverfahren V' an

Lernen: Schritte 3 und 4

Lernverfahren

Lernen durch

- Auswendiglernen (gegebener Beispiele)
- Nachahmen
- Anleitung (Anweisungen)
- logische Ableitung neuer Lösungsverfahren
- Analogie (zu gegebenen Beispielen)
  anhand Ähnlichkeit
- Erfahrung (durch gegebene Beispiele)
  Fähigkeit zur Verallgemeinerung
- Probieren und Beobachten
  (Erzeugen eigener Beispiele)

nach Art des Lernenden:

- natürliches Lernen
- maschinelles (künstliches) Lernen

Lernen durch gegebene Beispiele

nach der zum Lernen verwendbaren Information:

überwachtes Lernen (supervised learning)

- korrigierendes Lernen (corrective learning)
- bestärkendes Lernen (reinforcement learning)

unüberwachtes Lernen (unsupervised learning)

gewünschte Eigenschaften des Löseverfahrens:

- Korrektheit
  der Lösungen für die gegebenen Beispiele
- Generalisierung
  „sinnvolle“ Lösungen für ähnliche Aufgaben

Korrigierendes Lernen

Trainingsmenge: Menge von Paaren (Eingabe, Ausgabe)
(partielle Funktion an Stützstellen)

Lernziel: (möglichst einfache) Funktion, die an den Stützstellen mit der Trainingsmenge übereinstimmt

Rückmeldung: Trainer sagt nach jeder Ausgabe des Lernenden die korrekte Ausgabe.

Prinzip: Lernen durch Nachahmen (mit Korrektur)

Anwendung z.B. bei

- Klassifizierung (Zuordnung von Objekten / Fällen zu Klassen, abhängig von den Merkmalen der Objekte)
  z.B. Zuordnung Sensorwerte $\rightarrow$ Alarmklasse
  Trainingsmenge ist
  Menge von Paaren (Objekteigenschaften, Klasse)
- Lernen von Funktionen: Trainingsmenge ist
  Menge von Paaren (Parameter, Funktionswert)

Bestärkendes Lernen (reinforcement learning)

Trainingsmenge: Menge von Paaren (Eingabe, Erfolg $\in \{$ja, nein$\}$)

Lernziel: (möglichst einfache) Funktion, die den Stützstellen korrekte Werte zuordnet

Rückmeldung: Trainer sagt nach jeder Ausgabe des Lernenden, ob diese Ausgabe korrekt war.

Idee: Lernen durch Probieren

Anwendung z.B. bei

- Klassifizierung: Trainingsmenge ist
  Menge von Objekten (mit ihren Eigenschaften)
  Bewertung der Lösung: ja, falls Zuordnung zur korrekten Klasse, sonst nein
- Lernen von Plänen (Anlagestrategien, Bewegungsabläufe usw.)
  z.B. Steuern eines autonomen Fahrzeuges
  Trainingsmenge: Strecke(n),
  Folge von Paaren (Sensordaten, Steuersignale)
  Bewertung der Lösung: ja, falls Plan zum Erfolg geführt hat (z.B. Fahrzeug fährt $> n$ km ohne Eingriff) , sonst nein

Unüberwachtes Lernen

Trainingsmenge: Menge von Eingaben

Lernziel:
- Gruppierung ähnlicher Muster
- oft auch topologisch sinnvolle Anordnung

Idee: Lernen ohne Trainer (ohne Rückmeldung)

Anwendung z.B. bei

- Entdecken von Strukturen
- Selbstorganisation von Objekten zu Gruppen
  (mit gemeinsamen Merkmalen, typische Vertreter)
- topologieerhaltende Abbildungen
  (analog Natur: Körperteile $\rightarrow$ Gehirnregionen)
- Assoziation (z.B. in Schrifterkennung)

Neuronale Netze

Neuron – Nerv (griechisch)

Modellierung und Simulation der Strukturen und Mechanismen im Nervensystem von Lebewesen

Biologisches Vorbild	Mathematisches Modell
Nervenzellen (Neuronen)	künstliche Neuronen
Struktur (eines Teiles) eines Nervensystems	künstliche neuronale Netze (KNN) unterschiedlicher Struktur
Aktivierung von Neuronen, Reizübertragung	künstlichen Neuronen zugeordnete Funktionen
Anpassung (Lernen)	Änderungen verschiedener Parameter des KNN

Natürliche Neuronen

ZNS besteht aus miteinander verbundenen Nervenzellen (Neuronen)

Struktur eines Neurons:

- Zellkörper
- Dendriten
- Synapsen (verstärkende, hemmende)
- Axon

Natürliche Neuronen – Funktionsweise

Informationsübertragung durch elektrochemische Vorgänge:

- aktivierte Zelle setzt an Synapsen Neurotransmitter frei,
- Neurotransmitter ändern die Durchlässigkeit der Zellmembran für Ionen an den Dendriten der empfangenden Zelle,
- Potential innerhalb der empfangenden Zelle ändert sich durch diffundierende Ionen,
- überschreitet die Summe der an allen Synapsen entstandenen Potentiale (Gesamtpotential) der Zelle einen Schwellwert, entsteht ein Aktionsspotential (Zelle feuert),
- Aktionspotential (Spannungsspitze) durchquert das Axon (Nervenfaser) zu den Synapsen zu Nachbarzellen,
- aktivierte Zelle setzt an Synapsen Neurotransmitter frei, usw.

Stärke der Information durch Häufigkeit der Spannungsspitzen (Frequenzmodulation).

Eigenschaften natürlicher neuronaler Netze

- geringe Taktrate $10^{-}3$ s
- parallele Arbeit sehr vieler (10^{11}) Neuronen
- Neuronen sehr stark miteinander vernetzt
  (ca. 10 000 Nachbarn)
- Verarbeitungseinheit = Speicher

Vorteile:

- hohe Arbeitsgeschwindigkeit durch Parallelität,
- Funktionsfähigkeit auch nach Ausfall von Teilen des Netzes,
- Lernfähigkeit,
- Möglichkeit zur Generalisierung

Ziel: Nutzung dieser Vorteile zum Problemlösen
durch Modellierung der Zielfunktion (und deren Berechnung)
als künstliche neuronale Netze

Natürliche Neuronen – Lernen

Speicherung von Informationen durch Anpassung der Durchlässigkeit (Leitfähigkeit) der Synapsen

- Regel von Hebb (1949):
  Synapsen zwischen gleichzeitig aktiven Zellen werden immer durchlässiger (Reizschwelle wird verringert),
  Verbindung an dieser Synapse wird stärker
- lange nicht benutzte Synapsen verlieren mit der Zeit ihre Durchlässigkeit
  Verbindung an dieser Synapse wird schwächer.

Anwendungen künstlicher neuronaler Netze

Anwendungsgebiete:

- Bildverarbeitung, z.B.
  - Objekterkennung
  - Szenenerkennung
  - Schrifterkennung
  - Kantenerkennung
- Medizin, z.B. Auswertung von Bildern, Langzeit-EKGs
- automatische Spracherkennung
- Sicherheit, z.B. Biometrische Identifizierung
- Wirtschaft, z.B. Aktienprognosen, Kreditrisikoabschätzung
- Robotik, z.B. Lernen vom Bewegungsabläufen
- Steuerung autonomer Fahrzeuge

Geschichte künstlicher neuronaler Netze

- 1943, Warren McCulloch, Walter Pitts:
  A logical calculus of the ideas immanent in nervous activity
- 1949, Hebb: The organization of behaviour (Lernmodell)
- 1957 Frank Rosenblatt: Perzeptron (1 Schicht)
  erster Neurocomputer MARK 1
  (Ziffernerkennung in 20×20-Bildsensor)
- 1969, Marvin Minsky, Seymour Papert: Perceptrons
- 1971 Perzeptron mit 8 Schichten
- 1974 Backpropagation (Erfindung)
- 1975 / 1983, Fukushima: Cognitron, Neocognitron (Faltungsnetze)
- 1982, Teuvo Kohonen: selbstorganisierene Karten
- 1982, John Hopfield: Hopfield-Netze
- 1985, Backpropagation (Anwendung)
- 1997, long short-term memory (Erfindung)
- 2000, Begriff Deep Learning für KNN (DNN), Faltungsnetze (CNN)
- 2006, long short-term memory (Anwendung)
- 2009, verstärkt Training mit GPUs
- 2012, Wiederentdeckung Faltungsnetze
- 2017, AlphaGo, AlphaGo Zero, ...

Künstliche Neuronen: McCulloch-Pitts-Neuron ohne Hemmung

einfaches abstraktes Neuronenmodell von
McCulloch und Pitts, 1943

Aufbau eines künstlichen Neurons u (Tafel)

Eingabe:	$x = (x_1, \ldots, x_{m_u}) \in \{0,1\}^{m_u}$	(ankommende Reize)
Schwellwert:	$\theta_u \in \mathbb{R}$	(Reizschwelle)
Ausgabe:	$f(x_1, \ldots, x_{m_u}) \in \{0,1\}$	(weitergegebener Reiz)

Parameter eines McCulloch-Pitts-Neurons u ohne Hemmung:

- m_u: Anzahl der (erregenden) Eingänge
- θ_u: Schwellwert

McCulloch-Pitts-Neuron ohne Hemmung: Funktionen

Eingangsfunktion des Neurons u: I_u: $\{0,1\}^{m_u} \to \mathbb{R}$ mit

$$I_u(x_1, \ldots, x_{m_u}) = \sum_{i=1}^{m_u} x_i$$

(Summe aller erregenden Eingänge des Neurons u)
Aktivierungsfunktion des Neurons u (abhängig vom Schwellwert θ_u): A_u: $\mathbb{R} \times \mathbb{R} \to \{0,1\}$ mit

$$A_u(\theta_u, v) = \begin{cases} 1 & \text{falls } v \geq \theta_u \\ 0 & \text{sonst} \end{cases}$$

(Stufenfunktion mit Stufe bei θ_u)
Ausgabefunktion des Neurons u: O_u: $\{0,1\} \to \{0,1\}$ mit

$$O_u(v) = v$$

(Identität)

McCulloch-Pitts-Neuron ohne Hemmung: Berechnung

vom Neuron u berechnete Funktion: $f_u: \{0,1\}^{m_u} \to \{0,1\}$ mit

$$\begin{aligned} f_u(x_1, \ldots, x_{m_u}) &= O_u(A_u(\theta_u, I_u(x_1, \ldots, x_{m_u}))) \\ &= \begin{cases} 1 & \text{falls } \sum_{i=1}^{m_u} x_i \geq \theta_u \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

m_u-stellige Boolesche Funktion

McCulloch-Pitts-Neuron ohne Hemmung: Beispiele

elementare Boolesche Funktionen $\vee, \wedge$

mehrstellige $\vee, \wedge$

Existiert zu jeder Booleschen Funktion $f : \{0,1\}^n \to \{0,1\}$ ein McCulloch-Pitts-Neuron ohne Hemmung, welches f berechnet?

Nein, nur monotone Boolesche Funktionen,
z.B. $\neg$ nicht

Warum?

Geometrische Interpretation

Jedes McCulloch-Pitts-Neuron u mit m_u Eingängen teilt die Menge $\{0,1\}^{m_u}$ in zwei Teilmengen:

$$\begin{aligned} f_u^{-1}(1) &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid f(x_1,\ldots,x_{m_u}) = 1\} \\ &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid \sum_{i=1}^{m_u} x_i \geq \theta_u\} \end{aligned}$$

und

$$\begin{aligned} f_u^{-1}(0) &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid f(x_1,\ldots,x_{m_u}) = 0\} \\ &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid \sum_{i=1}^{m_u} x_i < \theta_u\} \end{aligned}$$

geometrische Interpretation als Teilräume des R^m
Grenze zwischen beiden Bereichen:
$(m_u - 1)$-dimensionaler Teilraum $\sum_{i=1}^{m_u} x_i = \theta$
parallele Schnitte (abhängig von θ)

Geometrische Interpretation: Beispiele

Beispiele:

- Neuron u mit $m_u = 2$ Eingängen und Schwellwert $\theta_u = 1$

$$f_u(x_1, x_2) = \begin{cases} 1 & \text{falls } x_1 + x_2 \geq 1 \\ 0 & \text{sonst} \end{cases}$$

Bereich der x_1, x_2-Ebene mit $f_u(x_1, x_2) = 1$ ist die Halbebene mit $x_2 \geq 1 - x_1$.

$x_2 = g(x_1) = 1 - x_1$ ist eine lineare Trennfunktion zwischen den Halbebenen mit $f_u(x_1, x_2) = 0$ und $f_u(x_1, x_2) = 1$.

- Neuron v mit $m_v = 3$ Eingängen und $\theta_v = 1$

Linear trennbare Funktionen

Zwei Mengen $A, B \subseteq \mathbb{R}^n$ heißen genau dann linear trennbar, wenn eine lineare Funktion $g : \mathbb{R}^n \to \mathbb{R}$ mit $g(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i x_1$ existiert, so dass

- für alle $(x_1, \ldots, x_n) \in A$ gilt $g(x_1, \ldots, x_n) > 0$
- für alle $(x_1, \ldots, x_n) \in B$ gilt $g(x_1, \ldots, x_n) < 0$

(eindeutig beschreiben durch $n+1$-Tupel $(a_0, a_1, \ldots, a_n)$)

Eine Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ heißt genau dann linear trennbar, wenn die Mengen $f^{-1}(0)$ und $f^{-1}(1)$ linear trennbar sind.

Beispiele: $\vee, \wedge, \neg x_1, x_1 \to x_2, x_1 \wedge \neg x_2$

Die Boolesche Funktion `XOR` ist nicht linear trennbar.

McCulloch-Pitts-Neuron mit Hemmung

McCulloch-Pitts-Neuron u mit Hemmung:

Eingabewerte:	$x = (x_1, \ldots, x_{m_u}) \in \{0,1\}^{m_u}$	erregend
	$y = (y_1, \ldots, y_{m'_u}) \in \{0,1\}^{m'_u}$	hemmend
Schwellwert:	$\theta_u \in \mathbb{R}$	
Ausgabe:	$f(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u}) \in \{0,1\}$	

Parameter eines McCulloch-Pitts-Neurons u (mit Hemmung):

- m_u: Anzahl der erregenden Eingänge
- m'_u: Anzahl der hemmenden Eingänge
- θ_u: Schwellwert

Funktionen bei hemmenden Eingängen

Eingangsfunktion des Neurons u: $I_u : \{0,1\}^{m_u+m'_u} \to \mathbb{R} \times \mathbb{R}$

$$I_u(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u}) = \left(\sum_{i=1}^{m_u} x_i, \sum_{i=1}^{m'_u} y_i \right)$$

(Summe aller erregenden Eingänge des Neurons u,
Summe aller hemmenden Eingänge des Neurons u)
Aktivierungsfunktion des Neurons u (abhängig von θ_u):
$A_u : \mathbb{R} \times (\mathbb{R} \times \mathbb{R}) \to \{0,1\}$

$$A_u(\theta_u, (x,y)) = \begin{cases} 1 & \text{falls } x \geq \theta_u \text{ und } y \leq 0 \\ 0 & \text{sonst} \end{cases}$$

(Stufenfunktion)
Ausgabefunktion des Neurons u: O_u: $\{0,1\} \to \{0,1\}$ mit

$$O_u(v) = v$$

(Identität)

Berechnung bei hemmenden Eingängen

Gesamtfunktion des Neurons u

$$f_u(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u}) = O_u(A_u(\theta_u, I_u(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u})))$$

Jedes McCulloch-Pitts-Neuron u mit m_u erregenden Eingängen, m'_u hemmenden Eingängen und Schwellwert θ_u repräsentiert die Boolesche Funktion $f_u : \{0,1\}^{m_u+m'_u} \to \{0,1\}$:

$$f_u(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u}) = \begin{cases} 1 & \text{falls } \sum_{i=1}^{m_u} x_i \geq \theta_u \\ & \text{und } \sum_{i=1}^{m'_u} y_i \leq 0 \\ 0 & \text{sonst} \end{cases}$$

Beispiele mit Hemmung:

- elementare Boolesche Funktion: $\neg$
- komplexere Boolesche Funktionen, z.B.
  $x_1 \wedge \neg x_2$
  $\neg x_1 \wedge x_2 \wedge x_3$,
  $\neg(x_1 \vee \neg x_2 \vee \neg x_3)$

Was bisher geschah

- biologisches Vorbild künstlicher Neuronen und künstlicher neuronaler Netze
- biologische Lernvorgänge
- mathematisches Modell: McCulloch-Pitts-Neuron
  - Boolesche Eingänge (erregend, hemmend)
  - ein Boolescher Ausgang
  - Eingangs- , Aktivierungs- und Ausgangsfunktion
  - berechnet Boolesche Funktion
  - geometrische Interpretation, Teilung des Raumes in zwei Mengen
  - linear trennbare Mengen / Boolesche Funktionen
  - Analogie zu logischen Gattern
- McCulloch-Pitts-Neuron
  ohne / mit (absolut) hemmenden Eingängen

McCulloch-Pitts-Netze

McCulloch-Pitts-Netz:

gerichteter Graph mit

- McCulloch-Pitts-Neuronen als Ecken und
- gerichteten Kanten zwischen Neuronen
  zwei Arten: erregend, hemmend

Berechnung der Neuronen-Funktionen
(entsprechend Struktur des Netzes):

- parallel
- sequentiell
- rekursiv

McCulloch-Pitts-Netze

Ein-Schicht-McCulloch-Pitts-Netz

parallele Schaltung mehrerer
Mc-Culloch-Pitts-Neuronen
repräsentiert Boolesche Funktionen mit mehreren Ausgaben
Beispiel: Parallelschaltung von $x_1 \wedge \neg x_2$ und $\neg x_1 \wedge x_2$

Mehr-Schicht-McCulloch-Pitts-Netz

parallele und sequentielle Schaltung mehrerer
Mc-Culloch-Pitts-Neuronen
Beispiel: XOR

Analogie zu logischen Schaltkreisen

Jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ lässt sich durch ein McCulloch-Pitts-Netz berechnen.

McCulloch-Pitts-Netz mit zwei Schichten genügt
(analog DNF, CNF in Aussagenlogik)

Modifikationen von McCulloch-Pitts-Neuronen

- Durch Vervielfachung eines Einganges erhöht sich seine Wirkung (sein Gewicht).
- Vervielfachung (absolut) hemmender Eingänge ändert die berechnete Funktion nicht.
- relative Hemmung:
  hemmende Eingänge verhindern das Feuern der Zelle nicht völlig, sondern erschweren es (erhöhen den Schwellwert, negatives Gewicht).
- Absolute Hemmung lässt sich simulieren durch relative Hemmung mit großer Schwellwerterhöhung (auf Anzahl aller erregenden Eingänge $+1$).
- Durch Einführung von Gewichten wird Trennung in hemmende und erregende Eingänge überflüssig.

Parameter künstlicher Neuronen

verschiedene künstliche Neuronenmodelle unterscheiden sich in:

- Anzahl Typen der Ein- und Ausgabewerte,
- zulässige Gewichte an den Eingangskanten,
- Eingabe-, Ausgabe- und Aktivierungsfunktion

Jedes Neuron mit m Eingängen repräsentiert eine Funktion von m Eingabewerten

Schwellwertneuronen

Idee: gewichtete Eingänge

- zur Modellierung der Stärke der synaptischen Bindung
- ermöglichen Lernen durch Änderung der Gewichte

Mathematisches Modell:
Schwellwertneuron (Perzeptron)

Eingabewerte:	$x = (x_1, \ldots, x_m) \in \{0,1\}^m$	
Eingangsgewichte:	$w = (w_1, \ldots, w_m) \in \mathbb{R}^m$	
Schwellwert:	$\theta \in \mathbb{R}$	
Ausgabe:	$a(x_1, \ldots, x_m) \in \{0,1\}$	Aktivität

Parameter eines Schwellwertneuronens u:

- m_u: Anzahl der (erregenden) Eingänge
- $(w_1, \ldots, w_{m_u}) \in \mathbb{R}^{m_u}$: Eingangsgewichte
- θ_u: Schwellwert

Schwellwertneuronen: Funktionen

Eingangsfunktion des Neurons u (abhängig von $(w_1, \ldots, w_{m_u})$):
I_u: $\mathbb{R}^{m_u} \times \{0,1\}^{m_u} \to \mathbb{R}$ mit

$$I_u(w_1, \ldots, w_{m_u}, x_1, \ldots, x_{m_u}) = \sum_{i=1}^{m_u} w_i x_i$$

(gewichtete Summe aller Eingänge des Neurons u)
Aktivierungsfunktion des Neurons u (abhängig von θ_u):
A_u: $\mathbb{R} \times \mathbb{R} \to \{0,1\}$ mit

$$A_u(\theta_u, v) = \begin{cases} 1 & \text{falls } v \geq \theta_u \\ 0 & \text{sonst} \end{cases}$$

(Stufenfunktion)
Ausgabefunktion des Neurons u: O_u: $\{0,1\} \to \{0,1\}$ mit

$$O_u(v) = v$$

(Identität)

Schwellwertneuronen: Berechnung

vom Neuron u berechnete Funktion: f_u: $\{0,1\}^{m_u} \to \{0,1\}$ mit

$$\begin{aligned} f_u(x_1,\ldots,x_{m_u}) &= O_u(A_u(\theta_u, I_u(w_1,\ldots,w_{m_u},x_1,\ldots,x_{m_u}))) \\ &= \begin{cases} 1 & \text{falls } \langle w,x\rangle \geq \theta_u \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

Wiederholung:
$\sum_{i=1}^{n} w_i x_i = \langle w,x\rangle$ Skalarprodukt
der Vektoren $w = (w_1,\ldots,w_n)$ und $x = (x_1,\ldots,x_n)$

Jedes Schwellwertneuron u mit m_u Eingängen repräsentiert eine Boolesche Funktion $f_u : \{0,1\}^{m_u} \to \{0,1\}$

Auch mit Schwellwertneuronen lassen sich nur linear trennbare Boolesche Funktionen berechnen (XOR nicht).
Beispiele: $\vee, \wedge, \to$, $(x_1 \wedge \neg x_2) \vee x_3$

Schwellwertneuronen: geometrische Interpretation

Jedes Schwellwertneuron u mit m_u Eingängen teilt die Menge $\{0,1\}^{m_u}$ der Eingabevektoren (Punkte im $\mathbb{R}^{m_u}$) in zwei Teilmengen (Teilräume des R^{m_u}):

$$\begin{aligned} f_u^{-1}(1) &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid f(x_1,\ldots,x_{m_u}) = 1\} \\ &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid \langle w,x\rangle \geq \theta_u\} \end{aligned}$$

und

$$\begin{aligned} f_u^{-1}(0) &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid f(x_1,\ldots,x_{m_u}) = 0\} \\ &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid \langle w,x\rangle < \theta_u\} \end{aligned}$$

Grenze: durch $\langle w,x\rangle = \theta_u$ beschriebene (m_u-1)-dimensionale Hyperebene (Teilraum)
(parallele Schnitte)

Schwellwert als Gewicht (Bias-Neuronen)

Neuron mit Schwellwert θ
Hinzufügen eines zusätzlichen Eingangs x_0 (bias neuron)
mit Wert $x_0 = 1$ (konstant)

Gewicht des Einganges x_0: $w_0 = -\theta$

$$\sum_{i=1}^{n} w_i x_i \geq \theta \quad \text{gdw.} \quad \sum_{i=1}^{n} w_i x_i - \theta \geq 0$$

$$\text{gdw.} \quad \sum_{i=0}^{n} w_i x_i \geq 0$$

Feed-Forward-Netze (FFN)

- $V = \bigcup_{k=1}^{n} V_k$ mit $\forall i < j \in \{1, \ldots, n\} : V_i \cap V_j = \emptyset$
  Zerlegung der Menge der Neuronen in n disjunkte Schichten
- Menge der Eingangsneuronen: V_1 (je ein Eingang)
- Menge der Ausgangsneuronen: V_n (je ein Ausgang)
- Neuronen aller anderen Schichten heißen versteckte Neuronen
- $E \subseteq \bigcup_{k=1}^{n-1} V_k \times V_{k+1}$
  nur vorwärtsgerichtete Kanten zwischen benachbarten Schichten
- Gewichte bilden $m \times m$-Matrix (mit $m =$ Anzahl aller Neuronen)
- für FFN besteht die Gewichtsmatrix aus unabhängigen Blöcken
  Blöcke sind die Gewichtsmatrizen zwischen den Schichten

FFN als Berechnungsmodell:

- parallele Berechnung (in den Neuronen einer Schicht)
- sequentielle Berechnung (in miteinander verbundenen Neuronen benachbarter Schichten)

Künstliche Neuronen mit reellen Ein- und Ausgängen

Parameter:

Eingänge:	$x_1, \ldots, x_m \in \mathbb{R}^m$
Eingangsgewichte	$w_1, \ldots, w_m \in \mathbb{R}^m$
Ausgang:	$f(\langle x, w \rangle) \in \mathbb{R}$

- Eingangsfunktion $I : \mathbb{R}^m \to \mathbb{R}$
- Aktivierungsfunktion $A : \mathbb{R} \to \mathbb{R}$
- Ausgabefunktion $O : \mathbb{R} \to \mathbb{R}$

Gesamtberechnung $f : \mathbb{R}^m \to \mathbb{R}$ des Neurons:

$$f(x_1, \ldots, x_m) = O(A(I(x_1, \ldots, x_m)))$$

Klassifikation durch Ein-Schicht-FFN

Klassifikation:
Zerlegung einer Menge M von Werten in (paarweise disjunkte) Klassen $\{C_1, \ldots, C_n\}$, welche die Wertemenge vollständig überdecken

$$\bigcup_{i=1}^{n} C_i = M \quad (\forall i \neq j : C_i \cap C_j = \emptyset)$$

Klassifikation des $\mathbb{R}^m$ durch KNN:

- Eingänge $(x_1, \ldots, x_m) \in \mathbb{R}^m$
- Ausgänge $(y_1, \ldots, y_n) \in \{0, 1\}^n$
  für jede Klasse C_i ein Ausgabeneuron y_i
  Ausgang $y_i = 1$ gdw. Eingabe $(x_1, \ldots, x_m) \in C_i$

überwachtes Training des Ein-Schicht-FFN:

- zufällige Startgewichte
- schrittweise Modifikation der Gewichte zur Verringerung des Fehlers

Ein-Schicht-FFN erkennt nur linear trennbare Klassen

Problem: Wie trainiert man Mehr-Schicht-FFN?

Auswahl durch Mehr-Schicht-FFN – Beispiel

Beispiel: Auswahl aller Punkte im Einheitsquadrat

$$y = \begin{cases} 1 & \text{falls } 0 \leq x_1 \leq 1 \wedge 0 \leq x_2 \leq 1 \\ 0 & \text{sonst} \end{cases}$$

durch das 2-Schicht-FFN mit

- Eingängen x_1, x_2 und x_0 (bias)
- Ausgang y
- versteckten Neuronen $z_1, \ldots, z_4$ und z_0 (bias)
- Gewichte der ersten Schicht (zwischen (x_0, x_1, x_2) und $(z_1, \ldots, z_4)$:

$$W_1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

  z_1 feuert gdw. $x_0 - x_1 \geq 0$ gdw. $x_1 \leq 1 (= x_0)$, z_2 feuert gdw. $x_1 \geq 0$
  z_3 feuert gdw. $x_0 - x_2 \geq 0$ gdw. $x_2 \leq 1$, z_4 feuert gdw. $x_2 \geq 0$

- Gewichte der zweiten Schicht (zwischen $(z_0, \ldots, z_4)$ und y):

$$W_2 = (-7/2, 1, 1, 1, 1)^T$$

Gesamtmatrix des FFN – Beispiel

	x_0	x_1	x_2	z_0	z_1	z_2	z_3	z_4	y
x_0	0	0	0	0	1	0	1	0	0
x_1	0	0	0	0	-1	1	0	0	0
x_2	0	0	0	0	0	0	-1	1	0
z_0	0	0	0	0	0	0	0	0	$-7/2$
z_1	0	0	0	0	0	0	0	0	1
z_2	0	0	0	0	0	0	0	0	1
z_3	0	0	0	0	0	0	0	0	1
z_4	0	0	0	0	0	0	0	0	1
y	0	0	0	0	0	0	0	0	0

Was bisher geschah

- biologisches Vorbild neuronaler Netze und Lernvorgänge darin
- maschinelle Lernverfahren:
  - überwacht
    - korrigierend, z.B. durch Δ-Regel
    - bestärkend
  - unüberwacht
- künstliche Neuronen (mit binären Ein- und Ausgängen):
  - McCulloch-Pitts-Neuron (ohne Eingangsgewichte)
  - Schwellwertneuron (mit Eingangsgewichten)
- parallele und sequentielle Kombination künstlicher Neuronen
- Feed-Forward-Netze
  gerichteter Graph mit Schicht-Struktur und Kantengewichten (Gewichtsmatrix)

Überwachtes Lernen einzelner Schwellwertneuronenn

Aufgabe: Konstruktion eines Schwellwertneurons zur Berechnung einer Booleschen Funktion $f : \{0,1\}^m \to \{0,1\}$

Trainingsmenge: Menge T von Paaren (x, t) aus

- Eingabevektoren $x \in \{0,1\}^m$ und
- Funktionswerten $t = f(x) \in \{0,1\}$

(Werte der Funktion f an Stützstellen)

Struktur des Schwellwertneuronens: Schwellwertneuron mit $m+1$ Eingängen (bias x_0) und Eingangsgewichten $(w_0, \ldots, w_m) \in \mathbb{R}^{m+1}$

Idee: automatisches Lernen der Funktion durch (wiederholte) Änderung der Gewichte

Lernziel: Gewichte $(w'_0, \ldots, w'_m) \in \mathbb{R}^{m+1}$, so dass das Schwellwertneuron die Funktion f berechnet (Korrektheit an Stützstellen)

Δ-Regel

Idee: Lernen aus (der Korrektur von) Fehlern

Delta-Regel:

$$\forall i \in \{0, \ldots, m\} : w_i' = w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = \eta x_i (t - y)$$

- Trainingswert t
- vom Netz berechneter Wert y
- Lernrate $\eta \in \mathbb{R}$ (Grad der Verstärkung der Verbindung)

korrigierendes Lernen,
(Änderung, falls x_i aktiv und $y \neq t$)

Beispiel: $\neg, \wedge, \rightarrow$

Δ-Lernverfahren für Schwellwertneuronen

- Beginn mit zufälligen Eingangsgewichten $(w_0, \ldots, w_n) \in \mathbb{R}^m$ (Schwellwert als Gewicht),
- die folgenden Schritte so oft wiederholen, bis der Fehler verschwindet (oder hinreichend klein ist):
  1. Bestimmung der Schwellwertneuron-Ausgabe y für Trainingspaar (x, t)
  2. Bestimmung des Fehlers $t - y$ der tatsächlichen zur gewünschten Ausgabe vom Trainingsziel t (als Funktion $e(w_0, \ldots, w_m)$ von den aktuellen Gewichten $w_0, \ldots, w_m$),
  3. Bestimmung geeigneter Gewichtsänderungen Δw_i
  4. Zuordnung der neuen Gewichte $w_i' = w_i + \Delta w_i$ zur Verringerung des (zukünftigen) Fehlers $(e(w_0', \ldots, w_n') < e(w_0, \ldots, w_n))$

Online-Lernen und Batch-Lernen

Lernen durch schrittweise

1. Berechnung des Fehlers
2. Berechnung der notwendigen Gewichtsänderungen
3. Änderung der Gewichte

Verfahren nach Zeitpunkt der Gewichtsänderung:

Online-Lernen Berechnung von Fehler und Gewichtsdifferenzen für jedes Trainingsmuster,
Änderung der Gewichte sofort für jedes Trainingpaar

Batch-Lernen (Lernen in Epochen)
Epoche: Berechnung für jedes Paar der Trainingsmenge
Berechnung von Fehler und Gewichtsdifferenzen für die gesamte Trainingsmenge (z.B. Summe über alle Trainingpaare)
Änderung der Gewichte erst nach einer ganzen Epoche

Konvergenz des Lernverfahrens

Konvergenzsatz:
Für jede Trainingsmenge

$$T \subseteq \{(x^{(i)}, t^{(i)}) \mid \forall i \in \{1, \ldots, n\} : x^{(i)} \in \{0,1\}^m\} \wedge t^{(i)} \in \{0,1\}\},$$

für welche die Mengen
$T_0 = \{x \mid (x,0) \in T\}$ und $T_1 = \{x \mid (x,1) \in T\}$
linear trennbar sind,
terminieren sowohl Online- als auch Batch-Lernen eines Schwellwertneuronens (passender Struktur) nach endlich vielen Schritten.

Die vom so trainierten Schwellwertneuron berechnete Funktion trennt die Mengen T_0 und T_1 voneinander.

Netze aus Schwellwertneuronen

Ein-Schicht-Schwellwertneuronen-Netz

parallele Schaltung mehrerer Schwellwertneuronen
repräsentiert Boolesche Funktionen mit mehreren Ausgaben
Beispiel: Parallelschaltung von $x_1 \wedge x_2$ und $\neg x_1 \wedge \neg x_2$

Mehr-Schicht-Schwellwertneuronen-Netz

parallele und sequentielle Schaltung mehrerer Schwellwertneuronen

Jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ lässt sich durch ein Schwellwertneuronen-Netz berechnen.

Schwellwertneuronen-Netz mit zwei Schichten genügt
(analog DNF, CNF in Aussagenlogik)

Mehr-Schicht-FFN mit linearer Aktivierung

Netzeingänge:	$(x_1, \ldots, x_{k_0}) \in \mathbb{R}^m$	
Netzausgänge:	$(y_1, \ldots, y_{k_l}) \in \mathbb{R}^n$	
Neuronen (l Schichten):	$(z_1^0, \ldots, z_{k_0}^0) \in \mathbb{R}^{k_1}$	(Eingabeneuronen)
	$\vdots$	(versteckte Neuronen)
	$(z_1^l, \ldots, z_{k_l}^l) \in \mathbb{R}^{k_l}$	(Ausgabeneuronen)
Gewichtsmatrizen	$W^{(j)} \in \mathbb{R}^{k_j \times k_{j+1}}$ für jedes $j \in \{0, \ldots, l-1\}$	

lineare Aktivierungsfunktion $I : \mathbb{R} \to \mathbb{R}$ mit $I(x) = mx$

Ausgabe des Neurons z_i^j in Schicht j:

$$f(z_1^{j-1}, \ldots, z_{k_{j-1}}^{j-1}) = O(A(I(x_1, \ldots, x_{k_{j-1}}))) = m\left(\sum_{l=1}^{k_{j-1}} w_{li}^{(j)} z_l^{(j-1)}\right)$$

Netzausgabe:

$$f(x_1, \ldots, x_m) = m'(x_1, \ldots, x_m)W^{(0)} \cdots W^{(l-1)} = m'(x_1, \ldots, x_m)W$$

mit $W = W^{(0)} \cdots W^{(l-1)}$ (Matrixmultiplikation)

Jede Funktion, die von einem Mehr-Schicht-FFN mit linearer Aktivierung berechnet wird, kann also auch durch ein Ein-Schicht-FFN mit linearer Aktivierung berechnet werden.

Perzeptron (historisch)

1958 Frank Rosenblatt, Idee: Modell der Netzhaut (Retina)
Aufbau des Perzeptrons:

1. Schicht (Eingabeschicht) : Menge S von Stimulus-Zellen (Verteilung)
2. Schicht (Mittelschicht) : Menge A von Assoziations-Zellen (Vorverarbeitung)
3. Schicht (Perzeptron-Schicht) : Menge R von Response-Zellen Muster-Assoziator aus Schwellwertneuronen (eigentliche Verarbeitung)

Verbindungen:

- zufällig zwischen Neuronen der Eingabeschicht und Neuronen der Mittelschicht
  feste Gewichte (zufällig)
- von jedem Neuron der Mittelschicht zu jedem Neuron der Ausgabeschicht
  trainierbare Gewichte

Jedes Ausgabeneuron teilt die Eingabemuster in zwei Klassen (akzeptierte und nicht-akzeptierte)

Ein-Schicht-FFN

- Abstraktion von der Eingabeschicht im historischen Perzeptron-Modell
- nur Perzeptron-Schicht (Muster-Assoziator)
- Parallele Berechnung mehrerer künstlicher Neuronen (hier Schwellwertneuronen)

Eingänge:	$(x_1, \ldots, x_m) \in \{0,1\}^m$
Ausgänge:	$(y_1, \ldots, y_n) \in \{0,1\}^n$
Gewichtsmatrix	$W \in \mathbb{R}^{m \times n}$

Gesamtberechnung des Ein-Schicht-FFN $f : \{0,1\}^m \to \{0,1\}^n$ des Neurons mit gewichteter Summe als Aktivierungsfunktion:
$f(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$ mit $\forall k \in \{1, \ldots, n\}$:

$$y_k = \begin{cases} 1 & \text{falls } \sum_{i=1}^{m} = x_i w_{ij} \geq 0 \\ 0 & \text{sonst} \end{cases}$$

(Matrixmultiplikation)

Ein-Schicht-FFN: Training mit Δ-Regel

überwachtes Lernen

Trainingsmenge: Menge von Paaren (x, t) aus

- Eingabevektoren $x \in \{0,1\}^m$ und
- gewünschten Ausgabeverktoren $t \in \{0,1\}^n$

Lernen mit Delta-Regel für Ein-Schicht-FFN:

- Beginn mit zufälligen Eingangsgewichten $w_{ij} \in \mathbb{R}$,
- für jede Eingabe der Trainingsmenge (x, t):
  1. Netz berechnet die Ausgabe $y = xW$,
  2. Zuordnung neuer Gewichte w'_{ij} durch Delta-Regel:

$$w'_{ij} = w_{ij} + \Delta(w_{ij}) \quad \text{mit} \quad \Delta(w_{ij}) = \eta x_i (t_j - y_j)$$

- wiederholen, bis der Fehler klein genug ist.

Das Lernverfahren mit Delta-Regel konvergiert für

- jede linear trennbare Boolesche Funktion f und
- hinreichend kleine Lernquote η

in endliche vielen Schritten zu einem Ein-Schicht-FFN, welche die Funktion f berechnet.

Berechnung der Gewichts-Verschiebungen in Mehr-Schicht-FFN

Ziel:
Minimierung des Fehlers durch schrittweise Verschiebung des Gewichtsvektors

Methode: Gradientenabstiegsverfahren
Verschiebung des Gewichtsvektors in Richtung des steilsten Abstieges (entgegen dem steilsten Anstieg) der Fehlerfunktion (als Funktion der Gewichte)

steilster Anstieg: Gradient (partielle Ableitungen)

Gradientenabstiegsverfahren führt oft, aber nicht immer zu einem geeigneten (globalen) Minimum der Fehlerkurve,
endet mitunter in lokalem Minimum

Voraussetzung: Fehlerfunktion ist differenzierbar

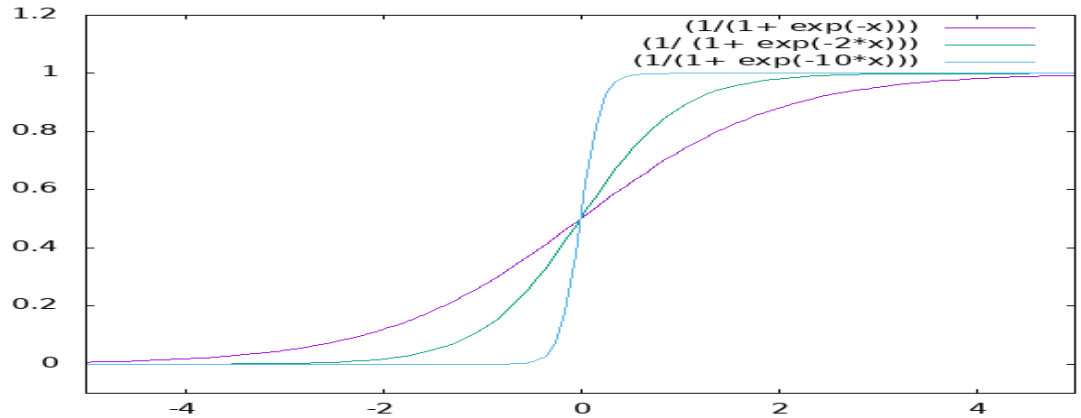
zur Anwendung in KNN: differenzierbare Aktivierungsfunktion

Sigmoide Aktivierungsfunktion

differenzierbare Approximation der Stufenfunktion:

sigmoide Funktion

$$f(x) = \frac{1}{1+e^{-x}} \quad \text{mit Parameter } c > 0: \quad f(x) = \frac{1}{1+e^{-cx}}$$



\+ überall differenzierbar
Ableitung im Punkt x:

$$s'(x) = \left(\frac{1}{1+e^{-x}}\right)' = \frac{e^{-x}}{\left(1+e^{-x}\right)^2} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = s(x)(1-s(x))$$

in jedem Punkt eindeutige Abstiegsrichtung

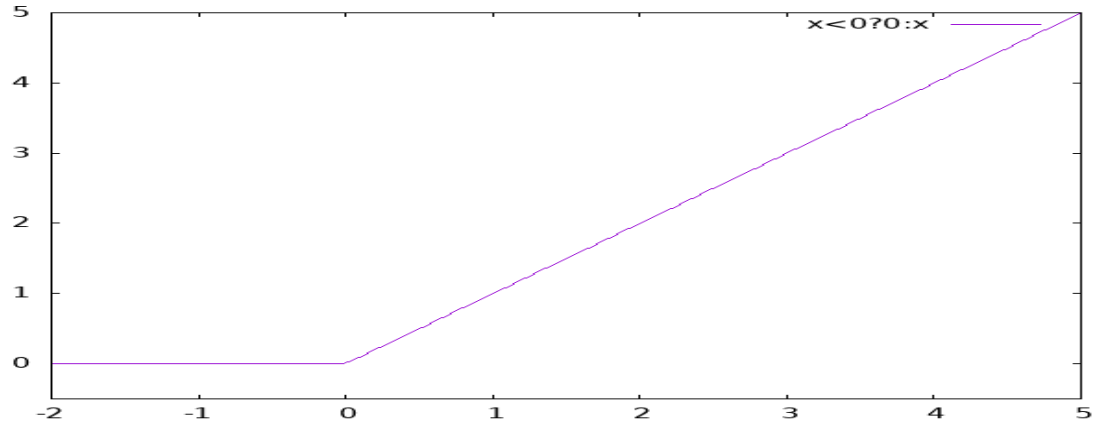
\- erreicht die Werte 0 und 1 nie,
Toleranzbereiche notwendig, so entstehen Ungenauigkeiten

Lineare Aktivierungsfunktionen und ReLU

lineare Aktivierung: $\forall x \in \mathbb{R} : A(x) = mx + n$

\+ einfach (schnell) zu berechnen
überall differenzierbar
Ableitung: konstant m
in jedem Punkt $x > 0$ eindeutige Abstiegsrichtung

ReLU(Rectified Linear Units): $\forall x \in \mathbb{R} : A(x) = \max(0, x)$



\+ einfach (schnell) zu berechnen
fast überall differenzierbar
Ableitung: Stufenfunktion, 0 bei $x < 0$, 1 bei $x > 0$,
in jedem Punkt $x > 0$ eindeutige Abstiegsrichtung

\- Problem: Ableitung nicht definiert bei $x = 0$
(aber praktisch inzwischen kaum noch relevant)

Backpropagation in FFN

(Bryson, Ho 1969, Rummelhard, McClelland 1986)

Ziel: geeigntete Erweiterung des Lernens mit Δ-Regel auf Mehr-Schicht-FFN

gesucht: geeignete Modifikation der Gewichte in allen Schichten des FFN zur Verrringerung des Gesamtfehlers

Idee: Minimierung der Fehlerfunktion durch Gradientenabstiegs-Verfahren (hier am Beispiel mit sigmoider Aktivierungsfunktion)

- Betrachte jedes Gewicht w_{uv} als Eingangsgewicht des Teilnetzes zwischen Neuron v und Netz-Ausgängen
- Netzeingabe in dieses Teilnetz ist $w_{uv} o_u$ mit Netzausgabe o_u des Neurons u
- partielle Ableitung $\frac{\partial E}{\partial w_{uv}} = o_u \delta_v$ mit Fehleranteil $\delta_v = o_v(1 - o_v) \sum_p w_{vp} \delta_p$, wobei p über alle direkten Nachfolger von v läuft

Backpropagation-Training

in jedem Schritt 2 Durchläufe des FFN:

Vorwärts-Schritt: Berechnung der Netzausgabe
Speichern der Netzausgabe o_u in jedem Neuron u
Speichern der Ableitung der Netzausgabe $o_u(1 - o_u)$ in jedem Neuron u

Rückwärts-Schritt: Berechnung des Fehleranteils δ_u jedes Neurons aus den Fehleranteilen aller Nachfolger-Neuronen
$\delta_u = o_u(1 - o_u) \sum_p w_{vu}\delta_p$,
Speichern der Fehleranteile δ_u in jedem Neuron u

danach Anpassung aller Gewichte um $\Delta w_{uv} = -\eta o_u \delta_v$

Backpropagation-Lernen für Mehr-Schicht-FFN

- Instanziierung aller Gewichte mit kleinen zufälligen Werten
- BP-Verfahren für eine Epoche:
  - BP-Verfahren für jedes Traingsmuster (x, t):
    - Vorwärtsschritt (Ausgabe-Berechnung):
      für jede Schicht s (Beginn bei Eingabeschicht):
      Berechnung der Vektoren $z^{(s)} = I(y^{(s-1)})$ und
      $y^{(s)} = A(z^{(s)}) = A(I(y^{(s-1)}))$ für jedes Neuron der Schicht s
    - Rückwärtsschritt (Gewichtsdifferenzen):
      für die Ausgabeschicht k:
      Berechnung des Vektors $d^{(k)} = (t - y^{(k)})y^{(k)}(1 - y^{(k)})$
      für jede Schicht s (Beginn bei letzter versteckter Schicht
      $k - 1$):
      Berechnung des Vektors $d_j^{(s)} = y_j^s(1 - y_j^{(s)}) \sum_{m=1}^{n^{(s+1)}} d_m^{(s+1)} w_{mj}$
      für jedes Neuron j der Schicht s
    - Aktualisierung aller Gewichte: $w_{ij}^{(s)} := w_{ij}^{(s)} + \eta d_j^{(s)} y_i^{(s)}$

    danach weiter mit nächstem Trainingsmuster (x', t')

  danach weiter mit nächster Epoche
- Ende, falls erreichte Änderung des Fehlers klein (unter einer Schranke)

Backpropagation-Lernen mit Trägheit

zur Vermeidung von

- Oszillationen in „Schluchten“ und
- Abbremsen auf Plateaus

$$w_{ij}^{(s)} := (1 + \alpha) w_{ij}^{(s)} + \eta d_j^{(s)} y_i^{(s)}$$

mit Trägheit α

Qualität von BP-Netzen

gute Generalisierung:
KNN klassifiziert die meisten neuen Eingabemuster einer Testdatenmenge (nicht aus der Trainingsmenge) richtig
abstrahiert von kleinen Abweichungen
abhängig von

- Netzarchitektur (nicht zu viele versteckte Neuronen)
- Auswahl der Trainingsmenge

Problem:
übertrainierte Netze kennen die Trainingsmenge „auswendig“

Hybride Netze

Idee: Kombination verschiedenartiger Neuronen

Beispiel:
Perzeptron (1958): 2-Schicht-FFN mit

Mittelschicht : Schwellwert-Neuronen mit zufälligen Verbindungen mit zufälligen Gewichten von Eingabeschicht, kein Trainig der Gewichte

Perzeptron-Schicht : Schwellwert-Neuronen, Training der Gewichte (von Mittelschicht) mit Δ-Regel

Cognitron

Beobachtungen im visuellen System:

- sendet vorverarbeitete Signale an Gehirn
- heterogenes Netz: verschiedene Neurone haben verschiedene Wirkungen (Funktionen)
- Neuronen derselben Schicht haben dieseslbe Funktion
- Verbindung benachbarter Neuronen
  horizontale Zellen berechnen Mittelwert (der Helligkeit)
  wirken hemmend auf Signale nahe beim Mittelwert
- ähnlich Faltung in digitaler Bildverarbeitung (Tafel):
  Funktionswert eines Pixels hängt von Werten benachbarter Pixel ab

Multiskalen-Strategien

(alte) Idee aus der digitalen Bildverarbeitung / -erkennung
Ziel: Bildanalyse in verschieden Größen (Auflösungsstufen)

- wenig aufwendige Untersuchung grober Strukturen
- feinere Untersuchung feiner Strukturen im Bild

Idee: Multiskalenraum (ähnlich menschlicher Wahrnehmung)
enthält Originalbild B_0 und weniger detaillierte Versionen B_k
Erzeugung der Multiskalen-Bilder (Pyramiden) $(B_0, B_1, B_2, \ldots)$
durch wiederholte Ausführung der Schrittfolge

1. Glättung (durch Tiefpass-Filter, z.B. Gauß-Filter)
2. reduce: Komprimierung durch geringere Abtastrate,
   z.B. Gauß-Pyramide: Löschen jeder zweiten Zeile und Spalte
3. expand: Umkehrung durch Interpolation (nicht verlustfrei)
   zur Vergleichbarkeit der Bilder verschiedener Auflösungen

Laplace-Pyramide (redundanzarme Darstellung):
Schichten enthalten Differenz
zwischen Bild B_k und expand(reduce B_k)

Neocognitron

Fukushima (1975):
Cognitron: A Self-Organizing Multilayered Neural Network Model
(1983) Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition
Motivation: Erkennung handschriftlicher Ziffern
Idee: Umsetzung der Multiskalen-Strategie als KNN
Aufbau Neocognitron:

- Eingabe-Schicht
- vier (oder mehr) versteckte Stufen aus je zwei Schichten:
  1. Transformation in 12 Bilder (Ebenen)
     Feature-Extraktion (Faltungen mit je einem 3×3-Kern)
     Filterkerne durch Eingangsgewichte definiert (weight sharing)
     Gewichte durch Trainingsmuster gelernt
  2. Kombination mehrerer transformierter Bilder
     z.B. punktweise gewichtete Summe, Max
     Gewichte nicht trainiert
- Ausgabe nach letzter Kombinations-Schicht (Klassifikation)
- inkrementelles Lernen stufenweise von Ein- zu Ausgabeschicht

mehrere Varianten mit überwachtem und unüberwachtem Lernen

Prominente Aktivierungsfunktionen

- Stufenfunktion $A(X) = (x \geq 0)$,
- sigmoide Funktion $A(x) = \frac{1}{1+e^{-x}}$, $A(x) = \tanh x$
  (differenzierbare Approximation der Stufenfunktion)
- lineare Funktion $A(x) = ax + b$
- ReLU $A(X) = \max(0, x)$,
- analytische Funktion $A(X) = \log(1 + e^x)$,
  (differenzierbare ReLU-Approximation, softplus)
- Radiale Basisfunktion $A(x) = r(d(x, w))$ mit
  Eingabe(-vektor) x, Gewicht-(svektor) w (Zentrum), Abstand d,
  radiale Funktion $r : \mathbb{R}_{\geq 0} \to [0, 1]$: monoton fallend, $r(0) = 1$
- Faltungen
- Softmax (hängt von mehreren Eingaben $(x_1, \ldots, x_n)$ ab
  $A(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$
  zur Klassifizierung in mehrere Klassen
  (Wahrscheinlichkeitsverteilung)

Convolutional Neural Networks

z.B. Alex Krizhevsky, . . . , 2012:
ImageNet Classification with Deep Convolutional Neural Networks
prinzipieller Aufbau:

- Eingabe-Schicht
- Versteckte Stufen aus je mehreren Schichten
  - Faltungs-Schicht (Feature-Maps)
    alle Gewichte gleich
  - evtl. ReLU-Schicht (nichtlinear)
  - gelegentlich Subsampling-Schicht (Pooling)

  mehrfache Wiederholung (deep), evtl. in verschiedenen Reihenfolgen
- evtl. klassische Schichten mit vollständigen Verbindungen zwischen benachbarten Schichten
- Ausgabe-Schicht

inzwischen auch komplexere Konstruktionen, z.B.

- AlexNet (Dropout-Schichten)
- GoogLeNet (Inception)
- ResNet (skip connections)

CNN-Schichten

aktuelle CNNs bestehen im Wesentlichen aus mehrfacher Wiederholung folgender Schichten:

- Faltung (convolutional)
- Auswahl (pooling, meist Durchschnitt oder Max)
- vollständig verbunden (fully connected, FC)
- Normalisierung (batch normalization, BN)
- softmax (Wahrscheinlichkeitsverteilung)

CNNs unterscheiden sich in

- Reihenfolge der Schichten
- Anzahl der Schichten / Wiederholungen
- spezielle topologische Merkmale,
  z.B. Vorwärtskanten, die Schichten überspringen

CNN – prominente Beispiele

- VGG16, ImageNet-Datenbank (seit 2010):
  - $< 15 \cdot 10^6$ annotierte Bilder
  - $< 10 \cdot 10^3$ Klassen

  ImageNet Large-Scale Visual Recognition Challenge (ILSVRC, seit 2010), Gewinner 2014: VGG16

- Nachcolorierung: Eingabe Grauwertbild, Ausgabe Farbbild
  `https://richzhang.github.io/colorization` (2016)

- SegNet (2015), Aufgabe: semantische Segmentierung
  (Jeder Bildposition wird Bedeutung zugeordnet)
  Eingabe und Ausgabe: Bild derselben Größe,
  Farben des Ausgabebildes sind Klassen
  Idee: Verknüpfung VGG16 mit „gespiegelter“ Version

CNN-Lernen

Überwachtes Lernen durch Backpropagation
(angepasste Verfahren für jeden Schicht-/ Neuronentyp)

Trainieren von CNN ist i.A. sehr aufwendig (Zeit, Daten)

deshalb häufig auch Nutzung vortrainierter Netze

Idee:

- Ausgangspunkt: Netz, welches schon auf allgemeine Daten trainiert wurde
  z.B. Klassifikation, Segmentierung
- Training auf spezielle Aufgabe anhand spezieller Datensätze
  z.B. andere bzw. feinere Klassen

CNN-Lernen

Überwachtes Lernen durch Backpropagation
(angepasste Verfahren für jeden Schicht-/ Neuronentyp)

Nutzung vortrainierter Netze zur
Beschleunigung des Trainingsprozesses

Idee:

- Ausgangspunkt: Netz, welches schon auf allgemeine Daten trainiert wurde
  z.B. Klassifikation, Segmentierung
- Training auf spezielle Aufgabe
  z.B. andere bzw. feinere Klassen

Was bisher geschah

- Maschinelles Lernen:
  - überwacht
    - korrigierend
    - bestärkend
  - unüberwacht
- Künstliche Neuronen:
  - Mathematisches Modell und Funktionen
  - McCulloch-Pitts, Schwellwertelement, allgemein
  - Eingabefunktion: gewichtete Summe, Ausgabefunktion: Identität
  - Aktivierungsfunktionen: Schwellwert, (linear,) Sigmoid, ReLU
- künstliche neuronale Netze
  - gerichteter Graph von Neuronen (Knoten) mit Kantengewichten
  - Feed-Forward-Netze (Ein-Schicht-, Mehr-Schicht-) Gewichts-Matrix
  - korrigierendes Lernen durch Backpropagation

Rekurrente Netze: Motivation

Ziel: Nachnutzung von Informationen aus vorangegangenen Schritten, z.B. zur

- Repräsentation zeitlicher Folgen von Mustern
- Zeitreihenanalyse und -voraussage
- Erkennung von Sätzen (Grammatik)
- Verarbeitung von Mustern variabler Länger (betrachtet als Sequenzen)

mögliche Ansätze

- gleitendes Zeitfenster:
  FFN mit n Eingabeneuronen
  Eingabemuster enthält Informationen aus n vorangegangenen Schritten
  Nachteil: beschränkte Breite des Zeitfensters
  - Erkennen „entfernter“ Abhängigkeiten schwierig
  - viele Eingabeneuronen nötig
- rekurrente KNN

Rekurrente Netze

Idee: direkte Abhängigkeit von Eingaben in vorangegangenen Schichten

Topologie:	Graphen mit Rückwärtskanten, ggf. Kontextneuronen zum Speichern von Zwischenwerten
Aktivierung	wie bisher, abhängig vom Neuronentyp
Lernen:	z.B. bei FFN mit Rückwärtskanten: Vorwärtskanten in der „Entwirrung“ (mehrere verbundene Kopien, für jeden Zeitschritt eine) des Netzes trainieren (Backpropagation through time) Gewichte der Rückwärtskanten meist fix
Besonderheit:	Zeitschritte definieren Zustände des Netzes Zustand: Zuordnung Neuron $\rightarrow \mathbb{R}$ (Aktivierung)

Rekurrente Netze – Beispiel

- zwei McCulloch-Pitts-Neuronen u, v
- Eingang $x \in \{0, 1\}$
- Ausgang $y \in \{0, 1\}$
- erregende Kanten: $(x, u), (x, v), (u, u), (u, v), (v, y)$
- hemmende Kanten $(v, v), (v, u)$ (Eingabe 1 verhindert Aktivierung)
- Schwellwerte $\theta_u = 1$, $\theta_v = 2$

Übergänge zwischen den Zuständen des Netzes (Aktivierung von x und y) definieren einen (deterministischen) endlichen Automaten mit Ausgabe (Tafel)

Mathematisches Modell: Rekursion

Wiederholung: KNN als Berechnungsmodell

FFN als Berechnungsmodell:

- parallele Berechnung (in den Neuronen einer Schicht)
- sequentielle Berechnung (in miteinander verbundenen Neuronen benachbarter Schichten)
  Nacheinanderausführung von Funktionen

rekurrentes Netz als Berechnungsmodell:

- mehrmalige Nacheinanderausführung einer Funktion (ohne Abbruchbedingung)
  Berechnung einer rekursiven Funktion (Fixpunkt)

„Entwirrung“ rekurrenter Netze

Idee:

- Verarbeitung von Eingaben zu Ausgaben eines Neurons kostet einen Zeitschritt
- für jeden Zeitschritt eine Kopie aller Neuronen und Kanten dazwischen,
- Ersetzung der Rückwärtskanten durch Vorwärtskanten zur nächsten Kopie
- Approximation der Rekursion durch endliche Anzahl von Kopien

In diesem expandierten Netz ist Lernen der Vorwärtskanten durch Backpropagation-Verfahren möglich:

- Durchlauf jeder Netz-Kopie ist ein Zeitschritt,
- Lernen durch Backpropagation des entwirrten KNN (Backpropagation through time, BPTT)

Beispiel: Jordan-Netze

Idee (1986): Nachnutzung der Netzausgaben
Netz-Topologie:

- Feed-Forward-Netz mit trainierbaren Vorwärtskanten,
- für jedes Ausgabeneuron ein zusätzliches Kontextneuron in der Eingabeschicht
  (zur Speicherung der Netzausgaben)
  Aktivierungsfunktion: Identität
- zusätzliche Verbindungen von jedem Neuron der Ausgabeschicht zu seinem Kontextneuron mit
  festen Gewichten λ (meist $\lambda = 1$),
  Speicherung der Ausgaben
- evtl. direkte Verbindungen von jedem Kontextneuron zu sich selbst mit festem Gewicht γ
  (zur weiteren Speicherung der Netzausgaben)
- zusätzliche Verbindungen von jedem Kontextneuron zu jedem Neuron der ersten versteckten Schicht mit
  trainierbaren Gewichten,
  (zur Verwendung der gespeicherten Ausgabe im Folgeschritt)

Long Short-Term Memory (LSTM)

Idee (1997): Nachnutzung der Aktivierung der versteckten Neuronen

- Knoten: LSTM unit (memory block + gate units) besteht aus
  - Kern: CEC (constant error carrousel)
    lineare Aktivierung und Rückkopplung zu sich selbst (feedback)
    speichert Zustand, Rückkopplung im nächsten Schritt
  - zusätzliche gate units vor und nach CEC steuern die Fähigkeit, Eingaben zu verarbeiten und speichern (input gates) bzw. Ausgaben weiterzugeben (output gates)
- Netz-Topologie: 3-Schicht-FFN, LSTM units in versteckter Schicht

prominente Modifikationen:

- forget gate steuert (multipliziert, i.A. schwächt) Gewicht der rückkoppelnden Kante
- Peephole Connections (gewichtete Kanten von CEC zu allen gates)
  bei schwacher Aktivierung: Isolation des CEC
- mehrere Schichten von LSTM-Zellen (Deep LSTM)

Anwendungen:

- Zeitreihenanalyse, -vorhersage
- Handschrifterkennung, Spracherkennung, Übersetzung

Assoziativspeicher

Ziel: Musterassoziation
(Training mit endlich vielen Musterpaaren)

Generalisierung:

- Aus Zuordnung: Muster $x \rightarrow$ Muster y
  folgt für jedes zu x ähnliche Muster x' die
  Zuordnung: Muster $x' \rightarrow$ Muster y.
- Ziel: sinnvolle Zuordnung „verrauschter“ oder unvollständiger Eingabemuster

Netztypen:

heteroassoziativ Eingabemuster $x \in \mathbb{R}^m$,
Ausgabemuster $y \in \mathbb{R}^n$

Mustererkennung Spezialfall heteroassoziativer Netze
assoziiert Muster mit Identifikator, z.B. für Klasse

autoassoziativ Spezialfall heteroassoziativer Netze
Ein- und Ausgabemuster $x \in \mathbb{R}^m$ (prinizipiell) gleich

Heteroassoziativer Speicher

Topologie: vollständig verbundenes Ein-Schicht-FFN,
Eingänge $x \in \mathbb{R}^m$, Ausgänge $y \in \mathbb{R}^n$,
alles Schwellwertneuronen, Gewichtsmatrix $W \in R^{m \times n}$

Aktivierung: Signum (Stufenfunktion)

$$A(x) = \text{sgn}(x) \begin{cases} -1 & \text{falls } x < 0 \\ 0 & \text{falls } x = 0 \\ 1 & \text{sonst} \end{cases}$$

Berechnung der Gewichte: Methode der kleinsten Quadrate =
Minimierung von

Berechnung der Eingangsfunktion analog Ein-Schicht-FFN:

$$I(x_1, \ldots, x_m) = (x_1, \ldots, x_m) \begin{pmatrix} w_{11} & \ldots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \ldots & w_{mn} \end{pmatrix}$$

dasselbe kürzer: $I(x) = xW$

Heteroassoziativer Speicher: Beispiel

Berechnung für 3-2-Netz mit Gewichtsmatrix W:

$$W = \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Eingabe $x = (1, -1, 1)$

Berechnung:

$$\text{sgn}(xW) = \text{sgn}\left((1, -1, 1)\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\right) = \text{sgn}(-3, 3) \mapsto (-1, 1)$$

Ausgabe $y = (-1, 1)$

Heteroassoziativer Speicher: Training

Idee: Lernen aus gleichzeitiger Aktivität

biologisches Vorbild: Synapsen zwischen gleichzeitig aktiven Neuronen (x_i und y) werden verstärkt (synaptische Plastizität)

Lernregel von Hebb (1949):

$$\forall i \in \{0, \ldots, m\} : w_i' = w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = \eta x_i y$$

Trainingsmenge $\{(x^{(i)}, y^{(i)}) \mid i \in \{1, \ldots, k\}\}$ (Bipolarvektoren)

Ziel Gewichtsmatrix W, so dass für jedes Trainingspaar $(x^{(i)}, y^{(i)})$ gilt: mit $\text{sgn}(x^{(i)} W) = y^{(i)}$ (komponentenweise)

Startgewichte alle $w_{ij} = 0$

Lernregel von Hebb $\Delta w_{ij} = \eta x_i y_j$, hier mit Lernrate $\eta = 1$

Training : Gewichtsbestimmung $\Delta w_{kl} = x_k y_l$ je Trainigspaar (x, y) einmal (W ist Korrelationsmatrix von x und y)

Alternative zum Training: direkte Berechnung der Gewichte (z.B. mit Methode der kleinsten Quadrate)

Bidirektionaler Assoziativspeicher (BAM)

(heteroassoziativer Speicher von Musterpaaren)

Netz-Topologie:

- Eingangsschicht, Ausgangsschicht, keine versteckten Neuronen
- Eingänge $x \in \{-1, 1\}^m$
- Ausgänge $y \in \{-1, 1\}^n$
- vollständige symmetrische Verbindungen zwischen jedem Eingangs- und jedem Ausgangsneuron
  (ungerichteter vollständiger bipartiter Graph $K_{m,n}$)
- Gewichte an jeder Kante (für beide Richtungen gleich)
  Gewichte in Gewichtsmatrix $W \in R^{m \times n}$
- Eingangsfunktion: gewichtete Summe
- Aktivierung: Signum
- Ausgangsfunktion: Identität

BAM: Berechnung und Training

(wie im heteroassoziativen Speicher)

überwachtes Lernen

Trainingsmenge $\{(x^{(i)}, y^{(i)}) \mid i \in \{1, \ldots, k\}\}$

Eingabe : Startzustand (initiale Zustände der Eingangsneuronen $x \in \{-1, 1\}^m$)

Lernregel von Hebb: $\Delta w_{ij} = \eta x_i y_j$ mit $\eta = 1$

Berechnung : Folge von Schritten (Zustandsübergängen),

- synchron oder
- abwechselnd

für beide Neuronenschichten:
Aktualisierung: Neuberechnung der Aktivierung der anderen Schicht
wiederholen, bis stabiler Zustand erreicht (Fixpunkt) oder Abbruch ausgelöst wird

Ausgabe : Zustand der Ausgangsneuronen des stabilen Netzes

BAM: Beispiel

ein Trainingspaar (x, y) mit
$x = (1, -1, 1)$ und $y = (-1, 1)$

Gewichtsmatrix W:

$$W = \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Berechnung:

$$\text{sgn}(xW) = \text{sgn}\left((1, -1, 1)\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\right) = \text{sgn}(-3, 3) \mapsto (-1, 1) = y$$

$$\text{sgn}(Wy^T) = \text{sgn}\left(\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix}\right) = \text{sgn}\begin{pmatrix} 2 \\ -2 \\ 2 \end{pmatrix} \mapsto \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = x^T$$

BAM: Training

(wie heteroassoziativer Speicher)

- für ein zu speicherndes Musterpaar $x \in \{-1,1\}^m, y \in \{-1,1\}^n$: $W = x^T y$
  (Korrelationsmatrix von x und y)
- für mehrere zu speichernde Musterpaare $(x^{(1)}, y^{(1)}), \ldots, (x^{(k)}, y^{(k)})$:

$$W = \sum_{i=1}^{k} W^{(i)} = \sum_{i=1}^{k} \left(x^{(i)}\right)^T y^{(i)}$$

Für alle k Trainingsmuster gleichzeitig:

- Eingabemuster $X \in \{-1,1\}^{k \times m}$
- Ausgabemuster $Y \in \{-1,1\}^{k \times n}$
- $XX^T \in \{-1,1\}^{k \times k}$ (alle Einträge positiv)
- $\text{sgn}(Y) = \text{sgn}(XX^T Y) \in \{-1,1\}^{k \times n}$
- $\text{sgn}(Y) = \text{sgn}(XW) \in \{-1,1\}^{k \times n}$ mit $W = X^T Y$

Hopfield-Netz – Neuronenmodell

Neuronenmodell (modifiziertes Schwellwertelement):

- Verarbeitung von Bipolarvektoren $x \in \{-1, 1\}^n$
- Eingangsfunktion: gewichtete Summe
- Aktivierungsfunktion (auch abhängig von der Aktivierung a im vorigen Schritt:

$$sgn(v, a) = \begin{cases} -1 & \text{falls } v < 0 \\ 1 & \text{falls } v > 0 \\ a & \text{falls } v = 0 \end{cases}$$

- Ausgangsfunktion: Identität

Hopfield-Netz

(autoassoziativer Musterspeicher)

Idee: Ein-und Ausgabeknoten in autoassoziativem BAM identifiziert

Topologie: K_n mit symmetrischer Gewichtsmatrix $W \in \mathbb{R}^n$
Jedes Neuron ist zugleich Ein- und Ausgang $x \in \{-1, 1\}^n$
keine Selbstrückkopplung, also $\forall i \in \{1, \ldots, n\} : w_{ii} = 0$

Zustand: Aktivierung aller Neuronen

Eingabe: Startzustand (initiale Zustände aller Neuronen)

Berechnung: Folge von Schritten (Zustandsübergängen):
Aktualisierung: Berechnung der Aktivierung aller Neuronen,
wiederholen, bis stabiler Zustand erreicht oder Abbruch

Konvergenz : Erreichen eines stabilen Zustandes (ändert sich bei Aktualisierung beliebiger Neuronen nicht)

Ausgabe : Zustand des stabilen Netzes

Aktualisierung in jedem Schritt:

synchron: gleichzeitige Zustandsänderung für alle Neuronen

asynchron: Zustandsänderung eines zufällige gewählten Neurons (faire Auswahl)

Hopfield-Netz – Beispiele

$$W = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

$x = (0,1,0) \mapsto (-1,1,-1) = x'$, $x'W = (0,-2,0) = y' \mapsto (0,0,0) = y$
$x = (0,0,0) \mapsto (-1,-1,-1) = x'$, $x'W = (-2,-2,-2) = y' \mapsto (0,0,0) = y$
gespeicherte (stabile) Muster: $(0,0,0)$ und $(1,1,1)$

$$W = \begin{pmatrix} 0 & -1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 \end{pmatrix} \quad x = (-1,-1,1,1)$$

$$W = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad x = (1,1,1)$$

$w_{ii} \neq 0 \rightarrow$ Oszillation

Hopfield-Netz – Training

direkte Berechnung der Gewichte möglich,
kein Training notwendig

- für ein zu speicherndes Muster $x \in \{-1, 1\}^m$:

$$W = x^T x$$

  mit Modifikation: alle Diagonalelemente 0

- für mehrere zu speichernde Muster
  $x^{(1)} \in \{-1, 1\}^m, \ldots, x^{(k)} \in \{-1, 1\}^m$:

$$W = \sum_{i=1}^{n} \left(x^{(i)}\right)^T x^{(i)}$$

  mit Modifikation: alle Diagonalelemente 0

Beispiel (Tafel):

- ein Muster: $x = (1, -1, 1, 1)$
- mehrere Muster: $x^{(1)} = (-1, 1, -1)$ und $x^{(2)} = (1, -1, 1)$