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

Sommersemester 2019

Was ist Künstliche Intelligenz?

EU-Factsheet on Artificial Intelligence:

> *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 (beschränkte) 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

- Simulation 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)

Logische / regelbasierte 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 10 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
- ...

führte zum aktuellen 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

Schwächen aktueller (statistischer) KI-Systeme

KI derzeit noch weit von menschlichen Fähigkeiten entfernt bzgl.

- Erkennung der eigenen Grenzen
- Intuition
- Aufstellen und Überprüfen sinnvoller Annahmen bei unvollständig vorhandener Information
- Lernen ohne vorheriges Training mit großen Mengen (manuell) annotierter Daten
- Übertragen von Wissen zwischen verschiedenen Anwendungsbereichen
- Kombination verschiedener Methoden
- Schließen bzgl. rechtlicher und moralischer Bezugssysteme, mentaler Modelle

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:

- Suche in Zustandsübergangssystemen (vollständig, heuristisch)
- Regelsysteme
- Logiken (klassische und nichtklassische)
- Logische Programme
- Unvollständiges Wissen (nichtmonotones Schließen)
- Ungenaues Wissen (unscharfes Schließen)
- Entscheidungstabellen, -bäume und -diagramme
- Constraint-Systeme
- statistische Verfahren

Literatur

Folien, Aufgaben, ... zur aktuellen Vorlesung unter
`https://informatik.htwk-leipzig.de/schwarz/lehre/ss19/kib`

Bücher:

- Ingo Boersch, Jochen Heinsohn, Rolf Socher:
  Wissensverarbeitung (Spektrum, 2007)
- Wolfgang Ertel:
  Grundkurs Künstliche Intelligenz (Springer, 2016)
- 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
- wöchentlich eine Übung (2 Gruppen)
- Prüfungsvorleistung: Beleg
  - Präsentation der Lösung der Übungsaufgaben
  - Autotool
- 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$=39.7^\circ$
Wissen	Information mit einem Nutzen, trägt zur Lösung eines Problemes bei, Nutzen abhängig von vorhandenem Kontextwissen im Beispiel: Kontext Körpertemperatur$>39.0^\circ$ ist Fieber, bei Fieber ist Fieberbehandlung notwendig, mögliche Fieberbehandlungen z.B. Wadenwickel, Medikamente
Wissenserwerb	selbständige Informationsgewinnung (auch zum Kontext) im Beispiel über Auslöser, Nebensymptome, Therapien für Körpertemperatur-Unregelmäßigkeiten
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
ermöglicht Anwendung 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.

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.

- Kochrezept
- Euklidischer Algorithmus
- aussagenlogisches Resolutionsverfahren
- Sortierverfahren

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.

ÜA: Missionare + Kannibalen

- Zu Beginn: 3 Missionare + 3 Kannibalen an einem Flussufer
- An diesem Ufer gibt es auch ein Boot, welches höchstens zwei Personen fasst.
- Sobald an einem Ufer die Kannibalen in der Überzahl sind, werden die anwesenden Missionare gefressen.

Aufgabe: Plan zum Übersetzen aller Personen

ÜA: Modellierung des relevanten Wissens

ÜA: Zahlenrätsel

Eine Person A wählt zwei natürliche Zahlen zwischen (einschließlich) 2 und 100 und verrät S deren Summe und P deren Produkt. Dann kommt es zu folgendem Gespräch:

P: Ich kenne die beiden Zahlen nicht.

S: Das weiß ich. Ich kenne sie auch nicht.

P: Dann kenne ich die beiden Zahlen jetzt.

S: Dann kenne ich sie jetzt auch.

Welche Zahlen hat A gewählt ?

ÜA: Modellierung des relevanten Wissens

ÜA: 3 Weise mit Hut

3 von 5 Hüten (2 weiß, 3 rot) werden 3 Weisen aufgesetzt, so dass keiner die Farbe seines Hutes, aber die Farben aller anderen Hüte sehen kann.

Jeder wird der Reihe nach gefragt, ob er die Farbe seines Hutes weiß.

Antworten:

1: Nein

2: Nein

3: Ja

- Warum weiß der dritte Gefragte seine Farbe auf jeden Fall?
- In welchen Fällen kennt der erste Gefragte seine Farbe?
- In welchen Fällen kennt der erste Gefragte seine Farbe nicht, aber der Zweite?

ÜA: Modellierung des relevanten Wissens

Was bisher geschah

- KI-geschichte
- KI-Tests (Turing, Chinese Room)
- schwache / starke KI
- Daten, Information, Wissen, Intelligenz
- 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

Darstellung von Wissen

formale Repräsentation des Wissens in einer Wissensbasis:
spezielle Form der Daten in der Wissensbasis abhängig von

- Problembereich
- geplante Verwendung

Wissen in Wissensbasis ist immer Abstraktion, beschreibt Modelle der Realität

- Auswahl von (für den Anwendungsbereich) wichtigem Wissen
- Vernachlässigung unwichtiger Details

Beispiele:

- Liniennetzplan
- Grundriss
- Stundenplan
- Kostenplan

Wissensverarbeitung

- 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: Fahrzeuge, Fehlfunktionen

teilweise bekannt aus den Modulen

- Modellierung
- Algorithmen und Datenstrukturen

Anforderungen an Wissensbasen

Qualitätskriterien bei der Modellierung:

- für Problembereich geeignete Abstraktion
- effektiv, redundanzfrei
- vollständig
- erweiterbar
- verständlich

Beispiele für Wissensrepräsentation und Problemlösen

Suche / Planen:

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

Lösungsverfahren: Suche (vollständig oder heuristisch)

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)

Programmierung und Wissensverarbeitung

Programmierung	Wissensverarbeitung
Entwurf eines Algorithmus zur Lösung des Problemes	Identifikation des zur Lösung des Problemes relevanten Wissens
Implementierung in einer geeigneten Programmiersprache	Darstellung des relevanten Wissens in einer geeigneten Repräsentationssprache
Problemlösung durch Ausführung des Programmes	Problemlösung durch Anwendung eines Standardverfahrens

Beispiel: n-Damen-Problem

Aufgabe: Setze n Damen ohne gegenseitige Bedrohungen auf ein $n \times n$-Spielfeld

Programmierung	Wissensrepräsentation
Entwurf geigneter Datenstrukturen und eines Algorithmus zur Lösungssuche	Identifikation der Bedingungen an Aufgabe und Lösung
Implementierung	Repräsentation von Spielfeld und Bedingungen an eine Lösung als logische Formeln (z.B. CNF)
Problemlösung durch Ausführung des Programmes	Problemlösung durch logisches Inferenzverfahren (z.B. Resolution, SAT-Solver, Prolog)

Programmierung und Wissensverarbeitung

Programmieren	Wissensverarbeitung

Erklärung der Lösung:

Verfolgen der Zustandsänderung bei Programmausführung (Debugging)	vom Inferenzverfahren verwendete Voraussetzungen

Fehlerbehandlung:

Debugging	fehlendes Wissen einfügen
Codeänderung	falsches Wissen löschen

Wissenserweiterung:

neuer Entwurf, Neuimplementierung	neues Wissen in Wissensbasis einfügen

Wissensverarbeitung

Teilaufgaben:

Repräsentation des Wissens

- geeignete Abstraktionsgrade
- Sprachen zur Wissensrepräsentation
- Modellierung

Verarbeitung des Wissens

- Erweiterung vorhandenen Wissens
- Herleitung neuen Wissens
- Verträglichkeit neuen Wissens mit vorhandenem

Anwendung des Wissens, z.B. zum

- Problemlösen
- Erklären
- Planen
- Klassifikation
- Diagnose
- Entscheidungsunterstützung

Beispiele wissensverarbeitender Systeme

- Expertensysteme
- Diagnosesysteme
- Schach- und andere Spielprogramme
- Datenanalyse
- Suchmaschinen
- Maschinelle Erkennung und Verarbeitung natürlicher Sprache
- Bild- und Zeichenerkennung (Klassifikation)
- Objekterkennung in digitalen Bildern
- Planungssysteme
- Steuerung autonomer Agenten,
  z.B. für Transport, Information, Unterhaltung, Rettung, Putzen

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
    - Suche im Graphen
- 3 Krüge
  - Aufgabe:
    - gegeben: 3 volle Krüge mit Volumen 4l, 7l, 9l,
    - gesucht: genau 6l in einem der 3 Krüge
  - Lösungsidee: Zustände als Knoten eines Suchbaumes

Darstellung von Aufgabe und Lösung

Aufgabe:

gegeben:
- Menge V von Zuständen (evtl. unendlich) oft beschrieben durch Eigenschaften
- Startzustand $s \in V$
- Menge $Z \subseteq V$ von Zielzuständen (oder Eigenschaften der Zielzustände)
- mögliche Übergänge zwischen Zuständen Übergangsrelation $E \subseteq V \times V$

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

Wissensrepräsentation: als Graph $G = (V, E)$ (Zustandsübergangssystem):
- Knotenmenge V: Zustände
- (gerichtete) Kanten: Zustandsübergänge

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

(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
- L_x Menge der expandierten Knoten
- s Startknoten
- φ Anforderungen an Lösung (Zielknoten)

Allgemeiner Suchalgorithmus:

1. $L_a = \{s\}, L_x = \emptyset$
2. solange $\neg\, L_a = \emptyset$:
   - 2.1 Verschiebe einen auf festgelegte Art ausgewählten Knoten u aus L_a in L_x
   - 2.2 Füge alle Nachbarn von u, die nicht in $L_a \cup L_x$ enthalten sind, auf eine festgelegte Art in L_a ein
     (Abbruch falls ein Nachbar v von u die Bedingung φ erfüllt, also eine Lösung repräsentiert)

prominente Spezialfälle:

Tiefensuche
- Verwaltung von L_a als Stack
- Einfügen der Nachbarn an den Anfang der Liste L_a
- festgelegter Knoten wurde zuletzt in L_a eingefügt

Breitensuche
- Verwaltung von L_a als Queue
- Einfügen der Nachbarn an das Ende der Liste L_a

Schrittweise 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)

Bewertungsfunktion 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: $k(u)$

Beispiele:

- Breitensuche (Kosten = Tiefe des aktuellen Knotens u)
- kürzeste Wege (Kosten = minimale bisher bekannte Kosten vom Startknoten zum aktuellen Knoten u)
  Dijkstra-Algorithmus

Was bisher geschah

- Daten, Information, Wissen
- Wissensrepräsentation und -verarbeitung

Wissensrepräsentation: Beschreibung von

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

- Knotenmarkierungen $l_v : V \to L_V$ mit
  L_V: Eigenschaften der Zustände
- Startzustand $s \in V$
- Eigenschaften der Zielzustände
  (z.B. Einschränkung der Variablenwerte)
- Kantenmarkierungen $l_E : V \to L_E$ mit
  L_E: mögliche / zulässige Aktionen (Übergänge)

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

Wissensverarbeitung: Pfadsuche im Graphen

- uninformierte (blinde) Suchverfahren: Tiefen-, Breitensuche
  Schrittweise Vertiefung
- Gleiche-Kosten-Suche

Heuristische Suche – Motivation

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

Anwendung bei

- 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)
- falls vollständige Suche zu aufwendig

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

repräsentiert die Zusatzinformation

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 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?

Beispiel Schiebefax

- Zustände $u \in \{0, \ldots, 8\}^{3\times 3}$, 3×3-Matrix mit Einträgen $\{0, \ldots, 8\}$ (jede Zahl genau einmal, 0 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		4
7	6	5

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

8		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)

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 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:

- vollständig?
- optimal?

Anwendungen

Planungsprobleme und kombinatorische Suchprobleme, z.B.

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

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

Brettspiel

- 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).

(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
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

Idee: Tiefensuche mit Verwaltung zusätzlicher Werte

α : bisher höchster Minimax-Wert an Max-Positionen

β : bisher geringster Minimax-Wert an Min-Positionen

Bei Berechnung des Minimax-Wertes der Wurzel eines Teilbaumes Berechnungen für Enkel auslassen, sobald bekannt ist, dass sie α und β nicht verbessern können

α-β-Pruning: Abtrennen jedes Kindes v eines

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

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

Beispiel (Tafel)

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
- Modallogiken:
  Wissen über verschiedene mögliche Welten (Zustände)
  Wissen über Wissen (z.B. verschiedener Agenten)
- Temporallogiken:
  Wissen über zeitliche Zusammenhänge
  (Zustandsübergangssysteme)
- Beschreibungslogiken:
  Wissen über Begriffe und Zusammenhänge zwischen diesen

Wissensverarbeitung in Logiken

Ziele:

- Beantwortung von Anfragen der Form:
  (Für welche Individuen) Gilt die Aussage ... unter den bekannten Voraussetzungen?
- Herleitung neuen Wissens
- Konsistenztests vorhandenen Wissens
- Konsistentes Zusammenfügen verschiedener Wissensquellen

Methoden:

- Suche nach Modellen
- semantische Methoden:
  semantisches Folgern, Wahrheitswerttabellen, Entscheidungstabellen, Entscheidungsbäume
- syntaktische Methoden:
  Schließen, Ableiten in logischen Kalkülen, Beweisen

Wissensrepräsentation und -verarbeitung in Logiken

Wissensbasis (Aufgabenbereich): Formelmenge Φ

Aufgabe Formel ψ
Fragestellung: Folgt ψ aus Φ?

Lösung
- ja / nein
- falls ja, evtl. Modell

Möglichkeiten zum Ableiten neuen Wissens (Formel) aus einer Wissensbasis (Formelmenge)

Folgern (semantisch), z.B. Wahrheitswerttabellen

Schließen (syntaktisch): Kalküle

Beispiel

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
$\Phi = \{z \wedge \neg t \rightarrow \neg p, z \wedge p\}$

Problem Stand ein Taxi am Bahnhof? $\psi = t$
Folgt ψ aus Φ?

Lösung ...

Wiederholung Aussagenlogik: Syntax

		Syntax	Semantik
		Symbol	Wahrheitswertfunktion
Junktoren	wahr	$\mathbf{t}$	1
	falsch	$\mathbf{f}$	0
	Konjunktion	$\wedge$	min
	Disjunktion	$\vee$	max
	Negation	$\neg$	$x \mapsto 1 - x$
	Implikation	$\rightarrow$	$\leq$
	Äquivalenz	$\leftrightarrow$	$=$

Atome Syntax: elementare Formeln
Semantik: Wahrheitswert

Literal Atom oder negiertes Atom

Klausel Disjunktion von Literalen

Formeln Syntax (induktive Definition): $\mathrm{AL}(P)$
- 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: Boolesche Funktion

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)

Erfüllbarkeit

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

- erfüllbar gdw. $\mathrm{Mod}(\varphi) \neq \emptyset$
- unerfüllbar gdw. $\mathrm{Mod}(\varphi) = \emptyset$
- allgemeingültig gdw. $\mathrm{Mod}(\neg\varphi) = \emptyset$

Erfüllbarkeit ist algorithmisch entscheidbar.

- semantisch z.B. durch Wahrheitswerttabellen
- syntaktisch durch Ableitung in Kalkülen

Werkzeuge: SAT-Solver

Normalformen

Junktorbasen $\{\vee, \wedge, \neg\}, \{\wedge, \neg\}, \{\rightarrow, \neg\}, \{\rightarrow, \mathbf{f}\}, \{\mathsf{NAND}\}$

DNF $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_j} l_{ij}$ mit Literalen
(Atome oder negierte Atome) l_{ij}

CNF $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_j} l_{ij}$ mit Literalen
(Atome oder negierte Atome) l_{ij}

NAND-NF $\neg\varphi \equiv \varphi \,\mathsf{NAND}\, \varphi$
$\varphi \wedge \psi \equiv (\varphi \,\mathsf{NAND}\, \psi) \,\mathsf{NAND} (\varphi \,\mathsf{NAND}\, \psi)$

Semantisches Folgern

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

Kurzform:

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

Beispiele:

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

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.

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. $\Phi \cup \{\neg\psi\} \models \mathbf{f}$

Syntaktisches Schließen

gegeben: Formelmenge Φ
Formel ψ
Frage : Gilt $\Phi \models \psi$?

Ziel: Verfahren zur Beantwortung dieser Frage durch syntaktische Operationen
(ohne Benutzung der Semantik, Modellmengen)

Syntaktische Ableitungsrelation $\vdash \subseteq 2^{\mathsf{AL}(P)} \times \mathsf{AL}(P)$
passend zur
semantischen Folgerungsrelation $\models \subseteq 2^{\mathsf{AL}(P)} \times \mathsf{AL}(P)$

$\vdash$ passt zu $\models$, falls für jede Formelmenge $\Phi \in \mathsf{AL}(P)$ und jede Formel $\psi \in \mathsf{AL}(P)$ gilt

$$\Phi \vdash \psi \quad \text{gdw.} \quad \Phi \models \psi$$

Kalküle

gegeben: aussagenlogische Formel $\varphi \in \mathrm{AL}(P)$
Frage: Ist φ allgemeingültig?

Lösungsansätze:

- φ ist allgemeingültig gdw. $\varphi \equiv \mathbf{t}$.
  Syntaktische (äquivalente) Umformungen von $\mathbf{t}$ zu φ.
  Simulation des mathematischen Schließens (Hilbert-Kalkül)
- φ ist allgemeingültig gdw. $\neg\varphi$ unerfüllbar.
  Erfüllbarkeitstest für $\neg\varphi$, z.B. mit SAT-Solver

Kalkül: Menge von Schlussregeln zur syntaktischen Umformung von Formeln
(Repräsentation von Mengen von Schlussregeln durch Regelschemata)
Kalkül K ist sinnvoll, wenn man zeigen kann:

Korrektheit: Jede in K ableitbare Formel ist allgemeingültig.

Vollständigkeit: Jede allgemeingültige Formel ist in K ableitbar.

Ableitung im Kalkül K

Definition (induktiv)

Die Menge aller im Kalkül K aus einer Formelmenge $\Phi \subseteq \mathrm{AL}(P)$ ableitbaren Formeln ist definiert durch:

1. Alle Formeln in Φ sind in K aus Φ ableitbar.
2. Alle Axiome in K sind in K aus Φ ableitbar.
   (Axiome sind nullstellige Schlussregeln)
3. Sind $\varphi_1, \ldots, \varphi_n$ in K aus Φ ableitbar und ist $(\varphi_1, \ldots, \varphi_n, \psi)$ eine Schlussregel in K, dann ist auch ψ in K aus Φ ableitbar.

Ableitungsbaum (Beweisbaum):
Blätter: Axiome und Voraussetzungen in Φ
innere Knoten: Schlussregeln (Instanzen von Regelschemata)

In K aus $\emptyset$ ableitbare Formeln heißen in K beweisbar.

Beispiel

Im einfachen Kalkül mit Axiomenschema $A \to A$ und Regelschema

$$\frac{A \qquad B}{A \wedge B}$$

sind aus $\Phi = \emptyset$ alle Formeln der Form $\bigwedge_{i=1}^{n}(\varphi_i \to \varphi_i)$ mit $\varphi_i \in \mathrm{AL}(P)$ ableitbar.

Beispiele:

- $(p \to p)$
- $(p \to p) \wedge (q \to q)$
- $((p \to p) \wedge (q \to q)) \wedge ((p \to p) \wedge (q \to q))$

Aussagenlogische Resolution

Formeln $p \vee \psi, \neg p \vee \eta$ haben die Resolvente $\psi \vee \eta$

Satz (Resolutionslemma)

Für jede CNF (Klauselmenge) Φ und die Resolvente R zweier Klauseln aus Φ gilt

$$\mathrm{Mod}(\Phi) = \mathrm{Mod}(\Phi \cup \{R\})$$

Idee: Schrittweise Erweiterung der Formelmenge Φ um Resolventen

Anwendung der Resolutionsregel:

$$\{\psi \vee p, \neg p \vee \eta\} \to \{\psi \vee p, \neg p \vee \eta, \psi \vee \eta\}$$

alternative Darstellung:

$$\{\neg\psi \to p, p \to \eta\} \to \{\neg\psi \to p, p \to \eta, \neg\psi \to \eta\}$$

Spezialfall: endliche Menge Φ von Formeln in CNF

Ableitungen durch Resolution

Resolutionsableitung aus einer Klauselmenge Φ (CNF):
endliche Folge $C_1, \ldots C_n$ von Klauseln, wobei für jede Klausel C_i gilt:

- $C_i \in \Phi$ oder
- C_i ist eine Resolvente von Klauseln C_j, C_k mit $j < i$ und $k < i$.

Resolutionsableitung der Klausel ψ aus Klauselmenge Φ:
Resolutionsableitung $C_1, \ldots C_n$ in Φ mit $C_n = \psi$

Beispiel: Resolutionsableitung von d aus

$$\Phi = \{a \vee b \vee c, \neg b \vee d, \neg a \vee d, \neg c \vee d\}$$

Baumdarstellung (Tafel)

Resolutionsableitungen von $\mathbf{f}$

Problem:
Es existiert keine Resolutionsableitung von $\neg a \vee \neg b \vee d$ aus

$$\Phi = \{a \vee b \vee c, \neg b \vee d, \neg a \vee d, \neg c \vee d\}$$

aber es gilt $\Phi \models \neg a \vee \neg b \vee d$.

Lösungsidee:
Es gilt $\Phi \models \psi$ gdw. $\Phi \cup \{\neg\psi\}$ unerfüllbar.

Unerfüllbarkeitsbeweis für $\Phi \cup \{\psi\}$ durch Resolutionsableitung von $\mathbf{f}$ aus $\Phi \cup \{\neg\psi\}$ (Klauselform)

Beispiel (Tafel): Resolutionsableitung von $\mathbf{f}$ aus

$$\Phi \cup \{\neg\psi\} = \{a \vee b \vee c, \neg b \vee d, \neg a \vee d, \neg c \vee d, a, b, \neg d\}$$

Syntaktische Ableitungsrelation $\vdash_R$

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

$$\Phi \models \psi \quad \text{gdw.} \quad \Phi \cup \{\neg\psi\} \text{ unerfüllbar}$$

Syntaktische Ableitungsrelation $\vdash_R \subseteq 2^{\mathsf{AL}(P)} \times \mathsf{AL}(P)$:
$\Phi \vdash_R \psi$ gdw.
eine Resolutionsableitung für $\mathbf{f}$ aus $\Phi \cup \{\neg\psi\}$ existiert.

Beispiele:

- $\{a \vee b \vee c, (a \vee b) \to d, c \to e, \neg d\} \vdash_R e$
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge p \wedge \neg r$ ist unerfüllbar.
- $\phi = (q \wedge r) \vee (\neg p \wedge \neg q \wedge r) \vee p \vee (\neg p \wedge \neg r)$ ist allgemeingültig.

Korrektheit und Vollständigkeit

Die folgenden beiden Sätze zeigen, dass $\vdash_R$ zu $\models$ passt, d.h. $\Phi \vdash_R \psi$ gdw. $\Phi \models \psi$

Satz (Korrektheit der Ableitungsrelation $\vdash_R$)

Für jede Formelmenge $\Phi \subseteq \mathsf{AL}(P)$ *und jede Formel* $\psi \in \mathsf{AL}(P)$ *gilt:*
Aus $\Phi \vdash_R \psi$ *folgt* $\Phi \models \psi$
(Wenn eine Resolutionsableitung von $\mathbf{f}$ *aus einer zu* $\Phi \cup \{\neg\psi\}$ *äquivalenten Klauselmenge existiert, dann gilt* $\Phi \models \psi$*.)*

Satz (Vollständigkeit der Ableitungsrelation $\vdash_R$)

Für jede Formelmenge $\Phi \subseteq \mathsf{AL}(P)$ *und jede Formel* $\psi \in \mathsf{AL}(P)$ *gilt:*
Aus $\Phi \models \psi$ *folgt* $\Phi \vdash_R \psi$
(Wenn $\Phi \models \psi$ *gilt, dann existiert eine Resolutionsableitung von* $\mathbf{f}$ *aus einer zu* $\Phi \cup \{\neg\psi\}$ *äquivalenten Klauselmenge.)*

Was bisher geschah

Wissensrepräsentation und -verarbeitung in klassischer Aussagenlogik:

- Modellierungsbeispiele
- Wiederholung Syntax:
  Aussagenvariablen, Junktoren, Formeln,
  Atom, Literal, Klausel, Normalformen
- Wiederholung Semantik:
  Belegungen, Modellmengen
  erfüllbar, allgemeingültig
- semantisches Folgern,
- syntaktisches Schließen (Kalküle)
- aussagenlogische Resolution

Modellierungsbeispiel in Aussagenlogik

Geheimnis eines langen Lebens

Wissen Ernährungsregeln:

- Falls ich kein Bier trinke, esse ich Fisch.
- Falls Bier und Fisch zugleich, dann kein Eis.
- Keinen Fisch, falls Eiscreme oder kein Bier.

Darstellung als aussagenlogische Formelmenge mit passenden Aussagenvariablen (Tafel)

Frage Ist Biertrinken notwendig?

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

Lösung ...

Modellierungsbeispiel in Prädikatenlogik (1. Stufe)

Wissensbasis (Aufgabenbereich):

allgemein:

- Personen mit einem gleichen Elternteil sind Geschwister.
- Nichten sind weibliche Kinder von Geschwistern.

speziell:

- Tina ist die Tochter von Anna und Max.
- Paul und Berta sind die Eltern von Anna und Otto.

Formeln ...

Frage Wer ist wessen Nichte?

Lösung ...

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

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

Substitutionen

Substitution: partielle Funktion $\theta : X \to \text{Term}(\Sigma, X)$
Notation als Aufzählung $[x \mapsto t_1, y \mapsto t_2, \ldots]$
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 $(P(b, f(y)) \to Q(x))\theta\sigma = \sigma(\theta(P(b, f(y)) \to Q(x))) =$
  $= P(b, f(f(a))) \to Q(b)$

Wiederholung: Äquivalenzen mit Quantoren

Für alle Formeln $\varphi, \psi \in \mathsf{FOL}(\Sigma, \mathbb{X})$ gilt

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

falls $x \notin \mathsf{fvar}(\psi)$ und $* \in \{\vee, \wedge\}$, gilt außerdem

$$\begin{aligned}
\forall x\varphi * \psi &\equiv \forall x(\varphi * \psi) \\
\exists x\varphi * \psi &\equiv \exists x(\varphi * \psi) \\
\exists y\psi &\equiv \exists x\psi[y \mapsto x] \\
\forall y\psi &\equiv \forall x\psi[y \mapsto x]
\end{aligned}$$

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

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.

Was bisher geschah

Wissensrepräsentation und -verarbeitung in Logiken

- klassische Aussagenlogik
- klassische Prädikatenlogik

- Syntax (Wiederholung)
- Semantik (Wiederholung)
- Substitutionen
- Prädikatenlogische Normalformen
  - bereinigte Form
  - Pränexform

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

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.

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.

Klauselform , wenn $\varphi = \forall x_1 \cdots \forall x_m \psi$, wobei alle x_i verschiedene Variablen und ψ eine CNF sind.
(Darstellung als Regelmenge)

Erfüllbarkeitsäquivalenz

Definition

Zwei Formeln φ, ψ heißen erfüllbarkeitsäquivalent, wenn gilt:
φ ist genau dann erfüllbar, wenn ψ erfüllbar ist.

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

Achtung:

- Erfüllbarkeitsäquivalenz ist schwächer als semantische Äquivalenz.

  Beispiel: $p \vee q$ und r sind erfüllbarkeitsäquivalent,
  aber nicht äquivalent.

- Erfüllbarkeitsäquivalenz bleibt beim Einsetzen in Formeln i.A. nicht erhalten.

  Beispiel: p und q sind erfüllbarkeitsäquivalent,
  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,
(existenzieller Abschluss und schrittweise Eliminierung der $\exists$ von links nach rechts durch Einführung neuer Funktionssymbole)
- eine Erfüllbarkeits-äquivalente Formel (Satz) in Klauselform.

Beispiele:

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

(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 $B = \{$ Anton, Tom, Paul $\}$

Prädikatenlogik: Grundinstanziierung

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)

Was bisher geschah

Wissensrepräsentation und -verarbeitung in Logiken

- klassische Aussagenlogik
- klassische Prädikatenlogik

- Syntax (Wiederholung)
- Substitutionen
- prädikatenlogische Normalformen:
  bereinigte Form, Pränexform, Skolemform, Klauselform
- Semantik (Wiederholung)
- Grundinstanziierung

Prädikatenlogik: Modellierungsbeispiel

Wissen

- Paul liest Kriminalromane.
- Bob liest Zeitungen.
- Tina liest Arztromane.
- Tina mag alle Krimileser.
- Krimileser mögen Zeitungsleser.
- Bob mag jeden, der einen Krimileser mag.
- Zwei Personen sind Freunde, wenn sie einander mögen.
- Tina liest alles, was ihre Freunde lesen.

Darstellung als prädikatenlogische Formelmenge

Fragen

- Mag Tina Bob?
- Wen mag Tina?
- Mag Bob Tina?
- Wer mag (wenigstens einen) Zeitungsleser?
- Wer mag wen?
- Wer ist mit wem befreundet?
- Wer liest was?

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

Lösungen ...

Lösungsidee

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

Nachteil: aufwendig

Idee: benötigte Instanzen nach Bedarf erzeugen

Wiederholung Substitutionen

Substitution: partielle Funktion $\theta : X \to \text{Term}(\Sigma, X)$
übliche Notation: $\theta = \{x \mapsto t_1, y \mapsto t_2, \ldots\}$
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 für jeden Term t_i gilt $\theta(t_i) = \rho(\sigma(t_i))$

(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 (Ü̈A)
- $\mathrm{mgu}(f(x, g(a, z)), f(f(y), f(x))$ existiert nicht (Ü̈A)
- $\mathrm{mgu}(f(x, x), f(y, g(y))$ existiert nicht (Ü̈A)
- $\mathrm{mgu}(f(x, g(y)), f(y, x)$ existiert nicht (Ü̈A)

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

Modellierungsbeispiel

Wissensbasis:

- Otto kann Salsa tanzen.
- Paul kann Tango und Walzer tanzen.
- Maria kann Salsa tanzen.
- Salsatänzer (beliebigen Geschlechts) mögen jeden, der Salsa tanzen kann.
- Anna mag alle, die (irgendeinen Tanz) tanzen können.
- Anna tanzt jeden Tanz, den jemand tanzt, den sie mag.

Fragen:

- Tanzt Anna Tango?
- Wer mag Otto?
- Welche Tänze tanzt Anna?
- Wer tanzt welche Tänze?

Lösungsidee (Wiederholung)

1. Formulierung der Wissensbasis als Formelmenge (Regelmenge)
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

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 Indianer eines Stammes gilt:
Sein Vater trägt eine rote Feder und er selbst trägt keine rote Feder.
$\varphi = \forall x(R(f(x)) \wedge \neg R(x))$ ist unerfüllbar.
Interpretiert in einer Struktur $\mathcal{A} = (A, [\![\cdot]\!]_{\mathcal{A}})$, repräsentiert φ die Formelmenge (Grundinstanziierung)

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

φ repräsentiert die Klauselmenge

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

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

Prädikatenlogische Resolution

(Einfache) Resolutionsregel:
falls $\theta = \text{mgu}(l, l')$ existiert:

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

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 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 $\theta = [x \mapsto a, y \mapsto b, z \mapsto a, w \mapsto b]$

Prädikatenlogische Resolution

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

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 $\Phi \vdash \psi$

Satz
In AL(P) *und* FOL($\Sigma, \mathbb{X}$) *ist das prädikatenlogische Resolutionsverfahren 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 $\mathbf{f}$ 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:
  $\neg b_1 \vee \ldots \vee \neg b_n \vee \equiv \neg b_1 \vee \ldots \vee \neg b_n \vee \mathbf{f} \equiv (b_1 \wedge \ldots \wedge b_n) \rightarrow \mathbf{f}$
  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

Logische Programmierung – Prolog

Regel: 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
Prolog-Syntax: `h :- b1, ..., bn.`

Fakt: Atom h (positives Literal, Regel ohne Rumpf)
Prolog-Syntax: `h.`

Zielklausel (Query): Klausel $\neg b_1 \vee \ldots \vee \neg b_n$ ohne positives Literal (Regel ohne Kopf)
äquivalent: $\neg(b_1 \wedge \ldots \wedge b_n)$
Prolog-Syntax: `?- b1, ..., bn.`

logisches Programm (Wissensbasis):
endliche Menge von (Fakten und) Regeln.

übliche Semantik der klassischen Prädikatenlogik

Logische Programmierung – Beispiel

Wissensbasis: Prolog-Programm P

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

repräsentiert die Formelmenge

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

Aufgabenbeschreibung: Zielklausel (Anfrage, Query)

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

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

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

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

Beispiel als Klauselmenge:

$$\{l(p, k), l(b, z), l(t, a), l(x, k) \rightarrow m(t, x), \neg m(t, y)\}$$

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

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: no

yes 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 repräsentiert die Formelmenge $\Phi \subseteq \mathrm{FOL}(\Sigma, \mathbb{X})$

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

   ($\forall$-Abschlüsse aller Regeln)
   Σ enthält genau alle in P vorkommenen Funktions- und Relationssymbole

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

3. Antwort:

   yes und Belegung $\beta : \mathbb{X} \rightarrow \mathrm{Term}(\Sigma_F, \emptyset)$,
   falls $\Phi \cup \{\beta(\psi)\}$ unerfüllbar
   no , falls $\Phi \cup \{\psi\}$ erfüllbar

Beispiele:

- Programm `a :- b,c. a :- d. b. b :- d. c. d :- e. d.`
  Anfrage `?- a.`
- Programm
  `p(a). p(b). q(a). q(b). r(b). s(X):-p(X),q(X),r(X).`
  Anfrage `?- s(X).`

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 Programmklausel resloviert wird
- Input-Resolution
  Folge von Resolutionsschritten, wobei in jedem Schritt nach einem Literal in der im vorigen Schritt erzeugten Klausel (zu Beginn Zielklausel) resolviert wird
- selection function: Auswahl
  - einer Klausel (unter mehreren passenden):
    von oben nach unten (von links nach rechts)
  - des Literals in der Klausel: von links nach rechts

  Tiefensuche mit Backtracking

Beispiel: $\{a \vee b, \neg a \vee b, a \vee \neg b, \neg a \vee \neg b\}$ unerfüllbar

Was bisher geschah

Wissensrepräsentation und -verarbeitung in Logiken

klassische Aussagenlogik: Syntax, Semantik, Resolution

klassische Prädikatenlogik:

- Wiederholung Syntax, Semantik
- Normalformen: bereinigt, Pränex-, Skolem-, Klausel-
- Substitutionen, Unifikatoren
- allgemeinster Unifikator ($\mathrm{mgu}(s, t)$), 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

Vergleich LP (Prolog) mit FP (Haskell)

Deklarative Programmierung enthält z.B.

- logische Programmierung
- funktionale Programmierung
- Constraint-Programmierung

und Kombinationen dieser Paradigmen

Paradigma	logisch	funktional
Beispielsprache	Prolog	Haskell
Berechnungsmodell	FOL	λ-Kalkül
Programmbaustein	Horn-Klausel	Termgleichung
Ausdrücke	Atome	Terme
Auszuwerten	Konjunktion von Atomen	Term
Finden passender Programmteile	Unifikation	Matching
Auswertung	SLD-Resolution	Reduktion von λ-Ausdrücken, Termersetzung

Beispiel – Peano-Zahlen

Prolog:
```
plus(zero,Y,Y).
plus(succ(X),Y,succ(Z)) :- plus(X,Y,Z).
```

Anfragen, z.B.:

```
?- plus(succ(zero),succ(succ(zero)),X).
?- plus(succ(zero),X,succ(succ(zero))).
?- plus(X,Y,succ(succ(zero))).
```

Haskell:
```
plus Zero y       = y
plus ( Succ x ) y = Succ (plus x y)
```

Anfrage, z.B.:

```
plus (Succ Zero) (Succ (Succ Zero))
```

Prolog:
```
times(zero,Y,zero).
times(succ(X),Y,Z) :- times(X,Y,H), plus(H,Y,Z).
```

Haskell:
```
times Zero y       = Zero
times ( Succ x ) y = plus (times x y) y
```

Logische Programme zur Lösung von Planungs-Aufgaben

generisches Programm:

```
plan(P) :- start(Start),
           action(Start,[Start],Q),
           reverse(Q,P).

action(S,P,P) :- goal(S).
action(S,Visited,P) :- next(S,N),
                       safe(N),
                       no_loop(N,Visited),
                       action(N,[N|Visited],P).

no_loop(S,Visited) :- \+member(S,Visited).
```

für jeweilige Aufgabe zu implementieren:

```
start(...).
goal(...).
next(S,N) :- ...
safe(S) :- ...
```

Beispiele: Münzspiel, Kannibalen, Ziege

Datalog

Datalog: Anfragesprache für relationale Datenbanken
(Tabellen repräsentieren Relationen, definieren 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 sowohl extensional als auch intensional vorkommen.

Datalog: Syntax und Semantik

Syntax: wie Prolog ohne Funktionssymbole mit Stelligkeit > 1

Datalog-Term: Konstantensymbol oder Variable

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

Datalog-Klausel: Regel `h :- b1 , ... , bn.` mit
Datalog-Atomen `b1 , ... , bn , h`
intensionales Prädikat in `h`,
intensionale oder extensionale Prädikate in `bi`

Datalog-Fakt: Datalog-Klausel mit $n = 0$

Datalog-Wissensbasis: endliche Menge von Datalog-Klauseln

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

deklarative Semantik wie in Prolog

Fixpunkt-Semantik über Konsequenzoperator

Beispiel

Aus der Wissensbasis

F1	Tom ist ein Baby. `b(t).`	(extensional)
F2	Tom ist männlich. `m(t).`	(extensional)
R1	Babies sind Kinder. `k(X) :- b(X).`	(intensional)
R2	Männliche Kinder sind Jungen. `j(X) :- k(X),m(X).`	(intensional)
R3	Weibliche Kinder sind Mädchen. `g(X) :- k(X),w(X).`	(intensional)

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

Instanz $b(t) \rightarrow k(t)$ der Regel R1 mit Substitution $x \mapsto t$
„feuert“ in der Faktenmenge $F = \{b(t), m(t)\}$,
weil $b(t) \in F$, also Voraussetzung (Rumpf der Regel R1) in F erfüllt.

schrittweise Erweiterung der Faktenmenge um gültige Fakten:

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

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

Folge $F_0, F_1, \ldots$ von Faktenmengen (Zuständen) $F_i \in 2^{\text{Atom}(P)}$

$$
\begin{aligned}
F_0 &= F \\
\forall i \in \mathbb{N} : F_{i+1} &= \{h \mid ((B \rightarrow h) \in (F \cup R)) \wedge (F_i \models B)\} \\
F^* &= \bigcup_{n \in \mathbb{N}} F_n
\end{aligned}
$$

Sind alle Regeln Hornklauseln, dann ist F^* kleinstes Modell für P.

(datenorientierte Suche)

Konsequenzoperator

gegeben: Wissensbasis (logisches Programm) $P = F \cup R$
P definiert den Konsequenzoperator $T_P : 2^{\text{Atom}(P)} \to 2^{\text{Atom}(P)}$
auf Faktenmegen $M \subseteq \text{Atom}(P)$:

$$\begin{aligned} T_P(M) &= \{h \mid b \to h \in F \cup R \text{ und } M \models b\} \\ &= \{h \mid (b_1 \wedge \cdots \wedge b_n) \to h \in F \cup R \text{ und } \{b_1, \ldots, b_n\} \subseteq M\} \end{aligned}$$

definiert die Mengen

$$\begin{aligned} T_P^0(\emptyset) &= F \\ T_P^{n+1}(\emptyset) &= T_P\left(T_P^n(\emptyset)\right) \\ &\vdots \\ T_P^*(\emptyset) &= \bigcup_{i \in \mathbb{N}} T_P^i(\emptyset) \end{aligned}$$

Falls $T_P^{n+1} = T_P^n$ gilt, dann ist $T_P^* = T_P^n$.
Für endliche Wissensbasen $P = F \cup R$ wird der Fixpunkt $T_P^*(\emptyset)$ nach endlich vielen Anwendungen von T_P erreicht.
Fixpunkt-Semantik für Datalog:
Ein Atom a ist genau dann aus P ableitbar, wenn $a \in T_P^*(\emptyset)$.

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 Wissensbasen $P = (R, F)$ aus definiten Regeln (ohne negative Literale in Rümpfen) ist die Funktion T_P^* ein Hüllenoperator.

Regelbasierte Systeme

Regel : Implikation $r = (\varphi \rightarrow \psi)$

definite Regel (Hornklausel, ohne negative Voraussetzungen):
Implikation $r = (\varphi \rightarrow \psi)$ mit
$\varphi = (b_1 \wedge \ldots \wedge b_n)$ und $\psi = h$
mit (aussagen- oder prädikatenlogischen) Atomen
$b_1, \ldots, b_n, h$

Bestandteile der Regel $r = (\varphi \rightarrow \psi)$:

Kopf ψ (Folgerung)

Rumpf φ (Voraussetzung)
oft $\varphi = (b_1 \wedge \ldots \wedge b_n \wedge \neg c_1 \wedge \ldots \wedge \neg c_m)$ mit
positiven Voraussetzungen $b_1, \ldots, b_n$
und negativen Voraussetzungen $c_1, \ldots, c_m$

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

Konflike bei Aktionsregeln

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

Regel $b_1 \wedge \cdots \wedge b_n \rightarrow h$ in F genau dann anwendbar, 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

Beispiel

Wissensbasis:

R1 Wenn Besuch zum Essen kommt,
gibt es Weiß- oder Rotwein.

R2 Zum Fisch gibt es Weißwein.

R3 Wenn Bob da ist, gibt es Rotwein.

F Bob kommt zum Fischessen.

Frage: Welches Getränk soll serviert werden?

Konfliktlösestrategien

Konfliktlösung durch Auswahl einer Teilmenge aller anwendbaren Regeln, durch deren Anwendung kein Konflikt entsteht.

oft durch (schrittweises) Entfernen ausgewählter Regeln

(einige) Konfliktlösestrategien:

Refraktionsprinzip: Keine Regelinstanz darf zweimal nacheinander feuern

Präferenzen: Bei der Wissensrepräsentation 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

Wissensrepräsentation durch aussagenlogische Regeln

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

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

Anfrage (Fall): Menge von Fakten als

- Formel
- Formelmenge

Lösung: ja / nein , evtl. mit Modell

Problemlöseverfahren: abhängig von der Art der Regeldarstellung

Beispiel

Einlösen eines Schecks unter den folgenden Bedingungen:

- R1 Wenn die vereinbarte Kreditgrenze des Ausstellers eines Schecks überschritten wird, das bisherige Zahlungsverhalten einwandfrei war und der Überschreitungsbetrag kleiner oder gleich 250 € ist, dann wird der Scheck eingelöst.
- R2 Wenn die Kreditgrenze überschritten wird, das bisherige Zahlungsverhalten einwandfrei war, aber der Überschreitungsbetrag über 250 € liegt, dann wird der Scheck eingelöst und dem Kunden werden neue Konditionen vorgelegt.
- R3 Der Scheck wird nicht eingelöst, wenn die Kreditgrenze überschritten wird und das Zahlungsverhalten nicht einwandfrei war.
- R4 Der Scheck wird eingelöst, wenn die Kreditgrenze nicht überschritten wird.

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.

Beispiel Scheckeinlösen (Tafel)

Wissensrepräsentation und -verarbeitung 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: einlösen, nicht einlösen

- 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 mit Alternative

Regeln der Form: 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

$\{\mathrm{ite}, \mathbf{t}, \mathbf{f}\}$ ist eine Junktorbasis.

Zu jeder aussagenlogischen Formel $\varphi \in \mathrm{AL}(P)$ existiert sogar eine äquivalente Formel ψ, die nur die Junktoren $\mathrm{ite}, \mathbf{t}, \mathbf{f}$ 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: $\mathbf{t}$ und $\mathbf{f}$ sind ite-Formeln.

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

Beispiele (Tafel):

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

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.

Wissensrepräsentation und -verarbeitung durch Entscheidungsbäume

Wissensbasis (Kontextwissen):
Entscheidungsbaum (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):
Beginn in der Wurzel,
schrittweises Verfolgen der Kanten im Entscheidungsbaum entsprechend der Merkmalswerte in jedem Knoten
Ablesen der Aussage (oder Aktion) im erreichten Blatt

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)

Was bisher geschah

Wissensrepräsentation und -verarbeitung in

- Zustandsübergangssystemen
- klassischer Aussagenlogik
- Prädikatenlogik und Fragmente,
  z.B. Hornlogik (Prolog)
- Logische Programmen
  Beispiele zum Planen
- Regelsystemen:
  - Wissensrepräsentation: Mengen von Regeln
    z.B. Prolog, Datalog
  - Wissensverarbeitung:
    - Rückwärtsverkettung (Ziel-orientiert)
      z.B. prädikatenlogische Resolution (Prolog)
    - Vorwärtsverkettung (Daten-orientiert)
      z.B. Erweiterungen von Faktenmengen (Datalog)
- Entscheidungstabellen
- Entscheidungsbäume

Beispiel Erreichbarkeit (ÜA)

Wissensbasis:

- Regelmenge:
  - R1 Feldwege sind befahrbar.
  - R2 Landstraßen sind befahrbar.
  - R3 Flüsse sind in Flussrichtung befahrbar.
- Faktenmenge:
  - F1 Feldwege gibt es zwischen A und C und zwischen B und D.
  - F2 Landstraßen gibt es zwischen C und D und zwischen B und E.
  - F3 Flüsse fließen von A nach B und von E nach D.

Frage: Ist D von A erreichbar?

neue Informationen:

- R1' (statt R1): Feldwege sind nur befahrbar, wenn es nicht regnet.
- F4 Es regnet.

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

Allgemeine (normale, definite) logische Programme (Regelmengen) sind endliche Mengen von allgemeinen (normalen, definiten) Regeln.

Wiederholung Konsequenzoperator

gegeben: Wissensbasis (logisches Programm) $P = F \cup R$
Konsequenzoperator $T_P : 2^{\text{Atom}(P)} \to 2^{\text{Atom}(P)}$ auf Faktenmegen $M \subseteq \text{Atom}(P)$:

$$T_P(F) = \{h \mid b \to h \in R \text{ und } F \models b\}$$

für definite Programme (nur positive Bedingungen):

$$T_P(F) = \{h \mid b_1 \wedge \cdots \wedge b_n \to h \in R \text{ und } \{b_1, \ldots, b_n\} \subseteq F\}$$

definiert die Mengen

$$\begin{aligned} T_P^0(\emptyset) &= F_0 = F \\ T_P^{n+1}(\emptyset) &= F_{n+1} = T_P\left(T_P^n(\emptyset)\right) \\ &\vdots \\ T_P^*(\emptyset) &= F_* = \bigcup_{i \in \mathbb{N}} T_P^i(\emptyset) \end{aligned}$$

Fixpunkt-Semantik (definiter) logischer Programme

$T_P^*(\emptyset)$ ist der kleinste Fixpunkt des Operators T_P.

Für definite Programme P:

- gilt $T_P^*(\emptyset) = \bigcap \mathrm{Mod}(P)$
- ist $T_P^*(\emptyset)$ das eindeutige kleinste 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

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

Nichtmonotones Schließen

Syntax der Wissensbasis:
endliche Menge von Regeln mit negierten Atomen im Rumpf (und Kopf)

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.

Konsequenzoperator T_P für nicht definite Programme ist i.A. nicht monoton (d.h. $F \subseteq F' \not\Rightarrow T_P(F) \subseteq T_P(F')$).

Beispiele:

- $P = \{\neg p \to q, \neg q \to p\}$
- $P' = P \cup \{p\}$

Beispiel

Wissensbasis:

F1 mann(Paul).

F2 mann(Otto).

F3 verheiratet(Paul).

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

- Otto-Instanz der Regel R
  mann(Otto) ∧¬ verheiratet(Otto)
  feuert in der Faktenbasis $\{F1, F2, F3\}$
- Paul-Instanz der Regel R
  mann(Paul) ∧¬ verheiratet(Paul)
  feuert in der Faktenbasis $\{F1, F2, F3\}$ wegen F3 nicht

Negative Voraussetzungen

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

verschiedene Ansätze:

1. starke Negation: $\neg p$ gilt genau dann, wenn $(\neg p) \in F$
   Vorteil: positive Antwort immer korrekt
   Probleme:
   - erfordert Verwaltung negativer Fakten in Faktenbasis F
   - Was gilt, falls weder p noch $\neg p$ in F? (Unbestimmtheit)
   - Was gilt, falls sowohl p als auch $\neg p$ in F? (Inkonsistenz)
2. schwache Negation:
   Nicht aus der Wissensbasis ableitbare Aussagen werden als falsch angenommen. (Freispruch aus Mangel an Beweisen)
   Vorteil: ergibt immer eine Antwort (zweiwertig)
   Problem: nach Erweiterung der Wissensbasis evtl. nicht mehr gültig
3. Nutzer fragen (falls möglich)
   Vorteil: Antwort führt zu Erweiterung des Wissens
   keine Antwort vom Nutzer $\rightarrow$ Fall 1 oder 2

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

Fixpunkt-Semantik logischer Programme mit Negation

WH: Für definite Programme P folgt
die Formel φ ($P \models \varphi$) genau dann aus P, wenn $T_P^*(\emptyset) \models \varphi$.
Problem: Für normal logische Programme P hat T_P i.A.
keinen kleinsten Fixpunkt, sondern eine (evtl. leere) Menge
minimaler Fixpunkte (intuitive Modelle).
Beispiel: $P = \{\neg p \to q, \neg q \to p\}$

Zwei prominente intuitive Folgerungsbegriffe aus Menge $M(P)$
mehrerer intuitiver Modelle von P: Formel φ folgt aus P

leichtgläubig (credulous, $P \models_c \varphi$)
gdw. $\exists M \in M(P) : M \models \varphi$
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. $\forall M \in M(P) : M \models \varphi$
mit P aus Beispiel oben:
$P \not\models_s p$, $P \not\models_s q$, aber $P \models_s p \vee q$

Regeln mit Negation im Rumpf

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

- positiven Bedingungen $p_1, \ldots, p_{n_i}$
- negativen Bedingungen $q_1, \ldots, q_{m_i}$

ist in der Faktenmenge F genau dann anwendbar, wenn

$$F \models (p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge \neg q_{m_i})$$

also

- $\{p_1, \ldots, p_{n_i}\} \subseteq F$ und
- $\{q_1, \ldots, q_{m_i}\} \cap F = \emptyset$

Vorwärtsverkettung auch möglich für Wissensbasen mit Regeln mit (schwacher) 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 $I \in \text{Mod}(P)$
(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

Auswahl intuitiver Modelle

für definite logische Programme:

- existiert ein (eindeutiges) kleinstes 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: $P = \{\neg p \to q\}$
intuitiver Folgerungsbegriff $P \models \varphi$ bei mehreren Modellen ?

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.

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,Z), farbe(Y,Z).
```

Stabile Modelle repräsentieren Lösungen (korrekte Färbungen)

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.

Beispiel

Wissensbasis (Problembereich): Baustein-Welt:

- Turm aus drei Bausteinen
  (von oben nach unten: A,B,C)
- A ist grün.
- C ist nicht grün.

Formeln ...

Problem Steht ein grüner Baustein direkt auf einem nicht-grünen?

Lösung ...

Inkonsistentes Wissen

widersprüchliche Aussagen bzw. Aktionen

fast immer unvermeidbar, (einige) mögliche Ursachen:

- ungenaue natürlichsprachliche Formulierungen
- Ausnahmen, Spezialfälle
- ungenaue Modellierung
- Nichtdeterminismus
- Wissen aus mehreren verschiedenen Quellen

Beispiel

Wissensbasis:

- Regelmenge:
  - Vögel können fliegen.
  - Pinguine sind Vögel.
  - Pinguine können nicht fliegen.
- Faktenmenge:
  Tweety ist ein Pinguin.

Problem: Kann Tweety fliegen?

Logische Programme mit negierten Folgerungen

Ziel:

- Darstellung von Regeln mit negierten Folgerungen
- Kombination starker und schwacher Negation

Syntax: erweiterte logische Programme:
im Regelrumpf sowohl starke $^-$ (`~ x`) als auch schwache Negation $\neg$ (`not x`)
im Regelkopf nur starke $^-$ Negation

Semantik: Faktenmengen
(Mengen von Atomen und stark negierten Atomen)

Problem:

- falls ein Atom a mit $\{a, \overline{a}\} \subseteq F$ existiert, ist die Faktenmenge F inkonsistent
(Semantik nicht definiert)
- bei jeder Erweiterung der Faktenmenge ist deren Konsistenz zu garantieren

Erweiterte logische Programme

Idee: p und $\overline{p}$ als unabhängige Atome betrachten
Konsistenz durch Constraints garantieren (z.B. $p \wedge \overline{p} \rightarrow \mathbf{f}$)

(erweitertes) 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: $\{p \rightarrow \overline{q}, \neg\overline{q} \rightarrow r, \neg r \rightarrow \overline{q}, \overline{p}\}$

Semantik: Answer-Sets

Answer-Sets:
ausgewählte Modelle erweiterter logischer Programme

Interpretation eines erweiterten logischen Programmes P:
Menge F von „Grundatomen“
(Grundatom oder stark negiertes Grundatom mit derselben Signatur wie P)

Interpretation F ist Answer-Set für P gdw.

abgeschlossen unter P:
für jede Regelinstanz $B \rightarrow h$ aus P gilt:
falls $F \models B$, dann $h \in F$
und konsistent, d.h.
für kein Atom $p \in F$ gilt $\{p, \overline{p}\} \subseteq F$

begründet: für jedes „Atom“ $\in F$ existiert eine Ableitung (Begründung) für in F

Wissensverarbeitung mit Answer-Sets

Bestimmung von Answer-Sets:
Interpretation F ist Answer-Set für P gdw.
F Modell des F-Reduktes P^F (analog stabilen Modellen)
Beispiel: $P = \{\neg c \to a, \neg b \to c, \neg b \to \overline{d}, a \wedge \neg \overline{b} \to b\}$
$F = \{a, b\}$, $F' = \{c, \overline{d}\}$
ASP-Solver-Notation:

```
a :- not c.
c :- not b.
~d :- not b.
b :- a, not ~b.
```

Wahrheitswert eines Atoms a in Answer-Set F (dreiwertige Logik):

wahr	gdw. $a \in F$
falsch	gdw. $\overline{a} \in F$
undefiniert	gdw. weder $a \in F$ noch $\overline{a} \in F$

Beispiel: in $F' = \{c, \overline{d}\}$ sind a und b undefiniert, c wahr und d falsch

Beispiel: Modellierung von Ausnahmen

Wissensbasis:

- Regelmenge:
  - Vögel können fliegen.
  - Pinguine sind Vögel.
  - Pinguine können nicht fliegen.
- Faktenmenge:
  Tweety ist ein Pinguin.

```
f :- v, not ~f.
v :- p.
~f :- p.
p.
```

hat Answer-Set $\{p, b, \overline{f}\}$

Anwendungsbeispiel: Terminplanung

Faktenbasis (Beschreibung des speziellen Problemes):

```
termin(m1), . . . , termin(mn)
zeit(t1), . . . , zeit(ts), raum(r1), . . . , raum(rm)
person(p1), . . . , person(pk)
mit(p1,m1), . . . , mit(p2,m3), . . .
```

Zuordnung von Zeiten und Räumen zu Terminen:

```
um(M, T)  :- termin(M), zeit(T), not ~um(M, T).
~um(M, T) :-  termin(M), zeit(T), not um(M, T).
in(M,R)   :- termin(M), raum(R), not ~in(M,R).
~in(M,R)  :- termin(M), raum(R), not in(M,R).
zeitvergeben(M) :-  um(M, T).
raumvergeben(M) :-  in(M,R).
```

Bedingungen:

```
:-  termin(M), not zeitvergeben(M).
:-  termin(M), not raumvergeben(M).
:-  termin(M), um(M, T), um(M, T'), T <> T'.
:-  termin(M), in(M,R), in(M,R'), R <> R'.
:-  in(M,X), in(M',X), um(M, T), um(M', T),M <> M'.
:-  mit(P,M), mit(P,M'), M <> M', um(M, T), um(M', T).
```

Softcomputing

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, probabilistische Logik
- 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 jedem Lernschritt die korrekte Ausgabe.

Prinzip: Lernen durch Nachahmen (mit Korrektur)

Anwendung z.B. bei

- Klassizierung (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 jedem Lernschritt, ob die Ausgabe korrekt war.

Idee: Lernen durch Probieren

- 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 ähnliche Muster
- oft auch topologisch sinnvolle Anordnung

Idee: Lernen ohne Trainer (ohne Rückmeldung)

- Entdecken von Strukturen
- Selbstorganisation von Objekten zu Gruppen
  (mit gemeinsamen Merkmalen, typische Vertreter)
- topologieerhaltende Abbildungen
  (z.B. 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 Wissensrepräsentation 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, Donald O. Hebb: Lernmodell
  The organization of behaviour
- 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)
- 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, Faltungsnetze (CNN)
- 2006, long short-term memory (Anwendung)
- 2009, verstärkt Training mit GPUs
- 2017, AlphaZero, ...

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:

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

und

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

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.
  $x_1 \wedge \neg x_2$
  $\neg x_1 \wedge x_2 \wedge x_3$,
  $\neg(x_1 \vee \neg x_2 \vee \neg x_3)$

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 durch relative Hemmung mit großer Schwellwerterhöhung (auf Anzahl aller erregenden Eingänge $+1$) simulieren.
- 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: $\vee, \wedge, \to$, $((x_1 \wedge (x_3 \vee \neg x_2) \vee (\neg x_2 \wedge x_3)$

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: $w_0 = -\theta$

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

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

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 Fehlern (und deren Korrektur)

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,
(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)

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)

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)

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.

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 Mehrschicht-FFN?

Auswahl durch Mehrschicht-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. $x_1 \leq 1$, z_2 feuert gdw. $x_1 \geq 0$
  z_3 feuert 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

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.

Approximation von Funktionen

gegeben: Menge von Trainingspaaren $\{(x^{(1)}, t^{(1)}), \ldots, (x^{(k)}, t^{(k)})\}$
k Stützstellen und Werte an diesen Stützstellen
(z.B. Messwerte)

Ziel:
Konstruktion eines KNN zur Approximation dieser Funktion durch

- lineare Funktionen
- Stufenfunktionen
- komplexere Funktionen

Quadratischer Fehler

Approximation einer Menge von Trainingspaaren
(Funktionswerte an Stützstellen)
durch Funktion gegebenen Typs (z.B. linear)

- Trainingsmenge liefert Stützstellen:

$$(x_{k1}, \ldots, x_{kn}, t_k)_{k \in \{1,\ldots,m\}}$$

- approximierende Funktion $f : \mathbb{R}^n \to \mathbb{R}$
- Fehler an der Stützstelle $(x_{k1}, \ldots, x_{kn})$:

$$t_k - f(x_{k1}, \ldots, x_{kn})$$

- quadratischer Fehler an der Stützstelle $(x_{k1}, \ldots, x_{kn})$:

$$E_k = (t_k - f(x_{k1}, \ldots, x_{kn}))^2$$

- quadratischer Gesamtfehler (Summe über alle Trainingspaare / Stützstellen):

$$E = \sum_{k=1}^{m} (t_k - f(x_{k1}, \ldots, x_{kn}))^2$$

Trainingsziel: Minimierung des quadratischen Fehlers

Beispiel

Bestimmung der Parameter m, n einer Geraden $y = f(x) = mx + n$ aus einer Menge gegebener (ungenauer) Trainingspaare (x, t), z.B.:

$$\{(1, 10), (2, 7), (4, 5), (5, 1)\}$$

(ganz einfaches) Ein-Schicht-FFN:

- ein Eingang x_1, ein Bias-Neuron x_0
- ein Ausgangsneuron y
- Gewichte: $w_0 = n, w_1 = m$

Funktionen des Ausgabeneurons y:

- Eingangsfunktion I: gewichtete Summe $nx_0 + mx_1 = mx_1 + n$
- Aktivierungsfunktion A: Identität (linear)
- Ausgangsfunktion O: Identität

Dieses Netz berechnet die Funktion

$$f(x) = O(A(I(x_1))) = I(x_1) = mx_1 + n$$

Ermittlung der Parameter m, n durch Training des Netzes (Δ-Regel)

Methode der kleinsten Quadrate

direkte Berechnung mit Hilfe der partiellen Ableitungen nach m und n

$$E = \sum_{k=1}^{l} (t_k - f(x_k))^2 = \sum_{k=1}^{l} (t_k - mx_k - n)^2$$

partielle Ableitungen nach m und n:

$$\begin{aligned}
\frac{\partial E}{\partial m} &= \sum_{k=1}^{l} (-2)\,(t_k - mx_k - n)\,x_k \\
&= -2\left(\sum_{k=1}^{l} t_k x_k - m\sum_{k=1}^{l} x_k^2 - n\sum_{k=1}^{l} x_k\right) \\
\frac{\partial E}{\partial n} &= \sum_{k=1}^{l} (-2)\,(t_k - mx_k - n) \\
&= -2\left(\sum_{k=1}^{l} t_k - m\sum_{k=1}^{l} x_k - nl\right)
\end{aligned}$$

Bestimmung der Parameter

Im Minimum von f sind alle partiellen Ableitungen 0.
Das ergibt ein lineares Gleichungssystem für m und n:

$$\begin{aligned} \sum_{k=1}^{l} t_k x_k - m \sum_{k=1}^{l} x_k^2 - n \sum_{k=1}^{l} x_k &= 0 \\ \sum_{k=1}^{l} t_k - m \sum_{k=1}^{l} x_k - ln &= 0 \end{aligned}$$

mit den Lösungen

$$\begin{aligned} n &= \frac{\sum_{k=1}^{l} t_k - m \sum_{k=1}^{l} x_k}{l} \\ m &= \frac{l \sum_{k=1}^{l} t_k x_k - \left(\sum_{k=1}^{l} t_k\right)\left(\sum_{k=1}^{l} x_k\right)}{\sum_{k=1}^{l} x_k^2 - \left(\sum_{k=1}^{l} x_k\right)^2} \end{aligned}$$

im Beispiel $m = -2, n = 47/4$

Berechnung der Gewichts-Verschiebungen

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

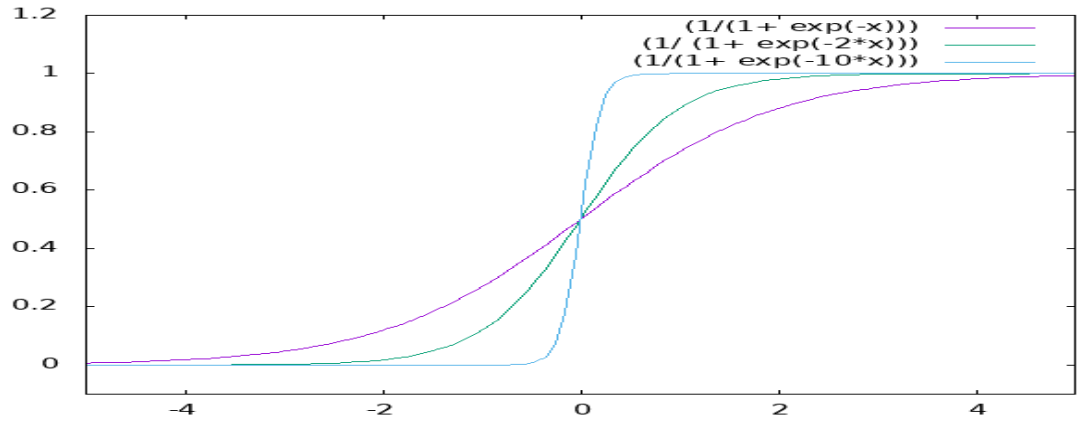
zur Anwendung in KNN: differenzierbare Aktivierungsfunktion

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:

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

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 nicht relevant)

Beispiel

(ganz einfaches) Ein-Schicht-FF-Netz: ein Neuron mit

- einem Eingang $x \in \mathbb{R}$,
- einem Gewicht $w \in \mathbb{R}$,
- Eingabefunktion $I(x) = wx$ (gewichtete Summe)
- verschiedene Aktivierungsfunktionen $A : \mathbb{R} \to \mathbb{R}$
- Ausgabefunktion: $O(x) = x$

berechnet eine Funktion $f : \mathbb{R} \to \mathbb{R}$ mit

$$y = f(x) = O(A(I(x))) = A(wx)$$

quadratischer Fehler für ein Trainingspaar (x, t):

$$E(w) = (t - y)^2 = (t - f(x))^2 = (t - A(wx))^2$$

Ableitung der Fehlerfunktion nach dem Eingangsgewicht w:

$$\frac{\partial E(w)}{\partial w} = E'(w) = 2(t - A(wx))A'(wx) = 2(t - A(wx))xA'(w)$$

Beispiel mit identischer Aktivierungsfunktion

$$y = f(x) = O(A(I(x))) = A(wx) = wx$$

quadratischer Fehler:

$$E(w) = (t - y)^2 = (t - A(wx))^2 = (t - wx)^2$$

Ableitung nach w:

$$\frac{\partial E(w)}{\partial w} = -2(t - wx)x = -2(t - y)x$$

Gewichtsänderung:

$$\Delta w = -\eta' \frac{\partial E(w)}{\partial w} = \eta(t - y)x \quad (\Delta\text{-Regel})$$

Beispiel mit sigmoider Aktivierungsfunktion

$$y = f(x) = O(A(I(x))) = A(wx) = \frac{1}{1 + e^{-wx}}$$

quadratischer Fehler:

$$E(w) = (t - y)^2 = (t - A(wx))^2 = \left(t - \frac{1}{1 + e^{-wx}}\right)^2$$

Ableitung nach w:

$$\frac{\partial E(w)}{\partial w} = -2(t - A(wx))A'(wx) = -2(t - y)y(1 - y)x$$

Gewichtsänderung:

$$\Delta w = -\eta \frac{\partial E(w)}{\partial w} = \eta(t - y)y(1 - y)x$$

(Backpropagation-Regel für die Ausgabeschicht)

Allgemeines Ein-Schicht-FF-Netz

Ein-Schicht-FF-Netz mit

- Eingängen $x \in \mathbb{R}^m$,
- Ausgängen $y \in \mathbb{R}^n$,
- Gewichtsmatrix $W \in \mathbb{R}^{m \times n}$
  (Gewicht w_{ij} zwischen Eingang i und Ausgang j),
- Eingangsfunktion $I(x) = \sum_{i=1}^{m} x_i w_{ij}$
  (gewichtete Summe der Eingänge am Neuron j, Skalarprodukt von x mit Spalte j der Gewichtsmatrix W)
- Ausgangsfunktion $O(x) = x$ (Identität)

berechnet eine Funktion $f : \mathbb{R}^m \to \mathbb{R}^n$ mit

$$y_j = f(x_1, \ldots, x_m) = O(A(I(x_1, \ldots, x_m))) = A\left(\sum_{i=1}^{m} x_i w_{ij}\right)$$

quadratischer Fehler für ein Trainingspaar $(x, t) \in \mathbb{R}^m \times \mathbb{R}^n$:

$$E = \sum_{j=1}^{n} (t_j - y_j)^2 = \sum_{j=1}^{n} \left(t_j - A\left(\sum_{i=1}^{m} x_i w_{ij}\right)\right)^2$$

Gewichtsänderungen

quadratischer Fehler für ein Trainingspaar $(x, t) \in \mathbb{R}^m \times \mathbb{R}^n$:

$$E = \sum_{j=1}^{n} (t_j - y_j)^2 = \sum_{j=1}^{n} \left(t_j - A \left(\sum_{i=1}^{m} x_i w_{ij} \right) \right)^2$$

Ableitung nach w_{ij}:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial A(I(x_1, \ldots, x_m))}{\partial I(x_1, \ldots, x_m)} \frac{\partial I(x_1, \ldots, x_m)}{\partial w_{ij}} \\ &= (t_j - y_j) \frac{\partial A(I(x_1, \ldots, x_m))}{\partial I(x_1, \ldots, x_m)} x_i \end{aligned}$$

Gewichtsänderungen:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta (t_j - y_j) \frac{\partial A \left(\sum_{i=1}^{m} x_i w_{ij} \right)}{\partial w_{ij}}$$

Beispiele

identische Aktivierung $A(x) = x$

$$\frac{\partial A\left(\sum_{i=1}^{m} x_i w_{ij}\right)}{\partial w_{ij}} = \frac{\partial \sum_{i=1}^{m} x_i w_{ij}}{\partial w_{ij}} = x_i$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta(t_j - y_j)x_i \quad \text{(Delta-Regel)}$$

sigmoide Aktivierung $A(x) = \frac{1}{1+e^{-x}}$

$$\frac{\partial A\left(\sum_{i=1}^{m} x_i w_{ij}\right)}{\partial w_{ij}} = y_j(1 - y_j)x_i$$

$$\Delta w_{ij} = -\eta' \frac{\partial E}{\partial w_{ij}} = \eta(t_j - y_j)y_j(1 - y_j)x_i$$

Mehrschicht-FFN

- Eingabeschicht x
- versteckte Schichten $z^{(1)}, \ldots, z^{(n)}$
- Ausgabeschicht y

gewichtete Verbindungen zwischen

- x und $z^{(1)}$
- für alle $i \in \{0, \ldots, n_i\}$ zwischen $z^{(i)}$ und $z^{(i+1)}$
- $z^{(n)}$ und y

Darstellung der Gewichte zwischen benachbarten Schichten als Matrizen
(nur relevante Blöcke der gesamten Gewichtsmatrix)

Backpropagation in FFN

(Bryson, Ho 1969, Rummelhard, McClelland 1986)

Ziel: Geeignete Modifikation aller Gewichte im FFN zur Verrringerung des Gesamtfehlers

Idee:

- 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 $\Delta w_{uv} = -\eta o_u \delta_v$

Zwei-Schicht-Feed-Forward-Netz – Beispiel

(ganz einfaches) Zwei-Schicht-Feed-Forward-Netz:

- Eingabe: ein Neuron x
  keine gewichteten Eingänge
  Eingangs-, Aktivierungs- und Ausgangsfunktion: Identität
- versteckte Schicht: ein Neuron h
  ein gewichteter Eingang (von x, Gewicht w_{xh})
  Eingangsfunktion: gewichtete Summe, hier nur $w_{xh}x$
  Aktivierungsfunktion: sigmoid $A_h(v) = \frac{1}{1+e^{-v}}$
  Ausgangsfunktion: Identität
- Ausgabe: ein Neuron y
  ein gewichteter Eingang (von h, Gewicht w_{hy})
  Eingangsfunktion: gewichtete Summe, hier nur $w_{hy}h$
  Aktivierungsfunktion: sigmoid
  Ausgangsfunktion: Identität

Netz berechnet die Funktion $f : \mathbb{R} \to \mathbb{R}$ mit

$$f(x) = f_y(f_h(x)) = O_y(A_y(I_y(O_h(A_h(I_h(x))) = A_y(w_{hy}A_h(w_{xh}x))$$

(Verkettung von Funktionen)

Backpropagation-Methode – Beispiel

Backpropagation-Schritte für ein Trainingspaar (x, t):

1. Vorwärts-Schritt: Funktionskomposition
   schichtweise Berechnung der Neuronen-Ein- und -Ausgaben
   - Berechnung der Ein- und Ausgaben jedes Neurons aus der Eingabe x
     $o_h = O_h(A_h(I_h(x))) = \frac{1}{1+e^{-w_{xh}x}}$,
     $o_y = O_y(A_y(I_y(h))) = \frac{1}{1+e^{-w_{hy}o_h}}$
   - Berechnung der Netzausgabe $y = o_y$
   - Berechnung des Fehlers $E = (y - t)^2$
2. Rückwärts-Schritt: Multiplikation
   schichtweise Berechnung der anteiligen Fehler δ_h, δ_y nach Gradientenabstiegsverfahren
   - Ausgabeschicht y:
     $\delta_y = -\frac{\partial E}{\partial A_y} = (t - o_y)A'_y = (t - o_y)o_y(1 - o_y)$
   - versteckte Schicht h: $\delta_h = \delta_y w_{hy} o_h(1 - o_h)$
3. Aktualisierung der Gewichte
   $\Delta w_{xh} = \eta\delta_h x$, $\Delta w_{hy} = \eta\delta_y o_y$

Allgemeine Mehr-Schicht-Feed-Forward-Netze

FFN mit k Schichten $s \in \{0, \ldots, k\}$ zu je n_s Neuronen und Gewichten $w_{ij}^{(s)}$ zwischen Ausgang des Neurons i der Schicht $s-1$ und Eingang des Neurons j der Schicht s
k Gewichtsmatrizen $W^s \in \mathbb{R}^{n_{s-1}} \times \mathbb{R}^{n_s}$

Verallgemeinerung der Backpropagation-Methode auf

- Parallelität (mehrere Neuronen je Schicht)
  - Vorwärts-Schritt: Addition mehrerer Eingaben
  - Rückwärts-Schritt: partielle Ableitungen
- Kantengewichte: Multiplikation (beide Richtungen)
- mehrere versteckte Schichten:
  mehrere Vorwärts- und Rückwärtsschritte

Backpropagation-Lernen allgemein

- 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

Anwendung von FFN mit Backpropagation

KNN zur Muster-Klassifikation

Klassifikation von Eingabemustern, z.B.

- optische Zeichenerkennung
  (z.B. Buchstaben, abstrahiert von Schriftart)
- Erkennung akustischer Signale (z.B. Stimmen)
- englische Ausspracheregeln (NETTALK)
- Datenkompression (Eingabe = Ausgabe, Code in der versteckten Schicht)
- Vertrauenswürdigkeit von Bankkunden (Risikoklassen)
- Vorhersage (Wetter, Aktienkurse)
- bisher: Boolesche Funktionen
  (Klassifikation von Eingabevektoren nach Ausgabe-Wahrheitswerten)

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“

Was bisher geschah

- Maschinelles Lernen
  - überwacht
    - korrigierend
    - bestärkend
  - unüberwacht
- Künstliche Neuronale Netze
  - biologisches Vorbild neuronaler Netze und Lernvorgänge darin
  - künstliche Neuronen
    z.B. McCulloch-Pitts-Neuron, Schwellwertneuron
  - Eingangs-, Aktivierungs-, Ausgangsfunktion
  - Lernen künstlicher Neuronen (Δ-Regel, überwacht)
  - Feed-Forward-Netze
    gerichteter Graph mit Kantengewichten (Matrix)
    (parallele und sequentielle Berechnung)
  - Verwendung künstlicher neuronaler Netze:

WH: Feed-Forward-Netze

Topologie: gerichteter Graph,
Knoten (Neuronen) in einer oder mehrere Schichten $\{0, \ldots, k\}$,
Kanten je nur von Knoten in Schicht i zu Knoten in Schicht $i + 1$ (mitunter auch $j > i$)
Eingabe: Eingänge der Neuronen in Schicht 0
Ausgabe: Ausgänge der Neuronen in Schicht k

Aktivierung: lineare, Stufen-, sigmoide Funktionen

Lernen:
- Ein-Schicht-FFN: Δ-Regel
- Mehr-Schicht-FFN: Backpropagation (Gradientenabstieg der Fehlerfunktion)

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 $r : \mathbb{R}_{\geq 0} \to [0, 1]$ mit
  - $r(0) = 1$
  - monoton fallend: $\forall x, y \in \mathbb{R}_{\geq 0} : (x < y) \to (r(x) < r(y))$

  z.B. d Manhattan-Metrik, $r(x) = \max(0, 1 - x)$
- 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)

Hybride Netze

Idee: Kombination verschiedenartiger Neuronen

Beispiel RBF-Netz (ca. 1989):
schichtweise verschiedene Aktivierungsfunktionen

Topologie: 2-Schicht-FFN
Eingabeschicht 0
RBF-Schicht 1 aus RBF-Neuronen
Ausgabeschicht 2 aus Schwellwertneuronen

Aktivierung abhängig von Schicht

1. RBF-Neuronen: radiale Funktion
2. Ausgabeneuronen: Stufen-, sigmoide Funktion

Lernen:

- RBF-Schicht: Clustering-Verfahren
- Schwellwert-Schicht: Δ-Regel

RBF-Netze: Beispiel

2-2-1-Netz für $\leftrightarrow$:

Idee: $x_1 \leftrightarrow x_2 \equiv (x_1 \wedge x_2) \vee \neg(x_1 \vee x_2)$

- erste Schicht (RBF): $w_{1,h1} = w_{2,h1} = 1$, $w_{1,h2} = w_{2,h2} = 0$,
  Eingabefunktion: Euklidische Metrik
  $d((x_1, x_2), (w_1, w_2)) = \sqrt{(w_1 - x_1)^2 + (w_2 - x_2)^2}$
  radiale Funktion $r_{h1}(x) = r_{h2}(x) = \max(0, x/2))$
  Aktivierung: Stufenfunktion
- zweite Schicht: Gewichte $w_{h1,y} = w_{h2,y} = 1$,
  Eingabefunktion: gewichtete Summe
  Aktivierung: linear
  Schwellwert $\theta_y = 0$

RBF-Netze zur Approximation von Funktionen $f : \mathbb{R} \to \mathbb{R}$ durch Linearkombination (gewichtete Summe) von radialen Funktionen, z.B.

- stückweise konstante Funktionen (Stufen)
- stückweise lineare Funktionen
- Gauß-Funktionen

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 $\theta_u = 1$, $\theta_v = 2$

Satz: Zu jedem NFA existiert ein rekurrentes Netz mit McCulloch-Pitts-Neuronen, welches dieselben Zustandsübergänge simuliert.

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

$$A(x) = \operatorname{sgn}(x) \begin{cases} -1 & \text{falls } x < 0 \\ 0 & \text{falls } x = 0 \\ 1 & \text{sonst} \end{cases}$$

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 (Hebb)

biologisches Vorbild: Synapsen zwischen gleichzeitig aktiven Neuronen (x_i und y) werden verstärkt (synaptische Plastizität)

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

(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

$$W = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \quad x = (1, -1, 1)$$

$$W = \begin{pmatrix} 0 & -1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 \end{pmatrix} \quad x = (-1, -1, 1, 1)$$

$$W = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad x = (1, 1, 1)$$

$w_{ii} \neq 0 \rightarrow$ 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 $x^{(2)} = (1, -1, 1)$

Unüberwachtes Lernen

bei unbekannter Zielfunktion

zur Analyse von Datenmengen

Ziel: Gruppierung ähnlicher Daten (Clustering)
z. B. erste Schicht in RBF-Netzen

Generalisierung durch Einordnung neuer Daten in vorhandene Gruppen (Cluster)

Training durch Menge von Trainingmustern
$T = \{x^i \mid i \in \{1, \ldots, k\} \wedge x^i \in \mathbb{R}^m\}$

Methode: Wettbewerbslernen (competitive learning)
nur „Gewinner“-Neuron feuert

Bündeln von Mustern (Clustering)

Eingabe: Menge von Trainingsmustern $\{x^{(i)}, \ldots, x^{(m)}\}$

Ziel: Gruppierung ähnlicher Muster

Anordnung von Mustern in Bündeln:

- Ähnlichkeit aller Muster eines Clusters
- Trennung von Mustern mit wenig Gemeinsamkeiten

Ähnlichkeit im $\mathbb{R}^n$

Zwei Punkte $x, y \in \mathbb{R}^n$ sind einander ähnlich, falls sie einen geringen Abstand voneinander haben

Beispiele für Abstandsfunktionen: siehe RBF-Netze

statt Euklidischem Abstand

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

einfachere Berechnung
quadrierter Euklidischer Abstand

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2$$

Zuordnung von Mustern zu Bündeln

Eingaben:

- Menge $\{1, \ldots, k\}$ von Bündeln,
- Muster x

x wird dem Bündel $j \in \{1, \ldots, k\}$ zugeordnet, von welchem es den geringsten Abstand hat

Idee: Jedes Bündel $j \in \{1, \ldots, k\}$ hat ein Zentrum p_j
(durchschnittliche Position aller Punkte im Bündel, Prototyp)

neues Muster x wird dem Bündel j genau dann zugeordnet, wenn der Abstand von x zum Zentrum p_j des Bündels j minimal ist, d.h.

$$\text{gdw. } \forall i \in \{1, \ldots, k\} : d(x, p_j) \leq d(x, p_i)$$

geometrisch: Zerlegung des R^n in Gebiete
z.B. im $\mathbb{R}^2$: Voronoi-Diagramme

Selbstorganisierende Karte (SOM)

(Teuvo Kohonen, ca. 1980)

Netz-Topologie: Ein-Schicht-Feed-Forward-Netz mit

- Eingabe $(x_1, \ldots, x_m) \in \mathbb{R}^m$, interpretiert als Punkt im m-dimensionalen Eingaberaum $\mathbb{R}^m$
- Ausgabeneuronen $(p_1, \ldots, p_n)$(Bündelneuronen) interpretiert als n Punkte $p_i = (w_{1i}, \ldots, w_{mi})$ im $\mathbb{R}^m$, Zentren der Bündel
- vollständig verbunden (alle Vorwärtskanten), Eingabegewichte zum Neuron j: $(w_{1j}, \ldots, w_{mj})$ interpretiert als Koordinaten des Zentrums des Bündels j

Selbstorganisierende Karte – Funktionen

Funktionen des Ausgabeneurons j:

- Eingabefunktion: Distanzfunktion (Metrik)
  z. B. quadrierter Euklidischer Abstand:

$$I_j(x_1, \ldots, x_m) = \sum_{i=1}^{m} (w_{ij} - x_i)^2$$

- Aktivierungsfunktion
  (Auswahl des Neurons mit dem kleinsten Abstand von x)

$$A_j(I_j(x)) = \begin{cases} 1 & \text{falls } \forall l \in \{1, \ldots, n\} : I_j(x) \leq I_l(x) \\ 0 & \text{sonst} \end{cases}$$

  Wettbewerbslernen: genau ein Ausgabeneuron feuert

- Ausgabefunktion: Identität

Selbstorganisierende Karte – Lernen

Beginn: zufällige Eingangsgewichte der Ausgabeneuronen
(Anordnung der Bündelzentren)

Trainieren der Gewichte
für jede Eingabe $x = (x_1, \ldots, x_m)$:

- Eingabe x
- Anpassung der Eingangsgewichte des aktivierten Bündelneurons j
  (Verschiebung des Bündelzentrums p_j in Richtung des aktuellen Eingabevektors x)

$$w'_{ij} = w_{ij} + \eta(x_i - w_{ij})$$

Topologie-erhaltende SOM

Idee: Topologie-erhaltende Abbildung in andere (geringere) Dimension

Anordnung der Bündelneuronen (z.B. linear, eben, räumlich)

Blockparty:
Berücksichtigung der Nachbarschaft (räumlichen Beziehungen) zwischen den Bündelneuronen (in der definierten Anordnung) bei der Aktualisierung der Positionen
z.B. Graph, Gitter, Abstandsfunktionen

Training Topologie-erhaltender SOM

- Start mit zufälligen Gewichten (Bündelzentren)
- Anpassung der Eingangsgewichte des aktivierten Bündelneurons p_j (in Richtung des Eingabevektors)
- Anpassung der Eingangsgewichte aller Bündelneuronen p_k in einer Nachbarschaft des aktivierten Bündelneurons p_j

$$w'_{ik} = w_{ik} + n(p_k, p_j)\eta(x_i - w_{ik})$$

mit Nachbarschaftsfunktion $n(p_k, p_j)$
(Einfluss fällt mit wachsendem Abstand auf 0, z.B. RBF)

SOM – Trainingsverlauf

Idee:
Anpassung (Verringerung) von Lernrate und Radius während der Lernphase

Heuristik (sinnvoller Trainingsverlauf):

- Start mit
  - großem Radius (nahe halbem Kartenradius)
  - großer Lernrate (nahe 1)

  unverändert über ca. 1000 Iterationen
  Ordnungsphase
- Nachbarschafts-Radius (Einflussbereich jedes Bündelneurons) und Lernrate
  werden mit der Zeit verringert (über ca. 10 000 Iterationen)
  Feinabstimmung

SOM – Beispiele

Topologie-erhaltende Abbildung zwischen Räumen (evtl. verschiedener Dimensionen)

Beispiele:

- Projektion Kugel $\rightarrow$ Ebene
  (Robinson-Projektion: Erde $\rightarrow$ Weltkarte)
- Sensorische Karten:
  Abbildung von Reizen auf benachbarten Bereichen der Haut auf benachbarte Bereiche im Gehirn
- Abbildung von akustischen Reizen benachbarter Frequenzen auf benachbarte Bereiche im Gehirn

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

Bild-Pyramiden

Features:

- Flächen gleicher Farbe
- Kanten
- Formen
- Texturen, ...

Idee aus DBV:
Bilder enthalten Informationen auf verschiedenen Ebenen,
kleinteilige Beobachtung lenkt evtl. von wesentlichen Merkmalen ab
Umsetzung durch Multiskalen-Bilder (Pyramiden)
entstehen durch mehrfache Wiederholung von

- Glättung (durch geeignete Filter)
- Komprimierung durch geringere Abtastrate,
  z.B. Gauß-Pyramide: Löschen jeder zweiten Zeile und Spalte

Umsetzung als KNN (feed-forward)

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

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

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

KNN zur Klassifikation

z.B. von handgeschriebenen Ziffern

- Linearer Klassifikator (keine versteckte Schicht)
- Ein-Schicht-FFN (eine versteckte Schicht)
- Mehr-Schicht-FFN (mehrere versteckte Schichten)
- CNN LeNet-1 (1989)
- CNN LeNet-5
- ...

CNN – interessante 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
  `https://richzhang.github.io/colorization` (2016)
  Eingabe: Grauwertbild
  Ausgabe: Farbbild
- 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)

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

Modul Grundlagen der Künstlichen Intelligenz – SS 2019

Die Studierenden kennen Grundlagen und praktische Anwendungen der Wissensverarbeitung und der Künstlichen Intelligenz.
Sie können basierend auf den Kenntnissen zu ausgewählten Formen der Darstellung von Wissen und zu Problemlösungsverfahren einfache Probleme aus dem Bereich der KI analysieren und lösen.

Lehrinhalte im Sommersemester 2019

- Einteilung symbolische / statistische KI
- Zustandsübergangssysteme, Planen (symb)
- Heuristische Suche (symb /stat)
- logische Programmierung (symb)
- Modellierung unvollständigen Wissens (symb)
- Künstliche neuronale Netze (stat)

Organisatorisches

- Prüfung (laut Modulbeschreibung: Klausur 90 min)
  am Freitag, dem 26.07.2019 um 9:00-10:30 in LNW006
  (gemeinsam mit KI für INM)
- Inhalt:
  - Vorlesungsinhalt
  - Aufgabentypen wie Übungsaufgaben
- Prüfungsvorleistung Beleg (PVB):
  ≥ 3 Punkte für Vorrechenen in Übungen
- (ausschließlich) zulässiges Hilfsmittel:
  A4-Blatt (beidseitig) handbeschrieben
- Notenbonus für alle LegoProI-Teilnehmer:
  0.3 bei bestandener Klausur

Wissen, Information, Daten

Umwelt		Eindrücke, Reize
System	Wahrnehmen, Beobachten	Daten
	Erkennen, Verstehen	Information
	Anwenden, Können, Lernen	Wissen
	Reflektieren, Verstehen	Intelligenz

Wissen, Information, Daten

Informatik Lehre von Darstellung und Verarbeitung von Information

Daten Darstellungsform (Syntax)
Zeichenketten, Bilder, Ton, ...

Information Bedeutung der Daten (Semantik)
in einem bestimmten Kontext

Wissen Information mit einem Nutzen,
trägt zur Lösung eines Problemes bei,
Nutzen abhängig von vorhandenem Kontextwissen

Wissensverarbeitung

Verarbeitung von Wissen:

- Speichern und Abrufen
- Herleitung neuen Wissens
- sinnvolle Verwaltung von Erweiterungen

grundlegende Voraussetzung:
sinnvoller Formalismus zur (maschinell lesbaren) Repräsentation von Wissen

Wissensrepräsentation und -verarbeitung

Repräsentation	Verarbeitung
Aussagenlogik	Resolution
Prädikatenlogik	Resolution
logisches Programm (Datalog)	Prolog-Interpreter
logisches Programm mit Negation (Datalog, ASP)	Konsequenzoperator ASP-Solver
Zustandsgraph	Suche (vollständig oder heuristisch)
KNN (gespeicherte Funktion)	Abfrage nach Training
kreative Leistung	Standardverfahren

Zustandsübergangssysteme: Wissensrepräsentation

Formalisierung des Wissens:
Problembeschreibung (Modellierung) durch
Zustandsgraphen:

- Knoten: Zustände
- Kanten: zulässige Übergänge zwischen Zuständen
- ausgezeichnete Startzustände
- Eigenschaften der Zielzustände

mögliche gewünschte Lösung:

- ein Zielzustand (bei definierten Zielbedingungen)
- alle Zielzustände
- Zielzustände mit guter Bewertung (Zusatzinformation)
- Pfade zu Zielzuständen (Pläne, Strategien)

Anwendungen, z.B.

- Verifikation
- kombinatorische Suchprobleme
- Planen
- Spielsituationen

Zustandsübergangssysteme: Wissensverarbeitung

Suchverfahren in Zustandsgraphen:

Tiefensuche (uninformiert)
Verwaltung entdeckter, aber noch nicht abgearbeiteter Knoten in Stack

Breitensuche (uninformiert)
Verwaltung entdeckter, aber noch nicht abgearbeiteter Knoten in Queue

heuristische Suche
zur Suche von Lösungen mit zusätzlichen (Optimalitäts-)Eigenschaften oder zur Beschleunigung der Suche
Verwaltung entdeckter, aber noch nicht abgearbeiteter Knoten in Priority-Queue mit geeignet gewählten Prioritäten

Spielbaum-Suche (Zwei-Personen-Spiele)
Minimax-Suche, α-β-Pruning

Aussagenlogik: Wissensrepräsentation

Formalisierung des Wissens:
Problembeschreibung (Modellierung) durch

- Definition der Aussagenvariablen
- Formeln für Zusammenhänge zwischen den Aussagen
  oft in speziellen Darstellungen, z.B. als
  - Menge aussagenlogischer Formeln (Regeln)
  - Formeln in spezieller Form, z.B. CNF
  - Entscheidungstabelle
  - Entscheidungsbaum (Regeln mit Alternative)
  - Entscheidungsdiagramm (BDD)

Modell = erfüllende Belegung der Aussagenvariablen mit Wahrheitswerten $\in \{0, 1\}$
(alternative Darstellung als Menge von Aussagenvariablen)
Fragen / Lösungen:

- Ist eine gegebene Belegung ein Modell?
- Existiert ein Modell?
- ein Modell
- Menge aller Modelle

Aussagenlogik: Wissensverarbeitung

semantische Verfahren:

- Wahrheitswerttabellen
- Spalte in Entscheidungstabelle finden
- Pfad in Entscheidungsbaum folgen
- Pfad in BDD folgen

syntaktische Verfahren:

- äquivalente Umformungen
- aussagenlogische Resolution

Prädikatenlogik: Wissensrepräsentation

Formalisierung des Wissens:
Problembeschreibung (Modellierung) durch

- Signatur zur Problembeschreibung:
  Funktions- und Relationssymbole mit Typ
  (abstrakter Datentyp, Interface)
- Formeln für Zusammenhänge zwischen den Aussagen (Axiome)

Modell (Interpretation) =

- algebraische Struktur und
- Belegung der Individuenvariablen mit Elementen der Trägermenge der Struktur

Fragen / Lösungen:

- Ist eine gegebene Interpretation ein Modell?
- Existiert ein Modell?
- ein Modell bzw. alle Modelle (i.A. unendlich)
- alle intendierten Modelle

- Haben zwei Formeln (Formelmengen) dieselben Modelle?
- Folgt eine Formel aus einer Formelmenge (Wissensbasis)?

Prädikatenlogik: Wissensverarbeitung

syntaktische Verfahren:

- prädikatenlogische Resolution mit Unifikation
- andere Kalküle
- äquivalente Umformungen

semantische Verfahren nur bei Strukturen mit endlichen Trägermengen möglich

1. Grundinstanziierung:
   Ersetzung aller Formeln durch die Menge alle ihrer Grundinstanzen
   Betrachtung der Grundatome als aussagenlogische Atome
2. Anwendung aussagenlogischer Verfahren

Logische Programmierung

Spezialfall (prädikatenlogischen) Schließens

Wissensrepräsentation:

- logische Programme: Mengen von
  - Fakten (Atome)
  - Regeln (Implikation)
- Anfrage (Konjunktion von Atomen)
- Lösung: Antwort auf Anfrage (ja / nein),
  evtl. mögliche Belegung der Individuenvariablen, so dass
  Anfrage aus Programm folgt

Wissensverarbeitung:

- Rückwärtsverkettung (PROLOG):
  prädikatenlogische Resolution (Unifikation)
- Vorwärtsverkettung (DATALOG):
  Fixpunkte von Konsequenzoperatoren

Nichtmonotones Schließen

Grundannahmen (entprechend menschlicher Intuition):

- closed world assumption (CWA):
  Was nicht in der Wissensbasis steht, gilt nicht.
- schwache Negation:
  Was nicht hergeleitet werden kann, gilt nicht.
- starke Negation:
  Genau die Aussagen, deren Negation hergeleitet werden kann, gelten nicht.

Semantik: Menge möglicher (intuitiver) Modelle

- definite Programme (Hornklauseln):
  (immer eindeutiges) minimales Modell
  Bestimmung: kleinster Fixpunkt des Konsequenzoperators
- normal logische Programme: stabile Modelle
  Bestimmung durch
  1. Raten einer Interpretation I und
  2. Testen: kleinstes Modell des (immer definiten) I-Reduktes (Gelfond-Lifschitz-Transformation)
- erweiterte logische Programme: Answer-Sets

Maschinelles Lernen

- überwacht
  - korrigierend
  - bestärkend (reinforcement)
- unüberwacht

Künstliche Neuronen

- biologisches Vorbild
- mathematisches Modell
- Eingangs-, Aktivierungs-, Ausgangsfunktion
- Lernregeln: Hebb, Δ
- McCullochs-Pitts-Neuron
- Schwellwert-Neuron
- RBF-Neuron
- Faltungs-Neuron
- ...
- geometrische Interpretationen

Künstliche Neuronale Netze

- Schichten-Struktur
- Ein-, Mehr-Schicht-FFN
- rekurrente Netze
- RBF-Netze
- Assoziativspeicher: BAM, Hopfield-Netz
- Cognitron, CNN
- Lernverfahren / Training
- Anwendungen