Künstliche Intelligenz (Wissensrepräsentation und -Verarbeitung)

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
- Neuronale Netzwerke
- Berechenbarkeitstheorie
- 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)
  Modellierung des Denkens
  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

Aktuelle Entwicklung

starker Fortschritt einiger KI-Methoden 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 starkes 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

Von Daten zur 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 (z.B. Körpertemperatur$= 39.7°$)
Wissen	Information mit einem Nutzen, trägt zur Lösung eines Problemes bei, Nutzen abhängig von vorhandenem Kontextwissen z.B. Kontext: Körpertemperatur$> 39.0°$ ist Fieber, bei Fieber ist Fieberbehandlung notwendig, mögliche Fieberbehandlungen z.B. Wadenwickel, Medikamente
Wissenserwerb	selbständige Informationsgewinnung über Gründe, 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

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

Inhalt der LV

- heuristische Suche
- Spielbaum-Suche
- Logisches Schließen
- Planen
- Unscharfes / probabilistisches Schließen
- Bayes-Netze
- Künstliche Neuronale Netze
- Kausalität (Zusammenhang von Ursache und Wirkung)
- Modellierung ethischer Prinzipien (mentale Modelle)

Organisation

6 ECTS
Präsenzzeit 56 h, Vor- und Nachbereitungszeit 124 h

- wöchentlich eine Vorlesung
- wöchentlich ein Seminar (Reading group)
  zu aktuellen Forschungsbeiträgen
  Literatur wird begleitend bekanntgegeben
  PVL und Notenbonus
- Klausur (90 min)
  zum Inhalt von Vorlesung und Seminar

Aufgaben für Seminar am 12. April 2019

Lesen Sie die KI-Ethikleitlinien der EU
Ethics guidelines for trustworthy AI
`https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai`

- Grundprinzipien
- Pläne zur Umsetzung
- Standards, Überprüfung
- ...

Informieren Sie sich über die Autoren (52 Expertinnen und Experten)
High-Level Expert Group on Artificial Intelligence

Vergleichen Sie die KI-Ethikleitlinien mit dem Entwurf
`https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=57112`
(optional)

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
- Künstliche neuronale Netze
- Evolutionäre Algorithmen

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 einer Aufgabeninstanz:
Schüler (System)

1. verwendet ein Lösungsverfahren V für diese Aufgabe
2. bestimmt eine Lösung l der gegebenen Aufgabeninstanz
3. erfährt (oder bestimmt) die 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)
- 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
- künstliches / maschinelles 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: Paare (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)

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

Bestärkendes Lernen

Trainingsmenge: Eingaben

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: 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. Aufstehen eines humanoiden Roboters
  Trainingsmenge: Menge von Parametern (Motorstellung)
  Bewertung der Lösung: ja, falls Plan zum Erfolg geführt hat (Roboter steht sicher), sonst nein

Unüberwachtes Lernen

Trainingsmenge: 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
- 2000, Begriff Deep Learning für KNN, Faltungsnetze
- 2009 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

Was bisher geschah

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

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.

Was bisher geschah

- biologisches Vorbild neuronaler Netze und Lernvorgänge darin
- künstliche Neuronen (mit binären Ein- und Ausgängen):
  - McCulloch-Pitts-Neuron (ohne Eingangsgewichte)
  - Schwellwertneuron (mit Eingangsgewichten)
- Feed-Forward-Netze
  gerichteter Graph mit Kantengewichten (Matrix)
  (parallele und sequentielle Berechnung)
- Verwendung künstlicher neuronaler Netze:
  - Lernphase (aufwendig, aber nur einmal auszuführen)
  - Einsatzphase (schnell, wird oft ausgeführt)
- Lernverfahren:
  - überwacht
    - korrigierend, z.B. durch Δ-Regel
    - bestärkend
  - unüberwacht
- überwachtes Lernen eines Schwellwertneurones durch schrittweise Änderung der Gewichte (Δ-Regel)

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

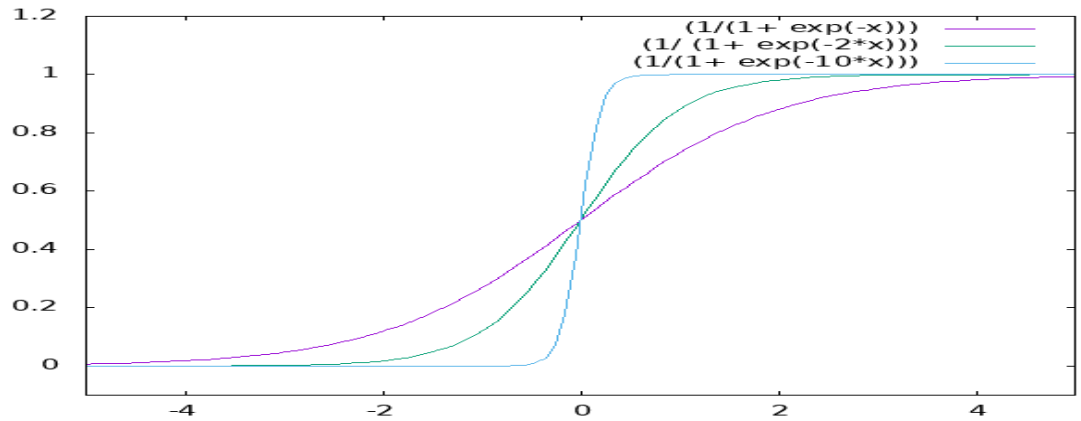
zur Anwendung in KNN: differenzierbare Aktivierungsfunktion

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:

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

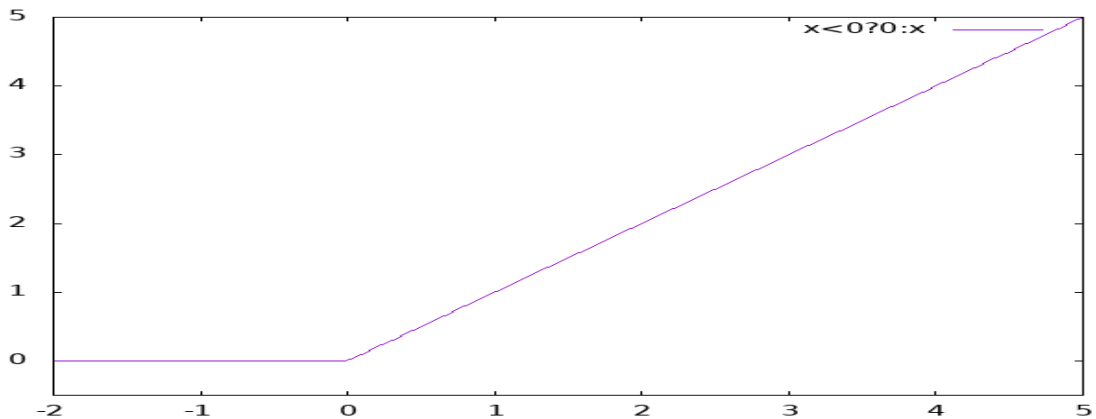
in jedem Punkt eindeutige Abstiegsrichtung

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

Aktivierungsfunktion 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“

Rekurrente Netze: Motivation

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

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

mögliche Ansätze

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

Wiederholung: allgemeine KNN

Netzstruktur (Topologie):
gerichteter Graph $G = (V, E)$ mit

- endliche Menge $V = \{v_1, \ldots, v_n\}$ von Knoten (Neuronen)
  evtl. einige als Eingabe- bzw. Ausgabeneuronen gekennzeichnet (nicht notwendig)
- Menge $E \subseteq V \times V$ von (gewichteten) Kanten

eine Gewichtsmatrix $R^{V \times V}$ für alle möglichen Verbindungen zwischen Neuronen

Rekurrente KNN

Netze mit Kanten zwischen beliebigen Neuronen

erlaubt Nachnutzung von Ausgaben aus vorangegangenen Schritten

Repräsentation zeitlicher Folgen von Mustern

Idee:
aktuelle Ausgaben als Eingaben im nächsten Schritt nutzen

„Kurzzeitgedächtnis“

Netzstruktur:

- analog Feed-Forward-Netz
- zusätzliche Neuronen und Kanten für Rückkopplung
  (Informationsspeicherung bis zum folgenden Schritt)

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 $(v, v), (v, u)$
- Schwellwerte $\theta_u = 1$, $\theta_v = 2$

Zustand rekurrenter Netze

Zustand eines neuronalen Netzes (zeitveränderlich)
Aktivierung aller Neuronen:
Zuordnung S: Neuron $\rightarrow \mathbb{R}$

(evtl. genügen Kontextneuronen)

Übersetzung in Zustandsübergangssysteme
(endliche Automaten)

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

Mathematisches Modell: Rekursion

Wiederholung: KNN als Berechnungsmodell

FFN als Berechnungsmodell:

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

rekurrentes Netz als Berechnungsmodell:

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

„Entwirrung“ rekurrenter Netze

Idee:

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

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

- Durchlauf jeder Netz-Kopie ist ein Zeitschritt,
- Lernen durch Backpropagation des entwirrten KNN (Backpropagation through time)

Jordan-Netze

Idee: Nachnutzung der Netzausgaben
Netz-Topologie:

- Feed-Forward-Netz mit trainierbaren Vorwärtskanten,
- für jedes Ausgabeneuron ein zusätzliches Kontextneuron in der Eingabeschicht
  (zur Speicherung der Netzausgaben)
  Aktivierungsfunktion: Identität
- zusätzliche Verbindungen von jedem Neuron der Ausgabeschicht zu seinem Kontextneuron mit
  festen Gewichten λ (meist $\lambda = 1$),
  Speicherung der Ausgaben
- evtl. direkte Verbindungen von jedem Kontextneuron zu sich selbst mit festem Gewicht γ
  (zur weiteren Speicherung der Netzausgaben)
- zusätzliche Verbindungen von jedem Kontextneuron zu jedem Neuron der ersten versteckten Schicht mit
  trainierbaren Gewichten,
  (zur Verwendung der gespeicherten Ausgabe im Folgeschritt)

Jordan-Netze: Berechnung

Beispiel: Eingang = gewichtete Summe, Aktivierung = Identität,
$x(t)$ – Netzeingabe zum Zeitpunkt t
$S(t)$ – Zustand (Aktivierung der Kontextneuronen) zum Zeitpunkt t
Ausgabe: $y(t) = f(x(t), S(t))$
(Zustands-)Übergangsfunktion: $S(t+1) = g(x(t), S(t))$
Zustand des Netzes nach mehreren Schritten (Schritt für gesamtes Netz), beginnend im Startzustand S_0

$$
\begin{aligned}
S(t) &= \begin{cases} S_0 & \text{falls } t = 1 \\ \gamma S(t-1) + \lambda y(t-1) & \text{falls } t > 1 \end{cases} \\
&= \gamma^{t-1} S_0 + \lambda \sum_{n=1}^{t-1} \gamma^{n-1} y(t-n)
\end{aligned}
$$

Spezialfall $S_0 = 0$ und $\lambda = 1$: $\quad S(t) = \sum_{n=1}^{t-1} \gamma^{n-1} y(t-n)$

exponentiell gewichtete Summe aller bisherigen Netzausgaben
$\gamma \in [0, 1]$ steuert „Erinnerungsvermögen“ des Netzes

Elman-Netze

Idee: Nachnutzung der Aktivierung der versteckten Neuronen

Netz-Topologie:

- Feed-Forward-Netz (z.B. SRN 3-Schicht-FFN)
- für jedes versteckte Neuron ein zusätzliches Kontextneuron in der vorigen Schicht
  (zur Speicherung der Aktivierung)
  Aktivierungsfunktion: Identität
- zusätzliche Verbindungen von jedem versteckten Neuron zu seinem Kontextneuron mit festem Gewicht 1
  Speicherung der Aktivierung aller versteckten Neuronen
- zusätzliche Verbindungen von jedem Kontextneuron zu jedem Neuron der Schicht des Originalneurons mit trainierbaren Gewichten,
  (zur Verwendung der gespeicherten Aktivierung im Folgeschritt)

Was bisher geschah

Künstliche Neuronen:

- Mathematisches Modell und Funktionen:
  Eingabe-, Aktivierungs- Ausgabefunktion
- Boolesche oder reelle Ein-und Ausgaben
- Aktivierungsfunktionen:
  - Schwellwertfunktion
  - lineare Funktion
  - sigmoide Funktion

Künstliche Neuronale Netze:

- Aufbau: gerichteter Graph mit Kantengewichten (Gewichtsmatrix)
- Feed-Forward-Netze
- Training (schrittweise Minimierung der quadratischen Abweichung auf der Trainingsmenge):
  - Δ-Regel für Ein-Schicht-Feed-Forward-Netze mit linearer oder Schwellwert-Aktivierung
  - Backpropagation für Mehr-Schicht-Feed-Forward-Netze mit sigmoider Aktivierung

Radiale-Basisfunktions-Netze

Anwendung zur Klassifizierung von Mustern (Merkmalsvektoren)

Annahmen:

- Klassen haben Zentren (Schwerpunkte),
- alle Eingabevektoren nahe dazu gehören zur selben Klasse

2-Schicht-FFN mit vollständig verbundenen Schichten

- Eingaben $x \in \mathbb{R}^m$
- Ausgaben $y \in \mathbb{R}^n$
- eine versteckte Schicht h (mit l Neuronen)
  enthält oft mehr Neuronen als die Eingabeschicht

Neuronen der verschiedenen Schichten haben verschiedene Aktivierungsfunktionen:

- versteckte Schicht: nichtlinear
- Ausgabeschicht: linear

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

Versteckte Neuronen im RBF-Netz

Idee:

- Eingangsgewichte eines Neurons j der versteckten Schicht interpretiert als Koordinaten eines Punktes $(w_{1j}, \ldots, w_{mj}) \in \mathbb{R}^m$ (Zentrum einer Klasse)
- Eingangsfunktion $I_j : R^m \to \mathbb{R}$ des Neurons j berechnet Abstand des Eingabevektors $(x_1, \ldots, x_m)$ vom Zentrum $(w_{1j}, \ldots, w_{mj}) \in \mathbb{R}^m$
- Aktivierungfunktion: radiale Basisfunktion $A_j : \mathbb{R} \to \mathbb{R}$ nimmt größten Wert im Zentrum an fällt mit wachsendem Abstand vom Zentrum
- das Neuron der versteckten Schicht am aktivsten, welches das zum Eingabevektor nächste Zentrum repräsentiert

Abstandsfunktionen

(Eingabefunktionen der versteckten Neuronen im RBF-Netz)

Abstandsfunktion $d : \mathbb{R}^{2m} \to \mathbb{R}$ mit den Eigenschaften:

- $\forall x, y \in \mathbb{R}^m : d(x, y) = 0$ gdw. $x = y$
- $\forall x, y \in \mathbb{R}^m : d(x, y) = d(y, x)$ (kommutativ)
- $\forall x, y, z \in \mathbb{R}^m : d(x, y) + d(y, z) \geq d(x, z)$ (Dreiecksungleichung)

Beispiele: $I(x_1, \ldots, x_m) = d_k(x, w_j) = \sqrt[k]{\sum_{k=1}^{m} (w_{kj} - x_k)^k}$

- für $k = 2$: $I(x_1, \ldots, x_m) = d_2(x, w_j) = \sqrt{\sum_{k=1}^{m} (w_{kj} - x_k)^2}$ Euklidischer Abstand zwischen Eingangs- und Gewichtsvektor
- für $k = 1$: $I(x_1, \ldots, x_m) = d_1(x, w_j) = \sum_{k=1}^{m} |w_{kj} - x_k|$ Manhattan-Metrik
- für $k \to \infty$: $I(x_1, \ldots, x_m) = \max\{|w_{kj} - x_k| \mid i \in \{1, \ldots, m\}\}$ Maximum-Metrik

Radiale Funktionen

Radiale Funktion $f : \mathbb{R}_{\geq 0} \to [0, 1]$ mit den folgenden Eigenschaften:

- aus $x < y$ folgt $f(x) \geq f(y)$ (monoton fallend)
- $f(0) = 1$
- $\lim_{x \to \infty} f(x) = 0$ (verschwindet im Grenzwert)

(fällt ausgehend vom Zentrum 0 in alle Richtungen)

Beispiele:

- Schwellwertfunktion (fallend)

$$f_\theta(x) = \begin{cases} 0 & \text{falls } x > \theta \\ 1 & \text{sonst} \end{cases}$$

- linear $f_m(x) = \max(0, 1 - mx)$
- Gauß-Funktion $f_c(x) = e^{-cx^2}$

Ausgabeneuronen im RBF-Netz

- Eingaben (von der versteckten Schicht): $h \in \mathbb{R}^l$
- Gewichte: $W' \in \mathbb{R}^{l \times n}$
- Ausgaben: $y \in \mathbb{R}^n$

- Eingabefunktion: gewichtete Summe
- Aktivierungsfunktion: Identität (linear)
- Ausgabefunktion: Identität

(Schwellwertneuronen mit linearer Aktivierung)

RBF-Netze: Beispiele

- 2-1-1 -Netz für $\wedge$
  - erste Schicht (RBF): Zentrum $w_{1,h} = w_{2,h} = 1$,
    Eingabefunktion: Euklidische Metrik
    Aktivierung: Stufenfunktion
    Radius $\theta_h = 1/2$
  - zweite Schicht: Gewicht $w_{h,y} = 1$,
    Eingabefunktion: gewichtete Summe
    Aktivierung: linear
    Schwellwert $\theta_y = 0$
- 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): Zentren $w_{1,h1} = w_{2,h1} = 1$,
    $w_{1,h2} = w_{2,h2} = 0$,
    Eingabefunktion: Euklidische Metrik
    Aktivierung: Stufenfunktion
    Radien $\theta_{h1} = \theta_{h2} = 1/2$
  - 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

Approximation einer Funktion $f : \mathbb{R} \to \mathbb{R}$ durch Linearkombination (gewichtete Summe) von radialen Funktionen, z.B.

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

Zwei-Schicht-FF-Netz:

- ein Eingabeneuron x
- k versteckte Neuronen $h_1, \ldots, h_k$
  jedes für eine Basisfunktion
- ein Ausgabeneuron y

Beispiel

Approximation n-stelliger Boolescher Funktionen:

- n Eingabeneuronen x_i
- 2^n versteckte Neuronen h_i
  Eingangsgewichte (jede mögliche Eingabe als Zentrum)
  Eingangsfunktion: Euklidische oder Manhattan-Metrik
  Aktivierung: Stufenfunktion
  alle Radien $1/2$
- ein Ausgabeneuron y
  zu bestimmende Gewichte w_i, Schwellwert 0

RBF-Netze – Lernen

übliches Vorgehen: nacheinander

1. Gewichte der ersten Schicht
   (Eingabe zu versteckten Neuronen):
   Bestimmung der Anfangspunkte der Zentren, z.B.
   - gleichmäßig überdeckend
   - alle Trainingsmuster
   - durch zufällige Auswahl von Trainingsmustern
   - durch Clustering-Techniken,
     z.B. unüberwachtes Training (später)
2. Gewichte der zweiten Schicht (zu Ausgabeneuronen):
   direkte Berechnung oder überwachtes Training
   (z.B. Delta-Regel)
   Bestimmung der Faktoren vor den Basisfunktionen

Eigenschaften von RBF-Netzen

Vorteile:

- einfache Topologie
- schnelle Berechnung
- Netzausgabe außerhalb der Trainingsmenge gering
- Gewichte können direkt bestimmt werden (ohne Training)

Nachteile:

- Qualität der Approximation durch Lage der Zentren bestimmt
- Lernerfolg hängt stark von der Start-Instanziierung der Gewichte der ersten Schicht (Zentren) ab
- Auswendiglernen der Trainingsdaten

Cognitron

Beobachtungen im visuellen System:

- sendet vorverarbeitete Signale an Gehirn
- Verbindung benachbarter Neuronen
  horizontale Zellen berechnen Mittelwert (der Helligkeit)
  wirken hemmend auf Signale nahe beim Mittelwert
- ähnlich Faltung in DBV

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)
  - 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-Lernen

Überwachtes Lernen durch Backpropagation:

- Faltungsschichten:
  Backpropagation durch Faltung mit gespiegelten Kernen
- Pooling-Schichten (z.B. bei Max-Pooling):
  auf Hinweg Position (Koordinaten) des maximalen Elementes speichern
  Backpropagation: Abstieg in Richtung dieser Position
- klassische Schichten: Gradientenabstieg wie bisher

Beispiel Missionare + Kannibalen

informale Problembeschreibung:

- Zu Beginn: 3 Missionare + 3 Kannibalen an einem Flussufer
- Ziel ist das Übersetzen aller Personen.
- Es gibt nur ein Boot, welches genau zwei Personen fasst.
- Alle Personen im Boot steigen am Ufer aus (und dann ggf. wieder ein).
- Sobald an einer Stelle (Ufer, Boot) mehr Kannibalen als Missonare sind, werden die Missionare gefressen.

formale Modellierung (Beispiel):

- Zustände: $S \subseteq \{0, \ldots, 3\}^4 \times \{-1, 1\}$ mit $\forall((m_s, k_s), (m_z, k_z), u) \in S : m_s + m_z = 3 \wedge k_s + k_z = 3 \wedge \ldots$
- Startzustand: $((3, 3), (0, 0), -1)$
- Zielzustand: $((0, 0), (3, 3), 1)$
- Zustandsübergänge $((m_s, k_s), (m_z, k_z), u) \in S \rightarrow ((m'_s, k'_s), (m'_z, k'_z), -u) \in S$ mit $\exists n_k, n_m : 1 \leq n_m + n_k \leq 2 \wedge \ldots$

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 bzw. Baum
- 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

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. 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

- blinde Suchverfahren: Tiefensuche, Breitensuche

Allgemeiner Suchalgorithmus

1. aktuelle Menge der zu untersuchenden Knoten $L_a = \{s\}$
2. aktuelle Menge der erledigten $L_x = \emptyset$
3. solange nicht (gefunden oder $L_a = \emptyset$) wiederhole:
   - 3.1 Verschiebe einen festgelegten Knoten u aus L_a in L_x
   - 3.2 Füge alle Nachbarn von u, die $L_a \cup L_x$ nicht enthält, (auf eine festgelegte Art) in L_a ein

Verschiedene Suchverfahren unterscheiden sich nur in der Auswahl des expandierten (festgelegten) Knotens aus L_a

nach Festlegung durch Datenstruktur zur Verwaltung von L_a

- Stack: Tiefensuche
- Queue: Breitensuche

Schrittweise Vertiefung (ID)

(iterative deepening)

Ziel: Verbindung der Vorteile von

- Tiefensuche (geringer Speicherbedarf)
- Breitensuche (Vollständigkeit)

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

nicht vollständig (Lösungszustände, die mehr als m von der Wurzel entfernt sind, werden nicht gefunden)

2. Idee: schrittweise Vertiefung

Nacheinanderausführung beschränkter Tiefensuchen für alle $m \in \mathbb{N}$ ($<$-geordnet), solange keine Lösung gefunden wurde

Vorteil: vollständig, optimal

Nachteil:

Knoten nahe des Startzustandes werden mehrfach expandiert

aber (asymptotischer) Zeit- und Platzbedarf wie Tiefensuche

Bidirektionale Suche

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

Speicherbedarf geringer als bei Breitensuche

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

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:

- I Breitensuche (Kosten = Tiefe des Knotens) I kürzeste Wege (Kosten = Abstand des Knotens vom Startknoten) Dijkstra-Algorithmus

Uniforme Kostensuche ist wie Breitensuche und Tiefensuche ein uninformiertes Suchverfahren

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 Nachbarn 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.
Aus zielerkennend und konsistent folgt nicht-überschätzend.

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 Besertungsfunktionfunktion 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 v und alle Nachfolger u von v 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

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

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
- Navigation (z.B. von Robotern)
- Scheduling
- Produktionsplanung

Reading Group KW 20
Robert C. Holte, 2010: Common Misconceptions Concerning Heuristic Search
`https://aaai.org/ocs/index.php/SOCS/SOCS10/paper/view/2073/2500`

ÜA (zur Information):
Serie 2 aus BA-Modul Grundlagen der Künstlichen Intelligenz

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: heuristische, 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)

Reading Group

Joseph K. Barker and Richard E Korf:
Solving Dots-And-Boxes
Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012
`https://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/viewFile/5126/5218`

ÜA (zur Info):
Serie 3 aus Modul Grundlagen der Künstlichen Intelligenz

Was bisher geschah

- Daten, Information, Wissen
- explizites und implizites Wissen
- intelligente Agenten

Wissensrepräsentation und -verarbeitung:

Wissensbasis: Kontextwissen

Formulierung der Aufgabe: fallspezifisches Wissen

Lösung: Bedingungen

Lösungsverfahren

Wissensrepräsentation und -verarbeitung in Zustandsübergangssystemen:

Wissensbasis: Graph (mit Knoten- und Kantenmarkierungen)

Formulierung der Aufgabe: Weg von Startknoten zu Lösung gesucht

Lösung: Bedingungen

Lösungsverfahren: Suchverfahren

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

informiert: Besten-, Greedy-, A*-Suche

Zwei-Personen-Spiele, MiniMax-Werte, α-β-Pruning

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 durch Logiken

Anforderungen an Formalismus zur Wissensrepräsentation:

- hinreichende Ausdrucksstärke
- syntaktisch und semantisch eindeutig
- Möglichkeit der maschinellen Verarbeitung

- klassische Aussagenlogik AL(P)
  - hinreichende Ausdrucksstärke: oft ja
  - syntaktisch und semantisch eindeutig: ja
  - Möglichkeit der maschinellen Verarbeitung: ja (algorithmische Entscheidbarkeit)
- klassische Prädikatenlogik (der ersten Stufe) FOL(Σ)
  - hinreichende Ausdrucksstärke: meist ja
  - syntaktisch und semantisch eindeutig: ja
  - Möglichkeit der maschinellen Verarbeitung: meist ja (Unentscheidbarkeit)
- nichtklassische Logiken:
  - Mehrwertige Logiken, z.B.Fuzzy-Logik
  - nichtmonotone Logiken
  - Modale Logiken, z.B. Temporallogiken

Wissensrepräsentation und -verarbeitung in Logiken

Wissensbasis: Formelmenge Φ

Problemdarstellung: Formel ψ
repräsentiert die Frage:
(Für welche Variablenbelegung) Folgt ψ aus Φ?

Lösung: ja / nein, evtl. erfüllende Belegung

Lösungsverfahren:

Folgern (semantisch):
z.B. Wahrheitswerttabellen, Modellmengen

Schließen (syntaktisch):
Kalküle, z.B. Resolution

Aussagenlogik – Syntax

Junktoren Syntax: Symbole $\mathbf{t}, \mathbf{f}$ (nullstellig),
$\neg$ (einstellig), $\vee, \wedge, \rightarrow, \leftrightarrow$ (zweistellig)
Semantik: Wahrheitswertfunktion

Atome Syntax: Aussagenvariablen (elementare Formeln)
Semantik: Wahrheitswert

Formeln Syntax (induktive Definition):

- **IA:** Alle Atome sind Formeln.
- **IS:** Sind j ein n-stelliger Junktor und $\varphi_1, \ldots, \varphi_n$ Formeln,
  dann ist auch $j(\varphi_1, \ldots, \varphi_n)$ eine Formel.

Baumstruktur
Semantik: Boolesche Funktion

Beispiele:

- $(p \wedge (q \rightarrow r)) \vee (r \rightarrow \neg p)$
- $\neg p \wedge p$

Bedeutung der Junktoren

		Syntax	Semantik
	Stelligkeit	Symbol	Wahrheitswertfunktion
wahr	0	$\mathbf{t}$	1
falsch	0	$\mathbf{f}$	0
Konjunktion	2	$\wedge$	min
Disjunktion	2	$\vee$	max
Negation	1	$\neg$	$x \mapsto 1 - x$
Implikation	2	$\rightarrow$	$\leq$
Äquivalenz	2	$\leftrightarrow$	$=$

Aussagenlogik – Semantik

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

Wert von $\varphi \in \mathrm{AL}(P)$ unter Belegung W: $W(\varphi)$ mit
$W(p)$ für $\varphi = p \in P$ und
induktive Berechnung für zusammengesezte Formeln

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 (und Allgemeingültigkeit) ist algorithmisch entscheidbar.

semantisch	z.B. durch Wahrheitswerttabellen
syntaktisch	z.B. durch Resolution

Werkzeuge: SAT-Solver

Modellierungsbeispiel (Aussagenlogik)

1. Es wird nicht mehr viel Eis gekauft, wenn es kalt ist.
2. Der Eisverkäufer ist traurig, wenn nicht viel Eis gekauft wird.
3. Es ist kalt.

Wissensbasis: ...

Problem: ...

Lösung: ...

Lösungsverfahren: ...

neue zusätzliche Aussage (Erweiterung der Wissensbasis):

4. Der Eisverkäufer ist nicht traurig.

Semantische Äquivalenz

Relation $\equiv \subseteq \mathsf{AL}(P) \times \mathsf{AL}(P)$
(Relation zwischen zwei Formeln)

$$\varphi \equiv \psi \quad \text{gdw.} \quad \mathsf{Mod}(\varphi) = \mathsf{Mod}(\psi)$$

Beispiele:

- $p \to q \equiv \neg p \vee q$
- $p \vee q \equiv \neg p \to q$
- $p \wedge q \equiv \neg(p \to \neg q)$
- $p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$

Regeln der klassische Aussagenlogik (z.B. DeMorgan, Distributivgesetze) ermöglichen rein syntaktische äquivalente Umformungen.

Normalformen

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

$$I(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$$

Zu jeder Formel $\varphi \in \mathrm{AL}(P)$ existieren äquivalente Formeln in

NNF Formeln, in denen das Negationssymbol $\neg$ höchstens auf Atome angewendet wird
Beispiel: $\neg p \vee ((\neg q \vee p) \wedge q)$

CNF Formeln der Form $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} l_{i,j}$
mit Literalen $l_{i,j}$
Beispiel: $(\neg p \vee \neg q) \wedge (p \vee q) \wedge \neg q$

DNF Formeln der Form $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} l_{i,j}$
mit Literalen $l_{i,j}$
Beispiel: $\neg p \vee (\neg q \wedge p) \vee (p \wedge q)$

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

IF-NF $I(p, \varphi, \psi)$ mit $p \in P$, (Entscheidungsbäume)

Semantisches Folgern

Folgerungsrelation $\models \subseteq 2^{\mathsf{AL}(P)} \times \mathsf{AL}(P)$
(Relation zwischen Formelmenge und Formel)

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

Notation: $\models \psi$ statt $\emptyset \models \psi$ und $\varphi \models \psi$ statt $\{\varphi\} \models \psi$
Beispiele:

- $\{p\} \models p$,
- $\{p \to q, \neg q\} \models \neg p$,
- $\emptyset \models p \to p$
- $\{p, \neg p, \neg q\} \models q$

Es gilt:

$$\models \psi \quad \text{gdw.} \quad \psi \text{ allgemeingültig}$$

$$\varphi \equiv \psi \quad \text{gdw.} \quad (\varphi \models \psi \text{ und } \psi \models \varphi)$$

Semantisches Folgern

Fakt

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

Fakt

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

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

Fakt

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

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

Folgerung:

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

Was bisher geschah

Wissensrepräsentation und -verarbeitung in Logiken:

Wissensbasis: Kontextwissen

Formulierung der Aufgabe: fallspezifisches Wissen

Lösung: Bedingungen

Lösungsverfahren

WH – klassische Aussagenlogik:

- Syntax
- Semantik
- semantisches Schließen

Wiederholung: Syntaktisches Ableiten

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

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

Wiederholung: Syntaktisches Ableiten

gegeben: Formel φ (Formelmenge Φ)
Formel ψ
Frage: Gilt $\Phi \models \psi$

Idee: schrittweises Ableiten (ohne Zugriff auf die Semantik der Formeln) von Folgerungen aus einer Formelmenge durch syntaktische Umformungen

logischer Kalkül Menge von Regeln zur syntaktischen Umformung von Formeln (Formelmengen)
(ohne Änderung der Semantik der Formelmengen)

Ein logischer Kalkül K ist sinnvoll, wenn man zeigen kann:

Korrektheit Jede in K ableitbare Formel ist allgemeingültig.

Vollständigkeit Jede allgemeingültige Formel ist in K ableitbar.

Wiederholung: 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

Wiederholung: 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)

Wiederholung: 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

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

Wiederholung: 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.

Wiederholung: 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.)*

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

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

Wiederholung: Einbettung Aussagen- in Prädikatenlogik

Jede Formel $\varphi \in \mathrm{AL}(P)$ ist (syntaktisch) auch
eine Formel in $\mathrm{FOL}(\Sigma, \emptyset)$ mit

$$\begin{aligned} \Sigma &= (\Sigma_F, \Sigma_R) \\ \Sigma_F &= \emptyset \quad \text{und} \quad \Sigma_R = \{(p, 0) \mid p \in P\} \end{aligned}$$

Semantik:
Jede Belegung $W : P \to \{0, 1\}$ für $\varphi \in \mathrm{AL}(P)$
definiert eine Σ-Struktur $\mathcal{S}_W = (S, [\![\cdot]\!]_{\mathcal{S}_W})$ durch

$$\forall p \in P : [\![\cdot]\!]_{\mathcal{S}_W} = W(p)$$

W ist Modell (erfüllende Belegung) für $\varphi \in \mathrm{AL}(P)$
gdw. $W(\varphi) = 1$
gdw. $\mathcal{S}_W$ ist Modell für $\varphi \in \mathrm{FOL}(\Sigma)$.
($\varphi \in \mathrm{FOL}(\Sigma)$ enthält keine Individuenvariablen, Variablenbelegung deshalb irrelevant)
Aussagenlogik ist ein Fragment der Prädikatenlogik

Wiederholung: Übersetzung Prädikaten- in Aussagenlogik

Grundinstanziierung einer Formelmenge $\Phi \in \mathrm{FOL}(\Sigma, \mathbb{X})$
in der Σ-Struktur $\mathcal{A} = (A, [\![\cdot]\!]_{\mathcal{A}})$:

1. Definition eines neuen Konstantensymbols für jedes $d \in A$
2. Ersetzung der Formeln $\varphi \in \Phi$ durch ihre Grundinstanzen (induktiv):
   - IA: für Atome $\varphi = p(t_1, \ldots, t_n)$:
     $G(\varphi) = G((p(t_1, \ldots, t_n)) = \{\beta(p(t_1, \ldots, t_n)) \mid \beta : \mathbb{X} \to A\}$
   - IS: für Formeln mit n-stelligen Junktor j
     $\varphi = j(\varphi_1, \ldots, \varphi_n)$: $G(\varphi) = j(G(\varphi_1), \ldots, G(\varphi_n))$
3. Übersetzung $\exists x\varphi \;\to\; \bigvee_{d \in A} \varphi[x \mapsto d]$ und $\forall x\varphi \;\to\; \bigwedge_{d \in A} \varphi[x \mapsto d]$
4. Ersetzung jedes Grundatoms durch eine Aussagenvariable,
5. $G(\Phi) = \bigcup_{\varphi \in Phi} \{\beta(\varphi) \mid \beta : \mathbb{X} \to A\}$

Beispiele: $p(x, f(y)) \wedge \exists x p(y, x)$ in $A = \{1, 2, 3\}$, n-Damen-Problem

Vorteil: aussagenlogische Methoden anwendbar, Entscheidbarkeit

Nachteil:
- nur für Interpretationen in Strukturen mit endlicher Trägermenge und Signaturen ohne < 0-stellige Funktionen (nur Konstanten) möglich
- große unübersichtliche Formelmengen

Wiederholung: Prolog-Syntax

Regel (Horn-Klausel) `a :- a1, ..., am.`
Bedeutung in Prädikatenlogik (der ersten Stufe)
$(\forall X_1 \cdots \forall X_n((a_1 \wedge \cdots \wedge a_m) \rightarrow a)$
wobei $X_1, \ldots X_n$ alle in $a, a_1, \ldots, a_n$ vorkommenden Variablen sind.
Rumpf $a_1 \wedge \ldots \wedge a_m$, Kopf a

Fakt Atom `a.` (positives Literal, Regel ohne Rumpf)
Bedeutung in Prädikatenlogik (der ersten Stufe):
$\forall X_1 \cdots \forall X_n a$, wobei $X_1, \ldots X_n$ alle in a vorkommenden Variablen sind.

Zielklausel (Query, Anfrage) `?- a1, ..., an.`
Bedeutung in Prädikatenlogik (der ersten Stufe):
$(\forall X_1 \cdots \forall X_n(a_1 \wedge \cdots \wedge a_m)$
wobei $X_1, \ldots X_n$ alle in $a_1, \ldots, a_m$ vorkommenden Variablen sind.

Variablennamen beginnen mit Großbuchstaben,
Funktions- und Relationssymbole mit Kleinbuchstaben

Wiederholung: Prolog-Programme

Programm P (Wissensbasis):
endliche Menge von Fakten und Regeln,
repräsentiert eine prädikatenlogische Formelmenge Φ,
(repräsentiert eine prädikatenlogische Formel
$\varphi = \bigwedge_{\psi \in \Phi} \psi$)

Beispiel: Programm P

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

repräsentiert die Formelmenge

$$\Phi = \{l(p,k), l(b,z), l(t,a), \forall x(l(x,k) \rightarrow m(t,x))\}$$

Wiederholung: Prolog-Anfragen

Zielklausel Atom
repräsentiert eine prädikatenlogische Formel ψ

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

repräsentiert die Frage: Wen mag Tina?

repräsentiert durch die Formel (Behauptung): $\psi = \exists X \mathsf{mag}(\mathsf{tina}, X)$

negiertes Einfügen in den Kontext: $\neg\varphi = \forall X \neg \mathsf{mag}(\mathsf{tina}, X)$

also Einfügen der Klausel: $\neg \mathsf{mag}(\mathsf{tina}, X)$

Wiederholung: Prolog-Auswertung

Ausgewertet werden Paare (Φ, ψ) aus

- Programm Φ
- Zielklausel ψ

(prädikatenlogische Darstellung von Programm und Anfrage)

Antwort: Substitution θ mit $\Phi \models \theta(\psi)$
(Prolog-Ausgabe: Grundinstanzen $\theta(\psi)$ der Zielklausel ψ)

Wiederholung Prolog: Bestimmung der Antworten

- durch Lösung der Aufgabe:
  Für welche Substitutionen θ gilt $\Phi \models \theta(\psi)$?
- durch Lösung der äquivalenten Aufgabe:
  Für welche Substitutionen θ ist die Formelmenge $\Phi \cup \neg\theta(\psi)$ unerfüllbar?
- durch Lösung der äquivalenten Aufgabe:
  Für welche Substitutionen θ gilt $\Phi \cup \neg\theta(\psi) \models \mathbf{f}$?
- durch Bestimmung der Substitutionen θ, für die $\mathbf{f}$ aus $\Phi \cup \neg\theta(\psi)$ syntaktisch herleitbar ist.
- durch prädikatenlogische Resolution mit festgelegter Auswertungsreihenfolge (SLD-Resolution)

Ausführung durch Prolog-Interpreter,
z.B. SWI-Prolog `http://www.swi-prolog.org`
oder online `http://www.learnprolognow.org`

Wiederholung: Beispiel für Prolog-Auswertung

Programm P:

```
p(a,b). p(b,c). p(c,d).
e(X,Y) :- p(X,Y).
e(X,Y) :- p(X,Z), e(Z,Y).
```

Zielklausel $e(X, d)$
Prädikatenlogische Bedeutung:

- Programm P:

$$\Phi = \left\{ \begin{array}{l} p(a,b), p(b,c), p(c,d), \forall X \forall Y (p(X,Y) \rightarrow e(X,Y)), \\ \forall X \forall Y \forall Z (p(X,Z) \wedge e(Z,Y) \rightarrow e(X,Y)) \end{array} \right\}$$

- Zielklausel $\psi = e(X, d)$, negiert $\neg\psi = \neg e(X, d)$
- kombiniert als Formelmenge

$$\{\neg\psi\} \cup \Phi = \left\{ \begin{array}{l} \neg e(X,d), p(a,b), p(b,c), p(c,d), \\ \forall X \forall Y (p(X,Y) \rightarrow e(X,Y)), \\ \forall X \forall Y \forall Z (p(X,Z) \wedge e(Z,Y) \rightarrow e(X,Y)) \end{array} \right\}$$

Antworten: X = c ; X = a ; X = b ; No

Prädikatenlogische Resolution

Berechnung einer prädikatenlogischen Resolvente der Klauseln $l_1 \vee \ldots \vee l_n\}$ und $l'_1 \vee \ldots \vee l'_m$:

- Variablenumbenennung (Klauseln haben keine gemeinsamen Variablen)
- Bestimmung eines allgemeinsten Unifikators σ für ein Paar von Literalen l_i und $\neg l'_j$
- Resolvente: $\sigma(l_1) \vee \ldots \vee \sigma(l_{i-1}) \vee) \sigma(l_{i+1}) \vee \ldots \vee \sigma(l_n) \vee \sigma(l'_1) \vee \ldots \vee \sigma(l'_{j-1}) \vee \sigma(l'_{j+1}) \vee \ldots \vee \sigma(l'_m)$

Beispiel

Klauselmenge

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

Resolution:
positive Literale $P(x, b), P(a, y)$
negatives Literal $\neg P(z, w)$

Substitution $\sigma = [x \mapsto a, y \mapsto b, z \mapsto a, w \mapsto b]$
Resolvente $Q(a, f(b)) \vee \neg Q(b, a)$

Prolog – Operationale Semantik (SLD-Resolution)

für Mengen von Hornklauseln
Auswahl der angewendeten Regeln in einer festen Reihenfolge:

- erste anwendbare Regel im Programm,
- in jedem Schritt entsteht eine Hornklausel
- neu erzeugte Regel sofort angewenden
- zuerst das erste Rumpf-Literal resolvieren
- Auswahl nur für dessen Resolutions-Partner (Suchbaum, meist Tiefensuche)

(immer Substitution β mitführen)

Prolog – Deklarative Semantik

Prolog-Programm P = endliche Menge von Fakten und Regeln (Formelmenge)

deklarative Semantik (Bedeutung) von P:
Menge aller Modelle der Formelmenge P
enthält für Hornklausel-Programme P genau ein minimales Modell (bzgl. $\subseteq$)

Prolog-Programm P, Anfrage ψ
deklarative Semantik (Bedeutung) von P mit Anfrage ψ:
Menge aller Variablenbelegungen β (Antwort), für die $\beta(\psi)$ im minimalen Modell von P gilt

Ideales und Reales Prolog

wie hier definiert (ideal, genügt hier):

- Semantik ist deklarativ
- Reihenfolge der Regeln im Programm und Atome in Regel-Rumpf beeinflusst Effizienz, aber nicht Korrektheit

reales Prolog:

- cut (!) zum Abschneiden der Suche
  - green cut: beeinflusst Effizienz
  - red cut: ändert Semantik

  merke: cut $\approx$ goto, grün / rot schwer zu unterscheiden
- Regeln mit Nebenwirkungen (u. a. für Ein/Ausgabe)

für beides: keine einfache denotationale Semantik

Was bisher geschah

Wissensrepräsentation und -verarbeitung durch

- Künstliche Neuronale Netze (insbes. auch CNN)
- Zustandsübergangssysteme
- Klassische Logiken
- Logische Programme (Prolog)
  Beispiele zum Planen

Regeln

Regel: Implikation $r = (\varphi \rightarrow \psi)$, meist mit
$\varphi = (b_1 \wedge \ldots \wedge b_n \wedge \neg c_1 \wedge \ldots \wedge \neg c_m)$ und $\psi = h$
mit (aussagen- oder prädikatenlogischen) Atomen
$b_1, \ldots, b_n, c_1, \ldots, c_m, h$

Bestandteile der Regel r:

Kopf h (Folgerung)

Rumpf $b_1 \wedge \ldots \wedge b_n \wedge \neg c_1 \wedge \ldots \wedge \neg c_m$
(Voraussetzungen)
positive Voraussetzungen $b_1, \ldots, b_n$
negative Voraussetzungen $c_1, \ldots, c_m$

Logisches Programm (regelbasiertes System) besteht aus

- Wissensbasis (R, F) mit
  - Regelmenge R
  - Faktenmenge F
- Regelinterpreter, z.B. Prolog-Interpreter

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),
keine Funktionssymbole höherer Stelligkeit

Semantik Interpretation über einer festen endlichen
Trägermenge,
meist Menge aller vorkommenden Konstantensymbole
Modelle: Mengen von Grundatomen (Konstanten)

Datalog: Syntax und Semantik

Datalog-Syntax:

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 Atomen
`b1, ... , bn, h`
Datalog-Fakten sind Datalog-Klauseln mit $n = 0$.

Datalog-Wissensbasis: endliche Menge von Datalog-Klauseln

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

übliche Semantik der klassischen Prädikatenlogik,
aber nur kleinstes Modell (bzgl. $\subseteq$) relevant

Semantik: Erweiterung der Faktenmenge

gegeben: Logisches Programm $P = (F, R)$ mit

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

Folge von Faktenmengen (Zuständen)

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

datenorientierte Suche

Beispiel

aus der Wissensbasis

F Tom ist ein Baby.

F Tom ist männlich.

R1 Babies sind Kinder.

R2 Männliche Kinder sind Jungen.

R3 Weibliche Kinder sind Mädchen.

folgt (ohne gezielte Anfrage):

Tom ist ein Kind.

Tom ist ein Junge.

Regel feuert (ist anwendbar) in einer Faktenmenge gdw. Voraussetzung (Regelrumpf) erfüllt.

Verfahren: Schrittweise Erweiterung der Faktenmenge um gültige Fakten (Köpfe feuernder Regeln):

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

Konsequenzoperator für definite Programme

gegeben: definites Programm $P = (R, F)$ (Wissensbasis)
Faktenmenge $M \subseteq \text{Atom}(P)$
Jedes Programm P definiert seinen
Konsequenzoperator $T_P : 2^{\text{Atom}(P)} \to 2^{\text{Atom}(P)}$

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

Diese definiert eine Folge von Faktenmengen F_i durch

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

Fixpunkt-Semantik logischer Programme

Für definite Programme P:

- ist F_P^* der kleinste Fixpunkt des Operators T_P.
- gilt $F_P^* = \bigcap \mathrm{Mod}(P)$
- ist F_P^* das eindeutige kleinste Modell für P.
- Falls $F_P^{n+1} = F_P^n$ gilt, dann ist $F_P^* = F_P^n$.
- Für endliche (grundinstanziierte) Programme P wird $F_P^* = F_P^n$ nach endlich vielen Anwendungen von T_P erreicht.

Fixpunkt-Semantik des logischen Programmes P:

- Ein Atom a folgt genau dann aus P, wenn $a \in F_P^*$.
- Eine Formel φ folgt genau dann aus P, wenn φ in F_P^* gilt.

Schließen in klassischer Logik

Für eine Formelmenge $\Phi \subseteq \mathsf{FOL}(\Sigma, \mathbb{X})$ heißt die Formelmenge

$$C(\Phi) = \{\psi \in \mathsf{FOL}(\Sigma, \mathbb{X}) \mid \Phi \models \psi\}$$

Menge aller Konsequenzen aus Φ.

Formelmenge Φ mit $\Phi = C(\Phi)$ heißt deduktiv abgeschlossen.

In klassischer Logik gilt:
Aus $\Phi \subseteq \Psi$ folgt $C(\Phi) \subseteq C(\Psi)$.
Bei Erweiterung des Wissens bleiben alle Fakten, die vorher schon abgeleitet werden konnten, wahr.
(nur Erweiterung des Wissens, keine Revision)

Hülleneigenschaften

Ein Hüllenoperator ist ein Operator $f : 2^M \to 2^M$ 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

In klassischer Logik ist C ein Hüllenoperator.

Unvollkommenes Wissen

einige mögliche Quellen der Unvollkommenheit:

- 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 von Entscheidungen oft trotzdem möglich.

Beispiel

Wissen Φ:

- Vögel können fliegen. $(\forall x(V(x) \rightarrow F(x)))$
- Tweety ist ein Vogel. $(V(t))$

Frage: Kann Tweety fliegen? $(F(t))$

zusätzliches Wissen Ψ: Es gibt Vögel, die nicht fliegen können, z.B.

- Pinguine sind Vögel $(\forall x(P(x) \rightarrow V(x)))$
- Pinguine können nicht fliegen $(\forall x(P(x) \rightarrow \neg F(x)))$

Problem: $\Phi \cup \Psi$ inkonsistent (enthält Widerspruch)

Lösungsansatz: „unnormale“ Vögel

- $\forall x(V(x) \wedge \neg U(x) \rightarrow F(x))$
- $\forall x(P(x) \rightarrow U(x))$

neue Information: Tweety ist ein Pinguin $(P(t))$

Negative Voraussetzungen

Problem: Wann gilt $\neg p$ in einer Faktenbasis F?
verschiedene Ansätze:

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

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

Regeln mit negativen Bedingungen

Regeln der Form

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

mit

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

ist in der Faktenmenge F genau dann anwendbar, wenn

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

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

Nichtmonotones Schließen

Syntax: Wissensbasen mit negierten Atomen in Kopf und Rumpf

Problem beim Schließen mit Regeln mit negativen Bedingungen:

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

Ansätze zum Umgang mit unvollständigem Wissen

verschiedene Ansätze zur Definition einer intuitiven Semantik für Regelmengen mit negativen Voraussetzungen, z.B.:

- Stabile Modelle, Answer-Sets:
  Idee: Programm hat mehrere mögliche Modelle
  Aussage folgt aus Wissensbasis, wenn sie in einem / ausgewählten / allen Modellen wahr ist.
- Wohlfundierte Modelle
  Idee: Programm hat ein Modell mit drei Wahrheitswerten (wahr, falsch, unbekannt)
  Aussage folgt aus Wissensbasis, wenn sie in diesem Modell wahr ist.
- Truth-Maintenance-Systeme:
  Protokollierung aller zum Ableiten einer Formel verwendeten Voraussetzungen
  Bei späterer Feststellung der Unwahrheit einer Voraussetzung, alle daraus gezogenen Schlüsse löschen (rekursiv).

Normale logische Programme

(negative Voraussetzungen erlaubt)

(erweitertes) logisches Programm P (Wissensbasis) enthält:

- Menge R von Regeln der Form

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

  mit Atomen p_i, q_i, h

spezielle Regeln:

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

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

Modelle normaler logischer Programme

Idee: P als Formelmenge

Herbrand-Interpretation eines normalen logischen Programmes P:
Menge I von Grundatomen (mit derselben Signatur wie P)
betrachtet als Aussagevariablen
Belegung der Aussagevariablen ist charakteristische Funktion von I

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

Herbrand-Modell eines normalen logischen Programmes P:
Herbrand-Interpretation I mit $I \in \mathrm{Mod}(P)$
(Belegung = charakteristische Funktion)

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

- $\{p, q, r\}, \{p, q\}$ sind Modelle für P
- $\{q, r\}, \emptyset$ sind keine Modelle für P

Auswahl intuitiver Modelle

Eigenschaften von Interpretationen I eines 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: Modelle für P, die begründet und unter P abgeschlossen sind.

Gelfond-Lifschitz-Transformation

gegeben:

- normal logisches Programm P
- Modell I für P

Programmtransformation:

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

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

Für jedes normale logische Programm P und jede Interpretation I ist das I-Redukt P_I ein Programm ohne negative Bedingungen.

Der Konsequenzoperator T_{P_I} ist also monoton.

Stabile Modelle normaler logischer Programme

Idee: Auswahl einer Menge von intuitiven Modellen für normale logische Programme

Modell I für P heißt stabiles Modell, falls

$$I = T^*_{P^I}$$

Beispiel:

- $P = \{p \to q, \neg q \to r, \neg r \to q, p\}$
  - $\{p, q\}$ ist stabiles Modell für P, weil $P^{\{p,q\}} = \{p \to q, q, p\}$ und $T^*_{P^{\{p,q\}}} = \{p, q\}$
  - $\{p, q, r\}$ ist kein stabiles Modell für P, weil $P^{\{p,q,r\}} = \{p \to q, p\}$ und $T^*_{P^{\{p,q,r\}}} = \{p, q\}$
- $P' = \{\neg p \to q, \neg q \to p\}$
- $P'' = \{\neg p \to q, p \to q, \neg q \to p\}$
- $P''' = \{\neg p \to p\}$

Beispiel: gefärbte Graphen

Faktenbasis (Beschreibung des speziellen Problemes):

- 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)

Erweiterte logische Programme

(sowohl starke $\overline{\ }$ als auch schwache Negation $\neg$ erlaubt)

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

(erweitertes) logisches Programm P (Wissensbasis) enthält:

- Menge R von Regeln der Form

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

  mit „Atomen“ p_i, q_i, h

spezielle Regeln:

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

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

Answer Sets

Answer-Sets:
ausgewählte Modelle erweiterter logischer Programme

Eigenschaften:

- abgeschlossen unter P oder
  für ein Atom p gilt $\{p, \overline{p}\} \subseteq I$ (inkonsistent)
- begründet in P

Interpretation eines erweiterten logischen Programmes P:
Menge I von Grundliteralen (mit derselben Signatur wie P)

I ist Answer-Set für P gdw. Modell des I-Reduktes P^I ist (analog stabilen Modellen)

Beispiel: 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)
```

Reading Group

- Esra Erdem, Volkan Patoglu, 2018:
  Applications of ASP in Robotics
- Tran Cao, Marcello Balduccini, 2018:
  Answer Set Planning in Single- and Multi-agent Environments
- Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., Wanko, P. (2019):
  Train Scheduling with Hybrid ASP

  `https://www.cs.uni-potsdam.de/wv/publications/DBLP_conf/lpnmr/AbelsJOSTW19.pdf`

Was bisher geschah

Wissensrepräsentation und -verarbeitung durch

- Künstliche Neuronale Netze (insbes. auch CNN)
- Zustandsübergangssysteme
- Klassische Logiken
- Regelsysteme in klassischer Aussagen- und Prädikatenlogik
- Logische Programme (Prolog, Datalog)
- Nichtmonotonen Schließens bei unvollständigem Wissen (closed world assumption, schwache Negation)
- Beispiele zum Planen
- Answer Set Programming

Unsicheres Wissen

Problem bei Antworten auf Fragen (Wahrheit von Fakten), falls Wert

- unbekannt
- ungenau
- unsicher, unzuverlässig
- aus mehreren Quellen zusammengefügt, evtl. widersprüchlich
- genauere Untersuchung unmöglich, zeitaufwendig, teuer

Abhilfe z.B. durch:

- Wahrscheinlichkeiten
- Vermutungen, Annahmen
- Heuristiken: Erfahrungswerte, Schätzungen

Mehrwertige Logiken

Erweiterung der klassischen Logiken mit
Wahrheitswertbereich $\{0, 1\}$
auf andere Wahrheitswertbereiche

- endlich-wertige Logiken
  z.B. 3- und 4-wertige Logiken
- fuzzy Logiken
  meist Wahrheitswertbereich $[0, 1]$
- probabilistische Logiken
  meist Wahrheitswertbereich $[0, 1]$

Dreiwertige Logiken

Annahmen aus klassischen Logiken:

A1 Jede Aussage ist wahr oder falsch.

A2 Keine Aussage ist sowohl wahr als auch falsch.

Bei unvollständigem Wissen gilt A1 nicht.

Dreiwertigen Logiken enthalten deshalb einen zusätzlichen Wahrheitswert für „unbekannt“,
Wahrheitswertbereich meist $\{0, \bot, 1\}$ (auch $\{0, U, 1\}$, $\{0, 1/2, 1\}$)
mit zwei Ordnungen:

- Wahrheits-Ordnung: $0 <_W \bot <_W 1$ (total)
- Informations-Ordnung: $\bot <_I 0$ und $\bot <_I 1$ (partiell)

prominente dreiwertige Logiken,
z.B. von Belnap, Peirce, Łukasiewicz, Gödel, Kleene
unterscheiden sich in Wahrheitswertfunktionen der Junktoren

Beispiel: Dreiwertige Łukasiewicz-Logik

Jan Łukasiewicz and A. Tarski (1930):
Untersuchungen über den Aussagenkalkül
Semantik ist definiert über die Wahrheitswertfunktion der Implikation
Semantik im Wahrheitswertbereich $\{0, 1/2, 1\}$ (und auch in $[0, 1]$):

$$\begin{aligned} [\![\mathbf{f}]\!] &= 0 \\ [\![\varphi \to \psi]\!] &= \min(1, 1 - [\![\varphi]\!] + [\![\psi]\!]) = \begin{cases} 1 & \text{falls } [\![\varphi]\!] \leq [\![\psi]\!] \\ 1/2 & \text{falls } [\![\varphi]\!] - [\![\psi]\!] = 1/2 \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

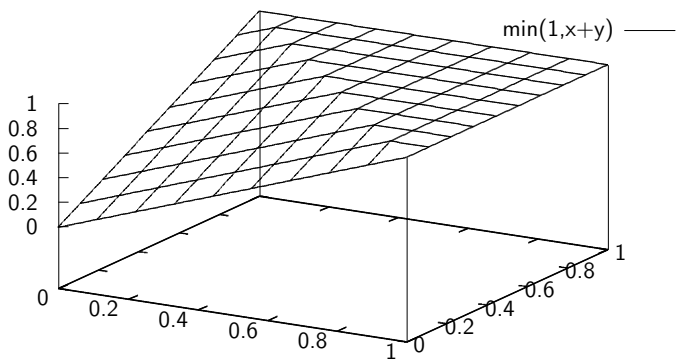
definierte Junktoren:

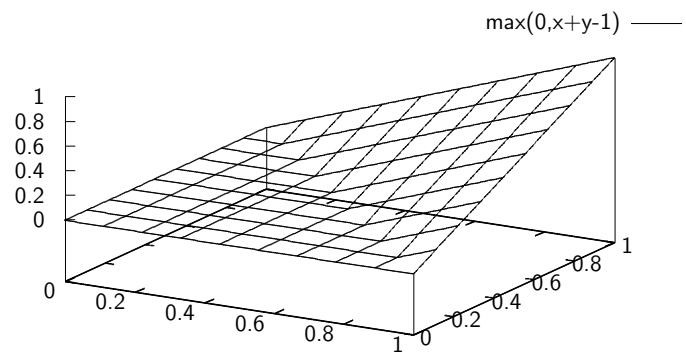
$$\begin{aligned} \neg\varphi &:= \varphi \to \mathbf{f} \\ \varphi \underline{\vee} \psi &:= \neg\varphi \to \psi && \text{(starke Disjunktion)} \\ \varphi \,\&\, \psi &:= \neg(\varphi \to \neg\psi) && \text{(starke Konjunktion)} \\ \varphi \vee \psi &:= (\varphi \to \psi) \to \psi && \text{(schwache Disjunktion)} \\ \varphi \wedge \psi &:= \varphi \,\&(\varphi \to \psi) && \text{(schwache Konjunktion)} \end{aligned}$$

Łukasiewicz-Logik: Wahrheitswertfunktionen

Aus der Definition der Junktoren lassen sich deren Wahrheitswertfunktionen berechnen:

$$
\begin{aligned}
\neg x &= 1 - x \\
x \to y &= \min(1, 1 - x + y) \\
x \vee y &= \max(x, y) \\
x \wedge y &= \min(x, y) \\
x \underline{\vee} y &= \min\{1, x + y\} \\
x \,\&\, y &= \max\{0, x + y - 1\} \\
x \leftrightarrow y &= 1 - |x - y|
\end{aligned}
$$





Semantik in dreiwertiger Łukasiewicz-Logik

Wahrheitswerttabellen:

$\neg$	0	1/2	1
	1	1/2	**0**

$\rightarrow$	0	1/2	1
0	**1**	1	**1**
1/2	1/2	1	1
1	**0**	1/2	**1**

$\wedge$	0	1/2	1
0	**0**	0	**0**
1/2	0	1/2	1/2
1	**0**	1/2	**1**

$\vee$	0	1/2	1
0	**0**	1/2	**1**
1/2	1/2	1/2	1
1	**1**	1	**1**

$\&$	0	1/2	1
0	**0**	0	**0**
1/2	0	0	1/2
1	**0**	1/2	**1**

$\underline{\vee}$	0	1/2	1
0	**0**	1/2	**01**
1/2	1/2	1	1
1	**1**	1	**1**

Mehrwertige Łukasiewicz-Logik

Die Wahrheitswertfunktionen der dreiwertigen Łukasiewicz-Logik sind auf dem ganzen Intervall $[0, 1]$ (und Teilmengen davon) definiert.
Semantik in $[0, 1] \subseteq \mathbb{R}$ (bzw. geeigneten Teilmengen davon)

$$\begin{aligned} [\![\mathbf{f}]\!] &:= 0 \\ [\![\varphi \to \psi]\!] &:= \min(1, 1 - [\![\varphi]\!] + [\![\psi]\!]) \end{aligned}$$

Definition der abgeleiteten Junktoren wie in der dreiwertigen Łukasiewicz-Logik.

zweiwertige Łukasiewicz-Logik:
Bei Einschränkung der Wahrheitswertfunktionen auf die Menge $\{0, 1\}$

- stimmen die Werte für schwache und starke Konjunktion überein,
- stimmen die Werte für schwache und starke Disjunktion überein,
- ergibt sich genau die klassische zweiwertige Logik.

Klassische zweiwertige Logik ist also ein Spezialfall
(= zweiwertige Łukasiewicz-Logik)

Mehrwertige (Aussagen-)Logiken

Aussagenlogik $\mathrm{PL}(P, \mathbb{W})$ mit Parametern

P Aussagenvariablen

$\mathbb{W}$ Wahrheitswert-Bereich (algebraische Struktur)
$\mathbb{W} = (W, \ldots)$
meist mit $W \subseteq [0, 1]$

mit Symbolen und Wahrheitswertfunktionen für

- ein- und mehrstellige Junktoren, z.B. $\neg, \vee, \wedge, \ldots$
- nullstelligen Junktoren (Wahrheitswertkonstanten) für jedes Element (einer Teilmenge) von $\mathbb{W}$, wenigstens aber 0 und 1

für Prädikatenlogiken außerdem zu definieren:
Symbole und Wahrheitswertfunktionen für Quantoren

Mehrwertige Łukasiewicz-Logik(en)

Syntax:

Wahrheitswertkonstanten syntaktische Repräsentanten der Elemente in $\mathbb{W}$

Aussagenvariablen P

Junktoren $\neg, \&, \underline{\vee}$ (stark), $\vee, \wedge$ (schwach)
Wahrheitswertkontanten $c \in W$ (nullstellig),

Formeln $\varphi ::= p \mid \neg\varphi \mid \varphi * \psi \mid c$
mit Aussagenvariablen $p \in P$,
$* \in \{\&, \underline{\vee}, \vee, \wedge\}$, Formeln φ, ψ und
Wahrheitswertkontanten $c \in W$

NNF $\varphi ::= p \mid \neg p \mid \varphi * \psi \mid c$

Semantik von Formeln definiert entsprechend der Wahrheitswertfunktionen auf $\mathbb{W}$

Fuzzy-Logiken

Fuzzy-Logik: Sammelbegriff für verschiedene Logiken,
meist mit

- Wahrheitswertbereich $[0, 1] \subseteq \mathbb{R}$ (oder Teilmenge davon)
- $\neg x \mapsto 1 - x$
- Wahrheitswertkonstanten 0 und 1

und definiert über Wahrheitswertfunktionen für andere Junktoren,
oft $\wedge$, $\vee$ oder $\rightarrow$

Beispiele:

- Standard-Fuzzy-Logik: $\wedge \mapsto \min$, $\vee \mapsto \max$
- Produkt-Logik: $x \wedge y \mapsto xy$
- Łukasiewicz-Logik: $x \rightarrow y \mapsto \min(1, 1 - x + y)$

Was bisher geschah

Wissensrepräsentation und -verarbeitung durch

- Künstliche Neuronale Netze (insbes. auch CNN)
- Zustandsübergangssysteme
- Klassische Logiken
- Regelsysteme in klassischer Aussagen- und Prädikatenlogik
- Logische Programme (Prolog, Datalog)
- Nichtmonotonen Schließens bei unvollständigem Wissen (closed world assumption, schwache Negation)
- Beispiele zum Planen
- Answer Set Programming
- Mehrwertige Logiken, z.B. dreiwertige Łukasiewicz-Logik

Fuzzy-Logiken

Fuzzy-Logik: Sammelbegriff für verschiedene Logiken,
meist mit

- Wahrheitswertbereich $[0, 1] \subseteq \mathbb{R}$ (oder Teilmenge davon)
- $\neg x \mapsto 1 - x$
- Wahrheitswertkonstanten 0 und 1

und definiert über Wahrheitswertfunktionen für andere Junktoren,
oft $\wedge$, $\vee$ oder $\rightarrow$

Beispiele:

- Standard-Fuzzy-Logik: $\wedge \mapsto \min$, $\vee \mapsto \max$
- Produkt-Logik: $x \wedge y \mapsto xy$
- Łukasiewicz-Logik: $x \rightarrow y \mapsto \min(1, 1 - x + y)$

Vierwertige Logik

sinnvoll z.B. zum Umgang mit widersprüchlichen Informationen

Beispiel: parakonsistente Logik von Belnap (1977)

Wahrheitswertbereich $\{0, \bot, \top, 1\}$ (auch $\{0,1\}^2$)

mit zwei Ordnungen (beide partiell):

- Wahrheits-Ordnung: $0 <_W \bot <_W 1$ und $0 <_W \top <_W 1$
- Informations-Ordnung: $\bot <_I 0 <_I \top$ und $\bot <_I 1 <_I \top$

Wahrheitswertfunktionen:

$\neg$	$\bot$	0	1	$\top$
	$\bot$	1	0	$\top$

$\wedge$	$\bot$	0	1	$\top$
$\bot$	$\bot$	0	$\bot$	0
0	0	0	0	0
1	$\bot$	0	1	$\top$
$\top$	0	0	$\top$	$\top$

$\vee$	$\bot$	0	1	$\top$
$\bot$	$\bot$	$\bot$	1	1
0	$\bot$	0	1	$\top$
1	1	1	1	1
$\top$	1	$\top$	1	$\top$

Mehrwertige Mengen und Relationen

Übergang von zweiwertigen zu mehrwertigen Mengen:

- Menge $M \subseteq U$ mit
  charakteristischer Funktion $\chi_M : U \to \{0, 1\}$
- Mehrwertige Menge über Wahrheitswertbereich W
  Funktion $M : U \to W$
  ordnet jedem $x \in U$ einen Wahrheitswert
  (Zugehörigkeitsgrad) zu

Relation: Menge $R \subseteq A \times B$ von Paaren (Tupeln)
Übergang von zweiwertigen zu mehrwertigen Prädikaten
(Relationen):

- Relation $R \subseteq A \times B$ mit
  charakteristischer Funktion $\chi_R : A \times B \to \{0, 1\}$
- Mehrwertige Relation über Wahrheitswertbereich W
  Funktion $R : A \times B \to W$
  ordnet jedem Paar $(a, b) \in A \times B$ einen Wahrheitswert zu

Erinnerung: Eigenschaften sind einstellige Relationen (Mengen).

Unsichere Regelsysteme

Ansätze:

1. Unsicherheit in den Daten (Wahrheitswerte an Fakten)
   Regel $l_1 \wedge \ldots \wedge l_n \rightarrow h$
   angewendet auf Voraussetzungen l_i, je mit Wahrheitswert w_i
   ordnet h den Wert $f(w_1, \ldots, w_n)$ zu
2. Unsicherheit in den Regeln (Wahrheitswerte an Regeln)
   Regel $l_1 \wedge \ldots \wedge l_n \rightarrow_w h$
   angewendet auf Voraussetzungen l_i
   ordnet h den Wert w zu, falls alls l_i erfüllt sind
3. Kombination aus beiden

Beispiel

ProbLog
(`https://dtai.cs.kuleuven.be/problog/index.html`)

Fakten mit Wert:

```
0.5::heads1.
0.6::heads2.
```

Regeln:

```
twoHeads :- heads1, heads2.
```

Anfragen:

- `heads1 = 0.5`
- `heads2 = 0.6`
- `twoHeads = 0.3`

Beispiel

Fakten mit Wert:

```
0.5::heads1.
0.6::heads2.
```

Regeln:

```
someHead :- heads1.
someHead :- heads2.
```

Anfragen:

- `heads1 = 0.5`
- `heads2 = 0.6`
- `someHead = 0.8`

Beispiel

Fakten und Regeln mit Wert:

```
0.3::stress(X) :- person(X).
0.2::influences(X,Y) :- person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :- friend(X,Y), influences(Y,X), smokes(Y).

0.4::asthma(X) :- smokes(X).
```

WH: Wahrscheinlichkeiten

Begriffe:

- Zufalls-Experiment
- (endlicher) Wahrscheinlichkeitsraum $(\Omega, 2^\Omega, P)$
- Elementar-Ereignis
- zufälliges Ereignis

Beispiele:

- Experiment: dreimal würfeln,
- Ereignis V: Augenzahlen sind paarweise verschieden,
- Elementar-Ereignisse: $\{(x, y, z) \mid x, y, z \in \{1, \ldots, 6\}\}$
- $P(V)$ bei Gleichverteilung?

WH: Bedingte Wahrscheinlichkeiten

Definition:
Bedingte Wahrscheinlichkeit von Ereignis A unter Ereignis B:

$$P(A \mid B) = P(A \cap B)/P(B)$$

Beispiele:

- zwei Würfel, $A =$ Augensumme ist > 7,
  $B =$ beide Zahlen sind ungerade.
- B eine Ursache (für Fehler, Krankheit, usw.),
  A eine Auswirkung (Symptom) (leichter zu beobachten)

Unterschied zu bisher betrachteten Regelsystemen:

- bisher: Aussagen über Wahrheit (von Aussagen)
- jetzt: Aussagen über Wahrscheinlichkeit (von Ereignissen)

Satz von Bayes

Satz von Bayes (einfache Form):

$$P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A)$$

Beweis: Def. von $P(X \mid Y)$ einsetzen, vereinfachen.

Anwendung: Rechnen mit bedingten Wahrscheinlichkeiten

- 1/3 aller Studenten haben ein Notebook.
- 1/10 aller Studenten studieren Informatik.
- 9/10 aller Informatik-Studenten haben ein Notebook.
- Sie sehen einen Studenten mit einem Notebook.
- Mit welcher Wahrscheinlichkeit studiert er Informatik?

Das ist ein Beispiel für probabilistische Inferenz.
wird verallgemeinert auf längere Ketten von
Ursache-Wirkung-Beziehungen

WH: Unabhängige Ereignisse

Def: Ereignisse A, B heißen (stochastisch) unabhängig, falls $P(A \cap B) = P(A) \cdot P(B)$.

Satz: $P(B) > 0 \Rightarrow (A \text{ und } B \text{ unabh.} \iff P(A \mid B) = P(A))$.

Bsp:
zwei Würfel, $A =$ Augensumme > 7, $B =$ beide Zahlen ungerade.
A und B sind *nicht* unabhängig.

Def: Nicht unabhängige A, B heißen *korreliert*.

Vorsicht: das bedeutet nicht,
dass A die Ursache für B ist, oder B die für A.
Es könnte z.B. eine gemeinsame Ursache C für A und B geben.
(correlation does not imply causation)

Beispiele:

- $A =$ schweres Fahrzeug, $B =$ hoher Verbrauch,
  $C =$ unwegsames Gelände
- $A =$ geringes Geburtsgewicht, $B =$ hohe Säuglingssterblichkeit, $C =$ starkes Rauchen

Diskrete Zufallsgrößen

- Def: Zufallsgröße ist Funktion $X : \Omega \to$ endl. Menge ($\subseteq \mathbb{R}$)
- einfachster Fall: $\Omega = \{0,1\}^k$

  $X_k = (\vec{x} \mapsto \vec{x}_k)$ (die k-te Komponente)
- dann Wsk-Raum bestimmt durch Wsk der Elementar-E.,

  Bsp: $P(0,0) = 1/3, P(0,1) = 1/6, P(1,0) = 0, P(1,1) = 1/2$
- (Motivation für Bayes-Netz: beschreibt solchen Wsk-Raum durch deutlich weniger als 2^k Parameter)
- zu Zufallsgröße X betrachte Ereignis $X = e$,

  Bsp (Fortsetzung): $P(X_1 = 0 \cap X_2 = 1) = 1/6$.

  $P(X_2 = 1) = 1/6 + 1/2 = 2/3$, $P(X_1 = 0) = \ldots$
- Def. Zufallsgrößen X, Y sind unabhängig:
  jedes $X = e$ ist unabhängig von jedem $Y = f$

Kausal-Diagramme

Kausal-Diagramm: DAG

- Knoten: Sachverhalte
- Kanten: (vermutete) kausale (ursächliche) Beziehungen

Beispiel:

- Knoten: Winter, glatt, Tom betrunken, Unfall Tom / Jerry

Verbindungsmuster:

- seriell: $W \to G \to U$
- teilend: $G \to T$, $G \to J$
- zusammenführend: $B \to T$, $G \to T$

Bayes-Netze: Motivation, Definition

- Bayes-Netz (alternativ: *believe network*) ist DAG
  - Knoten: Zufallsvariablen
  - Kanten: (vermutete) kausale (ursächliche) Beziehungen
- Anwendung: probabilistisches Schließen,
  Bestimmung wahrscheinlicher Ursachen für Symptome
- BN erfunden von Judea Pearl, erhielt (u.a.) dafür den *ACM Turing Award* 2011,

  `https://amturing.acm.org/award_winners/pearl_2658896.cfm`
- benannt nach Thomas Bayes (1701–1761),
  Satz von Bayes über bedingte Wahrscheinlichkeiten

Definition Bayes-Netz

- Syntax: ein Bayes-Netz N ist ein Paar (G, Θ) mit
  - G ist DAG, Knoten sind Zufallsgrößen
  - Θ: für jeden Knoten X mit Eltern $X_1, \ldots, X_k$: Wahrscheinlichkeiten $P(X = e \mid X_1 = e_1 \cap \cdots \cap X_k = e_k)$ für alle $[e, e_1, \ldots, e_k] \in W^{k+1}$
- Semantik: N beschreibt Wahrscheinlichkeitsraum durch
  $$P(X = e) = P(X = e \mid \ldots X_k = e_k \ldots) \cdot \prod_k P(X_k = e_k)$$
  induktive Definition:
  IA: Quellen des DAG (ohne Vorgänger, d.h., ohne Bedingungen, d.h., $\prod \emptyset = 1$)

Beispiel Bayes-Netz

(nach Judea Pearl)

- Knoten: Einbruch R, Erdbeben E, Alarmanlage A (zuhause), John ruft (auf Arbeit) an J, Mary ruft an M.
- Kanten mit Parametern (Bsp)
  - $P(R = 1) = 0.001$, $P(E = 1) = 0.002$
  - $P(A = 1 \mid R = 0, E = 1) = 0.29, \ldots$

Graphische Darstellung: Tafel

Bedingte Unabhängigkeit und BN

- (Wdhlg.) Def A und B unabhängig, falls $P(A \cap B) = P(A) \cdot P(B)$.
- Def: A und B bedingt unabhängig bezüglich C: $P(A \cap B \mid C) = P(A \mid C) \cdot P(B \mid C)$. (Vorstellung: wir schränken den Wsk-Raum ein auf die Elementar-Ereignisse aus C, verwenden dort die Standard-Def. der Unabh.)
- Def: bedingte Unabh. von (diskreten) Zufallsgrößen entsprechend
- Satz: für jedes BN N, für alle $X, Y \in N$ mit $X \not\rightarrow_N^* Y$: X und Y sind bedingt unabh. bezüglich der Eltern von X.

Inferenz mit BN

- die Diagnose-Aufgabe: gegeben ein BN, gesucht sind bedingte Wahrscheinlichkeiten der Ursache(n),
  unter der Bedingung von Beobachtungen
- Bsp: $P(\text{Einbruch} = 1 \mid \text{John} = 1 \cap \text{Mary} = 1)$
- Bsp: $P(\text{Einbruch} = 1 \mid \text{John} = 1 \cup \text{Mary} = 1)$
- kann exakt bestimmt werden, dauert jedoch $2^{|N|}$
  kann nicht besser gehen, weil aussagenlogische Erfüllbarkeit
  auf dieses Inferenzproblem reduziert werden kann
- die Alternative sind schnellere (Simulations)Verfahren,
  die einen Näherungswert liefern

Reading Group KW 26

Judea Pearl (2018):
The Seven Tools of Causal Inference with Reflections on Machine Learning

`https://ftp.cs.ucla.edu/pub/stat_ser/r481.pdf`

Was bisher geschah

Wissensrepräsentation und -verarbeitung durch

- Künstliche Neuronale Netze (insbes. auch CNN)
- Zustandsübergangssysteme
- Klassische Logiken
- Regelsysteme in klassischer Aussagen- und Prädikatenlogik
- Logische Programme (Prolog, Datalog)
- Nichtmonotonen Schließens bei unvollständigem Wissen (closed world assumption, schwache Negation)
- Beispiele zum Planen
- Answer Set Programming
- Mehrwertige Logiken, z.B. dreiwertige Łukasiewicz-Logik, Fuzzy-Logiken
- probbabilistisches Schließen, Bayes-Netze
- Motivation Kausale Inferenz (RG)

Kausale Hierarchie (WH RG)

Korrelation von Daten entsprechen nicht notwendig kausalen Zusammenhängen.

3 Schichten:

1. Beobachtung $P(x|y)$
2. Intervention $P(x|\text{do}(y), z)$
   bedingte Wahrscheinlichkeit von $X = x$ unter der Bedingung, dass $Y = y$ gesetzt (würde) und $Z = z$ beobachtet wird
3. Counterfactuals $P(y_x|x', y')$

Kausales Modell

- Menge U von äußeren Variablen
  (außerhalb des Modells, beeinflussen aber Zusammenhänge innerhalb des Modells)
- Menge $V = \{V_1, \ldots, V_n\}$ von beobachteten inneren Variablen
  wobei jedes V_i von einer Menge $A_i \subseteq U \cup V \setminus \{V_i\}$ abhängt
- Menge von Funktionen $F = \{f_1, \ldots, f_n\}$ mit $v_i = f_i(a_i, u)$
- gemeinsame Wahrscheinlichkeitsverteilung $P(u)$ über U

Kausal-Diagramm: DAG G,

- Knoten $U \cup V$
- Kanten $E \subseteq (U \cup V) \times V$ mit
  $\forall W \in (U \cup V)\ \forall i \in \{1, \ldots, n\} : (W, V_i) \in E \leftrightarrow V \in A_i$

d-Separation

Aus Eigenschaften (Teilgraphen) des DAG G lässt sich Unabhängigkeit von Variablenmengen A, B herleiten:

A und B sind d-separiert gdw. für jeden (ungerichteten) Pfad Q von A nach B (wenigstens) eine der folgenden Bedingungen gilt:

- Q enthält Kette (Teilgraph $u \rightarrow v \rightarrow w$) mit Beobachtung v
- Q enthält Verzweigung (Teilgraph $u \leftarrow v \rightarrow w$) mit Beobachtung v
- Q enthält Zusammenführung (Collider, Teilgraph $u \rightarrow v \leftarrow w$)

A und B sind d-separiert ($A \perp\!\!\!\perp B | C$) unter Voraussetzung C gdw.

$$P(A, B|C) = P(A|C)P(B|C)$$

Interventionen

Idee: $P(Y = y|\text{do}(X = x))$ kann oft nicht experimentell bestimmt werden (unethisch oder aufwendig)

do-Kalkül: Regelsystem zur (schrittweisen) Transformation von Wahrscheinlichkeiten mit do in bedingte Wahrscheinlichkeiten

Aktion $\text{do}(X = x)$

- beeinflusst das kausale Modell (DAG)
  $M \mapsto M_x$
- ordnet der Zufallsvariablen X den festen Wert x zu
- Löschen aller Eingangskanten zu X
- Wahrscheinlichkeitsverteilung nach Intervention:
  $P_M(y|\text{do}(X = x)) = P_{M_x}(y)$

Diagramme zum Löschen von Ein- und Ausgängen $G_{\overline{X}}$, $G_{\underline{X}}$ (Tafel)

do-Kalkül (Pearl, 1995)

3 Regeln des do-Kalkül:
für disjunkte Variablenmengen X, Y, Z, W im DAG G

- Beobachtung (Z) ignorieren / einführen
  falls $(Y \perp\!\!\!\perp Z | X, W)$ in $G_{\overline{X}}$ (G mit gelöschten X-Eingängen):

  $$P(Y = y | \text{do}(X = x), Z = z, W = w) = P(Y = y | \text{do}(X = x), W = w)$$

- Aktion / Beobachtung (Z) tauschen (back-door-Kriterium)
  falls $(Y \perp\!\!\!\perp Z | X, W)$ in $G_{\overline{X}\underline{Z}}$
  (G mit gelöschten X-Ein- und Z-Ausgängen):

  $$\begin{aligned} & P(Y = y | \text{do}(X = x), \text{do}(Z = z), W = w) \\ = \quad & P(Y = y | \text{do}(X = x), Z = z, W = w) \end{aligned}$$

- Aktion ($\text{do}(Z = z)$) ignorieren / einführen
  falls $(Y \perp\!\!\!\perp Z | X, W)$ in $G_{\overline{X}\overline{Z(W)}}$
  (G mit gelöschten X- und $Z(W)$-Eingängen,
  $Z(W)$ = Menge aller Knoten in Z, die keine Vorfahren von W sind):

  $$\begin{aligned} & P(Y = y | \text{do}(X = x), \text{do}(Z = z), W = w) \\ = \quad & P(Y = y | \text{do}(X = x), W = w) \end{aligned}$$

Motivation Regel 1: Beobachtungen ignorieren

falls $(Y \perp\!\!\!\perp Z | X, W)$ in $G_{\overline{X}}$:

$$P(Y = y | \text{do}(X = x), Z = z, W = w) = P(Y = y | \text{do}(X = x), W = w)$$

Spezialfälle:

- $W = X = \emptyset$:
  $(Y \perp\!\!\!\perp Z)$ in $G_{\overline{X}} = G$ (Y und Z unabhängig),
  also $P(Y = y | Z = z) = P(Y = y)$
- (passive) Beobachtung $W \neq \emptyset$ und $X = \emptyset$:
  $(Y \perp\!\!\!\perp Z | W)$ in $G_{\overline{X}} = G$, Y und Z d-separiert,
  also $P(Y = y | Z = z, W = w) = P(Y = y | W = w)$
- keine Beobachtung $W = \emptyset$, aber Intervention $\text{do}(X = x)$:
  $(Y \perp\!\!\!\perp Z)$ in $G_{\overline{X}}$,
  also $P(Y = y | \text{do}(X = x), Z = z) = P(Y = y | \text{do}(X = x))$

Regel 1 oben ist Kombination dieser Fälle

Ableitungen

Ableitung im do-Kalkül auf eine Anfrage Q:
schrittweise Umformung von Q durch die Regeln, bis Ausdruck kein $\text{do}(x)$ mehr enthält

Ergebnis bei erfolgreicher Ableitung:
Schätzfunktion für Q anhand der beobachteten Daten

Modul Künstliche Intelligenz (Wissensrepräsentation und -verarbeitung)

Lernziele/Kompetenzen:
Bitte ersetzen Sie an den Stellen in Punkt 5, an denen Sie mich zitieren,
ëine statische künstliche Intelligenz"durch ëin vordefiniertes Standard-Verhalten"
und
ïst die statische KI zu unsicher"durch ßind vordefinierte Reaktionen nicht ausreichend"
(vordefiniertes Verhalten ist ja keine KI)
Die Studierenden sind in der Lage, Wissensrepräsentationen zur Modellierung zu benutzen, die über klassische Prädikatenlogik hinausgeht.
Insbesondere können sie dem Problem angemessene Wissensverarbeitungstechniken zur Simulation intelligenten Verhaltens auswählen.
Sie verstehen aktuelle Fachbeiträge und können eine verständliche Präsentation der dort vorgestellten Ansätze ausarbeiten und

Modul Wissensrepräsentation und -verarbeitung

Lehrinhalte:

Aktuelle Themen auf dem Gebiet der Wissensverarbeitung und künstlichen Intelligenz mit jährlich wechselnden Schwerpunkten, Sommersemester 2019:

- Einteilung symbolische / statistische KI
- Künstliche neuronale Netze (stat)
- Heuristische Suche (symb /stat)
- logische Programmierung (symb)
- nichtmonotones Schließen (symb)
- mehrwertige Logiken (symb)
- Bayes-Netze (symb /stat)
- Modellierung von Kausalität (symb /stat)
- Modellierung ethischer Prinzipien (symb /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 INB)
- Inhalt:
  - Vorlesungsinhalt
  - Aufgabentypen wie Übungsaufgaben
  - Inhalt der Artikel der Reading Group
- Prüfungsvorleistung Beleg (PVB):
  aktive Mitarbeit in der Reading Group
  (haben alle Teilnehmer bestanden)
- (ausschließlich) zulässiges Hilfsmittel:
  A4-Blatt (beidseitig) handbeschrieben

KI – allgemein

- Daten, Information, Wissen, Intelligenz
  explizites und implizites Wissen
- KI-Geschichte
- starke/ schwache KI
- Turing-Test, Chinese-Room-Test
- symbolische / statistische Ansätze

Maschinelles Lernen

- überwacht
  - korrigierend
  - bestärkend (reinforcement)
- unüberwacht

Künstliche Neuronen

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

Künstliche Neuronale Netze

- Schichten-Struktur
- Ein-, Mehr-Schicht-FFN
- rekurrente Netze
- Cognitron (Faltungs-Netz, Bild-Pyramiden)
- CNN
- Lernverfahren / Training
- Anwendungen

Zustandsübergangssysteme

Wissensrepräsentation: Darstellung von

- **Problem:** Zustandsübergangssysteme, d.h.
  Graphen mit Ecken (Zuständen) und Kanten (Übergänge)
  Zustände charakterisiert durch Eigenschaften
  Startzustände, Eigenschaften der Zielzustände
- **Lösung:** Zielzustand,
  Weg von einem Start- zu einem Zielzustand

Wissensverarbeitungsverfahren: Suche in Graphen
(Breiten-, Tiefen-, heuristische Suchverfahren)

Heuristische Suche in Graphen

Standard-Suchalgorithmus
Verwaltung der Menge der noch nicht erledigten Knoten bestimmt die Besuchsreihenfolge der Knoten

Schätzfunktionen

Eigenschaften von Schätzfunktionen:
perfekt, zielerkennend, sicher, konsistent, nicht-überschätzend

Suchverfahren:

uninformiert: Beiten-, Tiefensuche

informiert: Greedy-, Besten-, heuristische, A*-Suche

Spielbäume

- 2-Personen-Nullsummen-Spiele
- Gewinnstrategien
- Minimax-Werte
- α-β-Suche

Logische Programmierung

- aussagenlogische Resolution
- Unifikation
- prädikatenlogische Resolution
- (definite) logische Programme
- Grundinstanziierung
- Seantik, Modelle, minimale Modelle
- Prolog
  Beispiele: Wissensrepräsentation, Planen
- Datalog
- Konsequenzoperator

Schließen bei unvollständigem Wissen

- Modellierung unvollständigen Wissens
- Closed World Assumption
- starke und Default-Negation
- normal logische Programme, ASP
- Konsequenzoperator
- stabile Modelle
- Answer Sets
- leichtgläubiges / skeptisches Schließen

Schließen bei unpräzisem Wissen

Mehrwertige Logiken:

- dreiwertig: Łukasiewicz, fuzzy
- vierwertige Logik von Belnap
- mehrwertig: Łukasiewicz, fuzzy

- Probabilistische Ansätze
- Bayes-Netze

Modellierung von Kausalität

- Kausale Hierarchie (3 Stufen)
  beobachtung, Intervention, Counterfactual
- Kausal-Diagramm
- Verbindungsmuster:
  seriell, verzweigend, zusammenführend
- Modellierung von Interventionen:
  do-Kalkül

Modellierung ethischer Prinzipien

Emmanuelle-Anna Dietz Saldanha, Steffen Hölldobler, Sibylle Schwarz and Lim Yohanes Stefanus:
The Weak Completion Semantics and Equality
LPAR-22: Logic for Programming, AI Reasoning, 2018
`https://easychair.org/publications/paper/qbws`

- logische Programme
- Zustandsübergangssysteme, Pläne
- dreiwertige Łukasiewicz-Logik
- Konsequenzoperator
- Repräsentation von Counterfactuals