Künstliche Intelligenz

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

Sommersemester 2022

Was ist Künstliche Intelligenz?

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

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

Aussage über das derzeitige (eingeschränkte) Verständnis von KI

Können Maschinen denken?

Alan Turing 1950

Konkretisierung der Frage:
Können Maschinen denken?

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

Imitation Game

Imitation Game (Alan Turing 1950):

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

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

Wie erkennt man Intelligenz: Turing-Test

Turing-Test 1950: verschiedene Versionen des Imitation Game

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

Vorschlag zur Bewertung natürlichsprachlicher Kommunikationsfähigkeiten

Beginn koordinierter Forschung zur Künstlichen Intelligenz

John McCarthy	Marvin Minsky	Claude Shannon
Programmiersprachen	Kognitionswissenschaft	Informationstheorie

stellten 1955 die Vermutung auf, dass

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

Begriff Künstliche Intelligenz

McCarthy formulierte das Ziel,

„herauszufinden, wie man Machinen konstruiert, die

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

und prägte dafür den Begriff Künstliche Intelligenz.

Beginn koordinierter Forschung zur Künstlichen Intelligenz

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

Themen:

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

Forschung zur Künstlichen Intelligenz

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

Themen:

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

KI-Erfolge – Auswahl

- 1945 frühe Schachprogramme (ohne Implementierung)
- 1955 Logic Theorist: automatischer Beweiser
- 1958 erster erfolgreicher Neurocomputer Mark I Perceptron
- 1961 General Problem Solver, z.B. zum Lösen von Rätseln und Intelligenztests
- 1966 Eliza: Chatbot
- 1972 erster mobiler Roboter
- ab ca. 1970 Beschränkung auf spezialisierte Expertensysteme
- 1976 MYCIN (Medizinisches Diagnosesystem)
- 1980 Dendral (Molekülstruktur aus Massenspektrogramm)
- 1982 XCON (Konfiguration von Computersystemen)
- ab ca. 1980 Expertensystem-Shells
- seit 1993 RoboCup Roboter-Fußball
- 1997 Deep Blue gewinnt gegen amtierenden Weltmeister
- 2011 Watson schlägt zwei Meister in Quizshow Jeopardy!
- 2016 AlphaGo schlägt Go-Meister
- automatisiertes Fahren (seit 1992)
-

Phasen in der KI-Geschichte

wechselnde Betonung symbolischer und statistischer KI-Gebiete

- ca. 1950-70
  symbolisch: Inferenz-Maschinen, Eliza, Lisp
  statistisch: KNN
  Robotik, Verarbeitung natürlicher Sprache
- ca. 1970-85 (symbolisch)
  Prolog, Expertensysteme (z.B. medizinisch)
  nichtmonotones Schließen
- ca. 1985-2000 (statistisch)
  maschinelles Lernen, KNN, evolutionäre Alg.,
  Schwarm-Intelligenz, (Fuzzy-Logik) ...
  autonome Fahrzeuge (Ernst Dickmanns)
- ca. 2000-2010 (symbolisch)
  Constraint-Programmierung
  SAT-Solver, Theorem-Prover
  Ontologien (Semantic Web), Beschreibungslogiken
- seit ca. 2010 (statistisch)
  Deep Learning, CNN

Ansätze intelligenter Systeme

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

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

Kritik am Turing-Test

Kritik:

schwache KI genügt, um den Turing-Test zu bestehen

1966: Maschinelle Psychotherapeutin Eliza besteht Turing-Test

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

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

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

außerdem: praktisch nicht umsetzbar (Datenmenge)

Logische / regelbasierte KI-Methoden

Wissensrepräsentation: formale Beschreibung von Umwelt (Randbedingungen) und Problem

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

Beispiele:

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

typische Anwendungen klassischer KI-Methoden:

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

Statistische KI-Methoden

„Soft-Computing“ oft besser geeignet für Probleme

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

einige Ansätze:

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

Aktuelle Entwicklung

starker Fortschritt einiger KI-Methoden („Deep Learning“)
in den letzten 10 Jahren aufgrund der Entwicklung bei

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

sowie starkem Medieninteresse an bestimmten Erfolgen, z.B.

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

führte zum aktuellen Aufflammen der KI-Euphorie

Leistung aktueller (statistischer) KI-Systeme

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

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

prinzipielle Herausforderungen:

- Zuverlässigkeit, Sicherheit
- Begründung, Erklärung

Schwächen aktueller (statistischer) KI-Systeme

KI derzeit noch weit von menschlichen Fähigkeiten entfernt bzgl.

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

Einordnung in die Informatik

Informatik Wissenschaft von der Darstellung und Verarbeitung symbolischer Information durch Algorithmen

Einordnung in die Teilgebiete der Informatik:

theoretisch
- Sprachen zur Formulierung von Information und Algorithmen,
- Berechenbarkeit durch Algorithmen,

Grundlagen, z.B. Logik, formale Sprachen

technisch
- maschinelle Darstellung von Information
- Mittel zur Ausführung von Algorithmen

Parallelrechner, GPU, Anwendung z.B. in HW-Verifikation, technischer Diagnose

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

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

Organisation

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

- wöchentlich
  - eine Vorlesung
  - Seminar (Fachbeiträge)
  - Übungsaufgaben
- Prüfungsvorleistung: Beleg
  - Referate zu passenden Fachartikeln
  - Präsentation der Lösung der Übungsaufgaben
- Prüfung: Klausur 90 min

Inhalt der Lehrveranstaltung

KI allgemein

- Heuristische Suche / Spielbaumsuche
- Künstliche Neuronale Netze
- (mehrwertiges, unscharfes, probabilisisches) Schließen

Gemeinsames Spezialthema für Sommersemester 2022:
Auswahl aus den Folgenden:

- Kausalität, Kausale Inferenz
  Pearl: The Book of Why
- Menschliches Schließen, Kognition, Mentale Modelle
  Stenning, van Lambalgen:
  Human Reasoning and Cognitive Science
- Menschliches Schließen, Wissen über Wissen
  van Benthem, van Ditmarsch, van Eijck, Jaspars:
  Logic in Action

Literatur

Folien, Aufgaben, ... zur aktuellen Vorlesung unter

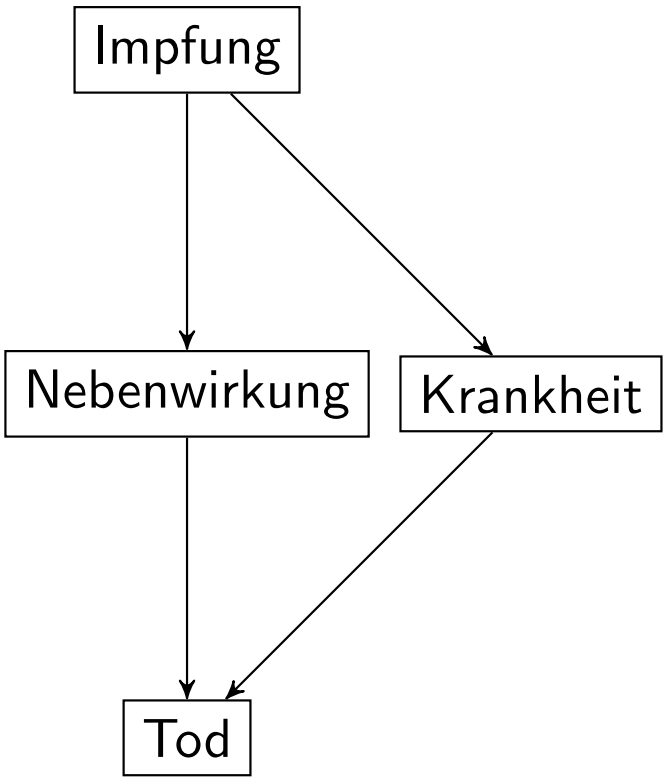
`https://informatik.htwk-leipzig.de/schwarz/lehre/ss22/kim`

Bücher:

- KI-Grundlagen:
  - Ingo Boersch, Jochen Heinsohn, Rolf Socher:
    Wissensverarbeitung (Spektrum, 2007)
  - Wolfgang Ertel:
    Grundkurs Künstliche Intelligenz (Springer, 2016)
    (elektronische Version in HTWK-Bibliothek)
  - Ronald Brachman, Hector Levesque:
    Knowledge Representation and Reasoning
    (Morgan Kaufmann 2004)
  - Stuart Russell, Peter Norvig:
    Künstliche Intelligenz (Pearson 2004)
- KNN:
  - Raúl Rojas: Neural Networks – A Systematic Introduction
    `https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf`

Kausale Inferenz

Judea Pearl: The Book of Why (`http://bayes.cs.ucla.edu/WHY/`)
Causality: Models, Reasoning, and Inference



$$
\begin{aligned}
p(I) &= 0.99, \quad p(\neg I) = 0.01 \\
p(N|I) &= 0.01, \quad p(N|\neg I) = 0 \\
p(K|I) &= 0, \quad p(K|\neg I) = 0.02 \\
p(T|N) &= 0.01 \\
p(T|K) &= 0.2
\end{aligned}
$$

bei 10^6 Personen mit Impfrate 99%:
40 Tote durch Krankheit + 99 Tote durch Impfung

Counterfactual:
Was wäre ohne Impfung ($p(I) = 0$)?
4000 Tote durch Krankheit

Kausalitätsstufen

(ladder of causality)

1. Assoziation (Wahrnehmen)
   Was ist? (Was wird beobachtet?)
2. Intervention (Beeinflussen)
   Was geschieht, wenn Aktion X ausgeführt wird (würde)?
3. Counterfactuals (Vorstellen, Reflektieren)
   Warum geschah . . . ?
   Lag es an Aktion X?
   Was wäre, wenn Aktion X nicht stattgefunden hätte?

Methoden:

- Kausale Modelle (Diagramme)
- do-Kalkül
- Counterfactuals

Menschliches Schließen, Kognition, mentale Modelle

Keith Stenning (Kognitionswissenschaften),
Michiel van Lambalgen (Logik):
Human Reasoning and Cognitive Science

- Wason Selection Task:
  4 Karten, eine Seite Ziffern, andere Seite Buchstaben
  sichtbar: [A] [K] [4] [7]
  Aussage: Wenn Vokal, dann gerade Zahl
  Wieviele / welche Karten müssen umgedreht werden, um die Wahrheit der Aussage zu überprüfen?
  häufigste Antwort (45%): [A] und [4]
- Sally-Anne-Test:
  Sally hat einen Korb, Anne eine Schachtel, beide verschlossen.
  Sally legt einen Apfel in den Korb und geht aus dem Raum.
  Anne legt den Apfel vom Korb in die Schachtel.
  Wo sucht Sally den Apfel, wenn sie wiederkommt?

Menschliches Schließen, Wissen über Wissen, Erklärungen

van Benthem, Hans van Ditmarsch, Jan van Eijck, Jan Jaspars:
Logic in Action (`http://www.logicinaction.org/docs/lia.pdf`)

z.B. Schließen aus individuellem und gemeinsamem Wissen

- 3 schlafenden Weisen wird die Stirn beschmiert.
  Nachdem alle aufgewacht sind, lachen sie sich gegenseitig aus.
  Ein später vorbeikommender Wanderer sagt:
  „Wenigstens einer von Euch hat eine beschmierte Stirn.“
  Kurz darauf hören alle zugleich auf zu lachen.
- Zahlenrätsel:
  Person A wählt zwei natürliche Zahlen zwischen (einschließlich) 2 und 100 und verrät S deren Summe und P deren Produkt.

  P: Ich kenne die beiden Zahlen nicht.
  S: Das weiß ich. Ich kenne sie auch nicht.
  P: Dann kenne ich die beiden Zahlen jetzt.
  S: Dann kenne ich sie jetzt auch.

Was bisher geschah

- KI-Geschichte
- KI-Tests (Turing, Chinese Room)
- statistische / symbolische KI

Daten, Wissen, Intelligenz

Umwelt		Reize, Eindrücke
Agent	Wahrnehmen, Beobachten	Daten
	Erkennen, Verstehen	Information
	Anwenden, Können	Wissen
	Lernen, Verbessern	Wissenserwerb
	Verstehen, Begründen, Erklären, Reflektieren, Grenzen erkennen	Intelligenz

Beispiel: Daten, Information, Wissen, Intelligenz

Daten	Darstellungsform (Syntax) Zeichenketten, Bilder, Ton, . . . (z.B 39.7)
Information	Bedeutung der Daten (Semantik) in einem bestimmten Kontext im Beispiel: Körpertemperatur= 39.7
Wissen	Information mit einem Nutzen, trägt zur Lösung eines Problemes bei, Nutzen abhängig von vorhandenem Kontextwissen im Beispiel: Kontext Körpertemperatur> 39.0 ist Fieber, Fieber ist Symptom von COVID-19 (27%) oder Nebenwirkung einer Corona-Impfung oder ... Bei Verdacht auf COVID-19 testen, bei kürzlicher Impfung beobachten, sonst ...
Wissenserwerb	selbständige Informationsgewinnung (auch zum Kontext) im Beispiel über (derzeit typische) Auslöser, Nebensymptome, Therapien
Intelligenz	Diagnose und Auswahl aus Therapie-Alternativen speziell für die zu behandelnde Person durch Abwägung der zu erwartenden Wirkungen, ggf. Überweisung zu Spezialisten

Explizites und implizites Wissen

explizites Wissen

z.B. Fakten, Aussagen, Zusammenhänge, Verfahren
ermöglicht maschinelle Verarbeitung

implizites Wissen

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

Kommuniziert werden kann nur explizites Wissen.

Transformation von implizitem in explizites Wissen notwendig

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.

formales Problem:

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

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

Darstellung von Aufgabe und Lösung

Aufgabe:

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

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

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

Entfaltung des Graphen zu einem Baum:
Pfade im Graphen = Knoten im Baum

Problemlösen durch Suchen

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

Möglichkeiten zum Problemlösen:

- Pfadsuche im Graphen
- Knotensuche im Baum

Suche in Graphen

(schon bekannte) Verfahren zur Suche in Graphen (und Bäumen):

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

Allgemeines Suchverfahren

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

Allgemeiner Suchalgorithmus:

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

prominente Spezialfälle:

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

Breitensuche
- Verwaltung von L_a als Queue
- Einfügen der Nachbarn an das Ende der Liste L_a
- festgelegter Knoten wurde zuerst in L_a eingefügt

Eigenschaften von Suchverfahren

Fairness: jeder Knoten wird expandiert

Vollständigkeit: jede Lösung wird gefunden

Optimalität: eine beste Lösung (bzgl. gegebener Bewertung) wird (zuerst) gefunden

Zeitaufwand: maximale Anzahl der Schritte von einem Startzustand zu einer Lösung

(Speicher-)Platzbedarf: maximale Länge der Liste der noch zu expandierenden Knoten (L_a)

Baum mit maximaler Tiefe t und maximalem Verzweigungsgrad d:

- Breitensuche: vollständig, optimal bzgl. Bewertung = Tiefe
  hoher Zeitaufwand: $O(d^t)$
  hoher Platzbedarf: $O(d^t)$
- Tiefensuche:
  vollständig (nur bei Bäumen ohne unendliche Pfade),
  nicht optimal bzgl. Bewertung = Tiefe
  hoher Zeitaufwand: $O(d^t)$
  geringerer Platzbedarf: $O(td)$

Schrittweise Vertiefung (ID)

(iterative deepening)

Ziel: Verbindung der Vorteile von

- Tiefensuche (geringer Speicherbedarf)
- Breitensuche (Vollständigkeit auch bei unendlichen Bäumen)

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 $m \in \mathbb{N}$ (aufsteigend geordnet), solange keine Lösung gefunden wurde

Vorteil: vollständig, optimal bzgl. Bewertung = Tiefe

Nachteil:

Knoten nahe des Startzustandes werden mehrfach expandiert

aber (asymptotischer) Zeit- und Platzbedarf wie Tiefensuche

Bidirektionale Suche (nach Lösungswegen, Plänen)

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

Speicherbedarf geringer als bei Breitensuche

- 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 (gleicher oder 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:

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

Uniforme Kostensuche ist wie Breitensuche und Tiefensuche ein uninformiertes Suchverfahren

Was bisher geschah

- Daten, Information, Wissen

Problemlösen durch Suche:

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

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

Lösung:

- Zielzustand (evtl. mehrere, mit Bewertung, ...)
- zulässiger Weg (Zustandsfolge $p \in V^*$)
  vom Start- zum Zielzustand

uninformierte (blinde) Suchverfahren:

- Tiefen-, Breitensuche, Gleiche-Kosten-Suche
- Schrittweise Vertiefung
- Bidirektionale Suche

Allgemeiner Suchalgorithmus (WH)

- aktuelle Menge der zu untersuchenden Knoten $L_a = \{s\}$
- aktuelle Menge der erledigten $L_x = \emptyset$
- solange nicht (Lösung gefunden oder $L_a = \emptyset$):
  1. Auswahl eines Knotens $u \in L_a$ (nach festgelegtem Verfahren)
  2. Verschiebung von u von L_a nach L_x $(L_a \setminus \{u\}, L_x \cup \{u\})$
  3. Einfügen aller (gerichteten) Nachbarn v von u mit $v \notin (L_a \cup L_x)$ (auf festgelegte Art) in L_a

Verschiedene Suchverfahren unterscheiden sich durch die Auswahl des expandierten (festgelegten) Knotens aus L_a.

nach Festlegung durch Datenstruktur zur Verwaltung von L_a als

- Stack: Tiefensuche
- Queue: Breitensuche
- Priority Queue: Gleiche-Kosten-Suche

Heuristische Suche – Motivation

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

Anwendung bei

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

Ziele:

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

Schätzfunktionen

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

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

repräsentiert die Zusatzinformation

Eigenschaften von Heuristiken

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

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

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

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

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

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

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

Besten-Suche

(best-first-search)

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

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

Beispiel: Suche eines kürzesten Weges zwischen Orten A und B

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

Besten-Suche – Eigenschaften

zwei Methoden:

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

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

Greedy-Suche (kleinste Restkosten)

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

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

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

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

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

Eigenschaften der Greedy-Suche:

- optimal?
- vollständig?

Bisherige Kosten

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

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

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

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

Beispiele für Kostenfunktionen:

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

A*-Suche (kleinste Gesamtkosten)

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

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

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

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

IDA*-Suche: Kombination von

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

Anwendungen

Planungsprobleme und kombinatorische Suchprobleme, z.B.

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

Fachbeitrag für Seminar in KW 18:
Robert C. Holte, 2010:
Common Misconceptions Concerning Heuristic Search
`https://aaai.org/ocs/index.php/SOCS/SOCS10/paper/view/2073/2500`

Beispiel Schiebefax (8-Puzzle)

- 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 ihrer Zielposition
- h_3 Summe der Manhattan-Abstände jeder Zahl zu ihrer 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 $h(u) \leq n$

Was bisher geschah

Daten, Information, Wissen

Modellierung duch

- Zustandsübergangssysteme:
  Graph mit markierten Knoten
  (Zustände und deren Eigenschaften)
- Startzustand, (Eigenschaften der) Zielzustände
- Lösung:
  - (optimaler) Zielzustand
  - Plan zum Erreichen eines Zielzustandes (Pfad im Graphen)
- Lösungsverfahren: Suche in Graphen

  uninformiert: Breiten-, Tiefen-, Gleiche-Kosten-Suche (WH)

  informiert: Greedy-, A*-Suche

  Optimierungen: ID, bidirektional
- Heuristiken, Eigenschaften

Zwei-Personen-Spiele

Zwei-Personen-(Brett)spiel:

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

Wissensrepräsentation (Spielbaum):

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

Beispiel Nim (Variante)

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

(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 $n = 6$

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 für

Max-Positionen u : $\alpha(u)$ untere Schranke für $m(u)$
(bisher höchster entdeckter Kandidat)

Min-Positionen u : $\beta(u)$ obere Schranke für $m(u)$
(bisher geringster entdeckter Kandidat)

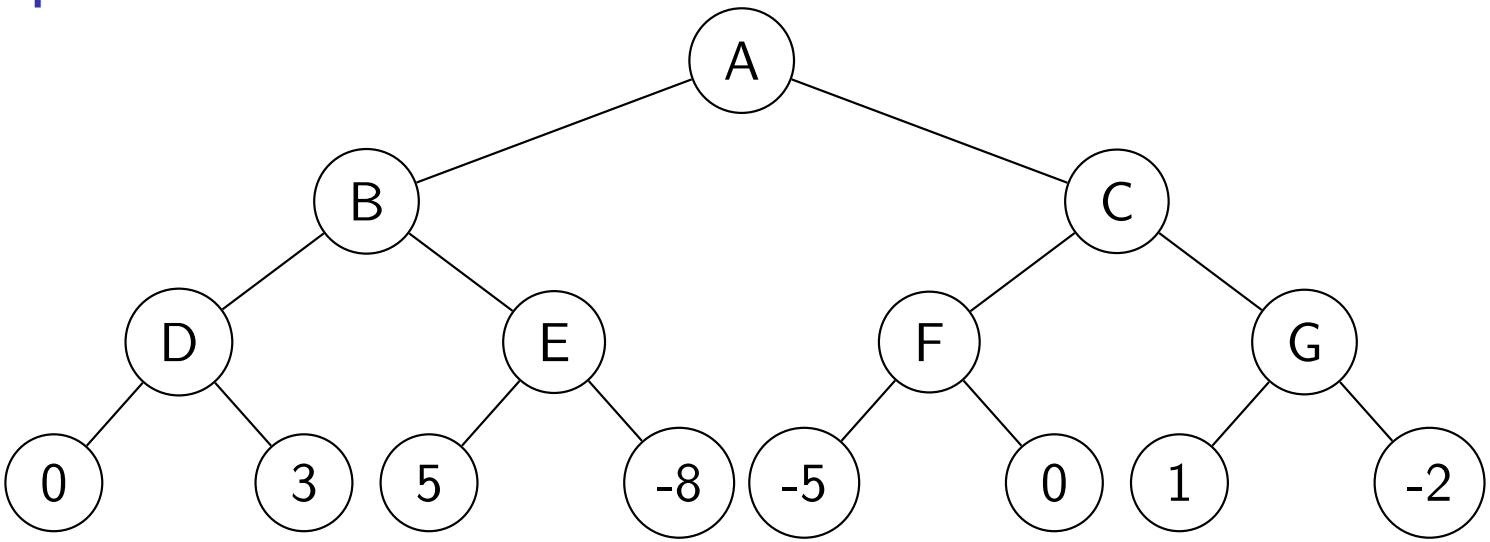
Idee: Bei Berechnung des Minimax-Wertes der Wurzel u eines Teilbaumes Berechnungen für Enkel von u auslassen, sobald bekannt ist, dass diese den Minimax-Wert $m(u)$ nicht ändern.

α-β-Pruning:
Abtrennen aller weiteren Knoten unterhalb jedes Kindes v des

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



Max beginnt, A, D, E, F und G sind Max-Knoten, B und C Min-Knoten, also $\alpha : \{A, D, E, F, G\} \to \mathbb{R}$, $\beta : \{B, C\} \to \mathbb{R}$

$ABD0$	$m(D) \geq 0$ $(\alpha(D) = 0)$
$D3$	$m(D) \geq 3$ $(\alpha(D) = 3)$
D	$m(D) = 3, m(B) \leq 3$ $(\beta(B) = 3)$
$BE5$	$m(E) \geq 5$ $(\alpha(E) = 5 > 3 = \beta(B))$
EB	$m(B) = 3, m(A) \geq 3$ $(\alpha(A) = 3)$
$ACF(-5)$	$m(F) \geq -5$ $(\alpha(F) = -5)$
$F0$	$m(F) \geq 0$ $(\alpha(F) = 0)$
F	$m(F) = 0, m(C) \leq 0$ $(\beta(C) = 0 < 3 = \alpha(A))$
CA	$m(A) = 3$

Automatische Berechnung heuristischer Funktionen

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

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

übliche Modifikationen:

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

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

Was bisher geschah

Abgrenzung der Begriffe: Daten, Information, Wissen, Intelligenz
Symbolische KI: Suchprobleme

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

Lösung: Pfad zu einem Zielzustand im Zustandsübergangssystem, Spielstrategie

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

Heuristische Funktionen:

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

Beispiel für Kombination symbolischer und statistischer Verfahren

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

Beispiel: Monte-Carlo-Baum-Suche (MCTS)

Entscheidungsunterstützung

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

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

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

Bewertung von Objekten / Situationen

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

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

Bewertung der Objekte anhand ihrer Merkmale, z.B.

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

Klassifikation

Ziel: Einteilung von Objekten (Fällen) anhand bestimmter Merkmale in Klassen (Auswahl einer Lösung für einen Fall aus einer Menge gegebener Alternativen)

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

- M Menge von Merkmalen m (Symptome, Attribute)
  jedes Merkmal m mit zugeordneter Menge V_m möglicher Werte, z.B. $M = \{a, f\}$ mit Alter a, $V_a = \mathbb{N}$, Augenfarbe f, $V_f = (0, 1)^3$ (RGB)
  Merkmalsraum: $\bigtimes_{m \in M} V_m$
- $O \subseteq M$ Menge aller Objekte (Fälle)
  Jedem Objekt $o \in O$ sind seine Merkmalswerte $m_o \in \bigtimes_{m \in M} V_m$ zugeordnet.
  z.B Tom $\in O$ mit $m_{\text{Tom}} = (5, (0, 0, 1))$
- K $(= W)$ Menge aller Klassen (Diagnosen, Lösungen)
  jede Klasse repräsentiert eine Teilmenge des Merkmalsraumes $\bigtimes_{m \in M} V_m$
  z.B. Baby, Kind

oft hilfreich: geometrische Interpretation

Klassifikationsprobleme

gegeben: Objekt $o \in O$ mit Merkmalswerten m_o
Zuordnung $K \to X_K$ mit $X_K \subseteq M$ (Extension)
jede Klasse definiert durch Merkmalswerte (oft Intervalle)

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

Beispiele:

- Klassifikation von Tieren
  - Objekte: Adler, Biber, Elefant, Fledermaus, Gorilla, Hecht, Pinguin, Specht
  - Merkmale: Federn, Schuppen, Fell, kann fliegen, kann schwimmen, legt Eier
  - Klassen: Säugetier, Fisch, Vogel
- Klassifikation von Bildern
  - Objekte: (digitale) Bilder
  - Merkmale: Größe, Farben, Texturen (auch von Teilbildern), vorkommende Muster, Metadaten (Name, Beschreibung, ...)
  - Klassen: Bildinhalte, z.B. Landschaften, Tiere, Gebäude

Statistische Verfahren

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
- Schwarm-Intelligenz

maschinelles Lernen:

- Ansätze zur ständigen Selbstverbesserung von Verfahren zum Problemlösen
- derzeit teilweise erfolgreicher Einsatz auf den Teilgebieten Bewertung, Klassifikation

(Natürliches und) Maschinelles Lernen

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

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

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

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

Lernen: Schritte 3 und 4

Lernverfahren

Lernen durch

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

nach Art des Lernenden:

- natürliches Lernen
- maschinelles (künstliches) Lernen

Lernen durch gegebene Beispiele

nach der zum Lernen verwendbaren Information:

überwachtes Lernen (supervised learning)

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

unüberwachtes Lernen (unsupervised learning)

gewünschte Eigenschaften des Löseverfahrens:

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

Korrigierendes Lernen

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

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

Rückmeldung: Trainer sagt nach jedem Lernschritt die korrekte Ausgabe.

Prinzip: Lernen durch Nachahmen (mit Korrektur)

Anwendung z.B. bei

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

Bestärkendes Lernen (reinforcement learning)

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

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

Rückmeldung: Trainer sagt nach jedem Lernschritt, ob die Ausgabe korrekt war.

Idee: Lernen durch Probieren

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

Unüberwachtes Lernen

Trainingsmenge: Menge von Eingaben

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

Idee: Lernen ohne Trainer (ohne Rückmeldung)

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

Neuronale Netze

Neuron – Nerv (griechisch)

Modellierung und Simulation der Strukturen und Mechanismen im Nervensystem von Lebewesen

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

Natürliche Neuronen

ZNS besteht aus miteinander verbundenen Nervenzellen (Neuronen)

Struktur eines Neurons:

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

Natürliche Neuronen – Funktionsweise

Informationsübertragung durch elektrochemische Vorgänge:

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

Stärke der Information durch Häufigkeit der Spannungsspitzen (Frequenzmodulation).

Eigenschaften natürlicher neuronaler Netze

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

Vorteile:

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

Ziel: Nutzung dieser Vorteile zum Problemlösen
durch Wissensrepräsentation als künstliche neuronale Netze

Natürliche Neuronen – Lernen

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

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

Anwendungen künstlicher neuronaler Netze

Anwendungsgebiete:

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

Geschichte künstlicher neuronaler Netze

- 1943, Warren McCulloch, Walter Pitts:
  A logical calculus of the ideas immanent in nervous activity
- 1949, Donald O. Hebb: Lernmodell
  The organization of behaviour
- 1957 Frank Rosenblatt: Perzeptron (1 Schicht)
  erster Neurocomputer MARK 1
  (Ziffernerkennung in 20×20-Bildsensor)
- 1969, Marvin Minsky, Seymour Papert: Perceptrons
- 1971 Perzeptron mit 8 Schichten
- 1974 Backpropagation (Erfindung)
- 1982, Teuvo Kohonen: selbstorganisierene Karten
- 1982, John Hopfield: Hopfield-Netze
- 1985, Backpropagation (Anwendung)
- 1997, long short-term memory (Erfindung)
- 2000, Begriff Deep Learning für KNN, Faltungsnetze (CNN)
- 2006, long short-term memory (Anwendung)
- 2009, verstärkt Training mit GPUs
- 2015, AlphaGo, AlphaZero, ...

Statistische Verfahren

Idee: Zuordnung eines gegebenen Falles zu ähnlichen (schon bekannten) Fällen

Einsatz zum Lösen von Problemen,

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

einige Ansätze:

- Künstliche neuronale Netze
- Evolutionäre Algorithmen
- Schwarm-Intelligenz
- Fuzzy-Logik, probabilistische Logik

maschinelles Lernen:

- Ansätze zur ständigen
  - Erweiterung der Datenmenge (bekannte Fälle)
  - Präzisierung der Ähnlichkeitsbewertung
- derzeit recht erfolgreicher Einsatz vorwiegend auf den Gebieten Bewertung, Klassifikation, Medienauswertung

Künstliche Neuronen: McCulloch-Pitts-Neuron ohne Hemmung

einfaches abstraktes Neuronenmodell von
McCulloch und Pitts, 1943

Aufbau eines künstlichen Neurons u (Tafel)

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

Parameter eines McCulloch-Pitts-Neurons u ohne Hemmung:

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

McCulloch-Pitts-Neuron ohne Hemmung: Funktionen

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

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

(Summe aller erregenden Eingänge des Neurons u)

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

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

(Stufenfunktion mit Stufe bei θ_u)

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

$$O_u(v) = v$$

(Identität)

McCulloch-Pitts-Neuron ohne Hemmung: Berechnung

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

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

m_u-stellige Boolesche Funktion

McCulloch-Pitts-Neuron ohne Hemmung: Beispiele

elementare Boolesche Funktionen $\vee, \wedge$

mehrstellige $\vee, \wedge$

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

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

Warum?

Geometrische Interpretation

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

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

und

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

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

Geometrische Interpretation: Beispiele

Beispiele:

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

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

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

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

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

Linear trennbare Funktionen

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

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

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

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

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

Die Boolesche Funktion `XOR` ist nicht linear trennbar.

McCulloch-Pitts-Neuron mit Hemmung

McCulloch-Pitts-Neuron u mit Hemmung:

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

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

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

Funktionen bei hemmenden Eingängen

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

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

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

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

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

$$O_u(v) = v$$

(Identität)

Berechnung bei hemmenden Eingängen

Gesamtfunktion des Neurons u

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

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

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

Beispiele mit Hemmung:

- elementare Boolesche Funktion: $\neg$
- komplexere Boolesche Funktionen, z.B.

  $x_1 \wedge \neg x_2$

  $\neg x_1 \wedge x_2 \wedge x_3$,

  $\neg(x_1 \vee \neg x_2 \vee \neg x_3)$

McCulloch-Pitts-Netze

McCulloch-Pitts-Netz:

gerichteter Graph mit

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

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

- parallel
- sequentiell
- rekursiv

McCulloch-Pitts-Netze

Ein-Schicht-McCulloch-Pitts-Netz

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

Mehr-Schicht-McCulloch-Pitts-Netz

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

Analogie zu logischen Schaltkreisen

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

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

Modifikationen von McCulloch-Pitts-Neuronen

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

Parameter künstlicher Neuronen

verschiedene künstliche Neuronenmodelle unterscheiden sich in:

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

Jedes Neuron mit m Eingängen repräsentiert eine Funktion von m Eingabewerten

Schwellwertneuronen

Idee: gewichtete Eingänge

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

Mathematisches Modell:
Schwellwertneuron (Perzeptron)

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

Parameter eines Schwellwertneuronens u:

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

Schwellwertneuronen: Funktionen

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

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

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

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

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

$$O_u(v) = v$$

(Identität)

Schwellwertneuronen: Berechnung

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

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

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

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

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

Schwellwertneuronen: geometrische Interpretation

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

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

und

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

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

Schwellwert als Gewicht (Bias-Neuronen)

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

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

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

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

Überwachtes Lernen einzelner Schwellwertneuronenn

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

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

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

(Werte der Funktion f an Stützstellen)

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

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

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

Δ-Regel

Idee: Lernen aus Fehlern (und deren Korrektur)

Delta-Regel:

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

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

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

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 für alle Trainingspaare 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 $(e(w_0', \ldots, w_n') < e(w_0, \ldots, w_n))$

Beispiel (Tafel): $\neg$
mit Startgewichten $w_0 = w_i = 0$ und Lernrate $\eta = 1/2$

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 nach jedem Trainingpaar

Batch-Lernen: Lernen in Epochen

Epoche : Durchlauf der gesamten Trainingsmenge

- Berechnung von Fehler und Gewichtsdifferenzen für jedes Trainingsmuster,
- Berechnung der Gewichtsdifferenzen für die gesamte Trainingsmenge (z.B. Summe über alle Trainingpaare)
- Änderung der Gewichte erst nach der 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.

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,
  Anordnung der Neuronen in Schichten (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)

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)

Konstruktion eines Schwellwertneuronen-Netzes

für eine Boolesche Funktion $f : \{0,1\}^m \to \{0,1\}$:

1. Bestimmung einer Formel in DNF mit Semantik f
2. Repräsentation jedes Disjunktionsgliedes φ_k durch ein Neuron u_k mit m_k Eingängen und den Gewichten
   für $i \in \{1, \ldots m_k\}$:

$$w_{ki} = \begin{cases} 2 & \text{falls } x_i \text{ in } \varphi_k \text{ nicht negiert vorkommt} \\ -2 & \text{falls } x_i \text{ in } \varphi_k \text{ negiert vorkommt} \\ 0 & \text{falls } x_i \text{ in } \varphi_k \text{ nicht vorkommt} \end{cases}$$

$$\text{und} \quad w_{k0} = m - 1 + \frac{1}{2} \sum_{i=1}^{m_k} w_{ki}$$

3. zusätzliches Schwellwertneuron v mit Ausgängen aller Disjunktionsneuronen als Eingänge mit den Gewichten

$$w_i = \begin{cases} 1 & \text{für } i = 0 \\ 2 & \text{für } i \in \{1, \ldots, k\} \end{cases}$$

Beispiele: XOR, $x_1 \leftrightarrow (x_2 \vee x_3)$

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
  aktueller Name für FFN mit mehr als einer Schicht: Deep Network
- 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
- gern verwendete Modifikation: Residual neural network
  enthalten Vorwärts-Kanten, die Schichten überspringen
- 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 Quadrat

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

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. $x_1 \geq -1$, z_2 feuert gdw. $x_1 \leq 0$
  z_3 feuert gdw. $x_2 \geq -1$, z_4 feuert gdw. $x_2 \leq 0$

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

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

Gesamtmatrix des FFN – Beispiel

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

Mehr-Schicht-FFN mit linearer Aktivierung

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

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

Ausgabe des Neurons z_i^j in Schicht j:

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

Netzausgabe:

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

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

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

Approximation von Funktionen

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

Ziel:
Konstruktion eines KNN zur Approximation dieser Funktion durch

- lineare Funktionen
- Stufenfunktionen
- komplexere Funktionen

Quadratischer Fehler

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

- Trainingsmenge liefert Stützstellen:

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

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

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

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

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

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

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

Trainingsziel: Minimierung des quadratischen Fehlers

Beispiel

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

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

(ganz einfaches) Ein-Schicht-FFN:

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

Funktionen des Ausgabeneurons y:

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

Dieses Netz berechnet die Funktion

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

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

Methode der kleinsten Quadrate

direkte Berechnung mit Hilfe der partiellen Ableitungen nach m und n

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

partielle Ableitungen nach m und n:

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

Bestimmung der Parameter

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

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

mit den Lösungen

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

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

Berechnung der Gewichts-Verschiebungen

Ziel: Minimierung des Fehlers durch schrittweise Verschiebung des Gewichtsvektors

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

steilster Anstieg: Gradient (partielle Ableitungen)

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

Voraussetzung: Fehlerfunktion ist differenzierbar

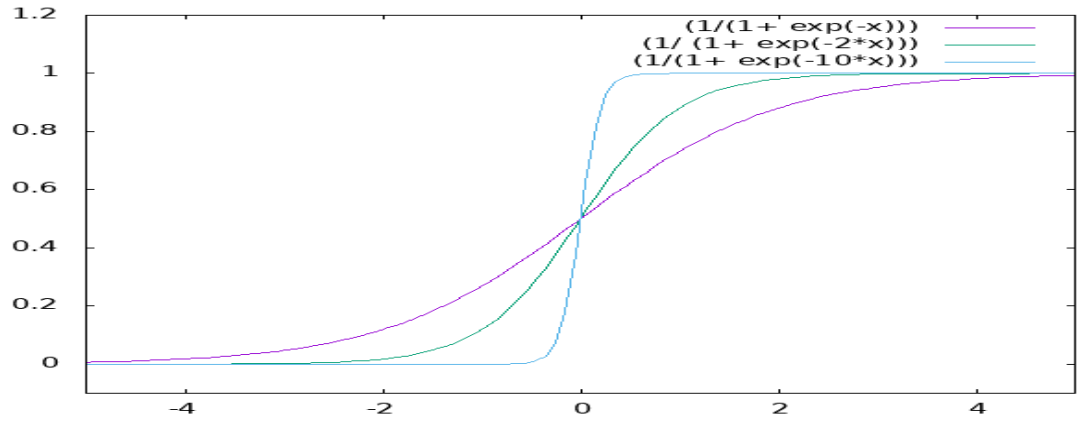
zur Anwendung in KNN: f_u differenzierbar gdw.

- Eingangsfunktion I_u differenzierbar: gewichtete Summe (ok)
- Ausgangsfunktion O_u differenzierbar : linear (ok)
- Aktivierungsfunktion A_u differenzierbar

Klassisch: 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

Aktuell: lineare Aktivierungsfunktionen und ReLU

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

+ einfach (schnell) zu berechnen
  überall differenzierbar
  Ableitung: konstant m
  in jedem Punkt $x > 0$ eindeutige Abstiegsrichtung
- Gesamtfunktion f_u bleibt linear

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 kaum relevant)

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

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“ (Overfitting)

Prominente Aktivierungsfunktionen

- Stufenfunktion $A(X) = (x \geq 0)$,
- sigmoide Funktion $A(x) = \frac{1}{1+e^{-x}}$, $A'(x) = \tanh x$
  (differenzierbare Approximation der Stufenfunktion)
- lineare Funktion $A(x) = ax + b$
- ReLU $A(X) = \max(0, x)$,
- analytische Funktion $A(X) = \log(1 + e^x)$,
  (differenzierbare ReLU-Approximation, softplus)
- Radiale Basisfunktion (mehrere Eingaben) $A(x) = r(d(x, w))$ mit Eingabe(-vektor) x, Gewicht-(svektor) w (Zentrum), Abstand d, radiale Funktion $r : \mathbb{R}_{\geq 0} \to [0, 1]$ mit
  - $r(0) = 1$
  - monoton fallend: $\forall x, y \in \mathbb{R}_{\geq 0} : (x < y) \to (r(x) < r(y))$

  z.B. d Manhattan-Metrik, $r(x) = \max(0, 1 - x)$
- Softmax (mehrere Ein- und Ausgaben) $A(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$
  zur Klassifizierung in mehrere Klassen
  (Wahrscheinlichkeitsverteilung)

Hybride Netze

Idee:

- Kombination verschiedenartiger Neuronen
- schichtweise verschiedene Aktivierungsfunktionen
- innerhalb derselben Schicht kommen (derzeit) meist nur Neuronen derselben Art vor
- Trennung von Neuronen in Folge mehrerer eigene Neuronen für Eingangs-, Aktivierungs-, Ausgangsfunktion
- verschiedene Lernverfahren je Schicht

Cognitron

Beobachtungen im visuellen System:

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

Faltung (1D)

stetig:

Für $f, g : \mathbb{R} \to \mathbb{R}$ ist die Faltung von f und g definiert durch

$$\forall p \in \mathbb{R} : (f * g)(p) = \int_{-\infty}^{\infty} f(x)g(p-x)dx$$

diskret:

Für $f, g : \mathbb{Z} \to \mathbb{Z}$ ist die Faltung (Convolution) von f und g definiert durch

$$\forall p \in \mathbb{Z} : (f * g)(p) = \sum_{x=-\infty}^{\infty} f(x)g(p-x)$$

diskret mit endlichem Bereich $\text{pos} = [m, \ldots, n] \subseteq \mathbb{Z}$ für f:

Für $f : \text{pos} \to \mathbb{Z}, g : \mathbb{Z} \to \mathbb{Z}$ ist die Faltung (Convolution) von f und g definiert durch

$$\forall p \in \mathbb{Z} : (f * g)(p) = \sum_{x \in \text{pos}} f(x)g(p-x)$$

Diskrete Faltung (1D) – Beispiel

$f : \mathbb{Z} \to \mathbb{Z}, g : \{1, 2\} \to \mathbb{Z}$ mit

$$f(x) = \begin{cases} 3 & \text{falls } x = 1 \\ 1 & \text{falls } x = 2 \end{cases} \qquad g(x) = \begin{cases} 2 & \text{falls } x = 1 \\ 1 & \text{falls } x = 2 \\ 0 & \text{sonst} \end{cases}$$

häufig: Darstellung von f (Faltungskern)
als Vektor mit endlichem Definitionsbereich $f = (f(1), f(2)) = (3, 1)$

$$(f * g)(p) = \begin{cases} 6 & \text{falls } p = 2 \\ 5 & \text{falls } p = 3 \\ 1 & \text{falls } p = 4 \\ 0 & \text{sonst} \end{cases}$$

Berechnung (des relevanten Teils):

$$\begin{aligned} (f * g)(1) &= f(1)g(0) + f(2)g(-1) = 3 \cdot 0 + 1 \cdot 0 = 0 \\ (f * g)(2) &= f(1)g(1) + f(2)g(0) = 3 \cdot 2 + 1 \cdot 0 = 6 \\ (f * g)(3) &= f(1)g(2) + f(2)g(1) = 3 \cdot 1 + 1 \cdot 2 = 5 \\ (f * g)(4) &= f(1)g(3) + f(2)g(2) = 3 \cdot 0 + 1 \cdot 1 = 1 \\ (f * g)(5) &= f(1)g(4) + f(2)g(3) = 3 \cdot 0 + 1 \cdot 0 = 0 \end{aligned}$$

Faltung (2D)

Für $f, g : \mathbb{R}^2 \to \mathbb{R}$ ist die Faltung (Convolution) von f und g definiert durch

$$\forall (p_x, p_y) \in \mathbb{R}^2 : (f * g)(p_x, p_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) g(p_x - x, p_y - y) dx dy$$

diskreter Fall (Bilder):
Für $f, g : \{0, \ldots, m-1\} \times \{0, \ldots, m-1\} \to$ ist die Faltung (Convolution) von f und g definiert durch
$\forall (p_x, p_y) \in \{0, \ldots, m-1\} \times \{0, \ldots, m-1\}$:

$$(f * g)(p_x, p_y) = \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} f(x, y) g(p_x - x, p_y - y)$$

Erweiterung auf höhere Dimensionen analog

Multiskalen-Strategien

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

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

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

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

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

Bild-Pyramiden durch Faltungen

zur Erkennung von Features:

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

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

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

CNN-Idee:
Umsetzung durch Aufeinanderfolgen verschiedener Schichten in (feed-forward)-KNN

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

KNN zur Klassifikation

z.B. von handgeschriebenen Ziffern

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

Convolutional Neural Networks

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

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

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

nach und nach Entwicklung komplexerer Konstruktionen, z.B.

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

CNN-Schichten

aktuelle CNNs bestehen im Wesentlichen aus mehrfacher Wiederholung folgender Schichten:

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

CNNs unterscheiden sich in

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

CNN – prominente Beispiele

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

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

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

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

CNN-Lernen

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

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

deshalb häufig auch Nutzung vortrainierter Netze

Idee:

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

Was bisher geschah

Maschinelles Lernen

- überwacht: korrigierend / bestärkend
- unüberwacht
- Anwendungen

Künstliche Neuronen

- Biologisches und mathematisches Neuronenmodell
- Eingang- / Aktivierungs- / Ausgangsfunktion mit Beispielen
- McCulloch-Pitts-Neuron ohne / mit Hemmung,
  Schwellwertelement,
  allgemeines Neuronenmodell

Künstliche Neuronale Netze: Feed-Forward-Netze

- Ein-Schicht-FFN (ohne versteckte Neuronen)
  Lernen mit Δ-Regel
- Mehr-Schicht-FFN / DNN (mit versteckten Schichten)
  Lernen mit Backpropagation (Gradientenabstieg)
- Spezialfall CNN: hybride Netze aus Schichten von Neuronen verschiedener Typen, u.A. Faltungsschichten

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: h_1, h_2, h_3 mit

- erste Schicht (RBF): Zentren $w_{x,h1} = -1$, $w_{x,h2} = 2$, $w_{x,h2} = 4$
  Radien $\theta_{h1} = \theta_{h2} = \theta_{h2} = 2$
- zweite Schicht: Gewichte $w_{h1,y} = w_{h3,y} = 1, w_{h2,y} = 2$,
  Eingabefunktion: gewichtete Summe
  Aktivierung: linear

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. Δ-Regel)
   (Bestimmung der Koeffizienten 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
- Gefahr des Auswendiglernens der Trainingsdaten

Rekurrente Netze: Motivation

Ziel: Nachnutzung von Informationen aus vorangegangenen Schritten (Gedächtnis, Feedback), z.B. zur

- Repräsentation zeitlicher Folgen von Mustern
- Zeitreihenanalyse und -voraussage
- Erkennung von Sätzen (Grammatik, Sprachverarbeitung)
- 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

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

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

Kanten zwischen u und v: feedback loops

Zustand rekurrenter Netze

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

Übersetzung in Zustandsübergangssysteme
(endliche Automaten mit Ausgabe in Knoten, Moore)

Satz: Zu jedem NFA existiert ein rekurrentes Netz aus 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
- Approximation der Rekursion durch endliche Anzahl von Kopien

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

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

Jordan-Netze

Idee (1986): Nachnutzung der Netzausgaben
Netz-Topologie:

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

Elman-Netze

Idee (1990): 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)

Long Short-Term Memory (LSTM)

Idee (1997): Nachnutzung der Aktivierung der versteckten Neuronen

- Knoten: LSTM unit (memory block + gate units) besteht aus
  - Kern: CEC (constant error carrousel)
    lineare Aktivierung und Rückkopplung zu sich selbst (feedback)
    speichert Zustand, Rückkopplung im nächsten Schritt
  - zusätzliche gate units vor und nach CEC steuern die Fähigkeit, Eingaben zu verarbeiten und speichern (input gates) bzw. Ausgaben weiterzugeben (output gates)
- Netz-Topologie: 3-Schicht-FFN, LSTM units in versteckter Schicht

prominente Modifikationen:

- forget gate steuert (multipliziert, i.A. schwächt) Gewicht der rückkoppelnden Kante
- Peephole Connections (gewichtete Kanten von CEC zu allen gates)
  bei schwacher Aktivierung: Isolation des CEC
- mehrere Schichten von LSTM-Zellen (Deep LSTM)

Anwendungen:

- Zeitreihenanalyse, -vorhersage
- Handschrifterkennung, Spracherkennung, Übersetzung

Assoziativspeicher

Ziel: Musterassoziation
(Training mit endlich vielen Musterpaaren)

Generalisierung:

- Aus Zuordnung: Muster $x \rightarrow$ Muster y
  folgt für jedes zu x ähnliche Muster x' die
  Zuordnung: Muster $x' \rightarrow$ Muster y.
- Ziel: sinnvolle Zuordnung „verrauschter“ oder unvollständiger Eingabemuster

Netztypen:

heteroassoziativ Eingabemuster $x \in \mathbb{R}^m$,
Ausgabemuster $y \in \mathbb{R}^n$

Mustererkennung Spezialfall heteroassoziativer Netze
assoziiert Muster mit Identifikator, z.B. für Klasse

autoassoziativ Spezialfall heteroassoziativer Netze
Ein- und Ausgabemuster $x \in \mathbb{R}^m$ (prinizipiell) gleich

Heteroassoziativer Speicher

Topologie: vollständig verbundenes Ein-Schicht-FFN,
Eingänge $x \in \mathbb{R}^m$, Ausgänge $y \in \mathbb{R}^n$,
alles Schwellwertneuronen, Gewichtsmatrix $W \in R^{m \times n}$

Aktivierung: Signum = Stufenfunktion

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

Berechnung der Gewichte: Methode der kleinsten Quadrate =
Minimierung von

Berechnung der Eingangsfunktion analog Ein-Schicht-FFN:

$$I(x_1, \ldots, x_m) = (x_1, \ldots, x_m) \begin{pmatrix} w_{11} & \ldots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \ldots & w_{mn} \end{pmatrix}$$

dasselbe kürzer: $I(x) = xW$

Heteroassoziativer Speicher: Beispiel

Berechnung für 3-2-Netz mit Gewichtsmatrix W:

$$W = \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Eingabe $x = (1, -1, 1)$

Berechnung:

$$\text{sgn}(xW) = \text{sgn}\left((1, -1, 1)\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\right) = \text{sgn}(-3, 3) \mapsto (-1, 1)$$

Ausgabe $y = (-1, 1)$

Heteroassoziativer Speicher: Training

Idee: Lernen aus gleichzeitiger Aktivität (Hebb)

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

Trainingsmenge $\{(x^{(i)}, y^{(i)}) \mid i \in \{1, \ldots, k\}\}$ (Bipolarvektoren)

Ziel Gewichtsmatrix W, so dass für jedes Trainingspaar $(x^{(i)}, y^{(i)})$ gilt: mit $\text{sgn}(x^{(i)}W) = y^{(i)}$ (komponentenweise)

Startgewichte alle $w_{ij} = 0$

Lernregel von Hebb $\Delta w_{ij} = \eta x_i y_j$, hier mit Lernrate $\eta = 1$

Training : Gewichtsbestimmung $\Delta w_{kl} = x_k y_l$ je Trainigspaar (x, y) einmal (W ist Korrelationsmatrix von x und y)

Alternative zum Training: direkte Berechnung der Gewichte (z.B. mit Methode der kleinsten Quadrate)

Bidirektionaler Assoziativspeicher (BAM)

(heteroassoziativer Speicher von Musterpaaren)

Netz-Topologie:

- Eingangsschicht, Ausgangsschicht, keine versteckten Neuronen
- Eingänge $x \in \{-1, 1\}^m$
- Ausgänge $y \in \{-1, 1\}^n$
- vollständige symmetrische Verbindungen zwischen jedem Eingangs- und jedem Ausgangsneuron (ungerichteter vollständiger bipartiter Graph $K_{m,n}$)
- Gewichte an jeder Kante (für beide Richtungen gleich) Gewichte in Gewichtsmatrix $W \in R^{m \times n}$
- Eingangsfunktion: gewichtete Summe
- Aktivierung: Signum
- Ausgangsfunktion: Identität

BAM: Berechnung und Training

(wie im heteroassoziativen Speicher)

überwachtes Lernen

Trainingsmenge $\{(x^{(i)}, y^{(i)}) \mid i \in \{1, \ldots, k\}\}$

Eingabe : Startzustand (initiale Zustände der Eingangsneuronen $x \in \{-1, 1\}^m$)

Lernregel von Hebb: $\Delta w_{ij} = \eta x_i y_j$ mit $\eta = 1$

Berechnung : Folge von Schritten (Zustandsübergängen),

- synchron oder
- abwechselnd

für beide Neuronenschichten:
Aktualisierung: Neuberechnung der Aktivierung der anderen Schicht
wiederholen, bis stabiler Zustand erreicht (Fixpunkt) oder Abbruch ausgelöst wird

Ausgabe : Zustand der Ausgangsneuronen des stabilen Netzes

BAM: Beispiel

ein Trainingspaar (x, y) mit
$x = (1, -1, 1)$ und $y = (-1, 1)$

Gewichtsmatrix W:

$$W = \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Berechnung:

$$\text{sgn}(xW) = \text{sgn}\left((1, -1, 1)\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\right) = \text{sgn}(-3, 3) \mapsto (-1, 1) = y$$

$$\text{sgn}(Wy^T) = \text{sgn}\left(\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix}\right) = \text{sgn}\begin{pmatrix} 2 \\ -2 \\ 2 \end{pmatrix} \mapsto \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = x^T$$

BAM: Training

(wie heteroassoziativer Speicher)

- für ein zu speicherndes Musterpaar $x \in \{-1,1\}^m, y \in \{-1,1\}^n$: $W = x^T y$
  (Korrelationsmatrix von x und y)
- für mehrere zu speichernde Musterpaare $(x^{(1)}, y^{(1)}), \ldots, (x^{(k)}, y^{(k)})$:

$$W = \sum_{i=1}^{k} W^{(i)} = \sum_{i=1}^{k} \left(x^{(i)}\right)^T y^{(i)}$$

Für alle k Trainingsmuster gleichzeitig:

- Eingabemuster $X \in \{-1,1\}^{k \times m}$
- Ausgabemuster $Y \in \{-1,1\}^{k \times n}$
- $XX^T \in \{-1,1\}^{k \times k}$ (alle Einträge positiv)
- $\text{sgn}(Y) = \text{sgn}(XX^T Y) \in \{-1,1\}^{k \times n}$
- $\text{sgn}(Y) = \text{sgn}(XW) \in \{-1,1\}^{k \times n}$ mit $W = X^T Y$

Hopfield-Netz

(autoassoziativer Musterspeicher)
Idee: autoassoziativer BAM (gleiche Anzahl n von Ein- und Ausgaben)
jedes Paar von Ein- und Ausgabeknoten identifiziert

Topologie: K_n mit symmetrischer Gewichtsmatrix $W \in \mathbb{R}^n$
Jedes Neuron ist zugleich Ein- und Ausgang $x \in \{-1, 1\}^n$
keine Selbstrückkopplung, also $\forall i \in \{1, \ldots, n\} : w_{ii} = 0$

Zustand: Aktivierung aller Neuronen

Eingabe: Startzustand (initiale Zustände aller Neuronen)

Berechnung: Folge von Schritten (Zustandsübergängen):
Aktualisierung: Berechnung der Aktivierung aller Neuronen
wiederholen, bis stabiler Zustand erreicht oder Abbruch

Konvergenz : Erreichen eines stabilen Zustandes (ändert sich bei Aktualisierung beliebiger Neuronen nicht)

Ausgabe : Zustand des stabilen Netzes

Aktualisierung in jedem Schritt:

synchron: gleichzeitige Zustandsänderung für alle Neuronen

asynchron: Zustandsänderung eines zufällige gewählten Neurons (faire Auswahl)

Hopfield-Netz – Beispiele

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

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

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

$w_{ii} \neq 0 \rightarrow$ Oszillation

Hopfield-Netz – Training

direkte Berechnung der Gewichte möglich,
kein Training notwendig

- für ein zu speicherndes Muster $x \in \{-1, 1\}^m$:

$$W = x^T x$$

  mit Modifikation: alle Diagonalelemente 0

- für mehrere zu speichernde Muster
  $x^{(1)} \in \{-1, 1\}^m, \ldots, x^{(k)} \in \{-1, 1\}^m$:

$$W = \sum_{i=1}^{n} \left(x^{(i)}\right)^T x^{(i)}$$

  mit Modifikation: alle Diagonalelemente 0

Beispiel (Tafel):

- ein Muster: $x = (1, -1, 1, 1)$
- mehrere Muster: $x^{(1)} = (-1, 1, -1)$ und $x^{(2)} = (1, -1, 1)$