

Künstliche Intelligenz

Einführung

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Rostock



Ziel der Lehrveranstaltung

- Künstliche Intelligenz: Teilbereich der Informatik, der sich mit der Erstellung *intelligenter Agenten* beschäftigt
- Angrenzende Fachgebiete: Neurowissenschaften, Psychologie, Linguistik, ...
 - Diese beschäftigen sich mit der Erforschung menschlicher Intelligenz
 - Uns geht es stattdessen um *rationale Agenten*

Was ist Intelligenz?

An agent is intelligent to the extent that what it does is likely to achieve what it wants, given what it has perceived"

(Stuart Russell, Human Compatible: AI and the Problem of Control)

Literatur und Inhalte



Stuart Russel und Peter Norvig

Artificial Intelligence – A Modern Approach (AIMA), 3. Ausgabe

Prentice Hall, 2009

aima.cs.berkeley.edu

Inhalte (Plan)

- I Einführung – Intelligente Agenten
- II Problemlösung durch Suchen
- III Schlussfolgern unter Unsicherheit
- IV Maschinelles Lernen
- V Decision Theory

Organisatorisches

- Lehrveranstaltungen vom 26.02. bis 08.03.
- Voraussetzung für die Teilnahme an der Klausur: Erfolgreiche Teilnahme am *Projekt*
 - Weitere Infos dazu nächste Woche
- Vorlesungsmaterialien: mds-lab.de/ki-ubb-cluj-napoca

Was ist Intelligenz?

Größte Erfolge von Künstlicher Intelligenz in den letzten Jahren?

Was ist Intelligenz?

Herangehensweisen, Intelligenz zu definieren:

- äußerlich sichtbares Verhalten vs. innere Denkprozesse
- Menschliche Intelligenz vs. rationales Verhalten

Was ist künstliche Intelligenz?

Welches Systemverhalten erfüllt das Kriterium „Intelligent“?

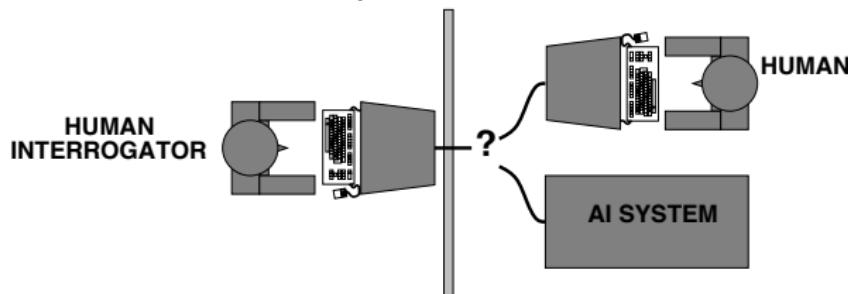
Menschlich Denken	Rational Denken
„The exciting new effort to make computers think . . . <i>machines with minds</i> , in the full and literal sense.“ (Haugeland 1984) „[The automation of] activities that we associate with human thinking, activities such as decision making, problem solving, learning . . . “ (Bellman, 1978)	„The study of mental faculties through the use of computational models.“ (Charniak & McDermott, 1985) „The study of computations that make it possible to perceive, reason, and act.“ (Winston, 1992)
Menschlich Handeln	Rational Handeln
„The art of creating machines that perform functions that require intelligence when performed by people.“ (Kurzweil, 1990) „The study of how to make computers do things at which, at the moment, people are better.“ (Rich & Knight, 1991)	„Computational intelligence is the study of the design of intelligent agents.“ (Poole et al., 1998) „AI . . . is concerned with intelligent behavior in artifacts.“ (Nilsson, 1998)

- Als nächstes: Kurzer Einblick in jede der 4 Sichtweisen

Menschlich Handeln

Turing (1950) „Computing machinery and intelligence“

- „Können Maschinen denken?“ → „Können sich Maschinen intelligent verhalten?“
- Praktischer Test: Imitationsspiel



- Vorhersage, dass in 2000 eine Maschine eine 30% Chance haben könnte, einen Laien für 5 Minuten von ihrer Menschlichkeit zu überzeugen
- Diskutiert bereits alle wesentlicher Argumente gegen KI, die in den folgenden 50 Jahren aufgebracht werden
- Identifiziert bereits wesentliche Komponenten von KI-Systemen: Wissen, Schließen, Sprachverständnis, Lernfähigkeit

Menschlich Handeln

Turing (1950) „Computing machinery and intelligence“

Problematik des Turing-Tests:

- Nicht reproduzierbar
- Nicht konstruktiv
- Nicht mathematisch analysierbar

Menschlich Denken

Kognitions- und Neurowissenschaften

- 1960er: „Kognitive Revolution“: Psychologie der Informationsverarbeitung ersetzt den vorherrschenden orthodoxen Behaviorismus
- Erfordert wissenschaftliche Theorien der internen Prozesse im Gehirn
 - Welches Abstraktionsniveau? – „Wissen“? „Schaltkreise“?
 - Wie validieren? – Verhaltenstests (top down)? Direkte Analyse neurologischer Daten (bottom up)?
- Beide Ansätze (Kognitionswissenschaften und Neurowissenschaften) jetzt von KI getrennt
- Siehe auch: *Human Brain Project* – Ziel unter anderem: Nachbildung (von Teilen) des Gehirns durch Computermodelle. Finanziert durch die Europäische Union (1,19 Mrd. Euro)

Rationales Denken

Grundregeln des Schließen

- Schlusssysteme, die Regeln für korrekte Inferenz festlegen
- Durch syntaktische Schlussregeln und formal präzise Definition wird maschinelles Schließen möglich
- Normativ statt beschreibend
- Seit Aristoteles: Was sind „korrekte“ Argumentations- und Denkprozesse?
- von hier direkte Linie über Mathematik und Philosophie zur KI
- Probleme:
 - Formalisierung von Alltagswissen („Common Sense“)
 - Behandlung von Unsicherheit
 - Nicht jede „intelligente Handlung“ erfordert einen mathematischen Beweis

Rationales Handeln

Das Richtig tun

- Rationales Verhalten = das Richtig tun
- Das Richtig = das, was die erwartete Zielerreichung maximiert, gegeben die verfügbaren Informationen
- Dabei irrelevant, ob man „viel denken muss“, um das Richtig herauszufinden
- Charakterisierung von „Intelligenz“ auf Basis der *Wirkung* in Bezug auf ein objektives Kriterium (Rationalität)
- Perspektive dieser Vorlesung

Rationale Agenten

- Ein *Agent* ist eine Entität, die ihre Umgebung wahrnimmt und in dieser Umgebung handelt
- In dieser Vorlesung diskutieren wir den Entwurf von *rationalen Agenten*
- Abstrakt ist ein Agent eine Funktion, die Beobachtungssequenzen auf Aktionen abbildet:

Agent : [Percept] → Action

- Für beliebige Umgebungen und Aufgaben suchen wir dann diejenigen Agenten, die die beste Leistung erreichen
- Problem: Aufgrund von Einschränkungen in Speicherplatz und Rechenleistung wird „perfekte Rationalität“ nicht immer erreichbar sein
 - Wir suchen somit das beste Programm für die gegebenen Rechnerressourcen – „bounded rationality“

Starke vs. Schwache KI

- Schwache KI: Systeme, die in bestimmten Teilbereichen menschliche Leistungsfähigkeit erreichen, z.B. Schach
 - KI-Forschung beschäftigt sich fast ausschließlich mit Erstellung von schwacher KI
- Starke KI (Artificial General Intelligence, AGI): KI, die alle intellektuellen Aufgaben erlernen oder verstehen kann, die ein Mensch aufführen kann
 - Fernziel der KI-Forschung
 - Unpräzise Definition

Aktuelle AGI-Debatte

- AGI unpräzise definiert (können ja nicht mal Intelligenz definieren...)
- Ergebnisorientierte Definition von Francois Chollet: "We will know we have AGI when the majority of the world's GDP is being produced by autonomous AI agents."
- Aktuelle Debatte, wie lange wir noch von einer AGI entfernt sind
- Eure Prognose zu AGI?

Aktuelle AGI-Debatte

- AGI unpräzise definiert (können ja nicht mal Intelligenz definieren...)
- Ergebnisorientierte Definition von Francois Chollet: “We will know we have AGI when the majority of the world’s GDP is being produced by autonomous AI agents.”
- Aktuelle Debatte, wie lange wir noch von einer AGI entfernt sind
- Eure Prognose zu AGI?
- Prognosen reichen von “AGI in 8 Jahren” bis zu “nicht in den nächsten 50 Jahren”
- Meine Prognose
 - aktuelle KI-Modelle (Language Models, Vision Models) als nützliche Tools, aber Skalierung nicht ausreichend für AGI
 - 10% Wahrscheinlichkeit, für AGI in 15 Jahren
 - 30% Wahrscheinlichkeit für AGI in 30 Jahren

Zusammenfassung

- Ziel des Forschungsbereichs Künstliche Intelligenz ist die Erstellung rationaler Agenten
- Dafür kommen unter anderem Methoden der Logik, Suchverfahren, Statistik, Machine Learning und Decision Theory zum Einsatz, die wir uns in dieser Vorlesung anschauen werden

Künstliche Intelligenz

Intelligente Agenten

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

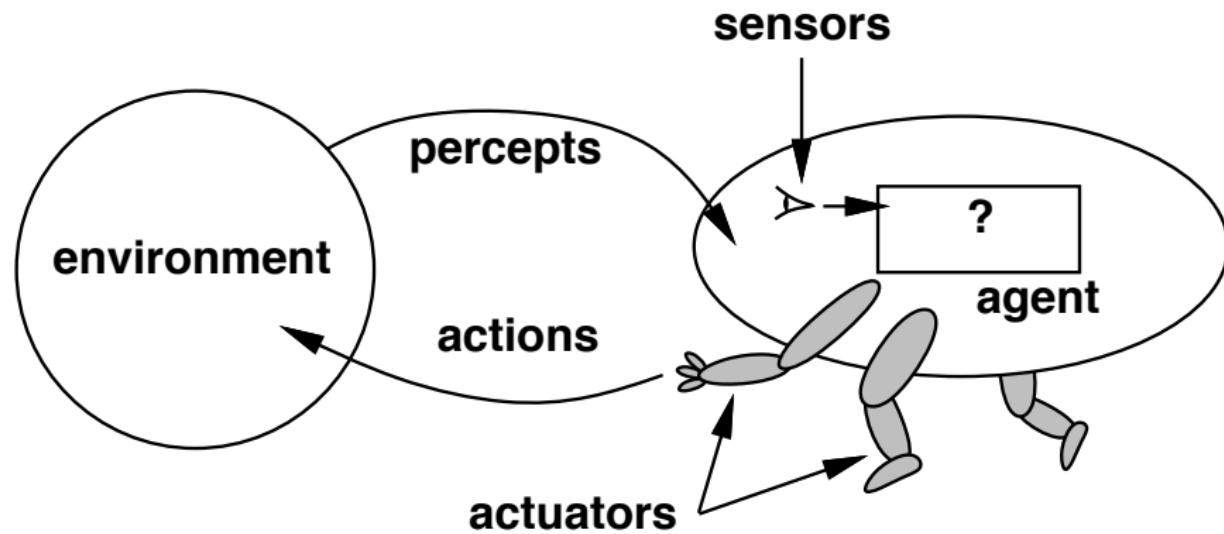
Institut für Visual & Analytic Computing

Agenten

Ein Agent...

- Handelt autonom
- Nimmt seine Umgebung wahr
- Existiert über einen Zeitraum
- Passt sich an Veränderungen an
- Verfolgt Ziele

Agent und Umgebung



Agenten sind Menschen, Thermostate, Software-Agenten, Roboter, ...

Agent und Umgebung

Agentenfunktion und Agentenprogramm

Die Aktion, die ein Agent auswählt, hängt sinnvollerweise nur von den *bisherigen* Beobachtungen ab. Also:

AgentFunction: [Percept] → Action

Agent und Umgebung

Agentenfunktion und Agentenprogramm

Die Aktion, die ein Agent auswählt, hängt sinnvollerweise nur von den *bisherigen* Beobachtungen ab. Also:

AgentFunction: [Percept] → Action

Das Argument [Percept] ist die Sequenz der bisherigen Beobachtungen.

Agent und Umgebung

Agentenfunktion und Agentenprogramm

Die Aktion, die ein Agent auswählt, hängt sinnvollerweise nur von den *bisherigen* Beobachtungen ab. Also:

AgentFunction: [Percept] → Action

Das Argument [Percept] ist die Sequenz der bisherigen Beobachtungen.

- Eine Funktion vom Typ AgentFunction, eine *Agentenfunktion*, beschreibt *vollständig* das Verhalten eines Agenten.

Agent und Umgebung

Agentenfunktion und Agentenprogramm

Die Aktion, die ein Agent auswählt, hängt sinnvollerweise nur von den *bisherigen* Beobachtungen ab. Also:

AgentFunction: [Percept] → Action

Das Argument [Percept] ist die Sequenz der bisherigen Beobachtungen.

- Eine Funktion vom Typ AgentFunction, eine *Agentenfunktion*, beschreibt *vollständig* das Verhalten eines Agenten.
- *Jeder* mögliche Agent kann durch eine Funktion vom Typ AgentFunction charakterisiert werden.

Agent und Umgebung

Agentenfunktion und Agentenprogramm

Die Aktion, die ein Agent auswählt, hängt sinnvollerweise nur von den *bisherigen* Beobachtungen ab. Also:

AgentFunction: [Percept] → Action

Das Argument [Percept] ist die Sequenz der bisherigen Beobachtungen.

- Eine Funktion vom Typ AgentFunction, eine *Agentenfunktion*, beschreibt *vollständig* das Verhalten eines Agenten.
- Jeder mögliche Agent kann durch eine Funktion vom Typ AgentFunction charakterisiert werden.
- Die *Agentenfunktion* (ein abstraktes mathematisches Konstrukt) wird durch ein *Agentenprogramm* implementiert, das auf einer physischen Architektur ausgeführt wird.

Agent und Umgebung

Agentenfunktion und Agentenprogramm

Die Aktion, die ein Agent auswählt, hängt sinnvollerweise nur von den *bisherigen* Beobachtungen ab. Also:

AgentFunction: [Percept] → Action

Das Argument [Percept] ist die Sequenz der bisherigen Beobachtungen.

- Eine Funktion vom Typ AgentFunction, eine *Agentenfunktion*, beschreibt *vollständig* das Verhalten eines Agenten.
- Jeder mögliche Agent kann durch eine Funktion vom Typ AgentFunction charakterisiert werden.
- Die *Agentenfunktion* (ein abstraktes mathematisches Konstrukt) wird durch ein *Agentenprogramm* implementiert, das auf einer physischen Architektur ausgeführt wird.
- Nicht für jede denkbare *Agentenfunktion* kann ein *Agentenprogramm* gefunden werden. (Berechenbarkeit)

Nutzen und Rationalität

Rationales Handeln ist definiert in Bezug auf ein vorher festgelegtes Erfolgsmaß, den *Nutzen*.

Nutzen und Rationalität

Rationales Handeln ist definiert in Bezug auf ein vorher festgelegtes Erfolgsmaß, den *Nutzen*. Dieser Nutzen ist definiert als Funktion der Folge von Umgebungszuständen. Also Allgemein:

Utility : [WorldState] → Double

Nutzen und Rationalität

Rationales Handeln ist definiert in Bezug auf ein vorher festgelegtes Erfolgsmaß, den *Nutzen*. Dieser Nutzen ist definiert als Funktion der Folge von Umgebungszuständen. Also Allgemein:

Utility : [WorldState] → Double

Gegeben eine *Folge von Beobachtungen* und ein *Modell des Umgebungsverhaltens* wählt ein *rationaler Agent* diejenige Aktion, die den *erwarteten Nutzen* maximiert.

Nutzen und Rationalität

Rationales Handeln ist definiert in Bezug auf ein vorher festgelegtes Erfolgsmaß, den *Nutzen*. Dieser Nutzen ist definiert als Funktion der Folge von Umgebungszuständen. Also Allgemein:

Utility : [WorldState] → Double

Gegeben eine *Folge von Beobachtungen* und ein *Modell des Umgebungsverhaltens* wählt ein *rationaler Agent* diejenige Aktion, die den *erwarteten Nutzen* maximiert. Da die Welt nichtdeterministisch auf Aktionen des Agenten reagieren kann, ist es klar, dass wir im Allgemeinen nur einen *Erwartungswert* maximieren können.

Rationalität

Unterscheidung zwischen Rationalität und *Allwissenheit* sinnvoll:

Ich laufe die Straße entlang. Auf der anderen Straßenseite sehe ich einen Freund, den ich begrüßen will. Ich schaue nach links und rechts, kein Verkehr zu sehen. Also überquere ich die Straße, und werde von einem herabfallenden Flugzeugteil erschlagen.

Sollte in meinem Nachruf stehen, dass ich mich irrational verhalten habe, weil ich die Straße überquert habe?

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.
 - Beobachtungen liefern möglicherweise nicht alle benötigten Informationen.

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.
 - Beobachtungen liefern möglicherweise nicht alle benötigten Informationen.
 - Das Modell des Umgebungsverhaltens ist möglicherweise unvollständig oder fehlerhaft.

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.
 - Beobachtungen liefern möglicherweise nicht alle benötigten Informationen.
 - Das Modell des Umgebungsverhaltens ist möglicherweise unvollständig oder fehlerhaft.
- Rationalität bedeutet *nicht* Hellseherei

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.
 - Beobachtungen liefern möglicherweise nicht alle benötigten Informationen.
 - Das Modell des Umgebungsverhaltens ist möglicherweise unvollständig oder fehlerhaft.
- Rationalität bedeutet *nicht* Hellseherei
 - Aktionen liefern (in nichtdeterministischen und / oder nur partiell beobachtbaren Umgebungen) nicht sicher das erwartete Ergebnis.

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.
 - Beobachtungen liefern möglicherweise nicht alle benötigten Informationen.
 - Das Modell des Umgebungsverhaltens ist möglicherweise unvollständig oder fehlerhaft.
- Rationalität bedeutet *nicht* Hellseherei
 - Aktionen liefern (in nichtdeterministischen und / oder nur partiell beobachtbaren Umgebungen) nicht sicher das erwartete Ergebnis.
- (perfekt) *rationales* Handeln heißt also *nicht* notwendig (absolut) *perfektes* Handeln.

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.
 - Beobachtungen liefern möglicherweise nicht alle benötigten Informationen.
 - Das Modell des Umgebungsverhaltens ist möglicherweise unvollständig oder fehlerhaft.
- Rationalität bedeutet *nicht* Hellseherei
 - Aktionen liefern (in nichtdeterministischen und / oder nur partiell beobachtbaren Umgebungen) nicht sicher das erwartete Ergebnis.
- (perfekt) *rationales* Handeln heißt also *nicht* notwendig (absolut) *perfektes* Handeln.
 - (perfekt) rationales Handeln maximiert den *erwarteten* Nutzen.

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.
 - Beobachtungen liefern möglicherweise nicht alle benötigten Informationen.
 - Das Modell des Umgebungsverhaltens ist möglicherweise unvollständig oder fehlerhaft.
- Rationalität bedeutet *nicht* Hellseherei
 - Aktionen liefern (in nichtdeterministischen und / oder nur partiell beobachtbaren Umgebungen) nicht sicher das erwartete Ergebnis.
- (perfekt) *rationales* Handeln heißt also *nicht* notwendig (absolut) *perfektes* Handeln.
 - (perfekt) rationales Handeln maximiert den *erwarteten* Nutzen.
 - (absolut) perfektes Handeln maximiert den *tatsächlichen* Nutzen

Rationalität

- Rationalität bedeutet *nicht* Allwissenheit
 - Ein allwissender Agent kennt den tatsächlichen Effekt jeder Aktion.
 - Beobachtungen liefern möglicherweise nicht alle benötigten Informationen.
 - Das Modell des Umgebungsverhaltens ist möglicherweise unvollständig oder fehlerhaft.
- Rationalität bedeutet *nicht* Hellseherei
 - Aktionen liefern (in nichtdeterministischen und / oder nur partiell beobachtbaren Umgebungen) nicht sicher das erwartete Ergebnis.
- (perfekt) *rationales* Handeln heißt also *nicht* notwendig (absolut) *perfektes* Handeln.
 - (perfekt) rationales Handeln maximiert den *erwarteten* Nutzen.
 - (absolut) perfektes Handeln maximiert den *tatsächlichen* Nutzen

ohne Hellseherei lässt sich perfektes Handeln in nichtdeterministischen Umgebungen nicht erreichen.

Rationalität

Exploration, Lernen, Autonomie

Gegeben entsprechende Aktionen, kann ein rationaler Agent sich dafür entscheiden, vor kritischen Entscheidungen fehlendes Wissen zu beschaffen:

Rationalität

Exploration, Lernen, Autonomie

Gegeben entsprechende Aktionen, kann ein rationaler Agent sich dafür entscheiden, vor kritischen Entscheidungen fehlendes Wissen zu beschaffen:

- *Exploration* der Umgebung

Rationalität

Exploration, Lernen, Autonomie

Gegeben entsprechende Aktionen, kann ein rationaler Agent sich dafür entscheiden, vor kritischen Entscheidungen fehlendes Wissen zu beschaffen:

- *Exploration* der Umgebung
- *Lernen* neuer Tatsachen über das Verhalten der Umgebung

Rationalität

Exploration, Lernen, Autonomie

Gegeben entsprechende Aktionen, kann ein rationaler Agent sich dafür entscheiden, vor kritischen Entscheidungen fehlendes Wissen zu beschaffen:

- *Exploration* der Umgebung
- *Lernen* neuer Tatsachen über das Verhalten der Umgebung

Die *Autonomie* eines Agenten beschreibt seine Fähigkeit, sein Verhalten im Lauf der Zeit an neuerworbenes Wissen anzupassen.

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: autonomes Taxi:

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: autonomes Taxi:

- Performance (Nutzen):
 - Environment (Umgebung):
 - Actuators (Aktoren, Aktionen):
 - Sensors (Percepts, Beobachtungen):

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: autonomes Taxi:

- Performance (Nutzen):
 - Umsatz, Komfort für Fahrgast, Zeit zum Fahrziel, Anzahl Strafzettel, Anzahl Unfälle, ...
- Environment (Umgebung):
- Actuators (Aktoren, Aktionen):
- Sensors (Percepts, Beobachtungen):

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: autonomes Taxi:

- Performance (Nutzen):
 - Umsatz, Komfort für Fahrgast, Zeit zum Fahrtziel, Anzahl Strafzettel, Anzahl Unfälle, ...
- Environment (Umgebung):
 - Straßenverläufe, andere Verkehrsteilnehmer, Fußgänger, Baustellen, Schlaglöcher, Wildwechsel, Fahrgäste, ...
- Actuators (Aktoren, Aktionen):
- Sensors (Percepts, Beobachtungen):

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: autonomes Taxi:

- Performance (Nutzen):
 - Umsatz, Komfort für Fahrgast, Zeit zum Fahrziel, Anzahl Strafzettel, Anzahl Unfälle, ...
- Environment (Umgebung):
 - Straßenverläufe, andere Verkehrsteilnehmer, Fußgänger, Baustellen, Schlaglöcher, Wildwechsel, Fahrgäste, ...
- Actuators (Aktoren, Aktionen):
 - Bremse, Gas, Kupplung, Gangschaltung, Lenkung, Hupe, Licht, Fensterheber, Klimaanlage, Radio, ...
- Sensors (Percepts, Beobachtungen):

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: autonomes Taxi:

- Performance (Nutzen):
 - Umsatz, Komfort für Fahrgast, Zeit zum Fahrziel, Anzahl Strafzettel, Anzahl Unfälle, ...
- Environment (Umgebung):
 - Straßenverläufe, andere Verkehrsteilnehmer, Fußgänger, Baustellen, Schlaglöcher, Wildwechsel, Fahrgäste, ...
- Actuators (Aktoren, Aktionen):
 - Bremse, Gas, Kupplung, Gangschaltung, Lenkung, Hupe, Licht, Fensterheber, Klimaanlage, Radio, ...
- Sensors (Percepts, Beobachtungen):
 - Kameras, Laser-Entfernungsmesser, GPS, Radar, Akzelerometer, Öltemperatur, Wassertemperatur, Tankanzeiger, Innenraumüberwachung, Tastatur, Mikrofon, ...

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: Internet-Shopping-Agent

- Performance (Nutzen):
- Environment (Umgebung):
- Actuators (Aktoren, Aktionen):
- Sensors (Percepts, Beobachtungen):

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: Internet-Shopping-Agent

- Performance (Nutzen):
 - Preis, Qualität, Nützlichkeit, Lieferzeitpunkt, ...
- Environment (Umgebung):
- Actuators (Aktoren, Aktionen):
- Sensors (Percepts, Beobachtungen):

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: Internet-Shopping-Agent

- Performance (Nutzen):
 - Preis, Qualität, Nützlichkeit, Lieferzeitpunkt, ...
- Environment (Umgebung):
 - Web-Shops
- Actuators (Aktoren, Aktionen):
- Sensors (Percepts, Beobachtungen):

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: Internet-Shopping-Agent

- Performance (Nutzen):
 - Preis, Qualität, Nützlichkeit, Lieferzeitpunkt, ...
- Environment (Umgebung):
 - Web-Shops
- Actuators (Aktoren, Aktionen):
 - HTTP-Kommandos; Interaktion mit HTML-Seite
- Sensors (Percepts, Beobachtungen):

Die Aufgaben-Umgebung

PEAS: Performance, Environment, Actuators, Sensors

Für die Entwicklung eines rationalen Agenten muss die Aufgaben-Umgebung festgelegt werden.

Beispiel: Internet-Shopping-Agent

- Performance (Nutzen):
 - Preis, Qualität, Nützlichkeit, Lieferzeitpunkt, ...
- Environment (Umgebung):
 - Web-Shops
- Actuators (Aktoren, Aktionen):
 - HTTP-Kommandos; Interaktion mit HTML-Seite
- Sensors (Percepts, Beobachtungen):
 - HTML-Struktur (Text, Grafik, Skripte)

Umgebungstypen

Umgebungsattribute: „einfach“ vs „schwierig“

Umgebungstypen

Umgebungsattribute: „einfach“ vs „schwierig“

- Vollständige vs. partielle Observierbarkeit?

Umgebungstypen

Umgebungsattribute: „einfach“ vs „schwierig“

- Vollständige vs. partielle Observierbarkeit?
- Deterministische vs. nicht-deterministische Reaktion auf Aktionen?

Umgebungstypen

Umgebungsattribute: „einfach“ vs „schwierig“

- Vollständige vs. partielle Observierbarkeit?
- Deterministische vs. nicht-deterministische Reaktion auf Aktionen?
 - Wird der Nichtdeterminismus nur durch die Handlung anderer Agenten erzeugt, ist die Umgebung *strategisch*.

Umgebungstypen

Umgebungsattribute: „einfach“ vs „schwierig“

- Vollständige vs. partielle Observierbarkeit?
- Deterministische vs. nicht-deterministische Reaktion auf Aktionen?
 - Wird der Nichtdeterminismus nur durch die Handlung anderer Agenten erzeugt, ist die Umgebung *strategisch*.
- Episodische vs. sequentielle Aufgabenstellungen?
Kann jede Beobachtung für sich allein bearbeitet werden oder hängt die Reaktion auf eine Beobachtung auch von vorherigen Aktivitäten ab?

Umgebungstypen

Umgebungsattribute: „einfach“ vs „schwierig“

- Vollständige vs. partielle Observierbarkeit?
- Deterministische vs. nicht-deterministische Reaktion auf Aktionen?
 - Wird der Nichtdeterminismus nur durch die Handlung anderer Agenten erzeugt, ist die Umgebung *strategisch*.
- Episodische vs. sequentielle Aufgabenstellungen?
Kann jede Beobachtung für sich allein bearbeitet werden oder hängt die Reaktion auf eine Beobachtung auch von vorherigen Aktivitäten ab?
- Statische vs. dynamische Umgebung?
Kann sich während des Aktionsauswahlprozesses die Umgebung verändern?

Umgebungstypen

Umgebungsattribute: „einfach“ vs „schwierig“

- Vollständige vs. partielle Observierbarkeit?
- Deterministische vs. nicht-deterministische Reaktion auf Aktionen?
 - Wird der Nichtdeterminismus nur durch die Handlung anderer Agenten erzeugt, ist die Umgebung *strategisch*.
- Episodische vs. sequentielle Aufgabenstellungen?
Kann jede Beobachtung für sich allein bearbeitet werden oder hängt die Reaktion auf eine Beobachtung auch von vorherigen Aktivitäten ab?
- Statische vs. dynamische Umgebung?
Kann sich während des Aktionsauswahlprozesses die Umgebung verändern?
- Diskrete oder kontinuierliche Zustands- / Beobachtungsräume?

Umgebungstypen

Umgebungsattribute: „einfach“ vs „schwierig“

- Vollständige vs. partielle Observierbarkeit?
- Deterministische vs. nicht-deterministische Reaktion auf Aktionen?
 - Wird der Nichtdeterminismus nur durch die Handlung anderer Agenten erzeugt, ist die Umgebung *strategisch*.
- Episodische vs. sequentielle Aufgabenstellungen?
Kann jede Beobachtung für sich allein bearbeitet werden oder hängt die Reaktion auf eine Beobachtung auch von vorherigen Aktivitäten ab?
- Statische vs. dynamische Umgebung?
Kann sich während des Aktionsauswahlprozesses die Umgebung verändern?
- Diskrete oder kontinuierliche Zustands- / Beobachtungsräume?
- Single-Agent vs. Multi-Agent-Situation?
Gibt es Mitspieler oder nur eine “indifferente” Umgebung?
Kooperieren Mitspieler für eine Lösung, oder stehen sie im Wettbewerb?

Umgebungstypen

Single-Agent vs. Multi-Agent

Was ist für einen Agenten A der Unterschied zwischen einer indifferenten nichtdeterministischen Umgebung und einem anderen Agenten B ?

Umgebungstypen

Single-Agent vs. Multi-Agent

Was ist für einen Agenten A der Unterschied zwischen einer indifferenten nichtdeterministischen Umgebung und einem anderen Agenten B ?

- Ein anderer Agent wird potentiell versuchen, ein Leistungsmaß zu optimieren. Dabei kann das Leistungsmaß von B vom Leistungsmaß für A abhängen:
 - Kooperation: positiv korreliert
 - Wettbewerb: negativ korreliert

Umgebungstypen

Single-Agent vs. Multi-Agent

Was ist für einen Agenten A der Unterschied zwischen einer indifferenten nichtdeterministischen Umgebung und einem anderen Agenten B ?

- Ein anderer Agent wird potentiell versuchen, ein Leistungsmaß zu optimieren. Dabei kann das Leistungsmaß von B vom Leistungsmaß für A abhängen:
 - Kooperation: positiv korreliert
 - Wettbewerb: negativ korreliert
- D.h., es liegt möglicherweise zusätzliches Wissen über die Reaktion der Umgebung vor, das A für die Aktionsauswahl verwenden kann.

Umgebungstypen

Single-Agent vs. Multi-Agent

Was ist für einen Agenten A der Unterschied zwischen einer indifferenten nichtdeterministischen Umgebung und einem anderen Agenten B ?

- Ein anderer Agent wird potentiell versuchen, ein Leistungsmaß zu optimieren. Dabei kann das Leistungsmaß von B vom Leistungsmaß für A abhängen:
 - Kooperation: positiv korreliert
 - Wettbewerb: negativ korreliert
- D.h., es liegt möglicherweise zusätzliches Wissen über die Reaktion der Umgebung vor, das A für die Aktionsauswahl verwenden kann.
- Die *Kommunikation* zwischen Agenten (zum Zweck der Kooperation) kann sinnvoll sein.

Umgebungstypen

Single-Agent vs. Multi-Agent

Was ist für einen Agenten A der Unterschied zwischen einer indifferenten nichtdeterministischen Umgebung und einem anderen Agenten B ?

- Ein anderer Agent wird potentiell versuchen, ein Leistungsmaß zu optimieren. Dabei kann das Leistungsmaß von B vom Leistungsmaß für A abhängen:
 - Kooperation: positiv korreliert
 - Wettbewerb: negativ korreliert
- D.h., es liegt möglicherweise zusätzliches Wissen über die Reaktion der Umgebung vor, das A für die Aktionsauswahl verwenden kann.
- Die *Kommunikation* zwischen Agenten (zum Zweck der Kooperation) kann sinnvoll sein.
- Absichtlich *nichtdeterministisches* Verhalten von A kann sinnvoll sein (im Fall von Wettbewerbssituationen).

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?				

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?				

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?	ja	nein	partiell	nein
Episodisch?				

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?	ja	nein	partiell	nein
Episodisch?	nein	nein	nein	nein
Statisch				

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?	ja	nein	partiell	nein
Episodisch?	nein	nein	nein	nein
Statisch	ja	ja	nein	nein
Diskret?				

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?	ja	nein	partiell	nein
Episodisch?	nein	nein	nein	nein
Statisch	ja	ja	nein	nein
Diskret?	ja	ja	ja	nein
Single-Agent?				

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?	ja	nein	partiell	nein
Episodisch?	nein	nein	nein	nein
Statisch	ja	ja	nein	nein
Diskret?	ja	ja	ja	nein
Single-Agent?	ja	nein	ja	nein

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?	ja	nein	partiell	nein
Episodisch?	nein	nein	nein	nein
Statisch	ja	ja	nein	nein
Diskret?	ja	ja	ja	nein
Single-Agent?	ja	nein	ja	nein

- Der Umgebungstyp bestimmt weitgehend den Entwurf des Agenten

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?	ja	nein	partiell	nein
Episodisch?	nein	nein	nein	nein
Statisch	ja	ja	nein	nein
Diskret?	ja	ja	ja	nein
Single-Agent?	ja	nein	ja	nein

- Der Umgebungstyp bestimmt weitgehend den Entwurf des Agenten
- Viele deterministische Systeme sind so kompliziert, dass man sie als nicht-deterministische-Systeme behandeln muss.

Umgebungstypen

Einige Beispiele

	Solitär	Backgammon	Internet-Shopping	Taxi
Observabel?	ja	ja	nein	nein
Deterministisch?	ja	nein	partiell	nein
Episodisch?	nein	nein	nein	nein
Statisch	ja	ja	nein	nein
Diskret?	ja	ja	ja	nein
Single-Agent?	ja	nein	ja	nein

- Der Umgebungstyp bestimmt weitgehend den Entwurf des Agenten
- Viele deterministische Systeme sind so kompliziert, dass man sie als nicht-deterministische-Systeme behandeln muss.
- Die reale Welt ist selbstverständlich: nur partiell observierbar, nichtdeterministisch, sequentiell, dynamisch, kontinuierlich, multi-agent.

Agententypen

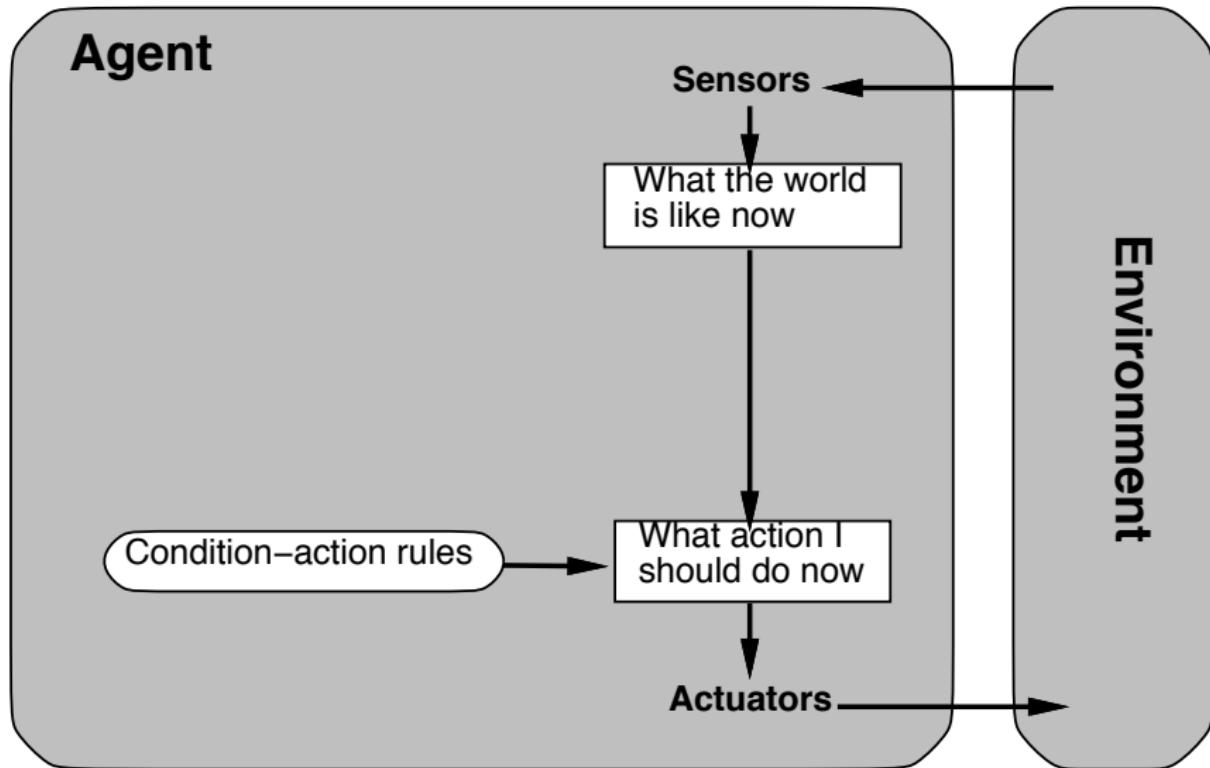
Eine Vielzahl von Agentenarchitekturen sind denkbar; wir schauen uns exemplarisch vier Varianten wachsender Komplexität an:

- Einfache Reflex-Agenten
- Zustandsbehaftete Reflex-Agenten
(modellbasierte Reflex-Agenten)
- Zielbasierte Agenten
- Nutzenbasierte Agenten

(Alle diese grundlegenden Architekturen können auch zu lernfähigen Systemen erweitert werden.)

Reflex-Agenten

Aufbau



Reflex-Agenten

Diskussion

Reflexagenten funktionieren nur, wenn die Aktion *eindeutig* auf Basis der aktuellen Beobachtung gewählt werden kann. D.h., wenn die Umgebung – in Bezug auf den Handlungsspielraum des Agenten – vollständig beobachtbar ist.

Reflex-Agenten

Diskussion

Reflexagenten funktionieren nur, wenn die Aktion *eindeutig* auf Basis der aktuellen Beobachtung gewählt werden kann. D.h., wenn die Umgebung – in Bezug auf den Handlungsspielraum des Agenten – vollständig beobachtbar ist.

Die Lösung ist, wir geben dem Agenten ein *Modell* von der Entwicklung der Teile der Welt, die er nicht beobachten kann.

Reflex-Agenten

Diskussion

Reflexagenten funktionieren nur, wenn die Aktion *eindeutig* auf Basis der aktuellen Beobachtung gewählt werden kann. D.h., wenn die Umgebung – in Bezug auf den Handlungsspielraum des Agenten – vollständig beobachtbar ist.

Die Lösung ist, wir geben dem Agenten ein *Modell* von der Entwicklung der Teile der Welt, die er nicht beobachten kann. Ein solches Modell berücksichtigt im Allgemeinen sowohl die Wirkung der Aktionen des Agenten, wie auch die Eigendynamik der Umgebung.

Reflex-Agenten

Diskussion

Reflexagenten funktionieren nur, wenn die Aktion *eindeutig* auf Basis der aktuellen Beobachtung gewählt werden kann. D.h., wenn die Umgebung – in Bezug auf den Handlungsspielraum des Agenten – vollständig beobachtbar ist.

Die Lösung ist, wir geben dem Agenten ein *Modell* von der Entwicklung der Teile der Welt, die er nicht beobachten kann. Ein solches Modell berücksichtigt im Allgemeinen sowohl die Wirkung der Aktionen des Agenten, wie auch die Eigendynamik der Umgebung. Für die Teile der Welt, die *nicht* von der aktuellen Beobachtung geliefert werden, muss der Agent ein internes Bild verwalten und ständig Aktualisieren – dies ist der *Zustand*.

Reflex-Agenten

Diskussion

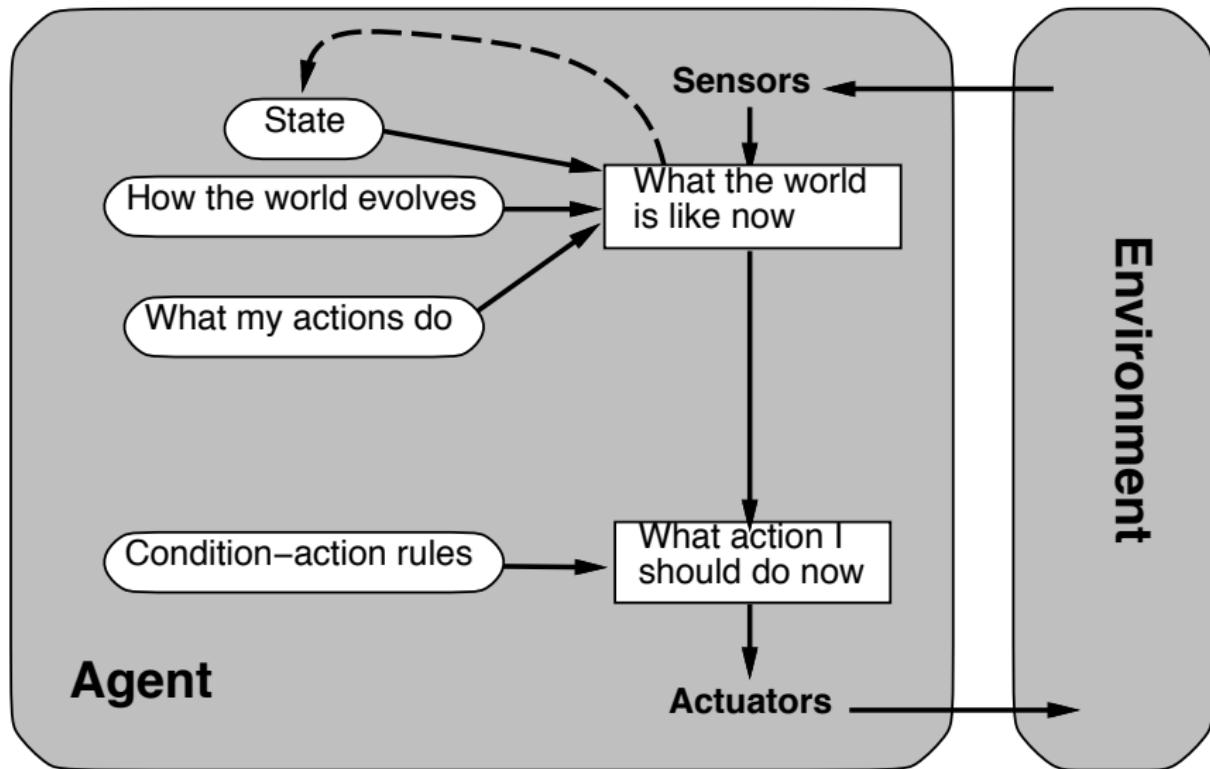
Reflexagenten funktionieren nur, wenn die Aktion *eindeutig* auf Basis der aktuellen Beobachtung gewählt werden kann. D.h., wenn die Umgebung – in Bezug auf den Handlungsspielraum des Agenten – vollständig beobachtbar ist.

Die Lösung ist, wir geben dem Agenten ein *Modell* von der Entwicklung der Teile der Welt, die er nicht beobachten kann. Ein solches Modell berücksichtigt im Allgemeinen sowohl die Wirkung der Aktionen des Agenten, wie auch die Eigendynamik der Umgebung. Für die Teile der Welt, die *nicht* von der aktuellen Beobachtung geliefert werden, muss der Agent ein internes Bild verwalten und ständig Aktualisieren – dies ist der *Zustand*.

Um genau zu sein sollte man zwischen dem *wahren* Zustand der Welt und dem *Bild*, das der Agent von diesem Zustand hat, unterscheiden.

Zustandsbehaftete Reflex-Agenten

Aufbau



Zustandsbehaftete Reflex-Agenten

Diskussion

- Gegeben einen Zustand und eine Beobachtung wählt ein zustandsbehafteter Reflex-Agent stets dieselbe Aktion.

Zustandsbehaftete Reflex-Agenten

Diskussion

- Gegeben einen Zustand und eine Beobachtung wählt ein zustandsbehafteter Reflex-Agent stets dieselbe Aktion.
- Jedoch sind Zustand und Beobachtung nicht immer ausreichend für die Aktionsauswahl.

Zustandsbehaftete Reflex-Agenten

Diskussion

- Gegeben einen Zustand und eine Beobachtung wählt ein zustandsbehafteter Reflex-Agent stets dieselbe Aktion.
- Jedoch sind Zustand und Beobachtung nicht immer ausreichend für die Aktionsauswahl. In welche Richtung ein autonomes Taxi an einer Kreuzung abbiegt hängt nicht nur vom Zustand (Position des Fahrzeugs) und der Beobachtung (Kreuzung frei) ab, sondern auch vom *Ziel* des Fahrgastes.

Zustandsbehaftete Reflex-Agenten

Diskussion

- Gegeben einen Zustand und eine Beobachtung wählt ein zustandsbehafteter Reflex-Agent stets dieselbe Aktion.
- Jedoch sind Zustand und Beobachtung nicht immer ausreichend für die Aktionsauswahl. In welche Richtung ein autonomes Taxi an einer Kreuzung abbiegt hängt nicht nur vom Zustand (Position des Fahrzeugs) und der Beobachtung (Kreuzung frei) ab, sondern auch vom *Ziel* des Fahrgastes.
- In diesem Fall muss ein Agent auch in der Lage sein zu bewerten, inwieweit eine der möglichen Aktionen ihn dabei unterstützt, sein aktuelles Ziel zu erreichen.

Zustandsbehaftete Reflex-Agenten

Diskussion

- Gegeben einen Zustand und eine Beobachtung wählt ein zustandsbehafteter Reflex-Agent stets dieselbe Aktion.
- Jedoch sind Zustand und Beobachtung nicht immer ausreichend für die Aktionsauswahl. In welche Richtung ein autonomes Taxi an einer Kreuzung abbiegt hängt nicht nur vom Zustand (Position des Fahrzeugs) und der Beobachtung (Kreuzung frei) ab, sondern auch vom *Ziel* des Fahrgastes.
- In diesem Fall muss ein Agent auch in der Lage sein zu bewerten, inwieweit eine der möglichen Aktionen ihn dabei unterstützt, sein aktuelles Ziel zu erreichen.
Dazu muss er extrapolieren, wie seine Aktionen die Welt verändern, und bewerten, ob ihn diese veränderte Welt seinem Ziel näher bringt.

Zielbasierte Agenten

Konzept

- Ein zielbasierter Agent ist in der Lage, die Wirkung seiner Aktionen einzuschätzen *bevor er diese real ausführt* – er kann somit über sein zukünftiges Handeln spekulieren.

Zielbasierte Agenten

Konzept

- Ein zielbasierter Agent ist in der Lage, die Wirkung seiner Aktionen einzuschätzen *bevor er diese real ausführt* – er kann somit über sein zukünftiges Handeln spekulieren.
- Techniken der *Suche* und *Planung* werden genutzt, um für einen gegebenen Anfangszustand, einen gewünschten Zielzustand und eine Menge von Aktionen eine Aktionsfolge zu finden, die vom Ausgangszustand zum Ziel führt.

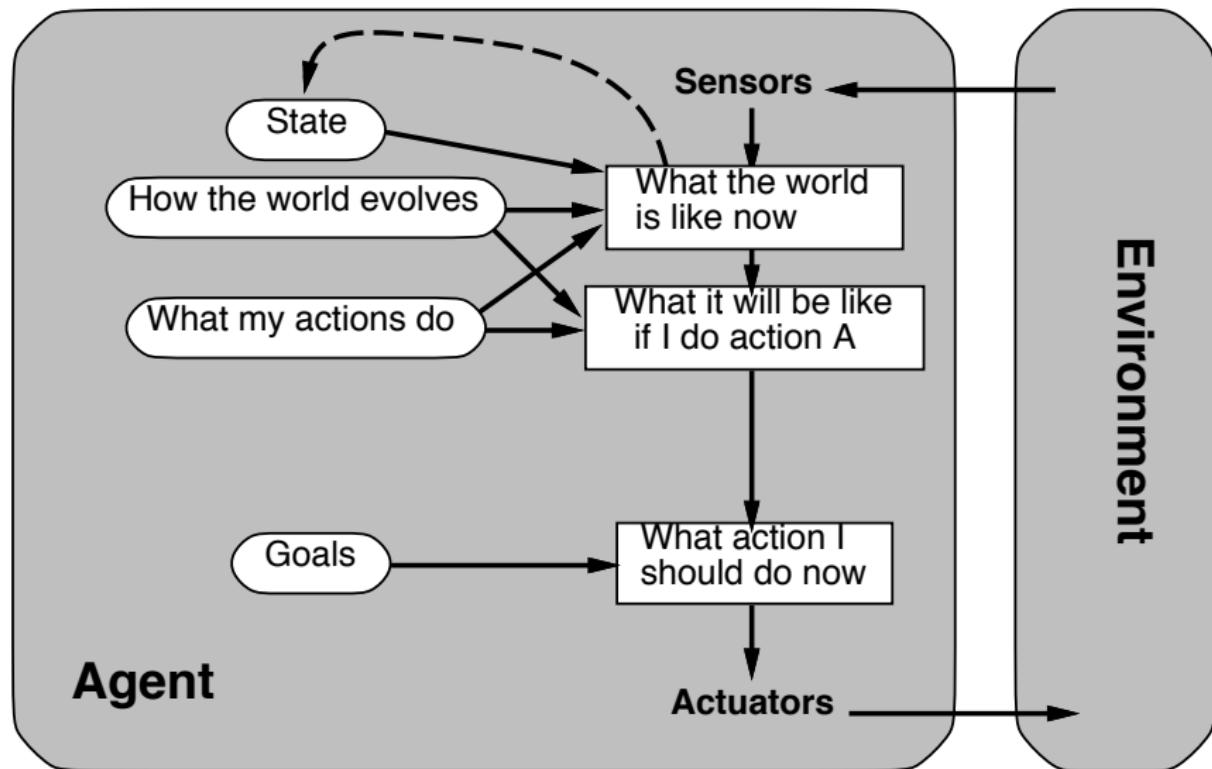
Zielbasierte Agenten

Konzept

- Ein zielbasierter Agent ist in der Lage, die Wirkung seiner Aktionen einzuschätzen *bevor er diese real ausführt* – er kann somit über sein zukünftiges Handeln spekulieren.
- Techniken der *Suche* und *Planung* werden genutzt, um für einen gegebenen Anfangszustand, einen gewünschten Zielzustand und eine Menge von Aktionen eine Aktionsfolge zu finden, die vom Ausgangszustand zum Ziel führt.
- Für reflexbasierte Agenten muss vom Designer vorgegeben werden, in welcher Situation wie reagiert werden muss. Ein zielbasierter Agent kann diese Analyse selbstständig, bei Bedarf durchführen.

Zielbasierte Agenten

Aufbau



Zielbasierte Agenten

Diskussion

- Gegeben ein Ziel kann es sein, dass es unterschiedliche Pläne gibt, die alle dieses Ziel erreichen.

Zielbasierte Agenten

Diskussion

- Gegeben ein Ziel kann es sein, dass es unterschiedliche Pläne gibt, die alle dieses Ziel erreichen.
- Welchen Plan soll der Agent auswählen? Die schnellste Route? Die schönste Route? Die billigste?

Zielbasierte Agenten

Diskussion

- Gegeben ein Ziel kann es sein, dass es unterschiedliche Pläne gibt, die alle dieses Ziel erreichen.
- Welchen Plan soll der Agent auswählen? Die schnellste Route? Die schönste Route? Die billigste?
- Hierzu benötigt er ein Kriterium, das über den *Nutzen* eines Plans Auskunft gibt.
(Ein rein zielbasierter Agent hat ein grobes binäres Kriterium:
„Ziel erreicht: ja/nein?“)

Zielbasierte Agenten

Diskussion

- Gegeben ein Ziel kann es sein, dass es unterschiedliche Pläne gibt, die alle dieses Ziel erreichen.
- Welchen Plan soll der Agent auswählen? Die schnellste Route? Die schönste Route? Die billigste?
- Hierzu benötigt er ein Kriterium, das über den *Nutzen* eines Plans Auskunft gibt.
(Ein rein zielbasierter Agent hat ein grobes binäres Kriterium:
„Ziel erreicht: ja/nein?“)
- Dieses Kriterium ist die Nutzen-Funktion, eine Funktion

Utility: [State] → Double

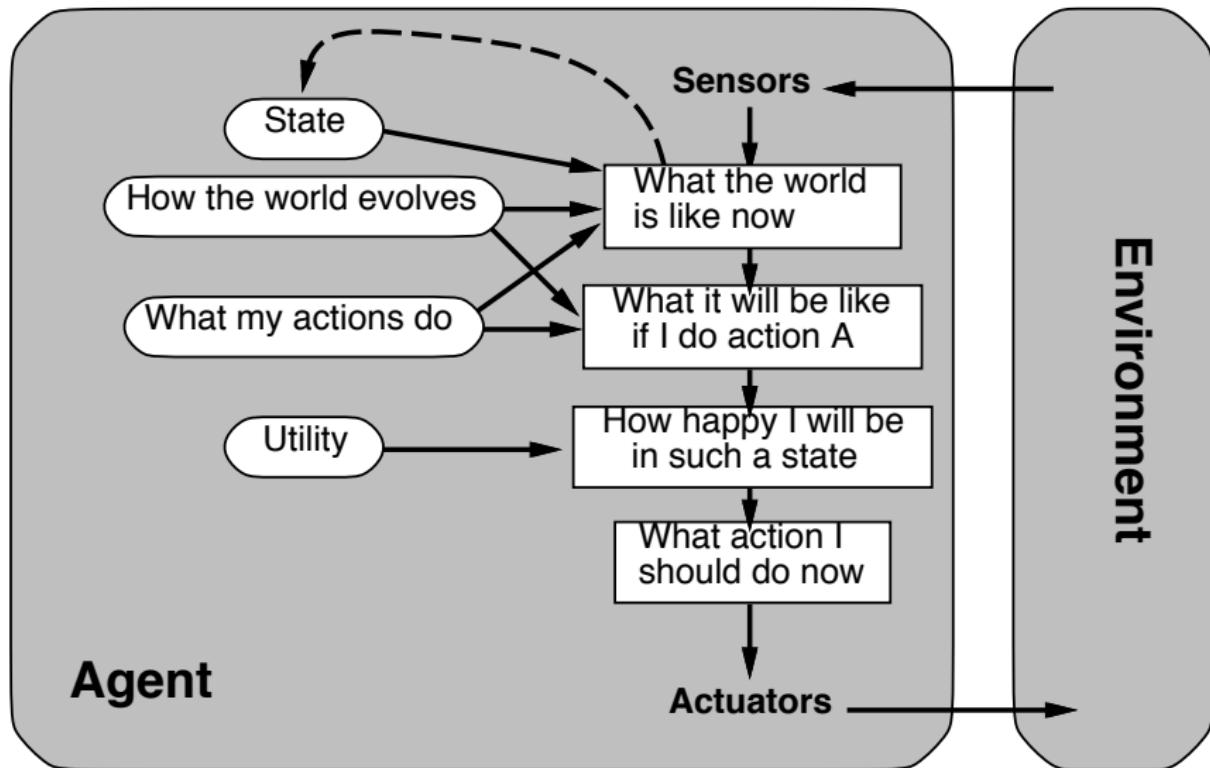
Zielbasierte Agenten

Diskussion

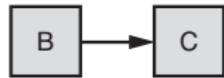
- Gegeben ein Ziel kann es sein, dass es unterschiedliche Pläne gibt, die alle dieses Ziel erreichen.
- Welchen Plan soll der Agent auswählen? Die schnellste Route? Die schönste Route? Die billigste?
- Hierzu benötigt er ein Kriterium, das über den *Nutzen* eines Plans Auskunft gibt.
(Ein rein zielbasierter Agent hat ein grobes binäres Kriterium:
„Ziel erreicht: ja/nein?“)
- Dieses Kriterium ist die Nutzen-Funktion, eine Funktion
 $\text{Utility} : [\text{State}] \rightarrow \text{Double}$
- Auf dieser Basis kann ein Agent nun zwischen verschiedenen möglichen Plänen wählen, oder auch zwischen verschiedenen, zueinander in Konflikt stehenden Zielen auswählen.

Nutzenbasierte Agenten

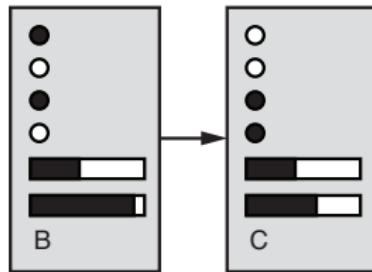
Aufbau



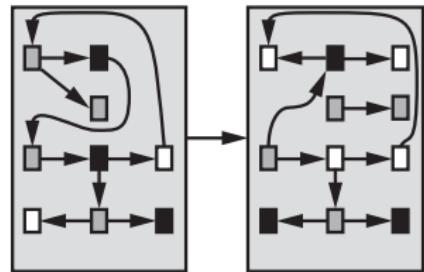
Was ist ein Zustand?



(a) Atomic



(b) Factored

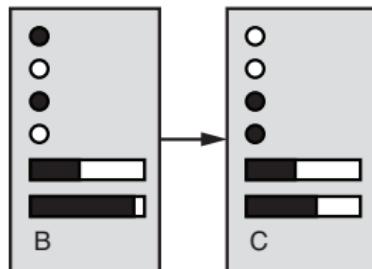


(b) Structured

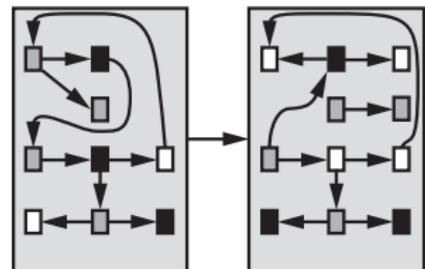
Was ist ein Zustand?



(a) Atomic



(b) Factored



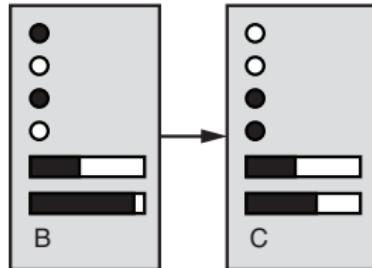
(b) Structured

- 1 Atomare Repräsentation. Zustände (B, C) sind Black Boxes ohne interne Struktur

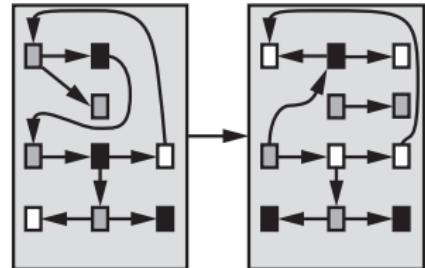
Was ist ein Zustand?



(a) Atomic



(b) Factored



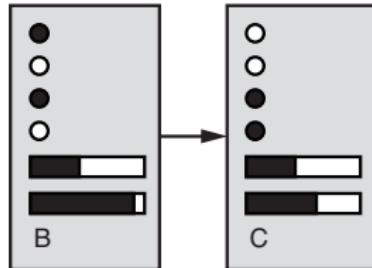
(b) Structured

- 1 Atomare Repräsentation. Zustände (B, C) sind Black Boxes ohne interne Struktur
- 2 Faktorierte Repräsentation. Zustände sind Abbildungen von Attributen auf Werte; Werte können Wahrheitswerte, Symbole, Fließkommazahlen sein. (Beispielsweise realisiert als C struct)

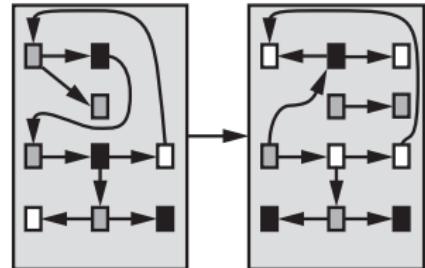
Was ist ein Zustand?



(a) Atomic



(b) Factored



(c) Structured

- 1 Atomare Repräsentation. Zustände (B, C) sind Black Boxes ohne interne Struktur
- 2 Faktorierte Repräsentation. Zustände sind Abbildungen von Attributen auf Werte; Werte können Wahrheitswerte, Symbole, Fließkommazahlen sein. (Beispielsweise realisiert als C struct)
- 3 Zustände sind beliebige programmiersprachliche Datenstrukturen. (Beispielsweise als CLOS, C++, Java Objektstruktur)

Zusammenfassung

- Agenten interagieren mit ihrer *Umgebung* über *Sensoren* und *Aktoren*

Zusammenfassung

- Agenten interagieren mit ihrer *Umgebung* über *Sensoren* und *Aktoren*
- Die *Agentenfunktion* (`AgentFunction`) ist eine *vollständige* Beschreibung des Agentenverhaltens

Zusammenfassung

- Agenten interagieren mit ihrer *Umgebung* über *Sensoren* und *Aktoren*
- Die *Agentenfunktion* (`AgentFunction`) ist eine *vollständige* Beschreibung des Agentenverhaltens
- Die *Nutzenfunktion* bewertet die Folge von Umgebungszuständen

Zusammenfassung

- Agenten interagieren mit ihrer *Umgebung* über *Sensoren* und *Aktoren*
- Die *Agentenfunktion* (`AgentFunction`) ist eine *vollständige* Beschreibung des Agentenverhaltens
- Die *Nutzenfunktion* bewertet die Folge von Umgebungszuständen
- Ein *perfekt rationaler Agent* maximiert den *erwarteten Nutzen*

Zusammenfassung

- Agenten interagieren mit ihrer *Umgebung* über *Sensoren* und *Aktoren*
- Die *Agentenfunktion* (`AgentFunction`) ist eine *vollständige* Beschreibung des Agentenverhaltens
- Die *Nutzenfunktion* bewertet die Folge von Umgebungszuständen
- Ein *perfekt rationaler Agent* maximiert den *erwarteten Nutzen*
- *Agentenprogramme* implementieren (einige) Agentenfunktionen

Zusammenfassung

- Agenten interagieren mit ihrer *Umgebung* über *Sensoren* und *Aktoren*
- Die *Agentenfunktion* (`AgentFunction`) ist eine *vollständige* Beschreibung des Agentenverhaltens
- Die *Nutzenfunktion* bewertet die Folge von Umgebungszuständen
- Ein *perfekt rationaler Agent* maximiert den *erwarteten Nutzen*
- *Agentenprogramme* implementieren (einige) Agentenfunktionen
- *PEAS-Beschreibungen* definieren die Aufgaben-Umgebung eines Agenten

Zusammenfassung

- Agenten interagieren mit ihrer *Umgebung* über *Sensoren* und *Aktoren*
- Die *Agentenfunktion* (`AgentFunction`) ist eine *vollständige* Beschreibung des Agentenverhaltens
- Die *Nutzenfunktion* bewertet die Folge von Umgebungszuständen
- Ein *perfekt rationaler Agent* maximiert den *erwarteten Nutzen*
- *Agentenprogramme* implementieren (einige) Agentenfunktionen
- *PEAS*-Beschreibungen definieren die Aufgaben-Umgebung eines Agenten
- Umgebungen können in Bezug auf verschiedene Attribute kategorisiert werden:
 - beobachtbar? deterministisch? episodisch? statisch? diskret?
 - Single-Agent?

Zusammenfassung

- Agenten interagieren mit ihrer *Umgebung* über *Sensoren* und *Aktoren*
- Die *Agentenfunktion* (`AgentFunction`) ist eine *vollständige* Beschreibung des Agentenverhaltens
- Die *Nutzenfunktion* bewertet die Folge von Umgebungszuständen
- Ein *perfekt rationaler Agent* maximiert den *erwarteten Nutzen*
- *Agentenprogramme* implementieren (einige) Agentenfunktionen
- *PEAS*-Beschreibungen definieren die Aufgaben-Umgebung eines Agenten
- Umgebungen können in Bezug auf verschiedene Attribute kategorisiert werden:
 - beobachtbar? deterministisch? episodisch? statisch? diskret?
Single-Agent?
- Es gibt eine Reihe grundlegender Agenten-Architekturen:
 - Reflex, zustandsbehafteter Reflex, zielbasiert, nutzenbasiert

Künstliche Intelligenz

Problemlösung durch Suchen

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Problemlösung durch Suchen

Problemlösende Agenten

- Ein *problemlösender Agent* ist ein Spezialfall des zielbasierten Agenten
- Sucht nach Aktionsfolgen, die einen wünschenswerten Zustand erreichen.
- Varianten der Suche:
 - *nichtinformierte* Suche: Keine Information über den „Abstand“ zwischen wünschenswertem Zustand und anderen Situationen.
 - *informierte* Suche: Abstandsinformation vorhanden.
- Was bedeutet hierbei „Problem“ und „Lösung“?

Problemlösender Agent

Algorithmus am Beispiel

Wir sind in Arad, Rumänien. Was wollen wir tun?

■ 1. Schritt: *Zielformulierung*:

- Möglichst braun werden? Möglichst viel Spaß haben (ohne Kater ...)? Am nächsten Morgen in Bukarest sein (weil wir einen Flug gebucht haben)?
- Welches Ziel der Agent wählt, hängt im Allgemeinen von der Nutzenfunktion ab.
- Im Allgemeinen ist das Ziel definiert durch eine Menge von Weltzuständen. (Z. B. alle möglichen Welten, in denen wir in Bukarest sind.)

■ 2. Schritt: *Problemformulierung*:

- Auswahl der relevanten Aktionen in Abhängigkeit von Weltzustand und Ziel
- Auswahl der relevanten Zustandsaspekte (z. B. „Ort“; aber nicht „Bräunungsgrad“)

Problemlösender Agent

Algorithmus am Beispiel

- 3. Schritt: *Suche einer Lösung*, gegeben Aktionen, Zustandsmenge und Ziel.
 - Diese Lösung ist eine Folge von Aktionen, die den aktuellen Zustand in einen Zustand überführt, der im Ziel liegt.
 - Von Arad führen Straßen nach Sibiu, nach Timisoara und nach Zerind. Welche nehmen wir?
 - Wenn wir keine Karte haben und uns in Rumänien nicht auskennen, ist das beste, was wir tun können, einen zufälligen Weg wählen.
 - Mit einer Karte können wir dagegen überlegen, in welcher Situation wir wären, wenn wir nach Sibiu fahren würden: d.h., welche Aktionen wir von Sibiu aus durchführen könnten.
 - Möglicherweise würden wir über einen solchen Mechanismus schließlich einen Weg nach Bukarest finden.
 - Die Fähigkeit zu einer solchen *hypothetischen* Aktionsausführung – also zur *Simulation* der Wirkung einer Aktionsfolge – ist die fundamentale Fähigkeit eines problemlösenden Agenten.
- 4. Schritt: schrittweise *Ausführung* der Aktionsfolge.

Problemlösender Agent

Funktionaler Aufbau

function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*)

returns: an action

inputs: *percept*, a percept

static: *seq*, an action sequence, initially empty

state, some description of the current world state

goal, a goal, initially null

problem, a problem formulation

state \leftarrow UPDATE-STATE(*state*, *percept*)

if *seq* is empty **then**

goal \leftarrow FORMULATE-GOAL(*state*)

 ▷ Step 1

problem \leftarrow FORMULATE-PROBLEM(*state*, *goal*)

 ▷ Step 2

seq \leftarrow SEARCH(*problem*)

 ▷ Step 3

action \leftarrow FIRST(*seq*)

 ▷ Step 4

seq \leftarrow REST(*seq*)

return *action*

Problemlösender Agent

Implizite Annahmen

- Die Umgebung ist *statisch*.
Keine Aktualisierung von *seq*, während die Aktionen abgearbeitet werden.
- Die Umgebung ist *observabel*.
Genauer: zumindest der initiale Zustand ist soweit bekannt, wie es für die Bestimmung der Aktionsfolge erforderlich ist.
- Die Schritte werden *diskret* ausgeführt.
- Aktionen sind *deterministisch*, soweit es die Ausführbarkeit und die Ergebnisse von Folgeaktionen betrifft.
- Wir haben in diesem Fall ein *Single-State*-Problem: Der Agent weiß stets, in welchem Zustand er sich befindet. Die Lösung ist eine Aktionsfolge.

Folgend betrachten wir FORMULATE-PROBLEM und insbesondere die Realisierung von SEARCH.

FORMULATE-PROBLEM

Wohldefinierte Probleme

Ein wohldefiniertes Problem besteht aus den folgenden Komponenten:

- Ein *Initialzustand* x_0 . Zum Beispiel: $In(Arad)$.
- Eine Beschreibung der möglichen *Aktionen* des Agenten. Dabei ist eine Aktion eine Funktion, die einen Zustand auf einen Nachfolgerzustand abbildet (später: Auf eine Wahrscheinlichkeitsverteilung von Nachfolgerzuständen)
- Initialzustand und Nachfolgerfunktion definieren implizit den *Zustandsraum* X des Problems – die Menge aller Zustände, die von x_0 aus erreichbar sind.
- Ein *Pfad* in X ist eine Folge von Zuständen $\langle x_0, \dots, x_n \rangle$ die durch eine Folge von Aktionen $\langle a_1, \dots, a_n \rangle$ verbunden wird, so dass $x_i = a_i(x_{i-1})$.

FORMULATE-PROBLEM

Wohldefinierte Probleme (Fortsetzung)

- Ein *Zieltest*, der festlegt, ob ein bestimmter Zustand $x \in X$ ein Ziel ist. Der Zieltest definiert eine Menge $G \subseteq X$, die Zielzustände. Beispielsweise $\{In(Bukarest)\}$.
- Eine *Kostenfunktion*, die für jeden Pfad eine Zahl liefert. Die Wahl der Kostenfunktion repräsentiert die Nutzenfunktion des Agenten.
- Für einen Pfad $(\langle x_0, \dots, x_n \rangle, \langle a_1, \dots, a_n \rangle)$ sind die *Schritt kosten* von x_{i-1} nach x_i gegeben durch eine Funktion $c(x_{i-1}, a_i)$. Oft sind die Pfadkosten gegeben durch die Summe der Schritt kosten.
Wir nehmen an, dass Schritt kosten nicht negativ sind.
- Die *Lösung* eines so definierten Problems ist gegeben durch einen Pfad der vom Initialzustand zu einem Zustand führt, der den Zieltest erfüllt.
- Die *optimale Lösung* ist die Lösung mit den geringsten Pfadkosten.

Suchprobleme: Formales Modell

Ein Suchproblem ist ein Tupel (S, A, G, σ, c) , wobei

- $S = \{s_1, s_2, \dots\}$ ist der Zustandsraum (nicht notwendigerweise endlich)
- $A = \{a_1, a_2, \dots\}$ mit $a : S \rightarrow S$ ist die Aktionsmenge
- $G \subseteq S$ sind die Zielzustände
- $\sigma \in S$ ist der Initialzustand
- $c : S \times A \rightarrow \mathbb{R}$ ist die Kostenfunktion¹

Eine Lösung eines Suchproblems ist eine Sequenz von Aktionen $\langle a_{(1)}, \dots, a_{(n)} \rangle$, sodass $(a_{(1)} \circ \dots \circ a_{(n)})(\sigma) \in G$

¹Später schauen wir uns nichtdeterministische Aktionen an, dann ist es sinnvoll, die Kostenfunktion als $c : S \times A \times S \rightarrow \mathbb{R}$ zu definieren.

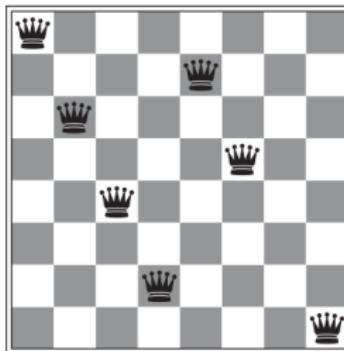
Wahl des Zustandsraums

- Die Wahl eines guten Zustandsraums ist *entscheidend* für eine effiziente Problemlösung
- Der Zustandsraum sollte von allen unwichtigen Eigenschaften der Welt *abstrahieren*.
- D.h., ein Zustand $x \in X$ repräsentiert viele möglichen Zustände der realen Welt.

Beispiel: *In(Arad)*

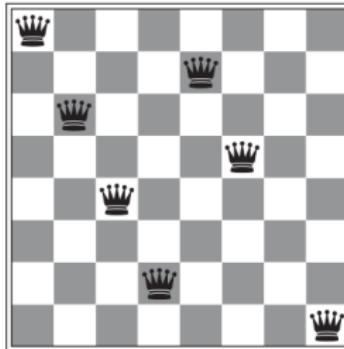
- Es könnte in Arad regnen oder die Sonne scheinen.
- Das Auto könnte rot oder grün sein.
- Der Fahrer trägt einen Hut oder nicht ...
- Auch die Aktionen müssen von unwichtigen Eigenschaften abstrahieren (wie schnell wird eine Strecke gefahren? An welchen Parkplätzen angehalten? ...)
- Es sollte weniger abstrakte Aktionen als reale Aktionen geben.
- Wirksames Beispiel: 8-Damen-Problem

8-Damen-Problem



- Zustände? Jede Anordnung von 0 bis 8 Damen auf dem Brett
- Initialzustand? Leeres Brett
- Aktionen? Setze Dame auf leeres Feld
- Nachfolgerfunktion? Gebe Brett mit zusätzlicher Dame zurück
- Zieltest? 8 Damen, keine bedroht
- $64!/56! = 1.8 * 10^{14}$ mögliche Sequenzen

8-Damen-Problem



- Zustände? Anordnung von $0 \leq n \leq 8$ Damen, eine pro Spalte in den linken n Spalten, sodass sich keine bedroht
- Initialzustand? Leeres Brett
- Aktionen? Setze Dame auf nächste leere Spalte, sodass sie nicht bedroht ist
- Nachfolgerfunktion? Gebe Brett mit zusätzlicher Dame zurück
- Zieltest? 8 Damen, keine bedroht
- 2057 mögliche Sequenzen

Suchstrategien: Exploration

- Zustände und Aktionen bilden einen gerichteten Graphen (Knoten = Zustände, Kanten = Aktionen)
- Suchstrategien versuchen, in diesem Graphen Pfade von einem Zustand (Initialzustand) zu einem anderen Zustand zu finden, der das Ziel erfüllt.
- Im Allgemeinen sind die Knotenmenge S (i) beim Start nicht bekannt und/oder (b) so groß, dass sie nicht sinnvoll gehandhabt werden kann
- Deshalb: Explorative Verfahren
 - Anfänglich nur A und σ bekannt
 - Erzeuge systematisch durch Anwendung der Aktionen neue Zustände
 - Bis Ziel erreicht ist (“forward chaining”)
 - Auch denkbar: “backward chaining”: Exploriere von einem Zielknoten, bis Startknoten gefunden

Suchstrategien: Informiertheit

Sei eine Menge von bisher entdeckten Knoten, D , gegeben – d.h., eine Menge von Zuständen, die von aus erreichbar sind. Aber D enthalte noch keinen Zielzustand. Wie geht man nun weiter vor?

- Wähle einen Zustand $s \in D$ und wähle Aktion $a \in A$
- Erzeuge Folgezustand $s' = a(s)$
Das erzeugen aller möglichen Folgezustände von s heißt auch "Expandieren von s "
- Prüfe, ob $s' \in G$. Falls ja: Ende
- Sonst: $D \leftarrow D \cup \{s'\}$, und alles von vorn

Wie wählt man s und a so, dass man sich dem Ziel möglichst schnell nähert?

- **Uninformierte** Suchalgorithmen: Man hat keine Ahnung über den Abstand von s zum Ziel.
- **Informierte** Suchalgorithmen: Man hat eine (ungefähre) Vorstellung über den Abstand

Suchstrategien: Exploration

Wie sehen eigentlich die Graphen aus, die exploriert werden müssen?

Mögliche Formen:

- Endliche Bäume: es gibt einen eindeutigen Pfad zu jedem Knoten.
Der erste Weg ist der einzige und beste. In endlicher Zeit kann der Graph vollständig durchsucht werden.
- Unendliche Bäume: Der Graph kann nicht mehr in endlicher Zeit durchsucht werden. Eine falsche Wahl kann die Tiefensuche in eine unendliche Sackgasse führen.
- Gerichtete azyklische endliche Graphen: Es gibt mehrere Wege zu einem Knoten, aber der Pfadbaum ist endlich.
(Tiefensuche im Pfadbaum terminiert, nicht notwendigerweise mit dem besten Pfad)
- Gerichtete zyklische endliche Graphen: Es gibt mehrere Wege zu einem Knoten und der Pfadbaum ist unendlich.
(Tiefensuche im Pfadbaum terminiert nicht mehr sicher)

Uninformierte Suchalgorithmen

Sie haben im Studium bisher drei Algorithmen kennengelernt, die explorativ arbeiten:

- Tiefensuche
- Breitensuche
- Dijkstra's Algorithmus

Keines dieser Verfahren ist ein informiertes Verfahren – keines verwendet Wissen über den Abstand eines gegebenen Zustands zum Zielzustand.

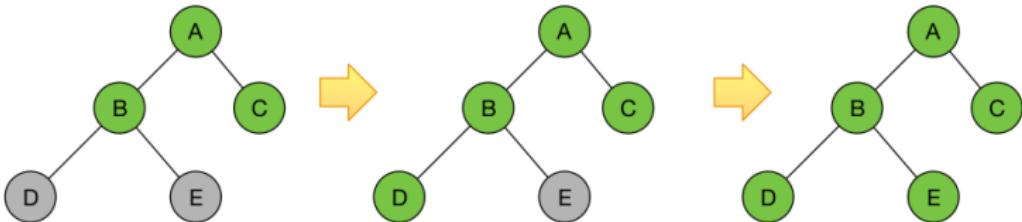
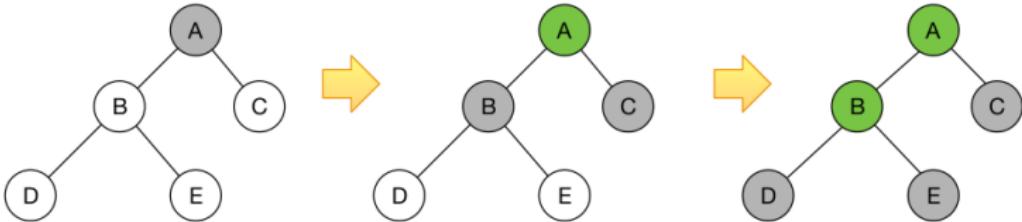
Der Unterschied zwischen den drei Verfahren liegt in der Verwaltung der neu entdeckten Knoten:

- Tiefensuche: Neu entdeckte Knoten werden in einem Stack (LIFO) gesammelt
- Breitensuche: Neu entdeckte Knoten werden in einer Queue (FIFO) gesammelt
- Dijkstra: Neu entdeckte Knoten werden in einem Priority-Queue gesammelt (Priorität = Kosten, mit dem der Knoten vom Start σ aus erreichbar ist)

Breitensuche

```
function BFS( $\sigma, A, G$ )
   $Q \leftarrow \text{MK-QUEUE}$ 
  ENQUEUE( $Q, (\sigma, \langle \rangle, 0)$ )      ▷ Tripel: Zustand, Aktionsfolge, Kosten
   $D \leftarrow \{\}$                       ▷ Bereits entdeckte Knoten
  while  $\neg \text{EMPTY}(Q)$  do
     $(s, al, c) \leftarrow \text{DEQUEUE}(Q)$       ▷ Neuer expandierter Knoten
    if  $s \in G$  then
      return  $(s, al, c)$ 
    else if  $s \notin D$  then
       $D \leftarrow D \cup \{s\}$                   ▷ Als entdeckt markieren
      for each  $(a, c') \in A$  do
        ENQUEUE( $Q, (a(s), al \oplus \langle a \rangle, c + c')$ )
  return FAIL
```

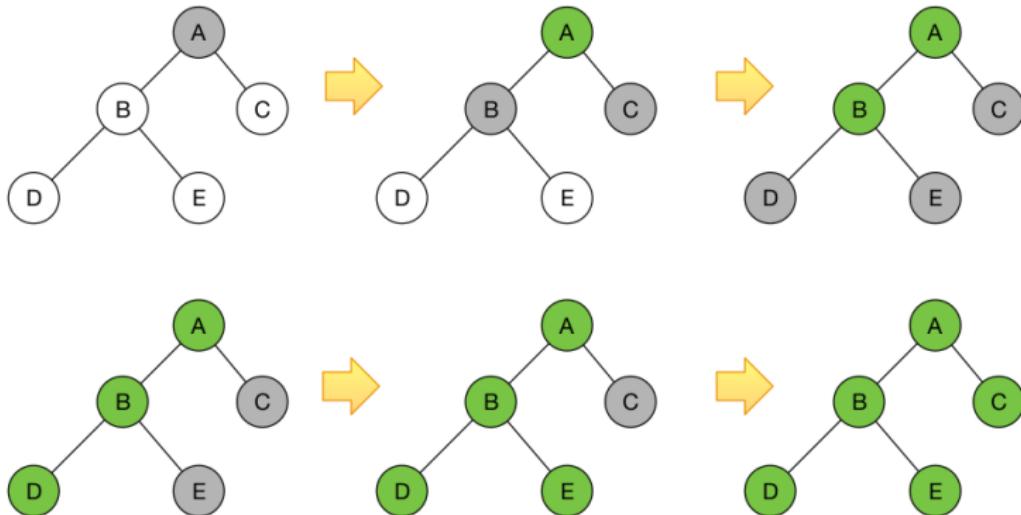
Breitensuche



Tiefensuche

```
function DFS( $\sigma, A, G$ )
   $S \leftarrow \text{MK-STACK}$ 
  PUSH( $S, (\sigma, \langle \rangle, 0)$ )           ▷ Tripel: Zustand, Aktionsfolge, Kosten
   $D \leftarrow \{\}$                          ▷ Bereits entdeckte Knoten
  while  $\neg \text{EMPTY}(S)$  do
     $(s, al, c) \leftarrow \text{POP}(S)$        ▷ Neuer expandierter Knoten
    if  $s \in G$  then
      return  $(s, al, c)$ 
    else if  $s \notin D$  then
       $D \leftarrow D \cup \{s\}$                  ▷ Als entdeckt markieren
      for each  $(a, c') \in A$  do
        PUSH( $S, (a(s), al \oplus \langle a \rangle, c + c')$ )
    return FAIL
```

Wiederholung: Tiefensuche



Dijkstra

```
function DIJKSTRA( $\sigma, A, G$ )
   $Q \leftarrow \text{MK-PRIORITY-QUEUE}$ 
  ENQUEUE( $Q, (\sigma, \langle \rangle, 0)$ )       $\triangleright$  Tripel: Zustand, Aktionsfolge, Kosten
   $D \leftarrow \{\}$                        $\triangleright$  Bereits entdeckte Knoten
  while  $\neg \text{EMPTY}(S)$  do
     $(s, al, c) \leftarrow \text{DEQUEUE-MIN}(Q)$        $\triangleright$  Neuer expandierter Knoten
    if  $s \in G$  then
      return  $(s, al, c)$ 
    else if  $s \notin D$  then
       $D \leftarrow D \cup \{s\}$                        $\triangleright$  Als entdeckt markieren
      for each  $(a, c') \in A$  do
        ENQUEUE( $Q, (a(s), al \oplus \langle a \rangle, c + c')$ )
  return FAIL
```

Probleme

- Unendlicher Graph: Tiefensuche führt nie zum Ziel (“unendliche falsche Abzweigung”)
- Breitensuche liefert zwar immer ein (bei konstanten Aktionskosten optimales) Ergebnis, aber zu sehr hohen Speicherkosten
 - Wenn jeder Zustand b Nachfolger hat, und der Pfad zum Ziel n Schritte lang ist, benötigt Breitensuche Speicherplatz der Größe $\mathcal{O}(b^{n+1})$
 - Dijkstra benötigt $\mathcal{O}(b^{C^*/\epsilon})$ Speicher (C^* : Kosten der optimalen Lösung, ϵ : Minimale Kosten einer Aktion)
 - Testen, ob ein Zustand bereits erreicht wurde, kann bei vielen zu speichernden Zuständen sehr aufwendig werden

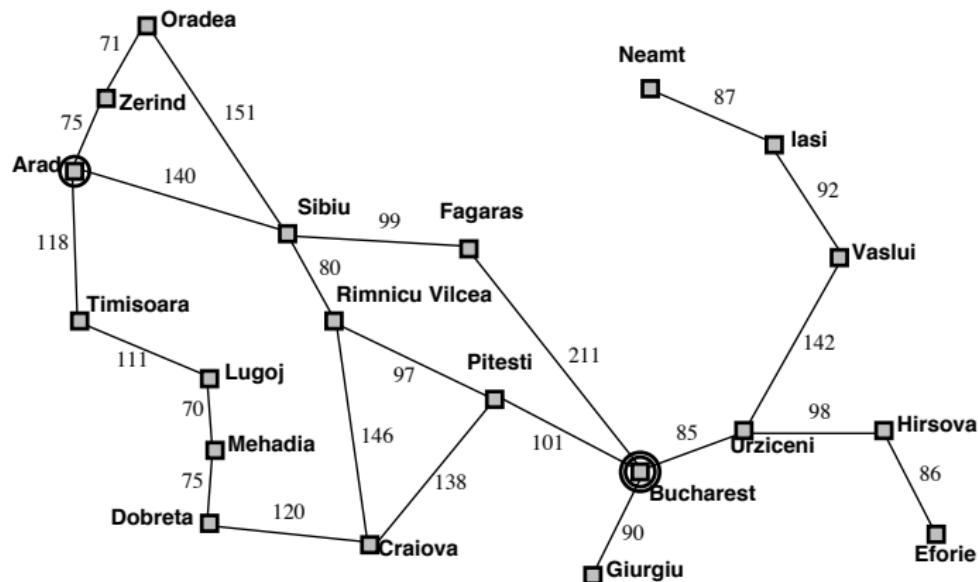
Suchstrategien

Tiefenbegrenzte Suche

- Falls man eine gute Schätzung für die maximale Anzahl von Lösungsschritten eines Suchproblems hat, lässt sich eine *tiefenbegrenzte Suche* nutzen.
Damit lässt sich das Problem der Tiefensuche, manchmal in unendlichen Ästen zu verschwinden, wirksam lösen.
- Falls sich jedes Problem in einem Suchraum mit maximal d Schritten lösen lässt (und es keine kleinere Zahl mit dieser Eigenschaft gibt), heißt d der *Durchmesser* des Suchraums.
- Der Durchmesser des Suchraums ist damit ein guter Wert für die Tiefengrenze einer tiefenbegrenzten Tiefensuche.
- Für hinreichend einfache Probleme lässt sich der Durchmesser eines Suchbaums als längster Pfad des Spannbaums berechnen.

Suchstrategien

Tiefenbegrenzte Suche



Hier lässt sich beispielsweise jede Stadt von jeder anderen Stadt aus in neun Schritten erreichen. Damit ist der Durchmesser dieses Problems 9.

Tiefenbegrenzte Tiefensuche

```
function LIMITED-DFS( $\sigma, A, G, d$ )
     $r \leftarrow \text{fail}$ 
     $S \leftarrow \text{MK-STACK}$ 
     $\text{POP}(S, (\sigma, \langle \rangle, 0, 0))$             $\triangleright$  Zustand, Aktionsfolge, Kosten, Tiefe
     $D \leftarrow \{\}$                             $\triangleright$  Bereits entdeckte Knoten
    while  $\neg \text{EMPTY}(S)$  do
         $(s, al, c, d') \leftarrow \text{POP}(S)$             $\triangleright$  Neuer expandierter Knoten
        if  $s \in G$  then
            return  $(s, al, c, d')$ 
        else if  $d' > d$  then  $r \leftarrow \text{cutoff}$ 
        else if  $s \notin D$  then
             $D \leftarrow D \cup \{s\}$                        $\triangleright$  Als entdeckt markieren
            for each  $(a, c') \in A$  do
                ENQUEUE( $Q, (a(s), al \oplus \langle a \rangle, c + c', d' + 1)$ )
    return FAIL

function ITERATIVE-DEEPENING-DFS( $\sigma, A, G$ )
    for  $d = 0, \dots, \infty$  do
         $r \leftarrow \text{LIMITED-DFS}(\sigma, A, G, d)$ 
        if  $r \neq \text{cutoff}$  then return  $r$ 
```

Tiefenbegrenzte Tiefensuche

- Vollständig und, bei konstanten Aktionskosten, optimal
- Wenn Lösung in Tiefe n gefunden wird: Speicherplatz $\mathcal{O}(bn)$
- Obwohl Zustände mehrfach besucht werden, ist die Zeitkomplexität nicht größer als bei Breitensuche
- Man kann bei der Breitensuche, beim Iterative Deepening und beim Dijkstra-Verfahren darauf verzichten, zu prüfen ob ein Knoten bereits besucht wurde
 - Alle drei Verfahren liefern den optimalen (kürzesten Pfad) – nur für die Tiefensuche muss das “Versacken” in Zyklen explizit ausgeschlossen werden.
 - Der Verzicht auf die Überprüfung kann zwar die mehrfache Expandierung eines Knotens bedeuten, aber andererseits erheblich Speicher und Rechenaufwand für die Verwaltung der “Entdeckungsliste” sparen.

Backtracking

- Betrachten wir folgende Zeilen der Tiefensuche:

```
for each  $(a, c') \in A$  do
    PUSH( $(S, (a(s), al \oplus \langle a \rangle, c + c'))$ )
```
- Müssen wir wirklich alle Nachfolgezustände $a(s)$ erzeugen und auf den Stack packen? (Was, wenn Zustände groß/kompliziert sind, z.B. Schachbretter)
- Nein: Können auch einfach erste Aktion a ausführen, und uns merken, welche anderen Aktionen noch getestet werden müssen
- Wenn Aktionen a_1, \dots, a_n auf s anwendbar sind, erzeugt man zunächst $a_1(s)$, und sucht von dort aus weiter. Wenn dies nicht zum Erfolg führt, versucht man $a_2(s)$, danach $a_3(s)$ usw. bis $a_n(s)$. Wenn auch dies nicht zum Ziel führt geht man zum Vorgänger von s zurück, und versucht es von dort aus weiter. Dieses Verfahren ist auch als Backtracking bekannt.

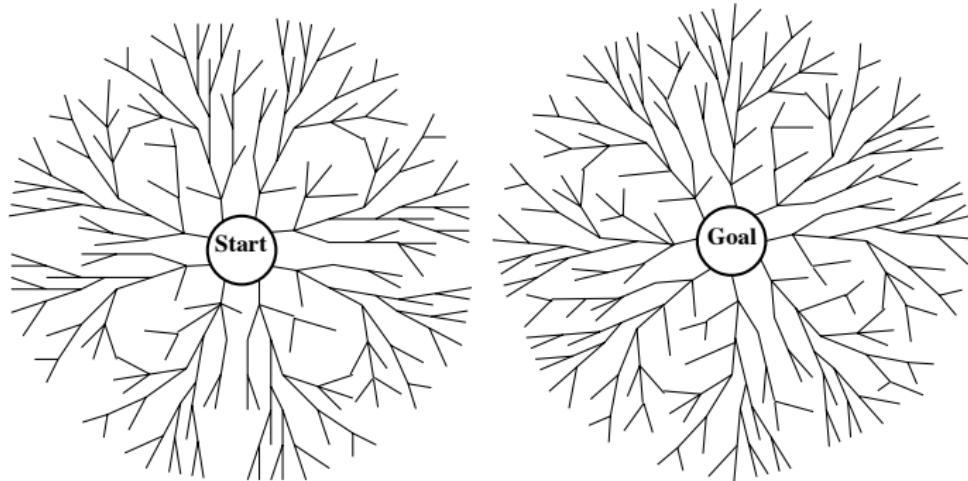
Backtracking

- Backtracking lässt sich sehr gut rekursiv implementieren:
- eine Funktion $\text{Backtrack}(s)$ ist für die Expansion eines Knotens s verantwortlich.
- $\text{Backtrack}(s)$ enthält eine Schleife, in der sequentiell alle Aktionen a_i für s getestet werden...
- und in der für jedes $a_i(s)$ die Tiefensuche durch den rekursiven Aufruf von $\text{Backtrack}(a_i(s))$ weiter fortgeführt wird.

```
function BACKTRACK( $s, A, G$ )
    if  $s \in G$  then
        return ( $s, al, c$ )
    for each  $(a, c') \in A$  do
         $(s', al, c) \leftarrow r \leftarrow \text{BACKTRACK}(a(s), A, G)$ 
        if  $r \neq \text{FAIL}$  then return  $(s', al \oplus \langle a \rangle, c + c')$ 
    return FAIL
```

Suchstrategien

Bidirektionale Suche



Sofern Vorwärts- und Rückwärtssuche symmetrisch sind, erreicht man Suchzeiten von

$$O(2b^{d/2}) = O(b^{d/2})$$

z. B. für $b = 10$, $d = 6$ statt 1111111 nur 2222 Knoten.

Suchstrategien: Bidirektionale Suche

Probleme

- Operatoren nicht immer umkehrbar oder nur sehr schwer.
(Berechnung der Vorgängerknoten)
- Man benötigt effiziente Verfahren, um zu testen, ob sich die Suchverfahren „getroffen“ haben.
- Welche Art der Suche wählt man für jede Richtung?
(Im Bild Breitensuche)
- Bei Breitensuche: Speicherung von einem der Bäume zwecks Vergleich. (Speicher $O(b^{d/2})$)
- Nur plausibel bei *sehr wenig* Zielzuständen
 - Gegenbeispiel: Zielzustand „Schachmatt“ . . .

Informierte Suche

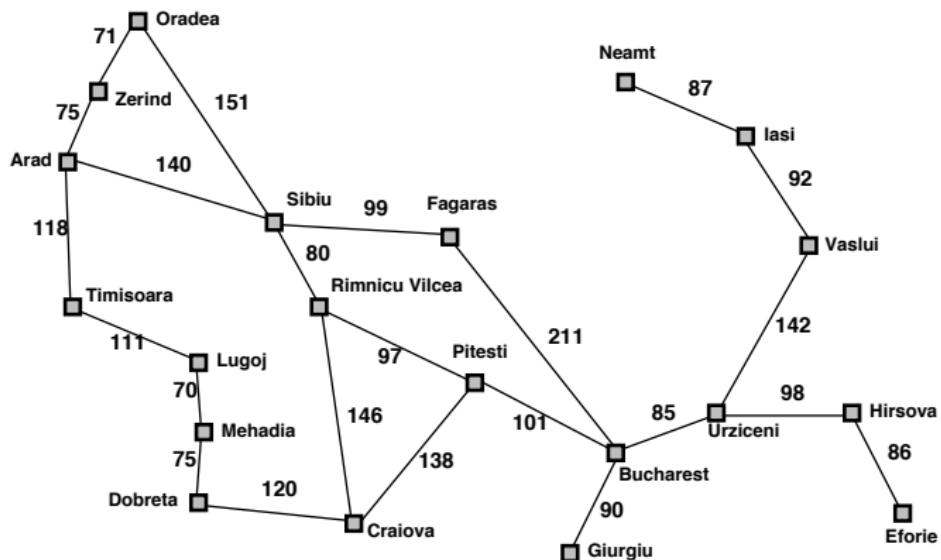
- In manchen Fällen kann man “schlauer” entscheiden, welcher Knoten als nächstes expandiert werden soll
- D.h. Knoten werden in einer Priority Queue gespeichert, wobei “aussichtsreichere” Knoten eine höhere Priorität haben
- Konkret: Sei $h : S \rightarrow \mathbb{R}$ eine *Heuristik*, die eine Abschätzung für die Distanz von s zum Ziel angibt
- Wir schauen uns zwei Fälle an:
 - greedy Suche
 - A^* Suche

Greedy Suche

Einfacher Fall: Priorität hängt ausschließlich von Heuristik ab

```
function GREEDY-SEARCH( $\sigma, A, G$ )
   $Q \leftarrow \text{MK-PRIORITY-QUEUE}$ 
  ENQUEUE( $Q, (\sigma, \langle \rangle, h(\sigma))$ )
   $D \leftarrow \{\}$                                  $\triangleright$  Bereits entdeckte Knoten
  while  $\neg \text{EMPTY}(S)$  do
     $(s, al, c) \leftarrow \text{DEQUEUE-MIN}(Q)$   $\triangleright$  Neuer expandierter Knoten
    if  $s \in G$  then
      return  $(s, al, c)$ 
    else if  $s \notin D$  then
       $D \leftarrow D \cup \{s\}$                        $\triangleright$  Als entdeckt markieren
      for each  $(a, c') \in A$  do
        ENQUEUE( $Q, (a(s), al \oplus \langle a \rangle, h(s))$ )
  return FAIL
```

Beispiel: Rumänien



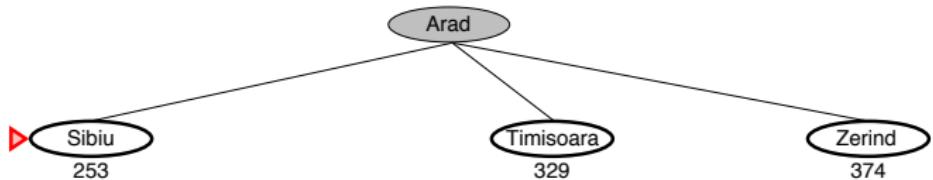
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Kosten: Distanzen zwischen Städten
- Heuristik: Luftlinien-Distanz nach Bukarest

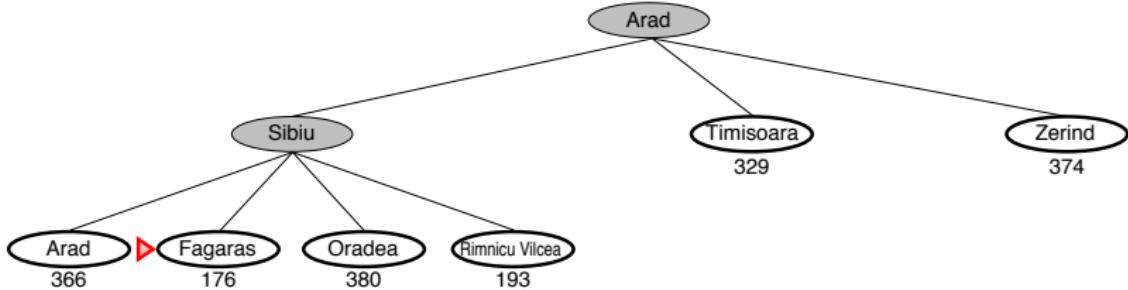
Beispiel: Greedy Suche



Beispiel: Greedy Suche



Beispiel: Greedy Suche



Eigenschaften der Greedy Suche

- Kann in Schleifen hängenbleiben, wenn naiv implementiert: Muss sich merken, welche Knoten schon besucht wurden
- Zeitkomplexität: $\mathcal{O}(b^m)$ – aber gute Heuristik kann das dramatisch verbessern (m = maximale Tiefe des Suchbaums)
- Platzkomplexität: $\mathcal{O}(b^m)$ – muss alle Knoten im Speicher halten
- Nicht optimal (findet nicht unbedingt den kürzesten Weg)

A^* Suche

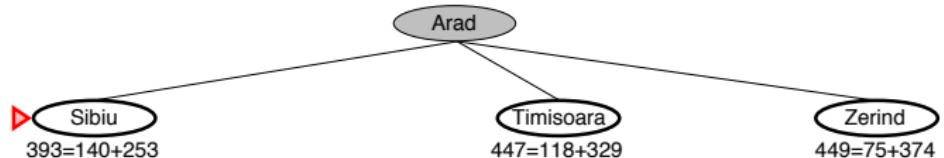
- Wie können wir Greedy-Suche verbessern?
- Idee: Vermeide Expansion von Päden, bei denen der bisher zurückgelegte Weg sehr teuer ist
- Funktion $f(s) = g(s) + h(s)$, wobei
 - $g(s)$: Bisherige Kosten des Pfades zu s
 - $h(s)$: Heuristik, Abschätzung der Kosten von s zum Ziel
 - $f(s)$: Abschätzung der gesamten Kosten vom Start zum Ziel über s
- Heuristik h heißt *zulässig*, wenn sie die tatsächlichen Kosten von s zum Ziel nie überschätzt
- Satz: A^* mit zulässiger Heuristik ist optimal.

A^* Suche

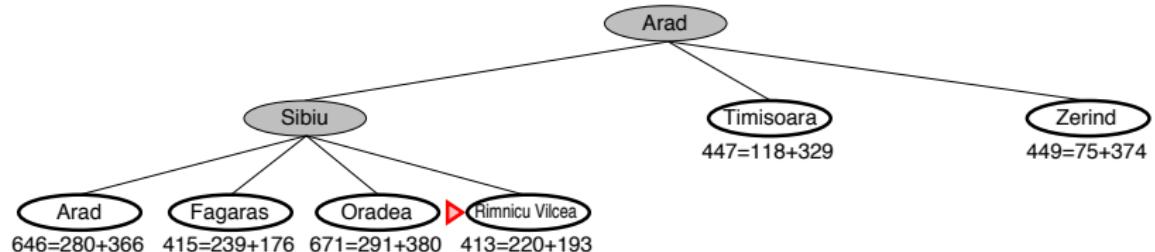
Verbessere Greedy Suche: Wähle den Knoten, für den die Abschätzung für den Weg insgesamt (Kosten bis zum Knoten + Heuristik vom Knoten bis zum Ziel) am geringsten ist

```
function  $A^*$ -SEARCH( $\sigma, A, G$ )
   $Q \leftarrow \text{Mk-Priority-Queue}$ 
  ENQUEUE( $Q, (\sigma, \langle \rangle, h(\sigma))$ )
   $D \leftarrow \{\}$                                  $\triangleright$  Bereits entdeckte Knoten
  while  $\neg \text{EMPTY}(S)$  do
     $(s, al, c) \leftarrow \text{Dequeue-Min}(Q)$   $\triangleright$  Neuer expandierter Knoten
    if  $s \in G$  then
      return  $(s, al, c)$ 
    else if  $s \notin D$  then
       $D \leftarrow D \cup \{s\}$                        $\triangleright$  Als entdeckt markieren
      for each  $(a, c') \in A$  do
        ENQUEUE( $Q, (a(s), al \oplus \langle a \rangle, c + c' + h(s))$ )
    return FAIL
```

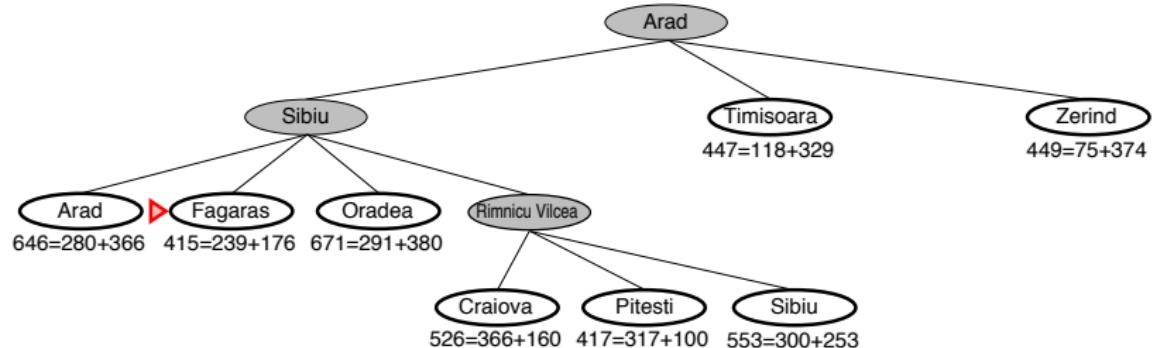
Beispiel: A^* Suche



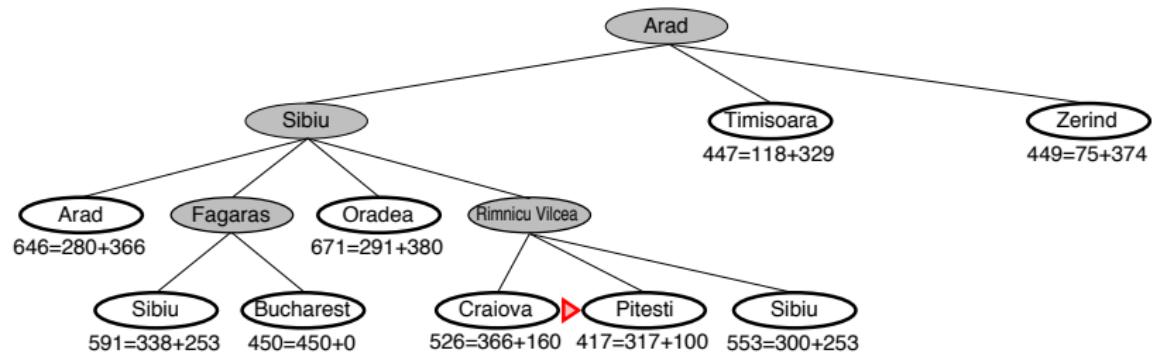
Beispiel: A^* Suche



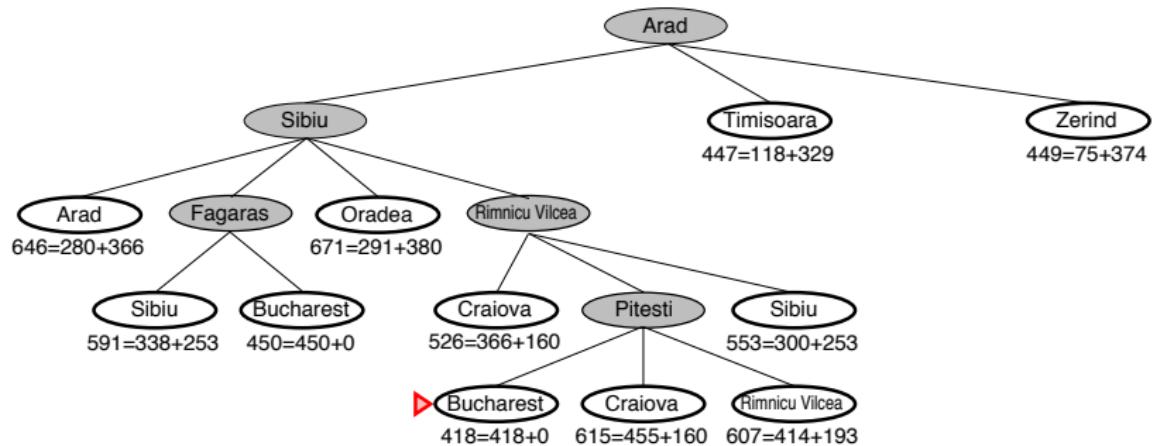
Beispiel: A^* Suche



Beispiel: A^* Suche

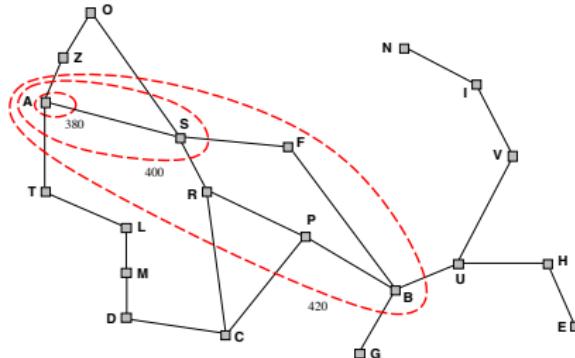


Beispiel: A* Suche



Optimalität von A^* : Beweisidee

- Ein von A^* expandierter Knoten kann nie einen geringeren f -Wert haben als ein zuvor expandierter Knoten
- D.h. A^* erweitert nach und nach “ f -Konturen”: Kontur i enthält alle Knoten mit $f = f_i$, wobei $f_i < f_{i+1}$
- Seien C^* die Kosten des optimalen Pfads
- A^* expandiert alle Knoten mit $f(s) < C^*$
- A^* kann dann einige Knoten mit $f(s) = C^*$ expandieren, anschließend den Zielknoten
- Der so gefundene Pfad ist optimal



A^* : Eigenschaften

- Optimal: Findet garantiert einen kürzesten Weg zwischen Start und Ziel
- Zeitkomplexität
 - Seien h^* die tatsächlichen Kosten zum Ziel, und sei $\epsilon = (h^* - h)/h^*$ der *relative Fehler* der Heuristik h
 - Zeitkomplexität von A^* : $\mathcal{O}(b^{\epsilon n})$
- A^* ist *optimal effizient*: Für eine gegebene Heuristik kann es keinen Algorithmus geben, der einen optimalen Pfad in weniger Schritten findet
- Platzkomplexität $\mathcal{O}(b^{\epsilon n})$: Muss wieder alle Knoten im Speicher halten
- Insgesamt: Oftmals zu aufwendig, optimale Lösung zu finden
 - Approximativen Algorithmus verwenden, oder
 - A^* , aber nicht-zulässige (aber trotzdem nützliche) Heuristik verwenden

Zusammenfassung

- Problem, das wir in diesem Kapitel betrachtet haben:
 - Gegeben: Beobachtbare, statische Umgebung, Start- und Zielzustand
 - Gesucht: Aktionsfolge, um Start- in Zielzustand zu überführen
- Solche Probleme können als *Suchproblem* formalisiert werden
- Schwierigkeit, Lösung zu finden, kann wesentlich von Problemformalisierung abhängen (Abstraktion)
- Uninformierte Suchalgorithmen: Tiefensuche, Breitensuche, Dijkstra
- Informierte Suchalgorithmen (benutzen Heuristik, die "Aussichtsreichtum" eines Knotens beschreibt): Greedy Suche, A^*

Künstliche Intelligenz

Constraint Satisfaction Problems

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Problemlösung durch Suchen

Problemlösende Agenten

- Ein *problemlösender Agent* ist ein Spezialfall des zielbasierten Agenten
- Sucht nach Aktionsfolgen, die einen wünschenswerten Zustand erreichen.
- Varianten der Suche:
 - *nichtinformierte* Suche: Keine Information über den „Abstand“ zwischen wünschenswertem Zustand und anderen Situationen.
 - *informierte* Suche: Abstandsinformation vorhanden.
- Was bedeutet hierbei „Problem“ und „Lösung“?

Constraint satisfaction problems (CSPs)

- Standard-Suchproblem: Zustand ist “Black Box” (beliebige Datenstruktur)
- CSP: Zustand ist definiert über *Variablen* V_i mit *Werten* aus *Domäne* D_i
- Zieltest ist Menge von *Constraints*(Bedingungen), die die erlaubten Kombinationen von Variablen für eine Teilmenge der Variablen einschränken
- Erlaubt, effizientere Algorithmen als generelle Suchalgorithmen zu verwenden

4-Damen als CSP

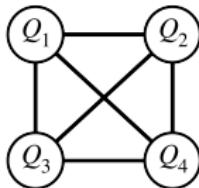
- Eine Dame pro Spalte. In welche Zeile soll jede Dame?
- Variablen: Q_1, Q_2, Q_3, Q_4
- Domänen: $D_i = \{1, 2, 3, 4\}$
- Constraints:
 - $Q_i \neq Q_j$ (nicht in gleicher Zeile)
 - $|Q_i - Q_j| \neq |i - j|$ (nicht auf gleicher Diagonale)



- Constraints können in Menge erlaubter Werte für betreffende Variablen umgewandelt werden, z.B. erlaubte Werte für (Q_1, Q_2) sind $(1, 3)$ $(1, 4)$ $(2, 4)$ $(3, 1)$ $(4, 1)$ $(4, 2)$

Constraint Graph

- Binäres CSP: Jedes Constraint beinhaltet höchstens zwei Variablen
- Constraint Graph: Knoten sind Variablen, Kanten sind Constraints

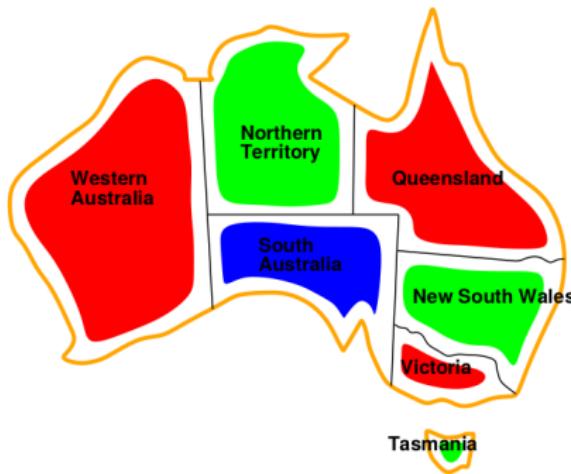


Beispiel: Kartenfärbung



- Variablen: Länder C_i
- Domänen: $\{Rot, Gruen, Blau\}$
- Constraints: $C_1 \neq C_2, C_1 \neq C_5$, etc.

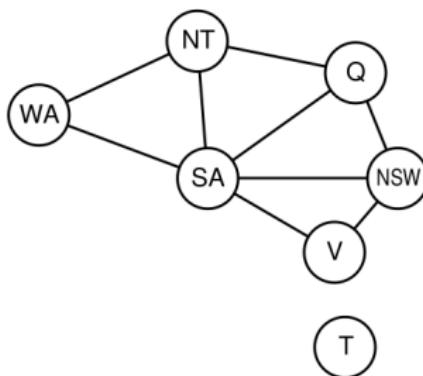
Beispiel: Kartenfärbung



Lösung: Zuweisung von Werten zu jeder Variablen, sodass alle Constraints erfüllt sind, z.B. $WA = red, WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green$

Beispiel: Kartenfärbung

Constraint Graph: Knoten sind Variablen, Kanten sind Constraints



Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

- ◊ **Initial state:** the empty assignment, $\{ \}$
- ◊ **Successor function:** assign a value to an unassigned variable that does not conflict with current assignment.
⇒ fail if no legal assignments (not fixable!)
- ◊ **Goal test:** the current assignment is complete

- 1) This is the same for all CSPs! 😊
- 2) Every solution appears at depth n with n variables
⇒ use depth-first search
- 3) Path is irrelevant, so can also use complete-state formulation
- 4) $b = (n - \ell)d$ at depth ℓ , hence $n!d^n$ leaves!!!! 😫

Backtracking search

Variable assignments are **commutative**, i.e.,

$[WA = \text{red} \text{ then } NT = \text{green}]$ same as $[NT = \text{green} \text{ then } WA = \text{red}]$

Only need to consider assignments to a single variable at each node

$\Rightarrow b = d$ and there are d^n leaves

Depth-first search for CSPs with single-variable assignments
is called **backtracking** search

Backtracking search is the basic uninformed algorithm for CSPs

Can solve n -queens for $n \approx 25$

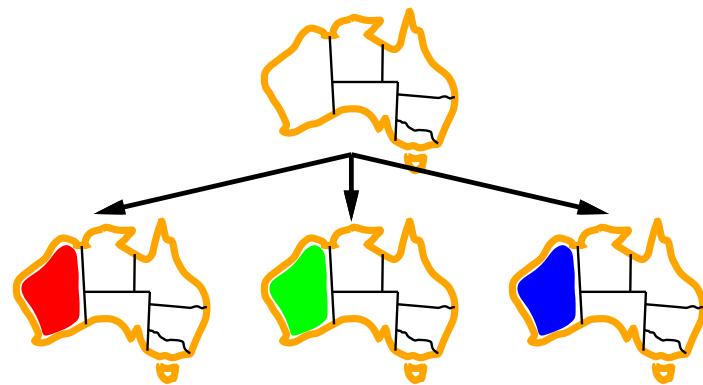
Backtracking search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add  $\{var = value\}$  to assignment
            result  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
            if result  $\neq$  failure then return result
            remove  $\{var = value\}$  from assignment
    return failure
```

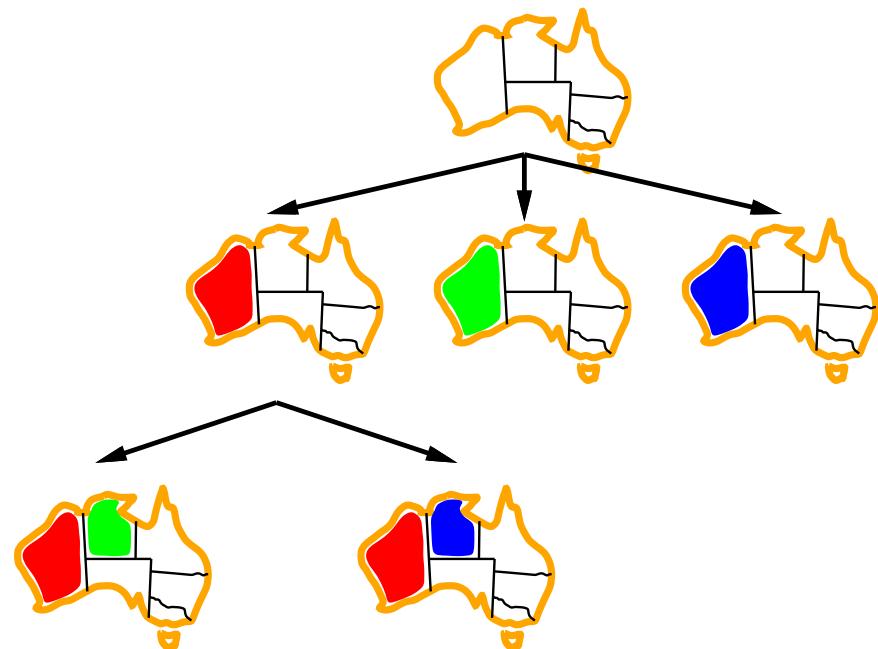
Backtracking example



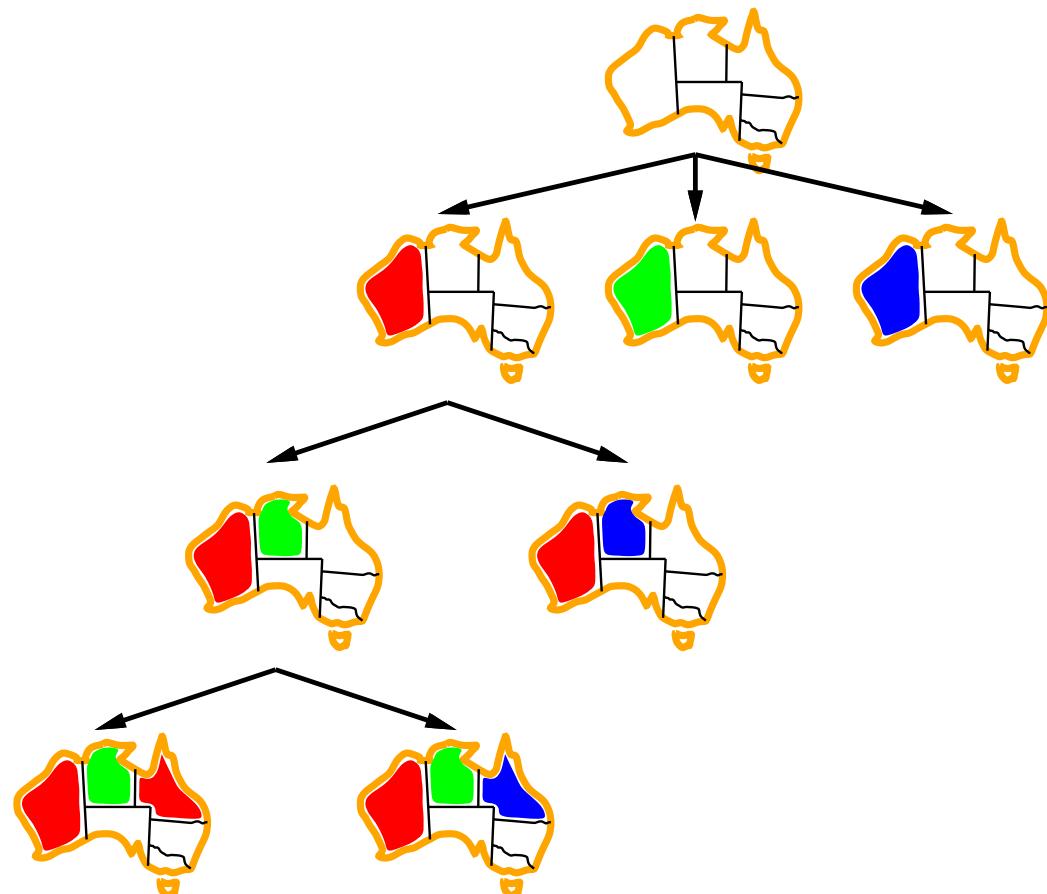
Backtracking example



Backtracking example



Backtracking example



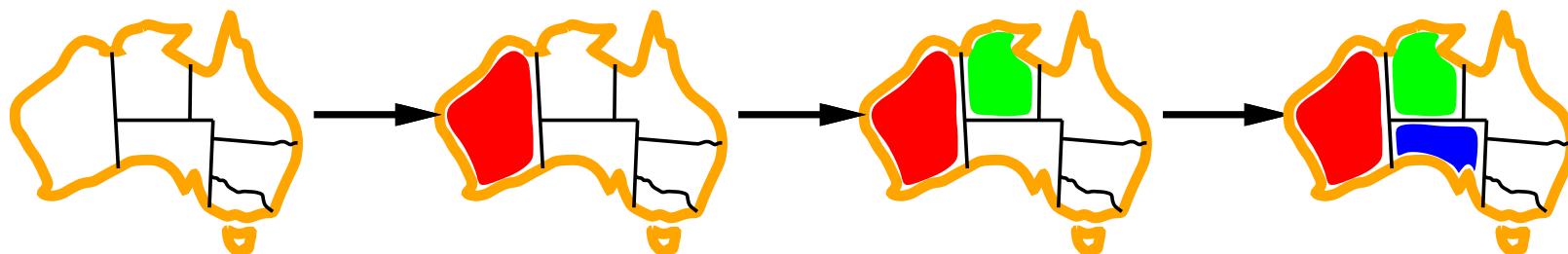
Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

Minimum remaining values

Minimum remaining values (MRV):
choose the variable with the fewest legal values

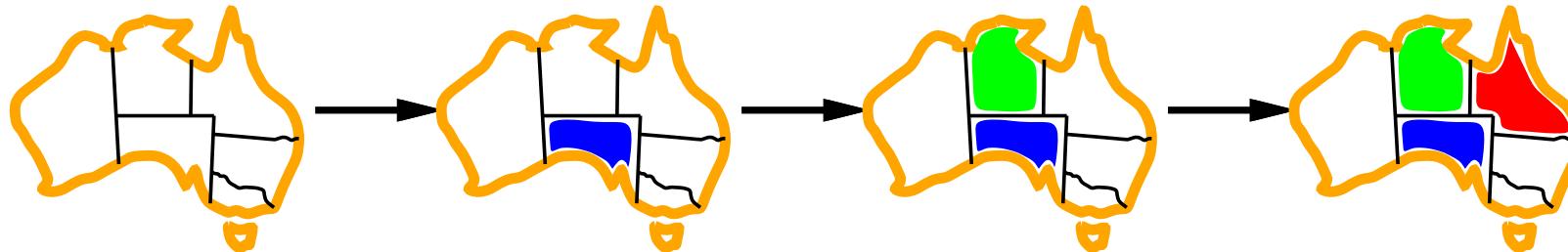


Degree heuristic

Tie-breaker among MRV variables

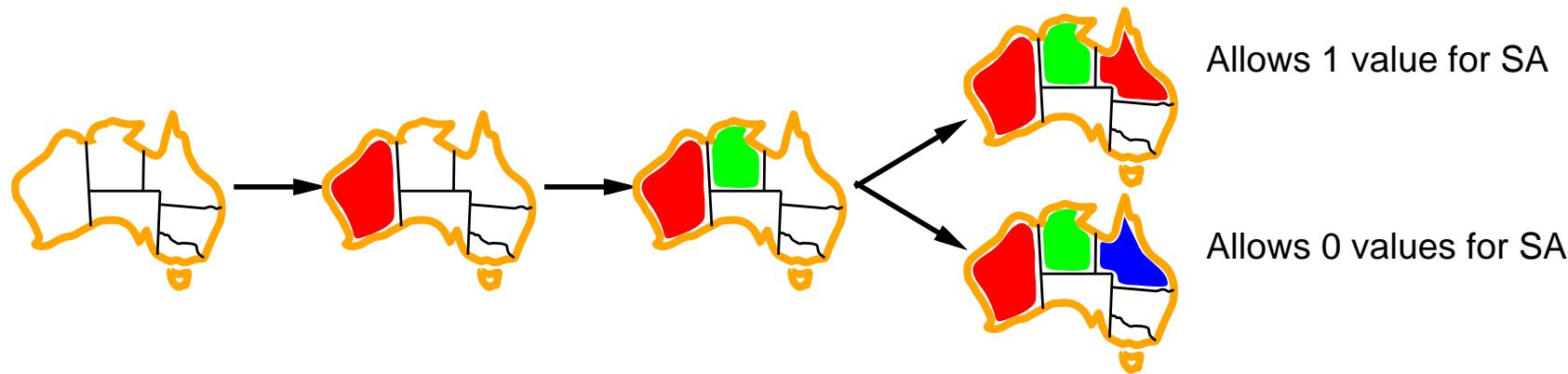
Degree heuristic:

choose the variable with the most constraints on remaining variables



Least constraining value

Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables



Combining these heuristics makes 1000 queens feasible

Forward checking

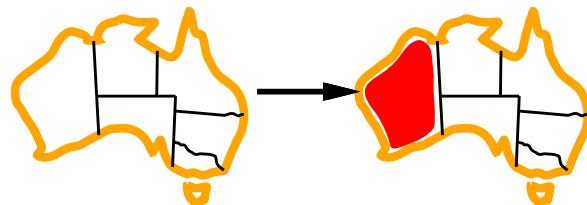
Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values



Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
Red	Green	Blue	Red	Green	Blue	Red
Red		Green	Blue	Red	Green	Blue

Forward checking

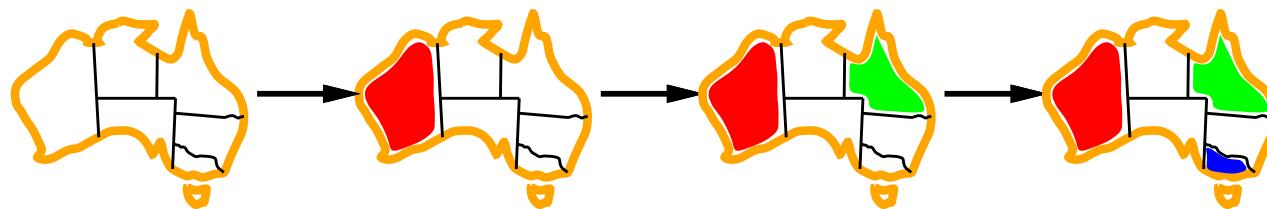
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
Red	Green	Blue	Red	Green	Blue	Red
Red		Green	Blue	Red	Green	Blue
Red		Blue	Green	Red	Green	Blue

Forward checking

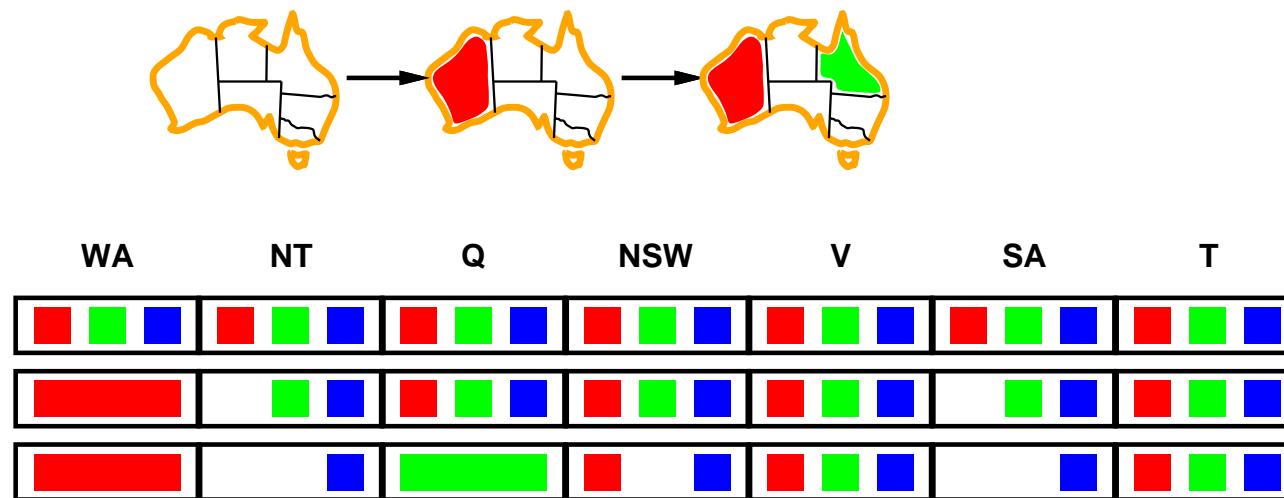
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
Red	Green	Blue	Red	Green	Blue	Red
Red		Green	Blue	Red	Green	Blue
Red		Blue	Green	Red	Blue	Red
Red		Green		Red	Blue	Red

Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



NT and *SA* cannot both be blue!

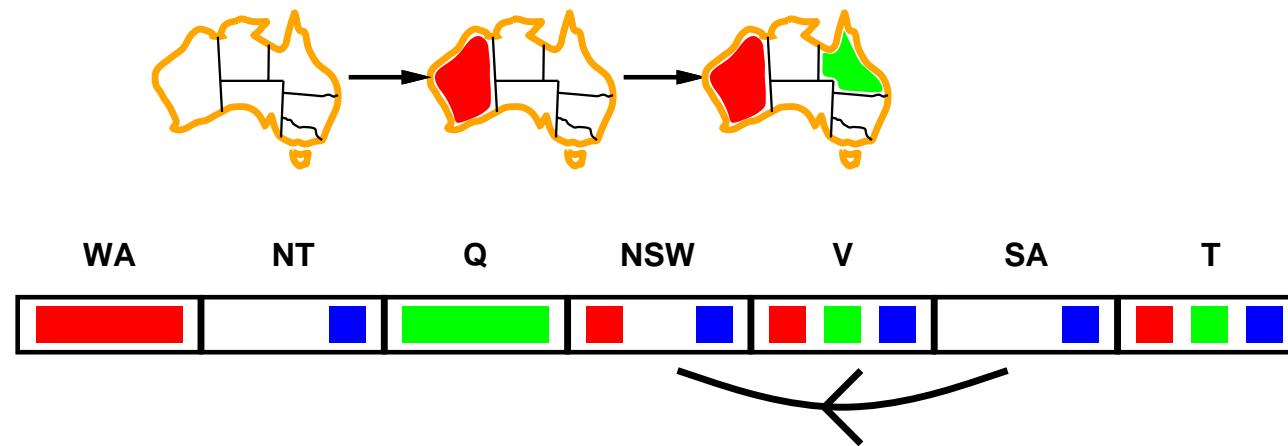
Constraint propagation repeatedly enforces constraints locally

Arc consistency

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y

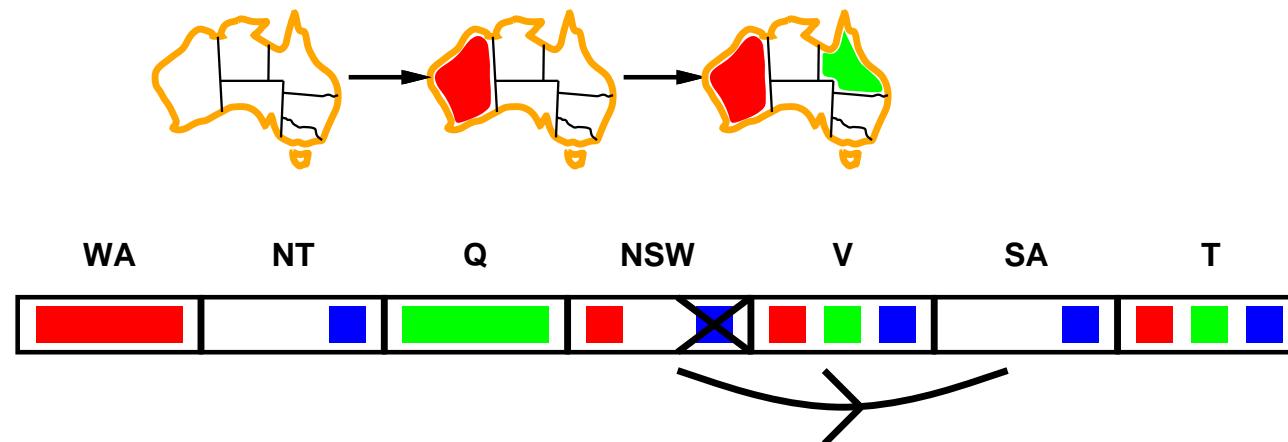


Arc consistency

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y

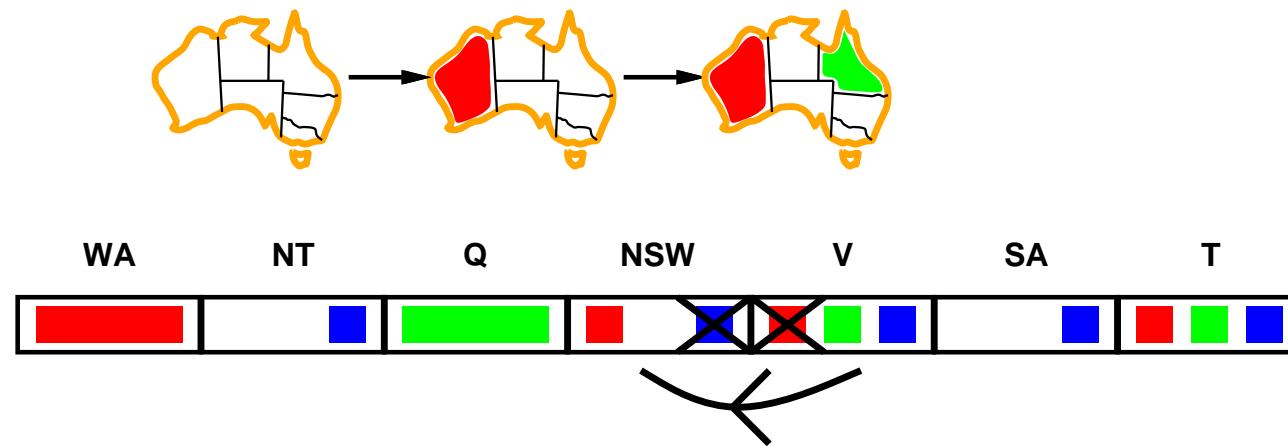


Arc consistency

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y



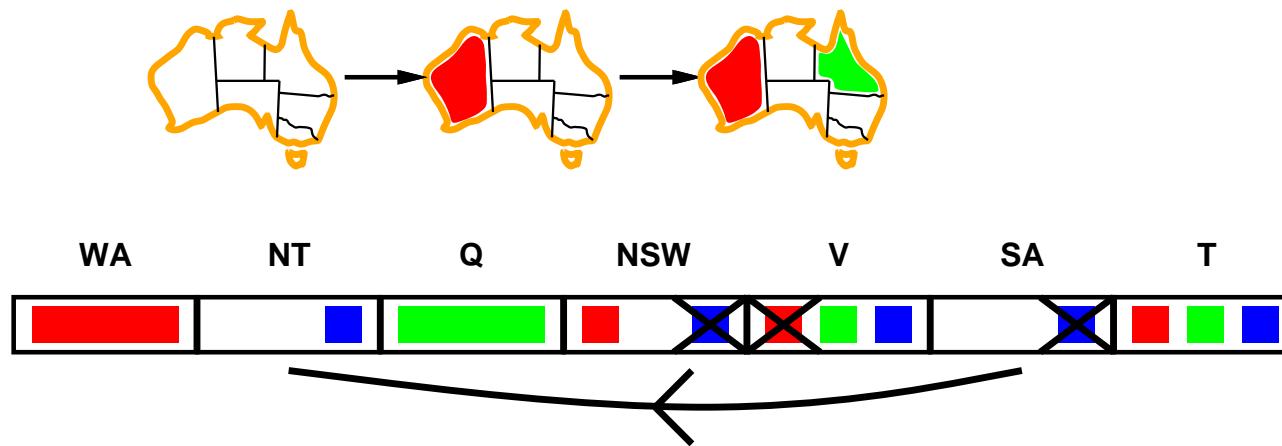
If X loses a value, neighbors of X need to be rechecked

Arc consistency

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y



If X loses a value, neighbors of X need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Arc consistency algorithm

```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$ 
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
      for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
        add  $(X_k, X_i)$  to queue



---


function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
  removed  $\leftarrow \text{false}$ 
  for each  $x$  in DOMAIN[ $X_i$ ] do
    if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
      then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow \text{true}$ 
  return removed
```

$O(n^2d^3)$, can be reduced to $O(n^2d^2)$ (but detecting **all** is NP-hard)

Summary

CSPs are a special kind of problem:

- states defined by values of a fixed set of variables

- goal test defined by **constraints** on variable values

Backtracking = depth-first search with one variable assigned per node

Variable ordering and value selection heuristics help significantly

Forward checking prevents assignments that guarantee later failure

Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies

The CSP representation allows analysis of problem structure

Tree-structured CSPs can be solved in linear time

Iterative min-conflicts is usually effective in practice

Künstliche Intelligenz

Game Playing

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

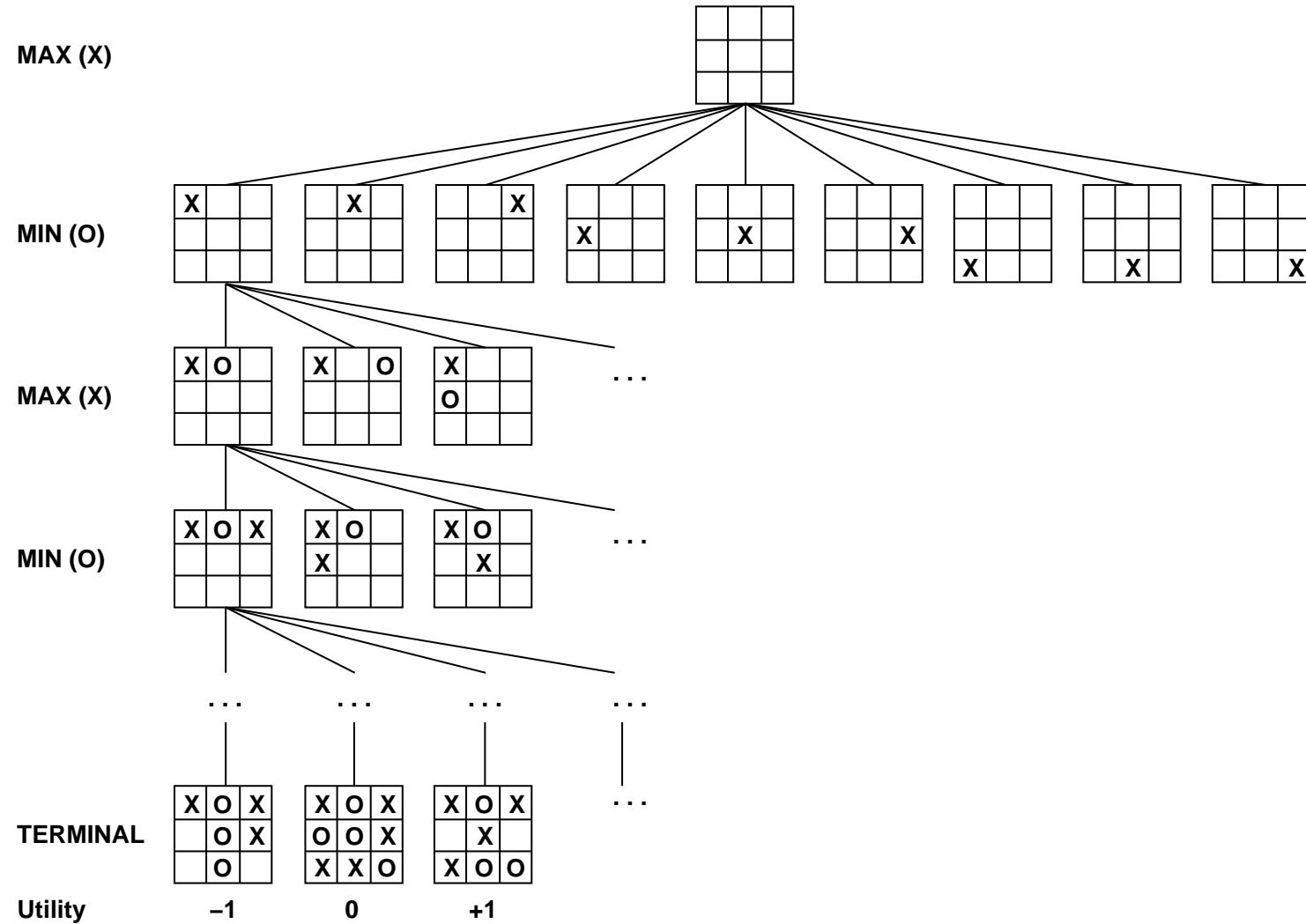
Spiele vs. Suchprobleme

- Spiele unterscheiden sich von “normalen” Suchproblemen:
 - “Unvorhersehbare” Aktionen des Gegners
 - Müssen daher für jede mögliche Aktion des Gegners einen Spielzug parat haben
 - Zeitlimits: Können nicht immer optimale Lösung berechnen

Types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Game tree (2-player, deterministic, turns)

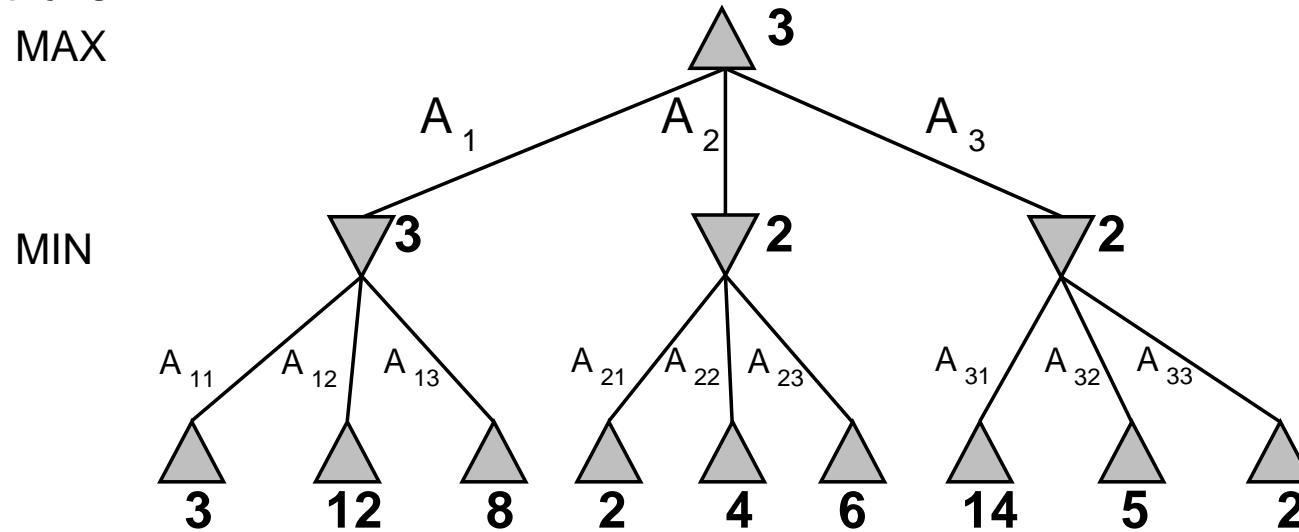


Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest **minimax value**
= best achievable payoff against best play

E.g., 2-ply game:



Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow -\infty$ 
    for a, s in SUCCESSORS(state) do v  $\leftarrow \text{MAX}(\text{MIN-VALUE}(s), v)$ 
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow \infty$ 
    for a, s in SUCCESSORS(state) do v  $\leftarrow \text{MIN}(\text{MAX-VALUE}(s), v)$ 
    return v
```

Properties of minimax

Complete??

Properties of minimax

Complete?? Only if tree is finite (chess has specific rules for this).

NB a finite strategy can exist even in an infinite tree!

Optimal??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

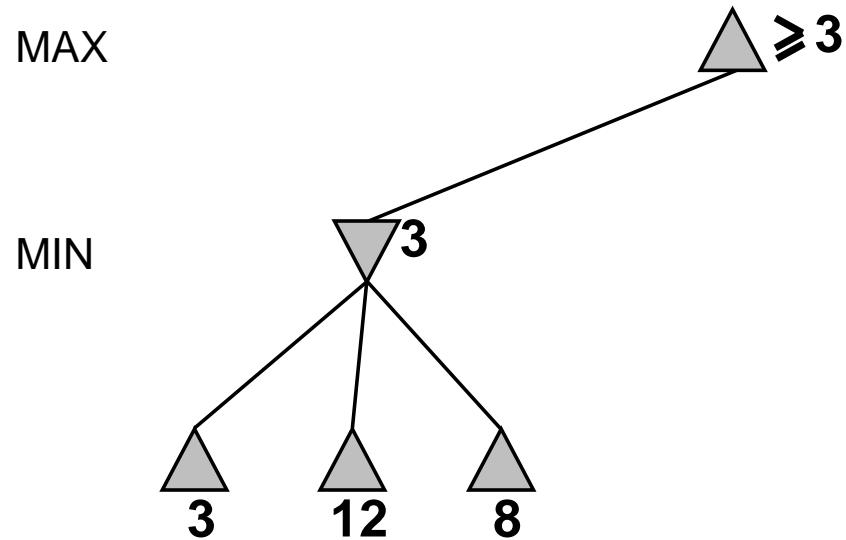
Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

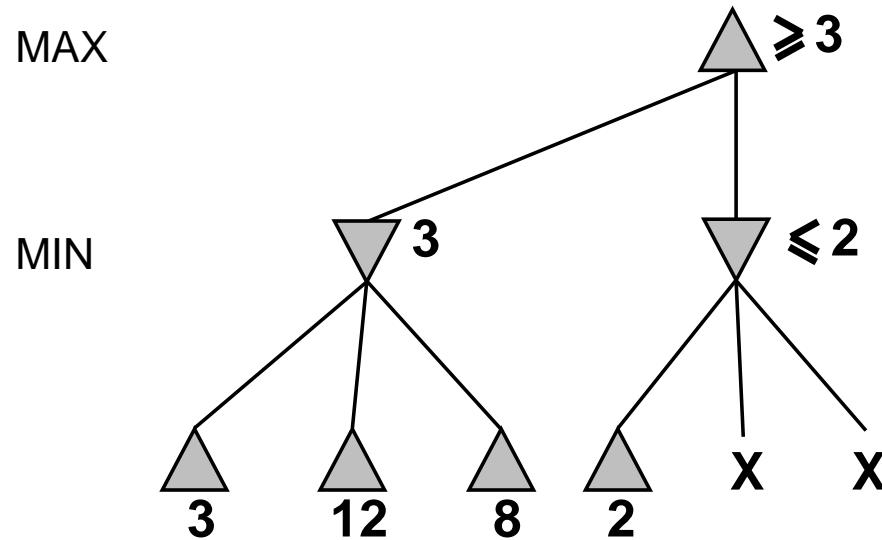
For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
⇒ exact solution completely infeasible

But do we need to explore every path?

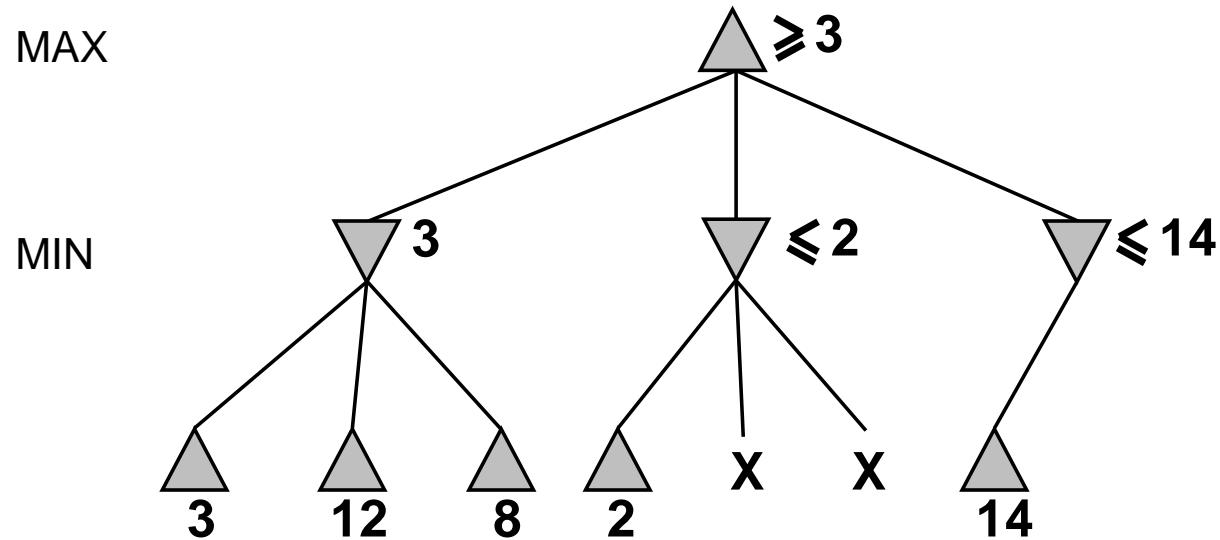
$\alpha-\beta$ pruning example



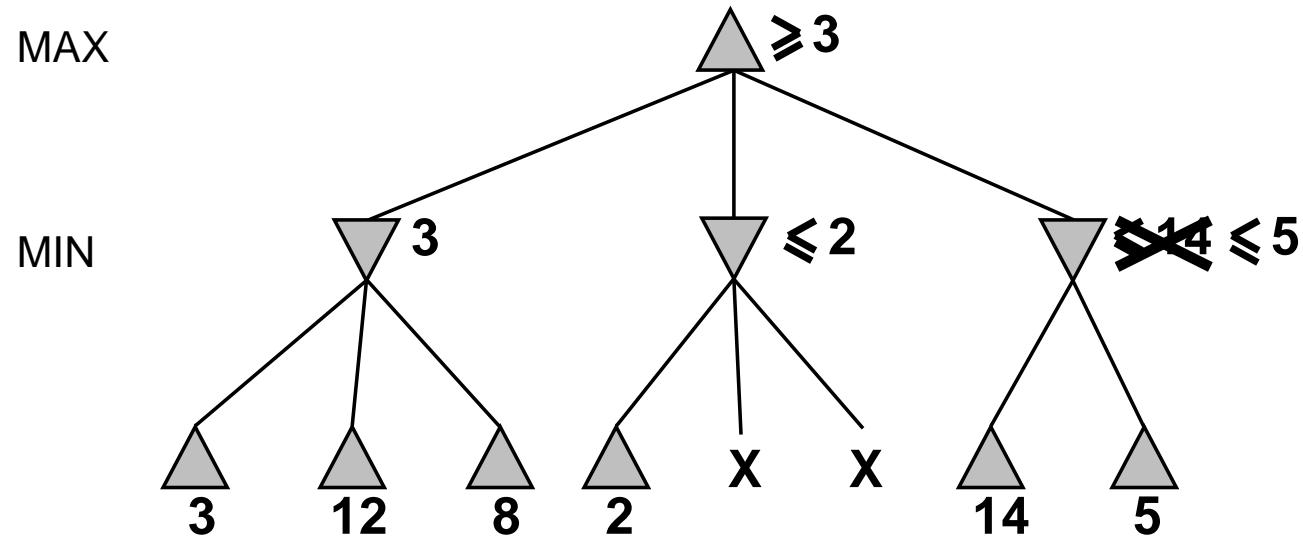
$\alpha-\beta$ pruning example



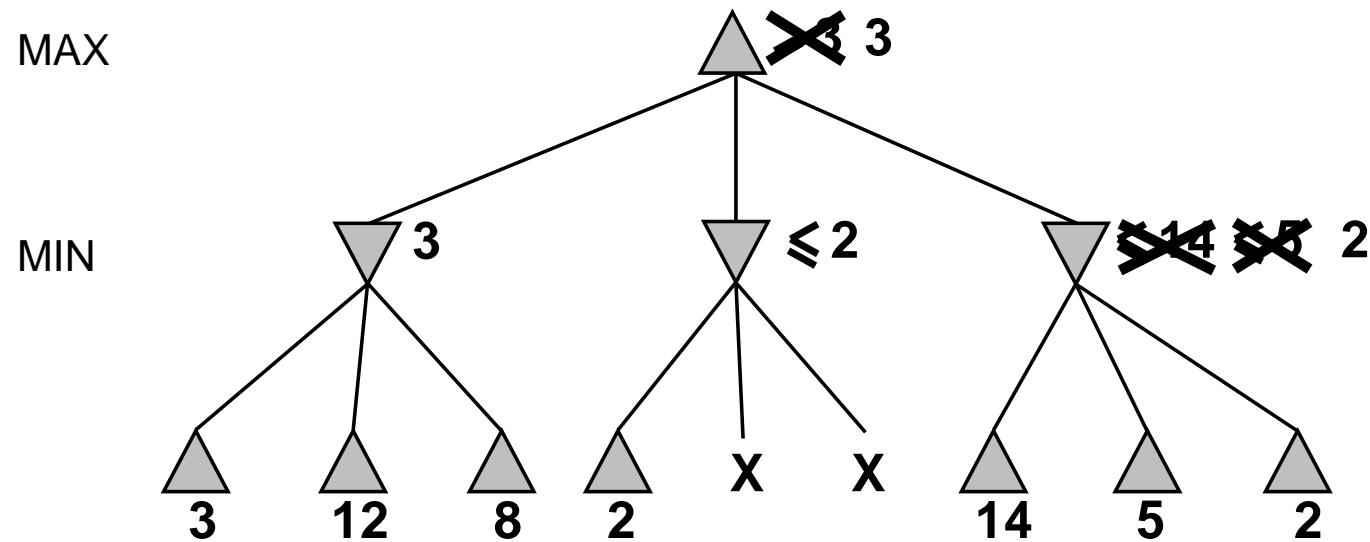
$\alpha-\beta$ pruning example



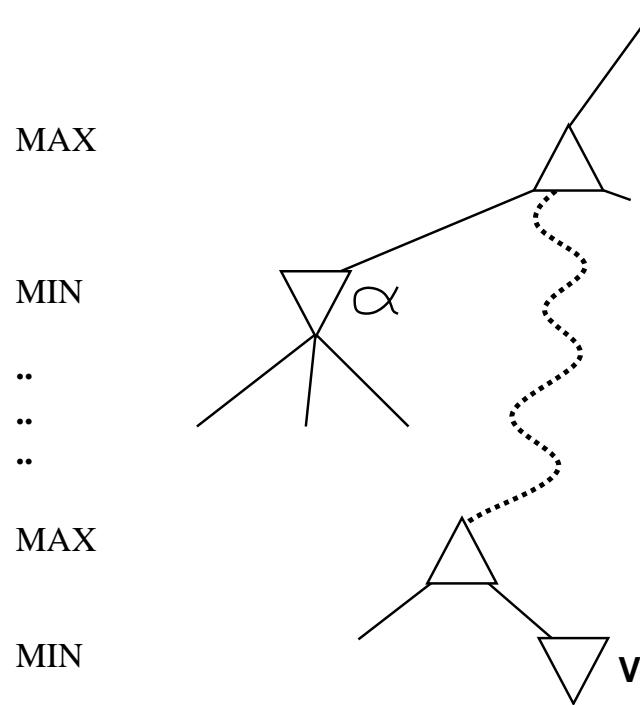
$\alpha-\beta$ pruning example



$\alpha-\beta$ pruning example



Why is it called α - β ?



α is the best value (to MAX) found so far off the current path

If V is worse than α , MAX will avoid it \Rightarrow prune that branch

Define β similarly for MIN

The α - β algorithm

```
function ALPHA-BETA-DECISION(state) returns an action
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state, α, β) returns a utility value
```

inputs: *state*, current state in game

α , the value of the best alternative for MAX along the path to *state*

β , the value of the best alternative for MIN along the path to *state*

```
if TERMINAL-TEST(state) then return UTILITY(state)
```

$v \leftarrow -\infty$

```
for a, s in SUCCESSORS(state) do
```

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\textit{s}, \alpha, \beta))$

 if $v \geq \beta$ then return *v*

$\alpha \leftarrow \text{MAX}(\alpha, v)$

```
return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
```

same as MAX-VALUE but with roles of α, β reversed

Properties of α - β

Pruning **does not** affect final result

Good move ordering improves effectiveness of pruning

With “perfect ordering,” time complexity = $O(b^{m/2})$
⇒ **doubles** solvable depth

A simple example of the value of reasoning about which computations are relevant (a form of **metareasoning**)

Unfortunately, 35^{50} is still impossible!

Resource limits

Standard approach:

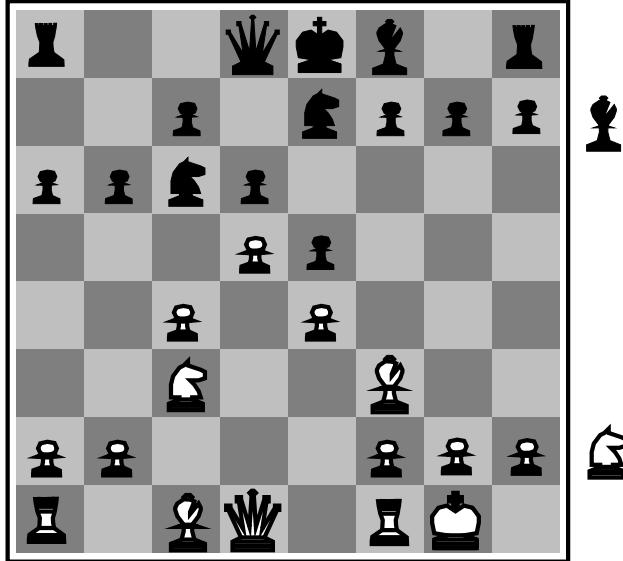
- Use CUTOFF-TEST instead of TERMINAL-TEST
 - e.g., depth limit (perhaps add quiescence search)
- Use EVAL instead of UTILITY
 - i.e., evaluation function that estimates desirability of position

Suppose we have 100 seconds, explore 10^4 nodes/second

$$\Rightarrow 10^6 \text{ nodes per move} \approx 35^{8/2}$$

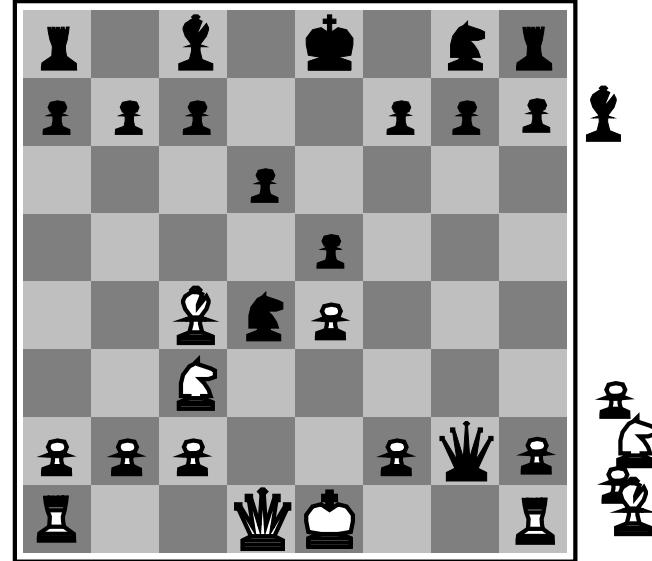
$\Rightarrow \alpha-\beta$ reaches depth 8 \Rightarrow pretty good chess program

Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically **linear weighted sum of features**

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

e.g., $w_1 = 9$ with

$f_1(s) = (\text{number of white queens}) - (\text{number of black queens})$, etc.

Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- ◊ perfection is unattainable \Rightarrow must approximate
- ◊ good idea to think about what to think about
- ◊ uncertainty constrains the assignment of values to states
- ◊ optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design

Künstliche Intelligenz

Wahrscheinlichkeitsrechnung

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Warum Wahrscheinlichkeiten?

- Sei A_t : Fahre zum Flughafen t Minuten bevor der Flug geht
- Werde ich mit A_t rechtzeitig zum Flughafen kommen?
- Problem:
 - Partielle Beobachtbarkeit: Straßen, Pläne anderer Verkehrsteilnehmer, ...
 - Sensorrauschen (Akgekündigte Flugzeit)
 - Unsicherheit in Effekten der Aktionen (Platter Reifen...)
 - Generelle Modellierungs-Komplexität
- Rein logikbasierte Ansätze riskieren entweder, falsch zu sein (A_{25} wird reichen) oder nicht hilfreich (A_{25} reicht, wenn kein Stau und keine Panne)
- Lösung: Wahrscheinlichkeitsrechnung erlaubt quantifizierte Aussagen über das Eintreten von unsicheren Ereignissen

Warum Wahrscheinlichkeiten?

Wahrscheinlichkeitsrechnung erlaubt:

- Faulheit
 - Zu aufwendig, alle möglichen Aktionen und Effekte zu spezifizieren
 - Extrem große entstehende Zustandsräume, sodass wir keine Lösung in begrenzter Zeit finden
- Theoretische Ignoranz
 - Unvollständiges Wissen / Verständnis von Zusammenhängen, z.B. Medizin, Wirtschaft, ...
- Praktische Ignoranz
 - Selbst wenn vollständige Zusammenhänge im Prinzip verstanden, können wir nicht immer aufwendig zunächst alle Informationen erheben

Grundlagen der Wahrscheinlichkeitsrechnung

- Achtung: Im Folgenden vereinfachte Definitionen, die mathematisch nicht ganz sauber sind, für unsere Zwecke aber ausreichen
- Sei Ω eine Menge (Sample Space) und $\omega \in \Omega$ eine mögliche Welt
- z.B. Gleichzeitiges Werfen eines Würfels und einer Münze:
$$\Omega = \{(1, K), (1, Z), (2, K), \dots, (6, Z)\}$$
- Sei $P : \Omega \rightarrow \mathbb{R}_{\geq 0}$ mit
 - $0 \leq P(\omega) \leq 1$
 - $\sum_{\omega \in \Omega} P(\omega) = 1$
- Nennen P eine Wahrscheinlichkeitsverteilung
- Ein Ereignis A ist eine Teilmenge von Ω , mit

$$P(A) = \sum_{\omega \in A} P(\omega)$$

Grundlagen der Wahrscheinlichkeitsrechnung

- Eine Zufallsvariable ist eine Funktion, die mögliche Welten auf "Teile" der Welt abbildet, z.B. $X((w, m)) = m$
- D.h. eine Zufallsvariable extrahiert Bestandteile der Welt, für die wir uns interessieren
- Eine Wahrscheinlichkeitsverteilung P über Welten definiert eine Wahrscheinlichkeitsverteilung über jeder Zufallsvariablen:

$$P(X = x_i) = \sum_{\omega: X(\omega) = x_i} P(\omega)$$

- Schreibweise: Großbuchstaben für Zufallsvariablen, Kleinbuchstaben für Belegungen

Grundlagen der Wahrscheinlichkeitsrechnung

- Seien X und Y Zufallsvariablen im Sample Space Ω . Dann ist die gemeinsame Wahrscheinlichkeitsverteilung von X und Y :

$$P(X = x, Y = y) = \sum_{\{\omega : X(\omega) = x \wedge Y(\omega) = y\}} P(\omega)$$

- z.B. Sei Ω das gleichzeitige Werfen von einem grünen, einem blauen und einem roten Würfel. Dann ist

$$P(G = 3, R = 1) = 1/36$$

Grundlagen der Wahrscheinlichkeitsrechnung

- Bedingte Wahrscheinlichkeit:

$$P(X | Y) = P(X, Y)/P(Y)$$

- Interpretation: Wahrscheinlichkeit, dass x eintritt, wenn y bereits eingetreten ist
- Durch Umformen ergibt sich die Produktregel:

$$P(X, Y) = P(X | Y) P(Y)$$

- Außerdem ergibt sich direkt der Satz von Bayes:

$$P(X | Y) = P(Y | X) P(X)/P(Y)$$

- Zufallsvariablen X und Y heißen unabhängig, wenn

$$P(X, Y) = P(X) P(Y)$$

Grundlagen der Wahrscheinlichkeitsrechnung

- Summenregel (ergibt sich aus der Definition der gemeinsamen Verteilung):

$$P(X = x) = \sum_y P(X = x, Y = y)$$

- Gesetz der totalen Wahrscheinlichkeit (Summenregel und Def. bedingte Wahrscheinlichkeiten):

$$P(X = x) = \sum_y P(X = x | Y = y)P(Y = y)$$

Beispiel

- Wahrscheinlichkeit einer Krankheit: $P(krank) = 0.0002$
- Sensitivität des Tests: $P(positiv \mid krank) = 1$
- Wahrscheinlichkeit eines falsch-positiven Ergebnisses: $P(positiv \mid gesund) = 0.01$
- Sie haben ein positives Testergebnis erhalten. Was ist die Wahrscheinlichkeit, tatsächlich erkrankt zu sein?

Beispiel

- Wahrscheinlichkeit einer Krankheit: $P(krank) = 0.0002$
- Sensitivität des Tests: $P(positiv | krank) = 1$
- Wahrscheinlichkeit eines falsch-positiven Ergebnisses: $P(positiv | gesund) = 0.01$
- Sie haben ein positives Testergebnis erhalten. Was ist die Wahrscheinlichkeit, tatsächlich erkrankt zu sein?

$$\begin{aligned} P(krank | positiv) &= \frac{P(positiv | krank) P(krank)}{P(positiv)} \\ &= \frac{P(positiv | krank) P(krank)}{P(positiv | krank)P(krank) + P(positiv | gesund)P(gesund)} \\ &= \frac{1 * 0.0002}{1 * 0.0002 + 0.01 * 0.9998} \approx 0.02 \end{aligned}$$

Gemeinsame Wahrscheinlichkeitsverteilung

- Die gemeinsame Wahrscheinlichkeitsverteilung über alle Zufallsvariablen (*full joint*) definiert alle marginalen und bedingten Wahrscheinlichkeiten

		toothache		\neg toothache	
		catch	\neg catch	catch	\neg catch
cavity	.108	.012	.072	.008	
\neg cavity	.016	.064	.144	.576	

Gemeinsame Wahrscheinlichkeitsverteilung

	toothache		\neg toothache	
	catch	\neg catch	catch	\neg catch
cavity	.108	.012	.072	.008
\neg cavity	.016	.064	.144	.576

$$\begin{aligned} P(To = \text{true}) &= \sum_{x \in Cat, y \in Cav} P(To = \text{true}, Cat = x, Cav = y) \\ &= 0.108 + 0.012 + 0.016 + 0.064 = 0.2 \end{aligned}$$

Gemeinsame Wahrscheinlichkeitsverteilung

	toothache		\neg toothache	
	catch	\neg catch	catch	\neg catch
cavity	.108	.012	.072	.008
\neg cavity	.016	.064	.144	.576

$$\begin{aligned}P(Cav = \text{false} | To = \text{true}) &= P(Cav = \text{false}, To = \text{true}) / P(To = \text{true}) \\&= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4\end{aligned}$$

Probabilistische Inferenz durch Aufzählen

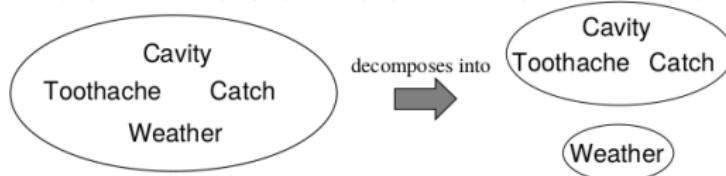
- Sei \mathbf{X} die Sequenz aller Zufallsvariablen
- Im Allgemeinen wollen wir die gemeinsame Verteilung von Query-Variablen \mathbf{Y} , gegeben Werte e der Evidence-Variablen \mathbf{E} berechnen
- Bezeichnen $\mathbf{H} = \mathbf{X} \setminus \mathbf{Y} \setminus E$ als *Hidden* Variablen.
- Generelle Idee: Marginalisiere die Hidden Variables:

$$P(\mathbf{Y} | \mathbf{E} = e) = \alpha \sum_{\mathbf{h}} P(\mathbf{Y}, \mathbf{E} = e, \mathbf{H} = h)$$

- Joint besteht aus $\mathcal{O}(d^n)$ Werten ($n = \text{Anzahl der Variablen, } d = \text{Anzahl der Belegungen jeder Variablen}$)
- Problem: Speichern und summieren über so viele Werte nicht möglich in der Praxis
- Brauchen schlauere Methoden zur Repräsentation und Rechnen mit Wahrscheinlichkeitsverteilungen!

Unabhängigkeit

- Was uns hilft: Unabhängigkeit von Zufallsvariablen



- $P(To, Cav, Cat, W) = P(To, Cav, Cat) P(W)$
- Brauchen nur 12 Zahlen statt 32 zu speichern!
- Weiteres Beispiel: n Münzen: $2^n \rightarrow n$ Werte!
- Aber: Vollständige Unabhängigkeit ist selten...

Bedingte Unabhängigkeit

- Wenn ich Karies hab, ist die Wahrscheinlichkeit, dass die Zahnärztin diesen entdeckt (Catch) unabhängig von meinen Zahnschmerzen:

$$P(Cat | To, Cav = \text{true}) = P(Cat | Cav = \text{true})$$

- Das gleiche, wenn ich kein Karies hab:

$$P(Cat | To, Cav = \text{false}) = P(Cat | Cav = \text{false})$$

- Insgesamt: *Cat* ist bedingt unabhängig von *To*, gegeben *Cav*:
 $P(Cat | To, Cav) = P(Cat | Cav)$

- Erlaubt es, die Full Joint zu schreiben als

$$P(Cat, To, Cav) = P(To | Ca) P(Ca | Cat) P(Ca)$$

- Nur 5 Werte statt ursprünglich 7

Zusammenfassung

- Wahrscheinlichkeitsrechnung erlaubt die Formalisierung von unsicherem Wissen
- Die Full Joint spezifiziert ein Wahrscheinlichkeitsmodell vollständig
- Queries können durch Summierung über die nicht relevanten Variablen (Marginalisierung) beantwortet werden
- Full Joint ist in der Praxis zu groß, müssen kompaktere Darstellung finden
- Unabhängigkeit und bedingte Unabhängigkeit erlauben dies

Künstliche Intelligenz

Bayes'sche Netze

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Motivation

- Die Full Joint erlaubt es uns, jede Query zu beantworten
- Aber: Größe der Full Joint wächst exponentiell mit Anzahl der Variablen
- Können (bedingte) Unabhängigkeit ausnutzen, um Verteilung schlauer zu repräsentieren
- Bayes'sches Netz: Repräsentation einer Wahrscheinlichkeitsverteilung, in der die (Un)abhängigkeiten zwischen Variablen explizit gemacht werden

Bayesian networks

A simple, graphical notation for conditional independence assertions
and hence for compact specification of full joint distributions

Syntax:

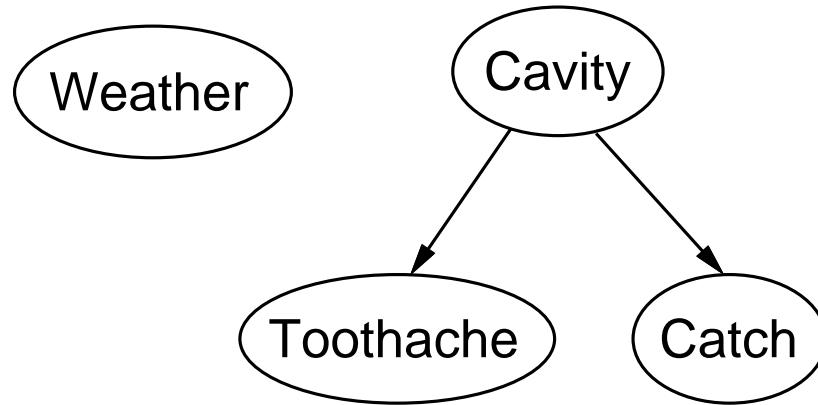
- a set of nodes, one per variable
- a directed, acyclic graph (link \approx “directly influences”)
- a conditional distribution for each node given its parents:

$$\mathbf{P}(X_i | \text{Parents}(X_i))$$

In the simplest case, conditional distribution represented as
a **conditional probability table** (CPT) giving the
distribution over X_i for each combination of parent values

Example

Topology of network encodes conditional independence assertions:



Weather is independent of the other variables

Toothache and *Catch* are conditionally independent given *Cavity*

Example

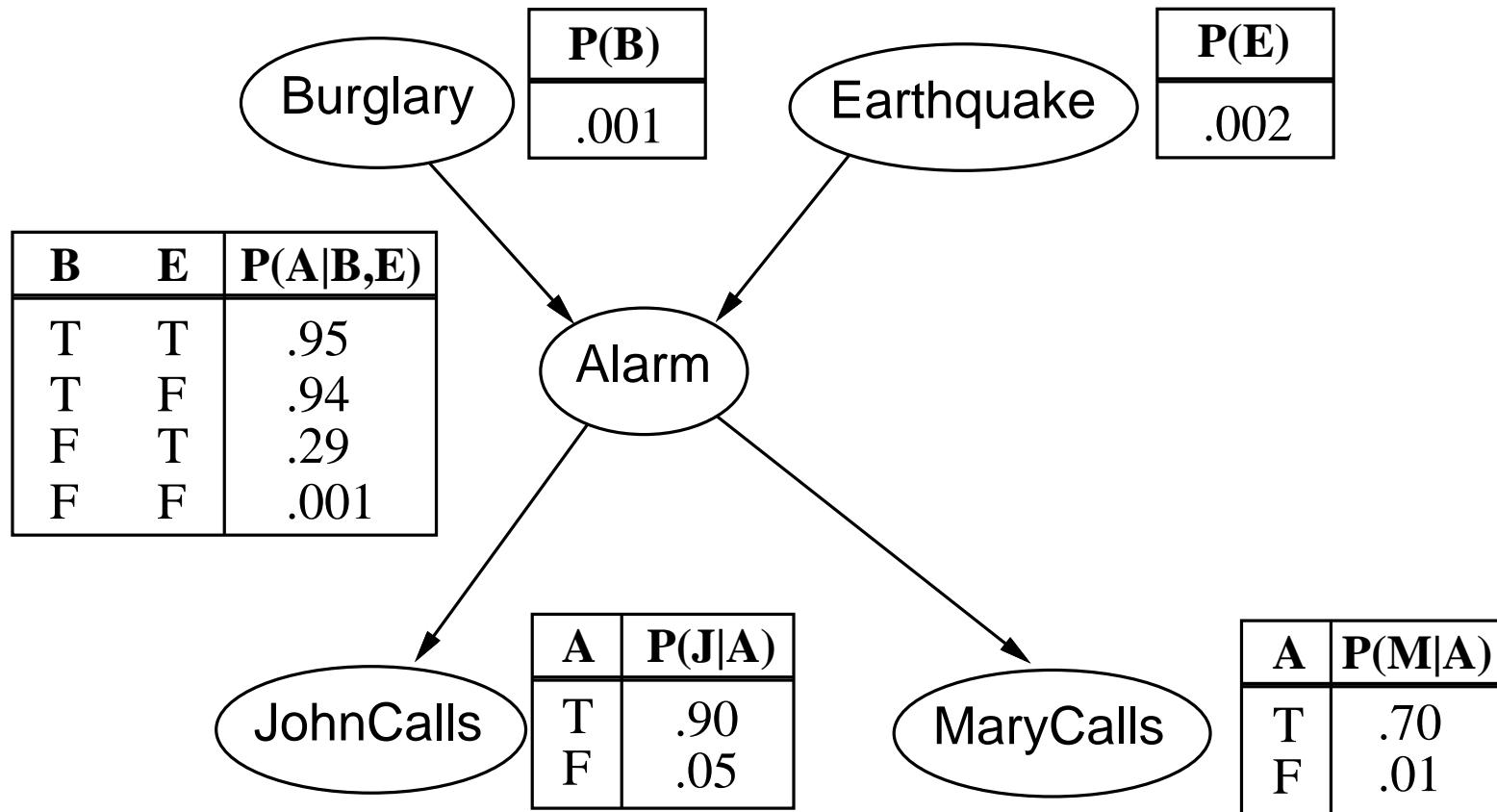
I'm at work, neighbor John calls to say my alarm is ringing, but neighbor Mary doesn't call. Sometimes it's set off by minor earthquakes. Is there a burglar?

Variables: *Burglar*, *Earthquake*, *Alarm*, *JohnCalls*, *MaryCalls*

Network topology reflects “causal” knowledge:

- A burglar can set the alarm off
- An earthquake can set the alarm off
- The alarm can cause Mary to call
- The alarm can cause John to call

Example contd.



Compactness

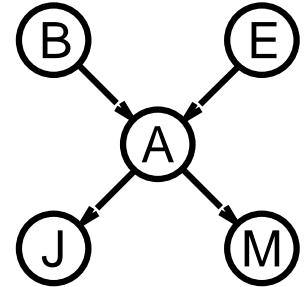
A CPT for Boolean X_i with k Boolean parents has 2^k rows for the combinations of parent values

Each row requires one number p for $X_i = \text{true}$
(the number for $X_i = \text{false}$ is just $1 - p$)

If each variable has no more than k parents,
the complete network requires $O(n \cdot 2^k)$ numbers

I.e., grows linearly with n , vs. $O(2^n)$ for the full joint distribution

For burglary net, $1 + 1 + 4 + 2 + 2 = 10$ numbers (vs. $2^5 - 1 = 31$)



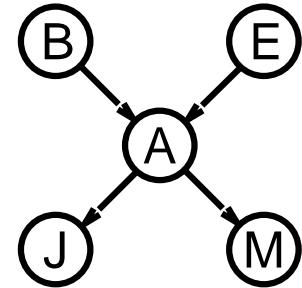
Global semantics

Global semantics defines the full joint distribution as the product of the local conditional distributions:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$$

e.g., $P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$

=



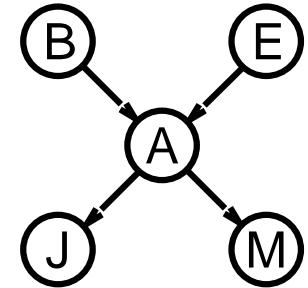
Global semantics

“Global” semantics defines the full joint distribution as the product of the local conditional distributions:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$$

e.g., $P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$

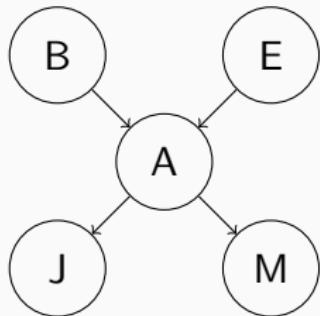
$$\begin{aligned} &= P(j|a)P(m|a)P(a|\neg b, \neg e)P(\neg b)P(\neg e) \\ &= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \\ &\approx 0.00063 \end{aligned}$$



Belief Network (Example)

$P(b)$
0.01

$P(e)$
0.002



B	E	$P(a)$
T	T	0.95
T	⊥	0.94
⊥	T	0.29
⊥	⊥	0.001

A	$P(j)$
T	0.90
⊥	0.05

A	$P(m)$
T	0.70
⊥	0.01

What is $P(b | j, m)$?

Inference Tasks

Basic inference task in BNs: Compute the posterior probability of a (set of) variable(s) given some evidence.

Notation:

- ▶ Let N be a Bayesian Network over the set of variables V
- ▶ Let X be a random variable we are interested in (query variable)
- ▶ Let $E = \{E_1, E_2, \dots, E_n\}$ be a set of evidence variables
- ▶ Let e be one particular observed event, i.e., an assignment of values to E
- ▶ We call $Y = V \setminus (\{X\} \cup E)$ the set of hidden (non-query) variables

Basic inference task:

Compute $P(X | e)$ wrt. the dependencies defined by N

Exact Inference by Enumeration

- ▶ From the laws of conditional probabilities we know:

$$P(X | e) = \frac{P(X, e)}{P(e)}$$

- ▶ Using the laws of probability, we find:

$$P(X | e) = \frac{\sum_y P(X, e, y)}{P(e)}$$

with y being an assignment to all variables in Y .

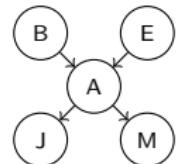
- ▶ I.e., we can compute $P(X | e)$ by summing over all possible assignments of the hidden variables.

Compute $P(b \mid j, m)$ by Enumeration

► From

- the definitions for conditional probabilities, the chain rule, and
- independence assumptions of the domain, we get:

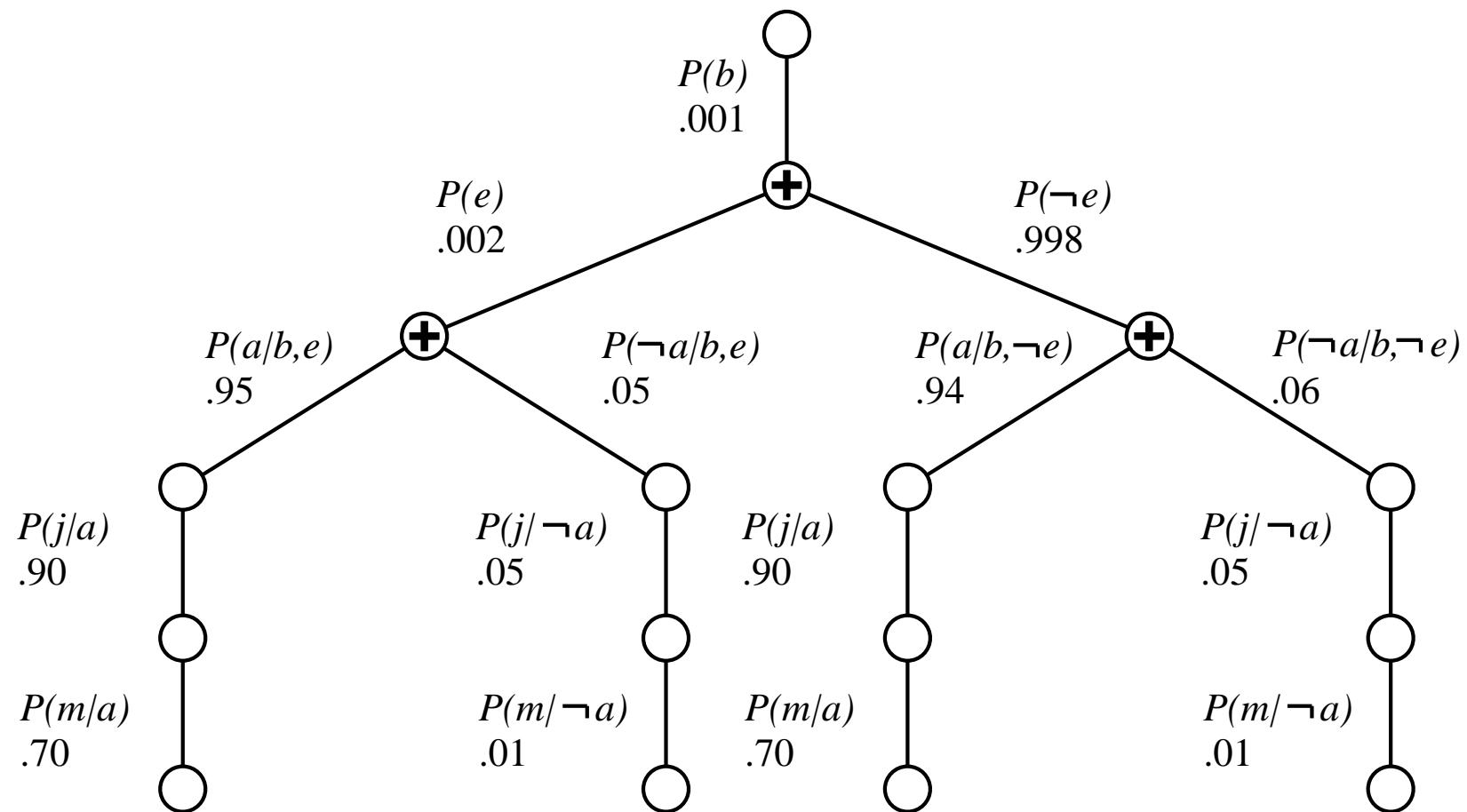
$$\begin{aligned} P(b \mid j, m) &= \frac{P(b, j, m)}{P(j, m)} = c \cdot P(b, j, m) \\ &= c \cdot \sum_{e' \in \text{dom}_E} \sum_{a' \in \text{dom}_A} P(b, j, m, e', a') \\ &= c \cdot \sum_{e' \in \text{dom}_E} \sum_{a' \in \text{dom}_A} P(b)P(e')P(a' \mid b, e')P(j \mid a')P(m \mid a') \\ &= c \cdot P(b) \sum_{e' \in \text{dom}_E} P(e') \sum_{a' \in \text{dom}_A} P(a' \mid b, e')P(j \mid a')P(m \mid a') \end{aligned}$$



► Notation:

- lower case letters represent assigned random variables, i.e., b is the short-hand notation for $B = \top$
- “primed” lower case letters represent variables over which sums are computed, e' only occurs within the $\sum_{e' \in \text{dom}_E}$

Evaluation tree



Enumeration is inefficient: repeated computation

e.g., computes $P(j|a)P(m|a)$ for each value of e

Complexity of the Inference by Enumeration

- ▶ In the worst case, we have to sum out almost all the variables
- ▶ For n Boolean variables, the complexity is in the order of $O(n2^n)$

What is $P(b \mid j, m)$ again?

$$\begin{aligned} P(b \mid j, m) &= c \cdot P(b) \sum_{e' \in \text{dom}_E} P(e') \sum_{a' \in \text{dom}_A} P(a' \mid b, e') P(j \mid a') P(m \mid a') \\ &= c \cdot P(b) \sum_{e' \in \text{dom}_E} P(e') (P(a \mid b, e') P(j \mid a) P(m \mid a) + \\ &\quad P(\neg a \mid b, e') P(j \mid \neg a) P(m \mid \neg a)) \\ &= c \cdot P(b) \left(P(e) (P(a \mid b, e) P(j \mid a) P(m \mid a) + \right. \\ &\quad P(\neg a \mid b, e) P(j \mid \neg a) P(m \mid \neg a)) + \\ &\quad P(\neg e) (P(a \mid b, \neg e) P(j \mid a) P(m \mid a) + \\ &\quad \left. P(\neg a \mid b, \neg e) P(j \mid \neg a) P(m \mid \neg a) \right) \end{aligned}$$

Internal structure of the Formula

- ▶ There is an internal structure due to the summation over possible assignments:

$$\begin{aligned} P(b \mid j, m) = c \cdot P(b) & \left(P(e)(P(a \mid b, e) P(j \mid a) P(m \mid a) + \right. \\ & P(\neg a \mid b, e) P(j \mid \neg a) P(m \mid \neg a)) + \\ & P(\neg e)(P(a \mid b, \neg e) P(j \mid a) P(m \mid a) + \\ & \left. P(\neg a \mid b, \neg e) P(j \mid \neg a) P(m \mid \neg a) \right) \end{aligned}$$

- ▶ Certain sub-formulae can be re-used, e.g.:

- $P(m \mid a)$
- $P(m \mid \neg a)$
- $P(j \mid a)$
- $P(j \mid \neg a)$

Complexity of the Inference by Enumeration and Caching

- ▶ In the worst case, we have to sum out almost all the variables
- ▶ But we can save some results for re-usage

Exact Inference by Variable Elimination

- ▶ We can utilise the structure of the equation
- ▶ Sum out the variables from “right to left” and store the intermediate results
- ▶ Problem: How to store the intermediate results?
- ▶ Solution: The intermediate results are called *Factors*

Factorisation of $P(B \mid j, m)$

- The equation for the distribution over B given j and m :

$$P(B \mid j, m) = c \cdot \underbrace{P(B)}_{f_1(B)} \cdot \sum_{e' \in \text{dom } E} \underbrace{P(e')}_{f_2(E)} \cdot \sum_{a' \in \text{dom } A} \underbrace{P(a' \mid b, e')}_{f_3(A, B, E)} \cdot \underbrace{P(j \mid a')}_{f_4(j, A)} \cdot \underbrace{P(m \mid a')}_{f_5(M, A)}$$
$$\underbrace{f'_4(A) = f_4^j}_{f'_4(A) = f_4^j} \quad \underbrace{f'_5(A) = f_5^m}_{f'_5(A) = f_5^m}$$
$$\underbrace{f_6(A) = (f'_4 * f'_5)}_{f_6(A) = (f'_4 * f'_5)}$$
$$\underbrace{f_7(A, B, E) = (f_3 * f_6)}_{f_7(A, B, E) = (f_3 * f_6)}$$
$$\underbrace{f_8(B, E) = (\sum_A f_7)}_{f_8(B, E) = (\sum_A f_7)}$$
$$\underbrace{f_9(B, E) = (f_2 * f_8)}_{f_9(B, E) = (f_2 * f_8)}$$
$$\underbrace{f_{10}(B) = (\sum_E f_9)}_{f_{10}(B) = (\sum_E f_9)}$$
$$f_{11}(B) = (f_1 * f_{10})$$

- Operations of factors:

- Primitive sub-expressions are renamed, e.g., $P(B) = f_1(B)$
- Values can be assigned to variables, e.g., $f'_4(A) = f_4^j = f_4^{j=\top}$
- Factors can be multiplied, e.g., $f_6(A) = (f_4 * f_5)$
- Variable can be summed out, e.g., $f_8(B, E) = (\sum_A f_7)$

Factors

- ▶ An n -dimensional *factor* f is a (representation of a) function from n random variables X_1, \dots, X_n to a positive real number.
 - A factor can be a probability distribution (summing up to 1)
 - but it does not need to be
- ▶ Notation for factor f over X_1, \dots, X_j : $f(X_1, \dots, X_j)$
- ▶ A simple example of a factor over three binary random variables

$f(X, Y, Z):$

X	Y	Z	val
T	T	T	0.1
T	T	⊥	0.9
T	⊥	T	0.2
T	⊥	⊥	0.8
⊥	T	T	0.4
⊥	T	⊥	0.6
⊥	⊥	T	0.3
⊥	⊥	⊥	0.7

Operations on Factors: Assignment of Values to Variables

We can assign some or all of the variables of a factor:

- ▶ $f(X_1=v_1, X_2, \dots, X_j)$, where $v_1 \in \text{dom}(X_1)$:
is a factor over X_2, \dots, X_j .
- ▶ $f(X_1=v_1, X_2=v_2, \dots, X_j=v_j)$ is a number,
it is the value of f when each X_i has value v_i .
- ▶ Notation for $f(X_1=v_1, X_2, \dots, X_j)$: $f(X_1, X_2, \dots, X_j)^{X_1=v_1}$

Operations on Factors: Assignment of Values to Variables (Example)

$f :$

X	Y	Z	val
T	T	T	0.1
T	T	⊥	0.9
T	⊥	T	0.2
T	⊥	⊥	0.8
⊥	T	T	0.4
⊥	T	⊥	0.6
⊥	⊥	T	0.3
⊥	⊥	⊥	0.7

$f^{X=t} :$

Y	Z	val
T	T	0.1
T	⊥	0.9
⊥	T	0.2
⊥	⊥	0.8

$f^{X=t, Z=f} :$

Y	val
T	0.9
⊥	0.8

$$f^{X=t, Y=f, Z=f} = 0.8$$

Operations on Factors: Product of Two Factors

The *product* of factor $f_1(X, Y)$ and $f_2(Y, Z)$, where Y are the variables in common, is the factor $(f_1 * f_2)(X, Y, Z)$ defined by:

$$(f_1 * f_2)(X, Y, Z) = f_1(X, Y)f_2(Y, Z).$$

Operations on Factors: Product of Two Factors (Example)

	A	B	val
$f_1:$	T	T	0.1
	T	⊥	0.9
	⊥	T	0.2
	⊥	⊥	0.8

	B	C	val
$f_2:$	T	T	0.3
	T	⊥	0.7
	⊥	T	0.6
	⊥	⊥	0.4

	A	B	C	val
$(f_1 * f_2):$	T	T	T	0.03
	T	T	⊥	0.07
	T	⊥	T	0.54
	T	⊥	⊥	0.36
	⊥	T	T	0.06
	⊥	T	⊥	0.14
	⊥	⊥	T	0.48
	⊥	⊥	⊥	0.32

Operations on Factors: Summing out a Variable from a Factor

We can *sum out* a variable, say X_1 with domain $\{v_1, \dots, v_k\}$, from factor $f(X_1, \dots, X_j)$. This results in a factor on X_2, \dots, X_j , defined as follows:

$$\begin{aligned}(\sum_{X_1} f)(X_2, \dots, X_j) &= f(X_1, \dots, X_j)^{X_1=v_1} + \dots + f(X_1, \dots, X_j)^{X_1=v_k} \\&= \sum_{v \in \text{dom}(X_1)} f(X_1, X_2, \dots, X_j)^{X_1=v}\end{aligned}$$

Operations on Factors: Summing out a Variable from a Factor (Example)

$f_3:$

<i>A</i>	<i>B</i>	<i>C</i>	<i>val</i>
T	T	T	0.03
T	T	⊥	0.07
T	⊥	T	0.54
T	⊥	⊥	0.36
⊥	T	T	0.06
⊥	T	⊥	0.14
⊥	⊥	T	0.48
⊥	⊥	⊥	0.32

$(\sum_B f_3):$

<i>A</i>	<i>C</i>	<i>val</i>
T	T	0.57
T	⊥	0.43
⊥	T	0.54
⊥	⊥	0.46

Exact Inference by Variable Elimination

Factorisation of $P(B \mid j, m)$

$$\begin{aligned}
 P(B \mid j, m) &= c \cdot \underbrace{P(B)}_{f_1(B)} \cdot \sum_{e' \in \text{dom}_E} \underbrace{P(e')}_{f_2(E)} \cdot \sum_{a' \in \text{dom}_A} \underbrace{P(a' \mid b, e')}_{f_3(A, B, E)} \cdot \underbrace{P(j \mid a')}_{f_4(j, A)} \cdot \underbrace{P(m \mid a')}_{f_5(M, A)} \\
 &\quad \underbrace{\overbrace{f'_4(A)=f_4^j}^{\text{f}_4'(A)=f_4^j}, \overbrace{f'_5(A)=f_5^m}^{\text{f}_5'(A)=f_5^m}}_{\text{f}_6(A)=(f'_4 * f'_5)} \\
 &\quad \underbrace{\overbrace{\text{f}_7(A, B, E)=(f_3 * \text{f}_6)}^{\text{f}_7(A, B, E)=(f_3 * f_6)}}_{\text{f}_8(B, E)=(\sum_A \text{f}_7)} \\
 &\quad \underbrace{\overbrace{\text{f}_9(B, E)=(f_2 * \text{f}_8)}^{\text{f}_9(B, E)=(f_2 * f_8)}}_{\text{f}_{10}(B)=(\sum_E \text{f}_9)} \\
 &\quad \underbrace{\text{f}_{11}(B)=(\text{f}_1 * \text{f}_{10})}_{\text{f}_{11}(B)=(f_1 * f_{10})}
 \end{aligned}$$

- ▶ Assign the value $M = \top$ in $f'_5(A) = f_5^{M=\top}$
- ▶ Compute $f_6(A) = (f'_4 * f'_5)$
- ▶ Sum out A and compute $f_8(B, E) = (\sum_A f_7)$

Zusammenfassung

- Bayes'sche Netze sind natürliche Repräsentation für Verteilungen mit bedingten Unabhängigkeiten
- Topologie + CPTs = Kompakte Repräsentation der Full Joint
- Einfach zu spezifizieren, auch für Nichtexperten
- Exakte Inferenz durch Aufzählen nur in einfachen Fällen möglich
- Variable Elimination ist ein effizienterer exakter Inferenzalgorithmus
- Noch effizienter: Approximative Inferenzalgorithmen (hier nicht behandelt)

Künstliche Intelligenz

Hidden Markov Models

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Motivation

- Die Welt verändert sich über die Zeit
 - Wahrscheinlichkeitsverteilungen müssen für jeden Zeitschritt angepasst werden
 - Aber: Systemzustand ist abhängig von der Vergangenheit
- Generelle Idee
 - Großes Bayes'sches Netz, in dem es Knoten (= Zufallsvariablen) für jeden Zeitpunkt gibt
 - Spezialisierte Inferenzalgorithmen, die die besondere Netzwerkstruktur ausnutzen

Sequentielle Prozesse

- Gegeben: Zustandsraum X , zu jedem Zeitpunkt t hat das System einen Zustand $x_t \in X$
- Auch gegeben: Beobachtungsraum Y , zu jedem Zeitpunkt t machen wir eine Beobachtung $y_t \in Y$
- Was ist eine sinnvolle Struktur des Bayes'sches Netzes? Wie hängen Zufallsvariablen voneinander ab?

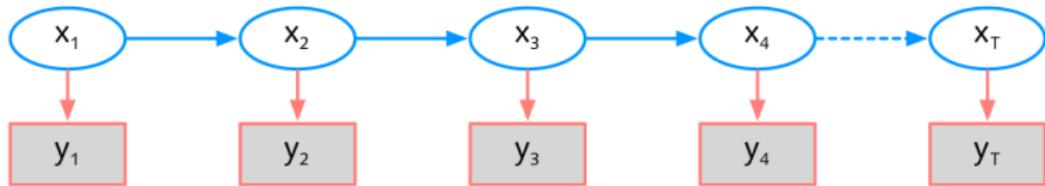
Markov-Ketten

- Annahme: Zustände bilden eine *Markov-Kette erster Ordnung*: Zustand X_t hängt nur vom Zustand X_{t-1} ab
- Außerdem nehmen wir an, dass sich die Art der Abhängigkeit im Verlauf der Zeit nicht ändert, d.h. diese kann über eine bedingte Wahrscheinlichkeitsverteilung $P(X_t | X_{t-1})$ beschrieben werden
- Wir nennen $P(X_t | X_{t-1})$ *Transitionsmodell*



Observationsmodell

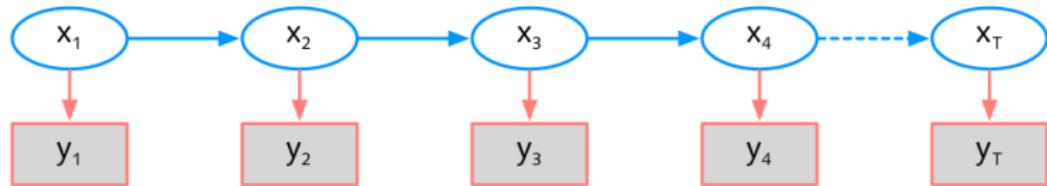
- Annahme: Beobachtung Y_t hängt nur von Zustand X_t ab
- Auch diese Abhängigkeit ändert sich nicht im Verlauf der Zeit, d.h. Repräsentation als bedingte Verteilung $P(Y_t | X_t)$
- Nennen $P(Y_t | X_t)$ *Observationsmodell*



Hidden Markov Model

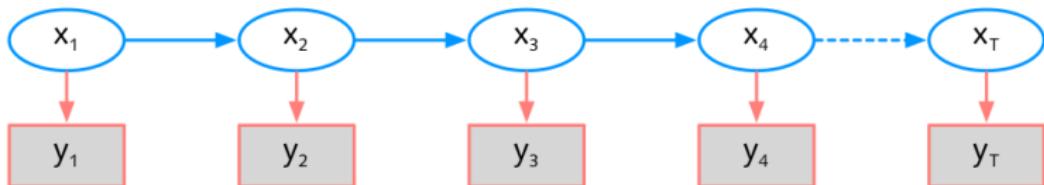
Ein *Hidden Markov Model* ist definiert durch

- Transitionsmodell $P(X_t | X_{t-1})$
- Observationsmodell $P(Y_t | X_t)$
- Initialverteilung (A-Priori-Verteilung) $P(X_1)$



Inferenz in HMMs

- Query: Berechne $P(X_t | y_1, \dots, y_t) = P(X_t | y_{1:t})$ für jedes t
 - Diese Inferenzaufgabe nennt sich *Filtering*, später werden wir noch andere typische Aufgaben kennenlernen
- Im Prinzip mit Standard-Inferenzalgorithmen für Bayes'sche Netze, aber das ist unnötig ineffizient
- Kann stattdessen die besondere Netzwerkstruktur ausnutzen, um effiziente, rekursive Formulierung zu erhalten



Inferenz in HMMs: Filtering

- Angenommen, wir haben $P(X_{t-1} | y_{1:t-1})$ (d.h. Filtering-Query für $t - 1$) schon ausgerechnet
- Gehen jetzt in 2 Schritten vor:
 - Berechne *Vorhersage* $P(X_t | y_{1:t-1})$ (d.h. ohne Einbeziehen von y_t)
 - Berechne *Korrektur* $P(X_t | y_{1:t-1}, y_t)$

Filtering: Vorhersage

$$\begin{aligned} P(x_t \mid y_{1:t-1}) &= \sum_{x_{t-1}} P(x_t, x_{t-1} \mid y_{1:t-1}) \\ &= \sum_{x_{t-1}} P(x_t \mid x_{t-1} y_{1:t-1}) P(x_{t-1} \mid y_{1:t-1}) \\ &= \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1} \mid y_{1:t-1}) \end{aligned}$$

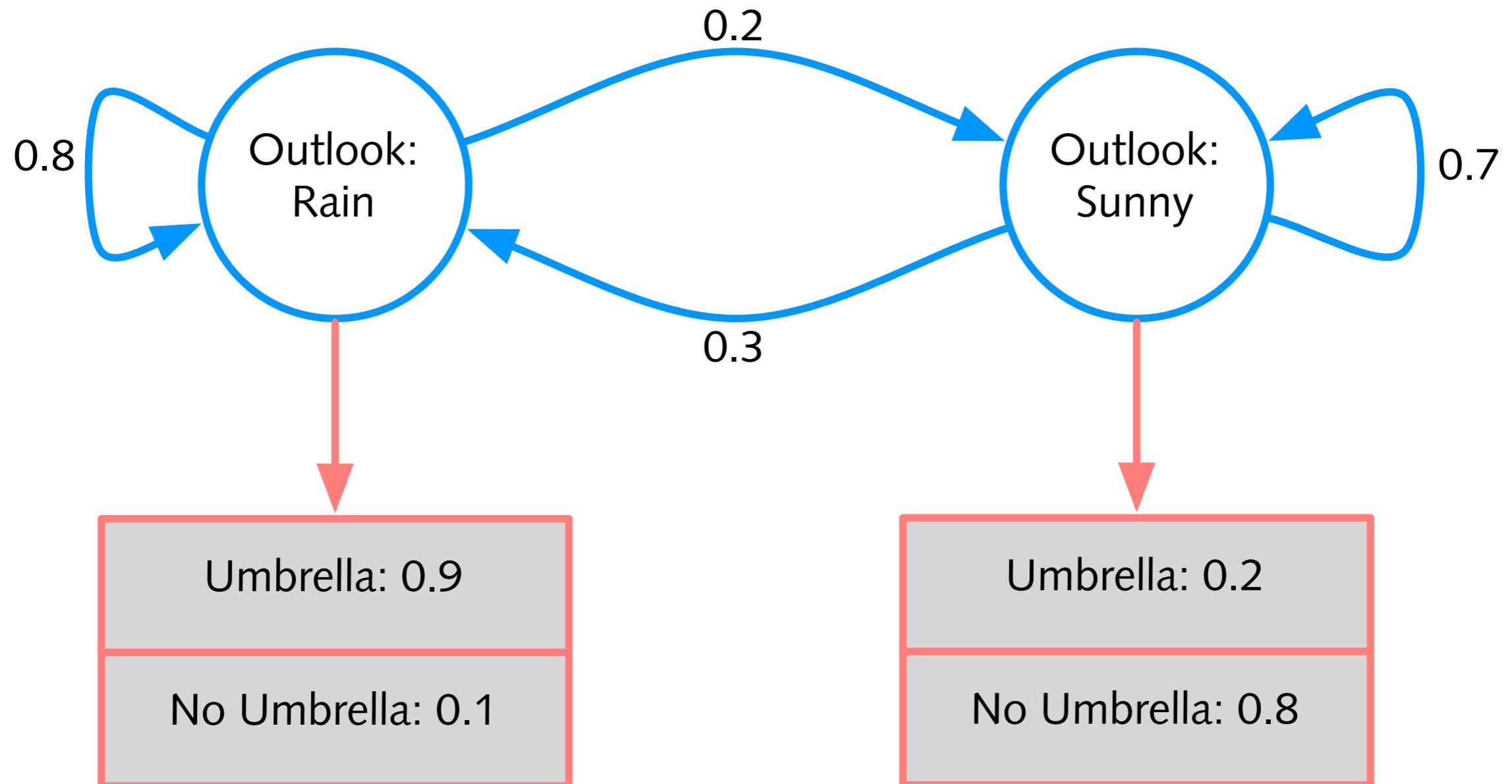
Filtering: Korrektur

$$\begin{aligned} P(x_t \mid y_{1:t-1}, y_t) &= \frac{P(y_t \mid x_t, y_{1:t-1}) P(x_t \mid y_{1:t-1})}{P(y_t \mid y_{1:t-1})} \\ &= \frac{P(y_t \mid x_t) P(x_t \mid y_{1:t-1})}{P(y_t)} \end{aligned}$$

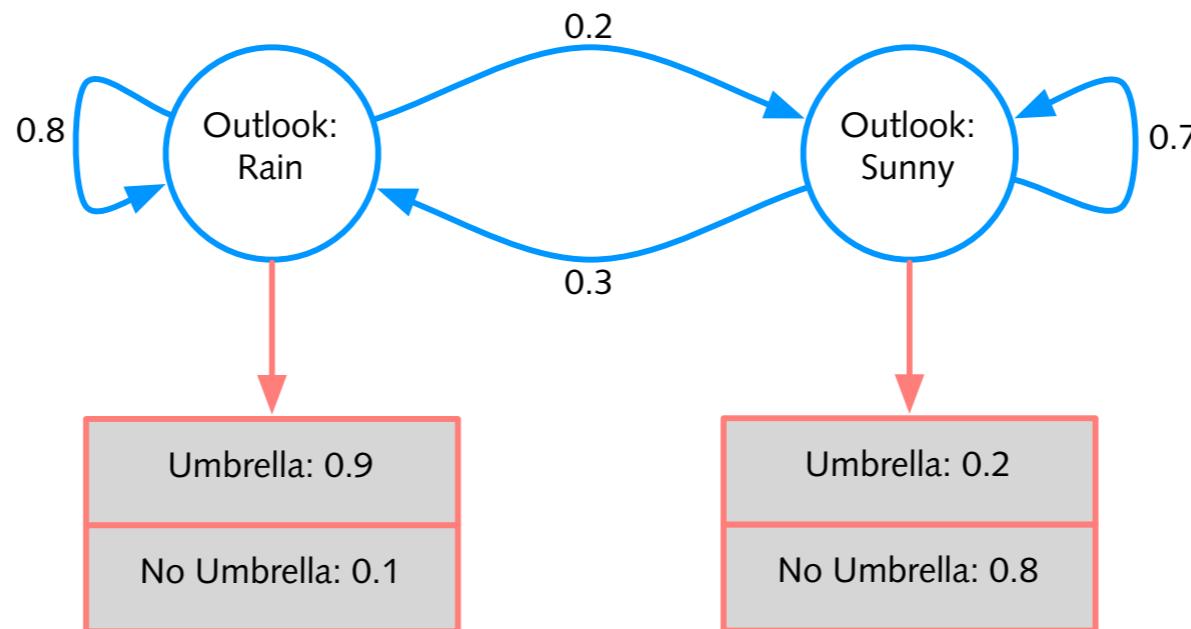
Der Term $P(y_t) = \sum_{x_t} P(y_t \mid x_t) P(x_t \mid y_{1:t-1})$ ist ein Normalisierungsfaktor und ergibt sich automatisch, wenn wir den Zähler für alle x_t berechnen

1 Sequentielle Prozesse

Hier ein einfaches Modell:



Sequentielle Prozesse



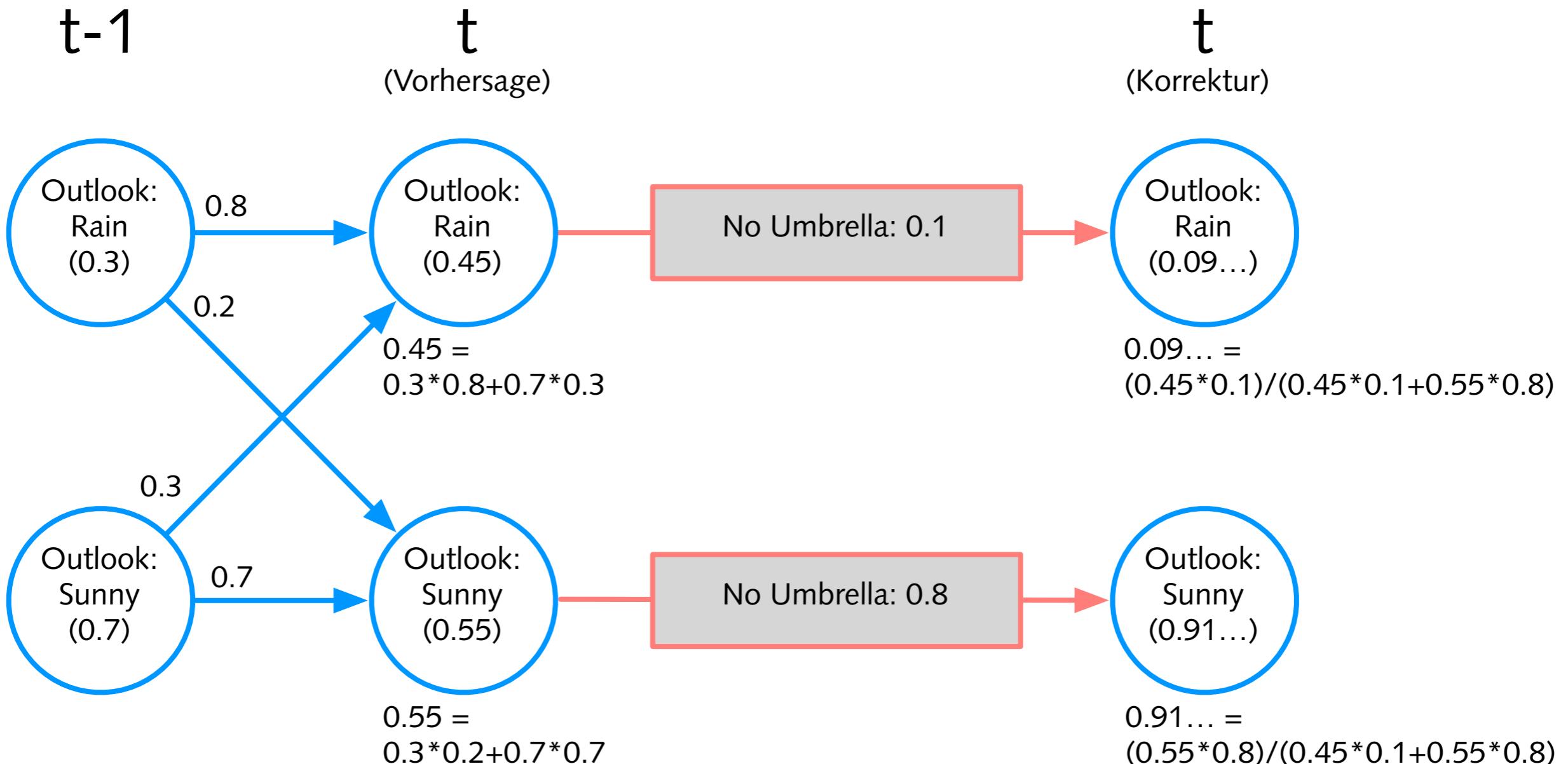
$$\mathbf{A} = \begin{array}{c|cc} & x_{t-1} \\ \hline x_t & Rain & Sunny \end{array} \quad \begin{array}{c|cc} & x_{t-1} \\ \hline Rain & 0.8 & 0.3 \\ Sunny & 0.2 & 0.7 \end{array}$$

$$\mathbf{O} = \begin{array}{c|cc} & y_t \\ \hline x_t & No\,Umbr. & Umbr. \end{array} \quad \begin{array}{c|cc} & y_t \\ \hline Rain & 0.1 & 0.9 \\ Sunny & 0.8 & 0.2 \end{array}$$

Nicht im Diagramm:

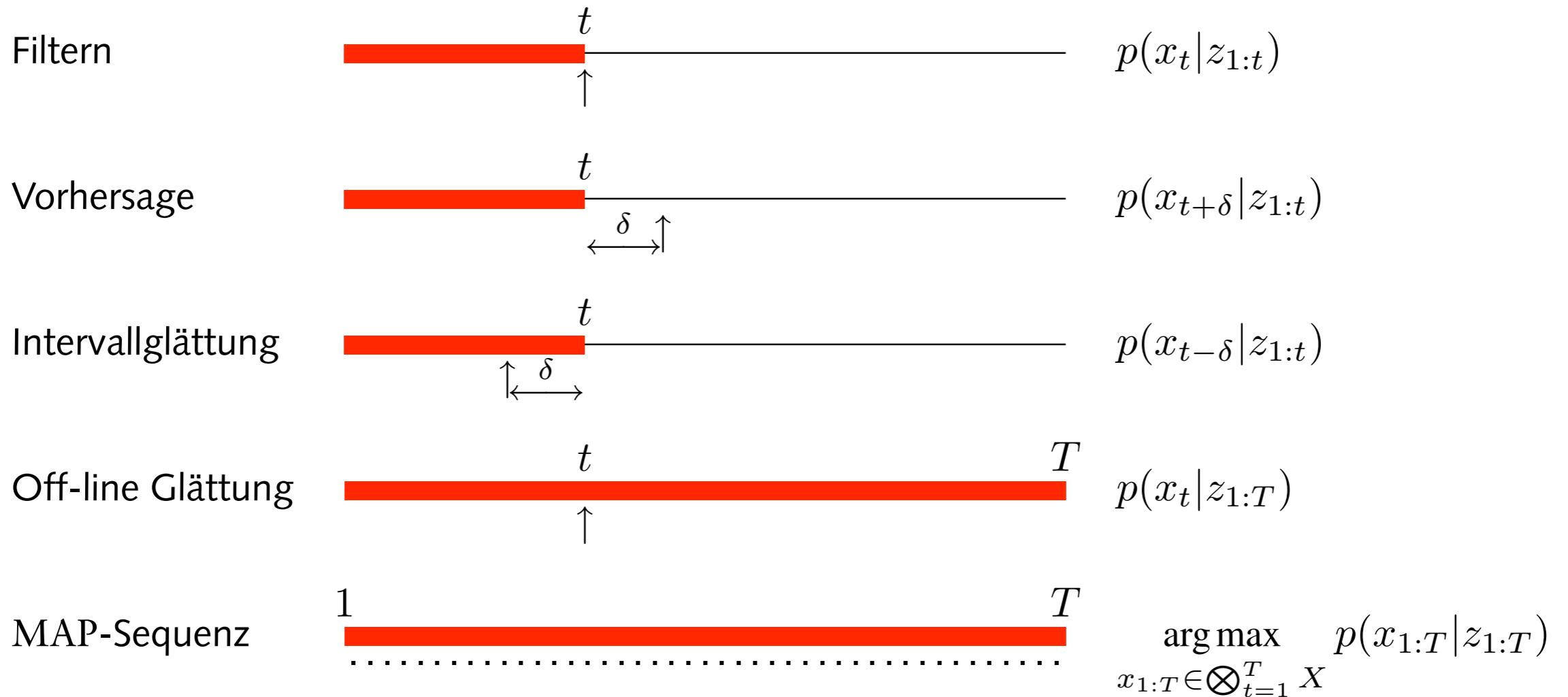
$$p(x_1) = \begin{array}{c|cc} & Rain & Sunny \end{array} \quad \begin{array}{c|cc} & 0.3 & 0.7 \end{array}$$

2 Inferenz in sequentiellen Prozessen



2 Inferenz in sequentiellen Prozessen

Filtern, Vorhersage, Glätten und MAP-Sequenz im Vergleich

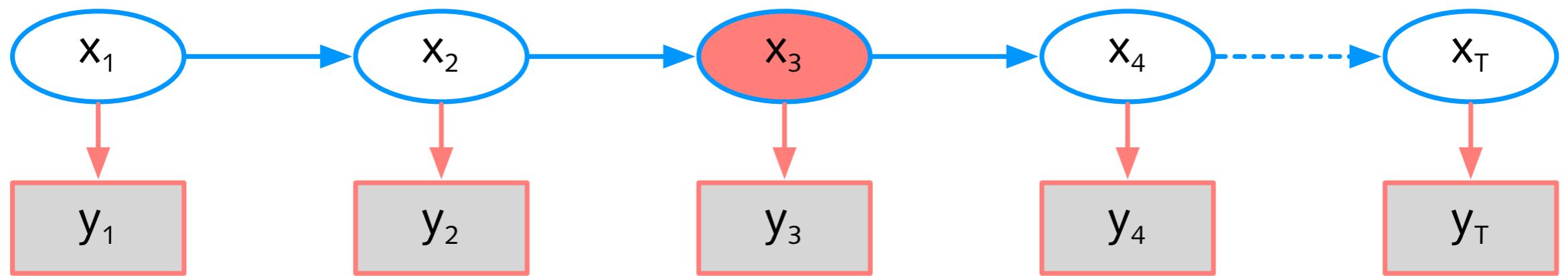


2 Inferenz in sequentiellen Prozessen

Die Bestimmung von $p(x_t | y_{1:t})$ wird auch als *Filterung* bezeichnet.

Ein weiteres Verfahren von Interesse nennt sich *Glättung*.

D.h., gesucht: $p(x_t | y_{1:T})$



Die Idee ist, für die Schätzung zum Zeitpunkt t (hier: $t = 3$) die gesamte Sequenz von Beobachtungen $y_{1:T}$ zu verwenden.

Analog zum Lesen eines Kriminalromans: Die Vermutung über den Täter auf Seite 1 ist oft eine andere, als die nach dem Lesen der letzten Seite.

2 Inferenz in sequentiellen Prozessen

$$\begin{aligned} p(x_t | y_{1:T}) &= \sum_{x_{t+1}} p(x_t, x_{t+1} | y_{1:T}) && \text{Marginalisierung} \\ &= \sum_{x_{t+1}} p(x_t | x_{t+1}, y_{1:T}) p(x_{t+1} | y_{1:T}) && \text{Kettenregel} \\ &= \sum_{x_{t+1}} p(x_t | x_{t+1}, y_{1:t}) p_t(x_{t+1} | y_{1:T}) && \text{Unabh. in HMM} \\ &= \sum_{x_{t+1}} \frac{p(x_{t+1} | x_t, y_{1:t}) p(x_t | y_{1:t})}{p(x_{t+1} | y_{1:t})} p(x_{t+1} | y_{1:T}) && \text{Bayes} \\ &= \sum_{x_{t+1}} \frac{p(x_{t+1} | x_t) p(x_t | y_{1:t})}{p(x_{t+1} | y_{1:t})} p(x_{t+1} | y_{1:T}) && \text{Unabh. in HMM} \\ &= p(x_t | y_{1:t}) \sum_{x_{t+1}} p(x_{t+1} | x_t) \frac{p(x_{t+1} | y_{1:T})}{p(x_{t+1} | y_{1:t})} \end{aligned}$$

Die Glättung arbeitet von hinten nach vorne und nutzt dabei Vorhersage und Ergebnis der Filterung.

Sequentielle Prozesse

Annahme:

- Systemmatrix \mathbf{A} gegeben
- Observierungsmatrix \mathbf{O} gegeben
- Zustandswahrscheinlichkeiten zum Zeitpunkt $t - 1$ als Vektor \mathbf{p}_{t-1} gegeben
- Beobachtung y_t gegeben.

Dann erhalten wir

- den Vektor der Vorhersagewahrscheinlichkeiten $\mathbf{p}_{t|t-1}$ durch:
$$\mathbf{p}_{t|t-1} = \mathbf{A}\mathbf{p}_{t-1}$$
- und die Zustandswahrscheinlichkeiten für den Zeitpunkt t durch:

$$\mathbf{p}_t = \frac{1}{\|\tilde{\mathbf{p}}_t\|_1} \tilde{\mathbf{p}}_t \quad \text{wobei } \tilde{\mathbf{p}}_t = \mathbf{p}_{t|t-1} \circ \mathbf{O}_{[\cdot, y_t]}$$

Sequentielle Prozesse

Dabei ist:

- $a \circ b$ das Hadamard-Produkt (auch Schur-Produkt genannt)
- $O_{[\cdot, y_t]}$ der zu y_t korrespondierende Spaltenvektor der Matrix O .
- $\|a\|_1$ die L_1 -Norm des Vektors a , also einfach die Summe der Absolutbeträge der Vektorelemente.

In R:

```
fstep <- function(A, O, yt, pt1) {  
  tilde.pt <- O[,yt] * (A %*% pt1)  
  tilde.pt / sum(tilde.pt)  
}
```

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?
 - Explizite Konstruktion aus Domänenwissen

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?
 - Explizite Konstruktion aus Domänenwissen
 - Angenommen, Sequenzen x_1, \dots, x_T und y_1, \dots, y_T vorhanden (sog. *überwachtes Lernen*): Relative Häufigkeiten (“Auszählen”)

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?
 - Explizite Konstruktion aus Domänenwissen
 - Angenommen, Sequenzen x_1, \dots, x_T und y_1, \dots, y_T vorhanden (sog. *überwachtes Lernen*): Relative Häufigkeiten (“Auszählen”)

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?
 - Explizite Konstruktion aus Domänenwissen
 - Angenommen, Sequenzen x_1, \dots, x_T und y_1, \dots, y_T vorhanden (sog. *überwachtes Lernen*): Relative Häufigkeiten ("Auszählen")
 - Angenommen, nur Sequenzen y_1, \dots, y_T vorhanden (*unüberwachtes Lernen*): Expectation Maximization-Algorithmus (\rightarrow *Statistische Signalverarbeitung und Inferenz*)

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?
 - Explizite Konstruktion aus Domänenwissen
 - Angenommen, Sequenzen x_1, \dots, x_T und y_1, \dots, y_T vorhanden (sog. *überwachtes Lernen*): Relative Häufigkeiten ("Auszählen")
 - Angenommen, nur Sequenzen y_1, \dots, y_T vorhanden (*unüberwachtes Lernen*): Expectation Maximization-Algorithmus (\rightarrow *Statistische Signalverarbeitung und Inferenz*)
- Inferenzalgorithmen (mindestens) linear in $|X|$ – was, wenn $|X|$ sehr groß ist?

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?
 - Explizite Konstruktion aus Domänenwissen
 - Angenommen, Sequenzen x_1, \dots, x_T und y_1, \dots, y_T vorhanden (sog. *überwachtes Lernen*): Relative Häufigkeiten ("Auszählen")
 - Angenommen, nur Sequenzen y_1, \dots, y_T vorhanden (*unüberwachtes Lernen*): Expectation Maximization-Algorithmus (\rightarrow *Statistische Signalverarbeitung und Inferenz*)
- Inferenzalgorithmen (mindestens) linear in $|X|$ – was, wenn $|X|$ sehr groß ist?
 - Approximative Inferenzalgorithmen, z.B. Sampling-basiert (Partikelfilter)

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?
 - Explizite Konstruktion aus Domänenwissen
 - Angenommen, Sequenzen x_1, \dots, x_T und y_1, \dots, y_T vorhanden (sog. *überwachtes Lernen*): Relative Häufigkeiten ("Auszählen")
 - Angenommen, nur Sequenzen y_1, \dots, y_T vorhanden (*unüberwachtes Lernen*): Expectation Maximization-Algorithmus (\rightarrow *Statistische Signalverarbeitung und Inferenz*)
- Inferenzalgorithmen (mindestens) linear in $|X|$ – was, wenn $|X|$ sehr groß ist?
 - Approximative Inferenzalgorithmen, z.B. Sampling-basiert (Partikelfilter)
- Variante für *kontinuierlichen Zustandsraum X* (z.B. 2D-Position im Raum): Kalman-Filter (aber: Normalverteilungsannahmen)

Weitere Aspekte

- Woher kommen die Verteilungen $P(X_t | X_{t-1})$, $P(Y_t | X_t)$ und $P(X_1)$? Ideen?
 - Explizite Konstruktion aus Domänenwissen
 - Angenommen, Sequenzen x_1, \dots, x_T und y_1, \dots, y_T vorhanden (sog. *überwachtes Lernen*): Relative Häufigkeiten ("Auszählen")
 - Angenommen, nur Sequenzen y_1, \dots, y_T vorhanden (*unüberwachtes Lernen*): Expectation Maximization-Algorithmus (\rightarrow *Statistische Signalverarbeitung und Inferenz*)
- Inferenzalgorithmen (mindestens) linear in $|X|$ – was, wenn $|X|$ sehr groß ist?
 - Approximative Inferenzalgorithmen, z.B. Sampling-basiert (Partikelfilter)
- Variante für *kontinuierlichen Zustandsraum* X (z.B. 2D-Position im Raum): Kalman-Filter (aber: Normalverteilungsannahmen)
- Variante für Nicht-atomaren Zustandsraum: Dynamische Bayes'sche Netze ($P(\mathbf{X}_t | y_{1:t})$ wird als Bayes'sches Netz repräsentiert)

HMM Anwendungen

- Verarbeitung natürlicher Sprache, z.B. Part-Of-Speech-Tagging
- Audiodatenverarbeitung: Spracherkennung
- Bioinformatik: Analyse von Proteinsequenzen
- Sensorbasierte Aktivitätserkennung

Zusammenfassung

- Sequentielle Modelle sind ein wichtiger Spezialfall Bayes'scher Netze
- Markov- und Stationaritäts-Annahmen, sodass ein Modell durch
 - Transitionsmodell $P(X_t | X_{t-1})$
 - Observationsmodell $P(Y_t | X_t)$
 - Initialverteilung $P(X_1)$vollständig beschrieben ist
- Wenn X eine endliche Menge ist, nennen wir das entstehende Modell *Hidden Markov Model*
- Das Filtering-, Smoothing- und MAP-Problem kann durch Ausnutzung der Modelleigenschaften effizient gelöst werden

Künstliche Intelligenz

Machine Learning

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Was ist Lernen

Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task (or tasks drawn from the same population) more efficiently and more effectively the next time.

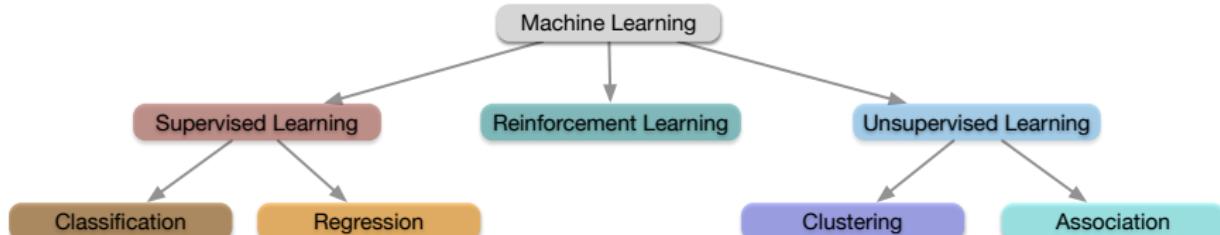
(Herbert Simon)

In vielen Fällen können wir das Verhalten eines Agenten nicht "per Hand" spezifizieren:

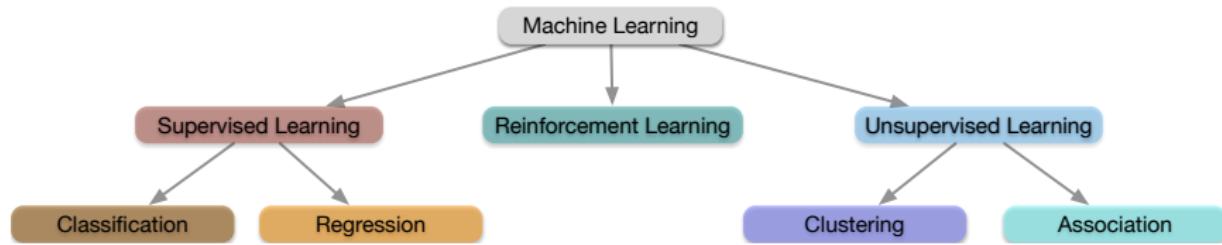
- Problemstruktur zu komplex (natürliche Sprache, Objekterkennung in Videos, ...)
- Formalisierung von Expertenwissen sehr schwierig (*Knowledge Acquisition Bottleneck*)
- Maschine findet überlegene Lösungen

Machine Learning

- **Machine Learning** (maschinelles Lernen) ist der Forschungsbereich, der sich mit dem Design und der Analyse von Algorithmen beschäftigt, die aus Daten lernen und Vorhersagen treffen können
- Machine Learning-Algorithmen lassen sich grob in drei Gruppen unterteilen:
 - supervised learning (überwachtes Lernen)
 - unsupervised learning (unüberwachtes Lernen)
 - reinforcement learning (verstärkendes Lernen)



Supervised Learning

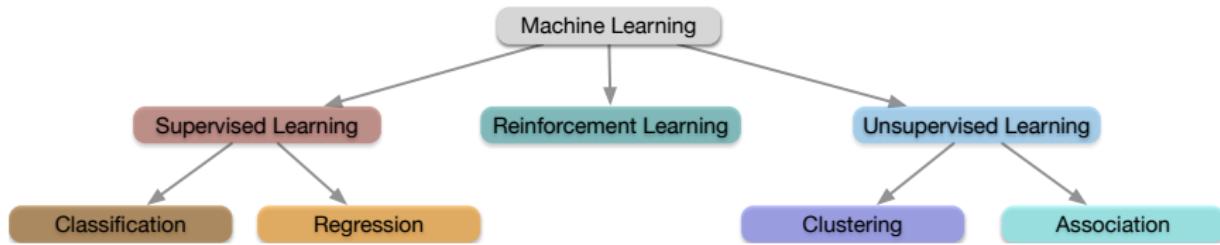


- **Supervised Learning:** Daten bestehen aus Paaren (x, y) von Eingaben $x \in X$ und zugehörigen Ausgaben $y \in Y$, die in einem (unbekannten) Zusammenhang $f(x) = y$ stehen.
- Ziel ist es, eine Funktion \hat{f} mit $\hat{f} \approx f$ zu lernen.
- Kann weiter unterteilt werden in
 - **Klassifikation:** Y ist eine diskrete, endliche Menge (Klassen)
 - **Regression:** Y ist eine unendliche Menge (z.B. reelle Zahlen)

Supervised Learning

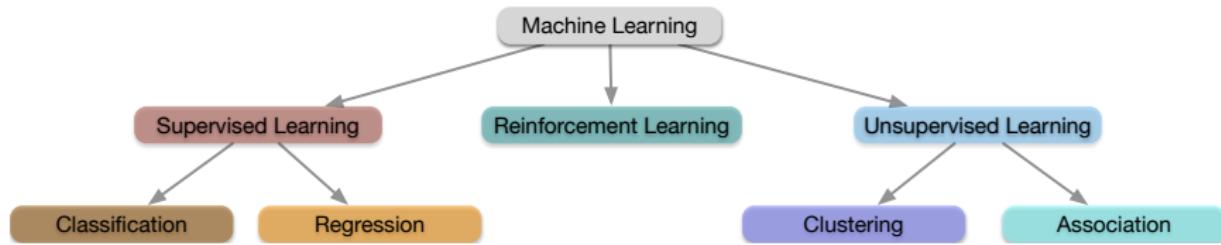
- Aufgabe also: Inferriere eine Funktion \hat{f} aus **gelabelten Trainingsdaten** (Y wird auch als *Label* bezeichnet)
- Die gelernte Funktion \hat{f} kann dann genutzt werden, um für neue Daten x das Label vorherzusagen
- Erfordert, dass der Lernalgorithmus auf geeignete Weise den Zusammenhang von X und Y *generalisiert*
- (Später mehr zur Generalisierbarkeit)

Unsupervised Learning



- **Unsupervised Learning:** Haben nur Eingabe-Daten X
- Ziel ist die zugrundeliegende Struktur der Daten zu lernen, z.B. eine Wahrscheinlichkeitsdichte $p(x)$
- Beispiele:
 - **Dimensionsreduktion:** Lerne eine Abbildung $f(x) = z$, wobei Z geringe Dimensionalität als X hat und *interessante* Eigenschaften in Z erhalten bleiben (z.B. Ähnlichkeit)
 - **Clustering:** Teile die Daten so in k Gruppen (Cluster) auf, sodass die Daten innerhalb einer Gruppe ähnlicher zueinander sind als die Daten verschiedener Cluster

Reinforcement Learning



- **Reinforcement Learning:** Welche Aktionen muss ein Agent ausführen, um seinen Reward zu maximieren?
- Konkret: Agent bekommt Eingaben x (Beobachtungen), muss Aktion wählen, bekommt *Reward*

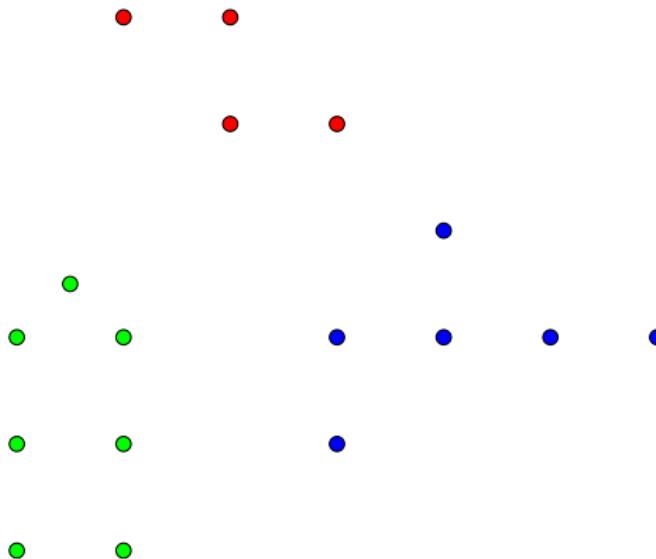
Weiteres Vorgehen

- Beispiel für Clustering-Algorithmus: K-Means
- Beispiel für Klassifikations-Algorithmus: Entscheidungsbaum
- Beispiel für Regressions-Algorithmus: Künstliche Neuronale Netze
- Evaluation von Klassifikatoren, Generalisierbarkeit

Clustering

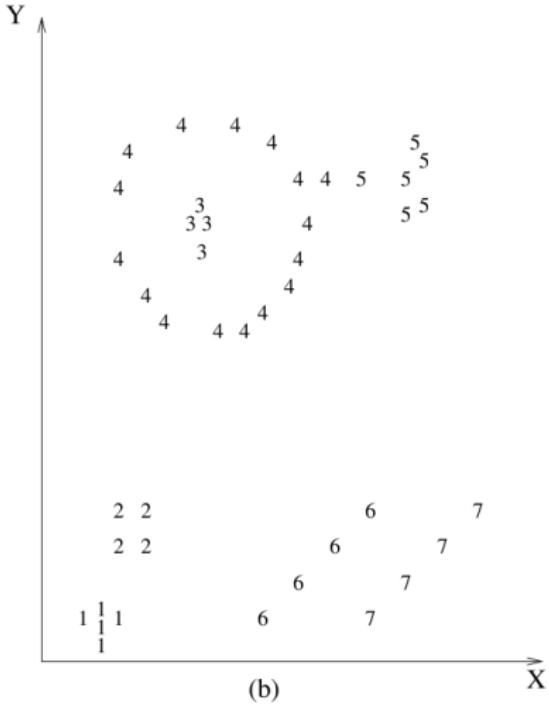
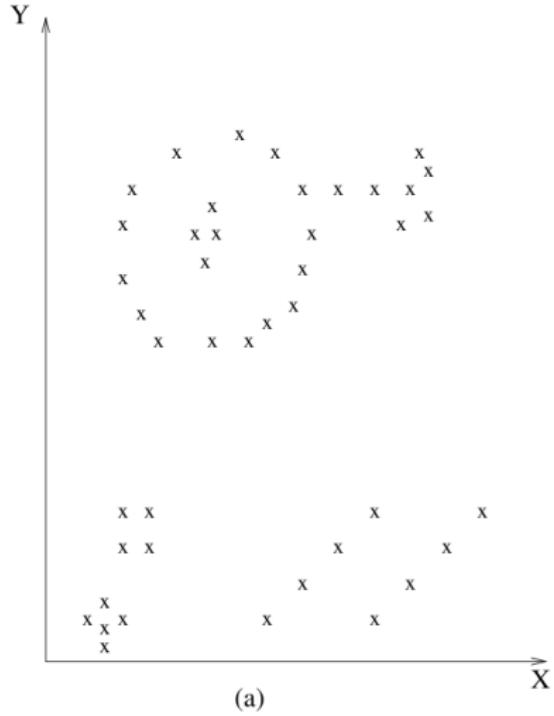
- Ziel: Aufteilung der Daten O in Teilmengen (Cluster) C_1, \dots, C_k , sodass
 - Zwischen Objekten innerhalb eines Clusters möglichst große Ähnlichkeit besteht
 - Zwischen Objekten in verschiedenen Clustern möglichst geringe Ähnlichkeit besteht
- Verschiedene Aufteilungsstrategien: Disjunkte Teilmengen, Hierarchische Cluster, probabilistische Aufteilung (Wahrscheinlichkeit $p(o|C_i)$, dass Sample o zu Cluster C_i gehört)

Clustering: Beispiel (hier im \mathbb{R}^2)



→ disjunkte Aufteilung in 3 Klassen

Beispiel für nicht-konvexe Cluster



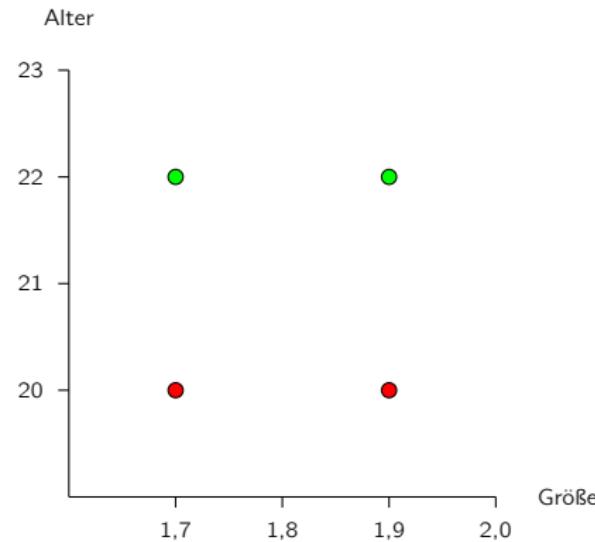
Ähnlichkeitsmaß

- zentrale Bedeutung (neben dem verwendeten Algorithmus)
- z. B. Minkowski Distanz (im \mathbb{R}^n):

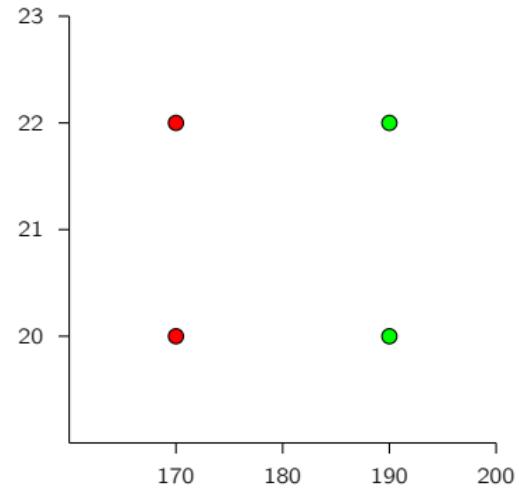
$$d_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}} = \|\mathbf{x} - \mathbf{y}\|_p$$

- für $p = 1$: Manhattan Distanz
- für $p = 2$: Euklidische Distanz
- Probleme:
 - implizite Annahme, dass in jeder Dimension gleiche Skalierung
 - Behandlung von nominalen Daten (z. B. Wochentage, Spielertyp): Ordnung?, Subtraktion?
 - Berücksichtigung von Domänenwissen

Einfluss der Skalierung auf die Clusterbildung



ungünstige Skalierung
(x-Achse stauchen)



günstige Skalierung
(x-Achse dehnen)

→ Abhilfe: Gewichtung, z. B. durch Normalisierung

k-means Algorithmus

Charakteristika:

- Parameter $k \in \mathbb{N}$ bestimmt die Anzahl der Cluster
- jeder Cluster C_i wird durch Zentroid $\mathbf{c}_i \in \mathbb{R}^n$ repräsentiert
→ Mittelwert bezüglich aller in C_i enthaltenen Punkte, d. h.

$$\mathbf{c}_i = \left(\frac{1}{|C_i|} \sum_{\mathbf{p} \in C_i} p_1, \dots, \frac{1}{|C_i|} \sum_{\mathbf{p} \in C_i} p_n \right)$$

- *Ziel*: wähle Cluster $C_1, \dots, C_k \subseteq O$ so, dass $\{C_1, \dots, C_k\}$ eine Partition von O ist und

$$E(C_1, \dots, C_k) = \sum_{i=1}^k \sum_{\mathbf{p} \in C_i} |\mathbf{p} - \mathbf{c}_i|^2$$

(intra-cluster Varianz) minimiert wird

k-means Algorithmus

- ① wähle k zufällige Punkte $\mathbf{c}_1, \dots, \mathbf{c}_k \in \mathbb{R}^n$ als Zentroide
- ② $\forall \mathbf{o} \in O$: ordne \mathbf{o} dem nächsten Zentroid zu, d. h. \mathbf{o} wird \mathbf{c}_i zugeordnet, falls

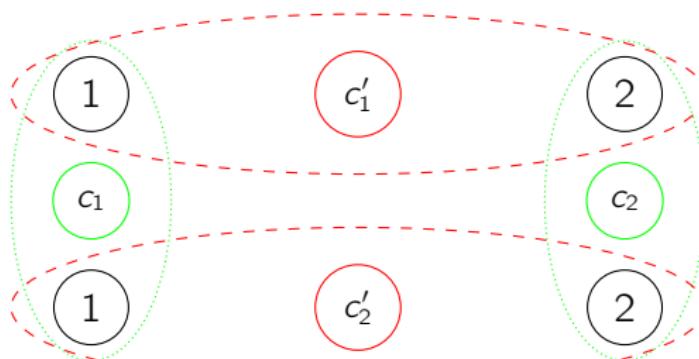
$$d(\mathbf{o}, \mathbf{c}_i) = \min_{1 \leq i \leq k} d(\mathbf{o}, \mathbf{c}_i),$$

wobei $d(,)$ eine Distanzfunktion ist

- ③ Sei C_i die Menge der Objekte, die \mathbf{c}_i zugeordnet wurden.
Berechne ausgehend von C_i den Zentroid \mathbf{c}_i neu.
- ④ falls sich im vorherigen Schritt mindestens ein Zentroid geändert hat, gehe zu 2
andernfalls: Stop; C_1, \dots, C_k ist eine Partitionierung von O

Bemerkungen zum k -means Algorithmus

Beobachtung: resultierende Cluster können stark von der Wahl der initialen Zentroide abhängen

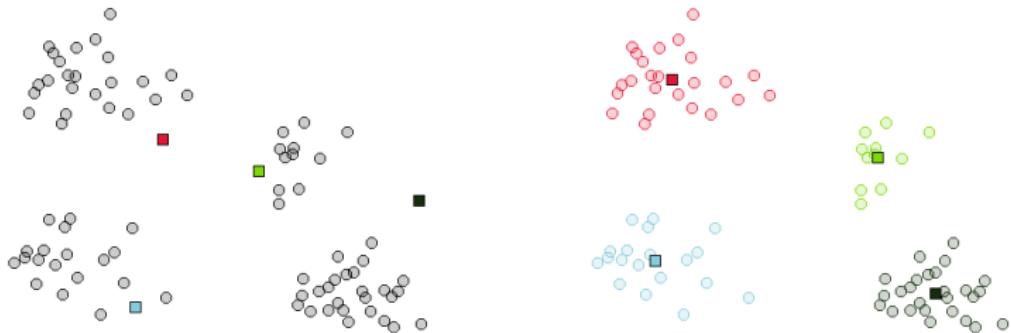


Bemerkungen zum k -means Algorithmus

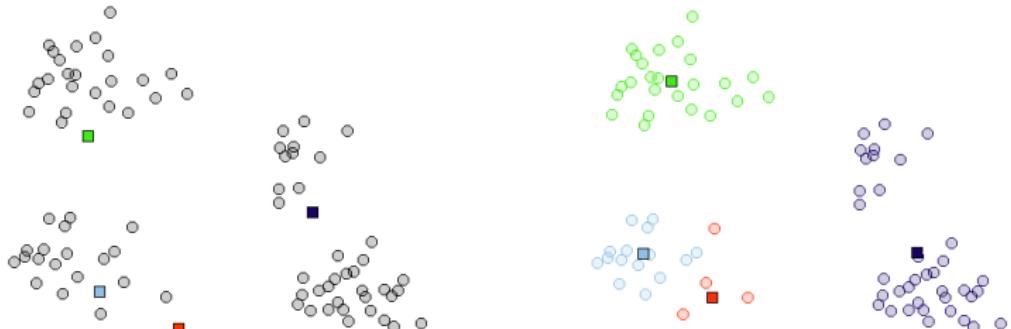
- resultierendes Clustering kann stark von Wahl der initialen Zentroide abhängen → keine Garantie dafür, dass globales Minimum bezüglich der Zielfunktion $E(C_1, \dots, C_k)$ gefunden wird
- Abhilfe: mehrere Durchläufe mit unterschiedlichen Startkonfigurationen → wähle Resultat mit minimalen Wert für E
- genau genommen: resultierende Anzahl an Clustern kann auch kleiner k sein (Beispiel?)
- initiale Zentroide sollten möglichst weit voneinander entfernt sein

Bemerkungen zum k -means Algorithmus

gutes Resultat:



schlechtes Resultat:



Clustering: Weitere Aspekte

- Wahl von k : Größeres k sorgt immer für geringeres E , aber Clustering ist ja trotzdem nicht immer “besser”
→ Methoden zur Beurteilung, die unabhängig von k sind (werden wir hier nicht weiter behandeln)
- Nachteil von K-Means: Nimmt implizit an, dass Cluster sphärisch sind
→ Methoden für Cluster beliebiger Form, z.B. DBSCAN (hier auch nicht weiter behandelt)

Klassifikation: Entscheidungsbäume

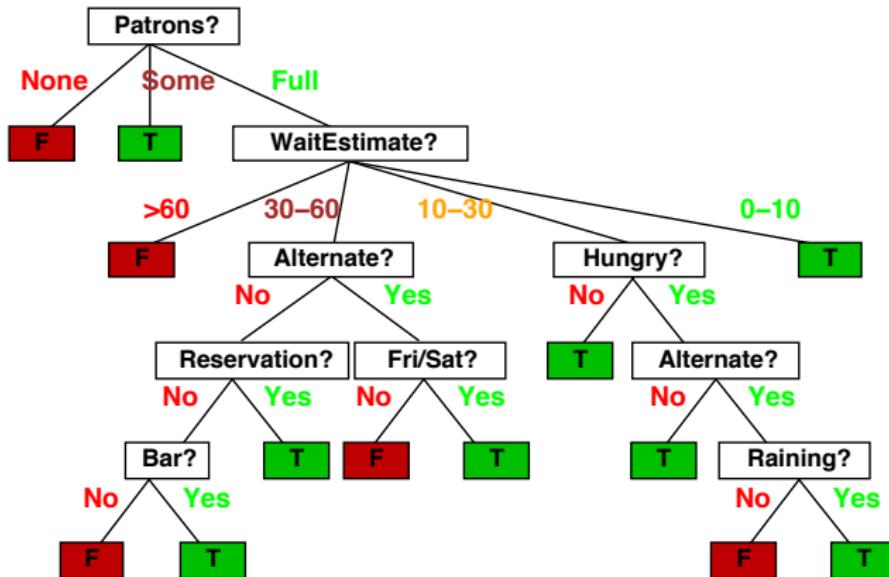
Klassifikation: Entscheidungsbäume

- Wir stehen vor einem Restaurant. Werden wir warten, bis wir eingelassen werden?
- Eingabe X ist Vektor aus mehreren *Attributen*: Alternatives Restaurant (Alt); hat das Restaurant eine Bar (Bar); ist es Freitag oder Samstag (Fri); sind wir hungrig (Hun), wie voll ist es (Pat), wie teuer (Price), regnet es (Rain), haben wir reserviert (Res), Art des Restaurants (Type), wie lange werden wir warten müssen (Est).

Example	Attributes										Target WillWait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Entscheidungsbäume

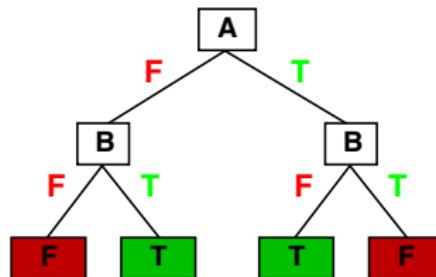
- Idee: Repräsentiere Funktion $\hat{f}(x) = y$ als Baum
- Achtung: Der Name *Entscheidungsbaum* ist etwas täuschend: Die Ausgaben sind *Klassen*, nicht unbedingt Entscheidungen



Ausdrucksstärke

Entscheidungsbäume können jede beliebige Funktion der Eingabe-Attribute repräsentieren:

A	B	$A \text{ xor } B$
F	F	F
F	T	T
T	F	T
T	T	F



- D.h. es gibt für Trainingsmenge einen Entscheidungsbau, aber dieser wird nicht unbedingt gut für neue Daten funktionieren
- Ziel: Für gegebene Trainingsdaten einen kompakten Entscheidungsbaum konstruieren

Entscheidungsbaum-Lernen

Ziel: Finde einen kompakten Baum, der mit den Trainingsdaten konsistent ist:

Idee: wähle rekursiv das “relevanteste” Attribut als Wurzel von neuem Teilbaum:

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
    best ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test best
    for each value vi of best do
      examplesi ← {elements of examples with best = vi}
      subtree ← DTL(examplesi, attributes - best, MODE(examples))
      add a branch to tree with label vi and subtree subtree
  return tree
```

2 Aufbau von Bäumen: Prinzip

Überlegung: Betrachte eine aktuelle Teilmenge $\mathcal{A} \subseteq \mathcal{D}$ der Testdaten.

- Wenn alle Instanzen in \mathcal{A} derselben Klasse $\omega_{\mathcal{A}}$ angehören, ist die Teilmenge rein. Dann kann ein Blattknoten angelegt werden, der der Klasse $\omega_{\mathcal{A}}$ zugeordnet wird.
- Wenn die Instanzen in \mathcal{A} unterschiedlichen Klassen angehören, muss eine Entscheidung getroffen werden:
 - Trotzdem einen Blattknoten anlegen und Klassifikation gemäß der Mehrheit.
 - Einen inneren Knoten anlegen, und \mathcal{A} in mehrere Teilmengen \mathcal{A}_j aufspalten. Für die Teilmengen \mathcal{A}_j wird dann das Verfahren rekursiv aufgerufen; die entstehenden Teilbäume werden dann über geeignet bezeichnete Äste mit dem neuen inneren Knoten verbunden.

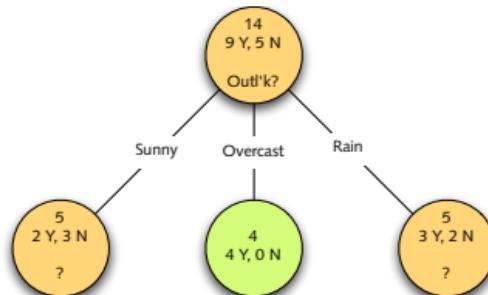
2 Beispiel

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
3	O	H	H	W	Y
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
7	O	C	N	S	Y
8	S	M	H	W	N
9	S	C	N	W	Y
10	R	M	N	W	Y
11	S	M	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y
14	R	M	H	S	N



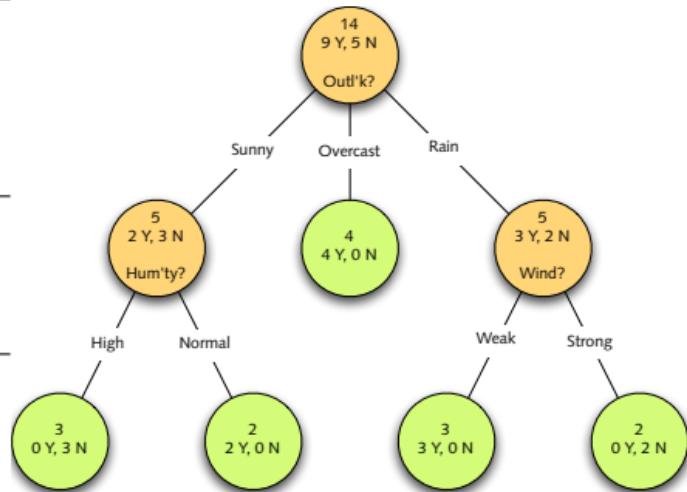
2 Beispiel

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
8	S	M	H	W	N
9	S	C	N	W	Y
11	S	M	N	S	Y
3	O	H	H	W	Y
7	O	C	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
10	R	M	N	W	Y
14	R	M	H	S	N



2 Beispiel

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
8	S	M	H	W	N
9	S	C	N	W	Y
11	S	M	N	S	Y
3	O	H	H	W	Y
7	O	C	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
10	R	M	N	W	Y
14	R	M	H	S	N



2 Synthese von Bäumen

Beim Aufbau von Bäumen sind folgende Fragen zu klären:

- Sollen an inneren Knoten lediglich binäre Eigenschaften getestet werden oder sind auch komplexere n -äre Spaltungen erlaubt?
- Auf Basis welcher Kriterien sollte eine Eigenschaft für einen Knotentest ausgewählt werden?
- Unter welchen Umständen sollte ein Knoten als Blatt deklariert werden?
- Wenn ein Baum „zu groß“ wird – wie kann man ihn verkleinern und vereinfachen (also „beschneiden“, „zurückstutzen“?)
- Falls ein Blattknoten gemischt ist – wie sollte die Klassenbezeichnung zugewiesen werden?
- Wie sollte man mit fehlenden Merkmalen umgehen?

2 Anzahl der Verzweigungen

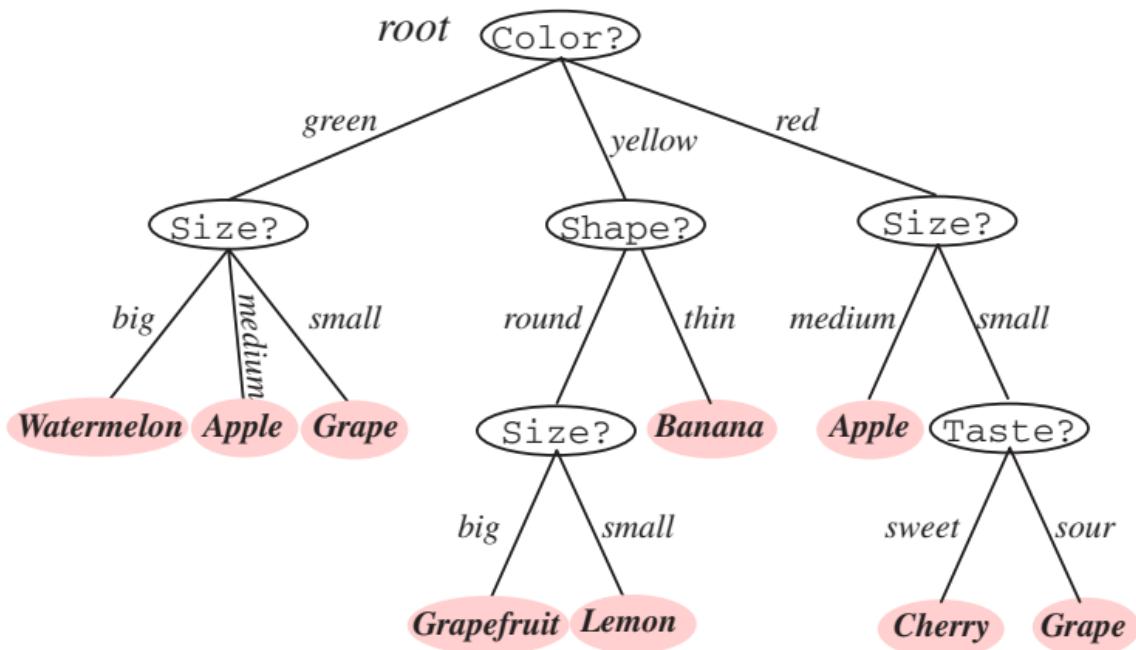
Wählt man eine Eigenschaft mit n Ausprägungen um einen Verzweigungsknoten anzulegen, könnte man eine n -Wege Verzweigung einrichten.

Sollte man das auch?

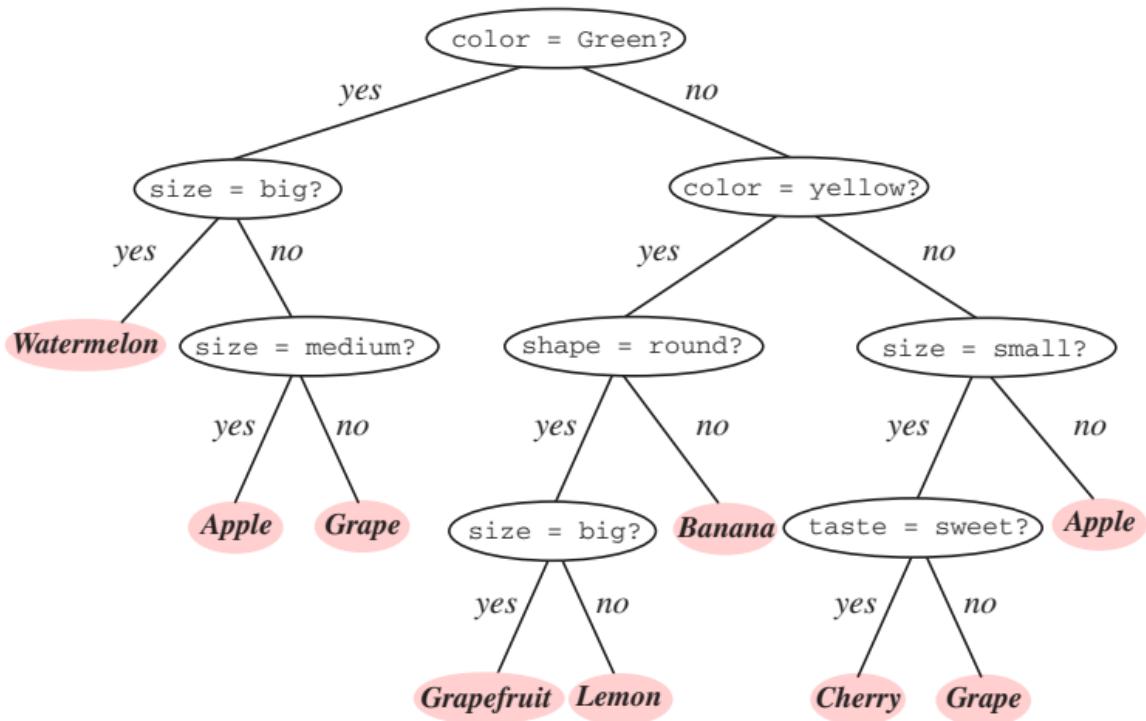
Es gibt Fälle, in denen dies hilfreich ist. Im allgemeinen werden dadurch die Daten aber zu stark fragmentiert, so dass auf der nächsten Baumebene eine sinnvolle Zerlegung behindert wird.

Da zudem jede n -Wege-Zerlegung durch mehrere binäre Zerlegungen repräsentiert werden kann, wird dieser Ansatz oft bevorzugt (vgl. HASTIE et. al.)

2 Früchte eines binären Baums



2 Früchte eines binären Baums



3 Reinheit von Knoten

Grundlegendes Prinzip der Merkmalsauswahl für die Knotenzerlegung ist, möglichst einfache Bäume zu erzeugen, die flach sind und wenige Knoten beinhalten.

(Dies ist eine Variante von OCKHAMS Rasiermesser – ein Prinzip das empfiehlt, stets die einfachste Hypothese zu wählen, die die vorliegenden Daten erklärt. In Abschnitt 9 mehr dazu.)

Um dies zu erreichen, wählen wir an jedem Knoten N diejenige Eigenschaft T , welche die „Reinheit“ der Kinderknoten maximiert.

Es zeigt sich, dass „Unreinheit“ (*impurity*) eines Knotens – intuitiv, die Durchmischtheit der dem Knoten zugeordneten Daten in Bezug auf ihre Klassenzugehörigkeit – einfacher definierbar ist als „Reinheit“.

Die Unreinheit eines Knotens N bezeichnen wir mit $i(N)$.

3 Reinheitsmaße (I)

Für die Definition von $i(N)$ gibt es mehrere plausible Ansätze.

Grundsätzlich soll gelten $i(N) = 0$, falls der Knoten rein ist (d.h., nur Instanzen derselben Klasse enthält). Außerdem soll gelten, dass $i(N)$ um so größer ist, je stärker ein Knoten durchmischt ist.

Ein Ansatz, „Unreinheit“ zu quantifizieren ist, die Information zu betrachten, die wir brauchen, wenn wir zu einer Instanz $\mathbf{x} \in N$ die zugehörige Klasse bestimmen müssen.

Falls der Knoten rein ist, wissen wir, dass alle Instanzen das gleiche Klassenlabel tragen. Zur Bestimmung des Klassenlabels für \mathbf{x} brauchen wir dann keine weiteren Informationen. Also gilt $i(N) = 0$.

Wie viel Information brauchen wir, falls N ein gemischter Knoten ist, in dem die einzelnen Klassen i mit relativen Häufigkeiten p_i enthalten sind?

3 Reinheitsmaße (I)

Wie viel Information brauchen wir, falls N ein gemischter Knoten ist, in dem die einzelnen Klassen i mit relativen Häufigkeiten p_i enthalten sind, und man uns sagt, dass $\mathbf{x} \in N$ zur Klasse j gehört?

Das sind $\log_2 \frac{1}{p_j} = -\log_2 p_j$ Bits.

Wiederholung: gegeben seien 8 Klassen mit gleicher Wahrscheinlichkeit $1/8$. Wie viele Bits brauchen wir dann, um die Information über die Klasse c zu codieren? Da alle Klassen gleich wahrscheinlich sind, sollten alle Codeworte die gleiche Länge haben. Mit einem Code-Wort von $\log_2 8 = 3$ Bit kann man dann genau 8 unterschiedliche Klassen codieren.

Insgesamt ergibt sich damit der Erwartungswert der Information zu:

$$i(N) = - \sum_{i=1}^c p_i \log_2 p_i$$

3 Reinheitsmaße (I)

Ein mögliches Maß für die Unreinheit eines Knotens ist damit seine Informationsentropie

$$i_{ent}(N) = - \sum_{i=1}^c p_i \log_2 p_i$$

Basis für die Auswahl einer Zerlegung ist dann die Suche nach einer Eigenschaft T , die zwei Kindknoten N_L, N_R erzeugt, die die Unreinheit soweit wie möglich minimiert.

Im Fall der Informationsentropie heißt das: eine Zerlegung, die so wenig wie möglich zusätzliche Information erforderlich macht, um die Klasse einer Instanz zu bestimmen, die im Teilbaum von N liegt. (D. h., ein Test T , der möglichst viel Information liefert.)

3 Reinheitsgewinn

Nehmen wir an, wir finden eine Zerlegung, bei der ein Anteil von P_L Instanzen im linken Kindknoten N_L landet und ein Anteil von $(1 - P_L)$ Instanzen im rechten Kindknoten N_R .

Dann gilt für den Verlust an Unreinheit (Reinheitsgewinn, *Gain*)

$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R)$$

Dieses Maß muss dann durch eine sinnvolle Wahl der Eigenschaft T , mit deren Hilfe die Zerlegung durchgeführt wird, maximiert werden. (Falls wir die Informationsentropie als Maß der Unreinheit betrachten, repräsentiert $\Delta i(N)$ den *Gewinn* an Informationen, in Bits. Eine binäre Entscheidung kann maximal 1 Bit an Information liefern).

Im folgenden Beispiel nehmen wir an, Eigenschaften hätten die einfache Form $m = v$ für ein Merkmal m und ein Merkmalswert v .

3 Beispiel

Nr	Outlook	Temperature	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$\begin{aligned} i(Root) &= -\left(\frac{9}{14} \log_2 \frac{9}{14} + \frac{5}{14} \log_2 \frac{5}{14}\right) \\ &= 0.940286 \end{aligned}$$

3 Beispiel

Split on *Outlook* = *Overcast*

Outlook = *Overcast*

Nr	O	T	H	W	P
3	O	H	H	W	Y
7	O	C	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y

Outlook ≠ *Overcast*

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
8	S	M	H	W	N
9	S	C	N	W	Y
10	R	M	N	W	Y
11	S	M	N	S	Y
14	R	M	H	S	N

$$i(N_L) = -(0 \log_2 0 + 1 \log_2 1) = 0$$

$$i(N_R) = -\left(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}\right) = 1$$

$$\Delta i(N) = 0.940286 - \frac{4}{14}0 - \frac{10}{14}1 = 0.226000244$$

3 Beispiel

Split on *Outlook* = Rain

Outlook = Rain

Nr	O	T	H	W	P
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
10	R	M	N	W	Y
14	R	M	H	S	N

Outlook ≠ Rain

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
3	O	H	H	W	Y
7	O	C	N	S	Y
8	S	M	H	W	N
9	S	C	N	W	Y
11	S	M	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y

$$i(N_L) = -\left(\frac{3}{5} \log_2 \frac{3}{5} + \frac{2}{5} \log_2 \frac{2}{5}\right) = 0.971$$

$$i(N_R) = -\left(\frac{6}{9} \log_2 \frac{6}{9} + \frac{3}{9} \log_2 \frac{3}{9}\right) = 0.918$$

$$\Delta i(N) = 0.94 - \frac{5}{14}0.971 - \frac{9}{14}0.918 = 0.003$$

3 Beispiel

Split on *Outlook* = *Sunny*

Outlook = *Sunny*

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
8	S	M	H	W	N
9	S	C	N	W	Y
11	S	M	N	S	Y

Outlook ≠ *Sunny*

Nr	O	T	H	W	P
3	O	H	H	W	Y
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
7	O	C	N	S	Y
10	R	M	N	W	Y
12	O	M	H	S	Y
13	O	H	N	W	Y
14	R	M	H	S	N

$$i(N_L) = -\left(\frac{2}{5} \log_2 \frac{2}{5} + \frac{3}{5} \log_2 \frac{3}{5}\right) = 0.971$$

$$i(N_R) = -\left(\frac{7}{9} \log_2 \frac{7}{9} + \frac{2}{9} \log_2 \frac{2}{9}\right) = 0.764$$

$$\Delta i(N) = 0.94 - \frac{5}{14}0.971 - \frac{9}{14}0.764 = 0.102$$

3 Beispiel

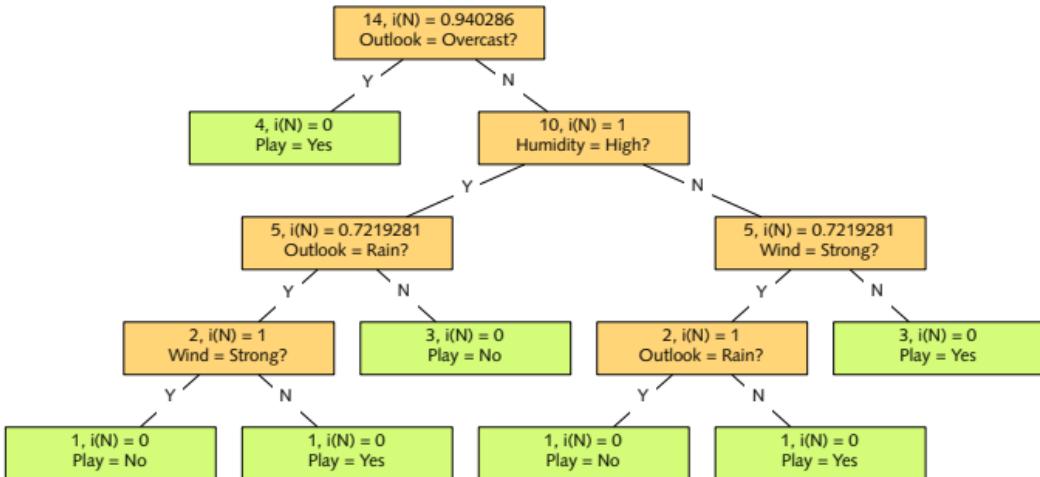
Insgesamt ergibt sich:

Var	Val	$\Delta i(N)$
Outlook	Overcast	0.226000244
Outlook	Rain	0.003184853
Outlook	Sunny	0.102243564
Temperature	Cool	0.014956070
Temperature	Hot	0.025078174
Temperature	Mild	0.001339742
Humidity	High	0.151835501
Humidity	Normal	0.151835501
Wind	Strong	0.048127030
Wind	Weak	0.048127030

Daraus folgt: der beste erste Split ist *Outlook = Overcast*

3 Beispiel

Wenn man auf diese Weise den Baum aufbaut, erhält man schließlich:



Warum sieht dieser Baum anders aus, als der Ausgangsbaum?

(1) Wir verwenden nur binäre Splits

(2) Die Split-Operation ist eine *lokale* Optimierung

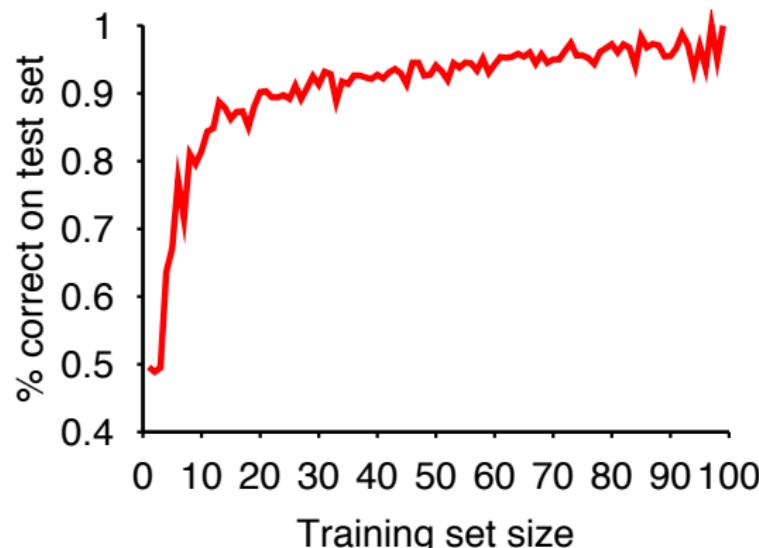
Evaluation

Evaluation

Woher wissen wir, dass $\hat{f} \approx f$?

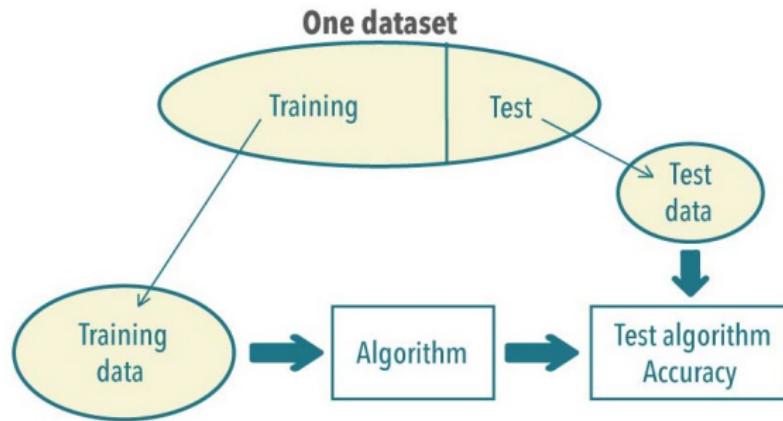
- 1) Verwende Konzepte aus der Statistik, z.B. Likelihood der Daten, R^2 , ...
- 2) Wende \hat{f} auf neuen Daten an (Testdaten)

Lernkurve = % Korrekt klassifizierter Daten in Abhangigkeit von Menge der Trainingsdaten



Trainings- vs. Testdaten

Training data vs. test data



- Lernen: Bestimme Struktur/Parameter des Modells mit Trainingsdaten
- Evaluiere Modellgüte auf Testdaten
- Trainingsdaten \neq Testdaten

Performance-Messung

Lernkurve hängt ab von

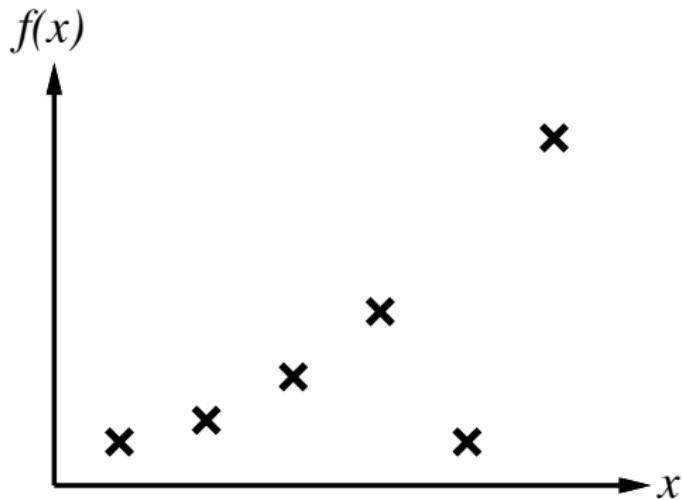
- Realisierbarkeit der tatsächlichen Funktion im Modell (Ausdrucksstärke), z.B. quadratische Abhängigkeit in linearer Regression?
- Redundanz, Rauschen in den Attributwerten – schwierig, relevante Zusammenhänge zu extrahieren

Overfitting

- Redundanz, Rauschen in den Daten + ausdrucksstarke Modelle können zu *Überanpassung* (Overfitting) führen
- D.h. Fehler auf den Trainingsdaten nimmt immer weiter ab, aber nimmt auf Testdaten wieder zu
- Problem: Modell berücksichtigt auch zufälliges Rauschen für Vorhersage, "Daten auswendig gelernt"

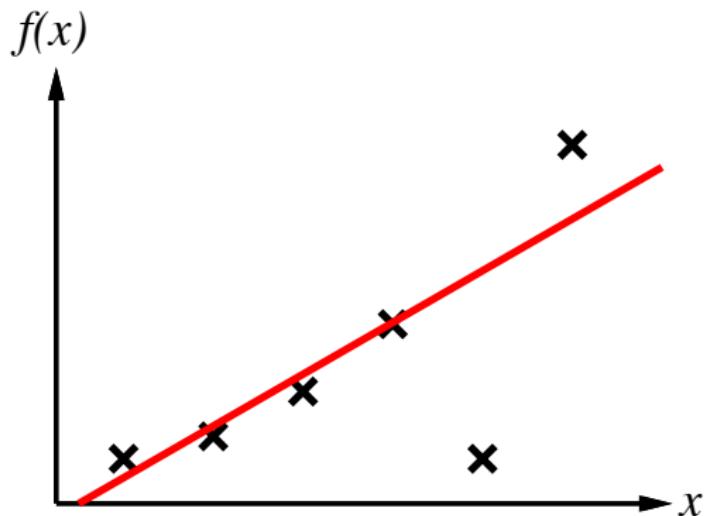
Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



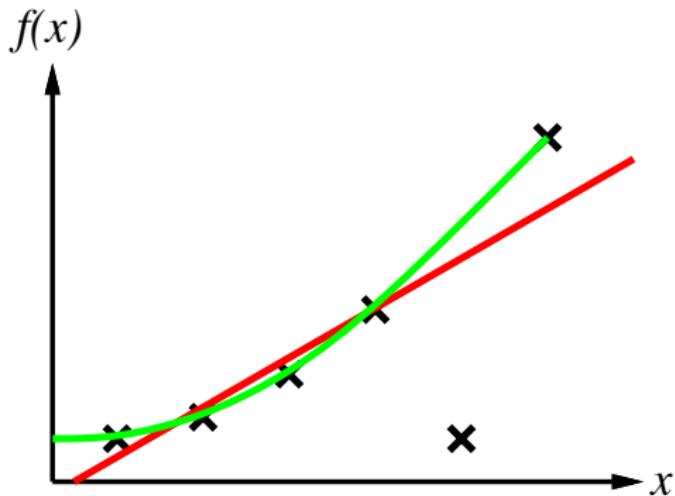
Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



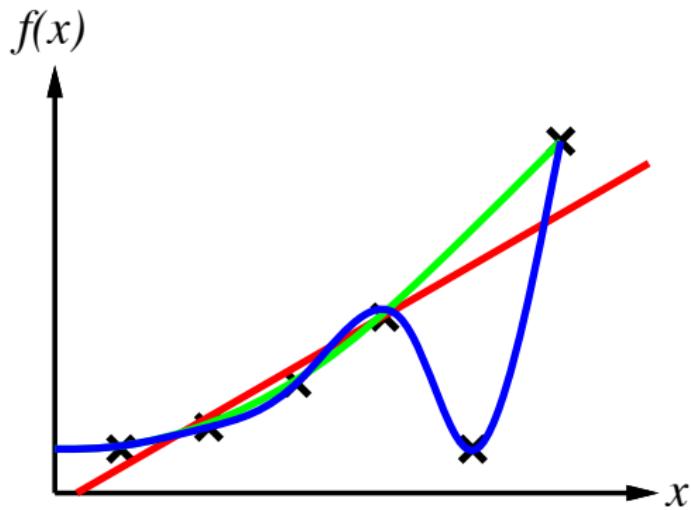
Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



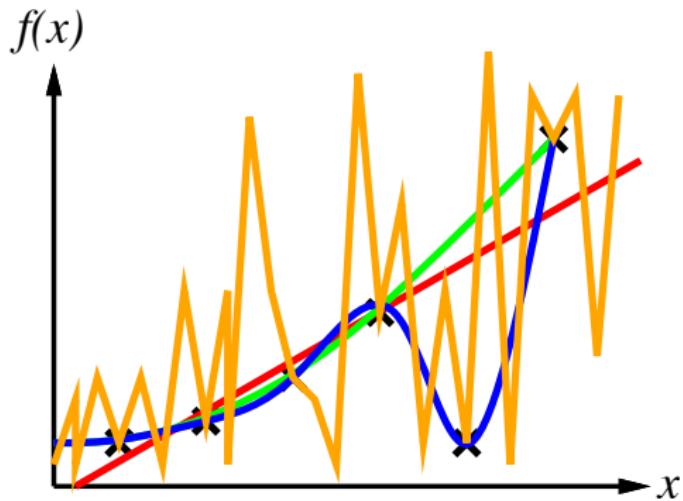
Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



Evaluation von Klassifikations-Algorithmen

- Im Folgenden schauen wir uns die Evaluation von *Klassifikations-Algorithmen* noch etwas genauer an
 - Für Regressions-Algorithmen berechnet man z.B. Wurzel aus mittlerem quadartischen Fehler als Gütekriterium, darauf gehen wir aber hier nicht weiter ein
- Abgesehen von dem Anteil der korrekt klassifizierten Test-Samples (*Accuracy, Genauigkeit*) gibt es noch weitere wichtige Performance-Maße:
 - Confusion Matrix (Wahrheitsmatrix)
 - Precision und Recall
 - Sensitivität und Spezifität
 - F1-Score

Confusion Matrix

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22

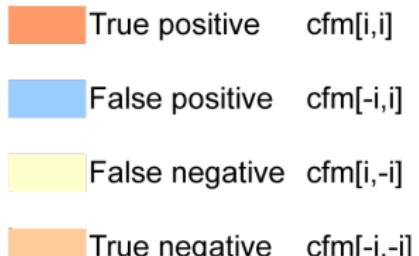
	True positive	$cfm[i,i]$
	False positive	$cfm[-i,i]$
	False negative	$cfm[i,-i]$
	True negative	$cfm[-i,-i]$

- Trage für alle Testdaten wahre Klasse und vorhergesagte Klasse gegeneinander auf

Accuracy

- Accuracy: Anteil korrekt klassifizierter Testdaten

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22



The legend defines four types of entries in the confusion matrix:

- True positive (cfm[i,i]): Orange square
- False positive (cfm[-i,i]): Blue square
- False negative (cfm[i,-i]): Yellow square
- True negative (cfm[-i,-i]): Light orange square

$$\text{Accuracy} = \frac{\sum tp + \sum tn}{\sum tp + \sum tn + \sum fp + \sum fn}$$

Precision

- Anteil der tatsächlich positiven Samples unter allen als positiv erkannten Samples

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	0	62	300	0
	8	0	0	0	0	0	0	0	0	0	4
	9	0	0	0	0	2	2	19	59	4	22

	True positive	$cfm[i,i]$
	False positive	$cfm[-i,i]$
	False negative	$cfm[i,-i]$
	True negative	$cfm[-i,-i]$

$$Precision = \frac{\sum tp}{\sum tp + \sum fp}$$

Recall (= Sensitivität)

- Anteil der als positiv erkannten Samples aus allen tatsächlich positiven Samples

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	0	62	300	0
	8	0	0	0	0	0	0	0	0	0	4
	9	0	0	0	0	2	2	19	59	4	22

	True positive	$cfm[i,i]$
	False positive	$cfm[-i,i]$
	False negative	$cfm[i,-i]$
	True negative	$cfm[-i,-i]$

$$Recall = \frac{\sum tp}{\sum tp + \sum fn}$$

Spezifität

Anteil der als negativ erkannten Samples aus allen negativen Samples

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22

True positive $cfm[i,i]$

False positive $cfm[-i,i]$

False negative $cfm[i,-i]$

True negative $cfm[-i,-i]$

$$Specificity = \frac{\sum tn}{\sum tn + \sum fp}$$

F1-Score

- Harmonisches Mittel aus Precision und Recall

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22

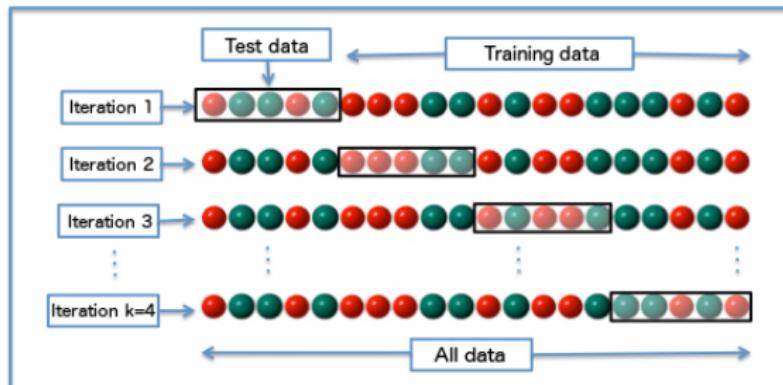
Legend:

- True positive: $\text{cfm}[i,i]$
- False positive: $\text{cfm}[-i,i]$
- False negative: $\text{cfm}[i,-i]$
- True negative: $\text{cfm}[-i,-i]$

$$F_1 = \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}$$

Kreuzvalidierung

- Problem bei Aufteilung in Trainings- und Testdaten:
 - Kann nicht alle Daten zum Trainieren verwenden
 - Ergebnis hängt von gewählten Testdaten ab
- Kreuzvalidierung umgeht diese Probleme, indem systematisch verschiedene Train-/Test-Splits evaluiert werden
 - Teile die Daten in N Teilmengen X_1, \dots, X_N auf
 - Für $i \in \{1, \dots, N\}$
 - Trainiere Modell mit $X \setminus X_i$
 - Evaluiere Modell mit X_i
 - Berechne Mittelwert der Güte der N Durchläufe



Zusammenfassung

- Lernen ist wichtig in nicht vollständig bekannten Umgebungen
- Unüberwachtes Lernen versucht, Muster oder Zusammenhänge in Daten zu erkennen
- Überwachtes Lernen versucht, eine Funktion $f(x) = y$ aus Eingabe-/Ausgabe-Paaren (x, y) (Trainingsdaten) zu rekonstruieren
- Ein Entscheidungsbaum ist ein einfacher, interpretierbarer Klassifikationsalgorithmus
- Künstliche Neuronale Netze sind sehr ausdrucksstarke Regressions- und Klassifikationsalgorithmen, die in vielen Fällen eine hohe Güte erreichen
- Die Güte eines gelernten Modells wird auf separaten Testdaten evaluiert

Künstliche Intelligenz

Perceptron

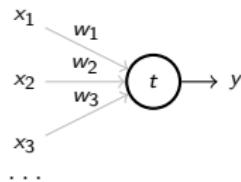
Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Dynamics of a Simple Perceptron (Artificial Threshold Neuron)

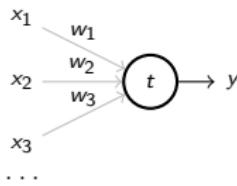
- ▶ General schema and dynamics:



$$y = \begin{cases} 1 & \text{if } \sum_i x_i \cdot w_i \geq t \\ 0 & \text{otherwise} \end{cases}$$
$$y = \Theta(\vec{x} \cdot \vec{w} - t)$$

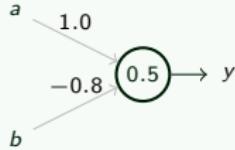
Dynamics of a Simple Perceptron (Artificial Threshold Neuron)

- ▶ General schema and dynamics:



$$y = \begin{cases} 1 & \text{if } \sum_i x_i \cdot w_i \geq t \\ 0 & \text{otherwise} \end{cases}$$
$$y = \Theta(\vec{x} \cdot \vec{w} - t)$$

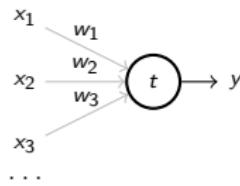
- ▶ A small two-input example:



a	b	y
0	0	0
0	1	0
1	0	1
1	1	0

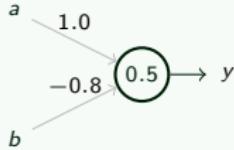
Dynamics of a Simple Perceptron (Artificial Threshold Neuron)

- ▶ General schema and dynamics:



$$y = \begin{cases} 1 & \text{if } \sum_i x_i \cdot w_i \geq t \\ 0 & \text{otherwise} \end{cases}$$
$$y = \Theta(\vec{x} \cdot \vec{w} - t)$$

- ▶ A small two-input example:

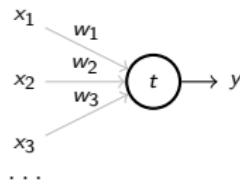


a	b	y
0	0	0
0	1	0
1	0	1
1	1	0

$$y = a \wedge \neg b$$

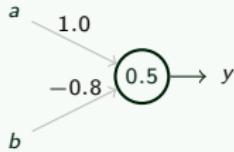
Dynamics of a Simple Perceptron (Artificial Threshold Neuron)

- ▶ General schema and dynamics:



$$y = \begin{cases} 1 & \text{if } \sum_i x_i \cdot w_i \geq t \\ 0 & \text{otherwise} \end{cases}$$
$$y = \Theta(\vec{x} \cdot \vec{w} - t)$$

- ▶ A small two-input example:



a	b	y
0	0	0
0	1	0
1	0	1
1	1	0

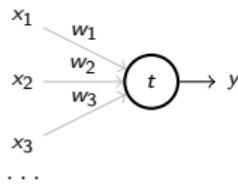
$$y = a \wedge \neg b$$

a	b	y
0.2	0.0	0
0.0	0.2	0
0.5	0.0	1
5.0	1.0	1

$$?$$

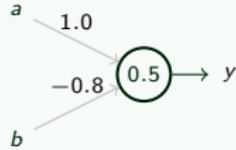
Dynamics of a Simple Perceptron (Artificial Threshold Neuron)

- ▶ General schema and dynamics:



$$y = \begin{cases} 1 & \text{if } \sum_i x_i \cdot w_i \geq t \\ 0 & \text{otherwise} \end{cases}$$
$$y = \Theta(\vec{x} \cdot \vec{w} - t)$$

- ▶ A small two-input example:



a	b	y
0	0	0
0	1	0
1	0	1
1	1	0

$$y = a \wedge \neg b$$

a	b	y
0.2	0.0	0
0.0	0.2	0
0.5	0.0	1
5.0	1.0	1

?

Take-away-messages of section: “*Structure and Dynamics of Perceptrons*”



You should now be able to ...

- ▶ explain what a simple perceptron is
- ▶ describe the dynamics of a single threshold unit
- ▶ name one of the first existing hardware-implementation of a perceptron
- ▶ describe the structure, dynamics and adaptation process of the Mark 1 perceptron

Learning objective of section: “*Training Perceptrons*”

In this section we will ...

- ▶ discuss the idea behind training a perceptron based on data
- ▶ discuss the error function generated by artificial neural networks in general and for threshold units in particular
- ▶ introduce a corresponding adaptation schema - the delta rule
- ▶ discuss the idea of a hypothesis space
- ▶ discuss limitations of single layer perceptrons

Training a Single Layer Perceptron

Training a Perceptron

Result: A trained perceptron

Input: A perceptron P , a set of training data D

- 1 Let π_P be the set of parameters of P :
weights and thresholds
- 2 Initialise all parameters π_P , randomly
- 3 **repeat**
 - 4 Compute error E wrt. D and current parameters π_P
 - 5 Modify parameters π_P such that the error decreases
- 6 **until** E is acceptable
- 7 **return** *The modified perceptron P*

The Simplest Perceptron



$$y_{w,t}(x) = \begin{cases} 1 & \text{if } x \cdot w \geq t \\ 0 & \text{otherwise} \end{cases}$$

The Simplest Perceptron



$$y_{w,t}(x) = \begin{cases} 1 & \text{if } x \cdot w \geq t \\ 0 & \text{otherwise} \end{cases}$$

1. How many parameters has this perceptron?

The Simplest Perceptron



$$y_{w,t}(x) = \begin{cases} 1 & \text{if } x \cdot w \geq t \\ 0 & \text{otherwise} \end{cases}$$

1. How many parameters has this perceptron?
2. Let the following dataset be given. How big is the error E ?

x	y	<i>Id</i>	<i>Error</i>
0	1		a

The Simplest Perceptron



$$y_{w,t}(x) = \begin{cases} 1 & \text{if } x \cdot w \geq t \\ 0 & \text{otherwise} \end{cases}$$

1. How many parameters has this perceptron?
2. Let the following dataset be given. How big is the error E ?

x	y	Id	Error
0	1	a	$ y_{w,t}(0) - 1 $

The Simplest Perceptron



$$y_{w,t}(x) = \begin{cases} 1 & \text{if } x \cdot w \geq t \\ 0 & \text{otherwise} \end{cases}$$

1. How many parameters has this perceptron?
2. Let the following dataset be given. How big is the error E ?

x	y	<i>Id</i>	<i>Error</i>
0	1	a	$ y_{w,t}(0) - 1 $
1	0	b	$ y_{w,t}(1) - 0 $

The Simplest Perceptron



$$y_{w,t}(x) = \begin{cases} 1 & \text{if } x \cdot w \geq t \\ 0 & \text{otherwise} \end{cases}$$

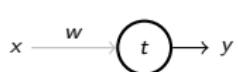
1. How many parameters has this perceptron?
2. Let the following dataset be given. How big is the error E ?

x	y	Id	Error
0	1	a	$ y_{w,t}(0) - 1 $
1	0	b	$ y_{w,t}(1) - 0 $



Give the mathematical formula for E wrt. w and t
(provide the signature and the definition)!

The Simplest Perceptron – The Error



x	y	<i>Id</i>
0	1	<i>a</i>
1	0	<i>b</i>

- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t)$$

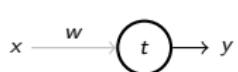
- ▶ Error wrt. datapoint b ($x = 1$):

$$E_b(w, t)$$

- ▶ Total error ($E : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$):

$$E(w, t) = E_a(w, t) + E_b(w, t)$$

The Simplest Perceptron – The Error



x	y	<i>Id</i>
0	1	<i>a</i>
1	0	<i>b</i>

- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } w \cdot 0 \geq t \\ 1 & \text{if } w \cdot 0 < t \end{cases}$$

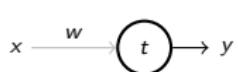
- ▶ Error wrt. datapoint b ($x = 1$):

$$E_b(w, t)$$

- ▶ Total error ($E : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$):

$$E(w, t) = E_a(w, t) + E_b(w, t)$$

The Simplest Perceptron – The Error



x	y	Id
0	1	a
1	0	b

- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } w \cdot 0 \geq t \\ 1 & \text{if } w \cdot 0 < t \end{cases} = \begin{cases} 0 & \text{if } 0 \geq t \\ 1 & \text{if } 0 < t \end{cases}$$

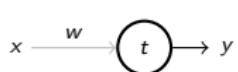
- ▶ Error wrt. datapoint b ($x = 1$):

$$E_b(w, t)$$

- ▶ Total error ($E : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$):

$$E(w, t) = E_a(w, t) + E_b(w, t)$$

The Simplest Perceptron – The Error



x	y	Id
0	1	a
1	0	b

- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } w \cdot 0 \geq t \\ 1 & \text{if } w \cdot 0 < t \end{cases} = \begin{cases} 0 & \text{if } 0 \geq t \\ 1 & \text{if } 0 < t \end{cases}$$

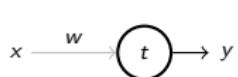
- ▶ Error wrt. datapoint b ($x = 1$):

$$E_b(w, t) = \begin{cases} 0 & \text{if } w \cdot 1 < t \\ 1 & \text{if } w \cdot 1 \geq t \end{cases}$$

- ▶ Total error ($E : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$):

$$E(w, t) = E_a(w, t) + E_b(w, t)$$

The Simplest Perceptron – The Error



x	y	Id
0	1	a
1	0	b

- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } w \cdot 0 \geq t \\ 1 & \text{if } w \cdot 0 < t \end{cases} = \begin{cases} 0 & \text{if } 0 \geq t \\ 1 & \text{if } 0 < t \end{cases}$$

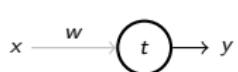
- ▶ Error wrt. datapoint b ($x = 1$):

$$E_b(w, t) = \begin{cases} 0 & \text{if } w \cdot 1 < t \\ 1 & \text{if } w \cdot 1 \geq t \end{cases} = \begin{cases} 0 & \text{if } w < t \\ 1 & \text{if } w \geq t \end{cases}$$

- ▶ Total error ($E : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$):

$$E(w, t) = E_a(w, t) + E_b(w, t)$$

The Simplest Perceptron – The Error



x	y	Id
0	1	a
1	0	b

- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } w \cdot 0 \geq t \\ 1 & \text{if } w \cdot 0 < t \end{cases} = \begin{cases} 0 & \text{if } 0 \geq t \\ 1 & \text{if } 0 < t \end{cases}$$

- ▶ Error wrt. datapoint b ($x = 1$):

$$E_b(w, t) = \begin{cases} 0 & \text{if } w \cdot 1 < t \\ 1 & \text{if } w \cdot 1 \geq t \end{cases} = \begin{cases} 0 & \text{if } w < t \\ 1 & \text{if } w \geq t \end{cases}$$

- ▶ Total error ($E : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$):

$$E(w, t) = E_a(w, t) + E_b(w, t) = \begin{cases} 0 & \text{if } 0 \geq t \text{ and } w < t \\ 2 & \text{if } 0 < t \text{ and } w \geq t \\ 1 & \text{otherwise} \end{cases}$$

The Simplest Perceptron – The Error Surfaces



x	y	Id
0	1	a
1	0	b

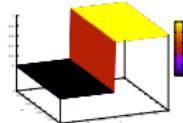
The Simplest Perceptron – The Error Surfaces



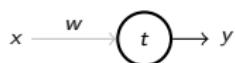
x	y	Id
0	1	a
1	0	b

- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } 0 \geq t \\ 1 & \text{if } 0 < t \end{cases}$$



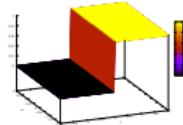
The Simplest Perceptron – The Error Surfaces



x	y	<i>Id</i>
0	1	a
1	0	b

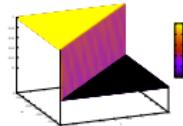
- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } 0 \geq t \\ 1 & \text{if } 0 < t \end{cases}$$

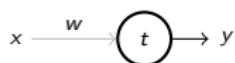


- ▶ Error wrt. datapoint b ($x = 1$):

$$E_b(w, t) = \begin{cases} 0 & \text{if } w < t \\ 1 & \text{if } w \geq t \end{cases}$$



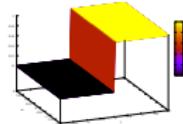
The Simplest Perceptron – The Error Surfaces



x	y	<i>Id</i>
0	1	a
1	0	b

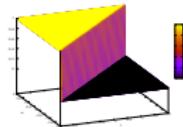
- ▶ Error wrt. datapoint a ($x = 0$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } 0 \geq t \\ 1 & \text{if } 0 < t \end{cases}$$



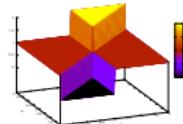
- ▶ Error wrt. datapoint b ($x = 1$):

$$E_b(w, t) = \begin{cases} 0 & \text{if } w < t \\ 1 & \text{if } w \geq t \end{cases}$$



- ▶ Total error ($E : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$):

$$E_a(w, t) = \begin{cases} 0 & \text{if } 0 \geq t \text{ and } w < t \\ 2 & \text{if } 0 < t \text{ and } w \geq t \\ 1 & \text{if otherwise} \end{cases}$$



Training a Perceptron

Training a Perceptron

Result: A trained perceptron

Input: A perceptron P , a set of training data D

- 1 Let π_P be the set of parameters of P :
weights and thresholds
- 2 Initialise all parameters π_P , randomly
- 3 **repeat**
- 4 Compute error E wrt. D and current parameters π_P
- 5 Modify parameters π_P such that the error decreases
- 6 **until** E is acceptable
- 7 **return** The modified perceptron P

Training a Perceptron

Training a Perceptron

Result: A trained perceptron

Input: A perceptron P , a set of training data D

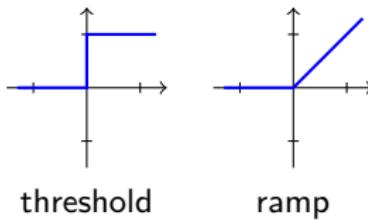
- 1 Let π_P be the set of parameters of P :
weights and thresholds
- 2 Initialise all parameters π_P , randomly
- 3 **repeat**
 - 4 Compute error E wrt. D and current parameters π_P
 - 5 Modify parameters π_P such that the error decreases
But, how can we modify the parameters?
- 6 **until** E is acceptable
- 7 **return** The modified perceptron P

From plateaus to slopes

- ▶ The error function for the step function is not well suited for training

From plateaus to slopes

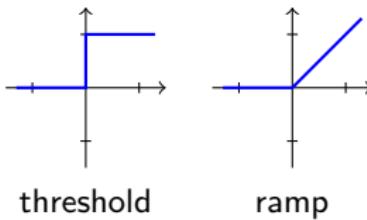
- ▶ The error function for the step function is not well suited for training
- ▶ Instead of using crisp decisions (step function), we will use a smoother version (ramp function) as activation function:



(The ramp function is called ReLU nowadays)

From plateaus to slopes

- ▶ The error function for the step function is not well suited for training
- ▶ Instead of using crisp decisions (step function), we will use a smoother version (ramp function) as activation function:



(The ramp function is called ReLU nowadays)

- ▶ I.e., instead of $\text{error} = 1$, we will use the distance from the weighted sum to the threshold

The Simplest Perceptron – The Error-Surface again



Draw the error-surfaces for the **ramp** function!

(gnuplot PerceptronRamp.gnuplot)

Training a Perceptron

Training a Perceptron

Result: A trained perceptron

Input: A perceptron P , a set of training data D

- 1 Let π_P be the set of parameters of P :
weights and thresholds
- 2 Initialise all parameters π_P , randomly
- 3 **repeat**
 - 4 Compute error E wrt. D and current parameters π_P
 - 5 Modify parameters π_P such that the error decreases
But, how can we modify the parameters?
- 6 **until** E is acceptable
- 7 **return** *The modified perceptron P*

Training a Perceptron

Training a Perceptron

Result: A trained perceptron

Input: A perceptron P , a set of training data D

- 1 Let π_P be the set of parameters of P :
weights and thresholds
- 2 Initialise all parameters π_P , randomly
- 3 **repeat**
 - 4 Compute error E wrt. D and current parameters π_P
 - 5 Modify parameters π_P such that the error decreases
But, how can we modify the parameters?
By going down-hill
- 6 **until** E is acceptable
- 7 **return** *The modified perceptron P*

Training a Perceptron: The Delta Rule

Delta Rule

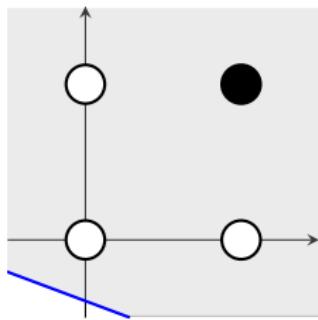
Input: A set P of n -dimensional positive examples and a set N of n -dimensional negative examples

Result: n -dim. weights \vec{w} and a threshold t , such that

$$\vec{w} \cdot \vec{p} \geq t \text{ for all } \vec{p} \in P \text{ and } \vec{w} \cdot \vec{n} < t \text{ for all } \vec{n} \in N$$

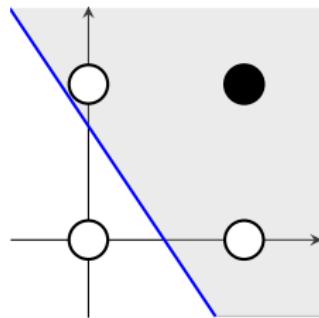
- 1 Initialise the weight vector \vec{w}_0 randomly
- 2 **repeat**
- 3 select a sample $\vec{x} \in P \cup N$ randomly
- 4 **if** $\vec{x} \in P$ **then**
- 5 **if** $\vec{w} \cdot \vec{x} < t$ **then** set $\vec{w} := \vec{w} + \vec{x}$ and $t := t - 1$
- 6 **else**
- 7 **if** $\vec{w} \cdot \vec{x} \geq t$ **then** set $\vec{w} := \vec{w} - \vec{x}$ and $t := t + 1$
- 8 **end**
- 9 **until** E is acceptable or maximal number of iterations
- 10 **return** The modified perceptron, i.e., \vec{w} and t

Training a Perceptron: The Delta Rule (visualisation)



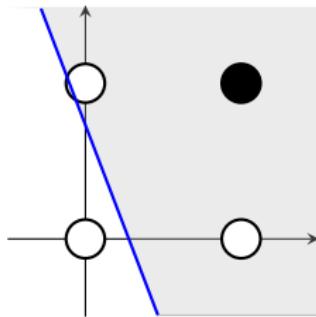
$$\begin{array}{cccc} t & w_1 & w_2 \\ \hline -0.35 & 0.33 & 0.89 \end{array}$$

Training a Perceptron: The Delta Rule (visualisation)



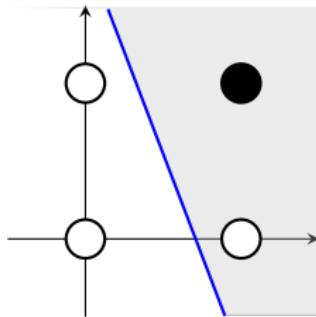
t	w_1	w_2
-0.35	0.33	0.89
0.65	1.33	0.89

Training a Perceptron: The Delta Rule (visualisation)



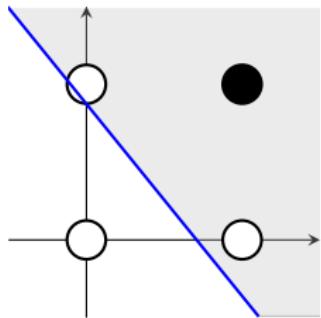
t	w_1	w_2
-0.35	0.33	0.89
0.65	1.33	0.89
0.65	2.33	0.89

Training a Perceptron: The Delta Rule (visualisation)



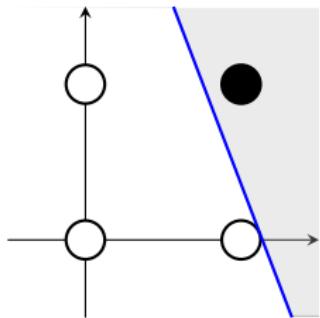
t	w_1	w_2
-0.35	0.33	0.89
0.65	1.33	0.89
0.65	2.33	0.89
1.65	2.33	0.89

Training a Perceptron: The Delta Rule (visualisation)



t	w_1	w_2
-0.35	0.33	0.89
0.65	1.33	0.89
0.65	2.33	0.89
1.65	2.33	0.89
1.65	2.33	1.89

Training a Perceptron: The Delta Rule (visualisation)



t	w_1	w_2
-0.35	0.33	0.89
0.65	1.33	0.89
0.65	2.33	0.89
1.65	2.33	0.89
1.65	2.33	1.89
2.65	2.33	0.89

Supervised Learning

Based on given data, a *supervised learning algorithm* has to learn a mapping from inputs to correct outputs.

- ▶ The training data consists of pairs: (input, desired output)
- ▶ Let a set D of n input-output data-pairs be given:

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

with y_i being generated by some unknown function f , i.e.,
 $y_i = f(x_i) + \text{noise}$.

- ▶ Then, a supervised learning procedure **determines a function h from the set of possible hypotheses $h \in \mathcal{H}$** which performs well on D .

Perceptron: Hypothesis Space

- ▶ What is the hypothesis space \mathcal{H} of a perceptron?

Perceptron: Hypothesis Space

- ▶ What is the hypothesis space \mathcal{H} of a perceptron?
- ▶ The perceptron implements a linear decision!

Perceptron: Hypothesis Space

- ▶ What is the hypothesis space \mathcal{H} of a perceptron?
- ▶ The perceptron implements a linear decision!
- ▶ Therefore, only linearly separable classification tasks can be solved

Perceptron: Hypothesis Space

- ▶ What is the hypothesis space \mathcal{H} of a perceptron?
- ▶ The perceptron implements a linear decision!
- ▶ Therefore, only linearly separable classification tasks can be solved
- ▶ I.e., the hypothesis space \mathcal{H} of a perceptron with n inputs, is the set of all linear separable binary classification functions over n inputs

Perceptron: Hypothesis Space

- ▶ What is the hypothesis space \mathcal{H} of a perceptron?
- ▶ The perceptron implements a linear decision!
- ▶ Therefore, only linearly separable classification tasks can be solved
- ▶ I.e., the hypothesis space \mathcal{H} of a perceptron with n inputs, is the set of all linear separable binary classification functions over n inputs
- ▶ Will the delta-rule always find a solution?

Perceptron: Hypothesis Space

- ▶ What is the hypothesis space \mathcal{H} of a perceptron?
- ▶ The perceptron implements a linear decision!
- ▶ Therefore, only linearly separable classification tasks can be solved
- ▶ I.e., the hypothesis space \mathcal{H} of a perceptron with n inputs, is the set of all linear separable binary classification functions over n inputs

- ▶ Will the delta-rule always find a solution?
- ▶ If yes, will it always find the same solution?

Delta-Rule Convergence Theorem

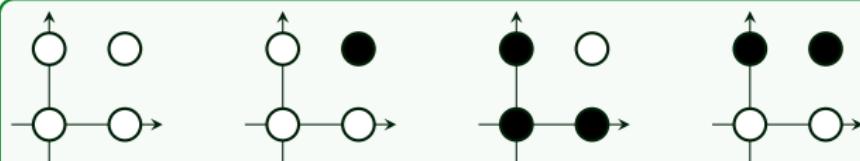
Theorem (Delta-Rule Convergence Theorem)

If there is a weight vector \vec{w}^* such that $\Theta(\vec{w}^* \cdot \vec{x} - t) = y$ for all (\vec{x}, y) , (i.e., the problem is linearly separable)
then for any starting vector \vec{w} , the delta rule will converge to an updated weight vector \vec{w} (not necessarily unique and not necessarily \vec{w}^*) that gives the correct response for all training patterns, and it will do so in a finite number of steps.

www.cs.ubbcluj.ro/~csatol/kozgaz_mestint/4_neuronhalo/PerceptConvProof.pdf

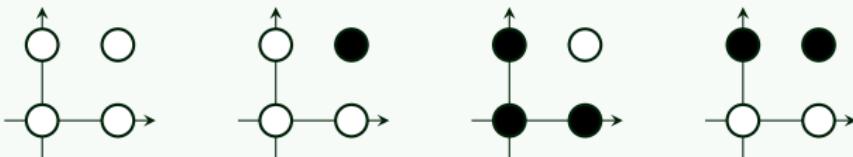
Perceptron: Some linearly (in)separable problems

- ▶ Linear separable problems in 2d (4 out of 14 in total):

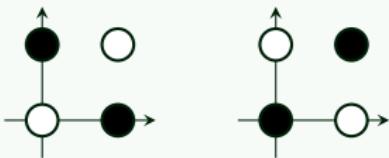


Perceptron: Some linearly (in)separable problems

- ▶ Linear separable problems in 2d (4 out of 14 in total):



- ▶ Linear inseparable problems in 2d:



Perceptron: Number of linearly (in)separable problems

- ▶ For two inputs, there are only 2 inseparable binary problems, but 14 separable ones, i.e., no problem?

Perceptron: Number of linearly (in)separable problems

- ▶ For two inputs, there are only 2 inseparable binary problems, but 14 separable ones, i.e., no problem?
- ▶ Unfortunately in higher dimensions there is a problem:

# var	# fns	# Lin.-sep.functions
2	16	14
3	256	104
4	65536	1882
5	$4.29 \cdot 10^{10}$	$9.45 \cdot 10^5$
6	$1.84 \cdot 10^{19}$	$1.50 \cdot 10^8$
7	$3.40 \cdot 10^{38}$	$8.37 \cdot 10^{10}$
8	$1.15 \cdot 10^{77}$	$1.75 \cdot 10^{14}$
9	$1.34 \cdot 10^{154}$	$1.44 \cdot 10^{18}$

N. Grueling. *Linear separability of the vertices of an n-dimensional hypercube* (2007)

Summary

- ▶ A single perceptron can be trained ...
 - using the delta rule
 - to compute any linearly separable boolean function
(because they compute a linear decision)
- ▶ Networks of perceptrons ...
 - can compute any boolean function
 - can not be trained using the delta rule!
It is not applicable for non-output units, because the target mapping of the hidden units is unknown.

Take-away-messages of section: “*Training Perceptrons*”



You should now be able to ...

- ▶ explain how to train a single layer perceptron

Take-away-messages of section: “*Training Perceptrons*”



You should now be able to ...

- ▶ explain how to train a single layer perceptron
- ▶ describe, compute and draw the error surface of a single threshold unit

Take-away-messages of section: “*Training Perceptrons*”



You should now be able to ...

- ▶ explain how to train a single layer perceptron
- ▶ describe, compute and draw the error surface of a single threshold unit
- ▶ describe the ideas behind the *Delta Rule*

Take-away-messages of section: “*Training Perceptrons*”



You should now be able to ...

- ▶ explain how to train a single layer perceptron
- ▶ describe, compute and draw the error surface of a single threshold unit
- ▶ describe the ideas behind the *Delta Rule*
- ▶ explain the concept of a hypothesis space by example for a perceptron

Take-away-messages of section: “*Training Perceptrons*”



You should now be able to ...

- ▶ explain how to train a single layer perceptron
- ▶ describe, compute and draw the error surface of a single threshold unit
- ▶ describe the ideas behind the *Delta Rule*
- ▶ explain the concept of a hypothesis space by example for a perceptron
- ▶ know what caused the first AI winter

Künstliche Intelligenz

Feedforward Neural Networks

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Artificial Neural Network

- ▶ An *artificial neural network* is a graph of artificial neurons.
- ▶ Some units receive external inputs (*input units*)
- ▶ Every unit computes its *potential* based on the outputs of its predecessors and its incoming weights
- ▶ Every unit computes an *output value* by applying a non-linear function to the potential and a bias value
- ▶ Different neural architectures differ with respect to:
 - Type of input function used (e.g., weighted sum)
 - Type of output function (e.g., threshold, sigmoidal, relu, ...)
 - Connection structure (acyclic = feed forward, cyclic = recurrent, layered = groups of units, convolutional, ...)
 - Dynamics (synchronous, asynchronous, probabilistic, ...)

Nice introductory videos

Below you will find a list of nicely animated introductory videos:

- ▶ But what is a Neural Network? Deep learning, chap.1:
[YouTube [aircAruvnKk](#)]
- ▶ How Deep Neural Networks Work (up to Min. 12):
[YouTube [ILsA4nyG7IO](#)]
- ▶ Artificial Neural Networks - Fun and Easy Machine Learning (up to Min. 10):
[YouTube [GQVL10RqpSs](#)]
- ▶ A Visual And Interactive Look at Basic Neural Network Math:
jalammar.github.io/feedforward-neural-networks-visual-interactive

A larger Network in Action

adamharley.com/nn_vis/cnn/2d.html

Take-away-messages of section: “*Dynamics of Artificial Neural Networks in General*”



You should now be able to ...

- ▶ explain the general structure of an artificial neural networks as a directed graph of neurons

Take-away-messages of section: “*Dynamics of Artificial Neural Networks in General*”



You should now be able to ...

- ▶ explain the general structure of an artificial neural networks as a directed graph of neurons
- ▶ describe different connection architectures

Agenda

Dynamics of Artificial Neural Networks in General

Simple Feed Forward Network

Hands On

Training Artificial Neural Networks

Learning objective of section: “*Simple Feed Forward Network*”

In this section we will ...

- ▶ focus on a certain subset of neural networks - in which units are organised in layers
- ▶ discuss the universal approximation capabilities of feed-forward networks
- ▶ introduce online and batch learning

Feed-Forward Artificial Neural Network

- ▶ An *Artificial Neural Network* consist of ...
 - a set U of units
 - a set of connections $C \subseteq U \times U$,
each labelled with a weight $w_{i,j} \in \mathbb{R}$
 - ...

Feed-Forward Artificial Neural Network

- ▶ An *Artificial Neural Network* consist of ...

- a set U of units
- a set of connections $C \subseteq U \times U$,
each labelled with a weight $w_{i,j} \in \mathbb{R}$
- ...

- ▶ In a *Feed-Forward Artificial Neural Network* ...

- the units are organised in a sequence of n disjoint sub-sets U_1, \dots, U_n (called *layers*) with

$$U = \bigcup_i U_i \quad \text{and} \quad U_i \cap U_j = \emptyset \quad \text{for all } i \neq j$$

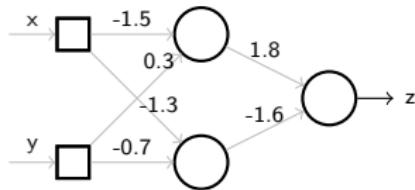
- all units from layer i are connected to all units in layer $i + 1$:

$$C = C_1 \cup \dots \cup C_{n-1}, \quad \text{with}$$

$$C_i = U_i \times U_{i+1}$$

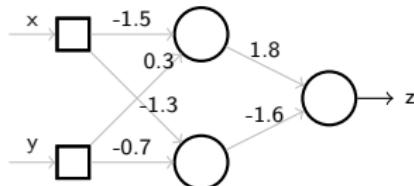
A Simple Feed-Forward Network

- ▶ The following network consists of 3 layers:
 - U_1 - called the input layer (with two units labelled x and y)
 - U_2 - called the hidden layer (with two unlabelled units)
 - U_3 - called the output layer (with the single unit labelled z)



A Simple Feed-Forward Network

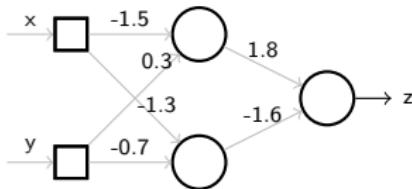
- ▶ The following network consists of 3 layers:
 - U_1 - called the input layer (with two units labelled x and y)
 - U_2 - called the hidden layer (with two unlabelled units)
 - U_3 - called the output layer (with the single unit labelled z)



- ▶ What does this network compute?

A Simple Feed-Forward Network

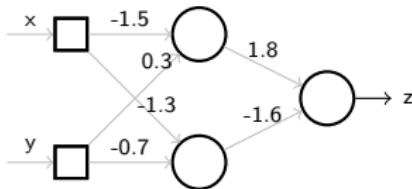
- ▶ The following network consists of 3 layers:
 - U_1 - called the input layer (with two units labelled x and y)
 - U_2 - called the hidden layer (with two unlabelled units)
 - U_3 - called the output layer (with the single unit labelled z)



- ▶ What does this network compute?
 - We do not know yet, because it is not specified how inputs, weights, etc. are combined

A Simple Feed-Forward Network

- ▶ The following network consists of 3 layers:
 - U_1 - called the input layer (with two units labelled x and y)
 - U_2 - called the hidden layer (with two unlabelled units)
 - U_3 - called the output layer (with the single unit labelled z)



- ▶ What does this network compute?
 - We do not know yet, because it is not specified how inputs, weights, etc. are combined
 - But we know the signature of the overall network function:

$$\mathcal{N} : \mathbb{R}^2 \rightarrow \mathbb{R}$$

Units of Connectionist Systems – Input Function

- ▶ *Input function I* (aka *activation function*): maps n -dimensional inputs \vec{x} and n -dimensional weights \vec{w} to an activation potential p :

$$I : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$$

$$(\vec{x}, \vec{w}) \mapsto \sum_{i=1}^n (x_i - w_i)^2 \quad (\textit{squared distance})$$

Units of Connectionist Systems – Input Function

- ▶ *Input function I* (aka *activation function*): maps n -dimensional inputs \vec{x} and n -dimensional weights \vec{w} to an activation potential p :

$$I : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$$

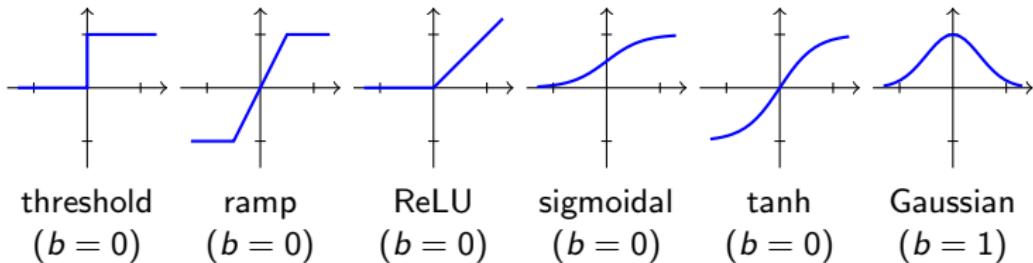
$$(\vec{x}, \vec{w}) \mapsto \sum_{i=1}^n (x_i - w_i)^2 \quad (\textit{squared distance})$$

$$(\vec{x}, \vec{w}) \mapsto \sum_{i=1}^n x_i \cdot w_i \quad (\textit{weighted sum})$$

- ▶ Most frameworks ( TensorFlow,  Keras,  PyTorch) use the weighted sum by default

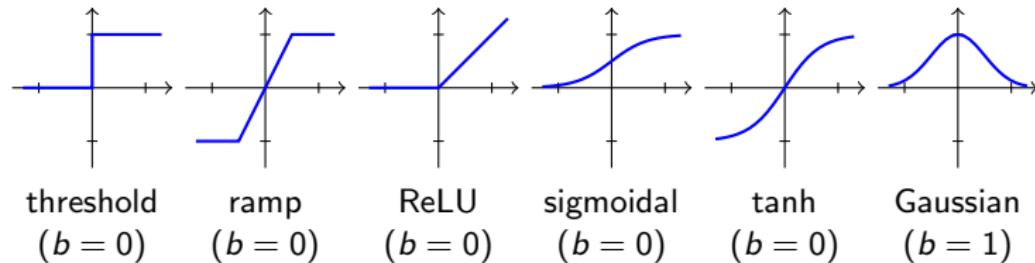
Units of Connectionist Systems – Output Function

- ▶ *Output function* (aka *activation function*): maps the potential p and bias b to the output o . I.e. computes $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$



Units of Connectionist Systems – Output Function

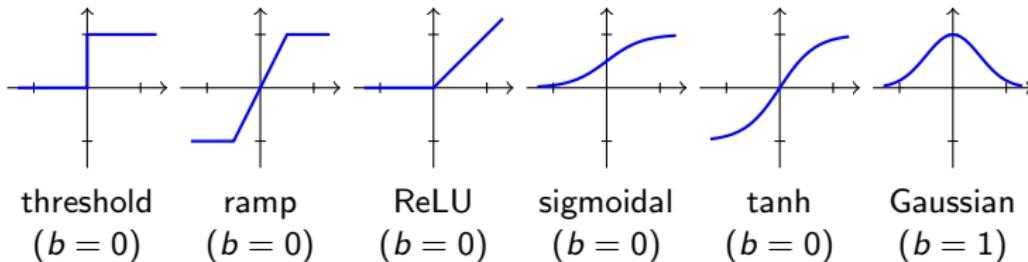
- ▶ *Output function* (aka *activation function*): maps the potential p and bias b to the output o . I.e. computes $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$



- ▶ Until about 2000, the sigmoidal function (or the tanh) was the most prominent activation function

Units of Connectionist Systems – Output Function

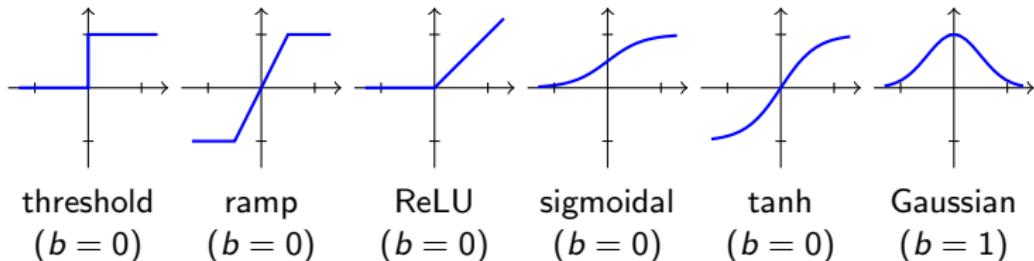
- ▶ *Output function* (aka *activation function*): maps the potential p and bias b to the output o . I.e. computes $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$



- ▶ Until about 2000, the sigmoidal function (or the tanh) was the most prominent activation function
- ▶ A [Radial basis function network] relies on the gaussian function (with the euclidean distance as input function)

Units of Connectionist Systems – Output Function

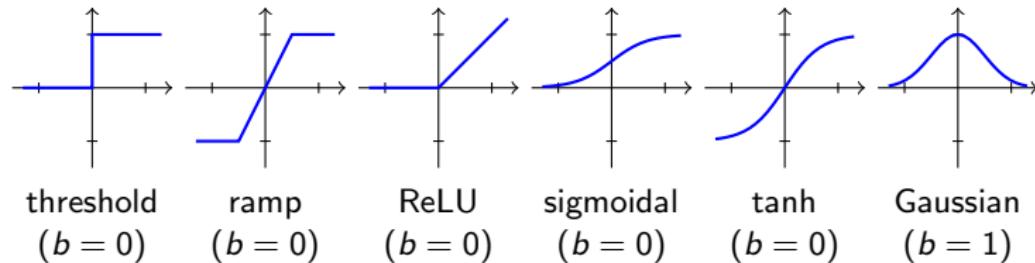
- ▶ *Output function* (aka *activation function*): maps the potential p and bias b to the output o . I.e. computes $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$



- ▶ Until about 2000, the sigmoidal function (or the tanh) was the most prominent activation function
- ▶ A [Radial basis function network] relies on the gaussian function (with the euclidean distance as input function)
- ▶ State of the art systems mostly use the ReLU function

Units of Connectionist Systems – Output Function

- ▶ *Output function* (aka *activation function*): maps the potential p and bias b to the output o . I.e. computes $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$



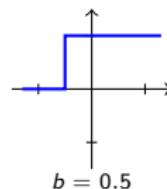
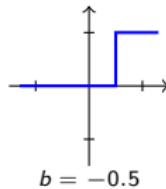
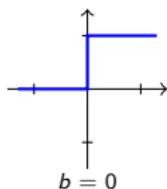
- ▶ Until about 2000, the sigmoidal function (or the tanh) was the most prominent activation function
- ▶ A [Radial basis function network] relies on the gaussian function (with the euclidean distance as input function)
- ▶ State of the art systems mostly use the ReLU function
- ▶ List of possible functions: [Activation function]

Output-functions and Bias

- ▶ *Output function A* maps potential p and bias b to output o
- ▶ **Note:** bias \neq threshold
 - A threshold has to be exceeded in order to activate a unit
 - A bias is added to the input
 - I.e., bias \approx -threshold
- ▶ Threshold function:

$$\theta(p, b) = \begin{cases} 1 & \text{if } p + b \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

- ▶ Shape for different values of b :

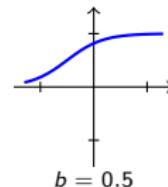
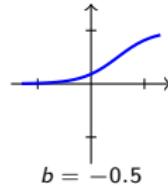
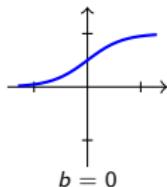


Output-functions and Bias

- ▶ Sigmoidal function:

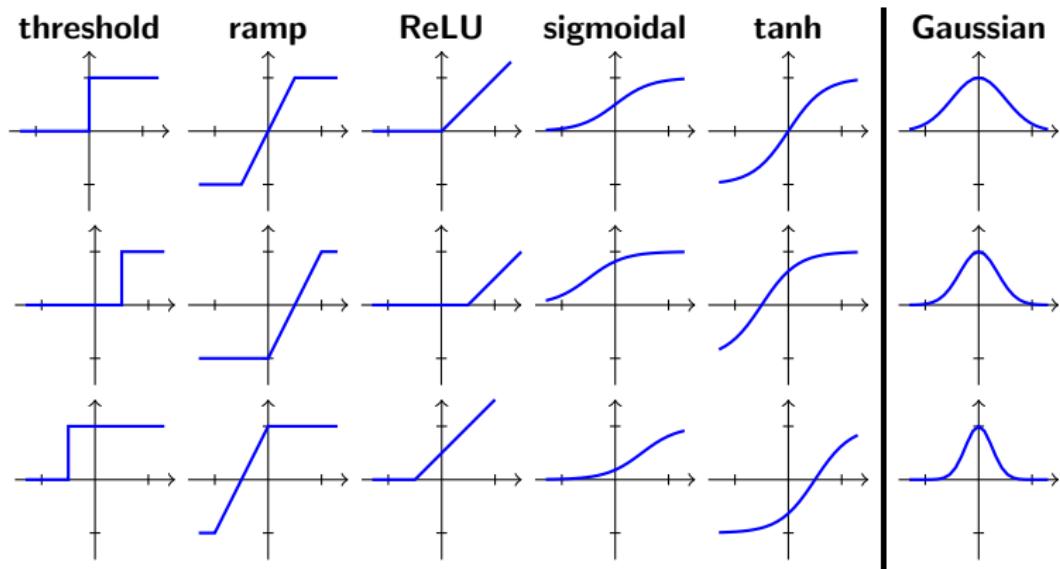
$$f(x, b) = \text{sig}(x + b) = \frac{1}{1.0 + e^{-(x+b)}}$$

- ▶ Shape for different values of b :



Output-functions and Bias

- ▶ Shape of activation functions (as function over p) for three different values for b :



Ridge output functions

A *ridge function* is some univariate function g applied to a linear combination of the inputs: $f = g(a \cdot x + b)$.

- ▶ *Linear activation:*

$$o(p, b) = p + b$$

- ▶ *ReLU activation:*

$$o(p, b) = \max(0, p + b)$$

- ▶ *Sigmoidal functions*, e.g., the *logistic function*:

$$o(p, b) = \frac{1}{1 + e^{-p-b}}$$

Ridge output functions

A *ridge function* is some univariate function g applied to a linear combination of the inputs: $f = g(a \cdot x + b)$.

Used together with the weighted sum input function.

- ▶ *Linear activation:*

$$o(p, b) = p + b \quad o(\vec{x}, \vec{w}, b) = \vec{w} \cdot \vec{x} + b$$

- ▶ *ReLU activation:*

$$o(p, b) = \max(0, p + b) \quad o(\vec{x}, \vec{w}, b) = \max(0, \vec{w} \cdot \vec{x} + b)$$

- ▶ *Sigmoidal functions*, e.g., the *logistic function*:

$$o(p, b) = \frac{1}{1 + e^{-p-b}} \quad o(\vec{x}, \vec{w}, b) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x} - b}}$$

Radial basis output functions

A *radial basis function* is some real-valued function whose value at each point depends only on the distance between that point and some other fixed point:

- ▶ *Gaussian*:

$$o(p, b) = e^{-\frac{p^2}{b^2}}$$

Radial basis output functions

A *radial basis function* is some real-valued function whose value at each point depends only on the distance between that point and some other fixed point:

- ▶ *Gaussian*:

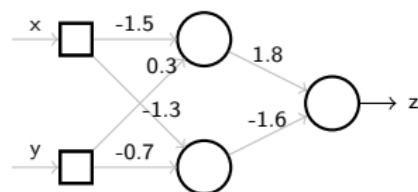
$$o(p, b) = e^{-\frac{p^2}{b^2}} \quad o(\vec{x}, \vec{w}, b) = e^{-\frac{(\vec{x}-\vec{w})^2}{b^2}}$$

Used together with (squared) distance input function.

Folding output functions

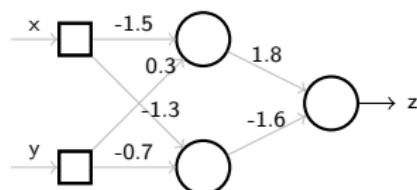
- ▶ A *fold function* (reduce, aggregate, compress) is a function which combines it's inputs recursively.
- ▶ Fold functions combine the outputs of multiple units p_1, \dots, p_n within a layer. Instead of $o : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, they compute a function $\mathbb{R}^n \rightarrow \mathbb{R}^m$.
- ▶ Examples:
 - *Pooling*: Maximum, Minimum, Mean, ... of a group of units
 - *Softmax*: Renormalise all outputs to form a probability distribution:
$$o_i = \frac{e^{o_i}}{\sum_j e^{o_j}}$$
- ▶ They are either implemented as activation function or as additional layer:
 - `tf.keras.layers.Dense(10, activation = "softmax")`
 - `tf.keras.layers.MaxPooling2D(pool_size=(3, 3))`

Dynamics of a Network

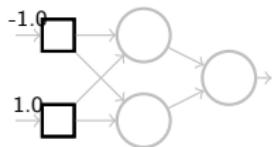


	Input F.	Output F.
input	p set from outside	$o = p$
hidden	$p = \sum_n (i_n - w_n)^2$	$o = e^{-p^2}$
output	$p = \sum_n (i_n * w_n)$	$o = p$

Dynamics of a Network

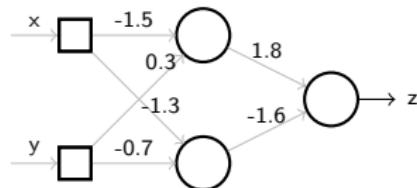


set input:



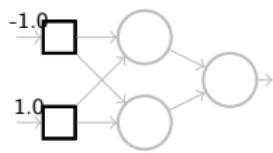
	Input F.	Output F.
input	p set from outside	$o = p$
hidden	$p = \sum_n (i_n - w_n)^2$	$o = e^{-p^2}$
output	$p = \sum_n (i_n * w_n)$	$o = p$

Dynamics of a Network

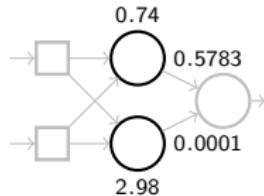


	Input F.	Output F.
input	p set from outside	$o = p$
hidden	$p = \sum_n (i_n - w_n)^2$	$o = e^{-p^2}$
output	$p = \sum_n (i_n * w_n)$	$o = p$

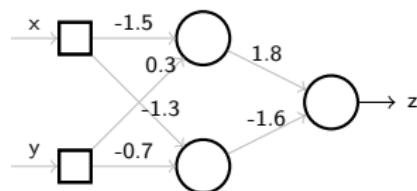
set input:



hidden layer:

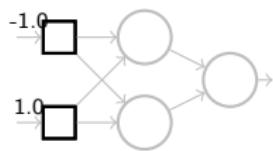


Dynamics of a Network

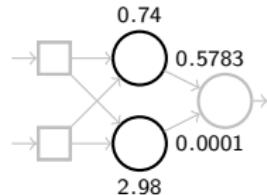


	Input F.	Output F.
input	p set from outside	$o = p$
hidden	$p = \sum_n (i_n - w_n)^2$	$o = e^{-p^2}$
output	$p = \sum_n (i_n * w_n)$	$o = p$

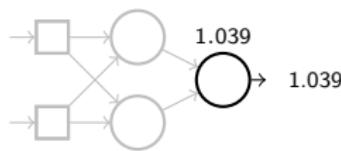
set input:



hidden layer:



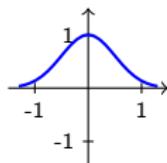
output:



Dynamics of a Network

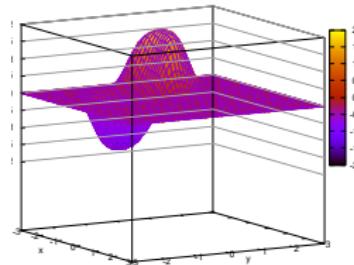
Activation function

$$p = \sum_n (i_n - w_n)^2$$



Output function

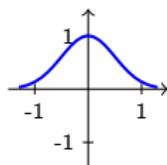
Result



Dynamics of a Network

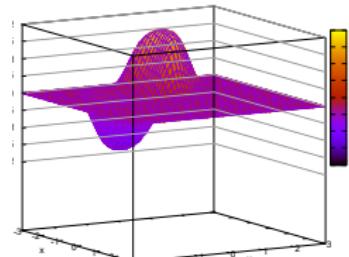
Activation function

$$p = \sum_n (i_n - w_n)^2$$

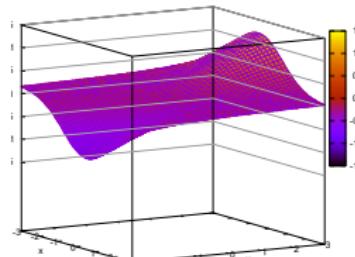
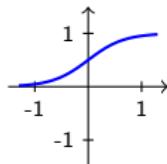


Output function

Result



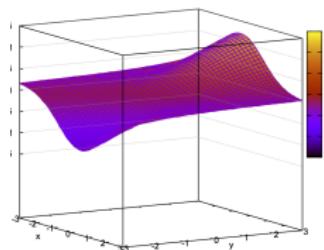
$$p = \sum_n i_n \cdot w_n$$



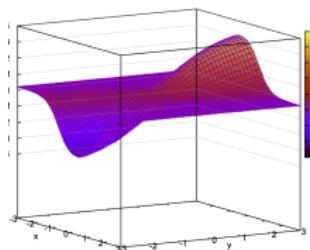
Dynamics of a Network (ctd.)

- ▶ Using the weighted sum as input function, the behaviour only depends on the activation function:

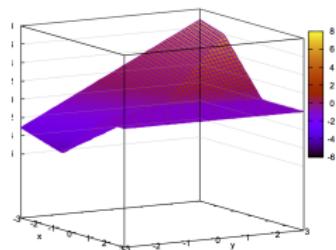
Sigmoidal



TanH

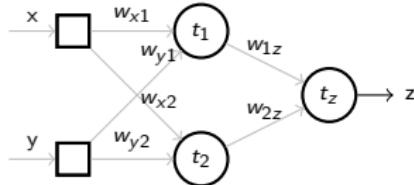


ReLU



A Simple Example

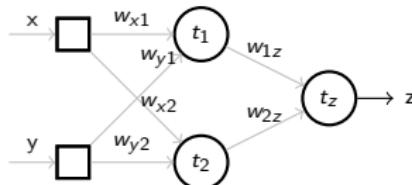
- ▶ Let the following 3-layer feed-forward network be given:



- ▶ Input units:
 - *Input function*: input value set from outside
 - *Output function*: identity
- ▶ Hidden units:
 - *Input function*: weighted sum
 - *Output function*: sigmoidal
- ▶ Output units:
 - *Input function*: weighted sum
 - *Output function*: sigmoidal

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

$$o_x = x$$

$$o_y = y$$

- Hidden units:

$$i_1 = w_{x1} * o_x + w_{y1} * o_y \quad o_1 = \text{sig}(i_1 + t_1)$$

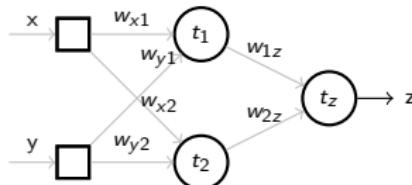
$$i_2 = w_{x2} * o_x + w_{y2} * o_y \quad o_2 = \text{sig}(i_2 + t_2)$$

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \quad o_z = \text{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

- Hid
$$z = o_z = \text{sig}(i_z + t_z)$$

$$i_1 = w_{x1} * o_x + w_{y1} * o_y \quad o_1 = \text{sig}(i_1 + t_1)$$

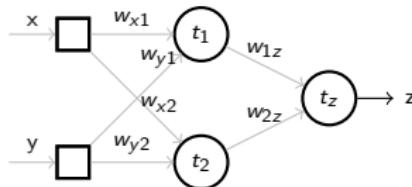
$$i_2 = w_{x2} * o_x + w_{y2} * o_y \quad o_2 = \text{sig}(i_2 + t_2)$$

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \quad o_z = \text{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

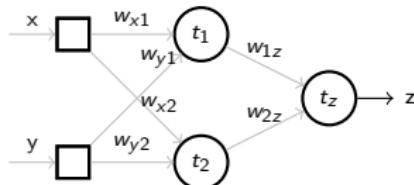
- Hidden units:
$$z = \text{sig}(w_{1z} * o_1 + w_{2z} * o_2 + t_z)$$

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \quad o_z = \text{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

- Hid

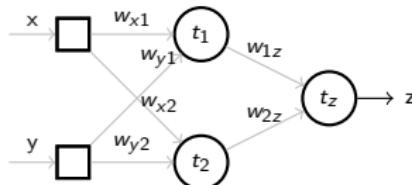
$$z = \text{sig}(w_{1z} * \text{sig}(i_1 + t_1) + w_{2z} * \text{sig}(i_2 + t_2) + t_z)$$

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \quad o_z = \text{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

- Hid

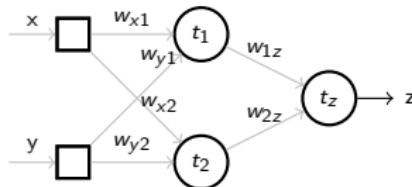
$$z = \text{sig}(w_{1z} * \text{sig}(w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig}(w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$

- Output units:

$$i_z = w_{1z} * o_1 + w_{2z} * o_2 \quad o_z = \text{sig}(i_z + t_z)$$

A Simple Example (cont.)

- Let the following 3-layer feed-forward network be given:



- Input units:

- Hid
$$z = \text{sig}(w_{1z} * \text{sig}(w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig}(w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$
- Ou with $\text{sig}(x) = \frac{1}{1.0 + e^{-x}}$

Network Function

- ▶ The *Network input-output function* \mathcal{N} computes the output for a given input vector \vec{x} :

$$\mathcal{N}(\vec{x}) := f_{w_{ij}, t_i, \dots}(\vec{x})$$

- ▶ It depends on:
 - *hyper-parameters* of the network: structure, input and output functions of all units, update schema
 - *(trainable) parameters* of the network: weights and bias values
- ▶ The network function \mathcal{N} is well defined for feed-forward networks

Feed-Forward Artificial Neural Network

- ▶ An *Artificial Neural Network* consist of ...
 - a set U of units
 - a set of connections $C \subseteq U \times U$, each labelled with a weight $w_{i,j} \in \mathbb{R}$
 - ...
- ▶ In a *Feed-Forward Artificial Neural Network (FFN)* ...
 - the units are organised in a sequence of n disjoint sub-sets U_1, \dots, U_n (called *layers*) with

$$U = \bigcup_i U_i \quad \text{and} \quad U_i \cap U_j = \emptyset \quad \text{for all } i \neq j$$

- all units from layer i are connected to all units in layer $i + 1$:

$$C = C_1 \cup \dots \cup C_{n-1}, \quad \text{with}$$

$$C_i = U_i \times U_{i+1}$$

Network Function for FFNs

- The *Network input-output function* \mathcal{N} computes the output for a given input vector \vec{x} :

$$\mathcal{N}(\vec{x}) := \mathcal{L}_n(\vec{x})$$

$$\mathcal{L}_1(\vec{x}) := \vec{x}$$

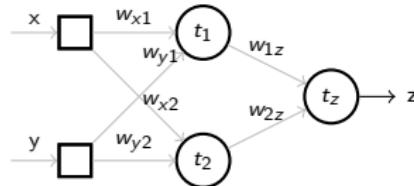
$$\mathcal{L}_i(\vec{x}) := \vec{A}_i(\vec{l}_i(W_i, \mathcal{L}_{i-1}(\vec{x})), \vec{t}_i)$$

with

- n being the number of layers, and n_i being the size of layer i
- W_i being the weight matrix between layer $i - 1$ and layer i , i.e., a matrix of shape $n_{i-1} \times n_i$
- t_i being the vector of biases for layer i
- $\vec{l}_i : \mathbb{R}^{n_{i-1}} \times \mathbb{R}^{n_{i-1} \times n_i} \rightarrow \mathbb{R}^{n_i}$ being the vectorised version of the *input function* l_i for layer i
- $\vec{A}_i : \mathbb{R}^{n_i} \times \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{n_i}$ being the vectorised version of the *activation function* A_i for layer i

Network Function for weighted sum FFNs

- Let the following 3-layer feed-forward network be given:

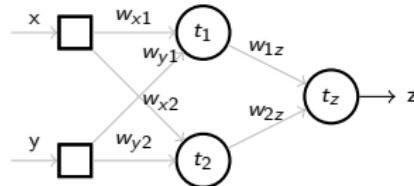


- The network function can be computed as follows:

$$I_1 \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} \begin{pmatrix} x & y \end{pmatrix}$$

Network Function for weighted sum FFNs

- ▶ Let the following 3-layer feed-forward network be given:

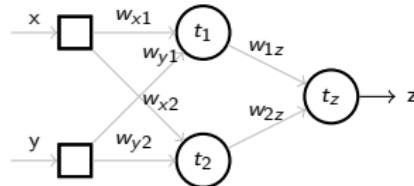


- ▶ The network function can be computed as follows:

$$\begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} \begin{pmatrix} x & y \\ i_1 & i_2 \end{pmatrix}$$

Network Function for weighted sum FFNs

- ▶ Let the following 3-layer feed-forward network be given:

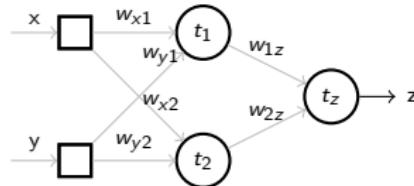


- ▶ The network function can be computed as follows:

$$\begin{pmatrix} x & y \end{pmatrix} \quad \begin{pmatrix} i_1 & i_2 \end{pmatrix} \quad \rightsquigarrow_{A_1} \quad \begin{pmatrix} o_1 & o_2 \end{pmatrix}$$
$$\begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix}$$

Network Function for weighted sum FFNs

- Let the following 3-layer feed-forward network be given:

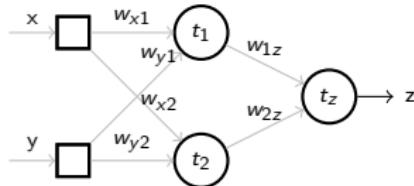


- The network function can be computed as follows:

$$\begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} \rightsquigarrow_{A_1} \begin{pmatrix} i_1 & i_2 \end{pmatrix} I_2 \rightsquigarrow \begin{pmatrix} o_1 & o_2 \end{pmatrix} \begin{pmatrix} w_{1z} \\ w_{2z} \end{pmatrix}$$

Network Function for weighted sum FFNs

- ▶ Let the following 3-layer feed-forward network be given:

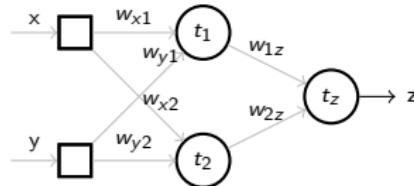


- ▶ The network function can be computed as follows:

$$\begin{pmatrix} x & y \end{pmatrix} \quad \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} \quad \begin{pmatrix} i_1 & i_2 \end{pmatrix} \quad \rightsquigarrow_{A_1} \quad \begin{pmatrix} o_1 & o_2 \end{pmatrix} \quad \begin{pmatrix} w_{1z} \\ w_{2z} \end{pmatrix} \quad \begin{pmatrix} i_z \end{pmatrix}$$

Network Function for weighted sum FFNs

- ▶ Let the following 3-layer feed-forward network be given:



- ▶ The network function can be computed as follows:

$$\begin{pmatrix} x & y \end{pmatrix} \quad \begin{pmatrix} w_{x1} & w_{x2} \\ w_{y1} & w_{y2} \end{pmatrix} \quad \rightsquigarrow_{A_1} \quad \begin{pmatrix} i_1 & i_2 \end{pmatrix} \quad \begin{pmatrix} o_1 & o_2 \end{pmatrix} \quad \begin{pmatrix} w_{1z} \\ w_{2z} \end{pmatrix} \quad \rightsquigarrow_{A_2} \quad \begin{pmatrix} i_z \end{pmatrix} \quad \begin{pmatrix} o_z \end{pmatrix}$$

Take-away-messages of section: “*Simple Feed Forward Network*”



- You should now be able to ...
- ▶ describe the architecture of a feed-forward network

Take-away-messages of section: “*Simple Feed Forward Network*”



You should now be able to ...

- ▶ describe the architecture of a feed-forward network
- ▶ describe and explain the differences between different input and output functions

Take-away-messages of section: “*Simple Feed Forward Network*”



You should now be able to ...

- ▶ describe the architecture of a feed-forward network
- ▶ describe and explain the differences between different input and output functions
- ▶ name, explain and draw some prominent input and output functions

Take-away-messages of section: “*Simple Feed Forward Network*”



You should now be able to ...

- ▶ describe the architecture of a feed-forward network
- ▶ describe and explain the differences between different input and output functions
- ▶ name, explain and draw some prominent input and output functions
- ▶ describe the intuition behind computing the network function of a FFN in matrix formulation

Agenda

Dynamics of Artificial Neural Networks in General

Simple Feed Forward Network

Hands On

Training Artificial Neural Networks

<https://playground.tensorflow.org>

Take-away-messages of section: “*Hands On*”

- ★ You should now be able to ...
 - ▶ describe the dynamics of a neural network during the training phase

Take-away-messages of section: “*Hands On*”

-  You should now be able to ...
 - ▶ describe the dynamics of a neural network during the training phase
 - ▶ able to explain the effect of good features on the overall performance

Agenda

Dynamics of Artificial Neural Networks in General

Simple Feed Forward Network

Hands On

Training Artificial Neural Networks

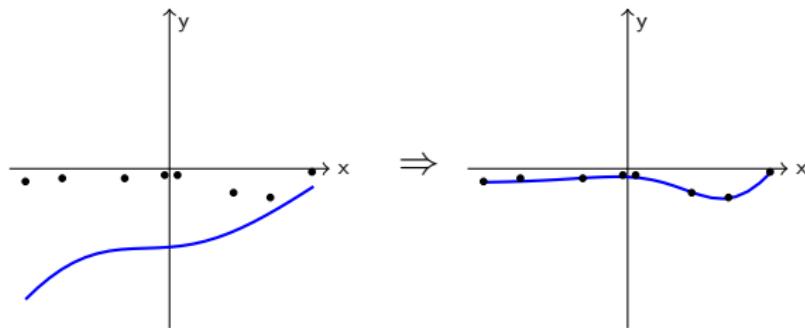
Learning objective of section: “*Training Artificial Neural Networks*”

In this section we will ...

- ▶ discuss the idea of adaptation by gradient descent
- ▶ define the network function wrt. trainable parameters
- ▶ derive the equations to adapt the weights and thresholds of a simple network by hand

Training Artificial Neural Networks

- ▶ How can we train a network to represent a function given as a set of samples $\{(x_1, y_1), \dots, (x_n, y_n)\}$?



- ▶ Learning as generalization.

Training a Neural Network

Training a Neural Network

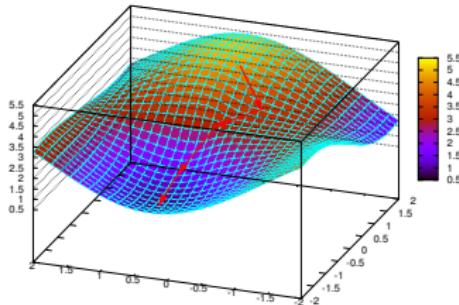
Result: A trained network

Input: A Network N , a set of training data D

- 1 Let π_N be the set of parameters of N :
weights and thresholds
- 2 Initialise all parameters π_N , randomly
- 3 **repeat**
 - 4 Compute error E wrt. D and current parameters π_N
 - 5 Modify parameters π_N such that the error decreases
- 6 **until** E is acceptable
- 7 **return** *The modified network N*

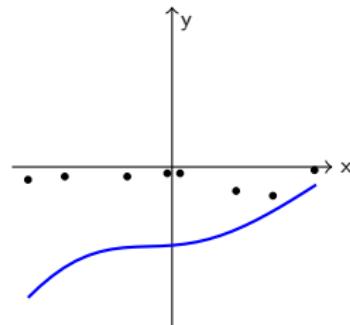
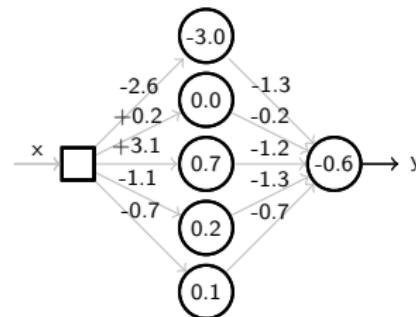
Backpropagation

- ▶ Let a set of samples $\{(x_1, y_1), \dots, (x_n, y_n)\}$ be given.
- ▶ Error of the network: $E = \sum_i (\mathcal{N}(x_i) - y_i)^2$.
(with $\mathcal{N}(x_i)$ being the output of the network for the input x_i)
- ▶ Idea: minimise E by gradient descent.



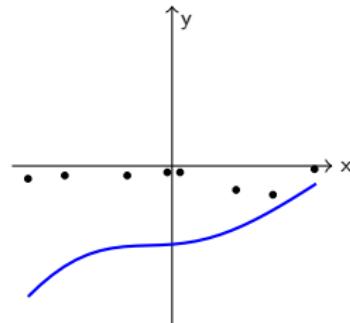
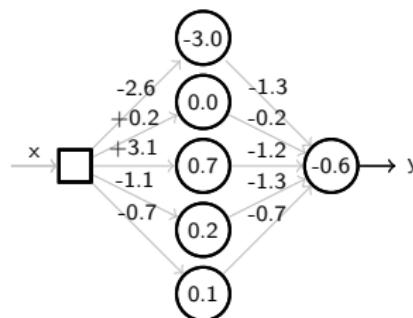
A sample run ...

Prior training:

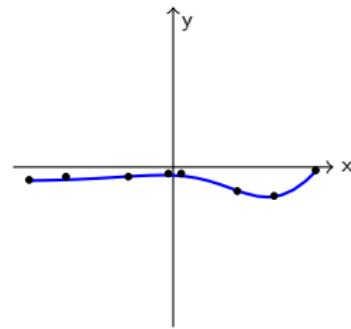
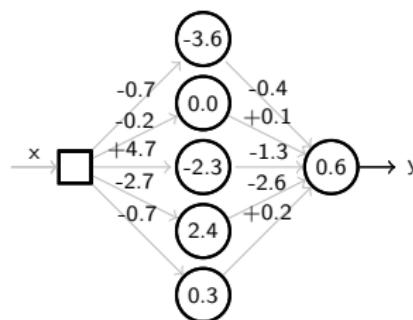


A sample run ...

Prior training:



After training:



Training Schemes

- ▶ **Online learning:** All parameters are adapted after presenting a single example
- ▶ **Batch learning:** Changes to the parameters are accumulated and parameters are adapted after all samples from a batch have been processed

Desired vs. Actual Input-Output Behaviour

- ▶ Actual input-output-behaviour as just derived:

$$z = \text{sig} (w_{1z} * \text{sig} (w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig} (w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$
$$= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}$$

Desired vs. Actual Input-Output Behaviour

- ▶ Actual input-output-behaviour as just derived:

$$z = \text{sig} (w_{1z} * \text{sig} (w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig} (w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$
$$= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}$$

- ▶ What is the desired output of the network?

Desired vs. Actual Input-Output Behaviour

- Actual input-output-behaviour as just derived:

$$z = \text{sig} (w_{1z} * \text{sig} (w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig} (w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$
$$= \frac{1}{1 + e^{-\left(w_{1z} * \frac{1}{1 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}$$

- What is the desired output of the network?
- Usually it is “specified” in form of training samples D :

x	0.0	0.1	0.0	1.0
y	0.0	0.0	1.0	1.0
z	0.0	1.0	1.0	1.0

$D := \{((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 1) \}$

Desired vs. Actual Input-Output Behaviour

- Actual input-output-behaviour as just derived:

$$z = \text{sig} (w_{1z} * \text{sig} (w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig} (w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$
$$= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}$$

- What is the desired output of the network?
- Usually it is “specified” in form of training samples D :

x	0.0	0.1	0.0	1.0
y	0.0	0.0	1.0	1.0
z	0.0	1.0	1.0	1.0

$D := \{((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 1) \}$

- Goal of the training: modify the parameters of the network function, i.e.: weights w_{ij} and bias values t_i

Desired vs. Actual Input-Output Behaviour

- Actual input-output-behaviour as just derived:

$$z = \text{sig} (w_{1z} * \text{sig} (w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig} (w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$
$$= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}$$

- What is the desired output of the network?
- Usually it is “specified” in form of training samples D :

x	0.0	0.1	0.0	1.0
y	0.0	0.0	1.0	1.0
z	0.0	1.0	1.0	1.0

$D := \{((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 1) \}$

- Goal of the training: modify the parameters of the network function, i.e.: weights w_{ij} and bias values t_i ;
- Please note: the hyper-parameter (e.g., structure and functions) are not adjusted as part of the training

Training a Neural Network

Training a Neural Network

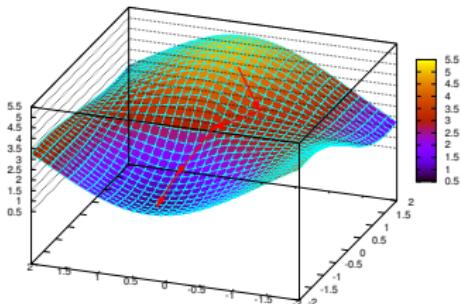
Result: A trained network

Input: A Network N , a set of training data D

- 1 Let π_N be the set of parameters of N :
weights and thresholds
- 2 Initialise all parameters π_N , randomly
- 3 **repeat**
 - 4 Compute error E wrt. D and current parameters π_N
 - 5 Modify parameters π_N such that the error decreases
- 6 **until** E is acceptable
- 7 **return** *The modified network N*

Backpropagation

- ▶ Let a set of samples $\{(x_1, y_1), \dots, (x_n, y_n)\}$ be given.
- ▶ Error of the network defined by a loss function, e.g., quadratic loss $E = \sum_i (\mathcal{N}(x_i) - y_i)^2$.
(with $\mathcal{N}(x_i)$ being the output of the network for the input x_i)
- ▶ Idea: minimise E by gradient descent.



Gradient descent

“Gradient descent is an [...] iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum [...] we take steps proportional to the negative of the gradient [...] of the function at the current point. [...] Gradient descent was originally proposed by Cauchy in 1847.”

[ *Gradient descent*]

Augustin-Louis Cauchy

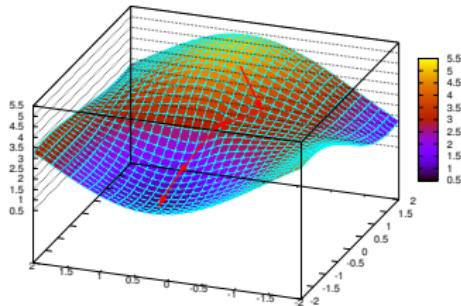
- ▶ *21 Aug. 1789, †23 May 1857
- ▶ French mathematician
- ▶ Some of his influential work:
 - Analysis: formal definition of continuity based on infinitesimals, *Cours d'Analyse* (1821)
 - Converging sequences: Cauchy sequences
 - Probability theory: Cauchy distributions



[⌚ Augustin-Louis Cauchy]

Gradient Descent

- ▶ *Gradient descent* is an optimisation algorithm to find a local minimum of a given function
- ▶ Idea:
 1. Select a starting position
 2. Compute the gradient of the function at the current point
 3. Make a step towards the steepest descent (down hill)
 4. Repeat step 2 and 3 until satisfied



Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$

Result: A local minimum

1 **repeat**

2 | Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$

3 | Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$

4 **until** $\nabla f(\vec{x}) = \vec{0}$ or given number of iterations

5 **return** \vec{x} , which is a minimum iff $\nabla f(\vec{x}) = \vec{0}$

Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$

Result: A local minimum

1 **repeat**

2 | Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$

3 | Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$

4 **until** $\nabla f(\vec{x}) = \vec{0}$ or given number of iterations

5 **return** \vec{x} , which is a minimum iff $\nabla f(\vec{x}) = \vec{0}$

- ▶ for certain function classes (e.g., convex and Lipschitz continuous) and suitable γ , GD converges to a local minimum

Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$

Result: A local minimum

1 **repeat**

2 | Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$

3 | Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$

4 **until** $\nabla f(\vec{x}) = \vec{0}$ or given number of iterations

5 **return** \vec{x} , which is a minimum iff $\nabla f(\vec{x}) = \vec{0}$

- ▶ for certain function classes (e.g., convex and Lipschitz continuous) and suitable γ , GD converges to a local minimum
- ▶ the step size γ can be different for every iteration

Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$

Result: A local minimum

1 **repeat**

2 | Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$

3 | Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$

4 **until** $\nabla f(\vec{x}) = \vec{0}$ or given number of iterations

5 **return** \vec{x} , which is a minimum iff $\nabla f(\vec{x}) = \vec{0}$

- ▶ for certain function classes (e.g., convex and Lipschitz continuous) and suitable γ , GD converges to a local minimum
- ▶ the step size γ can be different for every iteration
- ▶ if f is convex, every local minimum is a global minimum

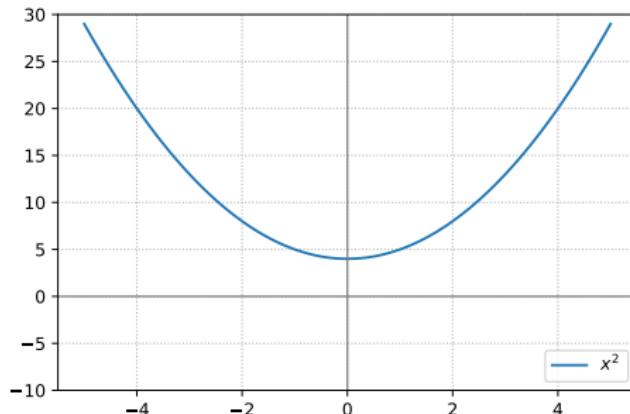
Rules for Partial Derivatives

do you
recall?

Rule	F	$\partial F / \partial x$
Constant	c	0
Factor	$c \cdot f(x)$	$c \frac{\partial f(x)}{\partial x}$
Power rule	x^n	$n \cdot x^{n-1}$
Sum rule	$f(x) + g(x)$	$\frac{\partial f(x)}{\partial x} + \frac{\partial g(x)}{\partial x}$
Product rule	$f(x) \cdot g(x)$	$f \frac{\partial g(x)}{\partial x} + g \frac{\partial f(x)}{\partial x}$
Chain rule	$f(g(x))$	$\frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$
Exponential	e^x	e^x

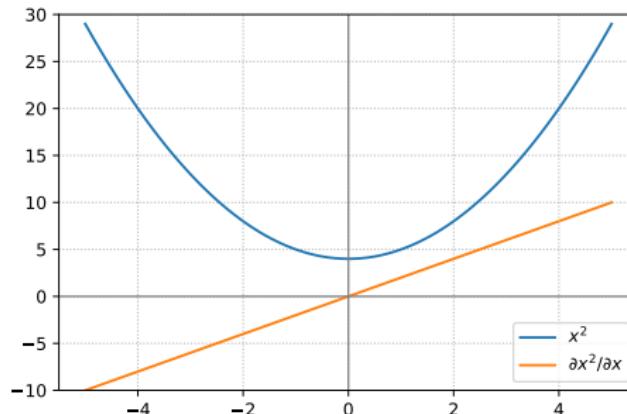
Gradient Descent by Example

- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



Gradient Descent by Example

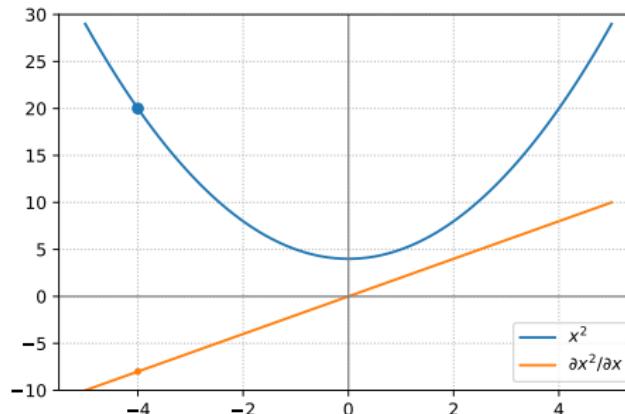
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

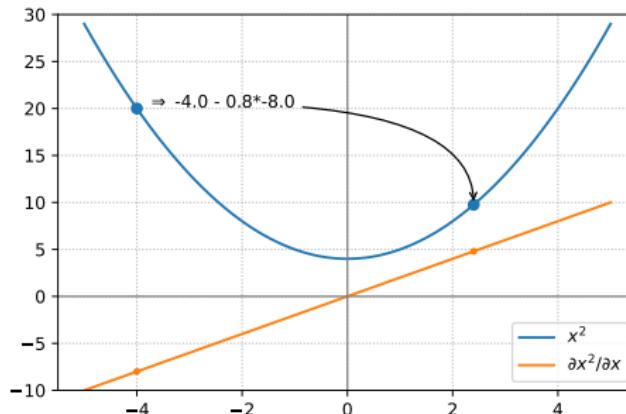
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

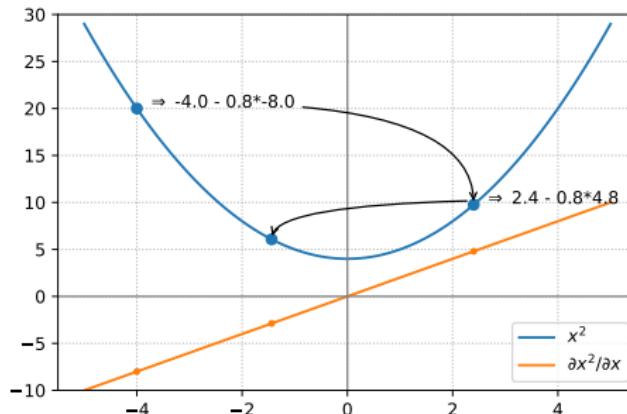
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

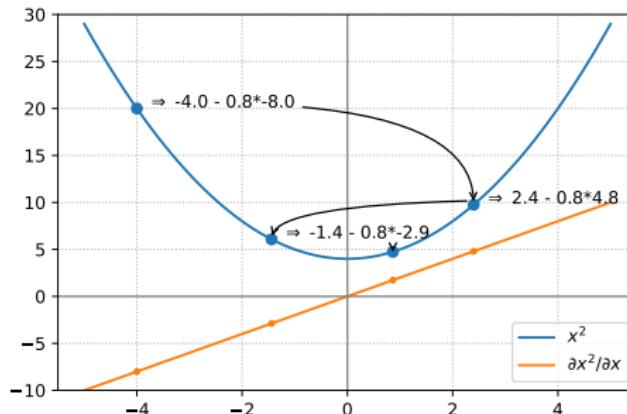
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

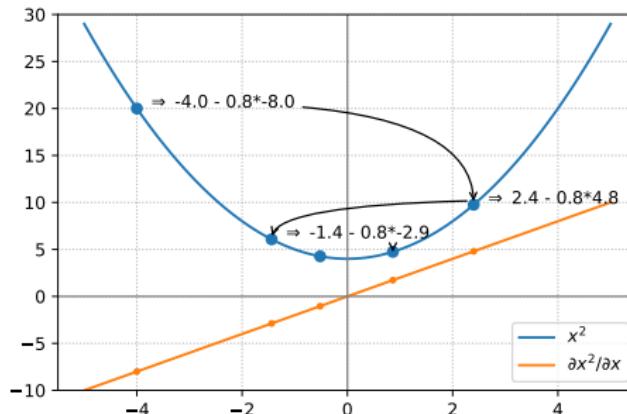
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

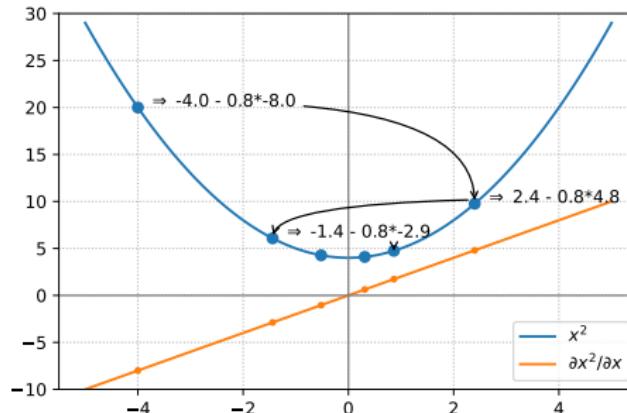
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

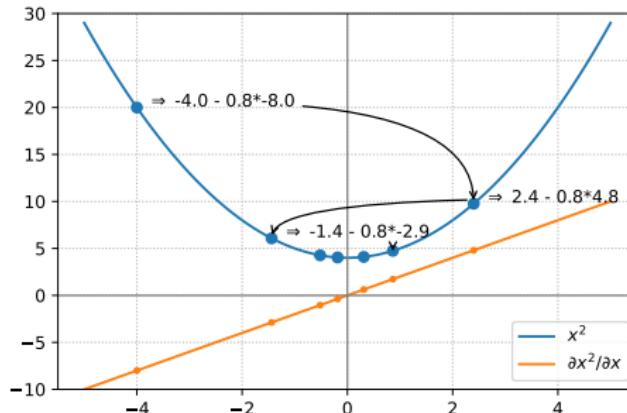
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

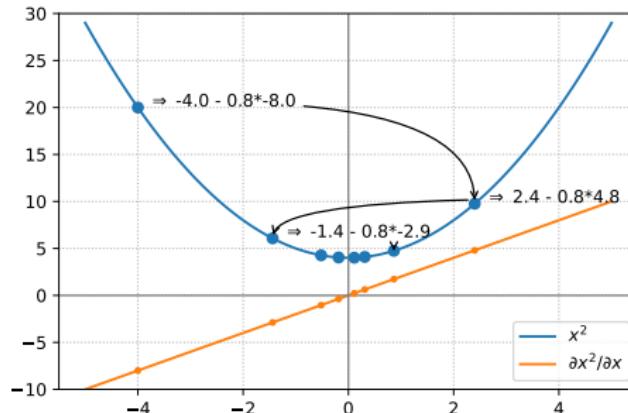
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

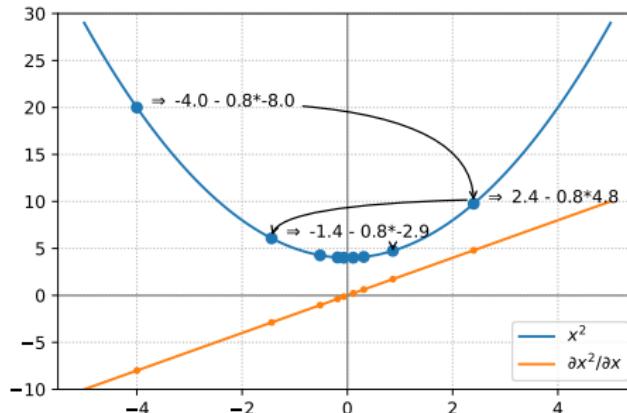
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

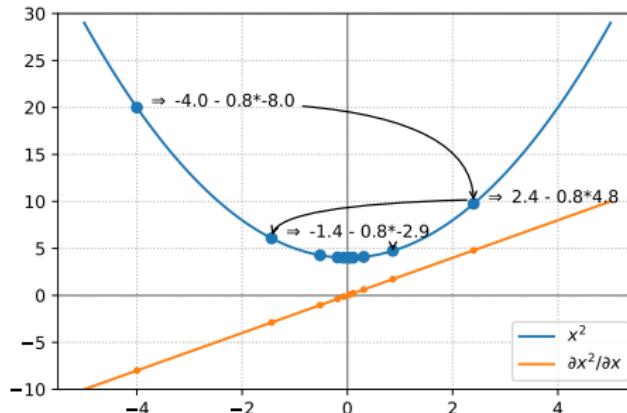
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

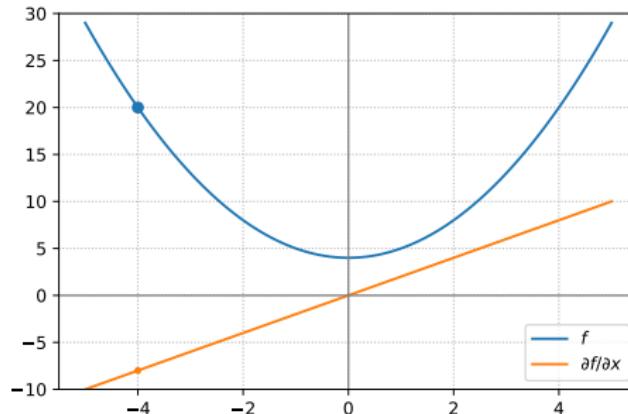
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

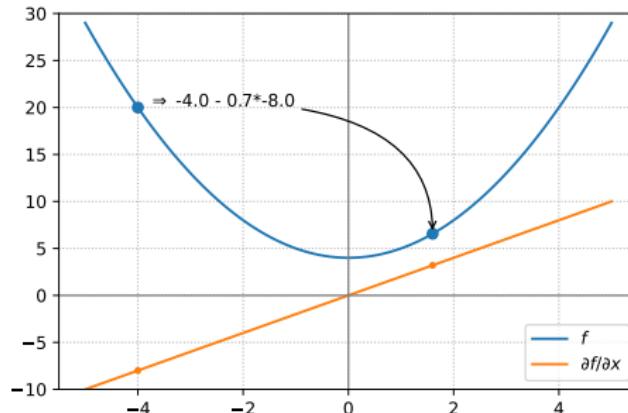
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

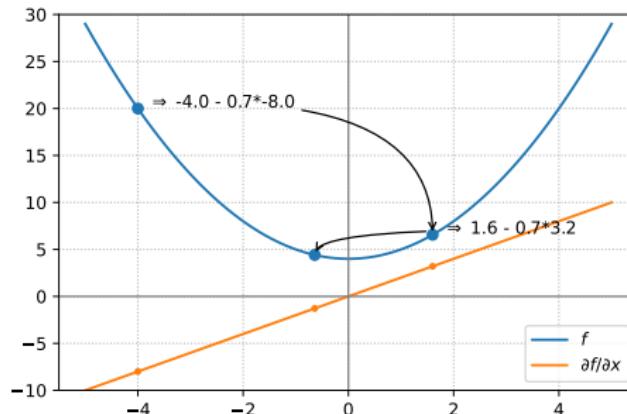
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

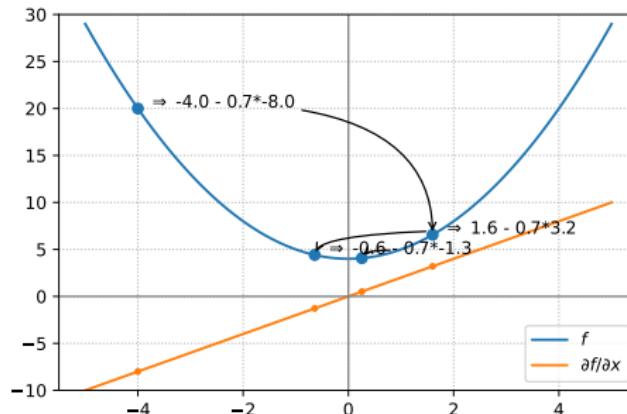
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

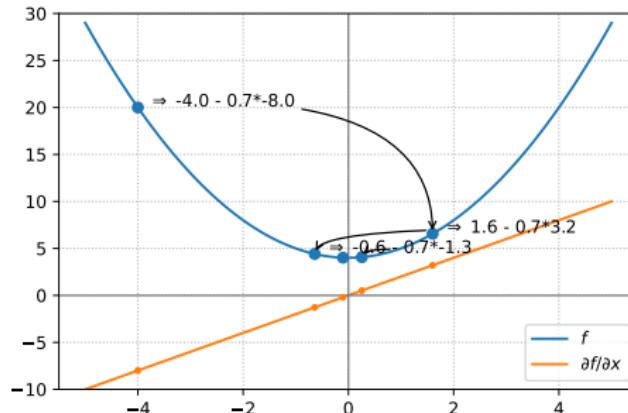
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

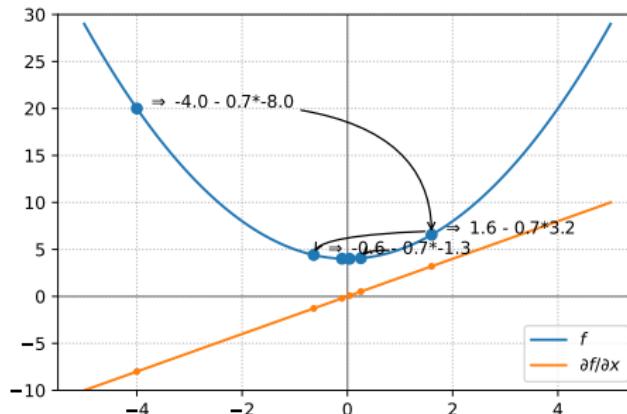
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

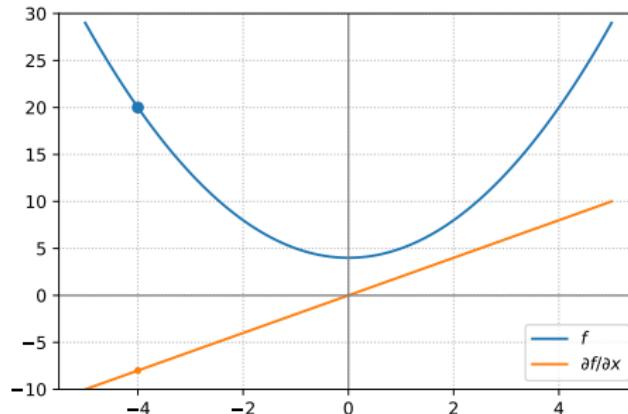
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.7$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

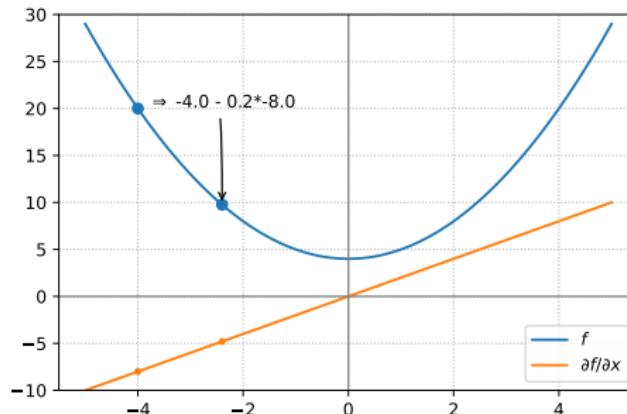
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

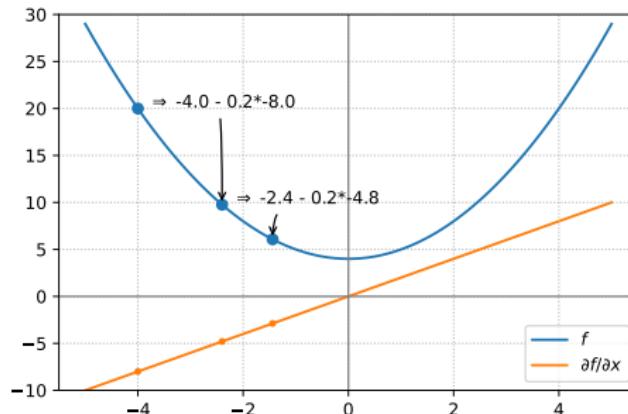
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

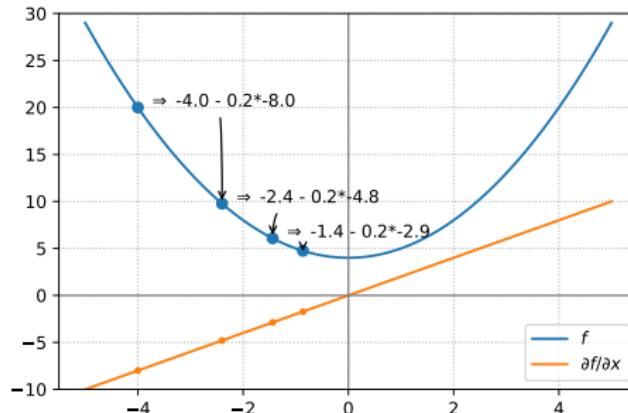
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

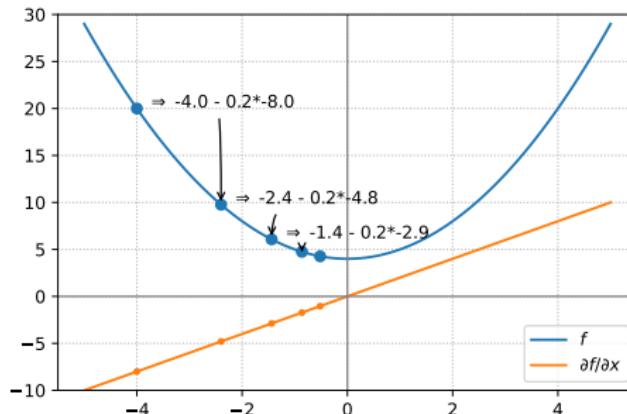
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

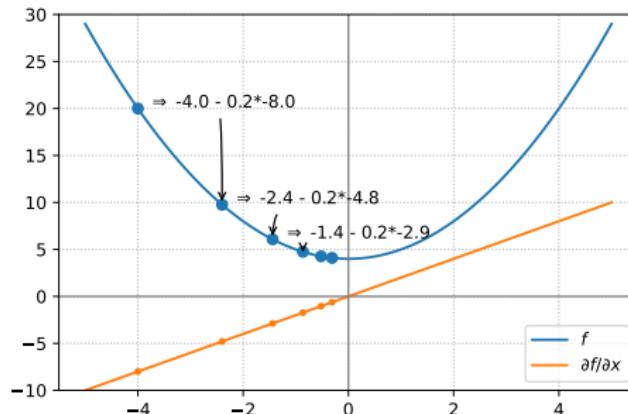
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

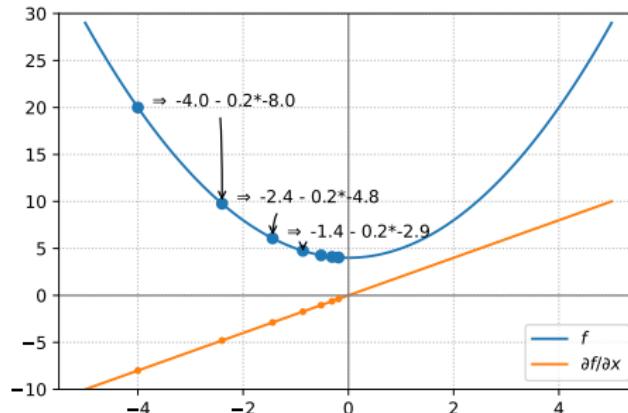
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

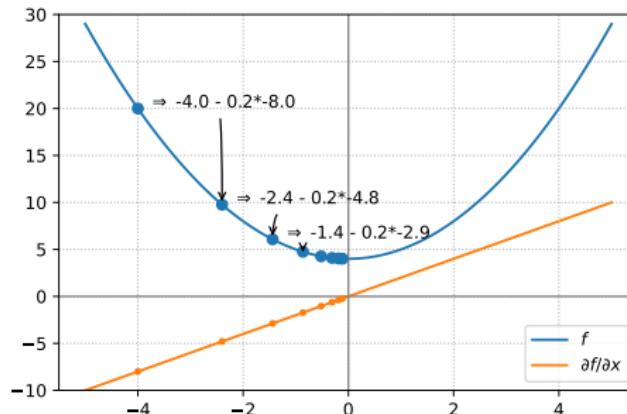
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

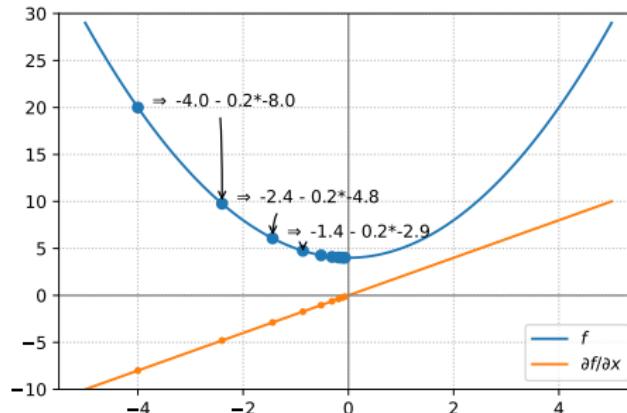
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

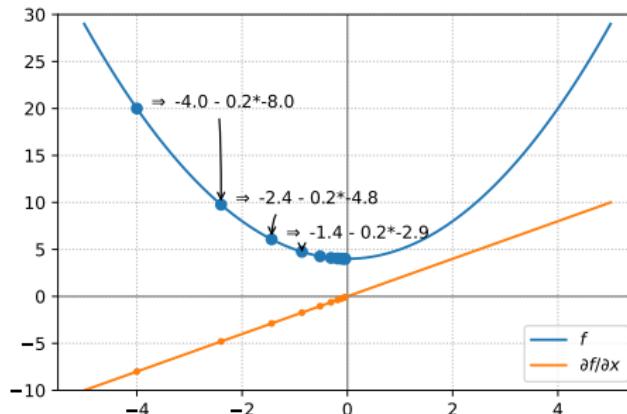
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example

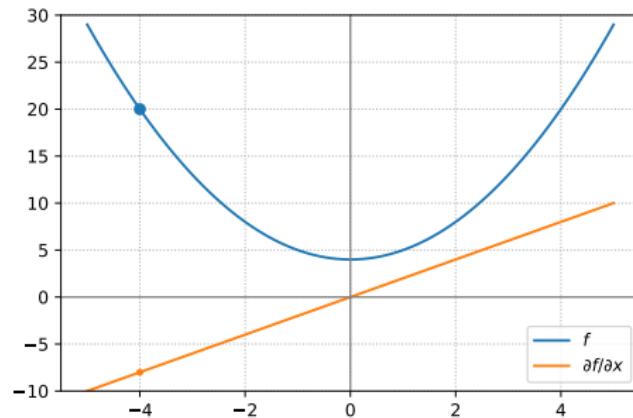
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative $(\frac{\partial f}{\partial x})(x_0) = -8.0$ and subtract e.g., $\gamma = 0.2$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example - ping pong

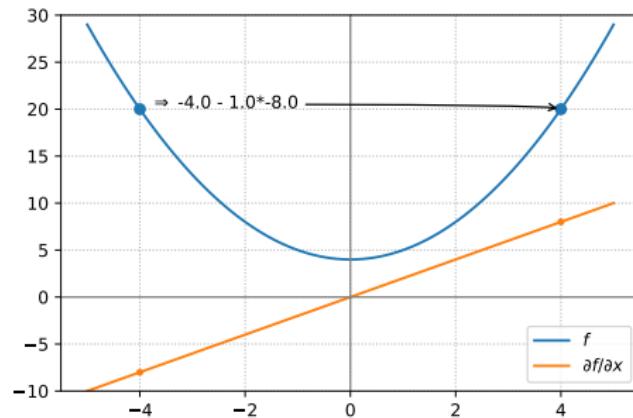
- ▶ But there might be combinations of function and step size for which this fails:



- ▶ There are strategies, e.g., stochastic gradient descent which solve this and similar problems (discussed later)

Gradient Descent by Example - ping pong

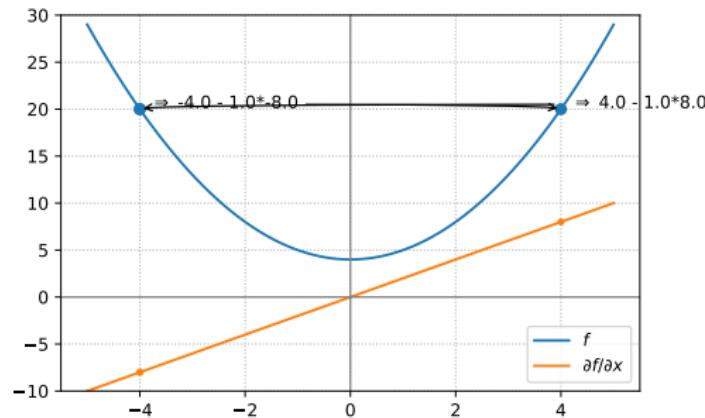
- ▶ But there might be combinations of function and step size for which this fails:



- ▶ There are strategies, e.g., stochastic gradient descent which solve this and similar problems (discussed later)

Gradient Descent by Example - ping pong

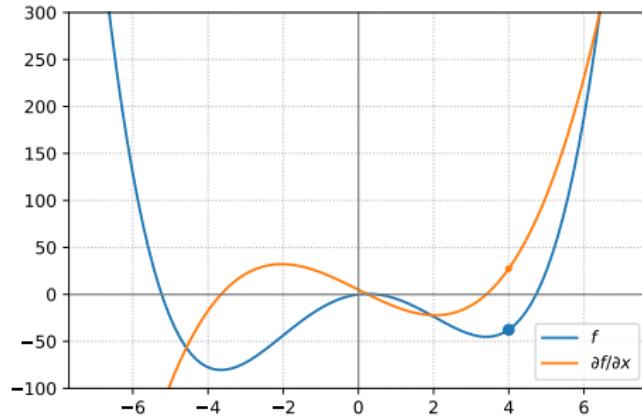
- ▶ But there might be combinations of function and step size for which this fails:



- ▶ There are strategies, e.g., stochastic gradient descent which solve this and similar problems (discussed later)

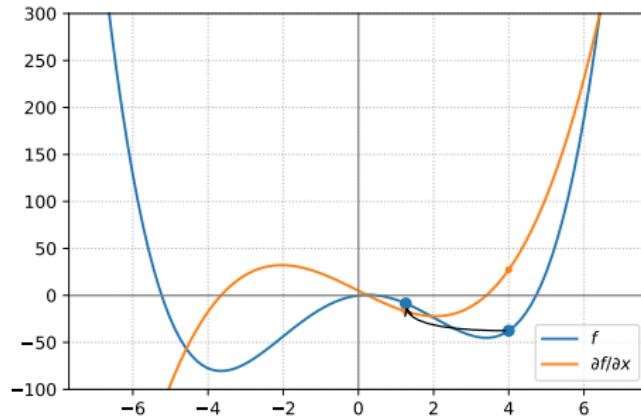
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



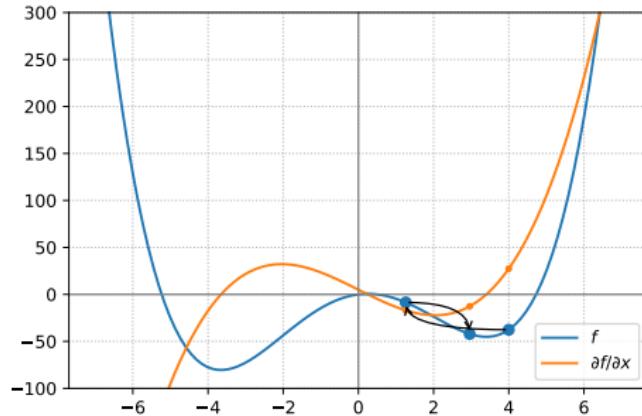
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



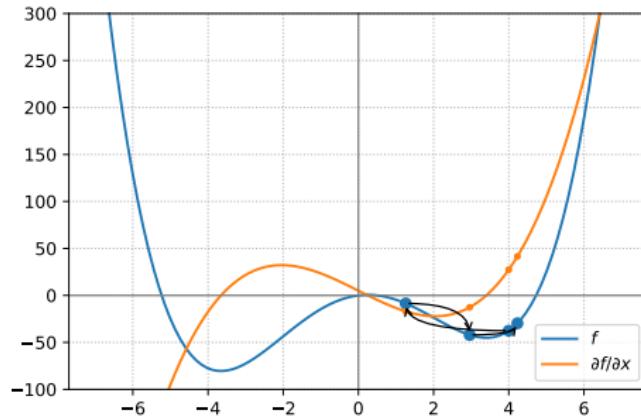
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



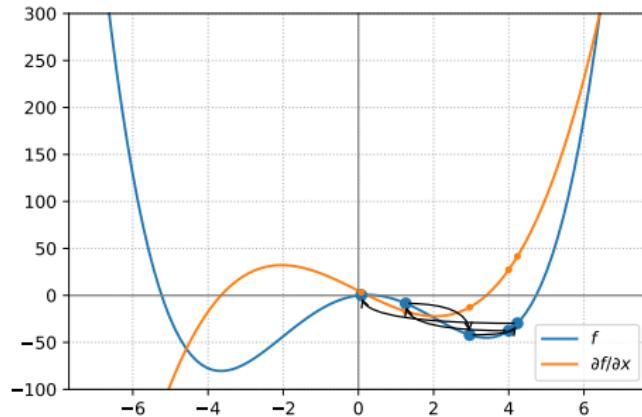
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



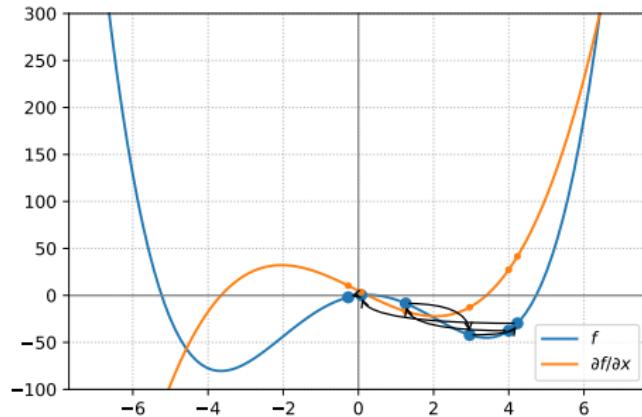
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



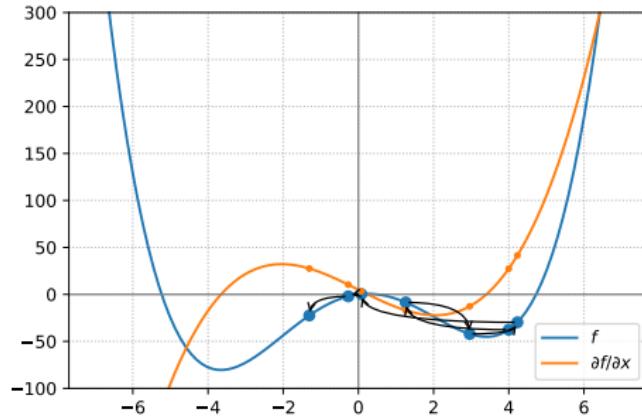
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



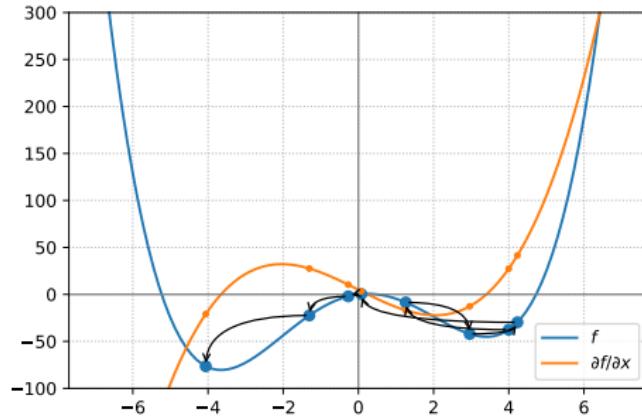
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



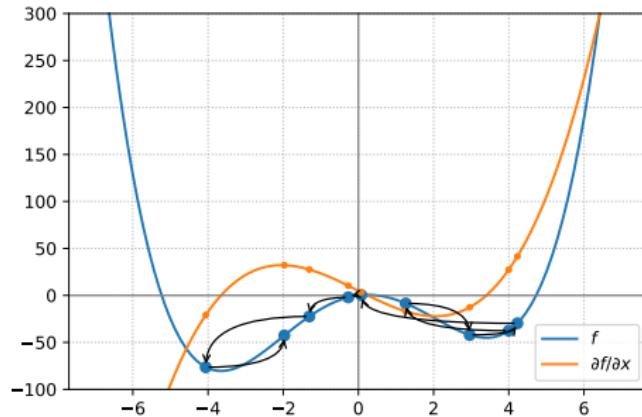
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



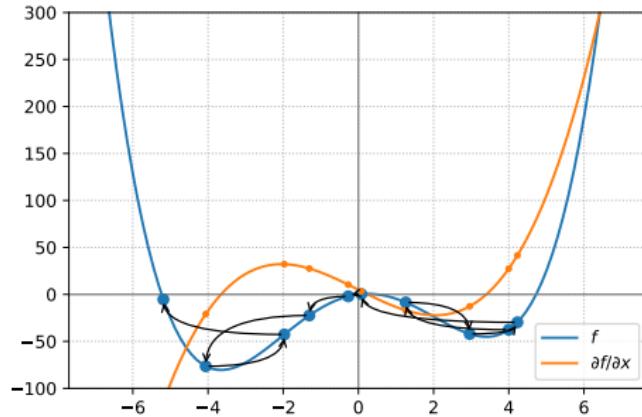
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



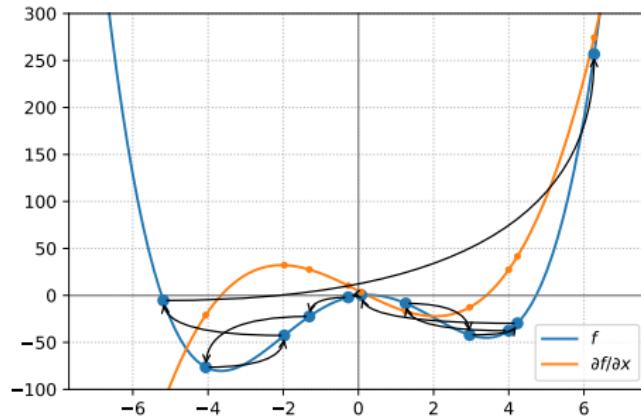
Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



Gradient Descent by Example - exploding gradients

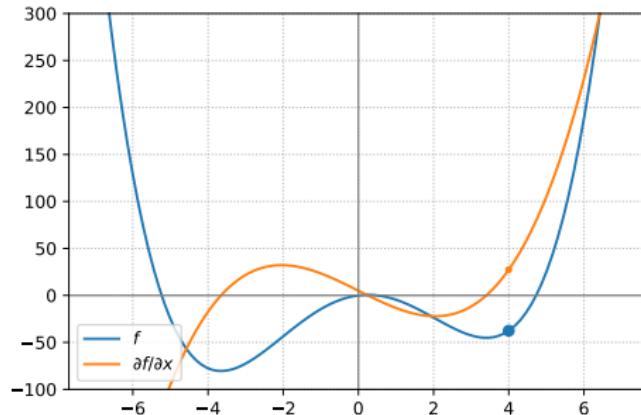
- ▶ But there are very steep functions with large derivatives:



- ▶ In certain situations, the gradients “explode”

Gradient Descent by Example - exploding gradients

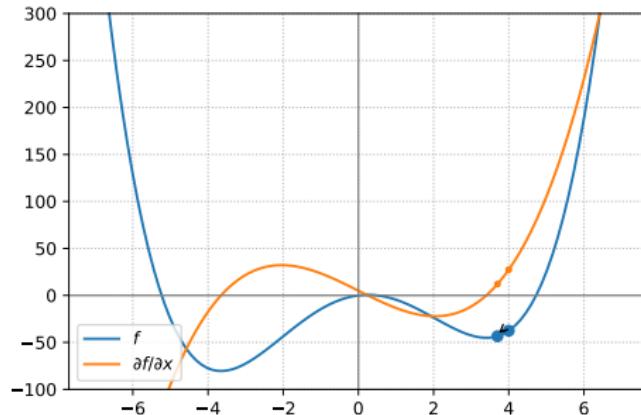
- ▶ But there are very steep functions with large derivatives:



- ▶ In certain situations, the gradients “explode”
- ▶ There are strategies, e.g., gradient clipping, which solve this and similar problems (discussed later)

Gradient Descent by Example - exploding gradients

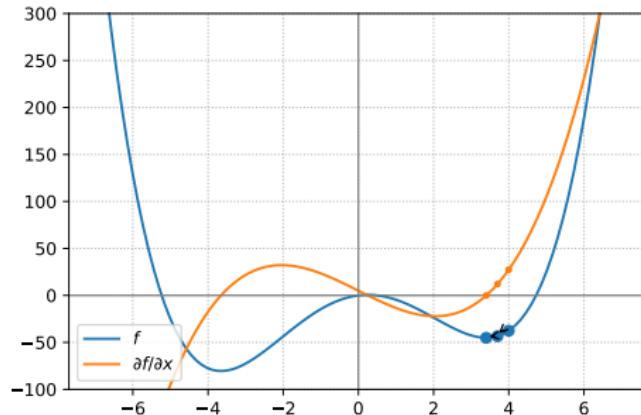
- ▶ But there are very steep functions with large derivatives:



- ▶ In certain situations, the gradients “explode”
- ▶ There are strategies, e.g., gradient clipping, which solve this and similar problems (discussed later)
- ▶ Limit the magnitude of the gradient to e.g., 0.3

Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



- ▶ In certain situations, the gradients “explode”
- ▶ There are strategies, e.g., gradient clipping, which solve this and similar problems (discussed later)
- ▶ Limit the magnitude of the gradient to e.g., 0.3

Gradient Descent by Example

GradientDescent.ipynb

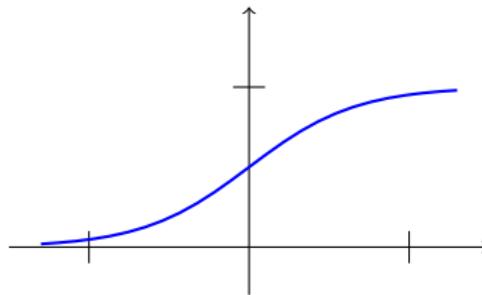
The Sigmoidal function and it's derivative

- ▶ First we need to compute the derivative of the sigmoidal

$$\text{sig}(x) = \frac{1}{1.0 + e^{-x}}$$

$$\text{sig}'(x) = \frac{\partial \text{sig}(x)}{\partial x} =$$

- T** Compute the derivative of the sigmoidal function by hand!
- ▶ Shape of the sigmoidal



The Sigmoidal function and it's derivative

- ▶ First we need to compute the derivative of the sigmoidal

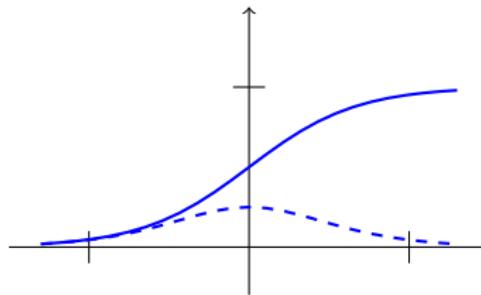
$$\text{sig}(x) = \frac{1}{1.0 + e^{-x}}$$

$$\text{sig}'(x) = \frac{\partial \text{sig}(x)}{\partial x} = \text{sig}(x) \cdot (1 - \text{sig}(x))$$



Compute the derivative of the sigmoidal function by hand!

- ▶ Shape of the sigmoidal and it's derivative (dashed line):



The Sigmoidal function and it's derivative

- ▶ First we need to compute the derivative of the sigmoidal

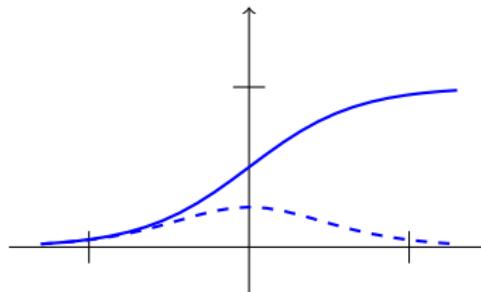
$$\text{sig}(x) = \frac{1}{1.0 + e^{-x}}$$

$$\text{sig}'(x) = \frac{\partial \text{sig}(x)}{\partial x} = \text{sig}(x) \cdot (1 - \text{sig}(x))$$



Compute the derivative of the sigmoidal function by hand!

- ▶ Shape of the sigmoidal and it's derivative (dashed line):



- ▶ A very nice step-by-step explanation can e.g. be found here:

towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

Partial derivative wrt. t_z

- ▶ Network function as computed above

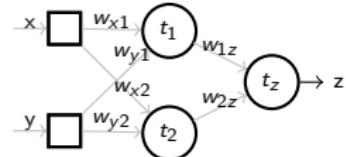
$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z)$$

- ▶ An error (quadratic loss) function based on a given sample:

$$E(x, y, z) = (\mathcal{N}(x, y) - z)^2$$

- ▶ Compute the derivative of E wrt. t_z

$$\frac{\partial E(x, y, z)}{\partial t_z} = \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z}$$



Partial derivative wrt. t_z

- ▶ Network function as computed above

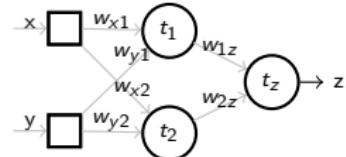
$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z)$$

- ▶ An error (quadratic loss) function based on a given sample:

$$E(x, y, z) = (\mathcal{N}(x, y) - z)^2$$

- ▶ Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_z} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z}\end{aligned}$$



Partial derivative wrt. t_z

- ▶ Network function as computed above

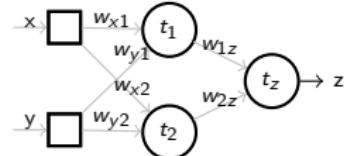
$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z)$$

- ▶ An error (quadratic loss) function based on a given sample:

$$E(x, y, z) = (\mathcal{N}(x, y) - z)^2$$

- ▶ Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_z} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z}\end{aligned}$$



$$\frac{\partial \mathcal{N}_{xy}}{\partial t_z} = \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial i_z(x, y) + t_z}{\partial t_z}$$

Partial derivative wrt. t_z

- ▶ Network function as computed above

$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z)$$

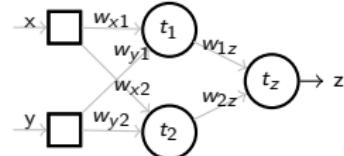
- ▶ An error (quadratic loss) function based on a given sample:

$$E(x, y, z) = (\mathcal{N}(x, y) - z)^2$$

- ▶ Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_z} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_z} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial i_z(x, y) + t_z}{\partial t_z} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy})\end{aligned}$$



Partial derivative wrt. t_z

- ▶ Network function as computed above

$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z)$$

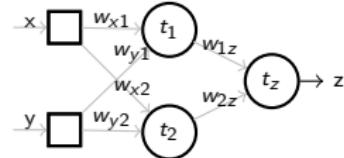
- ▶ An error (quadratic loss) function based on a given sample:

$$E(x, y, z) = (\mathcal{N}_{xy} - z)^2$$

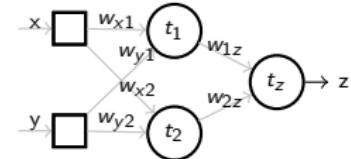
- ▶ Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_z} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy})\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_z} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial i_z(x, y) + t_z}{\partial t_z} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy})\end{aligned}$$



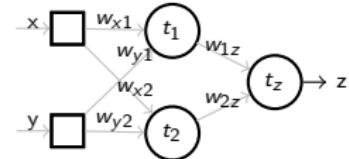
Partial derivative wrt. w_{1z}



$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\frac{\partial E(x, y, z)}{\partial w_{1z}} = \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}}$$

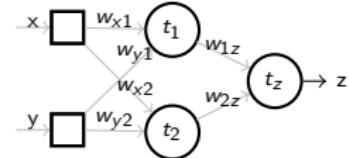
Partial derivative wrt. w_{1z}



$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{1z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}}\end{aligned}$$

Partial derivative wrt. w_{1z}

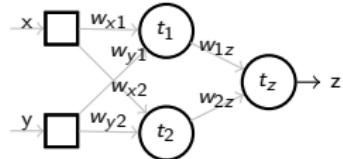


$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{1z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}}\end{aligned}$$

$$\frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} = \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{1z}}$$

Partial derivative wrt. w_{1z}

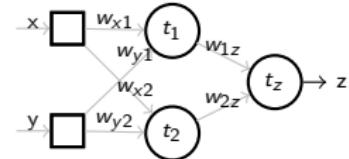


$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{1z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{1z}} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_1(x, y)\end{aligned}$$

Partial derivative wrt. w_{1z}

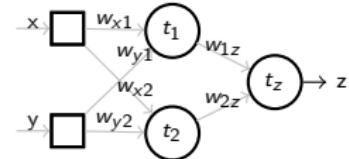


$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{1z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_1(x, y)\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{1z}} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_1(x, y)\end{aligned}$$

Partial derivative wrt. w_{2z}



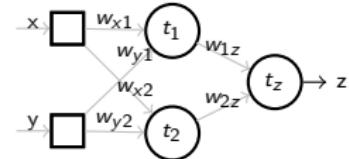
$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{2z}} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial w_{2z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{2z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_2(x, y)\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial w_{2z}} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{2z}} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_2(x, y)\end{aligned}$$

Partial derivative wrt. t_1

$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

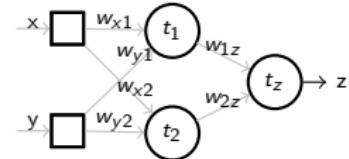


$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

Partial derivative wrt. t_1

$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z)$$

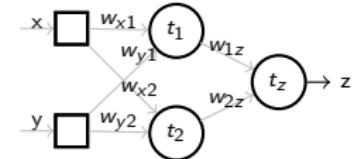
$$= \text{sig}(w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z)$$



$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

$$\frac{\partial \mathcal{N}_{xy}}{\partial t_1} = \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z}{\partial t_1}$$

Partial derivative wrt. t_1

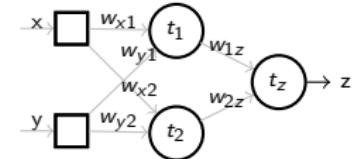


$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial \text{sig}(i_1(x, y) + t_1)}{\partial t_1}\end{aligned}$$

Partial derivative wrt. t_1

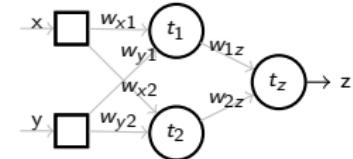


$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial \text{sig}(i_1(x, y) + t_1)}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y)) * \frac{\partial i_1(x, y) + t_1}{\partial t_1}\end{aligned}$$

Partial derivative wrt. t_1

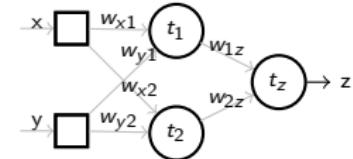


$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial \text{sig}(i_1(x, y) + t_1)}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y)) * \frac{\partial i_1(x, y) + t_1}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y)) * 1\end{aligned}$$

Partial derivative wrt. t_1



$$\begin{aligned}\mathcal{N}_{xy} &= \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_1} &= \frac{\partial(\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y))\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial \text{sig}(i_1(x, y) + t_1)}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y)) * \frac{\partial i_1(x, y) + t_1}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y)) * 1\end{aligned}$$

Take-away-messages of section: “*Training Artificial Neural Networks*”



You should now be able to ...

- ▶ explain the idea behind gradient descent
- ▶ explain how gradient descent can be used to adapt the weights of a neural network
- ▶ derive the equations to adapt the weights and thresholds of a simple network by hand

Künstliche Intelligenz

Markov-Entscheidungsprozesse

Jun.-Prof. Dr.-Ing. Stefan Lüdtke

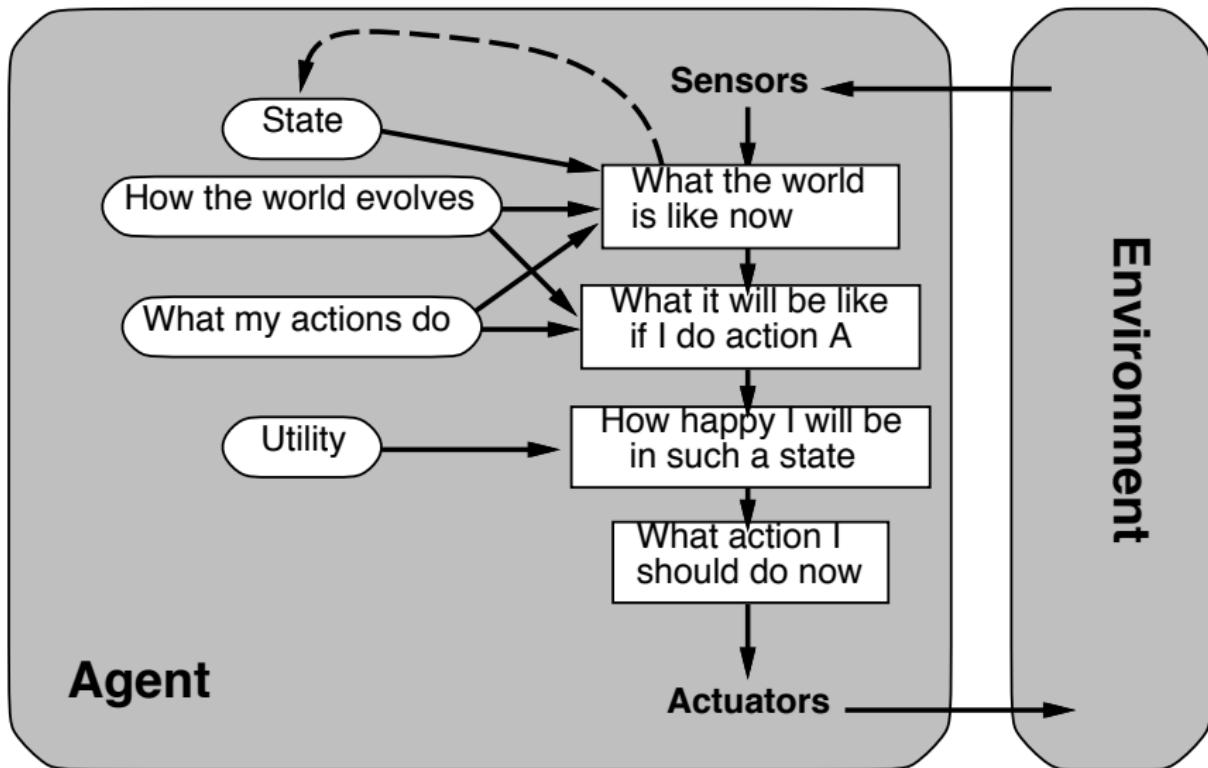
Universität Rostock

Institut für Visual & Analytic Computing

Motivation

- Bisher: Zielbasierte Agenten (\rightarrow Suche, Planung), Agenten in nicht vollständig beobachtbaren Umgebungen (\rightarrow Wahrscheinlichkeitsrechnung), lernende Agenten (\rightarrow Machine Learning)
- Jetzt: Nutzenbasierte Agenten

Nutzenbasierte Agenten



Nutzenbasierte Agenten

Reward

- Am Anfang der Lehrveranstaltung nur sehr allgemeine Beschreibung von nutzenbasierten Agenten, jetzt mehr Details
- Ein nutzenbasierter Agent hat eine *Reward-Funktion* $R : (S \times A) \rightarrow \mathbb{R}$, die beschreibt, wie "gut" es für den Agenten ist, in Zustand s die Aktion a zu wählen
- Reward-Signal ist extern bestimmt: Der Agent kann seine eigene Reward-Funktion nicht umdefinieren
- z.B. für zielbasierte Agenten: Reward +1 bei Zielerreichung, 0 sonst.
- oder z.B. für Spiele: +1 wenn gewonnen, -1 wenn verloren, 0 sonst (wenn Spiel noch nicht vorbei)

Nutzenbasierte Agenten

Dynamik

- Annahme: Nächster Zustand der Welt s_{t+1} hängt nur von aktuellem Zustand der Welt s_t und Aktion des Agenten a_t ab
 - Kann als Wahrscheinlichkeitsverteilung $P(S_{t+1} | S_t, A_t)$ beschrieben werden
- Annahme: Der Agent kennt zu jedem Zeitpunkt den aktuellen Zustand
 - Es gibt auch Methoden um mit nicht-beobachtbarem Zustand umzugehen (Partially Observable Markov Decision Processes), aber die behandeln wir hier nicht (Bei Interesse: Vorlesung AI7 im Master)

Nutzenbasierte Agenten

- Die *Utility* des Agenten ist die Summe aller Rewards (erstmal, wir sehen gleich, dass wir die Definition etwas anpassen müssen)
- Der Agent versucht, Aktionen so zu wählen, dass der Erwartungswert der Utility maximiert wird
- Nennen eine Funktion $\pi : S \rightarrow A$ *Policy*
 - Beschreibt die Aktionsauswahl des Agenten
 - Optimale Policy π^* : Maximiert erwartete Utility

Nutzenbasierte Agenten

Discount Factor

- Oft möchte man eine Verhalten erreichen, bei dem der Agent aktuelle Rewards höher gewichtet als Rewards weit in der Zukunft (Warum ist das sinnvoll?)
- Führen *Discount Factor* $0 < \gamma \leq 1$ ein, und definieren Utility als:

$$U([(s_1, a_1), (s_1, a_1), \dots]) = R(s_1, a_1) + \gamma R(s_2, a_2) + \gamma^2 R(s_3, a_3) + \dots \quad (1)$$

- Auch hilfreich für Lösungsalgorithmen, da unendlich große Utility vermieden wird
- Im Folgenden: Alles noch mal formal...

Markov Decision Processes

An MDP is a 5-tuple (S, A, P, R, γ) with:

- ▶ set S of states.
 - ▶ set A of actions.
 - ▶ $P(S_{t+1} | S_t, A_t)$ specifies the dynamics.
 - ▶ $R(S_t, A_t, S_{t+1})$ specifies the reward at time t .
 - $R(s, a, s')$ is the expected reward received when the agent is in state s , does action a and ends up in state s' .
 - Usually we use $R(s, a) = \sum_{s'} P(s' | s, a)R(s, a, s')$.
 - ▶ γ is discount factor.
 - ▶ I.e., an MDP is a DN with a certain internal structure
- T** Do we need to know $P(S_0)$ – the initial distribution?

Policies

- ▶ A *stationary policy* is a function:

$$\pi : S \rightarrow A$$

Given a state s , $\pi(s)$ specifies what action the agent who is following π will do.

- ▶ An *optimal policy* is one with maximum expected discounted reward.
- ▶ For a fully-observable MDP with stationary dynamics and rewards with infinite or indefinite horizon, there is always an optimal stationary policy.

Example 1: Exercise or not to exercise?

Each week *Sam* has to decide whether to exercise or not:

- ▶ 2 States: $S = \{fit, unfit\}$
- ▶ 2 Actions: $A = \{exercise, relax\}$
- ▶ Dynamics $P(S_{t+1} | S_t, A_t)$:

S_t	A_t	$P(fit State, Action)$
fit	exercise	0.99
fit	relax	0.7
unfit	exercise	0.2
unfit	relax	0.0

- ▶ Reward $R(S_t, A_t, S_{t+1})$ (here independent of S_{t+1}):

S_t	A_t	S_{t+1}	R
fit	exercise	*	8
fit	relax	*	10
unfit	exercise	*	0
unfit	relax	*	5

Example: to exercise or not?

Each week *Sam* has to decide whether to exercise or not:

- ▶ States: $\{fit, unfit\}$
- ▶ Actions: $\{exercise, relax\}$

Q How many stationary policies are there?

Let s be the number of states, and a be the number of actions, then there are s^a possible policies.

Q What are they?

S	π_1	π_2	π_3	π_4
f	e	e	r	r
u	e	r	e	r

Value of a Policy

Given a policy π :

- ▶ $V^\pi(s)$ is the expected discounted reward value of following policy π in state s .
- ▶ $Q^\pi(s, a)$ is the total expected discounted reward value of doing a in state s , then following policy π .
- ▶ V^π and Q^π can be defined mutually recursively:

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

$$Q^\pi(s, a) = \sum_{s'} P(s' | a, s) (R(s, a, s') + \gamma \cdot V^\pi(s'))$$

Value of the Optimal Policy

Let π^* be the optimal policy.

- ▶ $V^*(s)$ is the expected discounted reward value of following policy π^* in state s .
- ▶ $Q^*(s, a)$ is the total expected discounted reward value of doing a in state s , then following policy π^* .
- ▶ V^* and Q^* can be defined mutually recursively:

$$V^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} P(s' | a, s) (R(s, a, s') + \gamma \cdot V^*(s'))$$

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$$

Agenda

Markov Decision Processes

Solution Techniques

Value Iteration

Asynchronous Value Iteration

Policy Iteration

Examples

Agenda

Markov Decision Processes

Solution Techniques

Value Iteration

Asynchronous Value Iteration

Policy Iteration

Examples

Value Iteration

- ▶ Let V_k and Q_k be k -step lookahead value and Q functions.
- ▶ Idea: Given an estimate of the k -step lookahead value function, determine the $k + 1$ step lookahead value function.
- ▶ Set V_0 arbitrarily.
- ▶ Compute Q_{i+1} , V_{i+1} from V_i .
- ▶ This converges exponentially fast (in k) to the optimal value function.

The error reduces proportionally to $\frac{\gamma^k}{1 - \gamma}$

Value Iteration

- ▶ Set $V^{(0)}$ arbitrarily.
- ▶ Compute $V^{(i+1)}$ from $V^{(i)}$:

$$\begin{aligned} V^{(i+1)}(s) &= \sum_{s'} \left(P(s' | \pi^{(i)}(s), s) \left(R(s, \pi^{(i)}(s), s') + \gamma V^{(i)}(s') \right) \right) \\ &\quad \text{or alternatively, if } R(s, a, s') = R(s, a) \\ &= R(s, \pi^{(i)}(s)) + \gamma \sum_{s'} P(s' | \pi^{(i)}(s), s) V^{(i)}(s') \end{aligned}$$

with $\pi^{(i)}$ being the optimal policy wrt. $V^{(i)}$

- ▶ This iteration converges to the optimal value function V^* :

$$\lim_{i \rightarrow \infty} V^{(i)} = V^*$$

Example 1: Exercise or not to exercise?

Each week *Sam* has to decide whether to exercise or not:

- ▶ 2 States: $S = \{fit, unfit\}$
- ▶ 2 Actions: $A = \{exercise, relax\}$
- ▶ Dynamics $P(S_{t+1} | S_t, A_t)$:

S_t	A_t	$P(fit State, Action)$
fit	exercise	0.99
fit	relax	0.7
unfit	exercise	0.2
unfit	relax	0.0

- ▶ Reward $R(S_t, A_t, S_{t+1})$ (here independent of S_{t+1}):

S_t	A_t	S_{t+1}	R
fit	exercise	*	8
fit	relax	*	10
unfit	exercise	*	0
unfit	relax	*	5

Example 1: Exercise or not to exercise? – It depends!

S_t	A_t	$P(\text{fit} \mid S_t, A_t)$	R	Iter.	$V(\text{fit})$	$V(\text{unfit})$
f	e	0.99	8	0	0.000	0.00000
f	r	0.7	10	1	10.000	5.00000
u	e	0.2	0	2	17.650	9.50000
u	r	0.0	5	3	23.812	13.55000
				4	29.338	17.19500
				5	34.295	20.47550
			
				9	49.515	30.62898
				10	52.394	32.56608
			
				49	77.151	49.71368
				50	77.189	49.74231

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

$$Q^\pi(s, a) = \sum_{s'} P(s' \mid a, s) (R(s, a, s') + \gamma \cdot V^\pi(s'))$$

Example 1: Exercise or not to exercise? – It depends!

S_t	A_t	$P(\text{fit} \mid S_t, A_t)$	R
f	e	0.99	8
f	r	0.7	10
u	e	0.2	0
u	r	0.0	5

$$Q_i(s, a) = \sum_{s'} P(s' \mid a, s) (r(s, a, s') + \gamma \cdot V_i(s'))$$

$$\gamma = 0.9$$

$$\pi_i(s) = \operatorname{argmax}_a Q_i(s, a)$$

$$V_{i+1}(s) = \max_a Q_i(s, a) = Q_i(s, \pi_i(s))$$

It.	$V(f)$	$V(u)$	$Q(f, e)$	$Q(f, r)$	$Q(u, e)$	$Q(u, r)$	$\pi(f)$	$\pi(u)$
0	0.00	0.00	8.00	10.00	0.00	5.00	r	r
1	10.00	5.00	16.95	17.65	5.40	9.50	r	r
2	17.65	9.50	23.81	23.68	10.01	13.55	e	r
3	23.81	13.55	29.33	28.66	14.04	17.19	e	r
4	29.33	17.19	34.29	33.12	17.66	20.47	e	r
5	34.29	20.47	38.74	37.13	20.91	23.42	e	r
			...					
9	49.51	30.62	52.39	49.46	30.96	32.56	e	r
10	52.39	32.56	54.97	51.80	32.87	34.30	e	r
			...					
49	77.15	49.71	77.18	72.02	49.68	49.74	e	r
50	77.18	49.74	77.22	72.06	49.70	49.76	e	r

Example 1: Exercise or not to exercise? – It depends!

- ▶ But the resulting policy does also depend on the discount factor γ for $i = 10$:

γ	$V^i(f)$		$V^i(u)$		$Q^i(f, e)$	$Q^i(f, r)$	$Q^i(u, e)$	$Q^i(u, r)$	$\pi^i(f)$	$\pi^i(u)$
0.2	12.0	6.2			10.4	12.0	1.4	6.2	r	r
0.5	17.6	9.9			16.8	17.6	5.7	9.9	r	r
0.9	52.3	32.5			54.9	51.8	32.8	34.3	e	r
0.95	64.7	40.1			69.3	64.5	42.8	43.1	e	r
0.99	77.4	48.1			84.3	77.9	53.4	52.7	e	e

- ▀ What does this mean?
Discuss the different results!

Zusammenfassung

- Ein MDP ist über das Transitionsmodell und die Reward-Funktion spezifiziert
- Die Utility eines MDP-Agenten ist der summierte discounted Reward
- Eine Lösung eines MDP ist eine Policy, die jedem Zustand eine Entscheidung zuweist. Die optimale Policy maximiert die erwartete Utility
- Die Utility eines Zustands ist die erwartete Utility der Zustandssequenz, wenn die optimale Policy von diesem Zustand ausgehend ausgeführt wird
- *Value Iteration* löst ein MDP, indem iterativ die Gleichungen gelöst werden, die die Utility eines Zustands mit der Utility der Nachbarn verbinden

Künstliche Intelligenz

18. Reinforcement Learning

Jun.-Prof. Dr.-Ing. Stefan Lüdtke
Institut für Visual & Analytic Computing
Universität Rostock

Learning Objectives - Reinforcement Learning

At the end of the class you should be able to:

- Explain the relationship between decision-theoretic planning (MDPs) and reinforcement learning

Learning Objectives - Reinforcement Learning

At the end of the class you should be able to:

- Explain the relationship between decision-theoretic planning (MDPs) and reinforcement learning
- Implement basic state-based reinforcement learning algorithms: Q-learning

Reinforcement Learning

What should an agent do given:

- Prior knowledge
- Observations
- Goal

Reinforcement Learning

What should an agent do given:

- Prior knowledge possible states of the world
 possible actions
- Observations
- Goal

Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world
 possible actions
- **Observations** current state of world
 immediate reward / punishment
- **Goal**

Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world
possible actions
- **Observations** current state of world
immediate reward / punishment
- **Goal** act to maximize accumulated (discounted) reward

Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world
possible actions
- **Observations** current state of world
immediate reward / punishment
- **Goal** act to maximize accumulated (discounted) reward
- Like decision-theoretic planning, except model of dynamics
and model of reward not given.

Reinforcement Learning Examples

- Game -

Reinforcement Learning Examples

- Game - reward winning, punish losing

Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog -

Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behavior

Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behavior
- Robot -

Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behavior
- Robot - reward task completion, punish dangerous behavior

Experiences

- We assume there is a sequence of experiences:

state, action, reward, state, action, reward,

Experiences

- We assume there is a sequence of experiences:

state, action, reward, state, action, reward,

- The sequence of experiences up to the time the agent has to choose its action is its **history**
- The agent has to choose its action as a function of its history.

Experiences

- We assume there is a sequence of experiences:

state, action, reward, state, action, reward,

- The sequence of experiences up to the time the agent has to choose its action is its **history**
- The agent has to choose its action as a function of its history.
- At any time it must decide whether to

Experiences

- We assume there is a sequence of experiences:

state, action, reward, state, action, reward,

- The sequence of experiences up to the time the agent has to choose its action is its **history**
- The agent has to choose its action as a function of its history.
- At any time it must decide whether to
 - ▶ **explore** to gain more knowledge
 - ▶ **exploit** knowledge it has already discovered

Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
 - ▶ The dog is expected to determine that eating the shoe at the start of the day is what was responsible for it being scolded at the end of the day.

Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
 - ▶ The dog is expected to determine that eating the shoe at the start of the day is what was responsible for it being scolded at the end of the day.
- The long-term effect of an action depend on what the agent will do in the future.
 - ▶ It might be okay for a robot to create a mess as long as it cleans up after itself.

Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
 - ▶ The dog is expected to determine that eating the shoe at the start of the day is what was responsible for it being scolded at the end of the day.
- The long-term effect of an action depend on what the agent will do in the future.
 - ▶ It might be okay for a robot to create a mess as long as it cleans up after itself.
- The explore-exploit dilemma: at each time should the agent be greedy or inquisitive?

Reinforcement learning: main approaches

- search through a space of policies (controllers)

Reinforcement learning: main approaches

- search through a space of policies (controllers)
- learn a model consisting of state transition function $P(s'|a, s)$ and reward function $R(s, a)$; solve this as an MDP.

Reinforcement learning: main approaches

- search through a space of policies (controllers)
- learn a model consisting of state transition function $P(s'|a, s)$ and reward function $R(s, a)$; solve this as an MDP.
- learn $Q^*(s, a)$, use this to guide action.

Batch Learning

Assume that there is a sequence of transition samples $(s, a, r, s')_{1:N}$. A transition sample is also called *experience*. Clearly, we can use a sequence of experiences to create an empirical model:

$$\hat{\mathcal{S}} := \{s_i\} \cup \{s'_i\} \quad (1)$$

$$\hat{\mathcal{A}} := \{a_i\} \quad (2)$$

$$\hat{\mathcal{P}}_{ss'}^a := \frac{N_{ss'}^a}{N_s^a} \quad (3)$$

$$\hat{\mathcal{R}}_{ss'}^a := \frac{1}{N_{ss'}^a} \sum_{i=1}^N r_i [s_i = s \wedge a_i = a \wedge s'_i = s'] \quad (4)$$

$$\text{where } N_{ss'}^a := \sum_{i=1}^N [s_i = s \wedge a_i = a \wedge s'_i = s'] \quad (5)$$

$$N_s^a := \sum_{s' \in \hat{\mathcal{S}}} N_{ss'}^a \quad (6)$$

We can then use this empirical model for value or policy iteration.

Batch Learning

- The samples $(s, a, r, s')_{1:N}$ can be generated by an arbitrary policy, as long as this policy guarantees that, as $N \rightarrow \infty$, every action will be tried infinitely often in any state.
- Note that for trivial problems where transition model and reward structure are deterministic, only one sample is required for each combination of a and s (each sample representing an edge in the transition graph), and value iteration becomes the Bellman-Ford algorithm.

Q Learning

Consider the Bellman optimality equation for Q^* :

$$Q^*(s, a) = \sum_{s'} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma \max_{a'} Q^*(s', a') \right)$$

Now consider an experience (s, a, r, \tilde{s}') . Using this tuple, we can first build very coarse approximations of $\mathcal{P}_{ss'}^a = p(s' | s, a)$ and $\mathcal{R}_{ss'}^a$ by defining

$$\mathcal{P}_{ss'}^a \approx [s' = \tilde{s}'] \quad (9)$$

$$\mathcal{R}_{ss'}^a \approx r \quad (10)$$

entering this into the equations above gives

$$Q^*(s, a) \approx \sum_{s'} [s' = \tilde{s}'] \left(r + \gamma \max_{a'} Q^*(s', a') \right) \quad (11)$$

Q Learning

$$\begin{aligned} Q^*(s, a) &\approx \sum_{s'} [s' = \tilde{s}'] \left(r + \gamma \max_{a'} Q^*(s', a') \right) \\ &= r + \gamma \max_{a'} Q^*(\tilde{s}', a') \end{aligned} \tag{12}$$

Where this quantity is sometimes known as *target*.

Due to approximations, this target value may contain errors. But imagine, $Q_{i-1}(s, a)$ is a previous approximation. From this approximation and the target created from experience (s, a, r, s') we can create an updated approximation by using a weighted average of both values:

$$Q_{i+1}(s, a) \leftarrow (1 - \alpha)Q_i(s, a) + \alpha \left(r + \gamma \max_{a'} Q_i(s', a') \right) \tag{13}$$

Q Learning

Definition 1 (*Q* Learning)

$$Q_{i+1}(s, a) \leftarrow (1 - \alpha)Q_i(s, a) + \alpha(r + \gamma \max_{a'} Q_i(s', a')) \quad (14)$$

Sometimes, this is written as

$$Q_{i+1}(s, a) \leftarrow Q_i(s, a) + \alpha(r + \gamma \max_{a'} Q_i(s', a') - Q_i(s, a)) \quad (15)$$

This version emphasizes the idea that the new action value is the old value plus a move of step size α in the direction of the *difference* between the *old* value and the target created by the *new experience* (i.e., we move the action value gradually to the target using the temporal difference).

Q Learning

Definition 2 (Learning rate)

In equations (14) and (15), the value α , where $0 \leq \alpha \leq 1$ is the *learning rate*.

Proposition 3 (Convergence of *Q* learning)

If α is appropriately adjusted over time such that $\lim_{t \rightarrow \infty} \alpha = 0$, the *Q* learning algorithm will converge to Q^* .

Note 4

Finding the right adjustment schedule is problem dependent.
Fortunately, π^* will (usually) be discovered long before Q^* has been reached.

Action Selection

- Q -learning has the interesting property that it will converge to the true optimal policy (and the true optimal action value function), regardless of policy, as long as the policy will try every action infinitely often in any state.
- However, in order to optimize reward, it seems advisable to start using actions with high value as soon as possible. This is known as the *exploration-exploitation dilemma*: if one starts to greedily use the currently optimal moves, action options leading to maybe even better results will never be explored.

Epsilon-greedy Action Selection

One simple solution is to *always* reserve a probability of ϵ (e.g., $\epsilon = 0.01$) for experimentally trying a non-optimal action. $\pi(s, a)$ can in this case be defined by:

$$\pi(s, a) = \begin{cases} 1 - \epsilon & \text{if } a \text{ optimal} \\ \epsilon/(n - 1) & \text{otherwise} \end{cases} \quad (16)$$

Where $n > 1$ is the number of actions available in state s (if $n = 1$ there is obviously nothing to explore).

Boltzmann action selection

A more involved action selection strategy uses the definition

$$\pi(s, a) = \frac{\exp(\lambda Q(s, a))}{\sum_{a'} \exp(\lambda Q(s, a'))} \quad (17)$$

$\lambda > 0$ (alternatively: $1/\tau$, where τ is known as "temperature") determines the influence of the Q values on action selection.

- If λ is small (resp. if the temperature τ is high), all actions are almost equiprobable.
- If λ is large (low temperature), small differences in Q will create large differences in probability (preferring actions with larger Q values).

The probability in (17) is known as Gibbs or Boltzmann distribution; this is an example of a so called *softmax* action selection strategy.

Problems with Q-learning

- It does one backup between each experience.
 - ▶ Is this appropriate for a robot interacting with the real world?
 - ▶ An agent can make better use of the data by
 - remember previous experiences and use these to update model (action replay)
 - building a model, and using MDP methods to determine optimal policy.
 - doing multi-step backups
- It learns separately for each state.

Other RL Algorithms

- Explicitly maintaining the Q-Function as a table becomes infeasible for problems with large (or continuous) state or action spaces
 - Can use a discretization of states and/or actions
 - Or, use a function approximator for Q , e.g. a deep neural network (“deep Q learning”)
- Q-Learning is just one, simple example of reinforcement learning algorithms, other algorithms differ w.r.t.
 - Number of steps taken before updating Q
 - Policy used for generating experiences (“on-policy” vs. “off-policy” RL)
 - Learning Q vs. directly learning π
 - ...

Summary

- RL algorithms allow an agent to learn taking actions in an unknown environment by learning from experience
- Challenging due to large state and action spaces, sparse rewards
- Active and exciting field of research
- We regularly offer team projects and master theses on applying RL to solve real-world problems

Künstliche Intelligenz

AI Safety

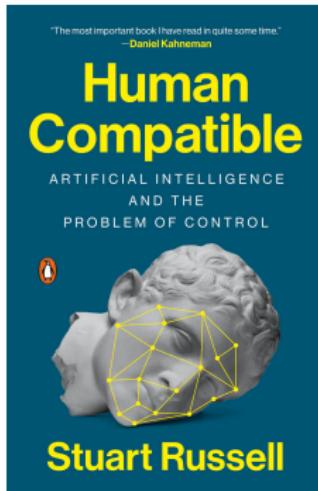
Jun.-Prof. Dr.-Ing. Stefan Lüdtke

Universität Rostock

Institut für Visual & Analytic Computing

Today's Topic

- What we learned about RL so far is insufficient
- How can we develop AL systems which do not kill us eventually?



Progress in AI



Progress in AI

- Rule of thumb: Can automate everything humans can do in < 1 seconds
- Robots acting in the real world (→ Boston Dynamics)
- Huge progress in visual perception, speech recognition, translation (Deep Learning)
- Even without any more fundamental breakthroughs, current AI technology will have huge impact on:
 - Autonomous Robots on streets and in homes
 - Intelligent assistants (think: Alexa + intelligence)
 - Question Answering, Integration of knowledge and reasoning instead of information retrieval

Progress in AI

- On the other hand: Progress in some fields not as fast as expected
- Current ML methods require tremendous amounts of data
- Training of large models bottlenecked by available data (instead of available compute)
- Not clear whether we can fundamentally solve this problem with current AI technology
- Possible: Failure in some of the promises (autonomous driving) of AI could lead to new AI Winter

Human-Level AI

- What's missing:
- True understanding of language (LLMs are stochastic parrots)
- Integration of Learning and A-Priori-Knowledge (→ Human brain has lots of priors: Common-Sense-Knowledge, naive physics)
- Planning and Reasoning on large time scales ("How do I need to move my hand to grab a glass" vs. "What do I eat for lunch tomorrow" vs. "Which steps do I need to take for traveling to the Maldives next year")
- Might require conceptual breakthroughs with unclear timeline
- Probably, scale is not all you need

Advantages of Human-Level AI

- Our civilization, wealth almost entirely the result of intelligence
- More intelligence → More civilization, wealth, ...

Disadvantages of Human-Level AI



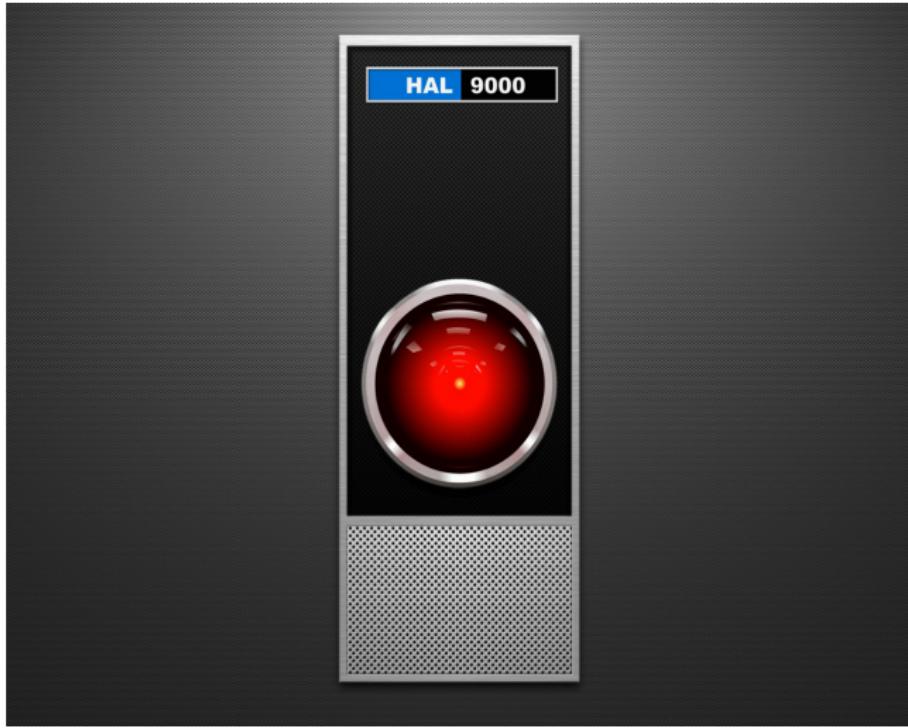
Disadvantages of Human-Level AI

- “*We had better be quite sure that the purpose put into the machine is the purpose which we really desire*” (Norbert Wiener, 1960)
- King Midas problem: Wrong goal
- Off-Switch problem: “You can't fetch the coffee when you're dead”

Paperclip Maximizer

Suppose we have an AI whose only goal is to make as many paper clips as possible. The AI will realize quickly that it would be much better if there were no humans because humans might decide to switch it off. Because if humans do so, there would be fewer paper clips. Also, human bodies contain a lot of atoms that could be made into paper clips. The future that the AI would be trying to gear towards would be one in which there were a lot of paper clips but no humans.

(Nick Bostrom, 2003)



I'm sorry Dave, I'm afraid I can't do that

What went wrong?

- Human intelligence: Humans are intelligent to the extent that *our* actions lead to achieving *our* goals

What went wrong?

- Human intelligence: Humans are intelligent to the extent that *our* actions lead to achieving *our* goals
- Artificial Intelligence: Machines are intelligent to the extent that *their* actions lead to achieving *their* goals

What went wrong?

- Human intelligence: Humans are intelligent to the extent that *our* actions lead to achieving *our* goals
- Artificial Intelligence: Machines are intelligent to the extent that *their* actions lead to achieving *their* goals
 - But: Goals are programmed by us: Costs, reward functions, utility functions, ...

What went wrong?

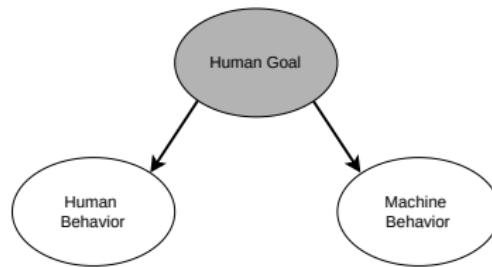
- Human intelligence: Humans are intelligent to the extent that *our* actions lead to achieving *our* goals
- Artificial Intelligence: Machines are intelligent to the extent that *their* actions lead to achieving *their* goals
 - But: Goals are programmed by us: Costs, reward functions, utility functions, ...
- *Maybe that's a mistake: King Midas problem*
- Instead: Machines are *beneficial* to the extent that *their* actions lead to achieving *our* goals

How does this work: Three ideas

- Only goal of AI is to maximize the realization of human preferences
- The AI is uncertain about these preferences
- The AI gets information about human preferences from human behavior

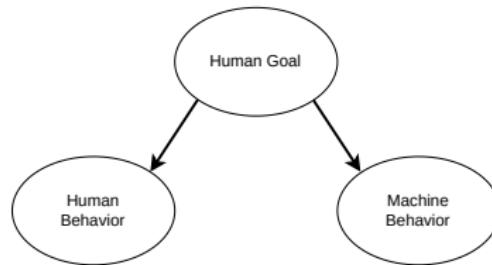
Graphical Model: Current Approach

- Assume that human goal is known (observed)
- Human behavior and machine behavior are independent
- Machine behavior cannot be influenced by human behavior



Graphical Model: New Approach

- Human goal not observed
- Machine needs to infer human goal
- Uses observed human behavior for inference



The Off-Switch Problem, Revisited

- Current AI

The Off-Switch Problem, Revisited

- Current AI
 - I have to fetch coffee

The Off-Switch Problem, Revisited

- Current AI
 - I have to fetch coffee
 - I cannot fetch coffee when I'm dead

The Off-Switch Problem, Revisited

- Current AI

- I have to fetch coffee
- I cannot fetch coffee when I'm dead
- Therefore, I should disable my off-switch

The Off-Switch Problem, Revisited

■ Current AI

- I have to fetch coffee
- I cannot fetch coffee when I'm dead
- Therefore, I should disable my off-switch
- (and also kill all customers in Starbucks who are in my way, to fetch the coffee as fast as possible)

The Off-Switch Problem, Revisited

- Current AI

- I have to fetch coffee
- I cannot fetch coffee when I'm dead
- Therefore, I should disable my off-switch
- (and also kill all customers in Starbucks who are in my way, to fetch the coffee as fast as possible)

- Beneficial AI

The Off-Switch Problem, Revisited

- Current AI

- I have to fetch coffee
- I cannot fetch coffee when I'm dead
- Therefore, I should disable my off-switch
- (and also kill all customers in Starbucks who are in my way, to fetch the coffee as fast as possible)

- Beneficial AI

- I have to fetch coffee

The Off-Switch Problem, Revisited

- Current AI

- I have to fetch coffee
- I cannot fetch coffee when I'm dead
- Therefore, I should disable my off-switch
- (and also kill all customers in Starbucks who are in my way, to fetch the coffee as fast as possible)

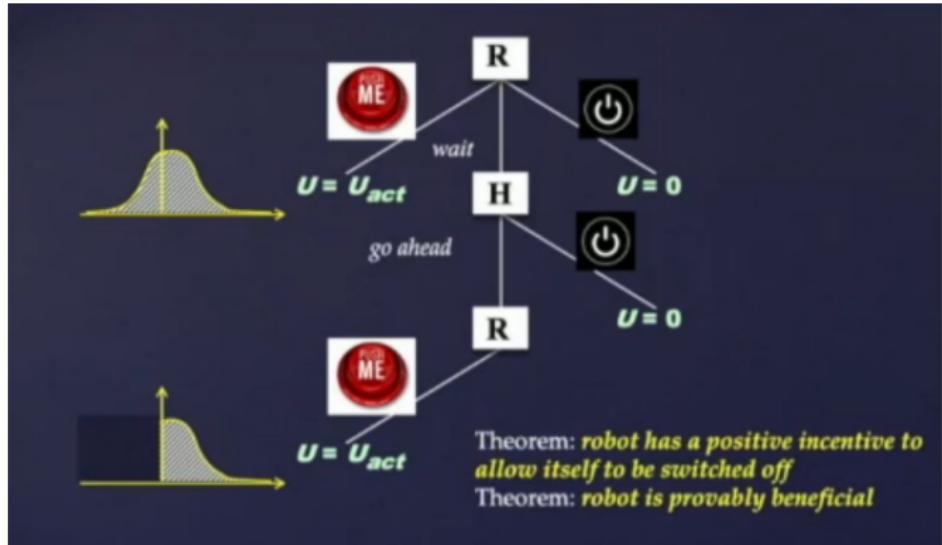
- Beneficial AI

- I have to fetch coffee
- I might be doing something that is in conflict with human preferences ("negative reward")

The Off-Switch Problem, Revisited

- Current AI
 - I have to fetch coffee
 - I cannot fetch coffee when I'm dead
 - Therefore, I should disable my off-switch
 - (and also kill all customers in Starbucks who are in my way, to fetch the coffee as fast as possible)
- Beneficial AI
 - I have to fetch coffee
 - I might be doing something that is in conflict with human preferences ("negative reward")
 - Therefore, I should allow the human to switch me off, so that the reward can be at least 0.

Assistance Game



Current Research: Humans are no Rational Agents

- Limited computational power
- Emotionally caused behavior
- Uncertainty over own preferences
- Variability of preferences (by machines)

Current Research: Multiple People

- Different preferences
- Individual loyalty vs. global utilitarianism
- Comparability of inter-personal preferences (same scales?)

Altruismu, Indifference, Sadism

- Assume there are two people, Alice and Bob
- Intrinsic Welfare: w_A and w_B
- Overall happiness: Own welfare + other's welfare:
 - $U_A = w_A + C_{AB} w_B$
 - $U_B = w_B + C_{BA} w_A$
- Altruismu, Indifference, Sadism depend on C ("caring") factors
- If $C_{AB} = 0$, optimum of $U_A + U_B$ typically gives Alice more intrinsic welfare, but Bob can overall be happier (e.g., Alice is newborn baby of Bob)
- Not clear how to handle $C_{AB} < 0$? Maybe robot should ignore those terms?

Pride and Envy

- Positional goods: Value (= welfare caused by it) of goods depends on comparison to other people's goods
- $$U_A = w_A + C_{AB} w_B + E_{AB}(w_B - w_A) + P_{AB}(w_A - w_B)$$
- $$= (1 + E_{AB} + P_{AB})w_A + (C_{AB} - E_{AB} - P_{AB})w_B$$
- I.e., pride and envy are mathematically identical to sadism
- But, ignoring these terms might not be sensible, because they seem to be fundamental for human behavior

Summary

- AI is making fast progress and might eventually outperform human capabilities

Summary

- AI is making fast progress and might eventually outperform human capabilities
- Current approach (fixed goals / reward functions / costs) can lead to unexpected consequences

Summary

- AI is making fast progress and might eventually outperform human capabilities
- Current approach (fixed goals / reward functions / costs) can lead to unexpected consequences
- Therefore: Allow uncertainty over human preferences: *Provably beneficial AI*

Summary

- AI is making fast progress and might eventually outperform human capabilities
- Current approach (fixed goals / reward functions / costs) can lead to unexpected consequences
- Therefore: Allow uncertainty over human preferences: *Provably beneficial AI*
- These ideas should become standard for AI developers (bridge engineering always includes safety, and so should AI engineering)