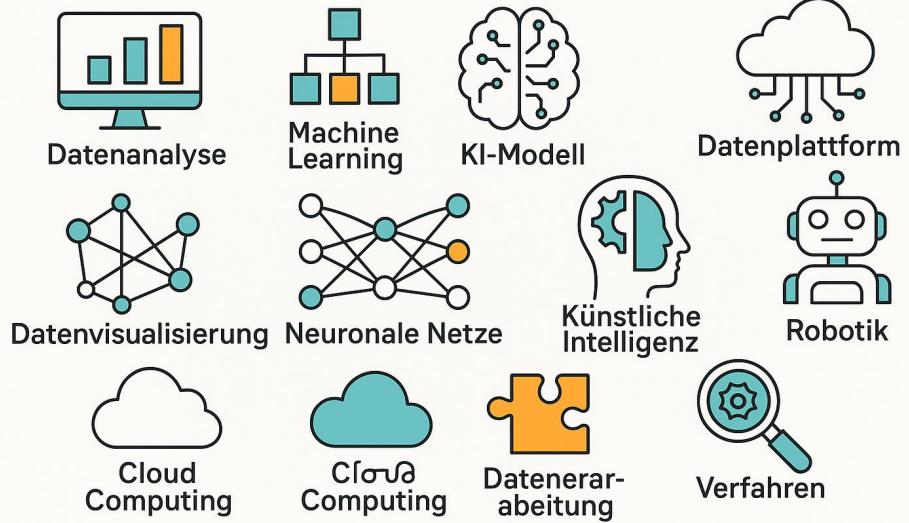


BIG DATA & DATA SCIENCE



Big Data & Data Science

Verschiedene KI-Technologien und Ansätze

WS 2025/26

Prof. Dr. Klemens Waldhör

- GAN - Generative Adversarial Networks
- Agentenbasierte KI Systeme
- Reinforcement Learning
- Problemlösen mittels Suchverfahren
- Transformer-Modelle
- Agentic AI

Real image



Reconstructed images



Blonde

Bangs

Smile

Male

Perarnau G, van de Weijer J, Raducanu B, Álvarez JM (2016) Invertible Conditional GANs for image editing.

GAN - Generative Adversarial Networks

GAN nach Ian Goodfellow et al. (2014)

- Generierung von Bildern mit Hilfe zweier künstlicher neuronaler Netze
 - Generator G und Diskriminatator D
 - Konzipiert als Zwei-Personen-Nullsummenspiel
 - Generator G generiert künstliche Daten
 - Diskriminatator D lernt künstliche von gefälschten Bildern zu unterscheiden
 - G lernt durch Feedback von D „bessere“ Bilder zu generieren
- G versucht Daten (Bilder) zu generieren, die D nicht als gefälscht erkennt
- D versucht zu erkennen, ob Daten (Bilder) echt oder gefälscht sind

Entwicklungen in der GAN-Forschung

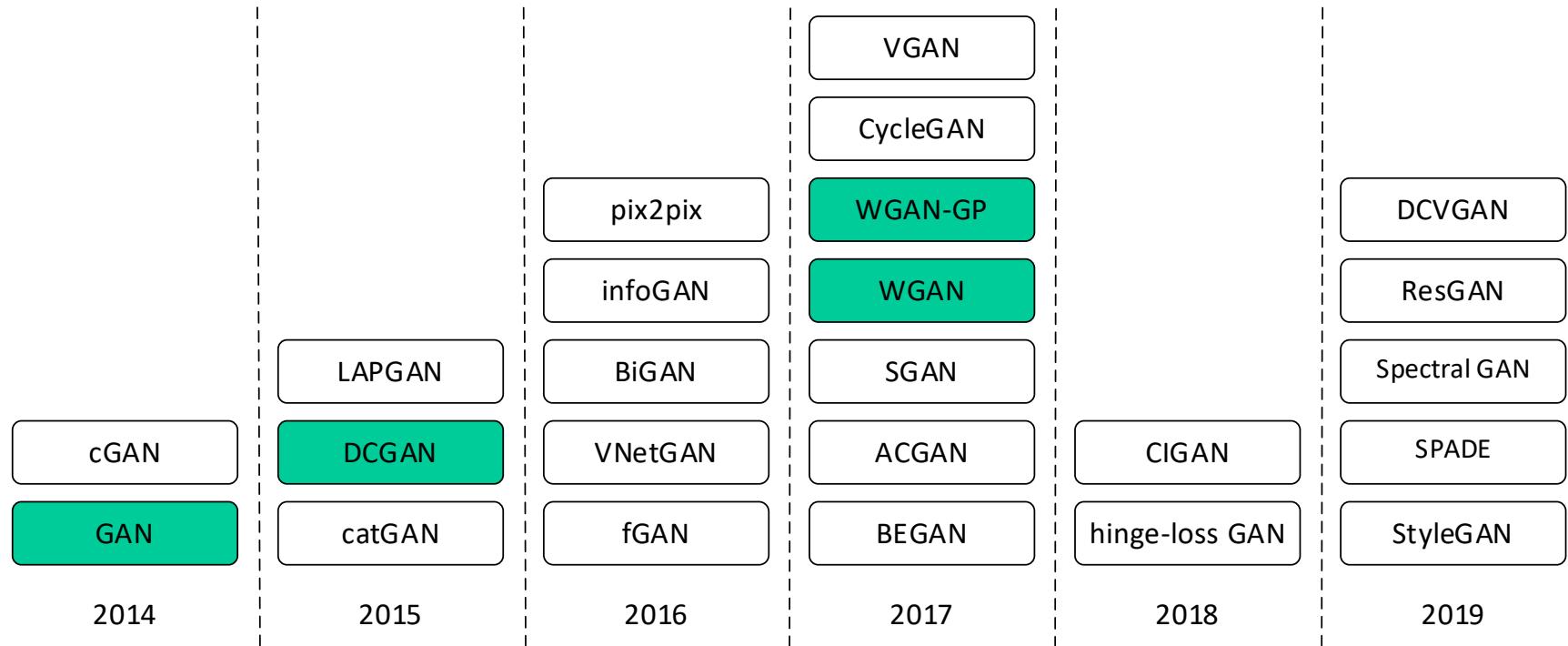


Abbildung 1: Wichtige Entwicklungen in der GAN-Forschung
In Anlehnung an Hajarolasvadi et al., 2020, S. 218524

Quelle: Täuber JG (2021) Generative Adversarial Networks: Eine empirische Untersuchung der Potenziale und Grenzen etablierter GAN-Architekturen bei der Generierung fotorealistischer Bilder. Bachelorthesis, Nürnberg.

Aufbau

- Trainingsphase für das Diskriminatornetz (Szenario 1) und eine für das Generatornetz (Szenario 2).
- Diskriminator wird trainiert eine aus einem Trainingsdatensatz stammende Probe x möglichst korrekt zu klassifizieren
- Im 2. Szenario treten Generator und Diskriminator gegeneinander an. Generator erzeugt künstliche Proben aus Eingangsrauschwerten
- D werden nun die von G erzeugten Fälschungen zur Klassifizierung vorgelegt.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

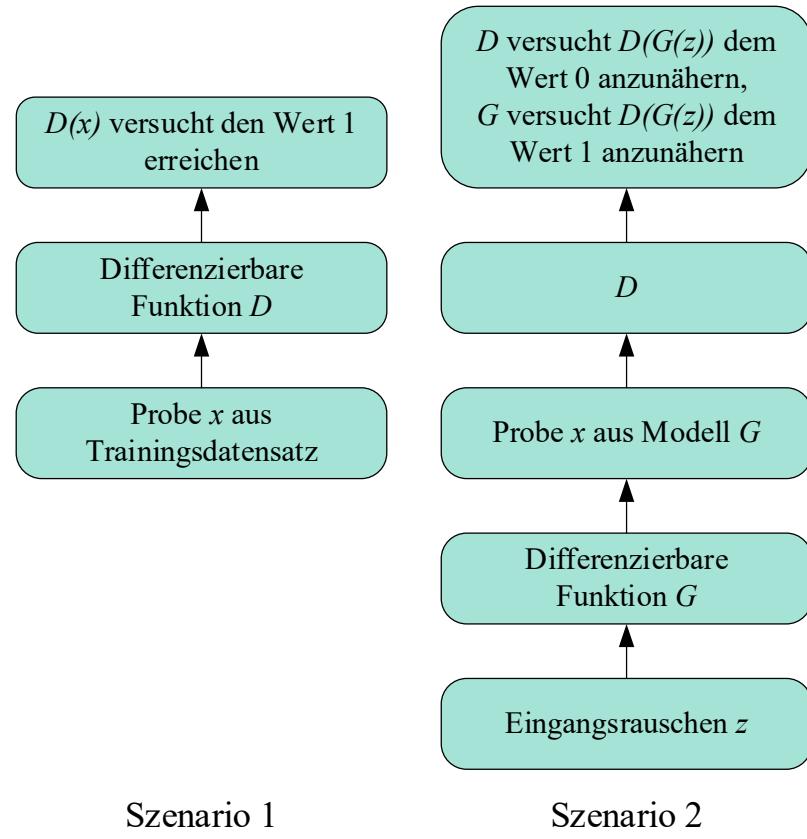


Abbildung 2: Das GAN-Minimax-Spiel
Eigene Übersetzung, in Anlehnung an Goodfellow, 2016a, S. 19

Quelle: Täuber JG (2021) Generative Adversarial Networks: Eine empirische Untersuchung der Potenziale und Grenzen etablierter GAN-Architekturen bei der Generierung fotorealistischer Bilder. Bachelorthesis, Nürnberg.

- Beispiele für Bilddatensätze generieren
- Fotografien von menschlichen Gesichtern erzeugen
- Realistische Fotografien erzeugen
- Cartoon-Charaktere generieren
- Image-to-Image Übersetzung
- Text-to-Image Übersetzung
- Semantisches Bild-zu-Foto-Mapping
- Frontalansicht eines Gesichts erzeugen
- Neue menschliche Posen generieren
- Fotos in Emojis umwandeln
- Fotobearbeitung Gesichtsalterung
- Foto-Blending Super-Resolution (Hochskalierung)
- Photo-Inpainting (Auffüllen fehlender Bildbereiche)
- Kleidungstransfer / Clothing Translation
- Videovorhersage
- 3D-Objekterzeugung

- Gesichtsgenerierung



2014

2015

2016

2017

- Brundage M, Avin S, Clark J, Toner H, Eckersley P, Garfinkel B, Dafoe A, Scharre P, Zeitzoff T, Filar B, Anderson H, Roff H, Allen GC, Steinhardt J, Flynn C, hÉigeartaigh SÓ, Beard S, Belfield H, Farquhar S, Lyle C, Crotoof R, Evans O, Page M, Bryson J, Yampolskiy R, Amodei D (2018) The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation.

- Gesichtsgenerierung



- Karras T, Aila T, Laine S, Lehtinen J (2017) Progressive Growing of GANs for Improved Quality, Stability, and Variation.

- Objekterzeugung



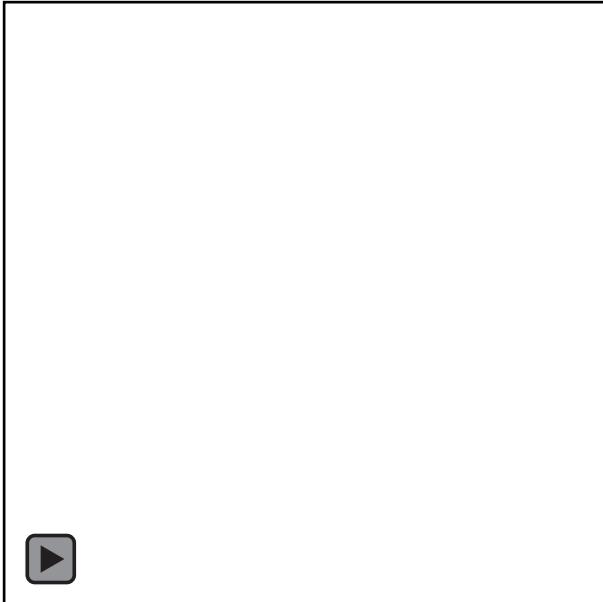
- Karras T, Aila T, Laine S, Lehtinen J (2017) Progressive Growing of GANs for Improved Quality, Stability, and Variation.

- Posengenerierung



- Ma L, Jia X, Sun Q, Schiele B, Tuytelaars T, van Gool L (2017) Pose Guided Person Image Generation.

WGAN-GP (30 Epochen)



fake_samples

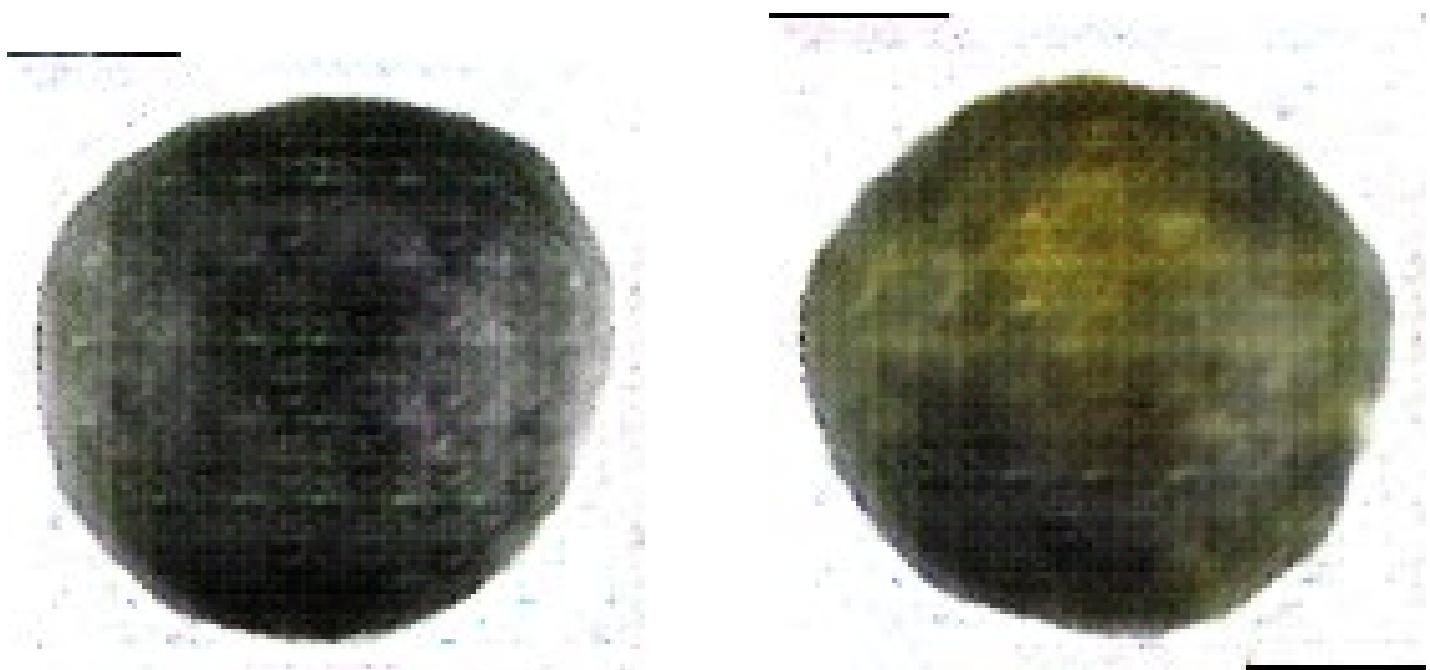


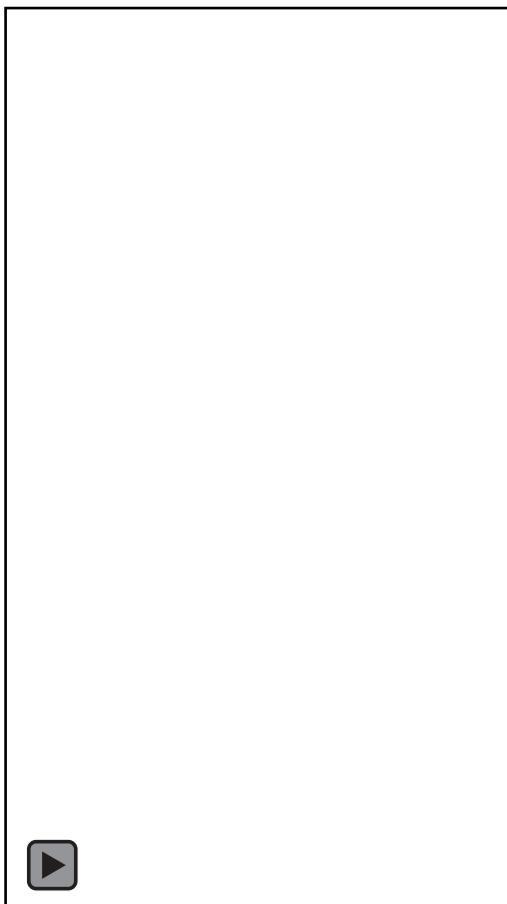
real_samples

Total time: 24.0h 16.0m 57s 745ms

Quelle: Täuber JG (2021) Generative Adversarial Networks: Eine empirische Untersuchung der Potenziale und Grenzen etablierter GAN-Architekturen bei der Generierung fotorealistischer Bilder. Bachelorthesis, Nürnberg.

Demonstration – Fake Mikrometeoriten





Reinforcement Learning

<https://fb.watch/9OI30GKS2J/>

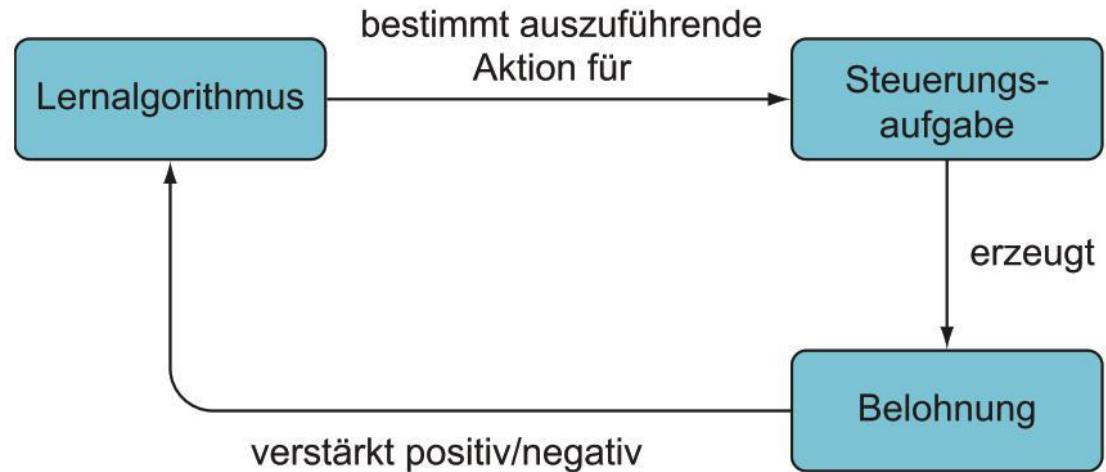
Ausgangspunkt

- Kann ein Agent ein gutes (optimales) Verhalten lernen, ohne dass er einen Lehrer hat bzw. benötigt?
- Reinforcement Learning kommt zum Einsatz, wenn entweder kein optimaler Weg für die Lösung einer Aufgabe bekannt ist oder keine oder wenige Trainingsdaten zur Verfügung stehen.
- Eingesetzt, wenn eine Aufgabe durchgeführt werden soll, für die eine herkömmliche Programmierung zu komplex wäre.
- Beispiel ist die Steuerung von Robotern

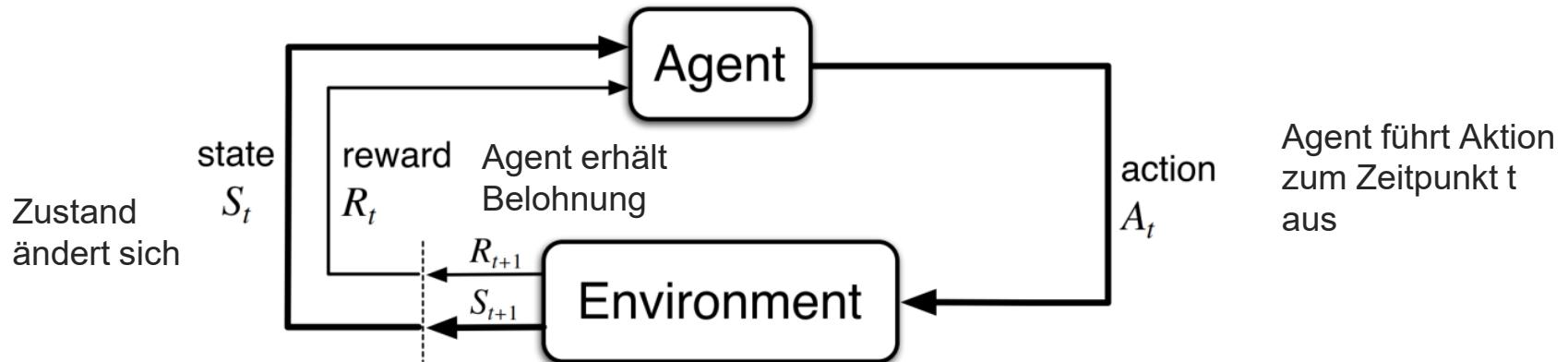
Grundprinzip

- Trial-and-Error – Versuch / Irrtum
- Belohnung / Bestrafung
- Anwendung des Agentenbasierter Systeme

- Es wird versucht, für einen bestimmten Zustand (state) anhand einer Strategie (policy) die bestmögliche Aktion (action) zu bestimmen.
- Jede Aktion zieht eine Belohnung (reward) mit sich
- Dies beeinflusst in der Regel die Ausprägung des Nachfolgezustands (next state).
- Eine Aktion hat damit Einfluss auf alle nachfolgenden Belohnungen.
- Die Belohnung kann auch negativ ausfallen und damit als ein Art Bestrafung fungieren.



- Die beiden „Größen“ im RL sind der Agent (agent) und die Umgebung (environment).
- Die Umgebung stellt die Welt dar, in welcher der Agent lebt und mit der er interagiert.
- Das zentrale Ziel des Agenten besteht darin, die Summe der Belohnungen (cumulative reward) zu maximieren.
- Abbildung zeigt den Zusammenhang zwischen dem Agenten und der Umgebung im Zeitschritt (time step) t und veranschaulicht die Interaktionen bzw. Elemente, welche die beiden Seiten verbinden:



Kern

- Markov-Entscheidungsprozesse

Model Based

- Ein Modell von der Umgebung wird erstellt, um Vorhersagen zu zukünftigen Zuständen oder Belohnungen treffen zu können.
- Planendes Vorgehen

Model Free

- Verzicht, die Umgebung verstehen zu wollen und Vorhersagen zu treffen
- Einfacher zu implementieren
- **Value-based-Ansatz**
Versuch die Wertefunktion kontinuierlich zu verbessern, um schließlich die optimale Wertefunktion zu lernen. Aus dieser lässt sich dann eine optimale Strategie ableiten.
- **Policy-based Ansatz**
Strategie direkt zu optimieren

Kartenspiele

- Schafkopf
Meyer M (2020) Evaluierung von Reinforcement Learning zur Implementierung einer KI für Kartenspiele. Bachelorthesis, Nürnberg.
- Watten
Amode J, Friedrich A, Haller A (2021) Reinforcement learning – KI für Watten. Seminararbeit, Nürnberg.

Andere Spiele

- Schach, Go (AlphaGo, AlphaGo Zero von Google)

Andere Anwendungen

- News recommendations
Zheng G, Zhang F, Zheng Z, Xiang Y, Yuan NJ, Xie X, Li Z (2018) DRN: A Deep Reinforcement Learning Framework for News Recommendation. In: Champin P-A, Gandon F, Lalmas M, Ipeirotis PG (Hrsg.) Proceedings of the 2018 World Wide Web Conference, Association for Computing Machinery-Digital Library; ACM Special Interest Group on Hypertext, Hypermedia, and Web, Republic and Canton of Geneva.



Problemlösen mittels Suchverfahren

Warum Suchverfahren?

- Viele Problemstellungen lassen sich auf Suche zurückführen.
 - Tourenplanung – kürzester, effizientester, billigster Weg
 - Spiele – aus einer gegebenen Anfangsstellung ein bestimmtes Ziel erreichen („Schachmatt“)
 - Mathematische Beweise führen
- Basis: Zustände (Anfang, Ende, Zwischenzustände), Regeln und Aktionen
 - Regeln geben vor, wie man einen (möglichen) Zustand A in einen nächsten (möglichen) Zustand B korrekt überführen kann. Wird eine Regel als Aktion durchgeführt, wird ein neuer (realer) Zustand erreicht.
 - Oft können mehrere Regeln als Aktion verwendet werden, dh. aus einem Zustand A sind mehrere Folgezustände $B_{1..n}$ möglich.
 - Man sucht nun diejenige Abfolge von Aktionen, die einen Zustand A in einen Zustand B (Zielzustand) überführen, sodass bestimmte Kriterien erfüllt sind, z.B. die kürzest mögliche Anzahl von Aktionen.

Probieren – Brute Force

- Alle möglichen Regeln als Aktionen durchführen:
 $A \rightarrow (B_1, B_2, \dots, B_n)$
 $B_1 \rightarrow (B_{11}, B_{12}, \dots, B_{1n})$
 $B_2 \rightarrow (B_{21}, B_{22}, \dots, B_{2n})$
- Problem: Der Lösungsraum explodiert selbst bei einfachen Problemen

Suchproblem in Graphen

- Suchproblem kann als gerichteter Graph aufgefasst werden
- Graph: Abstrakter Struktur aus Knoten (Ecken) und Kanten (Bögen)
- $G = (N, E)$ mit N Knoten und E Kanten und $E \subseteq N \times N$
- *Gegeben:* $S(N, E)$, $z_1 \in N$, *ziel:* $N \rightarrow (W, F)$
- *ziel* prüft, ob der N einem Zielzustand entspricht.
- Es kann natürlich mehrere Zielzustände geben.
- Gesucht: Folge von Knoten $k_0, \dots k_n$, sodass
 - 1: $z_1 = k_0$
 - 2: $(k_i, k_{i+1}) \in E$ für $i = 0 \dots n - 1$
 - 3: $\text{ziel}(k_n) = W$

Beispiel N-Puzzle, Schiebepuzzle

- <https://de.wikipedia.org/wiki/15-Puzzle>

In einer quadratischen Fläche mit NxN-1 durchnummerierten Kacheln ist eine Kachel leer. Ausgehend von einer Ausgangskonfiguration soll eine Zielkonfiguration durch Verschieben der Leerstelle bzw. Bewegen einer Kachel in die Leerstelle erreicht werden.

Initial State			Goal State		
1	2	3	2	8	1
8		4		4	3
7	6	5	7	6	5

<https://blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288>

- Keine Zusatzinformation über zu lösendes Problem vorhanden

Kand die Liste der Kandidaten von Wegen, die zu untersuchen sind

z_1 der Ausgangszustand

z und s Zustände (Knoten des Graphen)

p Wege durch den Zustandsraum

Ein Weg ist eine Folge von durch Kanten verbundenen Knoten.

FUNCTION SUCHE

Kand := [z_1];

Ziel_erreicht := false;

REPEAT

Wähle aus Kand einen Weg : $p := z_1 z_2 \dots z_m$ aus ;

Ziel_erreicht := Ziel(z_m) ;

IF NOT Ziel_erreicht THEN

Berechne alle Zustände, die von z_m in einem Schritt erreicht werden können : $[s_1, \dots, s_k]$;

Lösche p aus Kand heraus ;

Füge die Wege $z_1 z_2 \dots z_m s_1, \dots, z_1 z_2 \dots z_m s_k$ in Kand ein;

ENDIF;

UNTIL Ziel_erreicht;

RETURN p;

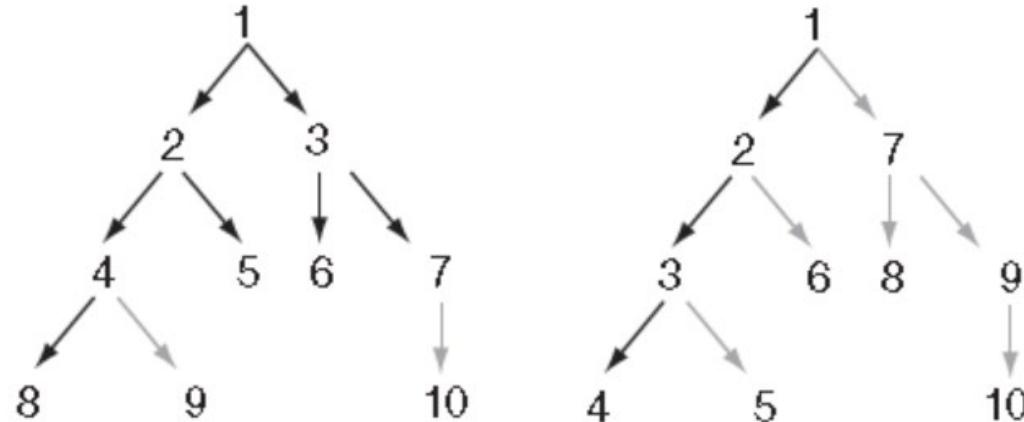
END

Freiheitsgrade bei der Suchsteuerung

- Einfügen neuer Wege
- Auswahl eines Kandidaten

Prinzipiell zwei Möglichkeiten

- Breitensuche: Expansion des Suchraums ebenenweise
- Tiefensuche: Expansion des Suchraums



Breiten und Tiefensuche

FUNCTION BREITENSUCHE

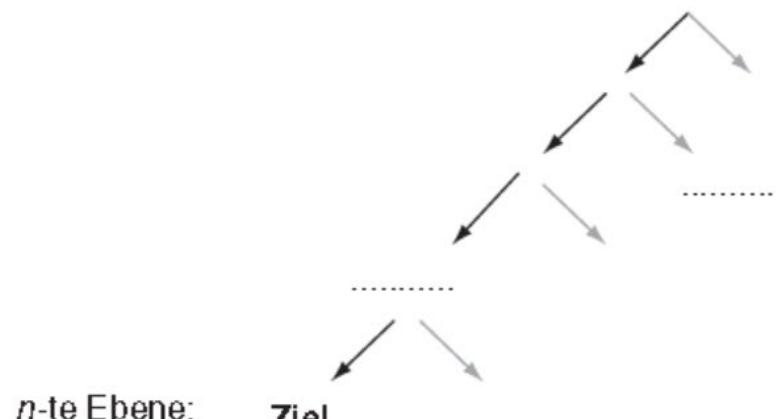
```

Kand := [z1];
Ziel_erreicht := false;
REPEAT
    Sei Kand = [z1z2 ... zm, p2, p3, ..., pn] ;
    p := z1z2 ... zm ;
    Ziel_erreicht := ziel(zm) ;
    IF NOT Ziel_erreicht THEN
        Berechne alle Zustände, die von zm in einem Schritt erreicht werden können : [s1, ..., sk] ;
        Kand = [p2, p3, ..., pn, z1z2 ... zm s1, ..., z1z2 ... zm sk] ;           ENDIF;
    UNTIL Ziel_erreicht;
    RETURN p;
END
```

FUNCTION TIEFENSUCHE

```

Kand := [z1];
Ziel_erreicht := false;
REPEAT
    Sei Kand = [z1z2 ... zm, p2, p3, ..., pn] ;
    p := z1z2 ... zm ;
    Ziel_erreicht := ziel(zm) ;
    IF NOT Ziel_erreicht THEN
        Berechne alle Zustände, die von zm in einem Schritt erreicht werden können : [s1, ..., sk] ;
        Kand = [z1z2 ... zm s1, ..., z1z2 ... zm sk, p2, p3, ..., pn] ;           ENDIF;
    UNTIL Ziel_erreicht;
    RETURN p;
END
```



Tiefensuche – günstiger Suchraum

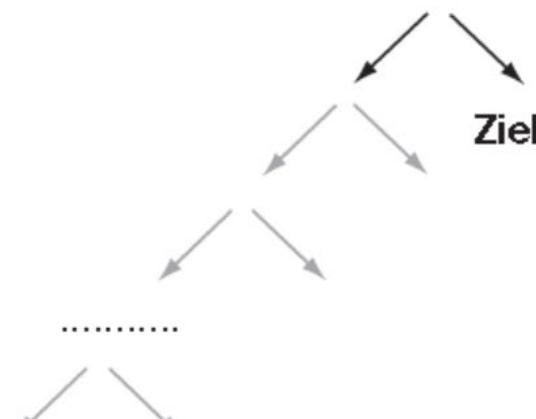


BILD 3.9 Breitensuche – günstiger Suchraum

- **Vollständigkeit:** Suchverfahren ist vollständig, wenn es immer eine Lösung findet (falls diese existiert)
- **Korrektheit:** Suchverfahren ist korrekt, wenn der vorgeschlagene Weg tatsächlich eine Lösung ist
- Breitensuche ist vollständig, Tiefensuche nur dann, wenn Suchraum endlich ist.
- Tiefensuche kann unendlich tiefen Ast enthalten (regelbedingt!)
- Suchverfahren sind korrekt, da Zielerreichung über ein Prädikat Wahr/Falsch geprüft werden kann.

Vorteile und Nachteile

- Beide Verfahren einfach zu implementieren
- Tiefensuche findet Lösung rascher als Breitensuche
- Erkennung und Vermeidung doppelter Pfade, Schleifen

Tiefen- und Breitensuche

- Erstellen Sie für folgenden Ausgangszustand des Schiebepuzzles eine Graphen, der die nächsten erlaubten drei Züge darstellt.
- Welche Züge sind erlaubt?
- Wie hängen diese vom aktuellen Zustand ab?
- Wie würde dieser Graph mit Breiten- bzw. Tiefensuche durchlaufen werden?

Initial State

1	2	3
8		4
7	6	5

Heuristische oder informierte Suche

- Bei größeren Problemen benötigen Breiten- und Tiefensuche viel Speicherplatz (Wege speichern) und Zeit.
- Als Konsequenz wird bei gegebenen Zeitlimit eine Lösung nicht gefunden.
- Heuristiken sollen helfen, nur erfolgversprechende Pfade zu durchlaufen
 - Etwa bei Schiebepuzzle keine Kreise
- Heuristiken strukturieren und / oder verkleinern den Suchraum
 - Strukturieren -> Suchverfahren vollständig
 - Verkleinern -> Vollständigkeit kann verloren gehen
- Oft existieren zusätzliche Anforderungen wie den kürzesten Weg zu finden.
- Verwendung von Zusatzinformationen wie Entfernungen, Preise, Materialverbrauch
 - Schach: Figurenbewertung, Qualität der Stellung, Entfernung zum Schachmatt
- Diese Information kann dazu verwendet werden, den möglichen Lösungsraum zu beschneiden.

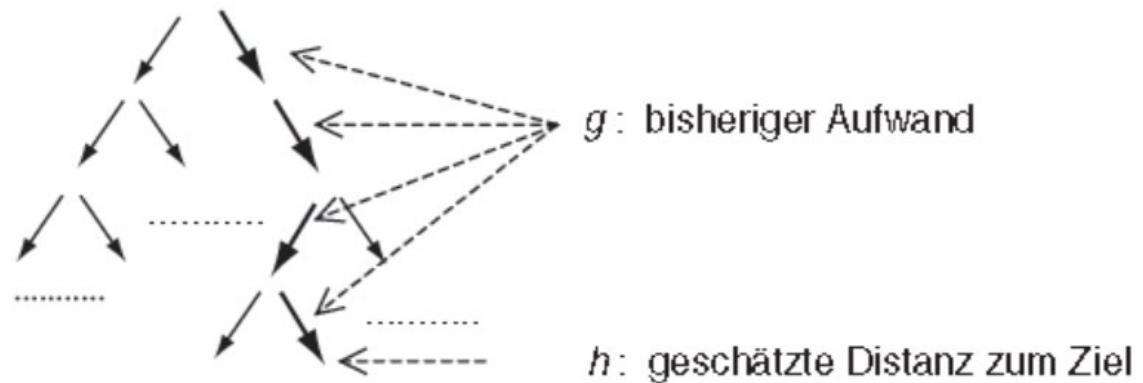
- Welche Heuristiken könnten beim Schiebepuzzle verwendet werden?

Heuristische oder informierte Suche

- Einsatz von Schätzfunktionen
- Man schätzt, wie weit man noch von der Lösung entfernt ist.

Heuristiken

- Heuristische Funktion h
 - Je besser die h -Bewertung eines Kandidaten ist, desto näher an der Lösung. Lösung hat h -Wert von 0 (Null). Keine Garantie, dass Lösung gut ist, da nur Schätzung.
- Heuristische Funktion g
 - Bisherigen Aufwand (mit)berücksichtigen, z.B. Weglänge, Anzahl der Züge, Schach: Qualität der Stellung



Nächster Nachbarheuristik/strategie

- Existenz einer Funktion g zur Aufwandsabschätzung
- Wähle aus den Kandidaten denjenigen mit dem geringsten Abstand vom aktuellen Zustand aus.
- Minimierung der Zustandsübergänge
 - Z.B. Wähle den Knoten mit dem minimalen Abstand vom aktuellen Knoten, z.B. der Entfernung zwischen Städten
- Unvollständig, da kein Zurückgehen vorgesehen ist

Bergsteigerstrategie

- Existenz einer Funktion h zur Zielentfernungsabschätzung
- Nur die Folgezustände gewählt, die näher am Ziel liegen
- Alle Zweige abschneiden, die schlechter als die Vorgänger sind
- Durchsuchen mehrere Kandidaten möglich
- Unvollständig, da Suchraum beschnitten wird

Bestensuche

- Variante der Breitensuche
- Existenz einer Funktion h zur Zielentfernungsabschätzung
- Wähle aus allen Kandidaten denjenigen mit der besten h Bewertung
- Strukturiert den Suchraum, daher vollständig

A* Suche

- Einsatz einer Funktion f bestehend aus der heuristischen Funktionen g und h
- Summe: $f = g + h$
- Vorgehen:
 - Zwei Knotenmengen
 - offen, bewertet, aber noch nicht expandiert OPEN
 - geschlossen (bewertet und expandiert) CLOSE
- **Zulässigkeit** einer Schätzfunktion h
 - Eine Funktion h zur Bewertung eines Knotens heißt zulässig, wenn sie die Kosten, um zum Ziel zu kommen, nicht überschätzt:
 - Für jeden Knoten n und jeden Zielknoten z , der von n aus erreichbar ist, gilt:
$$h(n) \leq g(z) - g(n).$$
 - Ist die Funktion h zulässig, so ist der A*-Algorithmus optimal.

FUNCTION A-STERN

```
OPEN := [z1]; CLOSED := [ ]; Ziel_erreicht := false;  
REPEAT  
    x := bester Knoten (bez. f) aus OPEN ;  
    Ziel_erreicht := ziel(x) ;  
    IF NOT Ziel_erreicht THEN  
        Lösche x in OPEN; Füge x in CLOSED ein;  
        SUCCESSORS := alle Nachfolger von x ;  
        FORALL y ∈ SUCCESSORS  
            IF y ∈ OPEN THEN  
                Vergleiche neuen Weg zu y mit bisherigem und lösche den schlechteren;  
            ELSEIF y ∈ CLOSED THEN  
                Vergleiche neuen Weg zu y mit bisherigem und lösche den schlechteren;  
            ELSE Füge y in OPEN ein ;  
            ENDIF;  
        ENDFORALL;  
    ENDIF;  
    UNTIL Ziel_erreicht;  
    RETURN x;  
END
```

Lämmel U, Cleve J (2012) Künstliche Intelligenz. Hanser, München. S.139

A* Suche – Beispiel Schiebepuzzle

```
initial = [ ['2', '8', '3'],
            ['1', '6', '4'],
            ['7', '_', '5']]
```

```
final = [ ['1', '2', '3'],
          ['8', '_', '4'],
          ['7', '6', '5']]
```

```
puz = Puzzle(3)
puz.process(initial, final)
```

Aktueller Zustand

2	3	
1	8	4
7	6	5

```
[[['2', '_'], ['3'], ['1', '8', '4']], [['7', '6', '5']]]:7
[[['1', '2'], ['3'], ['_', '8', '4']], [['7', '6', '5']]]:5
```

f Bewertung



Mögliche Folgezustand

A* Suche – Beispiel Schiebepuzzle

```

class Puzzle:
    def __init__(self,size):
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self,start,goal):
        # Heuristic Function to calculate hueristic value f(x) = h(x)
        + g(x)
        return self.h(start.data,goal)+start.level

    def h(self,start,goal):
        # Calculates the difference between the given puzzles
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp

```

```

def process(self, initial, final):
    # Accept Start and Goal Puzzle state
    start = initial
    goal = final
    start = Node(start,0,0)
    start.fval = self.f(start,goal)
    """ Put the start node in the open list"""
    self.open.append(start)
    print("\n\n")
    iNum = 1
    while True:
        cur = self.open[0]

        # If the difference between current and goal node is 0 we have
        reached the goal node
        if self.h(cur.data,goal) == 0:
            break
        for i in cur.generate_child():
            i.fval = self.f(i,goal)
            print(f"{i.data}:{i.fval}")
            self.open.append(i)
            self.closed.append(cur)
            del self.open[0]
            iNum = iNum + 1

        # sort the open list based on f value
        self.open.sort(key = lambda x:x.fval,reverse=False)
        print ("Number of draws: ", iNum)

```

<https://blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288>

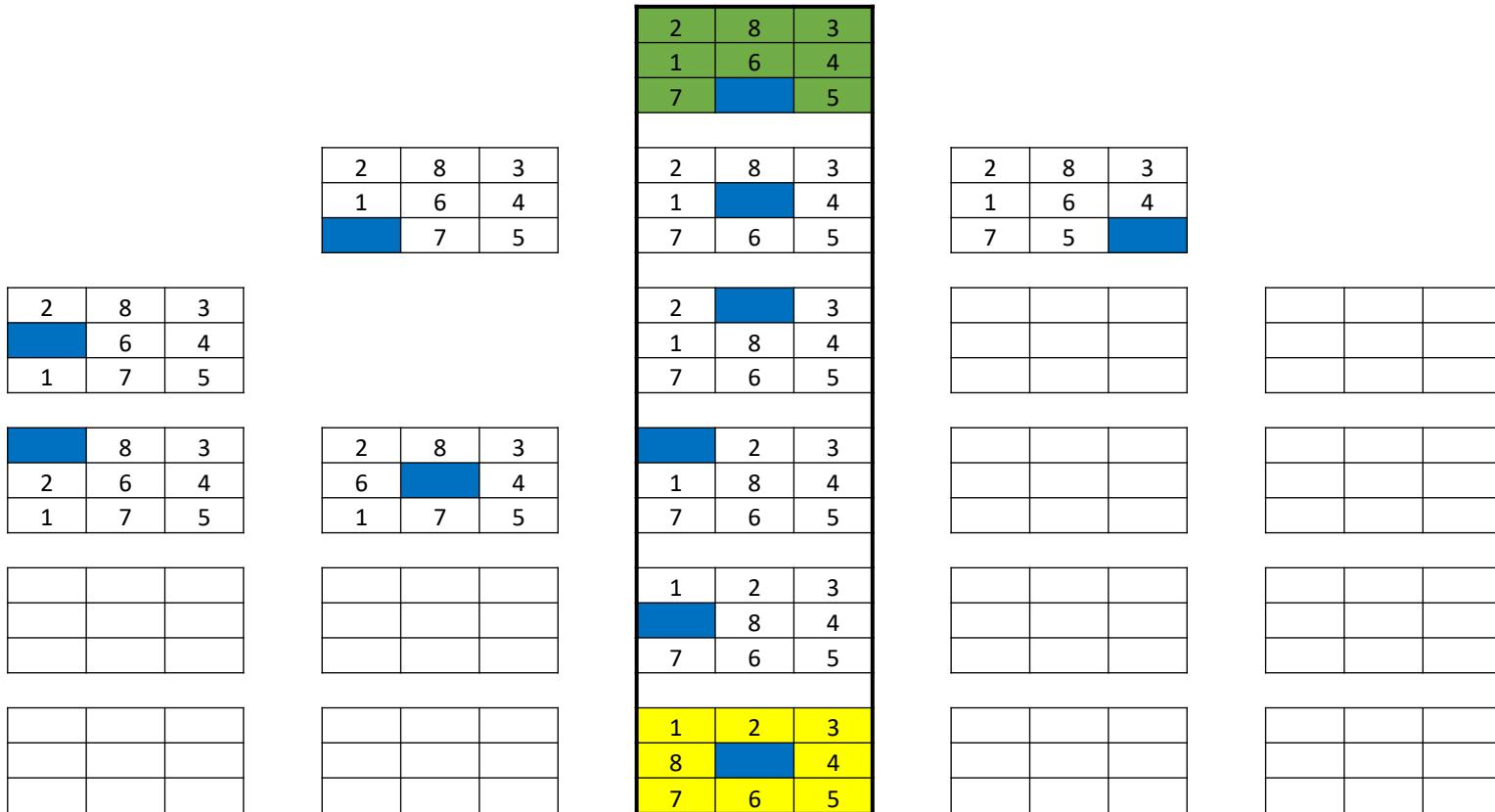


TABELLE 3.7 Suchverfahren und ihre Charakteristika

Verfahren	heuristische Funktion	Wesentliche Merkmale
Breitensuche	–	Alte Kandidaten werden den neuen vorgezogen.
Tiefensuche	–	Es wird mit den neuen Kandidaten weitergearbeitet.
Bestensuche	h	Aus den möglichen Kandidaten wird der bezüglich h beste Kandidat gewählt.
Bergsteiger	h	Es werden nur solche Folgezustände akzeptiert, deren h -Bewertung besser ist.
Nächster Nachbar	g	Es wird der Knoten als nächster besucht, der am nächsten zum aktuellen Knoten liegt.
A*-Algorithmus	g und h	Bestensuche mit heuristischer Funktion $g + h$. Mehrfachwege werden vermieden.

Lämmel U, Cleve J (2012) Künstliche Intelligenz. Hanser, München. S.145

- Wie lautet die Lösung für?

```
initial = [[ '1', '2', '3' ],  
           [ '8', ' ', '4' ],  
           [ '7', '6', '5' ]]
```

```
final = [ [ '2', '8', '3' ],  
          [ '1', '6', '4' ],  
          [ '7', ' ', '5' ] ]
```



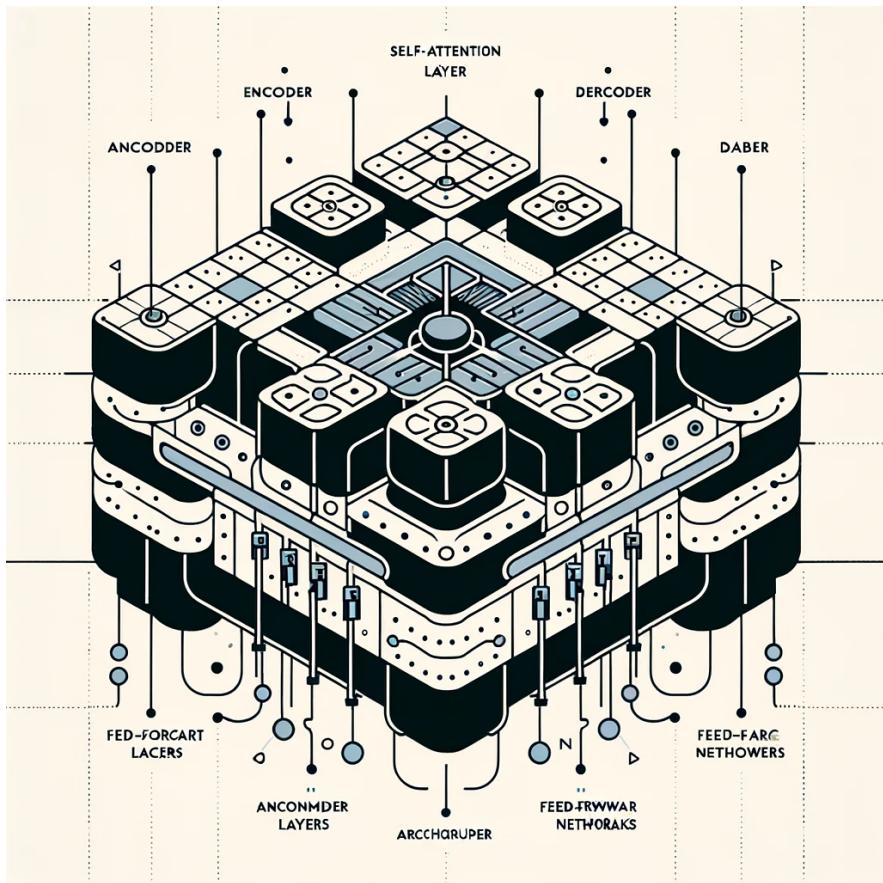
Transformer-Modelle Einführung und Funktionsweise

Was ist ein Transformer-Modell?

- Sie haben die NLP-Landschaft revolutioniert und RNNs und CNNs in vielen Aufgaben ersetzt.
- Hauptanwendung: Maschinelle Übersetzung, Textgenerierung, Sprachverständnis.

Architektur des Transformer-Modells

- Besteht aus zwei Hauptkomponenten: Encoder und Decoder.
- Der Encoder kodiert den Eingabetext in einen abstrakten, hochdimensionalen Raum.
- Der Decoder dekodiert diese Repräsentation und generiert den Ausgabetext.



Architektur der Transformer-Schichten

- Jede Encoder- und Decoder-Schicht besteht aus Sub-Layern:
 1. Multi-Head Self-Attention Mechanismus
 2. Punktweise vollständig verbundenes Feed-Forward Netzwerk
- Residual Connections (Sprungverbindungen) und Layer-Normalisierung werden nach jedem Sub-Layer angewendet.
- Multi-Head Attention erlaubt dem Modell, Informationen aus verschiedenen Repräsentationsräumen zu kombinieren.
- Das Feed-Forward Netzwerk ist für jede Position unabhängig und besteht aus zwei linearen Transformationen mit einer ReLU-Aktivierung dazwischen.

- Der Encoder verarbeitet die Eingabedaten parallel, was die Effizienz deutlich erhöht. Er verwendet Self-Attention-Mechanismen, um die Bedeutung verschiedener Wörter in der Eingabesequenz zu gewichten.
- Der Decoder erzeugt die Ausgabesequenz. Er nutzt sowohl Self-Attention als auch Encoder-Decoder-Attention-Mechanismen, um sich auf relevante Teile der Eingabe und der bereits generierten Ausgabe zu konzentrieren.

Encoder

- Besteht aus mehreren Schichten.
 - Jede Schicht hat zwei Hauptbestandteile:
 - Self-Attention-Mechanismus
 - Feed-Forward-Neuronales Netzwerk
- Hinzufügen von Residual Connections und Layer Normalization.

Decoder

- Funktioniert ähnlich wie der Encoder, aber mit zusätzlichen Komponenten.
- Zwei Arten von Attention: Self-Attention und Encoder-Decoder-Attention.
- Output von Encoder wird genutzt, um den Kontext während der Generierung zu berücksichtigen.

- Der Attention-Mechanismus ist ein zentrales Merkmal von Transformern. Er ermöglicht es dem Modell, sich beim Generieren jedes Wortes in der Ausgabesequenz auf unterschiedliche Teile der Eingabesequenz zu konzentrieren. Dadurch entstehen genauere und kohärentere Übersetzungen bzw. Textgenerierungen.
- Transformer besitzen von sich aus kein Verständnis für die Reihenfolge von Wörtern in einer Sequenz. Daher werden Positionsencodierungen zu den Eingabe-Embeddings hinzugefügt, um dem Modell Informationen über die Position jedes Wortes in der Sequenz zu geben.

Self Attention → Selbstaufmerksamkeit**

Selbstaufmerksamkeit ermöglicht es einem Modell, jedes Wort im Kontext des gesamten Satzes zu betrachten, anstatt isoliert, indem die Bedeutung der anderen Wörter in Beziehung zu jedem einzelnen Wort gewichtet wird.

- Erlaubt dem Modell, auf verschiedene Teile des Eingabetextes gleichzeitig zu achten.
- Berechnung der Aufmerksamkeit anhand von Query, Key und Value Matrizen.
- Multi-Head Attention: Mehrere Attention-Mechanismen werden parallel ausgeführt

Kernbegriffe

- Query (Q) – Was suche ich? Beschreibt die aktuelle Position, die gerade Aufmerksamkeit verteilen möchte.
- Key (K) – Wie relevant bin ich? Beschreibt jedes Wort im Satz im Hinblick darauf, ob es relevant für die aktuelle Query ist.
- Value (V) – Welche Information gebe ich weiter? Enthält die eigentliche Inhaltsinformation, die an die Query zurückgegeben wird.

1. Input:

Menge von Vektoren, die die Wörter eines Satzes repräsentieren.

2. Query, Key, Value:

Jedes Wort wird in drei Vektoren transformiert –
Query (Q), Key (K) und Value (V).

3. Attention Scores

Berechnung der Aufmerksamkeitswerte über das Skalarprodukt von Q und K,
gefolgt von einer Normalisierung.

4. Gewichtete Summe

Für jedes Wort wird ein neuer Vektor erzeugt, basierend auf der gewichteten
Summe der V-Vektoren.

5. Output

Neuer Vektor, der kontextabhängige aggregierte Information repräsentiert.

Beispielsatz: „NLP is amazing“

1. Eingabewortvektoren: NLP -> [1, 0, 1], is -> [0, 1, 0], amazing -> [1, 1, 0]
2. Lineare Transformationen zu Query, Key und Value Vektoren
3. Berechnung der Aufmerksamkeitswerte (Scores)
4. Anwendung der Softmax-Funktion
5. Gewichtete Summe der Value-Vektoren zur Berechnung der neuen Wortdarstellungen

- Transformer-Modelle haben keine inhärente Reihenfolge.
- Positionale Codierung wird hinzugefügt, um die Position der Wörter im Satz zu berücksichtigen.
- Verwendet Sinus- und Cosinus-Funktionen basierend auf der Position des Wortes.
- Formeln:

$$\text{PE(pos, } 2i\text{)} = \sin(\text{pos} / 10000^{(2i/d_model)})$$

$$\text{PE(pos, } 2i+1\text{)} = \cos(\text{pos} / 10000^{(2i/d_model)})$$

Beispielsatz: „NLP is amazing“

1. Wortvektoren: NLP -> [1, 0, 1, 0], is -> [0, 1, 0, 1], amazing -> [1, 1, 1, 1]

2. Positionale Codierungen berechnen:

Position 0: [0, 1, 0, 1]

Position 1: [$\sin(1)$, $\cos(1)$, $\sin(1)$, $\cos(1)$]

Position 2: [$\sin(2)$, $\cos(2)$, $\sin(2)$, $\cos(2)$]

3. Positionale Codierungen zu den Wortvektoren addieren.

Training und Optimierung

- Verwendung von großen Datensätzen für das Training.
- Adam Optimizer wird oft genutzt.
- Verlustfunktion: Normalerweise Kreuzentropieverlust für die Sequenzgenerierung.

- Maschinelle Übersetzung (z.B., Google Translate)
- Textgenerierung (z.B., GPT-Modelle)
- Sprachverständnis (z.B., BERT-Modelle)
- Frage-Antwort-Systeme, Textzusammenfassung

- Transformer überwinden die Einschränkungen von RNNs und LSTMs, wie die schwierige Parallelisierung der Verarbeitung sowie Probleme mit weitreichenden Abhängigkeiten in Sequenzen.
- Transformer werden in vielen NLP-Aufgaben eingesetzt, z. B. maschinelle Übersetzung, Textzusammenfassung und Frage-Antwort-Systeme.
- Die Transformer-Architektur stellt einen bedeutenden Fortschritt im Bereich NLP dar, da sie eine höhere Effizienz und Effektivität bei der Verarbeitung sequenzieller Daten ermöglicht.

Aufgabe 1:

- Gemeinsame Anwendung recherchieren (z. B. Übersetzung, Chatbot, Agentic AI).
- Gruppen bearbeiten unterschiedliche Aspekte des Transformers.
- Ziel: Funktionsweise anhand eines konkreten Beispiels erklären.
- 10 Minuten

Aufgabe 2: Einzelaspekte Transformer

- Sh. Folgefolien
- 20 Minuten

Präsentation

- Alle Teilaufgaben zu einer gemeinsamen Präsentation
- Jede Gruppe präsentiert 3–5 Minuten.
- Definitionen, Grafiken, Mini-Beispiel.
- Kernerkenntnisse zur gewählten Anwendung.

Aufgabe 2: Gruppe 1 – Query, Key, Value & Gewichtsmatrizen

- Was sind Q, K, V?
- Wie entstehen sie durch lineare Transformationen?
- Wie werden W^Q , W^K , W^V gelernt?
- Mini-Beispiel mit Bezug zur Anwendung.

Aufgabe 2: Gruppe 2 – Self-Attention

- Berechnung der Attention Scores.
- Softmax und gewichtete Summen.
- Warum Self-Attention kontextsensitiv ist.
- Rechenbeispiel zur Anwendung.

Aufgabe 2: Gruppe 3 – Encoder

- Rolle des Encoders im Transformer.
- Mehrere Attention- und FFN-Schichten.
- Residuals & Layer Normalization.
- Welche Informationen der Encoder weitergibt.

Aufgabe 2: Gruppe 4 – Decoder

- Masked Self-Attention.
- Encoder–Decoder-Attention.
- Wie der Decoder das nächste Token generiert.
- Interaktion zwischen Encoder und Decoder.

Aufgabe 3: Weitere Transformer-Modelle

Neben LLMs, den klassischen Einsatzgebiet von Transformern, gibt es noch andere Varianten.

Erstellen Sie eine Liste weiterer Transformer-Modelle.



Agentenbasierte KI Systeme

Kennzeichen

- Individuen basierte Modellbildung und Simulation
- Viele kleine mehr oder minder „intelligente“ Einheiten interagieren miteinander
- Systemverhalten entsteht durch das Verhalten der einzelnen Agenten
- Komplexes Verhalten entsteht aus einfachen Mustern/Verhaltensregeln
- Verteilte KI

1 Frochte, Jörg (2019): Maschinelles Lernen. Grundlagen und Algorithmen in Python. 2., aktualisierte Auflage. München: Hanser.

Beispiele

- Staus auf Autobahnen
- Ameisenstraßen
- Soziale Netzwerke
- Chatbots, Softbots
- Suchmaschinen
- Drohnenschwärme
- Roboterkoordination
- BargainFinder, NewsFinder
Agent sucht basierend auf einmal festgelegten Benutzerangaben (Suchkriterien) Angebote oder News.
- Email-Agent

Argumente für Agentensysteme

- Mehrwert: Kombination vieler Einzelagenten mehr als die reine Summe der Möglichkeiten von Einzelagenten
- Vermeiden des Ausfallrisikos eines zentralen Systems
- Vermeiden der Beschränkungen der Ressourcen eines zentralen Systems
Integration: Bestehende Systeme (z.B. Expertensysteme, konventionelle Programme) integrierbar
- Problemlösen: einfacher Lösungen finden durch Kombinationen mehrerer Agenten mit alternativen Ansätzen (klassisch, KI)
- Parallelisierung: Viele Agenten arbeiten an einem Problem
- Internet/WWW: Agenten laufen im Netz



Verteilte KI

Software-Agent

- „Ein Software-Agent ist eine Software, die – im Rahmen ihrer Fähigkeiten – zu eigenständigem bzw. autonomem Verhalten in der Lage ist. Das meint u. a. dass kein Mensch mehr involviert sein muss, wenn das Programm einmal gestartet ist.“¹
- „Ein Agent ist ein Computersystem, das sich in einer bestimmten Umgebung befindet und welches fähig ist, eigenständige Aktionen in dieser Umgebung durchzuführen, um seine (vorgegebenen) Ziele zu erreichen.“²

- Agent ist zu eigenständigem, autonomen Verhalten fähig
- Hat eigenen Zustand
- Kommunikationsfähigkeit
- Adoptions- und Modifikationsfähigkeit
- Hewitt: Ein "Actor"
"is a computational agent which has a mail address and a behavior. Actors communicate by message-passing and carry out their actions concurrently."³

1 Frochte, Jörg (2019): Maschinelles Lernen. Grundlagen und Algorithmen in Python. 2 aktualisierte Auflage. München: Hanser.

2 Wooldridge: Intelligent Agents: The Key Concepts. 2002

3 Hewitt, C.: "Viewing Control Structures as Patterns of Passing Messages". Artificial Intelligence (8) 3. S.323ff. North-Holland Publishing Company, Amsterdam, NL (1977).

Zentrales Problem

- Koordination der einzelnen Agenten zur Erreichung eines globalen Ziels, ev. optimalen Ziels

Koordination

- Aufteilung und Verteilung von Aufgaben auf verschiedene Agenten
- Koordination und Kooperation der Agenten untereinander
- Konfliktlösungs- oder Verhandlungsstrategien

Zielorientierung

- Ein Agent besitzt bestimmte Fähigkeiten, die er einsetzt, um seine Ziele zu erreichen.

Autonomie

- Eigenständige Entscheidungsmöglichkeiten

Soziale Fähigkeiten

- Kommunikation mit Umwelt

Reaktivität

- Beobachten und Reagieren auf Veränderungen in der Umwelt

Proaktivität

- Versucht Zukunft vorherzusehen und reagiert entsprechend

Lern- und Anpassungsfähigkeit

- Anpassung an Umweltveränderungen und Eingliedern in eigenes Wissenssystem

Rationalität

- Aktionen basieren auf Umwelt, Wissensstand und Zielen und Annahmen

Benutzermodell

- Wünsche, Ziele, Vorlieben des Benutzers werden in einem Benutzermodell repräsentiert („Digitaler Zwilling“)

Auftraggeber

- Ein Agent benötigt einen Auftraggeber (Benutzer), der Ziele, Aufgaben vorgibt

Schnittstellen

- Kommunikation mit der Umwelt

Rezeptoren

- Sensoren, ...

Aktoren

- Zur Umweltbeeinflussung

Welche Eigenschaften könnte ein Agent noch aufweisen?

Datenschutz

- Wie wird verhindert, dass ein Agent unerlaubt Informationen einsetzt oder weitergibt?

Haftung

- Wer haftet bei Fehlern von Agenten?
- Wer kontrolliert den Agenten?

Abrechnung

- Wie wird die Dienstleistung bezahlt?
- Agent beauftragt andere Agenten?

Ausfall

- Wer ist für den Schaden verantwortlich, der beim Ausfall von Agenten entstehen kann?
- Programmierfehler?

Betrügerische Agenten

- Agent „lügt“?
- Wie kann erkannt werden, dass ein Agent „ehrlich“ handelt?
- Wer verbirgt sich hinter dem Agenten?
- Wie verhindert man Veränderungen von Agenten / Zielen durch nicht berechtigte Personen / Agenten?
- Zertifizierung von Agenten

Lösungsmöglichkeiten

- „Organisational Structuring“
fest vorgegebene Struktur der Agenten
 - Master/slave agents
 - Blackboard-Architektur
- Contract nets - Vertragsnetze
Manager Agents teilen Aufgabe auf und Contractor Agents erfüllen Aufgaben
- Game theory-based Negotiation („spieltheoretischen Verhandlungsansatz“)
Einsatz von Nutzenfunktionen
- Marktplatz von Agenten
Agenten handeln mit Produkten – Preis basiert
- Multi-agent Planning - Multiagenten-Planung

Multi-agent Planning - Multiagenten-Planung

- Jeder Agent plant für sich eine Vorgehensweise zur Durchführung der zentralen Aufgabe

Koordination / Auflösung / Zuteilung

- Entweder **zentral** ("centralised multi-agent planning"): zentrale Instanz löst Inkonsistenzen und Konflikte
- oder
- Agenten sprechen sich **untereinander** ("distributed multi-agent planning")
Größere Menge an Information und Kommunikation der Agenten untereinander

Fünf Klassen

- Einfache reaktive Agenten (simple reflex agents)
- Beobachtende Agenten (model-based reflex agent)
- Zielbasierte Agenten (goal-based agents)
- Nutzenbasierte Agenten (utility-based agents)
- Lernende Agenten (learning agents)

Weitere Klassifizierung

- Robust
der Agent kompensiert (teilweise) äußere und innere Störungen
- Kognitiv
der Agent ist auf der Basis eigener Entscheidungen und Beobachtungen lernfähig
- Sozial
der Agent kommuniziert mit anderen Agenten

Eigenschaften

- **Sensoren** (Hardware, virtuellen Umgebung, ...)
- **Aktoren** zum Veranlassen von Aktionen (virtuell im Computer oder Teil eines realen Roboters)
- Einfacher gedächtnislosen **Regelsatz** (Condition-action rules)
- Rein reaktives Verhalten - eine Aktion wird durch einen äußeren Reiz hervorgerufen

Schema

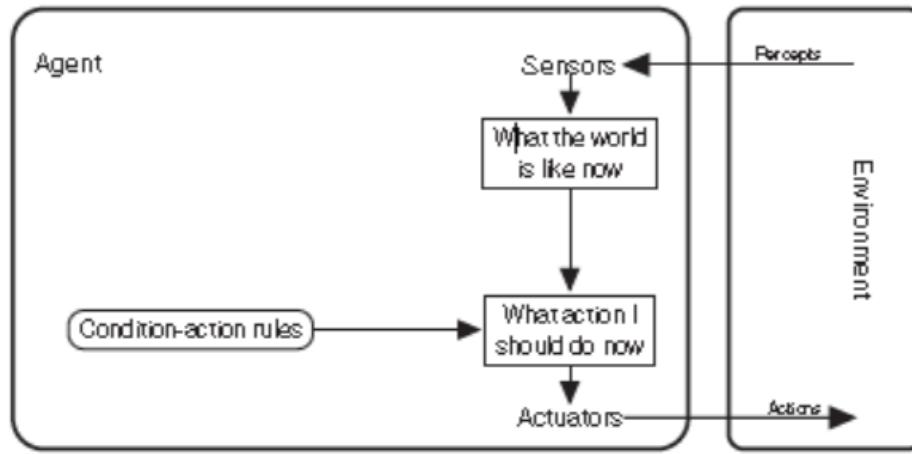


Abbildung 12.1 Schema eines reaktiven Agenten

Eigenschaften

- Gedächtnis, welches durch die Aspekte How the world evolves und Zustand sichtbar wird.
- Zustand bzw. State (Modell von seinem eigenen, vom Zustand der Welt)
- Aktionen auf der Basis dieser Modelle
- Regeln -> Sensorinformationen + gewonnenes Gesamtbild
- Basis für die Regeln flexibler, jedoch nicht die Regel selbst

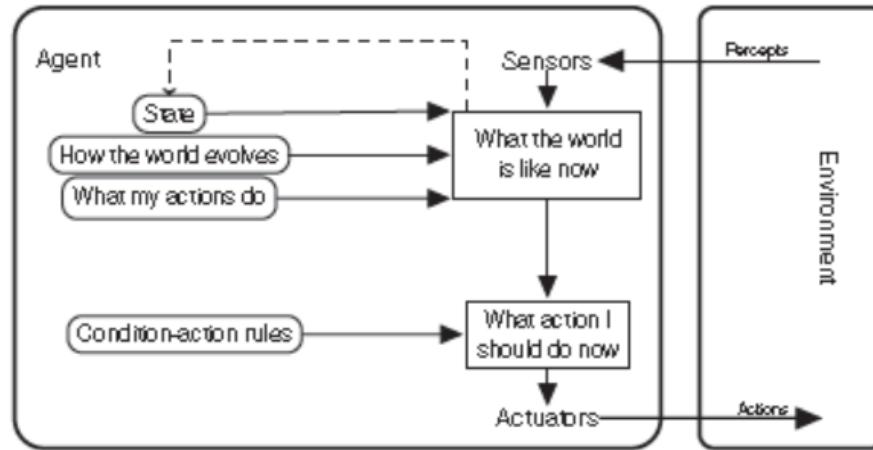


Abbildung 12.2 Schema eines beobachtenden Agenten

Lernender Agent

- Verhalten erfahrungsgebasisiert ändern (gemessene Reaktionen der Umwelt auf eigene Aktionen oder per Sensoren detektierten Bedingungen).
- Performance Element = beobachtender Agent
- Learning Element modifiziert das Performance Element
- Basierend auf dem Critic Modul
- Problem Generator – Lernen durch Durchführung nicht optimaler (besten) Aktionen

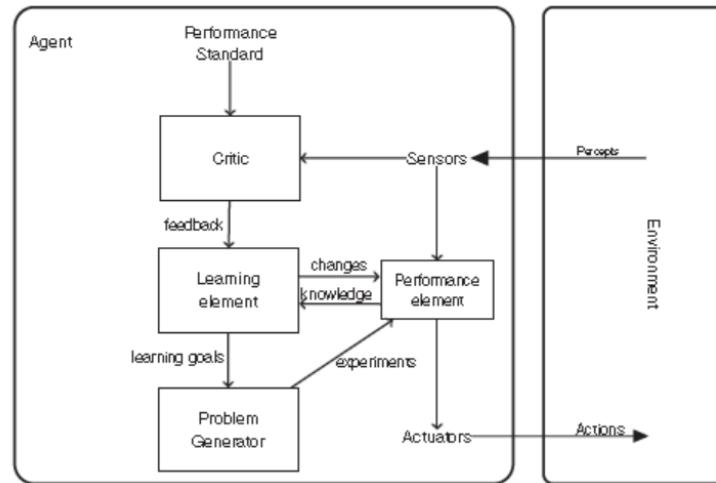


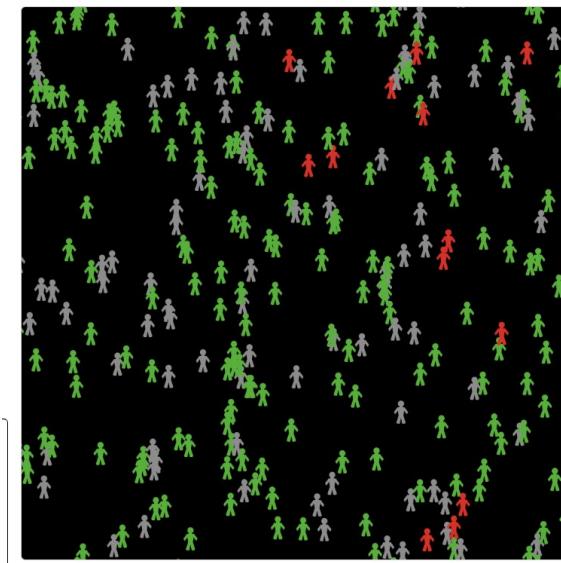
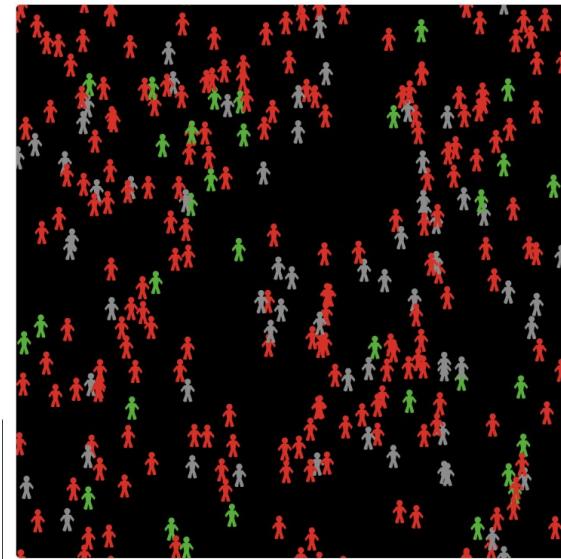
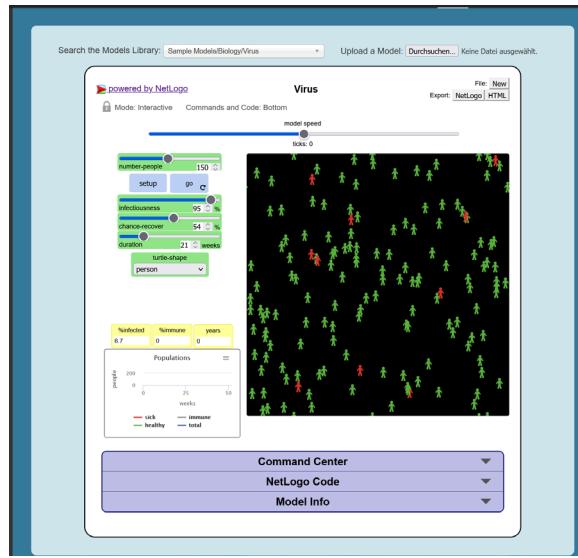
Abbildung 12.3 Schema eines lernenden Agenten

1 Frochte, Jörg (2019): Maschinelles Lernen. Grundlagen und Algorithmen in Python. 2., aktualisierte Auflage. München: Hanser.

Simulation

Test/Spielumgebung für agentenbasierte Systeme

- NetLogo
- <http://www.netlogoweb.org/launch#http://www.netlogoweb.org/assets/modelslib/Sample%20Models/Biology/Virus.nlogo>

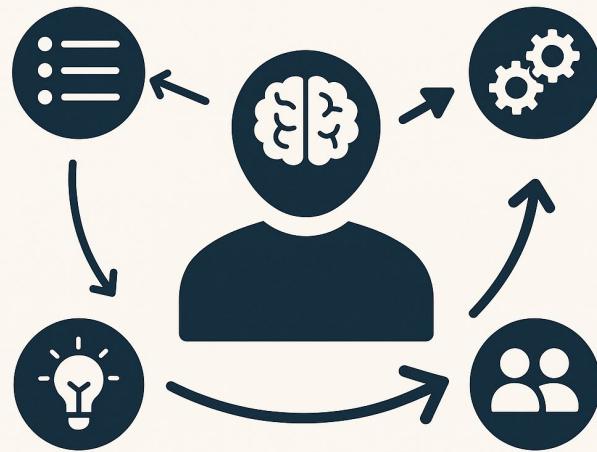


Welche Aufgaben könnte ein Software-Agent in Ihrem Unternehmen übernehmen?

Wofür würde er Intelligenz benötigen?

Suchen Sie Beispiele für Agentensystem im Beruf oder Alltag!

AGENTIC AI



**Agentic AI – Von LLMs zu autonomen
Agentensystemen**

Klassische Agenten

- Reaktiv:
Agent trifft Entscheidungen direkt auf Basis sensorischer Eingaben, führt keine Planung durch.
- Deliberativ:
Agenten besitzt internes Modell der Welt und plant Handeln explizit
- BDI (Belief–Desire–Intention):
Agent mit Wissen bzw. Annahmen über die Welt und Zielen/Wünschen, die der Agent erreichen möchte, und Agent verfolgt Ziele.

LLM-Beitrag

- LLMs als generelle Problemlöser durch Sprachverständnis und Weltwissen
- Frage: Wie wird aus einem Sprachmodell ein handlungsfähiger Agent?

Definition Agentic AI

- Agentic AI = Systeme, die mit LLMs autonom planen, handeln und lernen
- Charakteristika: Zielorientierung, Tool-Nutzung, Gedächtnis, Selbstreflexion
- Abgrenzung: Chatbot → reagiert / Agent → agiert

Mehr als ein Buzzword?

- Begriff wird oft inflationär und teilweise missbräuchlich verwendet
- FAQ-Bots = „Agenten“ ?
- Workflows mit 1–2 API-Calls = „autonome Systeme“
- LLM-Plugins werden als „Agenten“ vermarktet

Agentic AI bezeichnet Systeme, die...

- Ziele verfolgen statt nur Antworten geben,
- mehrere Schritte planen,
- Tools ausführen (APIs, Code, Systeme),
- Reflexion & Selbstkorrektur einsetzen,
- ein Memory besitzen
- mit anderen Agenten interagieren können.

Moderne Agentensysteme - Struktur

1. LLM (Core Reasoner): Sprachmodell als Denkmodul
2. Planner / Task Composer: Ziele in Teilaufgaben übersetzen
3. Memory: Langzeit- und Kurzzeitspeicher (z. B. Vektordatenbank)
4. Tool-Use Layer: API- und Funktionsaufrufe
5. Execution Environment: Framework (z. B. LangChain, CrewAI)

ReAct Pattern

- Kombiniert Reasoning + Acting in iterativen Schleifen
- Struktur: Thought → Action → Observation → Thought → ...
- Beispiel: Planen, Handeln, Beobachten

Memory & State

- Kurzzeitgedächtnis: aktueller Kontext der Konversation
- Langzeitgedächtnis: persistente Wissensbasis (RAG)
- Episodisches Gedächtnis: Erfahrungen und Selbstkorrekturen
- Reflektieren, Replanen

Tool-Use & Environments

- Agenten können aktiv handeln (z. B. Websuche, Code-Ausführung)
- Integration über Function Calling oder Tool-APIs
- Frameworks: LangChain, HuggingGPT

Agentic Loops

- Typischer Zyklus: Plan → Execute → Reflect → Replan
- Manager/Worker oder Actor/Critic-Architekturen
- Beispiel: CrewAI, AutoGPT

- Systeme, in denen mehrere LLM-getriebene Agenten parallel oder sequentiell zusammenarbeiten.
- Jeder Agent besitzt eigene Rollen, Fähigkeiten und Ziele.
- Kommunikation erfolgt über natürliche Sprache, strukturiert (JSON/Tools) oder frei

Interaktionsformen

Kooperation:

- Agenten lösen Aufgaben gemeinsam, teilen Wissen, korrigieren gegenseitig Fehler.
- Beispiel: „Researcher-Agent + Writer-Agent + Critic-Agent“.

Kompetition:

- Agenten verfolgen unterschiedliche Strategien oder Hypothesen.
- Vorteil: Exploration mehrerer Lösungswege → höhere Qualität.
- Beispiel: „Debating Agents“ zur Entscheidungsfindung.

Hierarchische Kontrolle:

- Manager-Agent plant → Worker-Agenten führen aus.
- Typisch in Frameworks wie CrewAI, LangGraph, AutoGen.

Multi-Agent-Systeme

Kommunikation („Societies of Mind“)

LLM-Agenten interagieren über protokollierte Messages, ähnlich wie menschliche Teams.

Mögliche Kommunikationsmuster:

- Broadcast: ein Agent verteilt Informationen an alle
- Peer-to-Peer: zielgerichtete Kommunikation zwischen Agenten
- Consensus Building (Diskurse, Abstimmungen)

Ziel: kollektive Intelligenz statt Einzelagenten-Leistung.

Typische Rollen in Agentic-AI-Multiagentensystemen

- Planner / Manager – Aufgaben zerlegen, Prozesse steuern
- Researcher – Informationen beschaffen, Web-Scraping, Literatur
- Analyst – Daten bewerten, Risiken analysieren
- Coder – Code generieren oder reparieren
- Critic / Evaluator – Qualitätskontrolle, Fehlerprüfung
- Memory-Agent – externe Wissensspeicher verwalten (RAG)

Zentrale Herausforderungen

- Koordinationslogik: Wie entscheiden mehrere Agenten gemeinsam?
- Konsistenz: Unterschiedliche Agentenantworten zusammenführen.
- Kosten- & Ressourcenmanagement: Viele Agenten → viele Modellaufälle.
- Sicherheit: Ein Agent darf keinen gefährlichen Tool-Call triggern.
- Fehlerverhalten: Agenten können sich gegenseitig in Halluzinationen verstärken.

Anwendungen

- Simulationen sozialer Systeme
 - Ökonomie, Stadtentwicklung, Organisationsverhalten
- Wissensmanagement
 - mehrere Expertenagenten erzeugen, verknüpfen und pflegen Wissensbasen
- Software Engineering
 - Team aus Coder, Tester, Reviewer, Architekt-Agents
- Customer Service
 - Routing-Agent, Problemlöser-Agent, Eskalations-Agent
- Forschung / Wissenschaft
 - Literaturextraktion,
 - Hypothesengenerierung, Peer-Review-Simulation

Evaluation & Benchmarks

- Herausforderungen: Reproduzierbarkeit, Kontrolle, Sicherheit
- Metriken: Task Success Rate, Plan Length, Reflection Accuracy
- Benchmarks: GAIA, WebArena, AgentBench

Risiken & Grenzen

- Tool-Missbrauch, unkontrollierte Aktionen, Alignment-Probleme
- Datenschutz bei persistentem Memory
- Verantwortung und Haftung bei autonomen Entscheidungen

Planungs- und Suchverfahren

Moderne Agentic-AI-Frameworks greifen weiterhin auf klassische Verfahren zurück, aber oft kombiniert mit LLMs.

Planungs- und Suchverfahren

- Wird genutzt für Zielfindung, Task-Decomposition, Strategieplanung.
- A*-Suche, Best-First Search Heuristische Planung
- Symbolische Planung (STRIPS, PDDL)
- HTN Planning (Hierarchical Task Networks) Constraint-Planning (CSP)
- Beispiel in Agentic AI: Ein LLM-Agent erzeugt einen Zielplan, ein klassischer Planner optimiert und prüft Konsistenz.

Logik- und Wissensrepräsentation

- Prädikatenlogik
- Ontologien (OWL)
- Wissensgraphen
- Symbolische Regeln / Rule-Based Systems
- Einsatz: LLM-Agenten nutzen Wissensgraphen für präzise Abfragen, um Halluzinationen zu reduzieren (Neuro-symbolische Systeme).

Multi-Agent-Koordination

- Auktionsmechanismen (Contract Net Protocol)
- Kooperative Spieltheorie
- Agentenkommunikationssprachen (KQML, FIPA-ACL)
- Schwarmautonomie / verteilte Systeme
- Einsatz: LLM-Agenten „verhandeln“ über sprachliche Nachrichten, aber die Koordinationslogik orientiert sich weiterhin an klassischen MAS-Methoden.

Planungs- und Suchverfahren

Das Herzstück moderner Agenten.

- LLMs / Transformer-Modelle
- GPT, Llama, Mistral, Claude etc.
- Zero-shot-, few-shot-, chain-of-thought reasoning
- ReAct (Reason + Act)
- Reflexion („self-correction“).

Reinforcement Learning

Wird genutzt für:

- Aktionsauswahl
- Strategieoptimierung
- Belohnungsmodelle (Reward Shaping)
- LLM-RL-Agents, die auf Feedback aus ihrer Umgebung lernen.

RAG (Retrieval-Augmented Generation)

RAG ist zentral, da Agenten persistentes Gedächtnis benötigen. Agenten nutzen Wissensdatenbanken.

- Vektorsuchen (FAISS, Chroma, Pinecone)
- Dokumentembedding
- Memory Agents

Self-Reflection / Self-Verification

Das sind meta-ML-Verfahren, die erst durch Agentic AI relevant werden.

- LLM-basierte Verfahren
- Reflexion Loops
- Critic-Agenten
- Self-Evaluation
- Test-Time Compute (wie ToT, GoT)

Hybride Verfahren (Kombination klassisch + ML) in Agentic AI

Neuro-symbolische Systeme Kombinieren

- Symbolisches Wissen
- LLM-Generatives Wissen
- Beispiel: LLM plant → symbolischer Planner validiert → Agent handelt Reinforcement Learning

LLMs als Controller + klassische Tools als Executor

Beispiel

- LLM „entscheidet“
- Pandas/NumPy/SQL sozusagen „handeln“

Das typische Agentsystem

- Reasoner (LLM) Tool-Executor
(klassische SW) Memory (Vector DB)

LLMs + RL (Hybrid)

LLMs übernehmen:

- Planung Strategie Selbstreflexion
- RL übernimmt: Optimierung Ausprobieren Belohnungslernen

Emergent Reasoning in Agentenhaushalten

- Debating Agents
 - „Mein Ansatz ist effizienter, weil...“
 - „Aber Methode B hat geringere Risiken...“
- Critic/Analyst Patterns
- Manager/Worker Architecture
- Swarm-Agenten

Kooperationsmechanismen Contract-Net-Protokolle

- Rollenverteilung
- Voting & Consensus
- Auktionsmechanismen

- LLMs verhandeln über einen Kommunikations- und Abstimmungsprozess zwischen mehreren Agenten in natürlicher oder seminatürlicher Sprache
- Verhandeln als technisches Abstimmungsverfahren, das über natürliche Sprache ausgedrückt wird.

Konsensbildung

Mehrere Agenten stimmen über eine Entscheidung ab.

- Voting (Mehrheitsentscheidung)
 - Bewertung über Scoring
 - Priorisierung / Ranking
- „Ich empfehle Option 1 aus Gründen A, B und C.“

Verfahren in Agentic AI – Zusammenfassung

Systemteil	Klassische KI	ML/LLM
Zielplanung	STRIPS/HTN	LLM-Task-Decomposition
Logikprüfung	Regeln, Ontologien	LLM-Reasoning
Wissensspeicher	Ontologien	Vektor-DB (RAG)
Handlungen	Symbolische Tools	LLM Function-Calling
Koordination	MAS-Protokolle	Sprachbasierte Koordination
Selbstreflexion	kaum vorhanden	Self-critic, reflection Loops
Lernen	RL, Heuristik	RLHF, Selbstlernen

Verfahren in Agentic AI – Logik- und Wissensrepräsentation

- Prädikatenlogik
- Ontologien (OWL)
- Wissensgraphen
- Symbolische Regeln / Rule-Based Systems
- Einsatz:
LLM-Agenten nutzen Wissensgraphen für präzise Abfragen, um Halluzinationen zu reduzieren (Neuro-symbolische Systeme).

Anwendungen

- Simulationen sozialer Systeme
Ökonomie, Stadtentwicklung, Organisationsverhalten, Bewerber Screening
- Wissensmanagement
mehrere Expertenagenten erzeugen, verknüpfen und pflegen Wissensbasen, Inhalte automatisiert erzeugen
- Software Engineering
Team aus Coder, Tester, Reviewer, Architekt-Agents
- Customer Service
Routing-Agent, Problemlöser-Agent, Eskalations-Agent
- Infrastruktur
SmartGrids, Energieerzeugung
- Forschung / Wissenschaft
Literaturextraktion, Hypothesengenerierung, Peer-Review-Simulation

Meist noch eher in Pilotierung bzw. Proof-of-Concept-Stadium

Modernisierung von 400+ Legacy-Applikationen einer Bank

- Technische Schulden, komplexe Abhängigkeiten, hoher Aufwand
- Ziel: Analyse, Dokumentation und Code-Modernisierung automatisieren

Research-Agent – Analyse von Code & Dokumentation

- Analysis-Agent – Architektur, Abhängigkeiten, Risiken
- Documentation-Agent – automatische technische Dokumentation
- Code-Agent – Refactoring-Vorschläge, Code-Generierung, Tests
- Critic-Agent – Qualitätskontrolle & Fehlerkorrektur
- Ablauf: Plan → Execute → Reflect → Replan (mit Human-Supervisor)

Über 50 % weniger Zeit für Analyse & Dokumentation

- Parallelbearbeitung mehrerer Systeme möglich
- Qualitätssteigerung durch Agenten-Reflexionsschleifen
- Eines der ersten großen realen Multi-Agent-Systeme

McKinsey / QuantumBlack – Agentic AI Anwendung

Bawcom, A., Fitzpatrick, M., Cheung, C. W., Collins, D., & Gabrielli, D. (2024). AI for IT modernization.

<https://www.mckinsey.com/capabilities/quantumblack/our-insights/ai-for-it-modernization-faster-cheaper-and-better>

Bawcom, A., Fitzpatrick, M., Cheung, C. W., Collins, D., & Gabrielli, D. (2025). Seizing the agentic AI advantage.

<https://www.mckinsey.com/capabilities/quantumblack/our-insights/seizing-the-agentic-ai-advantage>

McKinsey & Company. (2025). McKinsey's LegacyX: Rejuvenating legacy infrastructure with agentic AI.

<https://www.mckinsey.com/about-us/new-at-mckinsey-blog/mckinseys-legacyx-rejuvenating-legacy-infrastructure-with-agentic-ai>

QuantumBlack AI. (2025). Overcoming legacy tech through agentic AI. Medium.

<https://medium.com/quantumblack/overcoming-legacy-tech-through-agentic-ai-ca0d82e0d118>

Ausblick

- Von Einzelagenten zu Agentic Societies
- Koordination durch natürliche Sprache
- Verbindung klassischer Agentenlogik mit moderner LLM-Intelligenz

Zusammenfassung

- LLM → Reasoner | Agentic AI → Reasoner + Actor + Reflector + Memory
- Agentic AI verbindet KI-Agententheorie mit LLM-Systemen
- Ziel: proaktive, lernfähige und verantwortliche KI-Agenten