edv-buchversand.de

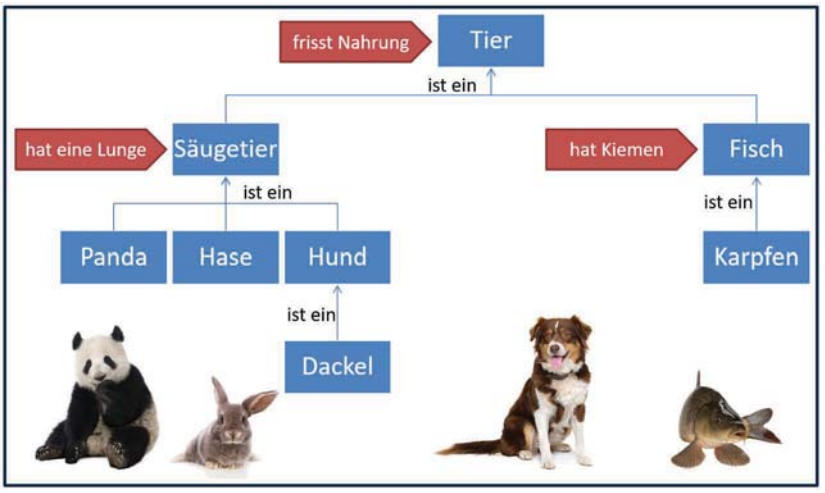
Künstliche Intelligenz selber programmieren
für Dummies Junior

» Hier geht's direkt zum Buch

DIE LESEPROBE

Diese Leseprobe wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

Kapitel 1
Denken



Panda: © Eric Isselée; Hase: drubig-photo; Hund: Erik Lam; Fisch: Sergey Goruppa. Alle stock.adobe.com.

Ich denke, also bin ich. Gilt das auch für Computer? In diesem Kapitel wollen wir dem Computer beibringen zu denken – wenigstens ein bisschen! Hierfür erklären wir dir, auf welch unterschiedliche Arten man denken kann.

Wie denken eigentlich Menschen?

Wir Menschen denken fast die ganze Zeit etwas, oft merken wir es gar nicht. Manchmal aber schon: Zum Beispiel, wenn du gefragt wirst, was 56 plus 18 ist, dann denkst du vielleicht: »Okay, 56 plus 10 ist 66 und jetzt muss ich noch 8 dazu tun, das sind dann 66 plus 8, also 74.« Vielleicht hättest du diese Rechenaufgabe auch anders gelöst, vielleicht hättest du erst plus 4 gerechnet und dann plus 14, aber du hättest sicher mehrere Rechenschritte nacheinander ausgeführt.

Wenn man einem Computer eine Aufgabe gibt, führt er auch meistens mehrere Rechenschritte nacheinander durch. Aber: Die sind vorprogrammiert. Das heißt, das Addieren von zwei Zahlen wird immer auf die gleiche Art erledigt. Du dagegen wirst beim Kopfrechnen unterschiedliche Lösungswege wählen – je nachdem welche Zahlen zu addieren sind. Wenn du dagegen schriftlich addierst, machst du es nach einer festen Vorschrift, wie du es im Matheunterricht gelernt hast, also ähnlich wie ein Computer.

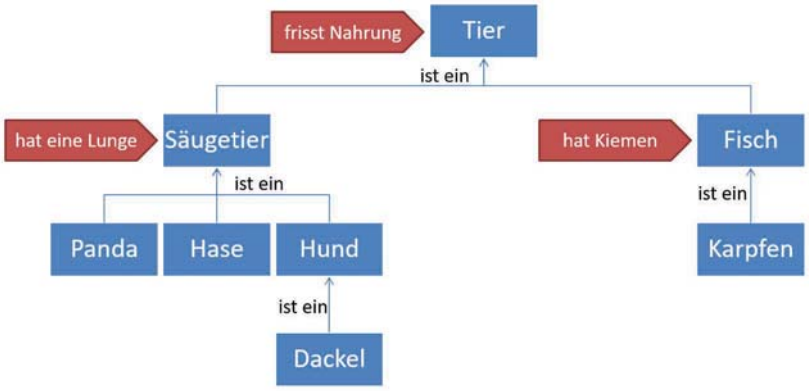
Häufig kann man nicht sofort mit dem Programmieren beginnen, sondern muss die Arbeitsschritte und Rechenvorschriften für Programme erst einmal allgemeiner beschreiben. Erst wenn man sich einen Überblick über die wichtigsten Schritte verschafft hat, kann man sich Detailfragen widmen.

Eine allgemeine Formulierung von Rechenschritten oder Handlungsvorschriften zur Lösung eines Problems heißt Algorithmus.

Das immer gleiche Anwenden von Regeln ist typisch für Computerprogramme. Will man eine Künstliche Intelligenz programmieren, muss man es hinkriegen, Computerprogramme zu schreiben, die nicht so starr nach einem immer gleichen Schema arbeiten, sondern auf verschiedene Situationen flexibel reagieren, ähnlich wie wir Menschen. Wir wollen dir das mal zeigen, wie das gehen kann. Dafür gehen wir mal weg von der Welt der Zahlen und schauen uns ein Beispiel aus der Biologie an. Wenn du gefragt wirst, ob ein Pandabär eine Lunge hat, wirst du sagen: »Ja, klar.« Vielleicht sagst du sogar: »Ja, logisch!«, und damit hast du dann tatsächlich den Nagel auf den Kopf getroffen!

Um die Frage zu beantworten, hast du, wie vorher bei dem Rechenproblem, verschiedene Denkschritte nacheinander ausgeführt – aber du hast es nicht gemerkt. Wir sind uns ziemlich sicher, dass du nicht gelernt hast, dass ein Panda eine Lunge hat. Was du im Biologieunterricht gelernt hast, ist, dass ein Panda ein Säugetier ist und dass Säugetiere Lungen haben. Um die Frage »Hat ein Panda eine Lunge?« zu beantworten, hast du eine Kette von Denkschritten ausgeführt. Genau genommen hast du sogar eine logische Schlussfolgerung gezogen!

Wenn du dir den Ausschnitt aus dem Schaubild zur Einteilung von Tieren anschaust, kannst du deine Denkschritte noch mal bewusst nachvollziehen: Zeige mit dem Finger auf den Panda und gehe immer eine Ebene nach oben, so lange bis du bei einem Wort bist, das mit dem Merkmal »hat eine Lunge« versehen ist.



Du kannst natürlich auch mehr als nur einen Denkschritt ausführen. Wenn dich jemand fragt, ob ein Panda frisst, dann wirst du dem zustimmen. Du weißt, dass ein Panda ein Säugetier ist und dass Säugetiere zu den Tieren gehören. Deshalb kannst du logisch schließen: Ein Panda frisst, weil er ein Säugetier und somit ein Tier ist und Tiere fressen Nahrung. Anders übrigens als Pflanzen, die ihre Nährstoffe mithilfe der Fotosynthese aufnehmen. Aber das ist ein anderes Thema . . .

Das schrittweise Nach-oben-Laufen entspricht einer wichtigen logischen Regel – dem sogenannten *transitiven Schluss*. Den hat schon der alte Grieche Aristoteles vor mehr als 2000 Jahren gekannt und unter dem Namen »modus barbara« zu einer der grundlegenden Regeln für logisches Schließen erklärt.

Aber was hat jetzt Aristoteles mit Künstlicher Intelligenz zu tun? Damals gab es doch gar keine Computer! Es ist tatsächlich so, dass Aristoteles mit seinen Regeln für logisches Schließen eine ganz wichtige Grundlage für Künstliche Intelligenz geschaffen hat. Er hat damit zumindest einen Teil der Art, wie Menschen denken, so beschrieben, dass man daraus ein Computerprogramm machen kann.

Netze ohne Spinnen – dafür mit Knoten und Kanten

Das Schaubild zur Einteilung von Tieren ist aus Sicht der Künstlichen Intelligenz ein *semantisches Netz*. »Netz« deshalb, weil die Begriffe miteinander verbunden sind, wie zum Beispiel die Stationen im Streckennetz der Bahn. »Semantisch« deshalb, weil jedes Wort – Panda, Hund, Säugetier und so weiter – eine Bedeutung hat. Das Fachwort für Bedeutung heißt *Semantik*.

Bei einem semantischen Netz werden die Begriffe als *Knoten* und die Verbindungen dazwischen als *Kanten* bezeichnet. Ein solches Gebilde – egal ob Streckennetz oder semantisches Netz – heißt in der Informatik *Graph*.

Logisches Schließen ist eine spezielle Art zu rechnen. In der Mathematik rechnet man mit Zahlen, in der Logik mit Symbolen, die für etwas stehen. So können wir `hund` schreiben und meinen damit einen Hund. Damit wir nun zum Beispiel berechnen können, ob es stimmt, dass ein Hund ein Tier ist, müssen wir das semantische Netz, das wir als Graph gezeichnet haben, in eine Form bringen, mit der wir logische Schlüsse ausrechnen können. Dazu zerlegen wir das Netz so, dass wir jede Kante mit den zugehörigen Knoten einzeln der Reihe nach aufschreiben:

```
ist_ein(saeugetier,tier)
ist_ein(fisch,tier)
ist_ein(panda,saeugetier)
ist_ein(hase,saeugetier)
ist_ein(hund,saeugetier)
ist_ein(dackel,hund)
ist_ein(karpfen,fisch)
```

Der Graph ist jetzt zu einer Menge von einzelnen Fakten geworden. Auf diese Art kann man beliebige Graphen speichern. Zum Beispiel kannst du ein U-Bahn-Netz als Liste von Paaren speichern, wobei hier die Werte direkt benachbarte Stationen sind.

Die Kanten im Schaubild zur Einteilung von Tieren haben eine Richtung: `("sauegetier", "tier")` *sagt, dass jedes Säugetier ein Tier ist. Die umgekehrte Aussage – jedes Tier ist ein Säugetier – ist falsch, denn es gibt ja verschiedene Tierarten. Neben Säugetieren gibt es zum Beispiel noch Vögel oder Reptilien. Die Richtung haben wir im Schaubild durch Pfeile angezeigt.*

Bei den U-Bahn-Stationen braucht man beide Richtungen – man kommt direkt von Bahnhof Zoo zum Ernst-Reuter-Platz und umgekehrt auch! Das heißt, in der Liste müssen beide Richtungen vorkommen: `("Bahnhof Zoo", "Ernst-Reuter-Platz")` *und* `("Ernst-Reuter-Platz", "Bahnhof Zoo")`.

Wir haben bei den Fakten erst den Namen der Kante geschrieben und dann in Klammern die Knoteninformation. Eine solche Schreibweise nennt man Präfixnotation. *Man könnte auch mathematische Ausdrücke so schreiben. Zum Beispiel kann man 4 + 7 auch so schreiben: +(4,7), was dasselbe bedeutet. Manche Programmiersprachen nutzen eine solche Präfixschreibweise. Man sieht gleich am ersten Symbol, worum es gehen soll, und muss nicht erst weiter schauen. Beispielsweise erkennt man bei 4 – 7 erst nachdem man die 4 gelesen hat, dass es um Subtraktion gehen soll. Bei -(4,7) hat man die Information, was man rechnen soll, gleich als Erstes. Die Symbole in Klammern nennt man auch* Argumente. *Das Anfangssymbol gibt an, in welcher Beziehung die Argumente stehen oder was man mit ihnen tun soll. Das kann eine mathematische Operation wie plus oder minus sein oder eben auch die* `ist_ein`*-Beziehung zwischen zwei Tieren.*

Du möchtest das semantische Netz mit Tieren programmieren? Schau mal in Kapitel 8 – unter »Denken« findest du den Code dazu.

Schlussfolgerndes Denken mit semantischen Netzen

Jetzt haben wir einen Ausschnitt unseres menschlichen Wissens so aufgeschrieben, dass wir damit »rechnen« können. Auf diese Art kann man auch dem Computer das Denken beibringen.

Wissensfragen

Wir fangen ganz einfach an und schauen uns an, wie man reine Wissensfragen mithilfe eines semantischen Netzes beantworten kann. Wir fragen die Beziehungen ab, die schon direkt gegeben sind, also die Fakten, die wir oben aufgeschrieben haben. Das ist so, wie wenn deine Lehrer und Lehrerinnen auswendig gelerntes Wissen abfragen.

Um Fragen wie

- Ist ein Säugetier ein Tier? JA
- Ist ein Fisch ein Karpfen? NEIN

zu beantworten, musst du die Frage in das von uns gewählte Format übersetzen, also zum Beispiel `ist_ein(karpfen, fisch)` und prüfen, ob der Fakt vorhanden ist. Wenn der Fakt da ist, kannst du mit »ja« (das weiß ich) antworten, ansonsten antwortest du »nein« (keine Ahnung, hat mir niemand gesagt).

Schlussfolgerungsfragen

Bei Menschen gilt üblicherweise, dass, wenn jemand die oben genannten Fakten kennt, die Person auch alle Schlussfolgerungen aus diesen Fakten kennt. Beispielsweise gilt, dass, wenn jemand weiß, dass ein Dackel ein Hund ist und ein Hund ein Tier ist, die Person auch weiß, dass ein Dackel ein Tier ist.

Diese Schlussfolgerung basiert auf der oben erwähnten logischen Regel zum Ziehen transitiver Schlüsse. Vielleicht kennst du das *Transitivitätsgesetz* aus der Mathematik: Wenn eine Zahl X kleiner ist als eine Zahl Y und eine Zahl Y kleiner als eine Zahl Z, dann ist X auch kleiner als Z:

Aus X < Y und Y < Z folgt, dass X < Z.

Weil das für alle Zahlen gilt, haben wir die Regel mit sogenannten Variablen `(X, Y, Z)` aufgeschrieben. So können wir das auch für unser semantisches Netz machen:

```
Aus ist_ein(X, Y) und ist_ein(Y, Z) folgt, dass gilt
ist_ein(X,Z).
```

Variablen sind Platzhalter für feste Werte wie konkrete Zahlen oder in unserem Fall Namen von Tieren.

Wenn wir prüfen wollen, ob gilt, dass ein Dackel ein Tier ist, dann setzen wir für `X dackel` und für `Z tier`. Jetzt müssen wir einen Fakt `ist_ein(dackel, Y)` finden. Wir finden `ist_ein(dackel, hund)`. Das heißt, die Variable `Y` wird durch `hund` ersetzt. Das macht man im kompletten Ausdruck. Das heißt, `ist_ein(Y, tier)` wird zu `ist_ein(hund, tier)`. Wir können also, gegeben das Wissen im semantischen Netz mit der Transitivitätsregel, schlussfolgern, dass ein Hund ein Tier ist.

Was wäre, wenn wir fragen würden: Ist ein Tier ein Hund? Dann würden wir analog vorgehen, also `ist_ein(tier, hund)` *anfragen mit* `X=tier` *und* `Z=hund`*. Wir finden keinen Fakt,* `ist_ein(tier, Y)`*, also lautet die Antwort »nein«.*

Komplizierte Schlussfolgerungsfragen

Mit den eingeführten Regeln können wir immer nur zwei `ist_ein`-Kanten miteinander kombinieren. Aber wir wissen ja auch, dass ein Dackel ein Tier ist.

Wenn man Schlüsse über beliebig viele Ebenen ziehen will, muss man die Transitivitätsregel verallgemeinern. Dazu schreiben wir die Regel von oben erstmal etwas um:

Erstens gibt es Fakten, die Paare von Tieren betreffen, die direkt mit einer Kante verbunden sind. Wenn wir über alle diese Fakten gemeinsam reden wollen, können wir wieder Variablen nutzen, also:

```
ist_ein(X,Y)
```

für alle Paare wie `ist_ein(dackel,hund)`, `ist_ein(hund, saeugetier)` und so weiter.

Um transitiv zu schließen, definieren wir uns eine Regel namens `istein*(X,Y)`. Schreiben wir erstmal auf, was wir schon definiert haben:

```
istein*(X,Y) wenn gilt ist_ein(X,Y)
istein*(X,Y) wenn gilt ist_ein(X,Z) und ist_ein(Z,Y)
```

Die erste Zeile der Regel besagt, dass wir wissen, dass ein `X` ein `Y` ist, wenn uns der Fakt direkt bekannt ist. Die zweite Regel besagt, dass wir auch wissen, dass ein `X` ein `Y` ist, wenn wir die Fakten kennen, dass dieses `X` ein `Z` ist und dass `Z` ein `Y` ist.

Oben haben wir die Transitivitätsregel in einer anderen Reihenfolge aufgeschrieben und nach `ist_ein(X,Z)` *gefragt. Jetzt fragen wir nach* `istein*(X,Y)`*, damit wir die gleichen Variablen nehmen, wie in der ersten Zeile. Wir hätten hier auch wieder* `X` *und* `Z` *wählen können wie oben. Das ist aber etwas verwirrend. Es ist egal, wie wir die Variablen nennen, wir müssen es nur einheitlich tun.*

Mit den beiden Regeln können wir nun nicht mehr als vorher: Direkte Fakten prüfen und Beziehungen zwischen Tieren, die über zwei `ist_ein`-Kanten verbunden sind. Die zweite Regel können wir jetzt noch allgemeiner schreiben, um schließen zu können: Ein `X` ist ein `Y`, wenn es eine Kante `ist_ein(X,Z)` gibt und wenn transitiv gilt, dass ein `Z` ein `Y` ist. In der oben eingeführten Regelform sieht das so aus:

```
istein*(X,Y) wenn gilt ist_ein(X,Z) und istein*(Y,Z)
```

Wir haben einfach den Namen der Regel nochmal genutzt. Dieses Prinzip heißt *Rekursion*.

Rekursive Definitionen werden in der Informatik und in der Mathematik immer dann verwendet, wenn man etwas mit unterschiedlich vielen Schritten, aber immer auf die gleiche Art, berechnen will. Rekursion kommt aus dem Lateinischen: recurrere *heißt zurücklaufen.*

Probieren wir es aus: Wir fragen, ob gilt, dass ein Dackel ein Tier ist. Dazu rufen wir die Regel auf mit

```
istein*(dackel,tier)
```

Die erste Regel prüft:

```
istein*(dackel,tier) wenn gilt ist_ein(dackel,tier)
```

Diesen Fakt finden wir nicht in unserer Liste, also probieren wir die zweite, rekursive Regel:

```
istein*(dackel,tier) wenn gilt ist_ein(dackel, Z) und
istein*(Z,tier)
```

Wir finden den Fakt `ist_ein(dackel,hund)` und ersetzen die Variable `Z` mit `hund`. Im zweiten Teil der Regel müssen wir prüfen `istein*(hund,tier)`. Wir schauen wieder zuerst die erste Regel an:

```
istein*(hund,tier) wenn gilt ist_ein(hund,tier)
```

Wieder finden wir keinen entsprechenden Fakt. Also gehen wir wieder zur zweiten Regel:

```
istein*(hund,tier) wenn gilt ist_ein(hund,Z) und
istein*(Z,tier)
```

Wir finden den Fakt `ist_ein(hund,saeugetier)` und ersetzen die Variable `Z` im aktuellen Regelaufruf mit `saeugetier`. Und wieder müssen wir noch den zweiten Teil der Regel prüfen, nämlich `istein*(saeugetier,tier)`. Wir schauen wieder zuerst die erste Regel an und siehe da:

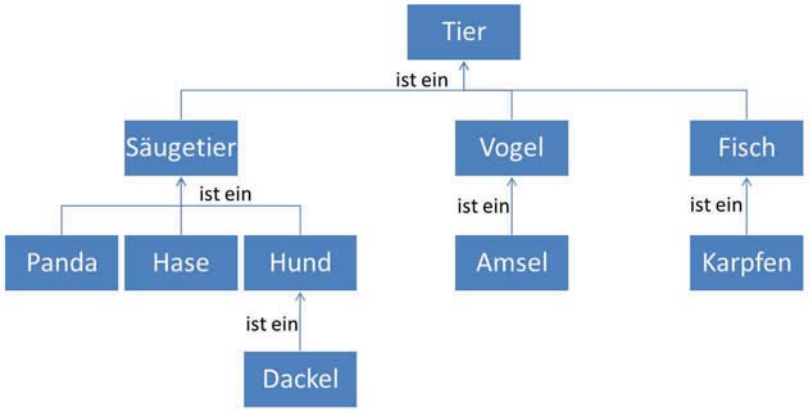
```
istein*(saeugetier,tier) gilt wenn ist_ein(saeugetier,tier)
```

gilt. Diesen Fakt haben wir ja. Nach drei Schritten haben wir also die Schlussfolgerung geschafft! Wir konnten mit den Regeln ausrechnen, dass gilt, dass ein Dackel ein Tier ist.

Baue dein eigenes semantisches Netz

Du kannst das semantische Netz um weitere Begriffspaare erweitern. In der Grafik vom Anfang des Kapitels würdest du also weitere blaue Kästchen einfügen. Auf welcher Ebene du Kästchen einfügst, ist dir überlassen. Du kannst zum Beispiel noch weitere Hundearten auf der Ebene einfügen, auf der bereits »Dackel« steht. Oder du wirst nochmal spezieller und fügst ein, dass ein Rauhaardackel ein Dackel ist. Du kannst aber auch eine Ebene darüber noch weitere Säugetiere hinzufügen. Oder Vögel.

Das war jetzt nur ein Beispiel, wie du dein semantisches Netz erweitern kannst. Dir fallen bestimmt noch andere Tiere ein.



Natürlich kannst du unser Netz auch als Vorlage für andere semantische Netze nehmen, die von ihrer Struktur her ähnlich sind. Versuche doch mal ein semantisches Netz mit Automarken oder mit deinen Lieblingsstars aus Musik, Serien und Filmen zu erstellen.

Ein Python-Programm für transitives Schließen in semantischen Netzen haben wir für dich auf der Webseite des Verlags hinterlegt. Alternativ kannst du auch die Programmiersprache Prolog nutzen, die speziell für das Ziehen von Schlüssen gemacht ist. Hierzu findest du eine Einführung unter https://www.inf-schule.de/deklarativ/logischeprogrammierung.

In Kapitel 8 geben wir dir ein Programmierbeispiel, wie du dein semantisches Netz erweitern kannst.

Was heißt eigentlich »Denken«?

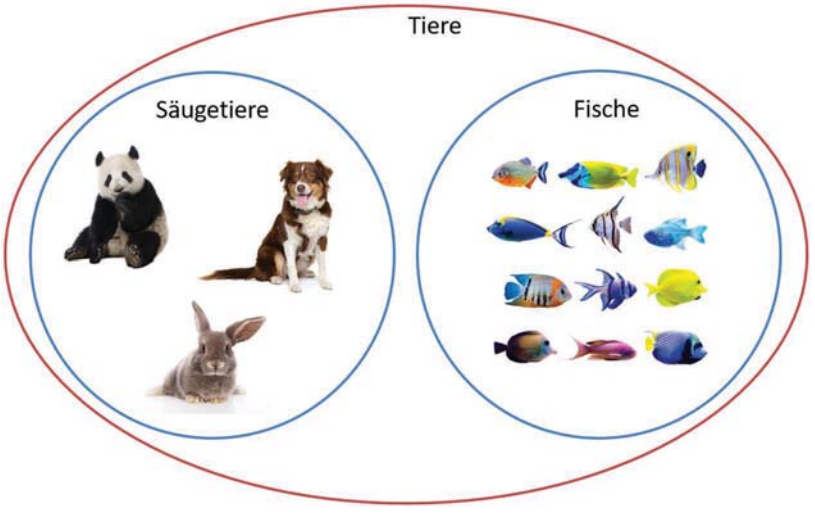
Du hast nun eine wichtige Regel kennengelernt, mit der man logische Schlüsse ziehen kann. Damit kann eine spezielle Art abgebildet werden, wie wir Menschen denken, nämlich indem wir transitive Schlüsse ziehen. Es gibt verschiedene Arten, wie man über Dinge nachdenken kann. Wir wollen sie dir hier kurz vorstellen.

Deduktives Denken

Die Art des Denkens, die wir bisher betrachtet haben, nennt sich schlussfolgerndes Denken oder – mit einem Fachwort aus dem Lateinischen – *Deduktion*. Die einfache, nicht-rekursive Transitivitätsregel, die du programmiert hast, kann man auch schreiben als:

Zusammenhang	Beispiel
A → B	Panda → Säugetier
B → C	Säugetier → Tier
A → C	Panda → Tier

Das heißt: Wenn gilt, dass ein A ein B ist (ein Panda ein Säugetier ist), und wenn gilt, dass ein B ein C ist (ein Säugetier ein Tier ist), dann folgt daraus logisch, dass ein A ein C ist (ein Panda ein Tier ist).



Panda: © Eric Isselée; Hase: drubig-photo; Hund: Erik Lam; Fische: tan4ikk. Alle stock.adobe.com.

Die logische Schlussregel (Deduktion) ist immer bei Aussagen möglich, die diesem Muster folgen:

- Peter ist größer als Maria und Maria ist größer als Fritz, also ist Peter größer als Fritz.
- Paris liegt in Frankreich und Frankreich liegt in Europa, also liegt Paris in Europa.
- Alle Radfahrer sind umweltbewusst und alle Umweltbewussten kaufen in Bioläden, also kaufen alle Radfahrer in Bioläden.

Der letzte Schluss ist doch ein bisschen merkwürdig. Wir jedenfalls kennen mindestens einen Radfahrer, der nicht im Bioladen einkauft. Ist die Logik doch nicht so logisch? Doch, schon, aber nur dann, wenn die einzelnen Behauptungen, die man verwendet, wahr sind! Sind wirklich alle Radfahrer umweltbewusst? Die logische Schlussregel sagt nur, wie wir Aussagen korrekt verknüpfen können, aber nichts darüber, ob die Aussagen selber wahr sind.

Viele unserer Denkprozesse folgen deduktiven Mustern. Eines der ersten großen Ziele der Künstlichen Intelligenz in den 1960er-Jahren war deshalb, Programme zu entwickeln, die logisch denken können.

Abduktives Denken

Wir Menschen nutzen noch andere Arten des Denkens. Wenn du zum Arzt gehst, weil du starken Husten hast, denkt der Arzt vielleicht so:

- Ich habe im Studium gelernt, dass Bronchitis sich so auswirkt, dass man viel hustet.
- Der Patient hustet.
- Also diagnostiziere ich Bronchitis.

So eine Schlussregel heißt *Abduktion* – auch ein lateinisches Fachwort. Diese Schlussregel ist nicht im strengen Sinne logisch. Der Arzt kann nämlich falsch liegen. Der Husten könnte auch eine andere Ursache haben. Vielleicht hat sich die Person ja einfach nur verschluckt.

Sherlock Holmes und Dr. Watson

Kennst du den berühmten Detektiv aus London, Sherlock Holmes, der mit seinem Mitarbeiter Dr. Watson die kniffeligsten Fälle löst? Dieser berühmte Detektiv behauptet in seinen Abenteuern oft, er würde deduktiv schließen. Meistens schließt er aber abduktiv: »Wer rennt, hinterlässt nur halbe Fußabdrücke. Hier sind halbe Fußabdrücke. Also ist hier jemand gerannt.« Das kann sein, es kann aber auch einen anderen Grund haben, zum Beispiel, dass jemand auf Zehenspitzen gelaufen ist.

Es gibt auch Künstliche-Intelligenz-Systeme, die die abduktive Art des Denkens nachbilden. Das sind Diagnosesysteme.

Ein ganz altes System, das sehr bekannt ist, heißt *Mycin*. Das ist ein System, das wie ein guter Arzt ansteckende Krankheiten mit hoher Trefferquote richtig erkennt.

Induktives Denken

Noch eine andere Art zu denken, ist die *Induktion* (ja, wieder ein lateinischer Fachbegriff). Das funktioniert so:

- Ich habe im Urlaub einen Schwan gesehen, der war weiß.
- Vorgestern habe ich auf dem Stadtteich einen Schwan gesehen, der war auch weiß.

» Und jetzt sehe ich einen Schwan am Flussufer und der ist auch weiß.

» Also nehme ich mal an, dass alle Schwäne weiß sind.

Auch die Induktion ist kein logischer Schluss. Wir haben aus einer kleinen Menge von Beobachtungen auf eine allgemeine Regel geschlossen. Aber irgendwann werden wir auf einen schwarzen Schwan treffen! Eine induktive Schlussfolgerung kann also auch falsch sein.

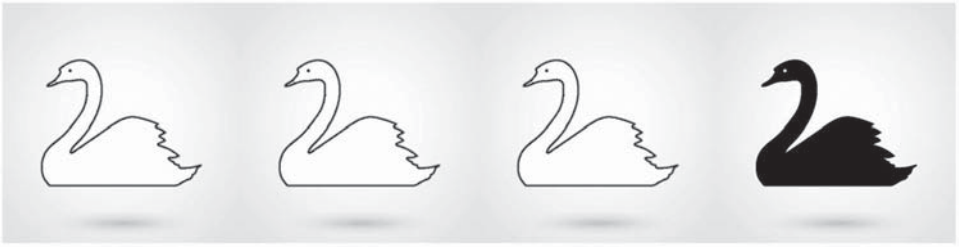


Illustration: © Erik Lam – stock.adobe.com

Mit Induktion werden wir uns im nächsten Kapitel genauer beschäftigen. Das ist nämlich eines der wichtigsten Prinzipien, nach denen Lernen funktioniert.

Denken mit Wahrscheinlichkeiten

Logisches Denken ist immer ein »ganz oder gar nicht«: Entweder es gilt, dass ein Panda ein Tier ist, oder es gilt eben nicht. Aber unser Denken und Schlussfolgern beruht nicht immer nur auf »harten Fakten«. Der Wetterbericht zum Beispiel sagt nicht, dass es morgen ganz sicher regnet oder ganz sicher nicht. Ob es morgen regnen wird, ist mehr oder weniger wahrscheinlich.

Zu spät oder krank?

Du denkst, ohne es zu merken, oft mit Wahrscheinlichkeiten. Wenn beispielsweise eine Klassenkameradin, die sehr zuverlässig ist, um 08:05 Uhr noch nicht im Klassenzimmer ist, nimmst du an, dass sie krank ist und nicht mehr kommt. Wenn aber eine eher chaotische Klassenkameradin noch nicht da ist, nimmst du eher an, dass sie zu spät kommt.

Es gibt Künstliche-Intelligenz-Programme, die das menschliche Schließen mit Wahrscheinlichkeiten – mit unsicherem Wissen – nachbauen. Zum Beispiel kann man Wissen in Bayes'schen Netzen (benannt nach dem Mathematiker Thomas Bayes) abbilden.

Ein Bayes'sches Netz funktioniert so ähnlich wie ein semantisches Netz. Aber hier werden Knoten und Kanten mit Wahrscheinlichkeiten versehen. So könntest du dir zum Beispiel nur zu 80 % sicher sein, dass das Tier, das du da siehst, ein Panda ist. Außerdem glaubst du nur zu 60 %, dass ein Panda ein Säugetier ist und so weiter. Schlussfolgern in einem solchen Netz heißt, dass man die Wahrscheinlichkeiten auf eine bestimmte Weise verrechnet. So könnte am Ende rauskommen, dass das Künstliche-Intelligenz-System sich zu 70 % sicher ist, dass ein Panda ein Tier ist.

Das klingt jetzt etwas merkwürdig, weil wir uns mit so großen Tieren gut auskennen. Aber stell dir ein semantisches Netz für Pilze vor: Da kann es schon mal vorkommen, dass man sich nicht sicher ist, ob ein Pilz giftig ist oder nicht.