mitp

Ian Goodfellow

Yoshua Bengio

Aaron Courville

Deep Learning

Das umfassende Handbuch

Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze

Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Ian Goodfellow
Yoshua Bengio
Aaron Courville

Deep Learning

Das umfassende Handbuch

Grundlagen, aktuelle Verfahren und
Algorithmen, neue Forschungsansätze

Übersetzung aus dem Amerikanischen
von Guido Lenz

mitp

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie. Detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-95845-701-0

1. Auflage 2018

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 7189 - 079
Telefax: +49 7953 7189 - 082

Authorized German translation from the English language edition, entitled Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
ISBN 9780262035613
Original English language edition published by The MIT Press, a department of Massachusetts Institute of Technology

Lektorat und fachliche Prüfung: Sabine Schulz, Janina Bahlmann, Lisa Kresse
Sprachkorrektorat: Petra Heubach-Erdmann
Fachkorrektur Mathematik: Stefan Niedergriese
Fachkorrektur Deep Learning: Alexander Bresk
Coverbild: madgooch/ stock.adobe.com
Satz: Dr. Joachim Schlosser, www.schlosser.info

Inhaltsverzeichnis

Website zum Buch

Auf der Website des Verlags finden Sie Hinweise zur Übersetzung sowie Errata, sofern Fehler bekannt sind unter

`www.mitp.de/700`

Darüber hinaus gibt es zu diesem Buch eine englischsprachige Website. Sie enthält ergänzende Materialien auf Englisch, darunter Übungen, Vortragsfolien, Korrekturen und andere Ressourcen, die für Leser und Lehrkräfte hilfreich sind. Diese finden Sie unter

`www.deeplearningbook.org`

Danksagung

Dieses Buch wäre ohne die Hilfe vieler Menschen nicht entstanden.

Wir danken all denen, die unser Buchkonzept unterstützt und uns dabei geholfen haben, seinen Inhalt und Aufbau zu planen: Guillaume Alain, Kyunghyun Cho, Çağlar Gülçehre, David Krueger, Hugo Larochelle, Razvan Pascanu und Thomas Rohée.

Wir danken auch den Menschen, die uns Feedback zum Text selbst gegeben haben. Einige davon haben zu vielen Kapiteln Rückmeldung gegeben: Martín Abadi, Guillaume Alain, Ion Androutsopoulos, Fred Bertsch, Olexa Bilaniuk, Ufuk Can Biçici, Matko Bošnjak, John Boersma, Greg Brockman, Alexandre de Brébisson, Pierre Luc Carrier, Sarath Chandar, Pawel Chilinski, Mark Daoust, Oleg Dashevskii, Laurent Dinh, Stephan Dreseitl, Jim Fan, Miao Fan, Meire Fortunato, Frédéric Francis, Nando de Freitas, Çağlar Gülçehre, Jurgen Van Gael, Javier Alonso García, Jonathan Hunt, Gopi Jeyaram, Chingiz Kabytayev, Lukasz Kaiser, Varun Kanade, Asifullah Khan, Akiel Khan, John King, Diederik P. Kingma, Yann LeCun, Rudolf Mathey, Matías Mattamala, Abhinav Maurya, Kevin Murphy, Oleg Mürk, Roman Novak, Augustus Q. Odena, Simon Pavlik, Karl Pichotta, Eddie Pierce, Kari Pulli, Roussel Rahman, Tapani Raiko, Anurag Ranjan, Johannes Roith, Mihaela Rosca, Halis Sak, César Salgado, Grigory Sapunov, Yoshinori Sasaki, Mike Schuster, Julian Serban, Nir Shabat, Ken Shirriff, Andre Simpelo, David Slate, Scott Stanley, David Sussillo, Ilya Sutskever, Carles Gelada Sáez, Graham Taylor, Valentin Tolmer, Massimiliano Tomassoli, An Tran, Shubhendu Trivedi, Alexey Umnov, Vincent Vanhoucke, Marco Visentini-Scarzanella, Martin Vita, David Warde-Farley, Dustin Webb, Kelvin Xu, Wei Xue, Ke Yang, Li Yao, Zygmunt Zając und Ozan Çağlayan.

Unser Dank geht auch an jene, deren Feedback einzelnen Kapiteln gewidmet war:

- Notation: Zhang Yuanhang
- Kapitel 1, Einleitung: Yusuf Akgul, Sebastien Bratieres, Samira Ebrahimi, Charlie Gorichanaz, Brendan Loudermilk, Eric Morris, Cosmin Pârvulescu und Alfredo Solano
- Kapitel 2, Lineare Algebra: Amjad Almahairi, Nikola Banić, Kevin Bennett, Philippe Castonguay, Oscar Chang, Eric Fosler-Lussier, Andrey Khalyavin, Sergey Oreshkov, István Petrás, Dennis Prangle, Thomas Rohée, Gitanjali Gulve Sehgal, Colby Toland, Alessandro Vitale und Bob Welland
- Kapitel 3, Wahrscheinlichkeits- und Informationstheorie: John Philip Anderson, Kai Arulkumaran, Vincent Dumoulin, Rui Fa, Stephan Gouws, Artem Oboturov, Antti Rasmus, Alexey Surkov und Volker Tresp
- Kapitel 4, Numerische Berechnung: Tran Lam AnIan Fischer und Hu Yuhuang
- Kapitel 5, Grundlagen für das Machine Learning: Dzmitry Bahdanau, Justin Domingue, Nikhil Garg, Makoto Otsuka, Bob Pepin, Philip Popien, Bharat Prabhakar, Emmanuel Rayner, Peter Shepard, Kee-Bong Song, Zheng Sun und Andy Wu
- Kapitel 6, Tiefe Feedforward-Netze: Uriel Berdugo, Fabrizio Bottarel, Elizabeth Burl, Ishan Durugkar, Jeff Hlywa, Jong Wook Kim, David Krueger, Aditya Kumar Praharaj und Sten Sootla
- Kapitel 7, Regularisierung: Morten Kolbæk, Kshitij Lauria, Inkyu Lee, Sunil Mohan, Hai Phong Phan und Joshua Salisbury
- Kapitel 8, Optimierung beim Trainieren von tiefen Modellen: Marcel Ackermann, Peter Armitage, Rowel Atienza, Andrew Brock, Tegan Maharaj, James Martens, Mostafa Nategh, Kashif Rasul, Klaus Strobl und Nicholas Turner
- Kapitel 9, CNNs: Martín Arjovsky, Eugene Brevdo, Konstantin Divilov, Eric Jensen, Mehdi Mirza, Alex Paino, Marjorie Sayer, Ryan Stout und Wentao Wu

- Kapitel 10, Sequenzmodellierung: RNNs und rekursive Netze: Gökçen Eraslan, Steven Hickson, Razvan Pascanu, Lorenzo von Ritter, Rui Rodrigues, Dmitriy Serdyuk, Dongyu Shi und Kaiyu Yang
- Kapitel 11, Praxisorientierte Methodologie: Daniel Beckstein
- Kapitel 12, Anwendungen: George Dahl, Vladimir Nekrasov und Ribana Roscher
- Kapitel 13, Lineare Faktorenmodelle: Jayanth Koushik
- Kapitel 15, Representation Learning: Kunal Ghosh
- Kapitel 16, Strukturierte probabilistische Modelle für Deep Learning: Minh Lê und Anton Varfolom
- Kapitel 18, Die Partitionsfunktion: Sam Bowman
- Kapitel 19, Approximative Inferenz: Yujia Bao
- Kapitel 20, Tiefe generative Modelle: Nicolas Chapados, Daniel Galvez, Wenming Ma, Fady Medhat, Shakir Mohamed und Grégoire Montavon
- Bibliographie: Lukas Michelbacher und Leslie N. Smith

Wir danken auch allen, die uns erlaubt haben, Abbildungen, Grafiken oder Daten aus ihren Werken zu verwenden. Wir haben diese Wiedergabe in den Bildunterschriften angemerkt.

Wir danken Lu Wang dafür, dass er pdf2htmlEX programmiert hat, mit dem wir die Webversion des Buchs erstellt haben, sowie für sein Angebot, uns bei der Verbesserung der Qualität des HTML-Codes zu unterstützen.

Wir danken Ians Ehefrau Daniela Flori Goodfellow für ihre Geduld, die sie Ian während der Abfassung des Buchs entgegengebracht hat. Und natürlich für ihre Hilfe beim Korrektorat.

Wir möchten dem Team hinter Google Brain dafür danken, dass es eine geeignete Umgebung geschaffen hat, in der wir so viel Zeit für die Arbeit an diesem Buch verbringen und gleichzeitig Feedback und Hilfe von den Kollegen annehmen konnten. Ganz besonders möchten wir Ians ehemaligem Vorgesetzten Greg Corrado und seinem derzeitigen Vorgesetzten Samy Bengio für ihre Unterstützung bei diesem Projekt danken. Und schließlich gebührt unser Dank Geoffrey Hinton, der uns bei Schwierigkeiten stets ermuntert hat, weiterzumachen.

Über die Fachkorrektoren zur deutschen Ausgabe

Fachkorrektor für den Bereich Deep Learning

Alexander Bresk ist Unternehmer, Data Engineer, Machine Teacher, Autor und Basketball-Nerd. Er arbeitet als Senior Data Engineer bei der LOVOO GmbH. Dort beschäftigt er sich mit den Themen Recommendation und Online Segmentation. Seinen Master of Science erwarb Alexander an der HTW Dresden, wo er Angewandte Informationstechnologien studierte. Seit seinem Master im Bereich des Question Answering forscht er aktiv an Verfahren zur Verarbeitung natürlicher Sprache (NLP), wobei er unter anderem Deep Learning Frameworks einsetzt. Außerdem organisiert er zusammen mit Kollegen und Freunden Meetups, Konferenzen und andere Veranstaltungen in den Bereichen Tech, Machine Learning und Startups. Mit seiner Firma Machine Rockstars berät er Unternehmen beim Einstieg in das Thema Digitalisierung sowie bei der Einführung von Machine Learning.

Alexander Bresks Website: `http://alexander.bre.sk`

Fachkorrektor für den Bereich Mathematik

Stefan Niedergriese hat Mathematik studiert und arbeitet bei einem Versicherungsunternehmen in der mathematischen Abteilung.

Notation

Dieser Abschnitt dient als Referenz. Er listet die im gesamten Buch verwendete Notation auf. Falls Ihnen einzelne der mathematischen Konzepte unbekannt sind, sollten Sie die Kapitel 2 bis 4 lesen – dort gehen wir auf die meisten davon näher ein.

Zahlen und Vektoren

a	Ein Skalar (ganzzahlig oder reell)
$\boldsymbol{a}$	Ein Vektor
$\boldsymbol{A}$	Eine Matrix
A	Ein Tensor
$\boldsymbol{I}_n$	Einheitsmatrix mit n Zeilen und n Spalten
$\boldsymbol{I}$	Einheitsmatrix mit aus dem Kontext ersichtlicher Dimensionalität
$\boldsymbol{e}^{(i)}$	Standardbasisvektor $[0, \dots, 0, 1, 0, \dots, 0]$ mit einer 1 an der Stelle i
$\text{diag}(\boldsymbol{a})$	Eine quadratische Diagonalmatrix mit Diagonaleinträgen aus $\boldsymbol{a}$
a	Eine skalare Zufallsvariable
$\mathbf{a}$	Eine vektorwertige Zufallsvariable
$\mathbf{A}$	Eine matrixwertige Zufallsvariable

Mengen und Graphen

$\mathbb{A}$	Eine Menge
$\mathbb{R}$	Die Menge der reellen Zahlen

$\{0, 1\}$	Die Menge, die 0 und 1 enthält
$\{0, 1, \ldots, n\}$	Die Menge aller ganzen Zahlen zwischen 0 und n
$[a, b]$	Das reellwertige Intervall der Mengen a und b
$(a, b]$	Das reellwertige Intervall ohne a, jedoch mit b
$\mathbb{A} \backslash \mathbb{B}$	Mengendifferenz (Komplement), also die Menge der Elemente aus $\mathbb{A}$, die nicht in $\mathbb{B}$ enthalten sind
$\mathcal{G}$	Ein Graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	Die Eltern von x_i in $\mathcal{G}$

Indizes

a_i	Element i des Vektors $\boldsymbol{a}$, die Indexmenge beginnt mit 1
a_{-i}	Alle Elemente des Vektors $\boldsymbol{a}$ bis auf das Element i
$A_{i,j}$	Element i, j der Matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Zeile i der Matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Spalte i der Matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) eines 3-D-Tensors A
$\mathsf{A}_{:,:,i}$	2-D-Schnitt eines 3-D-Tensors
a_i	Element i des Zufallsvektors $\mathbf{a}$

Lineare Algebra

$\boldsymbol{A}^\top$	Transponierte der Matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	Moore-Penrose-Pseudoinverse von $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Elementweises Produkt (Hadamard-Produkt) aus $\boldsymbol{A}$ und $\boldsymbol{B}$
$\det(\boldsymbol{A})$	Determinante von $\boldsymbol{A}$

Analysis

$\frac{dy}{dx}$	Ableitung von y bezüglich x
$\frac{\partial y}{\partial x}$	Partielle Ableitung von y bezüglich x

$\nabla_{\boldsymbol{x}} y$	Gradient von y bezüglich $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrixableitungen von y bezüglich $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor mit Ableitungen von y bezüglich $\mathbf{X}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobi-Matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ von $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x})$ oder $\boldsymbol{H}(f)(\boldsymbol{x})$	Hesse-Matrix von f im Eingabepunkt $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Bestimmtes Integral über gesamten Definitionsbereich $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Bestimmtes Integral bezüglich $\boldsymbol{x}$ über Menge $\mathbb{S}$

Wahrscheinlichkeits- und Informationstheorie

$\mathrm{a} \perp \mathrm{b}$	Die Zufallsvariablen a und b sind unabhängig
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	Sie sind bedingt unabhängig, wenn c zutrifft
$P(\mathrm{a})$	Eine Wahrscheinlichkeitsverteilung über eine diskrete Variable
$p(\mathrm{a})$	Eine Wahrscheinlichkeitsverteilung über eine stetige Variable oder eine Variable, deren Typ nicht angegeben wurde
$\mathrm{a} \sim P$	Zufallsvariable a mit der Verteilung P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ oder $\mathbb{E} f(x)$	Erwartung für $f(x)$ bezüglich $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Varianz von $f(x)$ unter $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Kovarianz von $f(x)$ und $g(x)$ unter $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon-Entropie der Zufallsvariablen x
$D_{\mathrm{KL}}(P \| Q)$	Kullback-Leibler-Divergenz von P und Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Normalverteilung über $\boldsymbol{x}$ mit Mittel $\boldsymbol{\mu}$ und Kovarianz $\boldsymbol{\Sigma}$

Funktionen

$f : \mathbb{A} \to \mathbb{B}$	Die Funktion f des Definitionsbereichs $\mathbb{A}$ mit dem Wertebereich $\mathbb{B}$
$f \circ g$	Komposition der Funktionen f und g

$f(\boldsymbol{x};\boldsymbol{\theta})$	Eine Funktion $\boldsymbol{x}$, parametrisiert durch $\boldsymbol{\theta}$. (Manchmal schreiben wir auch $f(\boldsymbol{x})$ und lassen das Argument $\boldsymbol{\theta}$ weg, um die Notation zu vereinfachen.)
$\log x$	Natürlicher Logarithmus von x
$\sigma(x)$	Logistische Sigmoidfunktion, $\frac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\|\|\boldsymbol{x}\|\|_p$	L^p-Norm von $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2-Norm von $\boldsymbol{x}$
x^+	Positiver Teil von x, d.h. $\max(0, x)$
$\mathbf{1}_{\text{Bedingung}}$	ist 1, wenn die Bedingung zutrifft, anderenfalls 0

Manchmal wenden wir eine Funktion f, deren Argument ein Skalar ist, auf einen Vektor, eine Matrix oder einen Tensor an: $f(\boldsymbol{x})$, $f(\boldsymbol{X})$ oder $f(\mathbf{X})$. Dies bezeichnet die elementweise Anwendung von f auf den Vektor, die Matrix oder den Tensor. Beispiel: Ist $\mathbf{C} = \sigma(\mathbf{X})$, dann gilt $C_{i,j,k} = \sigma(X_{i,j,k})$ für alle gültigen Werte von i, j und k.

Datensätze und Verteilungen

p_{data}	Die datengenerierende Verteilung
$\hat{p}_{\text{data}}$	Die empirische Verteilung, die anhand der Trainingsdatenmenge definiert wurde
$\mathbb{X}$	Eine Sammlung mit Trainingsbeispielen
$\boldsymbol{x}^{(i)}$	Das i-te Beispiel (Eingabe) eines Datensatzes
$y^{(i)}$ oder $\boldsymbol{y}^{(i)}$	Der mit $\boldsymbol{x}^{(i)}$ für das überwachte Lernen verknüpfte Zielwert
$\boldsymbol{X}$	Die Matrix $m \times n$ mit dem Eingabebeispiel $\boldsymbol{x}^{(i)}$ in Zeile $\boldsymbol{X}_{i,:}$

1

Einleitung

Schon lange träumen Erfinder von einer Maschine, die selbstständig denken kann. Dieser Wunsch lässt sich bis zu den alten Griechen zurückverfolgen. Die Sagengestalten Pygmalion, Dädalus und Hephaistos sind allesamt berühmte Erfinder. Und Galatea, Talos sowie Pandora lassen sich sämtlich als künstliche Lebensformen betrachten (*Ovid und Martin*, 2004; *Sparkes*, 1996; *Tandy*, 1997).

Als man – über 100 Jahre vor dem Bau des ersten programmierbaren Rechners – über die Möglichkeit solcher Computer nachdachte, fragten sich die Menschen, ob diese Maschinen wohl eines Tages intelligent werden würden (*Lovelace*, 1842). Heute ist die **Künstliche Intelligenz** (KI) eine erfolgreiche Disziplin mit einer Vielzahl praktischer Anwendungen und aktiver Forschungsbereiche. Wir verlassen uns auf »intelligente« Software, um Routineaufgaben zu erledigen, Sprache oder Bilder zu erfassen, medizinische Diagnosen zu stellen und uns bei grundlegender wissenschaftlicher Forschung unterstützen zu lassen.

In der Anfangszeit der Künstlichen Intelligenz hat die Forschung schnell Probleme in Angriff genommen und gelöst, die für Menschen schwierig zu lösen, aber relativ unkompliziert für Computer sind – dabei geht es um Probleme, die durch eine Reihe formaler mathematischer Regeln beschrieben werden können. Die wahre Herausforderung an eine Künstliche Intelligenz besteht jedoch darin, Aufgaben zu erledigen, mit denen Menschen keinerlei Probleme haben, obwohl sie schwer in Worte zu fassen sind – solche Dinge also, die wir intuitiv erledigen und die wir nebenbei tun. Gute Beispiele dafür sind das Erkennen der gesprochenen Sprache oder von Gesichtern in einer Fotografie.

In diesem Buch geht es um die Lösung solch intuitiver Aufgabenstellungen. Die Lösung besteht darin, Computern zu ermöglichen, aus Erfahrungen zu lernen und die Welt so erfassen, als ob sie aus hierarchischen Konzepten besteht, wobei jedes Konzept wiederum definiert ist durch die Beziehung zu einfacheren Konzepten. Indem Computer Wissen aus Erfahrung sammeln, ist es nicht notwendig, mit menschlicher Arbeitskraft formal das Wissen genau zu beschreiben, das der Computer zum Lösen solcher Aufgaben braucht. Die Hierarchie der Konzepte ermöglicht dem Computer das Erlernen komplexer Konzepte, indem er diese aus einfacheren zusammensetzt. Wenn wir diese Modelle grafisch darstellen, blicken wir von oben auf viele Schichten, die sich weit in die Tiefe erstrecken. Deshalb nennen wir diese Methode der Künstlichen Intelligenz **Deep Learning**.

Viele der frühen Erfolge der Künstlichen Intelligenz fanden in relativ sterilen und formalen Umgebungen statt. Das hatte den Vorteil, dass die Computer nur wenig Wissen über die echte Welt benötigten. Ein Beispiel ist IBMs Deep Blue. Dieses Schachprogramm besiegte im Jahr 1997 den Weltmeister Garri Kasparow (*Hsu*, 2002). Beim Schach handelt es sich um ein recht einfaches Universum mit nur 64 Feldern und 32 Figuren, deren Bewegungsmöglichkeiten zudem durch klare Regeln eingeschränkt sind. Die Entwicklung einer erfolgreichen Schachstrategie ist eine echte Meisterleistung. Das liegt aber nicht daran, dass es so kompliziert wäre, einem Computer die Regeln und zulässigen Züge beizubringen. Tatsächlich lassen sich die Regeln in eine sehr kurze Liste formaler Anweisungen fassen, die durch den Programmierer im Vorfeld eingegeben werden.

Die Ironie liegt gerade darin, dass die abstrakten und formalen Aufgaben, die von uns Menschen höchste Geistesleistungen erfordern, für den Computer am unteren Ende der Schwierigkeitsskala stehen. Schon lange können Computer auch die besten Großmeister im Schach schlagen. Doch erst seit kurzer Zeit vermögen sie es, auch bei Objekt- und Spracherkennung mit den Menschen gleichzuziehen. Wir alle greifen im Alltag auf einen gewaltigen Erfahrungsschatz zurück, mit dem wir uns in der Welt zurechtfinden. Ein Großteil unseres Wissens ist subjektiv und intuitiv, lässt sich also nur schwer formal ausdrücken. Damit Computer auf intelligente Weise agieren können, benötigen sie genau dieses Wissen und diese Erfahrungen. Eine der größten Herausforderungen auf dem Gebiet der Künstlichen Intelligenz besteht darin, dem Computer dieses informale und nicht eindeutige Wissen zu vermitteln.

In diversen Projekten der Künstlichen Intelligenz wurde versucht, unser Weltwissen fest in formalen Sprachen zu codieren. Mithilfe logischer Inferenzregeln kann ein Computer auf dieser Basis Schlüsse ziehen. Das ist die sogenannte **wissensbasierte** Herangehensweise an die Künstliche

Intelligenz. Allerdings hat keines dieser Projekte den großen Durchbruch gebracht. Ein berühmtes Beispiel ist Cyc (*Lenat und Guha*, 1989). Cyc ist eine Inferenzmaschine samt Datenbank mit Aussagen in der Sprache CycL. Diese Aussagen werden von menschlichen Supervisoren recht umständlich eingegeben. Für uns Menschen ist es ein großes Problem, formale Regeln aufzustellen, die komplex genug sind, um die Welt korrekt zu beschreiben. Das zeigte sich, als Cyc einen Bericht über Fred, der sich morgens rasierte, gründlich missverstand (*Linde*, 1992). Die Inferenzmaschine kam zu dem Schluss, dass der Bericht inkonsistent sei. Warum? Nun, Cyc wusste, dass Menschen keine elektrischen Bauteile enthalten. Aber da Fred einen elektrischen Rasierapparat in der Hand hielt, ging das System davon aus, dass »FredBeimRasieren« zum Teil aus Elektronik bestand. Daher wollte die Inferenzmaschine wissen, ob Fred beim Rasieren noch ein Mensch ist.

Die Schwierigkeiten, denen sich Systeme gegenüber sehen, die sich auf stark codiertes Wissen stützen, legen nahe, dass KI-Systeme in der Lage sein müssen, ihr eigenes Wissen zu erwerben, indem sie aus Rohdaten Muster extrahieren. Diese Fähigkeit wird als **Machine Learning** bezeichnet. Das Aufkommen des Machine Learnings versetzte Computer in die Lage, Probleme anzugehen, für deren Lösung Wissen über die Realität erforderlich ist, und dafür scheinbar subjektive Entscheidungen zu treffen. Ein einfacher Machine-Learning-Algorithmus namens **logistische Regression** kann entscheiden, ob ein Kaiserschnitt angeraten ist (*Mor-Yosef et al.*, 1990). Ein einfacher Machine-Learning-Algorithmus namens **Naive Bayes** kann erwünschte E-Mails und Spam voneinander trennen.

Die Leistung dieser einfachen Machine-Learning-Algorithmen hängt stark von der **Repräsentation** (also der Darstellung oder Aufbereitung) der Ausgangsdaten ab. Ein Beispiel: Wenn logistische Regression verwendet wird, um Empfehlungen zu einem Kaiserschnitt anzugeben, untersucht nicht das KI-System selbst die Patientin. Stattdessen gibt ein Arzt relevante Daten in das System ein, zum Beispiel ob eine Gebärmutternarbe vorhanden ist. Jede in der Repräsentation der Patientin enthaltene Angabe wird als **Merkmal** bezeichnet. Die logistische Regression »lernt«, wie jedes dieser Merkmale der Patientin mit unterschiedlichen Resultaten korreliert. Sie kann jedoch keinerlei Einfluss darauf nehmen, wie die Merkmale definiert werden. Würde der logistischen Regression eine MRT-Aufnahme der Patientin anstelle des formalisierten Arztberichts zur Verfügung gestellt, könnte das System damit keine nützliche Vorhersage machen. Die einzelnen Pixel des MRTs haben kaum eine Korrelation mit möglichen Komplikationen, die während der Entbindung auftreten können.

Diese Abhängigkeit von der Repräsentation ist ein allgemeines Phänomen, das sich durch die gesamte Informatik und sogar unseren Alltag zieht. In der Informatik erfolgt ein Vorgang wie das Durchsuchen einer Datensammlung exponentiell schneller, wenn die Datensammlung auf intelligente Weise strukturiert und indiziert ist. Menschen können problemlos mit arabischen Ziffern rechnen, doch mit römischen Zahlenzeichen fällt es den meisten sehr viel schwerer. Kein Wunder also, dass die Auswahl der Repräsentation oder Darstellung einen gewaltigen Effekt auf die Leistungsfähigkeit von Machine-Learning-Algorithmen hat. Abbildung 1.1 zeigt ein einfaches Beispiel.

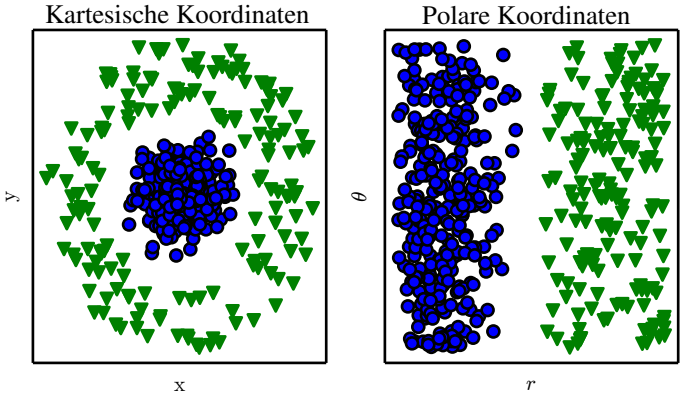

Abbildung 1.1: Beispiel für unterschiedliche Darstellungen: Zwei Datenkategorien sollen durch eine Gerade in einem Punktdiagramm voneinander getrennt werden. Im linken Diagramm werden die Daten mithilfe kartesischer Koordinaten dargestellt. Das macht die Lösung der Aufgabe unmöglich. Im rechten Diagramm sind die Daten mithilfe polarer Koordinaten dargestellt und eine einfache Senkrechte reicht aus, um die Aufgabe zu lösen. (Abbildung gemeinsam mit David Warde-Farley erstellt.)

Viele Aufgaben der Künstlichen Intelligenz können gelöst werden, indem man die richtige Zusammenstellung von Merkmalen konzipiert, die für eine Aufgabe eine Rolle spielen, und diese Merkmale einem einfachen Machine-Learning-Algorithmus zur Verfügung stellt. Um beispielsweise einen Sprecher anhand des Klangs seiner Stimme zu identifizieren, kann sich das Volumen seines Stimmapparats als nützliches Merkmal erweisen. Dieses Merkmal liefert einen guten Hinweis darauf, ob es sich beim Sprecher um einen Mann, eine Frau oder ein Kind handelt.

Für viele Aufgaben besteht jedoch die Schwierigkeit darin zu wissen, welche Merkmale wichtig sind und extrahiert werden müssen. Beispiel: Ein Programm soll Fotografien prüfen und entscheiden, ob darauf ein Pkw zu sehen ist. Wie wir wissen, haben Pkws Räder. Ein Merkmal könnte also das

Vorhandensein von Rädern sein. Aber wie lässt sich ein Rad in Form von Pixelwerten beschreiben? Schwierig, nicht wahr? Ein Rad weist eine einfache geometrische Form auf, aber im Bild gibt es viele andere Elemente, die eine Erkennung erschweren: Schatten, Sonnenblendungen auf Metallteilen des Rads, Kotflügel oder andere Objekte, die Teile des Rads verdecken usw.

Eine Lösung besteht darin, Machine Learning zu verwenden, um nicht nur aus dem Mapping zwischen Repräsentation und Ausgabe, sondern auch aus der Repräsentation (oder Darstellung) selbst Erkenntnisse zu gewinnen. Dieser Ansatz wird als **Representation Learning** bezeichnet. Erlernte Repräsentationen führen häufig zu einer sehr viel besseren Leistung gegenüber individuell angepassten Repräsentationen. Und sie befähigen KI-Systeme dazu, sich bei minimaler menschlicher Intervention schnell an neue Aufgaben anzupassen. Ein Representation-Learning-Algorithmus kann eine gutes Set von Merkmalen für eine einfache Aufgabe in wenigen Minuten erkennen. Bei komplexen Aufgaben kann er einige Stunden oder Monate benötigen. Die händische Konzipierung von Merkmalen für eine komplexe Aufgabe dagegen erfordert jede Menge menschliche Zeit und Mühe – ein Forschungsteam könnte Jahrzehnte damit beschäftigt sein.

Ein Musterbeispiel für einen Representation-Learning-Algorithmus ist der **Autoencoder**. Ein Autoencoder ist eine Kombination aus einer **Encoder**-Funktion, die Eingabedaten in eine andere Repräsentation umwandelt, und einer **Decoder**-Funktion, die diese neue Repräsentation wieder zurück in das Ausgangsformat umwandelt. Autoencoder werden darauf trainiert, so viele Daten wie möglich zu erhalten, wenn eine Eingabe die Encoder- und Decoder-Funktion durchläuft. Gleichzeitig werden sie aber auch darauf trainiert, der neuen Repräsentation einige nützliche Eigenschaften zu verleihen. Je nach Art des Autoencoders liegt das Augenmerk auf unterschiedlichen Eigenschaften.

Beim Entwickeln von Merkmalen oder Algorithmen zum Erlernen von Merkmalen ist es üblicherweise unser Ziel, die **Faktoren der Variation** zu separieren, die die beobachteten Daten erklären. In diesem Kontext bezieht sich das Wort »Faktor« rein auf die unterschiedlichen Einflussquellen. Der Begriff hat nichts mit einer Multiplikation zu tun. Diese Faktoren sind häufig keine direkt beobachteten Größen, sondern entweder unbeobachtete Objekte oder unbeobachtete Kräfte in der physischen Welt, die ihrerseits die einer Beobachtung zugänglichen Größen beeinflussen. Es kann sich auch um Konstrukte des menschlichen Denkens handeln, die nützliche verallgemeinerte Erklärungen oder vermutete Ursachen für die beobachteten Daten liefern. Sie können sich diese Faktoren als Konzepte oder Abstraktionen vorstellen, die dabei helfen, der großen Variabilität der Daten einen Sinn zu

geben. Beim Analysieren einer Sprachaufzeichnung gehören zu den Faktoren der Variation zum Beispiel das Alter und das Geschlecht des Sprechers, der Dialekt und die verwendeten Wörter. Beim Analysieren einer Abbildung eines Fahrzeugs wiederum gehören dessen Position und Farbe, der Betrachtungswinkel und die Helligkeit des Sonnenlichts zu den Faktoren der Variation.

Bei vielen KI-Anwendungen in der Praxis liegt eine große Schwierigkeit darin, dass viele der Faktoren der Variation sich auf jedes einzelne Datenelement auswirken, das wir beobachten können. Die einzelnen Pixel des Fotos eines roten Autos können bei Nacht nahezu schwarz erscheinen. Die Form der Fahrzeugumrisse ist abhängig vom Blickwinkel. Die meisten Anwendungen erfordern es von uns, dass wir die Faktoren der Variation *separieren* und diejenigen verwerfen, die nicht von Belang sind.

Das Extrahieren solch übergeordneter, abstrakter Merkmale aus den Rohdaten kann sich als extrem schwierig erweisen. Viele der Faktoren der Variation wie ein Dialekt können nur bestimmt werden, wenn ein tiefes, nahezu menschliches Verständnis der Daten vorhanden ist. Wenn es genauso schwierig ist, eine Repräsentation zu erzielen wie das ursprüngliche Problem zu lösen, scheint uns das Representation Learning auf den ersten Blick nicht hilfreich zu sein.

Deep Learning löst dieses zentrale Problem beim Representation Learning, indem Repräsentationen eingesetzt werden, die in Form von anderen simpleren Repräsentationen dargestellt werden. Deep Learning befähigt den Computer, komplexe Konzepte aus einfacheren Konzepten zu konstruieren. Abbildung 1.2 zeigt, wie ein Deep-Learning-System das Konzept einer Personenfotografie darstellen kann, indem es einfachere Konzepte kombiniert wie hier Ecken und Umrisse, die wiederum über Kantenverläufe definiert werden.

Ein Musterbeispiel für ein Deep-Learning-Modell ist ein tiefes Feedforward-Netz, auch **mehrschichtiges Perzeptron** (engl. *multilayer perceptron*, MLP) genannt. Ein mehrschichtiges Perzeptron ist eine mathematische Funktion, die eine Menge von Eingabewerten entsprechenden Ausgabewerten zuordnet. Die Funktion wird aus vielen einfacheren Funktionen zusammengesetzt. Dabei können Sie sich vorstellen, dass jede Anwendung einer anderen mathematischen Funktion eine neue Repräsentation der Eingaben liefert.

Das Konzept des Erlernens der richtigen Repräsentation für die Daten stellt eine Perspektive auf das Deep Learning dar. Eine weitere Perspektive rückt die Tiefe in den Mittelpunkt: Sie ermöglicht dem Computer das Erlernen eines mehrstufigen Computerprogramms. Jede Schicht der

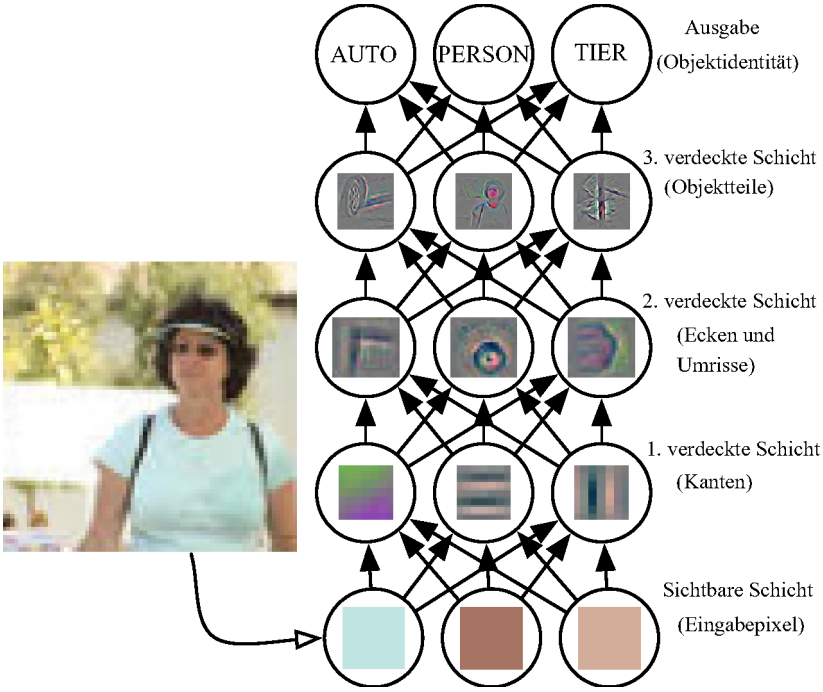

Abbildung 1.2: Darstellung eines Deep-Learning-Modells. Für den Computer ist es nicht einfach, die Bedeutung roher Sensoreingaben zu verstehen. Ein Beispiel ist dieses Foto, das als Ansammlung von Pixelwerten dargestellt wird. Die Funktion zur Zuordnung einer Pixelmenge zu einer Objektidentität ist sehr kompliziert. Das Erlernen oder Bewerten dieser Zuordnung erscheint unmöglich, wenn man das Problem direkt angehen will. Mit Deep Learning wird die Schwierigkeit jedoch vermindert, indem die erwünschte komplizierte Zuordnung in eine Reihe verschachtelter einfacherer Zuordnungen aufgeteilt wird. Jede davon beschreibt eine andere Schicht des Modells. Die Eingabe wird als **sichtbare Schicht** dargestellt. Die Bezeichnung wurde gewählt, weil sie die Variablen enthält, die wir beobachten können. Darauf werden in einer Reihe von **verdeckten Schichten** immer abstraktere Bildmerkmale extrahiert. Diese Schichten werden als »verdeckt« bezeichnet, weil ihre Werte nicht in den Daten enthalten sind, sondern das Modell selbst entscheiden muss, welche Konzepte zum Erklären der Beziehungen in den beobachteten Daten hilfreich sind. Die gezeigten Abbildungen visualisieren die Art von Merkmalen, die durch jede verdeckte Einheit dargestellt werden. Anhand der Pixel können in der ersten Schicht durch Vergleich der Helligkeitswerte benachbarter Pixel die Kanten bestimmt werden. Auf Basis der in der ersten verdeckten Schicht ermittelten Kanten erfolgt in der zweiten verdeckten Schicht die Suche nach Ecken und verlängerten Konturen, die als Aneinanderreihung von Kanten erkannt werden. Nachdem die zweite verdeckte Schicht das Bild anhand von Ecken und Konturen beschrieben hat, sucht die dritte verdeckte Schicht nach kompletten Teilen bestimmter Objekte. Dazu wird auf bestimmte Sammlungen von Konturen und Ecken geprüft. Im Ergebnis steht eine Bildbeschreibung in Form von Objektteilen zur Verfügung, die dann zum Erkennen der Objekte im Foto verwendet werden kann. (Abbildungen mit freundlicher Genehmigung von *Zeiler und Fergus* (2014))

Repräsentation wird dabei als Speicherzustand des Computers nach der parallelen Ausführung einer weiteren Anweisungsgruppe betrachtet. Netze mit einer größeren Tiefe können mehr Anweisungen nacheinander ausführen. Sequenzielle Anweisungen bieten mehr Leistungsfähigkeit, denn spätere Anweisungen können auf die Ergebnisse früherer Anweisungen zurückgreifen. Bei dieser Perspektive auf das Deep Learning codieren nicht alle Informationen in den Aktivierungen einer Schicht notwendigerweise Faktoren der Variation, die die Eingabe erklären. Die Repräsentation speichert außerdem Informationen über den Zustand, mit deren Hilfe ein Programm ausgeführt werden kann, das die Eingabe auswertet. Diese Zustandsinformationen ähneln einem Zähler oder Pointer in einem klassischen Computerprogramm. Sie haben nichts mit dem eigentlichen Inhalt der Eingabe zu tun, sondern helfen dem Modell, die Verarbeitung zu strukturieren.

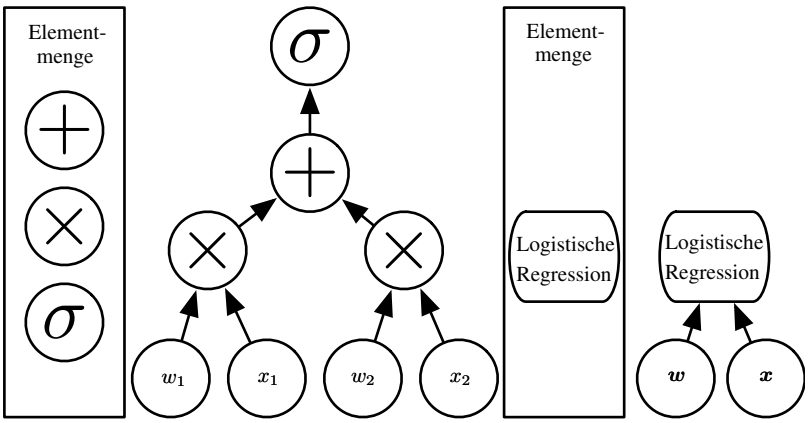

Abbildung 1.3: Abbildung des Berechnungsverlaufs (Graphen) für eine Eingabe, bei dem jeder Knoten auf dem Weg zur Ausgabe eine Berechnung durchführt. Die Tiefe ist der längste Pfad von der Eingabe zur Ausgabe. Sie richtet sich allerdings danach, was als möglicher Berechnungsschritt definiert ist. Die hier dargestellte Berechnung ist die Ausgabe eines logistischen Regressionsmodells $\sigma(\boldsymbol{w}^T\boldsymbol{x})$ mit σ als logistischer Sigmoidfunktion. Bei Verwendung von Addition, Multiplikation und logistischen Sigmoidfunktionen als Elemente unserer Programmiersprache hat dieses Modell eine Tiefe von 3. Gilt die logistische Regression als eigenes Element, hat das Modell die Tiefe 1.

Es gibt zwei wesentliche Möglichkeiten zum Messen der Modelltiefe. Bei der ersten wird die Anzahl sequenzieller Anweisungen bestimmt, die zum Bewerten der Architektur ausgeführt werden müssen. Das entspricht quasi dem längsten Pfad in einem Ablaufplan, der die Berechnung der einzelnen Ausgaben des Modells anhand der Eingaben beschreibt. So wie zwei äquivalente Computerprogramme abhängig von der Programmiersprache