iX KOMPAKT

Frühjahr 2018

Ein Sonderheft des Magazins für professionelle Informationstechnik

Machine Learning

Tutorials:
Maschinelles Lernen mit TensorFlow und Python
Automatisierte Textanalyse

Modernes JavaScript

Tutorial: Webapps mit Angular
APIs mit GraphQL
Microservices mit Node.js

Python

Tutorial: Parallelprogrammierung
Data Science mit Luigi

C++17 und C++20

Effizientes Multithreading
Besserer Code durch Concepts

Scala Days

Berlin – May 14th-17th, 2018

Scala Days Europe

14-15 May: Workshops **15-17 May: Conference Sessions**

Scala Days Europe 2018 is a fantastic and unique occasion to meet with Scala-using professionals and companies to exchange ideas and form business relations, as well as to discover the latest practical and theoretical developments concerning the Scala language. Scala Days Europe will bring together developers from all around the world to share their experiences. They talk about new ideas and creating applications with Scala, Akka, Play, Typelevel projects, and other community libraries and tools. Scala Days provides a unique opportunity for Scala users to interact with the key contributors to the language, Scala community leaders, and everyone using Scala.

Topics

Foundations
- Functional programming fundamentals
- Types
- Concurrency and parallelism

Experience reports
- Industrial adoption
- Open source issues
- Mentoring
- Teaching

Libraries and applications
- Distributed systems
- Machine learning
- Big data
- Databases
- Others

Workflow
- Programming methodologies
- Deployment
- DevOps
- Security issues

Tooling
- Compilers and virtual machines
- IDEs
- Testing frameworks
- Build tools
- Next generation tooling

>>> Opening Keynote by Martin Odersky, designer of Scala!

Platinum-Sponsor

LUNATECH

Gold-Sponsors

47 Audi BITROCK Evolution Gaming here JetBrains RISK IDENT SAP SIGNIFY TECHNOLOGY zalando

Silver-Sponsor

SOFTWAREMILL TRIVENTO

Lightbend heise Developer dpunkt.verlag

www.scaladays.org

Von lernenden Maschinen und neuen Standards

Programmierer stehen vor immer neuen Herausforderungen: Neue Techniken, Paradigmen, Standards und nicht zuletzt Moden erzwingen ständiges Dazulernen, will man nicht den Anschluss verlieren. Mit Machine Learning, JavaScript, Python und den Standards C++17 und C++20 greift dieses Heft vier aktuelle Trends in der Softwareentwicklung auf.

Künstliche Intelligenz und maschinelles Lernen kommen in immer mehr Bereichen zum Einsatz. Anwender erwarten mittlerweile, dass „intelligente" Software ihre Mail vorsortiert, neue TV-Serien auf Grundlage des bisher Gesehenen vorschlägt oder den Newsstream auf ihre Interessen zuschneidet. Den Datenmassen des Big-Data-Zeitalters ist ohne maschinelle Intelligenz ohnehin kaum mehr beizukommen. Daher zeigen wir in zwei umfangreichen Tutorials, wie man in Python mithilfe von TensorFlow neuronale Netze programmiert und einsetzt und wie man – ebenfalls mit Python und Open-Source-Bibliotheken – große Textmengen mit Methoden des maschinellen Lernens analysiert.

Überhaupt, Python: Die Skriptsprache hat sich zum Schweizer Taschenmesser für Programmierer gemausert. So lassen sich mit dem Python-Framework Luigi komplexe Datenanalysen mit wenigen Zeilen Python-Code organisieren und überwachen. Ein Tutorial erklärt, wie Multithreading, asynchrone Programmausführung und das Verteilen über mehrere Rechner Python-Programme beschleunigen.

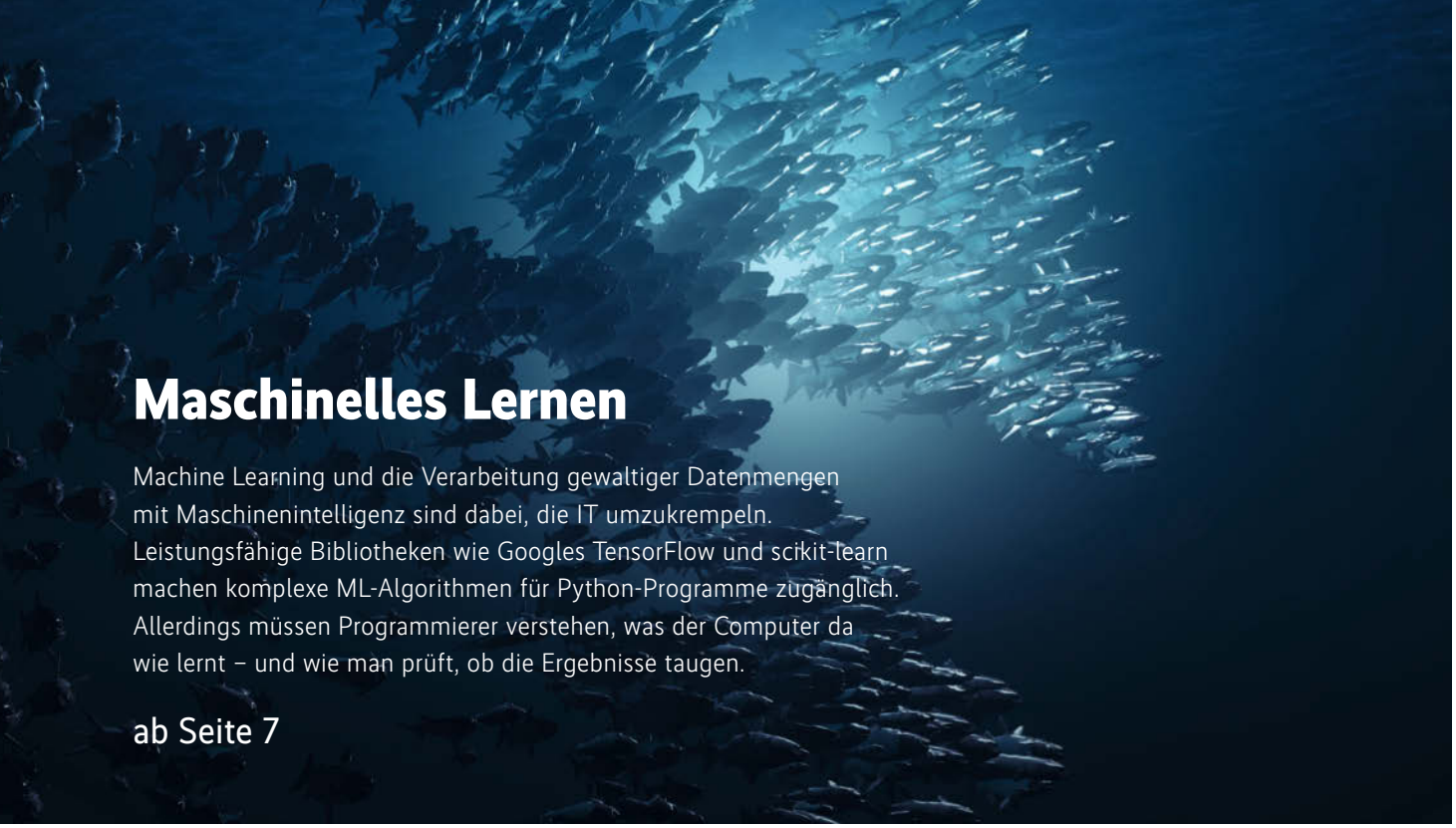
Der Webbrowser entwickelt sich immer mehr zur Plattform der Wahl für Anwendungen aller Art – entsprechend gehört JavaScript derzeit zu den populärsten Sprachen. Unser Angular-Tutorial zeigt, wie man eine komplexe Webapp vom Kaliber Wunderlist mit Googles populärem Webframework erstellt.

Facebook hat GraphQL als effiziente Alternative zu REST-APIs und JSON entwickelt; mittlerweile bieten immer mehr Webdienste GraphQL-Schnittstellen. Dass JavaScript auch auf dem Server seine Berechtigung hat, belegt der Artikel zur Implementierung von Microservices mit Node.js.

Der im Dezember verabschiedete Standard C++17 hat dem Programmiersprachen-Klassiker eine Reihe von Neuerungen gebracht, die jeder C++-Programmierer kennen sollte. Vor allem die Multithreading-Fähigkeiten werden durch C++17 und noch mehr durch den kommenden Standard C++20 stark erweitert. Concepts, ebenfalls für C++20 fest eingeplant, gehen in ihren Möglichkeiten weit über Templates hinaus und dürften die Art und Weise revolutionieren, wie Entwickler generischen Code schreiben. Der Abschnitt zu C++ im Heft erklärt, was Entwickler dazu wissen müssen.

Wir hoffen, dass Ihnen unser *iX* kompakt viele Anregungen und interessante Informationen liefert. Sämtliche Artikel stammen aus *iX*, dem Magazin für professionelle Informationstechnik von Heise Medien.

OLIVER DIEDRICH



Maschinelles Lernen

Machine Learning und die Verarbeitung gewaltiger Datenmengen mit Maschinenintelligenz sind dabei, die IT umzukrempeln. Leistungsfähige Bibliotheken wie Googles TensorFlow und scikit-learn machen komplexe ML-Algorithmen für Python-Programme zugänglich. Allerdings müssen Programmierer verstehen, was der Computer da wie lernt – und wie man prüft, ob die Ergebnisse taugen.

ab Seite 7

Maschinelles Lernen

Python

JavaScript

C++

Rubriken

Python

Ob bei der Datenauswertung, als Werkzeug zur Entwicklung von Prototypen oder einfach für das schnelle Skript zwischendurch: Python ist in der IT fast allgegenwärtig. Obwohl die Sprache schnell und einfach zu erlernen ist, lohnt es sich, tiefer einzusteigen und sich mit Themen wie Multithreading, Asynchronität und Jupyter/IPython zu beschäftigen.

ab Seite 53

C++17 und C++20

Die Standards C++17 und C++20 bringen dem Programmiersprachen-Klassiker etliche Neuerungen. C++-Experte Rainer Grimm beschreibt, was sich mit dem aktuellen Standard C++17 geändert hat und wie C++17 und C++20 Multithreading besser handhabbar machen. Besonderes Augenmerk liegt auf den neuen Concepts, die mit C++20 die bestehenden Schwächen der Templates ausbügeln sollen.

ab Seite 125

Modernes JavaScript

Sowohl auf GitHub wie auch auf StackOverflow führt JavaScript die Liste der meistgenutzten Programmiersprachen an. Grund genug, sich mal genauer anzusehen, was der aktuelle Stand der Dinge bei JavaScript ist, wie man heutzutage komplexe Webanwendungen programmiert, was GraphQL besser macht als REST-APIs und wie man mit Node.js moderne Microservice-Architekturen umsetzt.

ab Seite 79

Hand aufs Herz –

weißt Du wie wir funktionieren?

ÜBER 500 FASZINIERENDE BILDER

Jetzt erhältlich im Handel oder unter

shop.heise.de/wissen-mensch18

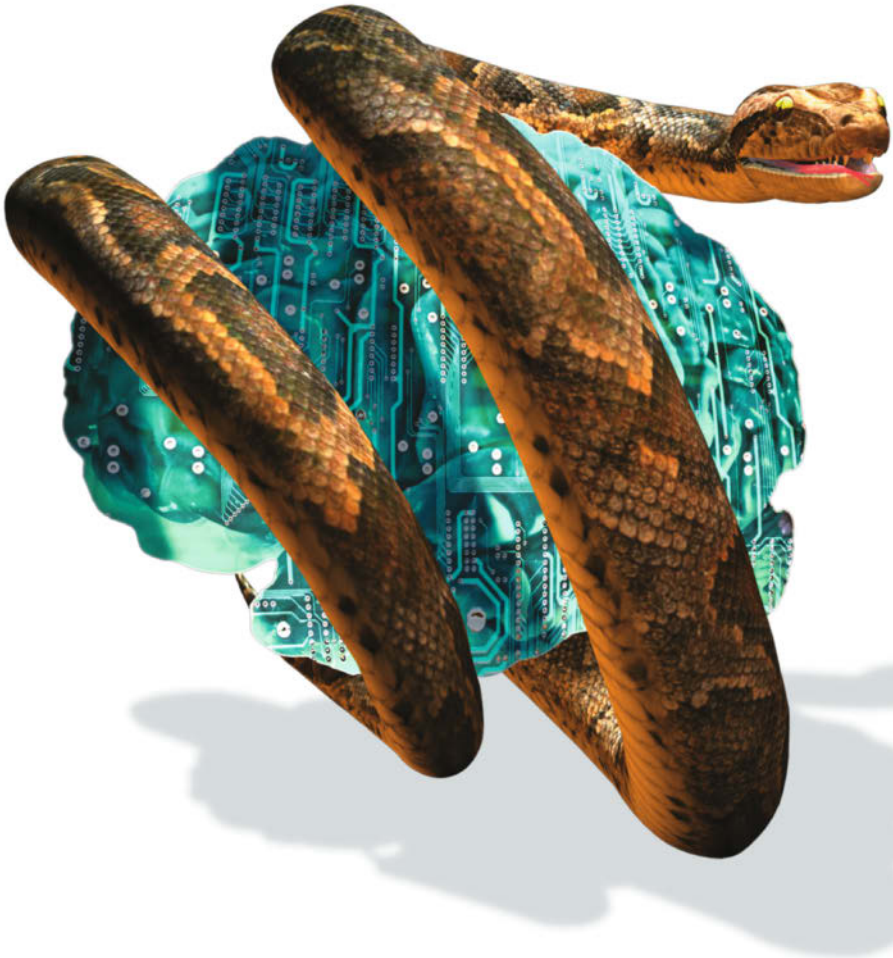
eMedia GmbH

Maschinelles Lernen: Die Intelligenz des Schwarms

Science-Fiction-Autoren wissen es schon lange: Maschinen können lernen. Zwar bilden heutige Algorithmen noch nicht komplett die menschliche Intelligenz nach oder übertreffen sie gar. Aber verbesserte Methoden des Deep Learning, spezialisierte Hardware oder Services im Netz führen schon zu überraschenden Ergebnissen. In Kombination mit Open-Source-Bibliotheken und frei verfügbaren Netzen zeigen Python-Programme, was in den Bereichen Bilderkennung und Textanalyse bereits möglich ist.

Python-Tutorial, Teil 1:
Maschinelles Lernen mit TensorFlow

Wie Programme lernen



Gerhard Völkl

Meldungen über wesentliche Verbesserungen in der automatischen Spracherkennung, beim Übersetzen von Texten oder dem Erkennen von Bildinhalten lassen aufhorchen. Das Buzzword in diesem Zusammenhang heißt „maschinelles Lernen“. Was steckt hinter dieser Technik und wie kann man sie mit Python und Googles TensorFlow-Bibliothek nutzen?

Ob Facebook automatisch Personen auf hochgeladenen Fotos identifiziert, Salesforce Einstein selbstständig neue Zusammenhänge in Kundendaten entdeckt oder Computer Patientendaten analysieren und Diagnosen stellen: Konzepte wie Deep Learning, neuronale Netze und maschinelles Lernen scheinen derzeit hinter vielen Fortschritten in der IT zu stehen.

Die einen sehen maschinelles Lernen als Allheilmittel, mit dem sich Programme alles selbstständig aneignen können – vom Übersetzen in beliebige Sprachen bis hin zum Autofahren –, ohne jemals einen Programmierer gesehen zu haben. Andere diskutieren über die negativen Auswirkungen der maschinellen Allmacht bis hin zum direkten Anschluss ans menschliche Gehirn. Wieder andere interessiert nicht nur, was maschinelles Lernen bewirken kann, sondern vor allem auch, wie es funktioniert.

Daher liegt der Schwerpunkt dieser Artikelreihe über maschinelles Lernen mit Python und TensorFlow darauf, fachliche Informationen über diese Technik zu liefern und Anregungen zu geben, sie in eigenen Python-Programmen auszuprobieren. Ein erster Schritt, um besser einschätzen zu können, was mit maschinellem Lernen möglich ist.

Da dieses Thema ein weites Feld ist, konzentrieren sich die Artikel auf die grundsätzliche Vorgehensweise und neuere Verfahren aus dem Bereich der neuronalen Netze, die mitverantwortlich für den Hype sind. Als Programmbibliothek kommt TensorFlow von Google zum Einsatz, da sie diverse Werkzeuge mitbringt und im Netz viele Informationen dazu zu finden sind. Ein Jupyter Notebook, das Interessierte über den *iX*-Listing-Service herunterladen und nachvollziehen können, enthält alle im Tutorial vorgestellten Beispiele (hierzu und zu weiteren Quellen im Web siehe „Alle Links“ am Ende des Artikels).

Auf den Algorithmus kommt es an

Im Zentrum der Programmierung steht immer das Problem, das die neue Software lösen soll – beispielsweise die Anfrage aus der Buchhaltung, alle Einkäufe im Bereich Bürobedarf zusammenzurechnen. Das ist eine relativ einfache Sache: Die Eingaben sind irgendwelche Zahlen aus den Rechnungen, herauskommen soll die Summe daraus. Wie man Zahlen zusammenzählt, ist ein einfacher, bereits in der Grundschule gelernter Algorithmus.

Nächstes Beispiel: Eine Mitarbeiterin der Gebäudeverwaltung eines großen Konzerns möchte ein Programm, das ihr ermittelt, was der Verkauf jeder Firmenimmobilie einbringen könnte. Als Eingabegrößen dienen die Wohnfläche in Quadratmetern und die Größe des bebauten Grundstücks, das gewünschte Ergebnis ist der Verkaufswert. Dieses Mal ist es nicht so einfach wie im ersten Fall. Es geht zwar wieder nur um Zahlen. Aber mit welchem Algorithmus man von der Eingabe zum gewünschten Ergebnis kommt, ist nicht bekannt.

```
def was_kostet_ein_Haus(flaeche,
                        grundstuecksgroesse):
    # ??
    return ergebnis
```

Optimale Formel finden

Aus den in der Vergangenheit durchgeführten Verkäufen ist bekannt, welche Häuser wie viel eingebracht haben. Über eine Formel kann man den Verkaufspreis in Abhängigkeit von Wohnfläche und Grundstücksgröße ausdrücken. Beim maschinellen Lernen findet der Computer eine optimale Formel, indem eine Software aus den vorgegebenen Daten lernt, welche Ausgabe bei welchen Eingabedaten erwartet wird. Damit justiert sie ihre internen Einstellungen, sodass sie sich den gewünschten Ergebnissen immer mehr annähert. Nach diesem Lernvorgang sollte die Software in der Lage sein, für noch nie gesehene Eingaben ein brauchbares Ergebnis auszuwerfen (Abbildung 1).

Die Computerwissenschaft hat über die Jahre unzählige Verfahren im Bereich maschinelles Lernen entwickelt. Zu den bekannteren zählen Entscheidungsbäume, Nearest-Neighbor-Methode, lineare Regression und neuronale Netze. Da sich Python zu der am häufigsten verwendeten Programmiersprache im maschinellen Lernen gemausert hat, gibt es dafür viele unterschiedliche Module wie *scikit-learn* oder *Theano*.

Dieses Tutorial verwendet Googles TensorFlow-Bibliothek, da sie eine umfangreiche, skalierbare Umgebung mitbringt und in vielen Projekten Anwendung findet – auch in anderen Programmiersprachen. Eins der Tools ist das nützliche Visualisierungswerkzeug TensorBoard, mit dem man einen Blick in die Verarbeitung werfen kann.

Darüber hinaus gibt es beispielsweise TensorFlow Serving, eine Plattformlösung mit einem Server, auf dem fertige Anwendungen laufen und durch Client-Programme genutzt werden können. Zurzeit gibt es TensorFlow für Arbeitsplatzrechner unter Linux, macOS und Windows sowie Varianten für mobile Geräte mit Android oder iOS. Die TensorFlow-API lässt sich grob in drei Ebenen einteilen (siehe Abbildung 2):
– Core bietet Zugriff auf alle einzelnen Bestandteile, aus denen TensorFlow besteht.
– Layer sind Module, die aus den einzelnen Bestandteilen entstanden sind.
– Estimators sind fertige Komponenten, die man einsetzen kann.

TensorFlow für unterschiedliche Hardware

Dass Google TensorFlow direkt auf GitHub für alle frei unter der Apache-2.0-Lizenz veröffentlicht hat, ist ein kleines Novum. Entweder man lädt sich von dort die notwendigen Dateien oder verwendet, wie in Python üblich, den Python Packet Manager *pip*:

```
pip install tensorflow
```

Dieses Installationspaket enthält eine Version von TensorFlow, die mit der CPU arbeitet und keinerlei sonstige Hardware benötigt. Wer mehr Geschwindigkeit braucht und eine entsprechende Grafikkarte besitzt, kann sich eine spezielle TensorFlow-Version laden:

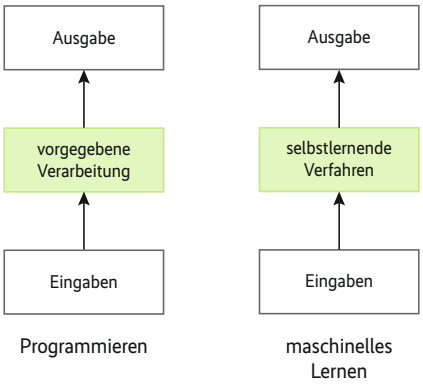
```
pip install tensorflow-gpu
```

Wichtig ist, dass die vorhandene Grafikkarte NVIDIAs CUDA (Compute Unified Device Architecture) unterstützt. Über diese Schnittstelle können Anwendungen die Grafikkarten (GPU) für alle möglichen Berechnungen nutzen, die nichts mit der Erstellung von Grafik zu tun haben. Die benötigten Rechenfähigkeiten sind an der Version der NVIDIA Compute Capability zu erkennen. Die Version 3.0 sollte mindestens vorhanden sein.

Jedes Programm, das mit TensorFlow arbeitet, besteht aus mindestens zwei Abschnitten:
– Aufbau eines Berechnungsgraphen (Computation Graph);
– Ausführen des Berechnungsgraphen.

TensorFlow trennt strikt die Definition einer Berechnung von deren Ausführung. Das hat unter anderem den Vorteil, dass man die Definition auf einem Rechner – etwa einem einfachen Arbeitsplatz-PC – erstellen kann und die Ausführung dann auf einem oder mehreren großen Serversystemen läuft. Darüber hinaus lässt sich ein Berechnungsgraph automatisiert in mehrere Teile zerlegen, wobei die einzelnen Teile wiederum auf unterschiedlichen Grafikkarten laufen können.

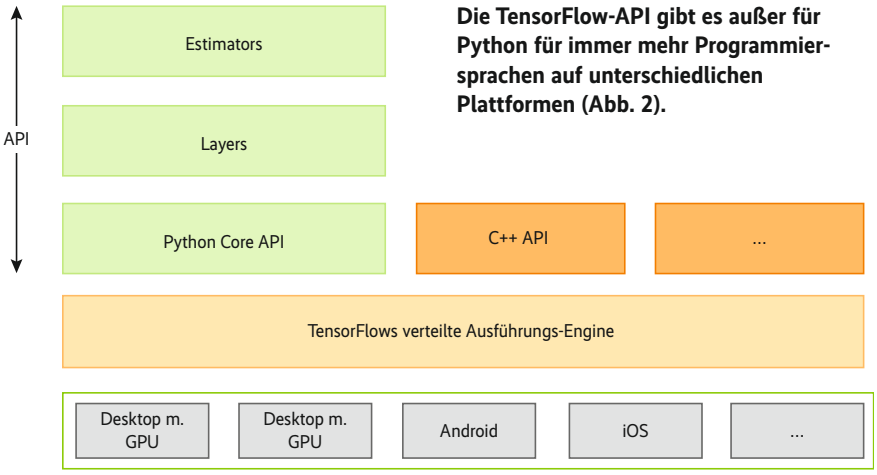
Ein Berechnungsgraph ist die abstrakte Darstellung eines Algorithmus. Er besteht aus Knoten, die für Eingabewerte oder



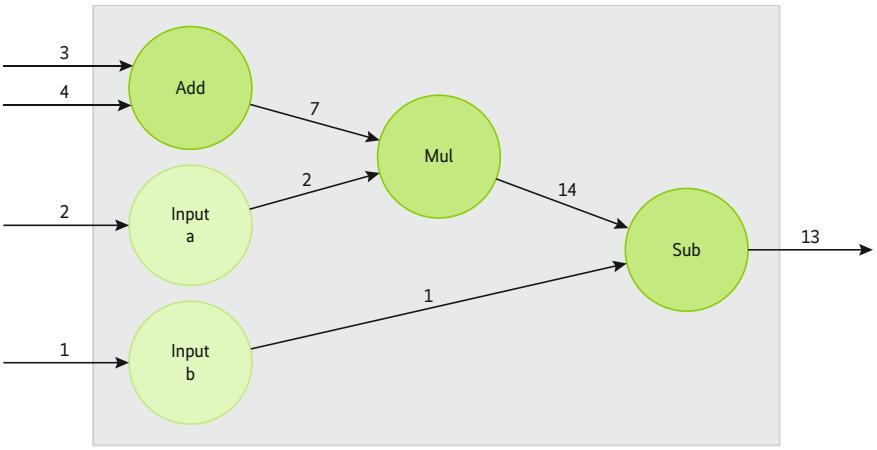
Selbstlernende Algorithmen haben in den Bereichen Erfolg, in denen kein einfaches Rechenverfahren zum gewünschten Ziel führt (Abb. 1).

iX-TRACT

- Googles Open-Source-API TensorFlow gehört zu den wichtigsten Bibliotheken für maschinelles Lernen. Anwendungen lassen sich unter anderem in Python und C schreiben.
- Jedes Programm, das mit TensorFlow arbeitet, besteht aus mindestens zwei Abschnitten: dem Aufbau eines Berechnungsgraphen und dessen Ausführung. Dabei repräsentieren die Knoten des Graphen mathematische Operationen und seine Kanten die dazwischen fließenden (Flow) Daten-Arrays (Tensoren).
- Wesentlich für das selbstständige Lernen von Programmen sind große Datenmengen, die als Erfahrungswerte dienen und auf deren Grundlage sich Zusammenhänge zwischen Eingabegrößen ermitteln lassen.
- Für ein möglichst gutes Resultat nutzt TensorFlow Optimierungsalgorithmen wie das Gradientenverfahren. In Schleifen führen selbstlernende Programme Trainingsdurchläufe aus, die die besten Ergebnisse für die Eingabewerte ermitteln sollen.



Die TensorFlow-API gibt es außer für Python für immer mehr Programmiersprachen auf unterschiedlichen Plattformen (Abb. 2).



Berechnungen oder ganze Programme sind als Graphen darzustellen (Abb. 3).

Verarbeitungsschritte stehen. Die Verbindungen zwischen den Knoten, oft als Pfeile dargestellt, zeigen an, wie die Daten fließen (float). Wie das konkret aussieht, zeigt das Beispiel einer relativ einfachen Berechnung:

```
(3 + 4) * a - b
```

Der Berechnungsgraph dafür könnte aussehen wie in Abbildung 3.

Bevor es mit TensorFlow losgeht, muss man das Modul in Python importieren:

```
import tensorflow as tf
```

Die Beispielberechnung beginnt mit dem Ausdruck „3 + 4“, zwei zu addierenden Konstanten. Das Erzeugen einer Konstante in TensorFlow übernimmt die Methode *tf.constant*:

```
node_const_1 = tf.constant(3.0, tf.float32, 7
                           name = 'const_1')
```

Neben dem eigentlichen Wert, hier *3.0*, enthält die Konstantendefinition den genauen Datentyp *Float32* sowie ihren Namen innerhalb von TensorFlow. Bei Bezeichnungen, egal ob für Konstanten oder andere Programmbestandteile, ist immer deren Name in Python von dem in TensorFlow zu unterscheiden. Um sich das Leben einfacher zu machen, ist es sinnvoll, hier inhaltlich ähnliche Namen zu wählen. Die Bezeichnungen in TensorFlow rücken in den Fokus, wenn man etwa die Abarbeitung der Berechnungen nachvollziehen will:

```
node_const_2 = tf.constant(4.0, tf.float32, 7
                           name = 'const_2')
node_add = tf.add(node_const_1, node_const_2)
```

TensorFlow-Operatoren (Auswahl)

mathematische Operatoren für einzelne Elemente	*Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal*
Array-Operatoren	*Concat, Slice, Split, Constant, Rank, Shape, Shuffle*
Matrix-Operatoren	*MatMul, MatrixInverse, MatrixDeterminant*
neuronale Netzwerke	*SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool*
Queues und Synchronisation	*Enqueue, Dequeue, MutexAcquire, MutexRelease*
Ablaufsteuerung	*Merge, Switch, Enter, Leave, NextIteration*

Der mathematische Operator *tf.add* addiert die Werte seiner Eingabeknoten. TensorFlow besitzt neben den verschiedenen mathematischen Operatoren viele, die interessant sind für die Berechnungen im Bereich maschinelles Lernen (siehe Tabelle „Operatoren“). Damit aber TensorFlow tatsächlich eine Berechnung ausführt, muss man ein *tf.Session*-Objekt anlegen:

```
session = tf.Session()
```

Hier ist die grundsätzliche Vorgehensweise ähnlich wie beim Lesen aus einer Datei: Man erzeugt als Erstes einen Zugriff auf etwas – in dem einen Fall auf eine Datei, im anderen auf ein *session*-Objekt – führt irgendwelche Verarbeitungsschritte durch und schließt das Objekt wieder. Dafür, dass das Schließen nicht in Vergessenheit gerät, erweist sich der Python-Befehl *with* als praktisch, der das automatisch übernimmt:

```
with tf.Session() as session:
    print(session.run(node_add))
```

Die Methode *session.run* führt den Berechnungsgraphen in TensorFlow aus und erhält als Rückgabewert das Ergebnis der Addition. Am Ende des *with*-Befehls läuft automatisch die Methode *session.close* und beendet die Session.

Nach dem Konstanten-Ausdruck *(3 + 4)* folgt in der Beispielberechnung der variable Wert *a*, mit dem multipliziert wird. Hier unterscheidet TensorFlow zwei Vorgehensweisen: Soll *a* ein Parameter sein, den Python am Anfang der Berechnung mit übergibt, oder handelt es sich um eine Variable, deren Wert sich während der Verarbeitung ändert? Im ersten Fall erzeugt die Methode *tf.placeholder* einen sogenannten Placeholder:

```
a = tf.placeholder(tf.float32, name = 'a')
```

Für die darauffolgende Multiplikation gibt es die Methode *tf.multiply*:

```
node_mul = tf.multiply(node_add, a)
```

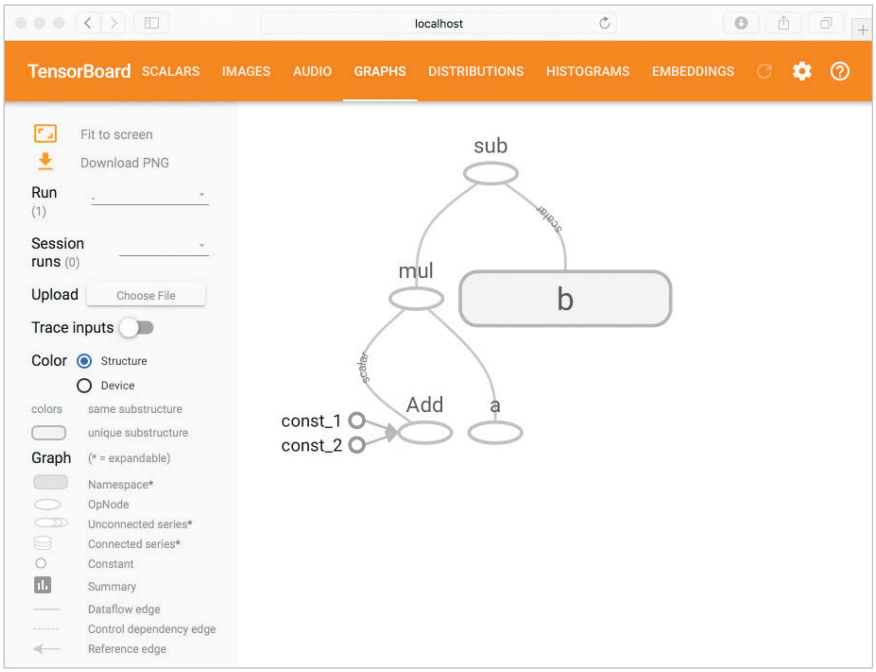
Alternativ kann man für eine einfachere Schreibweise den gewohnten Python-Operator * verwenden, da TensorFlow diesen mit seiner Methode *tf.multiply* überladen hat:

```
node_mul = node_add * a
```

Das Gleiche wäre oben bei *tf.add* mit dem Python-Operator + möglich.

Ein Placeholder in TensorFlow erhält seinen Wert vor der Ausführung der Berechnung durch ein Dictionary, übergeben in der Methode *session.run*:

```
session.run(node_add, {a: 2.0})
```



Das Werkzeug TensorBoard gewährt einen Blick auf die Abläufe in TensorFlow (Abb. 4).

Genau genommen ist ein Placeholder aus Sicht von TensorFlow eigentlich eine Konstante, da der Wert vom Beginn der Berechnung bis zum Ende fest ist. Placeholder sind dann interessant, wenn Berechnungen in Python stattfinden, deren Ergebnis an TensorFlow übergeben wird.

Richtige Variablen, die ihren Wert ändern können, erzeugt die Methode *tf.Variable*:

```
b = tf.Variable(1.0, tf.float32, name ='b')
```

Mit dem ersten Argument, hier *1.0,* initialisiert TensorFlow die Variable. Es folgen Datentyp und Name. Vor der eigentlichen Berechnung erfolgt das Initialisieren der Variablen:

```
init = tf.global_variables_initializer()
session.run(init)
```

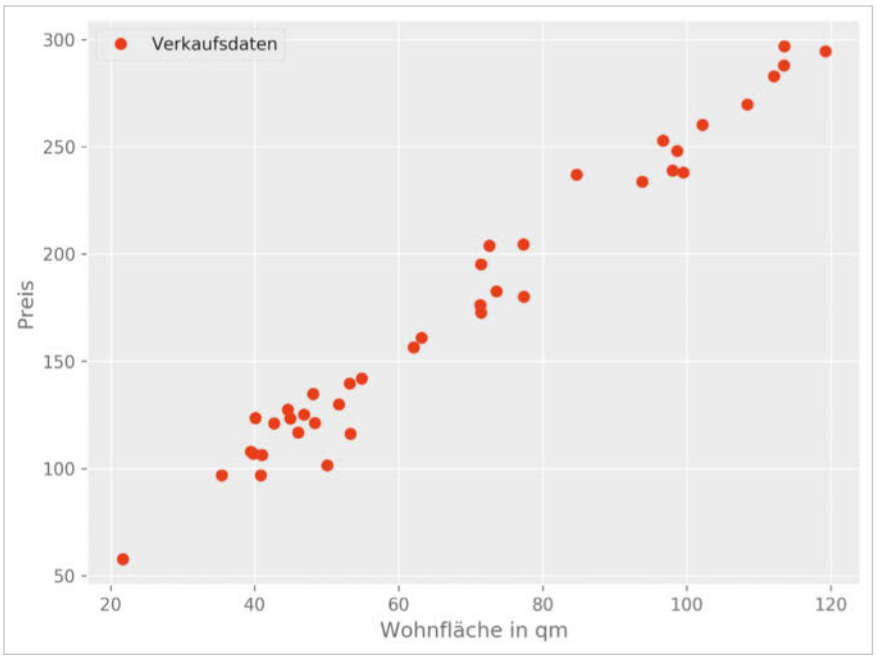
Die Methode *tf.global_variables_initializer* erzeugt einen „Initialisierer", den *session.run* ausführt. Danach haben alle Variablen ihren Anfangswert. Erst jetzt kann die eigentliche Berechnung wieder mit *session.run(..)* starten. Listing 1 zeigt den gesamten Ablauf im Überblick.

Einblicke in die Abläufe

Für den, der mehr über die Verarbeitung des Graphen in TensorFlow erfahren will, gibt es ein spezielles Werkzeug: TensorBoard (Abbildung 4). Damit lassen sich Berechnungsgraphen visualisieren und die Veränderungen der Variablen nachvollziehen. Voraussetzung dafür ist, dass die Python-Anwendung, die mit TensorFlow arbeitet, spezielle Informationen in eine Log-Datei schreibt. Da es sich bei TensorBoard um eine Webanwendung handelt, benötigt der Entwickler lediglich einen Browser, um damit zu arbeiten. Dadurch kann die Visualisierung genauso gut auf einem anderen als dem Entwicklungsrechner erfolgen.

Listing 1: *first_step.py*

```
<@$p>import tensorflow as tf

#graphen erzeugen
node_const_1 = tf.constant(3.0, tf.float32, name = 'const_1')
node_const_2 = tf.constant(4.0, tf.float32, name = 'const_2')

node_add = tf.add(node_const_1, node_const_2)

a = tf.placeholder(tf.float32, name = 'a')
node_mul = node_add * a

b = tf.Variable(1.0, tf.float32, name = 'b')
node_sub = node_mul - b

#ausführen
with tf.Session() as session:
    writer  =  tf.summary.FileWriter('./first_step', session.graph)
    init = tf.global_variables_ initializer()
    session.run(init)
    print(session.run(node_sub, {a:2.0}))

writer.close()
```



Zwischen Wohnfläche und Preis besteht ein linearer Zusammenhang (Abb. 5).

Was in TensorBoard zu sehen ist, legt der Entwickler selbst fest. Nur was die Python-Anwendung in die Log-Datei schreibt, kann er sich später anzeigen lassen. Ein Objekt vom Typ *tf.summary.FileWriter* übernimmt das Schreiben:

```
writer = tf.summary.FileWriter ↵
                ('./first_step', session.graph)
```

Das erste Argument ist das Verzeichnis, in das das *Writer*-Objekt die Log-Datei schreibt, das zweite stellt sicher, dass es gleich eine Beschreibung des Berechnungsgraphen aus dem aktuellen *session*-Objekt darin ablegt. Dies könnte genauso explizit über die Methode *writer.add_graph* erfolgen. Darüber hinaus gibt es weitere Methoden wie *add_summary* oder *add_event*, mit denen eine Anwendung der Log-Datei Informationen hinzufügen kann. Die Methode *writer.close* schließt die Log-Datei.

TensorBoard ist in der Installation von TensorFlow enthalten. Damit man sich allerdings den Inhalt der Log-Datei ansehen kann, muss man als Erstes die Webanwendung TensorBoard starten:

```
tensorboard --logdir="./first_step"
```

Mit einem Webbrowser ist sie anschließend über „localhost:6006“ zu erreichen. Unter dem Menü „Graphs“ findet man beispielsweise den von *first_step.py* erzeugten Graphen.

Tensor und die Mathematik

Dass der Begriff „Flow“ im Namen TensorFlow vom Fließen der Daten im Graphen kommt, leuchtet ein, aber woher kommt eigentlich die Bezeichnung „Tensor“? Der Begriff bezeichnet ein mathematisches Objekt der linearen Algebra und Differenzialgeometrie. In der Bibliothek TensorFlow ist der Begriff Tensor gleichbedeutend mit einem Array. Damit Berechnungen schnell ablaufen, ist es sinnvoll, nicht mit einzelnen Zahlen zu arbeiten, sondern gleichzeitig mit möglichst vielen, eben mit Datenstrukturen wie Arrays. Dieselbe Herangehensweise haben das bekannte Python-Modul *numpy* und die statistische Programmiersprache R.

Beispielsweise sind die beiden Zahlen *[1., 2.]* für Programmierer ein Array mit einer Dimension. Sie könnten für einen Punkt mit x- und y-Koordinate stehen. Der Mathematiker würde dies als einen Vektor bezeichnen. Auf die Welt von TensorFlow übertragen handelt es sich hier um einen Tensor 1. Stufe (Rank). Wichtig ist diese etwas mühsame Begriffsklauberei für das Verständnis der Dokumentation und der Beispiele im Internet.

Nächste Runde: Für einen Programmierer repräsentiert *[[1., 2.], [2., 3.]]* ein zweidimensionales Array, ein Mathematiker könnte von einer Matrix sprechen und in TensorFlow bezeichnet der Ausdruck einen Tensor 2. Stufe, ein Array mit drei Dimensionen einen Tensor 3. Stufe, eins mit vier Dimensionen einen Tensor 4. Stufe und so weiter. Damit lassen sich die unterschiedlichsten Daten abbilden und verarbeiten. Was noch im Gesamtbild fehlt, sind einzelne Zahlen, wie im Beispiel die Konstante *3.0*. Diese, von Mathematikern als Skalar bezeichnet, nennt TensorFlow einen Tensor der nullten Stufe.

Wichtig für Berechnungen ist auch, wie viele Elemente die einzelnen Dimensionen enthalten. Hier kommt der Begriff Shape ins Spiel. Ein Array *[[1., 2., 3.], [1., 2., 3.]]* hat einen Shape von *[2, 3]*, das heißt, die erste Dimension hat maximal zwei Elemente und in der zweiten Dimension gibt es maximal drei. Eine Übersicht über Tensoren zeigt die gleichnamige Tabelle.

Tensoren

3	Rank 0	einfache Zahl (Skalar) mit Shape []
[1., 2., 3.]	Rank 1	Vektor mit Shape [3]
[[1., 2., 3.], [1., 2., 3.]]	Rank 2	Matrix mit Shape [2, 3]
[[[1., 2., 3.]], [[5., 6., 7.]]]	Rank 3	Tensor mit Shape [2, 1, 3]

Preise der bisherigen Verkäufe (Ausschnitt)

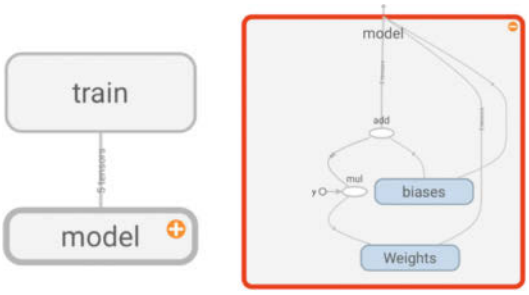
Grundstück	Fläche x [m²]	Preis y in [1000 Euro]
0	44,57	127,42
1	42,71	121,09
2	119,25	294,53
3	40,83	96,73
4	46,87	125,04

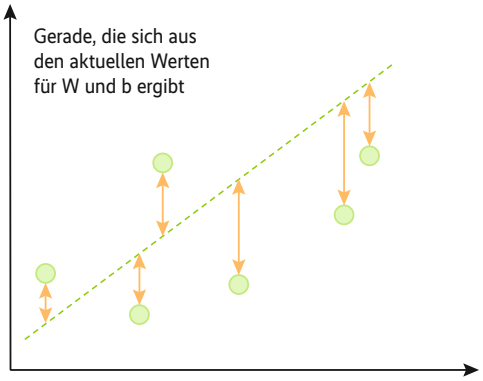
Lernen aus Erfahrungswerten

Erfahrungswerte, aus denen die Software lernen kann, sind ein wesentlicher Punkt bei jedem Verfahren im maschinellen Lernen. Die Entwicklung hin zur immer stärkeren Vernetzung aller Geräte und Informationen erzeugt zunehmend mehr Daten, die man für diesen Zweck nutzen kann. Dies ist einer der Gründe, warum gerade Firmen wie Google in letzter Zeit solche Fortschritte auf diesem Gebiet erzielt haben.

Das am Anfang beschriebene Beispiel aus der Gebäudeverwaltung braucht

TensorBuild kann zur besseren Übersicht alle Teile einer Formel zu Knoten zusammengefasst darstellen (Abb. 6).





Je geringer der Abstand zwischen der Geraden und den Punkten, desto besser die gefundenen Werte (Abb. 7).

möglichst viele Daten aus bereits durchgeführten Immobilienverkäufen, damit als Ergebnis realistische Preise herauskommen. Um die Berechnung zu vereinfachen, soll zunächst nur die Wohnfläche in Quadratmetern als Eingangsgröße Verwendung finden. Die Größe des Grundstücks bleibt im ersten Anlauf unberücksichtigt (siehe Tabelle „Preise der bisherigen Verkäufe“).

Wenn man sich die Verkaufsdaten in einer Grafik anzeigen lässt, sieht man, dass die einzelnen Verkäufe mehr oder weniger auf einer Linie liegen (Abbildung 5). Oder mathematischer ausgedrückt: Es besteht ein linearer Zusammenhang zwischen Wohnfläche und Preis.

Zum Schätzen neuer Preise wäre eine allgemeingültige Formel für eine Linie sinnvoll, die möglichst mitten durch die Punktewolke geht. Hätte man einen neuen Wert für eine Wohnfläche, könnte man diesen einfach in die Formel einsetzen und als Ergebnis einen adäquaten zu erwartenden Verkaufspreis erhalten. Ein Weg, an eine solche Formel zu kommen, ist das Verfahren „Lineare Regression“. Im maschinellen Lernen besagt der Begriff Regression nichts anderes, als dass das Ergebnis – hier der gesuchte Preis – eine Zahl ist. Grundsätzlich geht es hier um zwei Dinge: Entweder sucht man eine bestimmte Zahl (Regression) oder die Klasse, zu der ein bestimmtes Objekt gehört (Klassifikation). Ein typisches Beispiel für Klassifikationen wäre ein Sensor, der aufgrund von Eingangsinformationen erkennt, zu welcher Handelsklasse ein bestimmter Apfel gehört.

Ergebnisse durch lineare Regression

Die klassische Formel für eine Gerade ist relativ einfach:

```
y = W * x + b
```

Hinter *x* steht die Eingabegröße, im Beispiel die der Wohnfläche. Multipliziert man diese mit einem Faktor *W*, ergibt dies das Ergebnis *y*, den Preis. Um einen realistischen Preis *y* zu erhalten, muss man *x* entsprechend gewichten. Der Faktor wird daher normalerweise mit *W* (Weight) bezeichnet. Da es neben diesem noch einen anderen Einfluss geben kann, kommt in der Formel der Wert *b* (Bias) dazu:

```
with tf.name_scope('model'):
    W = tf.Variable([0.0], name = 'Weights')
    b = tf.Variable([0.0], name = 'biases')
    y = W * x_data + b
```

Bei der Umsetzung der Formel in TensorFlow werden *W* und *b* zu Variablen. *x_data* enthält die Flächen der bisherigen Verkäufe. Die Daten für das Verkaufsbeispiel befinden sich in einer Excel-Tabelle, die die Anwendung am Anfang in Arrays einliest. Hier kommt das Python-Modul *numpy* zum Einsatz. Mit dessen Arrays kann Python schnell arbeiten, und TensorFlow ist für den schnellen Umgang damit konzipiert. Die bekannten Verkaufsergebnisse enthält das Array *y_data*. In der Berechnungsformel für *y* verwendet TensorFlow automatisch seine eigenen Operatoren, und aus *x_data* macht es eine Konstante.

Der Ausdruck *with tf.name_scope ('model')* fasst alles, was zum Berech-

Listing 2: *linear_reg.py*

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import xlrd

DATA_FILE = 'immo_preis.xls'

# Daten einlesen
def read_data(filename):
    book = xlrd.open_workbook(filename, encoding_override = "utf-8")
    sheet = book.sheet_by_index(0)
    x_data = np.asarray([sheet.cell(i, 1).value for i in range(1, sheet.nrows)])
    y_data = np.asarray([sheet.cell(i, 2).value for i in range(1, sheet.nrows)])
    return x_data, y_data

x_data, y_data = read_data(DATA_FILE)

# Modell definieren
with tf.name_scope('model'):
    W = tf.Variable([0.0], name = 'Weights')
    b = tf.Variable([0.0], name = 'biases')
    y = W * x_data + b

# Training-Graph
with tf.name_scope('train'):
    loss = tf.reduce_mean(tf.square(y - y_data), name = 'loss')

    tf.summary.scalar('loss',loss)
    optimizer = tf.train.GradientDescentOptimizer(0.0001)
    train = optimizer.minimize(loss)

#Lernen
summary_op = tf.summary.merge_all()
print(summary_op)
with tf.Session() as session:
    writer  =  tf.summary.FileWriter('./linear_log', session.graph)
    init = tf.global_variables_initializer()
    session.run(init)

    #Training
    for i in range(1000):
        summary, _ = session.run([summary_op, train])
        writer.add_summary(summary, i)

    #Bewerten
    curr_W, curr_b, curr_loss  = session.run([W, b, loss])
    print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))

writer.close()
```

nungsmodell gehört, innerhalb von TensorFlow zusammen. Das Visualisierungswerkzeug TensorBuild stellt jetzt alle Teile, aus denen die Formel besteht, in einem Knoten dar. Will der Entwickler Genaueres wissen, klickt er einfach auf den Kasten und erhält die einzelnen Bestandteile angezeigt. Das macht gerade große Berechnungsgraphen wesentlich übersichtlicher (Abbildung 6).

Als Nächstes gilt es, die bestmöglichen Werte für *W* und *b* zu ermitteln. Das Beispiel in Listing 2 kann die Werte einfach berechnen, Tabellenkalkulationen und Statistikprogramme bringen dafür eine Funktion „Lineare Regression" mit. Je mehr Variablen in Betracht gezogen werden müssen, je größer die Datenmengen und je komplexer die Zusammenhänge zwischen den Variablen, desto eher kommt man mit klassischer Statistik nicht weiter.

Tutorial

Teil 1: Was ist maschinelles Lernen und wie sieht die grundsätzliche Herangehensweise bei selbstlernenden Python-Programmen aus?

Teil 2: Lernen mit neuronalen Netzen und ein Einblick in deren unterschiedliche Varianten mit Convolutional-Netzen und Deep Learning

Teil 3: Riesige, von anderen erstellte neuronale Netze verwenden oder für eigene Zwecke modifizieren

Trainieren mit vielen Daten

Das einfache Beispiel zeigt, wie man eine lineare Regression mit TensorFlow und Mechanismen des maschinellen Lernens implementiert. Es soll ein Modell trainieren, das für jede beliebige Wohnfläche einen Wert *y* als Verkaufspreis berechnet. Für die bekannten Wohnflächen in *x_data* soll *y* dabei möglichst nahe an *y_data* – den erzielten Verkaufspreisen – liegen (Abbildung 7).

Bei einem Verkauf mit einer Fläche von 44,57 m^2 (*x_data[0]*) gab es beispielsweise einen Verkaufspreis von 127 420 Euro. Bei einem ersten Versuch für *W* und *b* ergab die Formel einen Wert von 115,3 (*y[0]*) für diesen x-Wert. Die Differenz zwischen dem Wert 127,42 aus der Formel und dem tatsächlichen 115,3 ist also 12,12. Je kleiner diese Differenz, desto besser sind die gefundenen Werte für *W* und *b*.

Natürlich muss man diese Abweichung nicht nur für einen Punkt, sondern für alle ermitteln. Neben positiven Differenzen wird es sicherlich auch negative geben. Wichtig ist nur die Größe der Abweichung. Klassischerweise nimmt man das Quadrat der Differenz, denn damit fällt das Vorzeichen weg – minus mal minus ergibt plus. Bildet man aus diesen Differenzen im Quadrat das arithmetische Mittel (*mean*), hat man ein Maß dafür, wie gut die Werte für *W* und *b* sind. Je kleiner dieses Maß für die Abweichung (*loss*), desto besser ist es:

```
loss = tf.reduce_mean(tf.square(y - y_data), 7
                                    name = 'loss')
```

Der Operator *tf.reduce_mean* berechnet den Durchschnittswert und gibt als Ergebnis einen einzelnen Wert (Skalar) aus:

```
tf.reduce_mean([1.0,2.0,3.0,4.0])
```

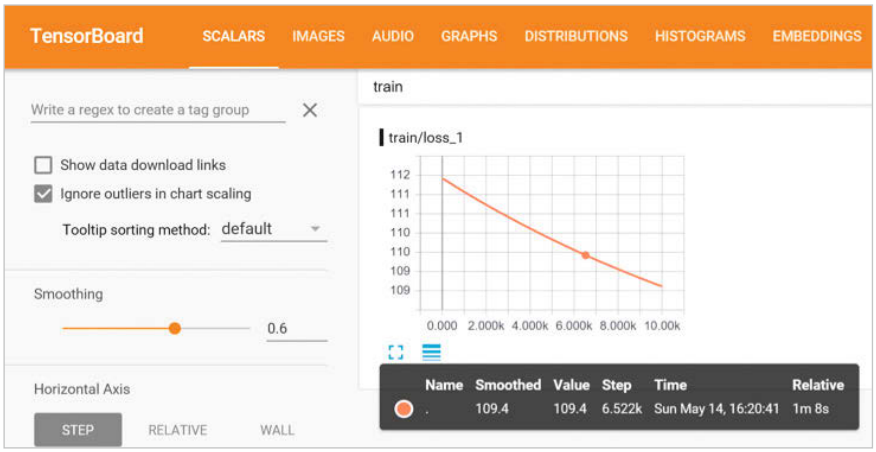
ergibt den Wert 2,5.

Optimieren durch steilen Abstieg

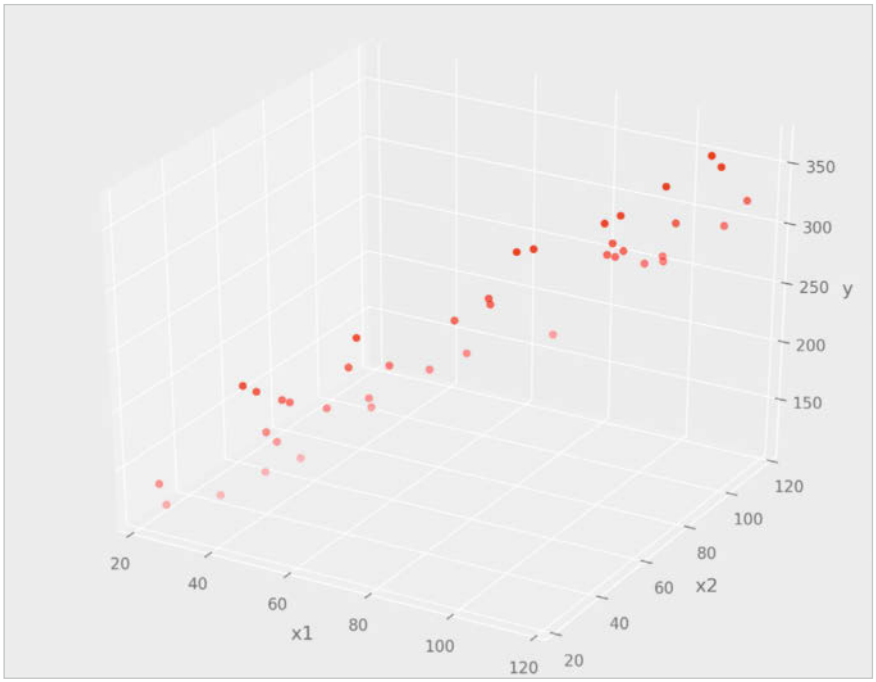
Jetzt einfach alle denkbaren Werte für *W* und *b* auszuprobieren, mag bei einfachen Modellen gehen. In komplizierteren Fällen bringt dies wenig und führt zu großen Laufzeiten. Daher hat sich das maschinelle Lernen etwas im Bereich der Optimierungsrechnung umgesehen. Hier gibt es die verschiedensten Strategien, um möglichst schnell zu optimalen Werten zu kommen. Eine der im maschinellen Lernen verwendeten Herangehensweisen ist das Gradientenverfahren (Gradient Descent), auch Verfahren des steilsten Abstiegs genannt.

Der als *loss* berechnete Wert ist zu minimieren, das heißt, der Wert sollte nach unten gehen. Die Versinnbildlichung wäre, wenn man sich vorstellt, man stünde auf einem Berg und es ist so neblig, dass man nichts mehr sieht, und Aufgabe wäre es, möglichst schnell den Weg nach unten zu finden. Man spürt mit seinen Füßen die Steigung des Bergs. Geht es steil nach unten, ist der Weg nicht schlecht. Geht es immer weiter nach unten, ist man noch nicht im Tal angekommen und muss weitergehen. Erst wenn es flach wird, ist das Ziel in der Nähe.

Ähnlich arbeitet das Gradientenverfahren. Es berechnet die „Steigung"



In TensorBoard lassen sich Variablen analysieren (Abb. 8).



Bei zwei Eingangsgrößen ergibt sich bei linearem Zusammenhang eine Ebene (Abb. 9).

von *loss*. Solange diese nach unten zeigt, sucht der Algorithmus nach Werten. TensorFlow enthält diese und weitere Optimierungsverfahren im Modul *tf.train*:

```
optimizer = ⤸
      tf.train.GradientDescentOptimizer(0.0001)
train = optimizer.minimize(loss)
```

Der *GradientDescentOptimizer* hat einen Parameter *learning_rate* – hier mit dem Wert *0.0001* vorbelegt –, der der Schrittweite entspricht, mit der der Optimierer sich dem Minimum annähert. Der eigentliche Lernvorgang erfolgt in der Schleife

```
for i in range(1000):
       session.run(train)
```

Bei jedem Durchlauf führt TensorFlow einmal den Optimierer aus, der sich auf die Suche nach dem minimalen Verlustwert (*loss*) macht. Jedes Mal justiert dieser die Werte für *W* und *b* nach. Nach tausend Durchläufen hat man beispielsweise das Ergebnis

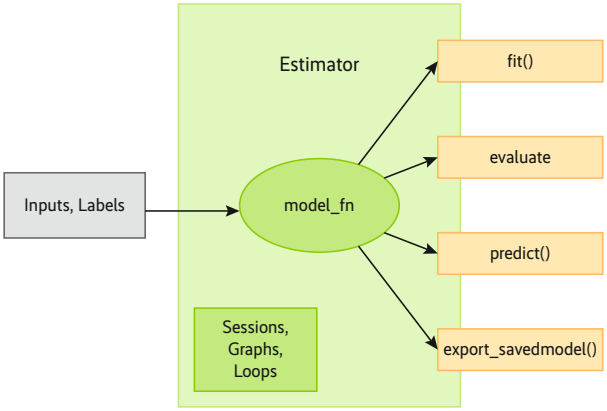
```
curr_W, curr_b, curr_loss = session.run([W, ⤸
                                    b, loss])
> W: [ 2.54120636] b: [ 0.22009991] loss: ⤸
                                    111.065
```

Die Methode *session.run* führt alle Argumente aus. Bei *W* ist der Rückgabewert eben der aktuelle Wert dieser Variablen. Listing 2 zeigt das komplette Programm.

Verbessern durch mehr Training

Der einfachste Ansatz, den Wert für *loss* weiter zu verkleinern, ist die Erhöhung der Anzahl der Trainingsläufe. Wie sieht es bei 10 000 oder bei 100 000 Versuchen aus? TensorBuild kann die Werte der Variablen in einer Grafik darstellen:

```
tf.summary.scalar('loss',loss)
```



TensorFlows High-Level-API Estimator enthält sofort anwendbare Verfahren des maschinellen Lernens (Abb. 10).

Dabei legt die Methode *tf.summary.scalar* fest, dass der Inhalt der Variablen *loss* in die Log-Datei für TensorBuild kommt. Im Modul *tf.summary* gibt es Methoden für die unterschiedlichen Datentypen. *tf.summary.merge_all* fasst die Abfrage der aktuellen Werte aller Variablen, die man beobachten will, in einer Operation zusammen:

```
summary_op = tf.summary.merge_all()
```

TensorFlow führt die Operation *summary_op* bei jedem Durchlauf der Trainingsschleife aus und liefert so die passenden Werte für die Log-Datei:

```
for i in range(1000):
    summary, _ = session.run([summary_op, ↵
                                    train])
    writer.add_summary(summary, i)
```

Die Methode *writer.add_summary* schreibt alles weg.

Die grafische Darstellung in TensorBoard (Abbildung 8) zeigt, dass der Wert für *loss* bei jedem Trainingsdurchlauf gesunken ist. Es wäre sicher einen Versuch wert, noch mehr Durchläufe berechnen zu lassen.

Zwei Größen, eine Ebene

Neben der Größe der Wohnfläche gab es beim Preisbeispiel am Anfang des Artikels eine weitere Eingangsgröße: die Fläche des Grundstücks. Lässt man sich die Eingangsgrößen *x1* und *x2* zusammen mit dem Ergebnis *y* in einer dreidimensionalen Grafik darstellen, sieht man, dass alle Punkte nahe einer Ebene liegen (Abbildung 9). Da hier lineare Zusammenhänge zu vermuten sind, könnte man es wieder mit der linearen Regression, jetzt der multiplen linearen Regression, versuchen.

Anstelle der bisher verwendeten Klassen und Methoden von TensorFlow aus dem Bereich Core soll die High-Level-API (Abbildung 10) von TensorFlow zum Einsatz kommen, die fertige Klassen für verschiedene Verfahren des maschinellen Lernens bereitstellt.

Estimator ist die „Instant-Version" – wie bei Instantsuppen schnell gemacht, schmeckt nicht schlecht und wer sich nicht mit Kochen beschäftigen will, ist damit gut beraten. Wer mehr will, kann mit TensorFlows Core-API die Abläufe und Komponenten genau selbst festlegen. Der erste Schritt bei der High-Level-API ist die Definition der Eingangsgrößen (*features*). Die Wohnraumgrößen aus den bisher getätigten Verkäufen sind wieder im Array *x1_data* gespeichert und die zusätzlichen Grundstücksflächen in *x2_data*. Die erzielten Preise enthält *y_data*:

```
features = [tf.contrib.layers.real_valued_ ↵
                    column("x1", dimension=1),
            tf.contrib.layers.real_valued_ ↵
                    column("x2", dimension=1)]
```

Die Methode *real_valued_column* legt fest, dass es sich bei der Eingangsgröße *x1*, der Wohnraumgröße, um reale Zahlen handelt. Da *x1_data* ein einfaches Array ist, hat dies die Dimension 1. Genauso ist es bei *x2_data*. Die eigentliche Verarbeitung übernimmt ein Objekt, das zu einer von der Klasse *estimator* abgeleiteten Klasse gehört. Im Falle der linearen Regression heißt diese Klasse *Linear-Regressor*:

```
estimator = tf.contrib.learn.LinearRegressor ↵
     (feature_columns=features, model_dir='./ ↵
                          linear_estimator_2')
```

Beim Erzeugen des *estimator*-Objekts bekommt es als Parameter die verwendeten Eingangsparameter *features* und das Verzeichnis, in dem TensorFlow alle Informationen ablegt, übergeben.

Bevor es an das Trainieren des Berechnungsmodells geht, muss das *estimator*-Objekt noch wissen, woher es die Eingabedaten bekommt. Dies geschieht über die Definition einer Funktion, im Beispiel *input_fn_train*, die ihre Daten aus den *numpy*-Arrays bezieht:

```
input_fn_train = tf.contrib.learn.io.numpy_ ↵
        input_fn({"x1":x1_data,"x2":x2_data}, ↵
                        y_data, num_epochs=1000)
```

TensorFlow kennt dafür die Funktion *numpy_input_fn*. Der Parameter *num_epochs* zeigt an, wie oft TensorFlow beim Lernen den gesamten Datenbestand verwenden soll. Das eigentliche Training übernimmt die Methode *fit* des *estimator*-Objekts, die als Parameter die Input-Funktion bekommt:

```
estimator.fit(input_fn=input_fn_train)
```

Mit wenigen Zeilen ist ein komplettes Modell trainiert. Wie gut es wirklich ist, ermittelt die Methode *estimator.evaluate*:

```
estimator.evaluate(input_fn=input_fn_train)
```

Für die Bewertung verwendet das Beispiel einfachheitshalber dieselben Daten wie für das Training. Im realen Leben sollten hier immer andere Daten zum Einsatz kommen, damit man sieht, was das Berechnungsmodell wirklich hergibt. Dazu könnte man beispielsweise die vorhandenen Daten in einen Trainings- und einen Evaluationsdatensatz aufteilen. Die Methode *evaluate* liefert neben anderen Informationen das Ergebnis der von TensorFlow verwendeten *loss*-Funktion. Daran kann man erkennen, wie gut das Modell ist oder wie sich Änderungen der Vorgaben auswirken. Für Vorhersagen zu bestimmten Eingabewerten gibt es die Methode *predict*:

```
estimator.predict(x={'x1': x1_pre, 'x2': ↵
                                    x2_pre})
```

Ausblick

Die grundsätzlichen Vorgehensweisen in TensorFlow, die im vorgestellten Beispiel mit relativ einfachen Daten zum Ziel führen, gelten genauso bei komplexeren Aufgaben – etwa das Erkennen handgeschriebener Ziffern. Im zweiten Teil dieser Artikelreihe (siehe Seite 52) geht es um neuronale Netze, die genau das tun. (ka@ix.de)

Gerhard Völkl

ist Fachjournalist für .NET-Techniken, Datenbankprogrammierung und Computergrafik.

Literatur

[1] Wolfgang Ertel; Grundkurs Künstliche Intelligenz; Springer Vieweg 2016

[2] Aurélien Géron; Hands-On Machine Learning with Scikit-Learn and TensorFlow; O'Reilly 2017

Alle Links: ix.de/ix1709042