# iX KOMPAKT

Frühjahr 2018

Ein Sonderheft des Magazins für professionelle Informationstechnik

## Machine Learning

Tutorials:
Maschinelles Lernen mit TensorFlow und Python
Automatisierte Textanalyse

## Modernes JavaScript

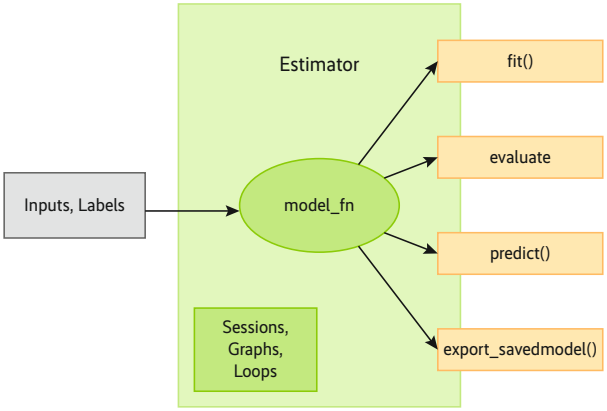
Tutorial: Webapps mit Angular
APIs mit GraphQL
Microservices mit Node.js

## Python

Tutorial: Parallelprogrammierung
Data Science mit Luigi

## C++17 und C++20

Effizientes Multithreading
Besserer Code durch Concepts




**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

MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK

***iX* kompakt Frühjahr 2018 – Programmieren heute**

Postfach 61 04 07, 30604 Hannover; Karl-Wiechert-Allee 10, 30625 Hannover

**Redaktion:** Telefon: 0511 5352-387, Fax: 0511 5352-361, E-Mail: post@ix.de

**Chefredakteur:** Jürgen Seeger (js@ix.de) -386, Stellv.: Dr. Oliver Diedrich (odi@ix.de) -616

**Konzeption und redaktionelle Leitung:** Kersten Auel (ka@ix.de) -367, Dr. Oliver Diedrich (odi@ix.de) -616

**Redaktion:** Dr. Jan Bundesmann (jab@ix.de) -384, Moritz Förster (fo@ix.de) -374, André von Raison (avr@ix.de) -361

**Ständiger Mitarbeiter:** Christian Kirsch (ck@ix.de)

**Autoren dieser Ausgabe:** Stephanie Fischer, Rainer Grimm, Mark Keinhörster, Jens Oliver Meiert, Dr. Holger Schwichtenberg, Sebastian Springer, Gerhard Völkl, Dr. Christian Winkler

**Abbildungen © Shutterstock:** dawou486 (Titel, S. 7), Rich Carey (S. 53), Napat (S. 79), Shane Gross (S. 125)

**Redaktionsassistenz:** Michael Mentzel (mm@ix.de) -153, Carmen Lehmann (cle@ix.de) -387

**Layout und Satz:** Madlen Grunert, Lisa Hemmerling, Kirsten Last, Steffi Martens, Matthias Timm, Ninett Wagner, Hinstorff Media, Rostock

**Korrektorat:** Barbara Gückel; Thomas Ballenberger, Ninett Wagner, Hinstorff Media, Rostock

**Titelidee:** *iX*, Titel- und Aufmachergestaltung: Matthias Timm

**Verlag:** Heise Medien GmbH & Co. KG, Postfach 61 04 07, 30604 Hannover; Karl-Wiechert-Allee 10, 30625 Hannover; Telefon: 0511 5352-0, Telefax: 0511 5352-129

**Geschäftsführer:** Ansgar Heise, Dr. Alfons Schräder

**Mitglieder der Geschäftsleitung:** Beate Gerold, Jörg Mühle

**Verlagsleiter:** Dr. Alfons Schräder

**Anzeigenleitung:** Michael Hanke (-167), E-Mail: michael.hanke@heise.de, www.heise.de/mediadaten/ix

**Druck:** Dierichs Druck + Media GmbH & Co. KG, Frankfurter Straße 168, 34121 Kassel

**Verantwortlich:** Textteil: Jürgen Seeger; Anzeigenteil: Michael Hanke

***iX* kompakt Frühjahr 2018 – Programmieren heute:** Einzelpreis € 12,90, Österreich € 14,20, Schweiz CHF 25,80, Luxemburg: € 14,80, Italien, Spanien: € 16,80



## Die Inserenten



## Veranstaltungen



Die hier abgedruckten Seitenzahlen sind nicht verbindlich. Redaktionelle Gründe können Änderungen erforderlich machen.

Python-Tutorial, Teil 2: Neuronale Netze und Deep Learning

# Zeichen-erkennung

**Gerhard Völkl**

Handgeschriebene Ziffern oder Gesichter zu erkennen, ist für Menschen eine leichte Sache. Computer hingegen taten sich damit bisher schwer. Verbesserte neuronale Netze (Deep Learning) erzielen verblüffende Ergebnisse, die man mit Python und der Bibliothek TensorFlow nachvollziehen kann.

Professor Yann LeCun und seine Kollegen aus dem Bereich der künstlichen Intelligenz der New York University standen vor dem Problem, dass es jede Menge Verfahren zur Erkennung von Schriften gab, aber keinerlei objektive Vergleichsmöglichkeiten. Welcher von zwei Algorithmen ist der bessere? Die Lösung, die Professor LeCun entwickelte, ist eine freie Bilddatenbank, die mehr als 70 000 Bilder von handgeschriebenen Ziffern enthält.

Diese als MNIST (Modified National Institute of Standards and Technology Database) bekannte Bilderbasis entstand aus 60 000 Ziffern, die von Schriftproben der Mitarbeiter des American Census Bureau stammen, und weiteren 10 000 von US-High-School-Studenten. An dieser Sammlung kann sich jeder neue Algorithmus versuchen. Einige schaffen es in die Bestenliste, die zusammen mit den Daten auf der Webseite zu finden ist (siehe „Onlinequellen“, [a]). Häufig verwenden die Verfahren des maschinellen Lernens die handgeschriebenen Ziffern der Mitarbeiter zum Trainieren und die der Studenten, um später die Qualität zu überprüfen.

## Integrierter Zugriff auf die Datenbasis

Bequemerweise enthält die im ersten Tutorialteil vorgestellte Bibliothek TensorFlow [1, b] bereits eine einfache Möglichkeit, an die MNIST-Bilder zu kommen:

```
from tensorflow.examples.tutorials.mnist ⤸
                                    import input_data
mnist = input_data.read_data_sets ⤸
                    ("MNIST_data/", one_hot=True)
```

Dabei holt die Methode *read_data_sets* die handgeschriebenen Ziffern aus dem Internet und legt sie im als Parameter übergebenen Verzeichnis *MNIST_data* ab. Anschließend stehen alle Daten in der Variablen *mnist* zur Verfügung. Diese enthält drei Datasets: *mnist.train* mit den Trainingsbildern (circa 55 000), *mnist.test* zum Testen (circa 10 000) sowie *mnist.validation* zum Überprüfen der Ergebnisse (etwa 5000). Jedes Dataset besteht aus den eigentlichen Bildern, die sich in einem Array, beispielsweise mit dem Namen *mnist.train.images*, befinden, und der dazugehörigen Auflösung, um welche Zahlen es sich handelt *(mnist.train.labels)*.

Wie in Abbildung 1 zu sehen, bestehen die Bilder aus 28 × 28 Bildpunkten. Da es sich um Graustufenbilder handelt, lässt sich jedes Pixel durch eine Zahl zwischen 0 für Weiß und 1 für Schwarz darstellen; 0,75 entspricht demnach einem dunklen und 0,25 einem hellen Grau. Zur Verein-

fachung der späteren Berechnungen sind die 784 Bildpunkte eines Bildes in einem eindimensionalen Array gespeichert.

So liefert der Ausdruck *mnist.train.images[100][6]* dann vom hundertsten Bild den sechsten Bildpunkt. Zum Ausgeben des Werts für die tatsächlich dargestellte Zahl reicht die Zeile

```
mnist.train.label[100]
>[ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
```

Wie man an der Ausgabe sieht, bekommt man, wenn es sich etwa um eine 4 handelt, nicht einfach den jeweiligen Integer-Wert zurück, sondern ein Array, bei dem das fünfte Element (Python zählt Arrays ab 0), also *target[4]*, auf 1 gesetzt ist:

```
0 = 1000000000
1 = 0100000000
2 = 0010000000
et cetera
```

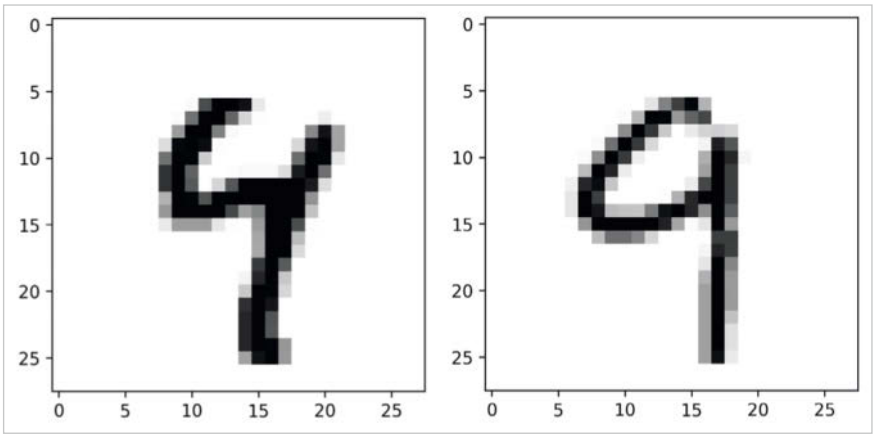
Diese Darstellung der Zahlen, manchmal One-Hot-Verschlüsselung genannt, vereinfacht die spätere Verarbeitung mit TensorFlow etwas.

## Neuronale Netze zur Schrifterkennung einsetzen

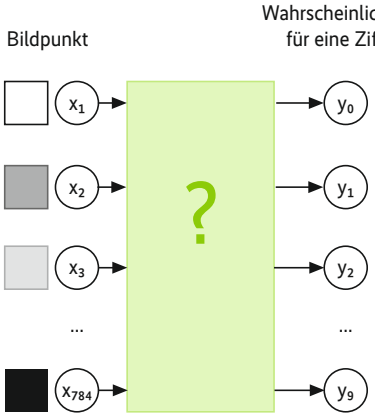
Seit den 1950er-Jahren schwirrt der Begriff „neuronale Netze“ durch die Welt der Programmierung. Damals hatten Forscher entdeckt, dass die Neuronen im Gehirn unterschiedlich gewichtete Eingabeimpulse bekommen und daraus einen Ausgabeimpuls erzeugen, der wieder anderen Neuronen als Eingabe dient. Computerwissenschaftler haben versucht, dies mit der damaligen Technik durch Matrixberechnungen nachzuprogrammieren.

Was heutzutage bei der aktuellen Technik neuronaler Netze übrig geblieben ist, sind auf jeden Fall die Matrixberechnungen. Aktuelle neuronale Netze haben eigentlich nichts mehr mit der Hirnforschung zu tun. Der Vergleich zwischen menschlichen Gehirnen und Elektronenrechnern ist zwar gutes Marketing, kann aber bei dem Blick auf die technischen Hintergründe eher verwirren.

Aus Informatiksicht sind neuronale Netze nur eine Verkettung von Funktionen, deren Parameter sich über Lernbeispiele anpassen lassen. Die Funktionen rechnen mit Matrizen oder mehrdimensionalen Feldern, die die Python-Bibliothek TensorFlow als Tensoren bezeichnet.



**Die freie Bilddatenbank MNIST enthält 60 000 Graustufenbilder handgeschriebener Ziffern von unterschiedlichen Personen (Abb. 1).**




**Aus den Bildpunkten als Eingabe ermittelt das neuronale Netz als Ausgabe die Wahrscheinlichkeiten für eine bestimmte Ziffer (Abb. 2).**

## Tutorialinhalt

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

## Eine nichttriviale Aufgabenstellung: Schrifterkennung

Komplizierte Aufgaben – wie das Erkennen handgeschriebener Ziffern – lassen sich mit einem einfachen EVA (Eingabe-Verarbeitung-Ausgabe)-Schema aus der klassischen Programmierung darstellen (siehe Abbildung 2).

Bei der Schrifterkennung ist die Eingabe die Farbe der einzelnen Bildpunkte. In diesem Fall sind es $28 \times 28$, also 784 Eingabevariablen. In der Mathematik werden diese klassischerweise mit dem Buchstaben $x$ bezeichnet, in diesem Fall also mit $x_1$ bis $x_{784}$. Oder man speichert sie gleich in einem Array $X$, bestehend aus 784 Elementen.

Die Ausgabe $Y$ ist die Wahrscheinlichkeit, dass es sich um eine bestimmte Ziffer handelt. Beispielsweise steht das Symbol $y_0$ für die Wahrscheinlichkeit, dass es sich um eine 0 handelt, $y_1$ dafür, dass es

### iX-TRACT

- Nach wie vor stellt das zuverlässige Erkennen handschriftlicher Ziffern und Zeichen maschinelle Verfahren vor große Herausforderungen.
- Neuronale Netze realisiert man im Wesentlichen über die Verkettung von mit Matrizen oder mehrdimensionalen Feldern rechnenden Funktionen, deren Parameter sich über Lernbeispiele anpassen lassen.
- Unter dem Stichwort „Deep Learning“ können in mehreren Schritten lernende neuronale Netze bei der automatisierten Beurteilung unterschiedlicher Methoden deutliche Verbesserungen bewirken.
- Über die Faltung verschiedener Funktionen – sogenannte Convolutional-Operationen, quasi Produkte von Funktionen – lassen sich verschiedene Verfahren miteinander kombinieren.

eine 1 ist, et cetera. Das Ergebnis *Y* ist wieder ein Array, und zwar aus zehn Elementen *($y_0$, …, $y_9$)*.

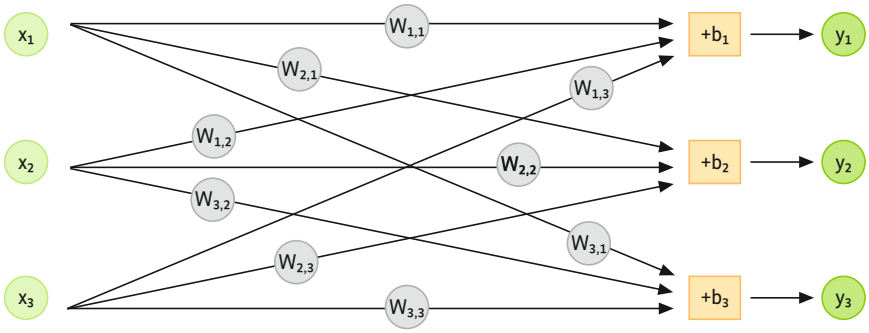
## Logistische Regression

Man könnte auch sagen, es gibt die Klassen 0 bis 9. Die Aufgabe besteht jetzt darin, zu ermitteln, zu welcher Klasse eine geschriebene Ziffer gehört. Daher stuft man dies im maschinellen Lernen als Klassifizierungsproblem ein. Ein Grund, warum die Ausgabe mit Wahrscheinlichkeiten arbeitet und nicht nur einfach das beste Ergebnis als Zahl ausgibt, ist die Option, nicht nur das bestmögliche Ergebnis nachsehen zu können, sondern genauso, welches das zweit- oder drittbeste gewesen wäre.

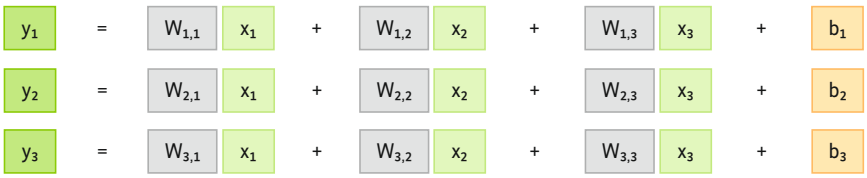
Der Ansatz, dass jeder Bildpunkt $x_i$ irgendwie die Ausgabe *Y* beeinflusst, ist zwar nicht sehr präzise, aber bestimmt nicht ganz verkehrt. Um es etwas genauer zu formulieren: Jede Eingabe *X* beeinflusst jede Ausgabe *Y* mit einem bestimmten Gewicht (Weight), einem bestimmten Faktor. Eine Formel für die Wahrscheinlichkeit, dass es sich um die Ziffer 0 handelt, könnte mit dem Bias *b* dann so aussehen:

$$y_0 = W_{11} * x_1 + W_{12} * x_2 + W_{13} * x_3 + \ldots + b_1$$

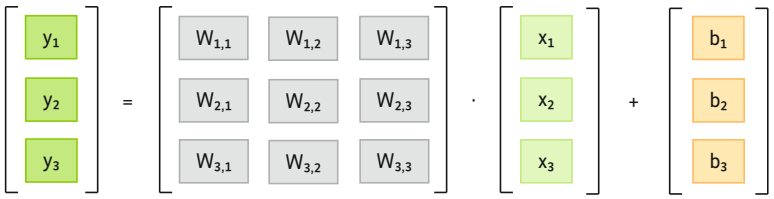
Die zehn Formeln – für jede Ziffer von 0 bis 9 eine – lassen sich in eine einzige Formel mit Matrizen und Vektoren zusammenfassen (siehe Abbildung 3). Das hat den Vorteil, dass TensorFlow genau darauf ausgelegt ist. Eine Einführung zu diesem Modul enthält der erste Artikel dieser Reihe ab Seite 42 [1].



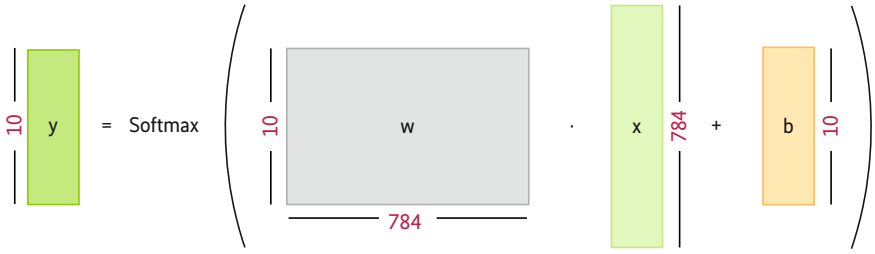

als Formel:




in Matrix-Schreibweise:




**Wie die Eingaben *x* die Ausgaben *y* mit bestimmten Gewichten beeinflussen, lässt sich als Flussdiagramm, Formel oder in Matrix-Schreibweise darstellen (Abb. 3).**




**Die gesamte Berechnung, mit Softmax zum Abschluss (Abb. 4)**

```
import tensorflow as tf
x = tf.placeholder(tf.float32, [None, 784])
```

Die Eingabe *x* enthält die Bildpunkte als Gleitkommazahlen *(float32)*. In diesem Fall ist *x* als zweidimensionales Array definiert, mit dem sich gleichzeitig mehrere Bilder übergeben lassen. Die erste Dimension steht für das Bild, die zweite für die Bildpunkte, so ist *x[4][6]* der sechste Bildpunkt des vierten Bilds. Das Array *x* ist in TensorFlow mit *[None, 784]* definiert. *None* steht dabei für eine unbekannte Größe, hier eine vorab nicht bekannte Zahl von Bildern.

Da *x* nur zur Übergabe von Daten aus Python an TensorFlow dient, reicht ein *tf.placeholder* vollkommen aus. Die Gewichtung *W* und das Bias *b* sind dagegen Variablen, die sich während der Berechnung ändern und daher zu definieren sind:

```
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```

Beide sind mit 0 vorbelegt.

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Nachdem TensorFlow mit der oben beschriebenen Formel *($y_0$ = …)* das Ergebnis ermittelt hat, kommt zusätzlich die Funktion *tf.nn.softmax* zum Einsatz. Die grundsätzliche Aufgabe einer Softmax-Funktion ist es, große Wahrscheinlichkeiten herauszuheben und niedrige abzudämpfen, damit das wahrscheinlichste Ergebnis in den Vordergrund tritt.

Alle von der vorhergehenden Berechnung stammenden Werte bringt Softmax im Wertebereich zwischen 0 und 1 unter. Der Wert 1 bedeutet, dass das Ergebnis zu 100 %, der Wert 0, dass es gar nicht zutrifft, und 0,5 entspricht einer 50%igen Wahrscheinlichkeit. Mehr dazu im Kasten „Softmax“.

## Ohne Training geht es nicht

Im nächsten Schritt sind die Werte in der Matrix *W* und dem Vektor *b* so anzupassen, dass das Verfahren möglichst viele Ziffern richtig erkennt. Dafür benötigt man ein Maß (Loss Function), wie gut diese aktuell gefundenen Werte sind. Hierfür kommt häufig die Berechnung der Kreuzentropie (Cross Entropy) zum Einsatz. Dies ist ein Maß zum Vergleich zweier Ergebnisreihen, die aus den glei-