

## Künstliches neuronales Netzwerk zur Erkennung von Zahlen

Künstliche neuronale Netzwerke haben in der Informatik an grosser Bedeutung gewonnen. Intelligente Autos, intelligente Computer und intelligente Überwachungssysteme haben immer mehr Einfluss auf unseren Alltag und sind nicht nur noch Zukunftsvisionen. In dieser Arbeit wird aufgezeigt, wie mithilfe des Python Paket `scikit-learn`<sup>1</sup> ein künstliches neuronales Netzwerk erstellt wird. Dieses wird verwendet, um handgeschriebene Zahlen von null bis neun zu identifizieren. Ähnliche Algorithmen werden von der Post eingesetzt, um die Postleitzahl automatisch zu erkennen, um so die Briefe zu sortieren. Im zweiten Teil der Arbeit wird beschrieben, wie die eigene Handschrift mittels des erstellten künstlichen neuronalen Netzwerks erkannt werden kann.

Ein künstliches neuronales Netzwerk ist dem menschlichen Gehirn nachempfunden. Es besteht aus digitalen Neuronen die Units genannt werden. Jede Unit besitzt eine mathematische Funktion die aufgrund von mehreren Inputs von anderen Units einen Output berechnet. Mehrere Units zusammen bilden einen Layer welcher in Abbildung 1 als Spalte erkennbar ist. Wie bei seinem biologischen Vorbild werden mehrere der Units miteinander verbunden. Eine Unit hat Verbindungen zu Units seines direkten vor- und nachfolgenden Layer. Abbildung 1 zeigt grafisch auf wie ein minimales künstliches neuronales Netzwerk aufgebaut ist. Es besteht immer aus einem Input-, beliebig vielen aber mindestens aus einem Hidden- sowie einem Outputlayer.

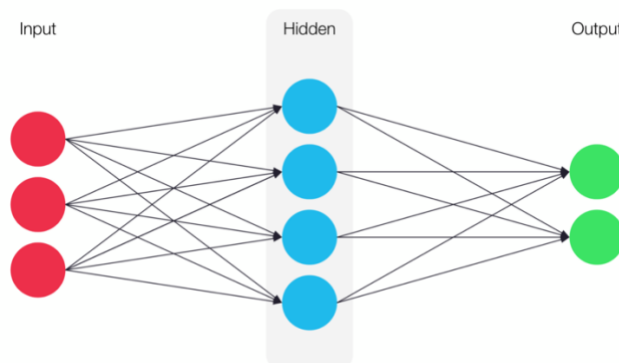


Abbildung 1: Künstliches neuronales Netzwerk

(Quelle: <https://rocketloop.de/kuenstliche-neuronale-netze/>)

Ein künstliches neuronales Netzwerk benötigt Daten, um zu lernen und um eine eigene künstliche Intelligenz zu erlangen. Daher ist ein wichtiger Schritt die Aufbereitung der Daten. Es werden nicht nur viele Daten benötigt, um ein künstlich neuronales Netzwerk zu trainieren, sondern auch die Qualität der Daten muss stimmen. Wird ein künstliches neuronales Netzwerk mit schlechten Daten trainiert, liefert es einen schlechten Output. Deshalb ist die Datensammlung und Aufbereitung ein sehr grosser und wichtiger Teilschritt, damit ein künstlich neuronales Netzwerk den gewünschten Output liefern kann. Das Python Paket `scikit-learn` bittet über tausend Datensätze von handgeschriebenen Zahlen, die für das Training des künstlichen neuronalen Netzwerks verwendet werden können. Ein Datensatz besteht aus einem 64-Bit Bild und derjenigen Zahl, welche das Bild repräsentiert. Es gibt verschiedene Arten von künstlich neuronalen Netzwerken. Eine sehr beliebte Art ist der Multilayer-Perzeptron. Snippet 1 zeigt auf wie ein Multilayer-Perzeptron

<sup>1</sup> `scikit-learn`. (2019). *scikit-learn*. Abgerufen am 28. 10 2019 von <https://scikit-learn.org/stable/index.html>

Netzwerk erstellt und mit den Datensätzen trainiert wird. Ein wichtiger Punkt beim Trainieren eines künstlich neuronalen Netzwerks ist, dass ein Teil der Daten beiseitegelegt werden, um die Erkennungsrate zu berechnen. Somit kennt das künstlich neuronale Netzwerk die Testdaten nicht und es kann somit eine bessere Aussage über die Genauigkeit des künstlich neuronalen Netzwerks gemacht werden. Im Snippet 1 werden 75% der Daten für das Training verwendet und 25% der Daten, um das trainierte künstliche neuronale Netzwerk zu testen (Matthew A. Russell, 2019).

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn import datasets
digits = datasets.load_digits()
X, y = digits.data / 255., digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=100, solver='adam', tol=1e-4, random_state=1, learning_rate_init=.1)
mlp.fit(X_train, y_train)
print('Prozentwert der Trainingsdaten: {0}'.format(mlp.score(X_train, y_train) * 100))
print('Prozentwert der Testdaten: {0}'.format(mlp.score(X_test, y_test) * 100))
```

Snippet 1: Datenaufbereitung und KNN trainieren

Prozentwert der Trainingsdaten: 100.0  
Prozentwert der Testdaten: 97.56

Abbildung 2: Output Snippet 1

Für die Handschrifterkennung wurde ein künstlich neuronales Netzwerk mit einem Hiddenlayer mit 100 Units erstellt. Dies erzielte mit dem Testdatensatz die Beste Erkennungsrate von 97.56%. Beim Training muss darauf geachtet werden, dass das Netzwerk nicht übertrainiert wird. Was bedeutet, dass das Netzwerk alle Daten auswendig gelernt hätte und somit bei Werten, die sie noch nie gesehen hat, keine Chance hat diese zu erkennen. Ein übertrainiertes Netzwerk kann identifiziert werden, indem es sich bei Trainings- und Testdaten um verschiedene Datensätze handelt. Bei scikit-learn ist es möglich das Multilayer-Prezeptor Netzwerk in vielen Bereichen zu konfigurieren. Die wichtigsten Parameter sind in der folgenden Tabelle 1 kurz erklärt.

Parameter	Funktion
hidden_layer_sizes	Über diesen Parameter können die Anzahl Layer und Anzahl der Units angegeben werden.
max_iter	Beschränkung der maximalen Anzahl von Iterationen, die betätigt werden dürfen, bis die Toleranz erreicht wird.
tol	Die Toleranz die erreicht werden muss, in den angegebenen Iterationen.
solver	Über diesen Parameter kann angegeben werden, welche Funktion angewendet werden soll für die Units Gewichtoptimierung.
random_state	Mittels dieses Parameters kann der Startwert für die Zufallszahlengenerator bestimmt werden.
learning_rate_init	Mittels dieses Parameters, kann angegeben werden ab welchem Wert die Gewichte der Units aktualisiert werden müssen.

Tabelle 1: MLP-Netzwerk Parameterbeschreibung

Die Anzahl von Input-Units und Outputs-Units kann nicht eingestellt werden, da diese jeweils über die Datensätze definiert werden. Im aufgeführten Beispiel der Handschrifterkennung handelt es sich um 64 Input Units, da ein Bild aus 64-Bit besteht und jedes Bit als Input Unit verwendet wird. Diesbezüglich gibt es zehn Output Units, welche jeweils eine Zahl zwischen null und neun repräsentieren.

Damit die eigene Handschrift nun als Input in das künstlich neurale Netzwerk eingegeben werden kann, muss zuerst das Bild auf die korrekte Datenstruktur angepasst werden. Der scikit-learn Datensatz besteht aus Bildern mit einer maximalen Auflösung von 64 Pixel. Was einem Bildformat von acht mal acht entspricht. Somit muss der Input zuerst auf diese Grösse skaliert werden. Für die Erkennung der Zahl werden keine Informationen zu den Farben im Bild benötigt. Somit wird das herunterskalierte Bild als Einkanalbild konvertiert, was als Graustufenbild interpretiert wird. Dies bedeutet, dass nur die Leuchtdichte gespeichert wird, jedoch keine Farben. Mit dem Python Paket `numpy`<sup>2</sup> kann aus dem Graustufenbild ein Datenarray erzeugt werden, welches scikit-learn als Input akzeptiert. Snippet 2 zeigt auf wie ein Bild eingelesen, skaliert, konvertiert und in ein Datenarray umgewandelt wird (Matthew A. Russell, 2019).

```
import numpy
from PIL import Image
img = Image.open('./img/number_seven.png')
img = img.resize((8, 8), Image.ANTIALIAS).convert('L')
arr = numpy.array(img)
arr = arr / 255.
predict = mlp.predict(arr.reshape(1, -1))
print('Vorhersage: {0}'.format(predict))
```

### Snippet 2: Eigene Handschrift erkennen

Wird nun Abbildung 3 verwendet als Input bei Snippet 2, somit wird das über 500 mal 500 Pixel grosse Bild auf ein acht mal acht grosses Bild herunterskaliert und eine Graustufenumwandlung vorgenommen. Abbildung 4 zeigt auf wie das Graustufenbild aussieht, was aus Abbildung 3 erstellt wird. **Error! Reference source not found.** repräsentiert ein Array aus 64 Werten. Jeder einzelne Wert spiegelt die Leuchtdichte eines Pixels, die wiederum für die 64 Input-Units verwendet werden.

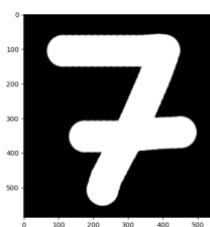


Abbildung 3: Testobjekt

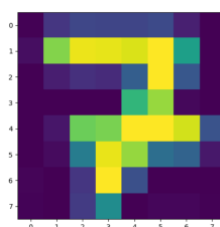


Abbildung 4: Graustufenbild

```
[ [
0.004 0.157 0.212 0.208 0.208 0.224 0.086 0.000
0.035 0.816 0.973 0.965 0.945 1.000 0.565 0.000
0.000 0.082 0.137 0.122 0.298 1.000 0.271 0.000
0.000 0.000 0.000 0.000 0.659 0.847 0.020 0.000
0.000 0.059 0.776 0.800 1.000 1.000 0.933 0.255
0.000 0.027 0.420 0.969 0.843 0.361 0.318 0.063
0.012 0.000 0.157 1.000 0.243 0.000 0.000 0.000
0.008 0.000 0.173 0.486 0.000 0.016 0.016 0.004
]]
```

Abbildung 5: Datenarray

Das im Snippet 1 erstellte Multilayer-Prezeptor Netzwerk mit der Erkennungsrate von 97.56% hat alle handgeschriebenen Zahlen des Autors korrekt erkannt.

<sup>2</sup> NumPy Developers. (2019). *NumPy*. Abgerufen am 28. 10 2019 von <https://numpy.org>

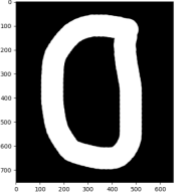
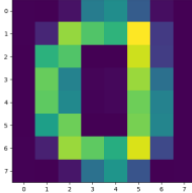
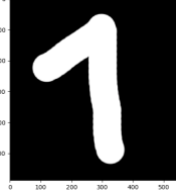
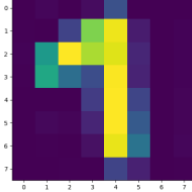

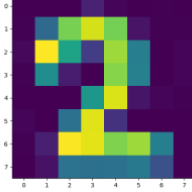
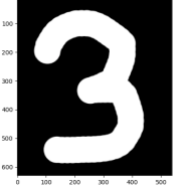
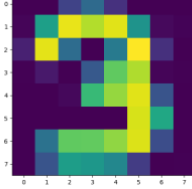
Künstliche neuronale Netzwerke bekamen in den letzten Jahren eine enorme Popularität. Dies nicht zuletzt aufgrund von sozialen Netzwerken, die mithilfe von ihren Mitgliedern einen gigantischen Datensatz aneigneten. Doch haben alle neuronalen Netzwerke ein gemeinsames Problem. Die Ergebnisse sind schwer zu interpretieren. Dies ist, warum viele Menschen ein künstliches neuronales Netzwerk als Black Box betrachten. Das vom Autor beschriebene Multilayer-Prezeptor Netzwerk im Snippet 1 ist mit seinen 64 Input-Units, den 100 Hidden-Units und dem Output-Unit noch sehr verständlich und die einzelnen Gewichte könnten mit überschaubarem Aufwand zurückgerechnet werden. Jedoch bei Grösseren künstlichen neuronalen Netzwerken, die für selbstfahrende Autos aufgebaut werden, ist dies fast nicht mehr möglich, wo Millionen von Gewichten an Millionen von Units gleichzeitig betrachtet und ausgewertet werden müssen.

## Testresultate

Mit Snippet 3 und den Bildern aus der Tabelle 2 sind alle Ziffern von null bis neun erfolgreich getestet.

```
import numpy
from PIL import Image
images = ['0.png', '1.png', '2.png', '3.png', '4.png', '5.png', '6.png', '7.png', '8.png', '9.png']
for test_img in images:
    img = Image.open('./img/' + test_img)
    plt.imshow(img)
    plt.show()
    img = img.resize((8, 8), Image.ANTIALIAS).convert('L')
    plt.imshow(img)
    plt.show()
    arr = numpy.array(img)
    arr = arr / 255.
    predict = mlp.predict(arr.reshape(1, -1))
    print('Bild Nummer {0} Vorhersage: {1}'.format(test_img, predict))
```

Snippet 3: Test – Handschrift Autor

Bild	Graustufenbild (8 x 8)	Zahl	Resultat
		0	Bild Nummer 0.png Vorhersage: [0]
		1	Bild Nummer 1.png Vorhersage: [1]
		2	Bild Nummer 2.png Vorhersage: [2]
		3	Bild Nummer 3.png Vorhersage: [3]

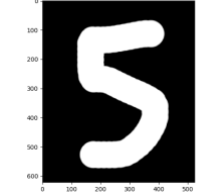
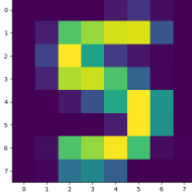
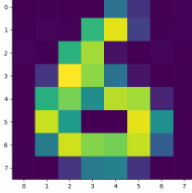
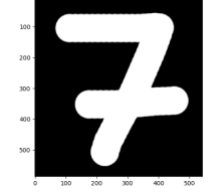
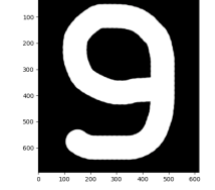
		4	Bild Nummer 4.png Vorhersage: [4]
		5	Bild Nummer 5.png Vorhersage: [5]
		6	Bild Nummer 6.png Vorhersage: [6]
		7	Bild Nummer 7.png Vorhersage: [7]
		8	Bild Nummer 8.png Vorhersage: [8]
		9	Bild Nummer 9.png Vorhersage: [9]

Tabelle 2: Testbilder - Handschrift Autor

asdf

## Literaturverzeichnis

Matthew A. Russell, M. K. (2019). *Mining the social web*. O'Reilly Media. Inc.

## Abbildungsverzeichnis

Abbildung 1: Künstliches neuronales Netzwerk.....	1
Abbildung 2: Output Snippet 1 .....	2
Abbildung 3: Testobjekt.....	3
Abbildung 4: Graustufenbild.....	3
Abbildung 5: Datenarray .....	3

## Tabellenverzeichnis

Tabelle 1: MLP-Netzwerk Parameterbeschreibung .....	2
Tabelle 2: Testbilder - Handschrift Autor .....	6

## Snippetverzeichnis

Snippet 1: Datenaufbereitung und KNN trainieren .....	2
Snippet 2: Eigene Handschrift erkennen .....	3
Snippet 3: Test – Handschrift Autor .....	5