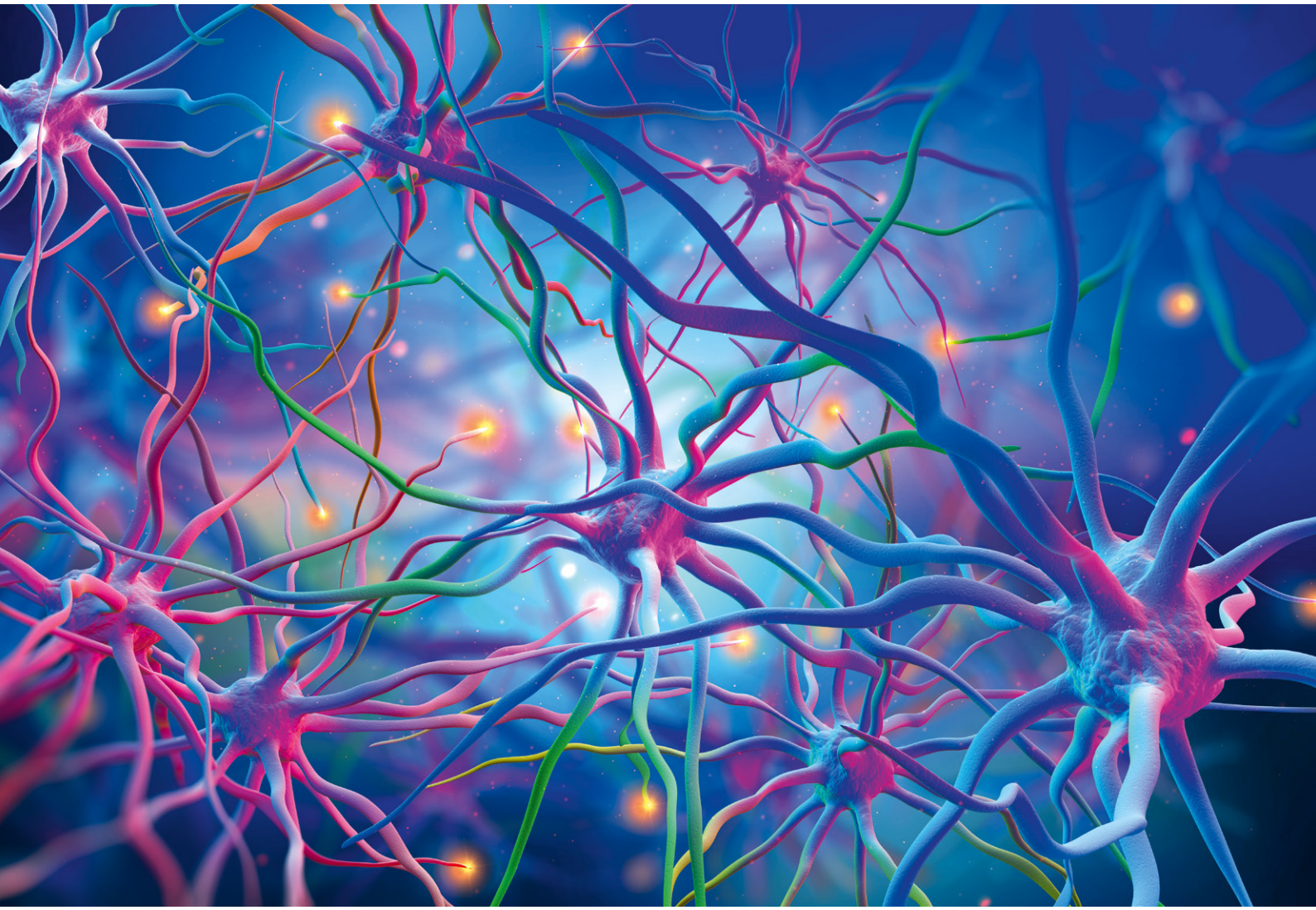
KI-Praxis II

Teil 2

Neuronale Netze – Aufbau und Training

Im ersten Beitrag zu dieser Reihe wurden im ELVjournal 3/2021 die Grundlagen des maschinellen Lernens erläutert. Den Abschluss bildete dort die grafische Analyse verschiedener Irisarten in einem Jupyter Notebook auf einem Raspberry Pi. In diesem Artikel soll es nun darum gehen, ein Neuronales Netz und die Methoden des Deep Learning einzusetzen. Auch diese Aufgabe kann wieder problemlos mit einem Raspberry Pi 4 gelöst werden.



Auszug aus dem Iris-Datensatz

	sepal length	sepal width	petal length	petal width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
50	7	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
53	5.5	2.3	4	1.3	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor
55	5.7	2.8	4.5	1.3	Iris-versicolor
56	6.3	3.3	4.7	1.6	Iris-versicolor
57	4.9	2.4	3.3	1	Iris-versicolor
58	6.6	2.9	4.6	1.3	Iris-versicolor
59	5.2	2.7	3.9	1.4	Iris-versicolor
100	6.3	3.3	6	2.5	Iris-virginica
101	5.8	2.7	5.1	1.9	Iris-virginica
102	7.1	3	5.9	2.1	Iris-virginica
103	6.3	2.9	5.6	1.8	Iris-virginica
104	6.5	3	5.8	2.2	Iris-virginica
105	7.6	3	6.6	2.1	Iris-virginica
106	4.9	2.5	4.5	1.7	Iris-virginica
107	7.3	2.9	6.3	1.8	Iris-virginica
108	6.7	2.5	5.8	1.8	Iris-virginica
109	7.2	3.6	6.1	2.5	Iris-virginica

Tabelle 1

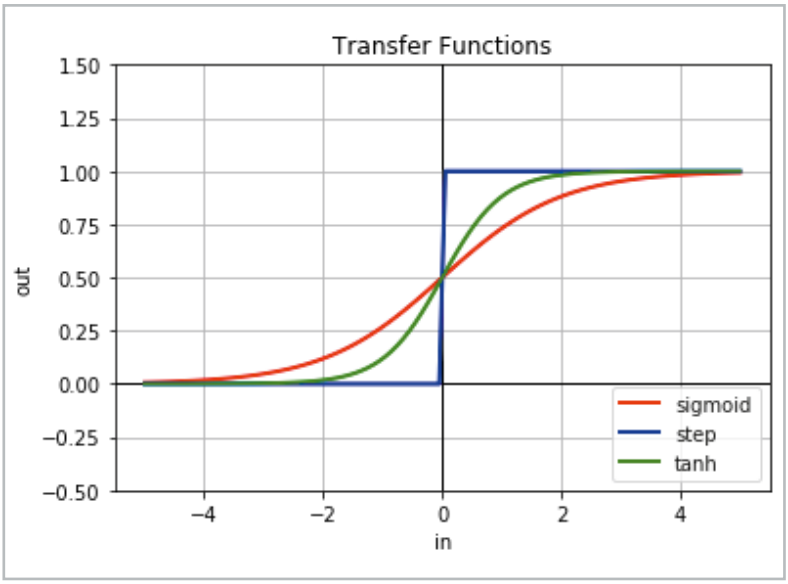


Bild 1: Übertragungsfunktionen

Neuronale Netze

Der Iris-Datensatz besteht aus drei Klassen (Iris setosa, Iris virginica und Iris versicolor), die nicht linear voneinander trennbar sind. D. h., es ist nicht möglich, einen Satz linearer Funktionen zu finden, die aus den Blütenblattdaten (sepal-/petal-length, sepal-/petal-width, Angaben in cm) eindeutig die Zugehörigkeit zu einer bestimmten Irisart berechnen. Damit hat man ein klassisches Beispiel für den Einsatz eines Neuronalen Netzes vor sich. Die Tabelle 1 zeigt einen Auszug aus dem Iris-Datensatz.

Es stehen damit vier Messwerte pro Datensatz zur Verfügung. Daraus ergibt sich, dass das zugehörige Neuronale Netz vier Knoten in der Eingabeschicht (Input Layer) haben muss. Da jedes Blütenexemplar in eine von drei Klassen einsortiert werden soll, werden drei Ausgabeknoten benötigt.

Netze mit mehreren Schichten

Im Folgenden wird das sogenannte „mehrschichtige Perzeptron“ (MLP: Multi Layer Perceptron) verwendet. Dabei handelt es sich um ein künstliches neuronales Netzwerkmodell, das beliebige numerische Eingabedaten auf festgelegte Ausgabedaten abbildet. Ein MLP besteht definitionsgemäß immer aus mehreren Schichten. Typischerweise ist jede Schicht vollständig mit der nachfolgenden verbunden. Die Knoten der Schichten bestehen aus künstlichen Neuronen, die mit nichtlinearen Aktivierungsfunktionen arbeiten. Das Konzept der Aktivierungsfunktion geht auf das Verhalten von biologischen Neuronen im menschlichen Gehirn zurück. Diese kommunizieren über aktive oder inaktive Aktionspotenziale. Dabei handelt es sich um einen Mittelweg zwischen einem eher digitalen Ein- und Ausschalter und einem rein analogen System mit beliebigen Zwischenwerten. Bei künstlichen neuronalen Netzen imitieren die Knoten das neuronale Verhalten durch Anwendung einer Schwellwertfunktion. Bild 1 zeigt verschiedene gängige Varianten dazu:

- Sigmoide
- Stufenfunktion
- Tangens hyperbolicus

„Sigmoid“ bezieht sich auf eine Krümmung in zwei Richtungen. Es existieren verschiedene Sigmoid-Funktionen. Die sogenannte logistische Funktion kommt deshalb häufig zum Einsatz, weil sie mehrere Vorteile bietet. Der Rechenaufwand dafür ist zwar höher als bei der Stufenfunktion, durch effiziente Algorithmen kann die logistische Funktion dennoch sehr schnell ausgewertet werden. Zudem verhält sie sich im späteren Training eines Netzwerks besonders vorteilhaft im Vergleich zur Stufenfunktion. Die anderen Varianten wie z. B. die Tanhyp-Funktion kommen daher eher selten zum Einsatz.

Über die Schwellwertfunktion erhält das Neuronale Netz eine gewisse Nichtlinearität. Ohne diese wäre seine Funktionalität deutlich eingeschränkt. Die Theorie dazu ist vergleichsweise komplex. Es kann jedoch leicht gezeigt werden, dass rein lineare Transformationen nicht in der Lage sind, komplexe Aufgaben wie Bild- oder Spracherkennung effektiv zu lösen.

Nach der Festlegung einer Transferfunktion kann man nun einzelne Neuronen zu einem Netzwerk verknüpfen. Zwischen der Eingabe- und der Ausgabeebene können sich dabei eine oder mehrere innere oder „verborgene“ Ebenen befinden.

Es gibt kein Standardverfahren für die Bestimmung der Anzahl von Schichten und Knoten in einem Neuronalen Netzwerk. Die Topologie eines Netzes hängt stark von der jeweiligen Aufgabenstellung ab. Hier kommen immer auch die Erfahrung und der Wissensstand des Entwicklers mit ins Spiel. Häufig muss man verschiedene Kombinationen austesten, um herauszufinden, welche Netzstruktur wirklich optimal ist.

Für die Iris-Klassifizierung hat sich die in Bild 2 gezeigte Topologie als guter Einstieg erwiesen.

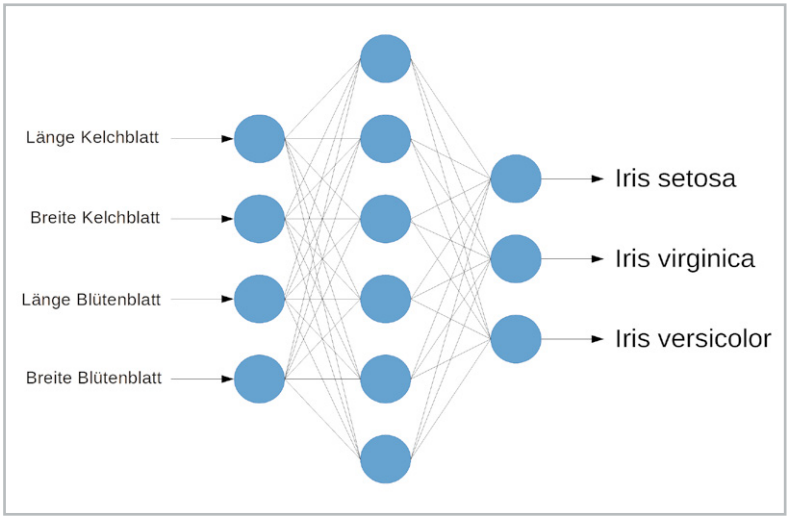


Bild 2: Neuronales Netzwerk zur Klassifizierung von Irisarten

Daten-Vorverarbeitung

Nach diesen Vorüberlegungen kann nun das im ersten Beitrag vorgestellte und installierte Jupyter Notebook auf dem Raspberry Pi gestartet werden. Zudem sollte das Downloadpaket zu diesem Artikel [1] auf den Pi geladen werden. Dies kann direkt über eine Internetverbindung oder über den Zwischenschritt PC und USB-Stick (bzw. einen FTP-Client wie Filezilla [2]) erfolgen. Bild 3 zeigt das geöffnete Notebook aus dem Downloadpaket auf dem Pi.

Damit steht der Entwicklung eines Neuronalen Netzes auf dem Raspberry nichts mehr im Wege.

Wie auch die Tabelle 1 zeigt, enthält der Iris-Datensatz fünf Spalten. Die Aufgabe besteht darin, die Klasse, d. h. die Werte in der fünften Spalte, vorherzusagen. Als Eingabedaten dienen die ersten vier Spalten, also die Kelchblattlänge (sepal length), Kelchblattbreite (sepal width), Blütenblattlänge (petal length) und Blütenblattbreite (petal width).

Nach dem Start des Jupyter Notebook mit der Eingabe

jupyter notebook

in einem Terminalfenster wird das aktuelle Verzeichnis in der Dateiübersicht des Jupyter-Notebooks angezeigt. In diesem Verzeichnis sollte auch die Datei aus dem Downloadpaket abgelegt sein, die man jetzt auswählt und öffnet.

Eine detaillierte Beschreibung zur Installation des Jupyter Notebooks finden Sie im ersten Teil der Beitragsreihe [3].

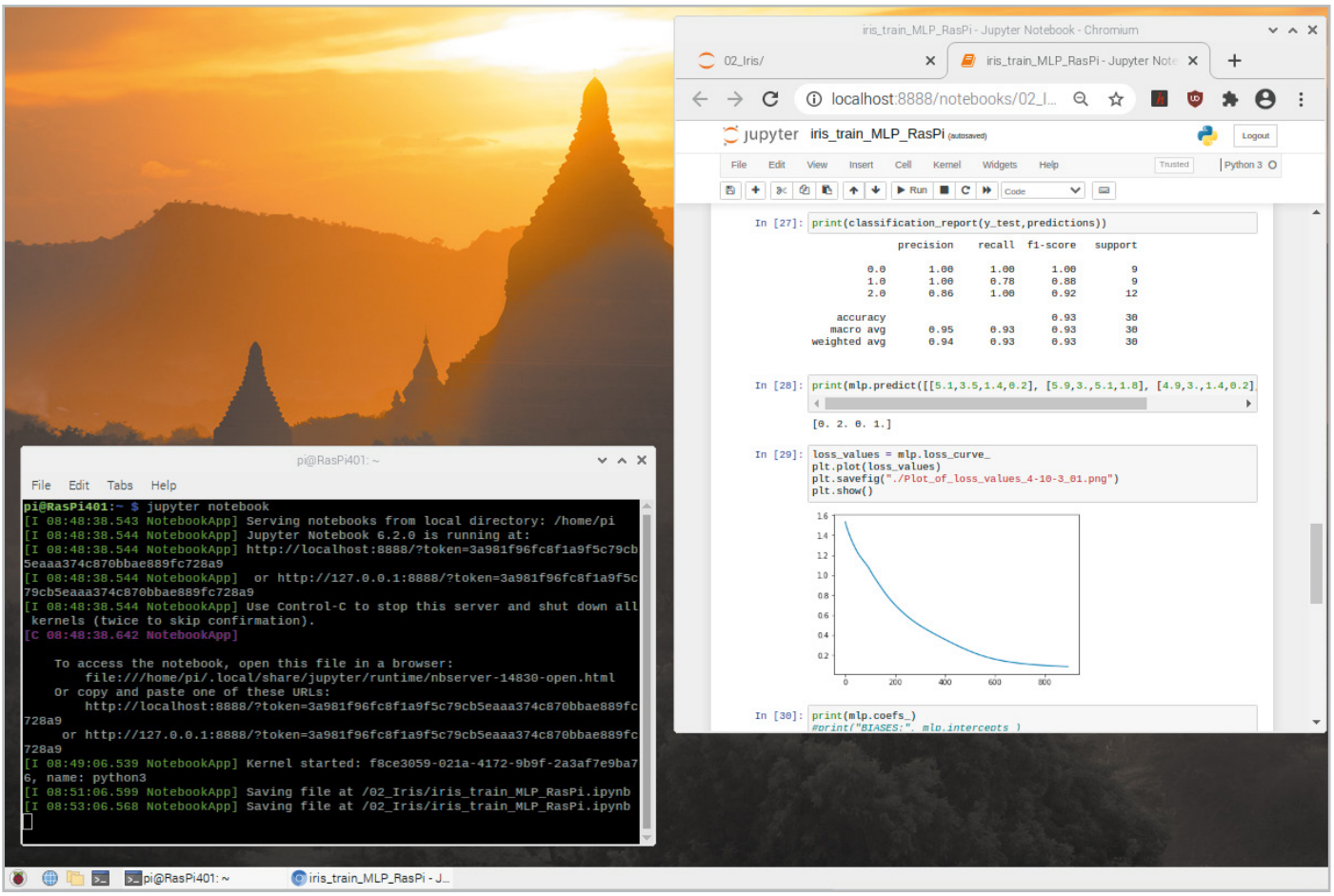


Bild 3: Das Jupyter Notebook auf dem Raspberry Pi

Wir gehen den Code im Folgenden durch.

Zunächst werden die erforderlichen Bibliotheken:

```
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import time
```

für das Anwendungsbeispiel importiert.

In der ersten Zelle des Notebooks werden die Daten selbst wieder direkt aus dem Internet geladen (Zelle 2):

```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'species']
data_train = pd.read_csv(url, names=names)
```

Anschließend werden die einzelnen Spalten mit Namen versehen. Mit

```
print(data_train)
```

kann die geladene Tabelle in Augenschein genommen werden. Alternativ könnten die Daten (iris_D.csv) auch als Tabelle (z. B. via USB-Stick) geladen werden, falls etwa der Raspberry Pi nicht über eine Internetverbindung verfügt. Dann könnte die Umwandlung in ein Array über

```
data_train = pd.read_csv('/home/pi/DATA/IRIS/iris_D.csv')
```

erfolgen.

Die Werte in der letzten Spalte bestehen aus alphanumerischen Bezeichnungen (Kategorienamen). Neuronale Netze können jedoch nur mit rein numerischen Daten arbeiten. Die nächste Aufgabe besteht deshalb darin, diese Kategorien in Zahlenwerte umzuwandeln:

```
data_train.loc[data_train['species']=='Iris-setosa', 'species']=0
data_train.loc[data_train['species']=='Iris-versicolor', 'species']=1
data_train.loc[data_train['species']=='Iris-virginica', 'species']=2
data_train = data_train.apply(pd.to_numeric)
```

Das Ergebnis kann wieder mit print(data_train) überprüft werden und sollte so aussehen:

	sepal-length	sepal-width	petal-length	petal-width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

[150 rows x 5 columns]

Trainings- und Testdaten

Der vollständige Datensatz wird nun in Trainings- und Testdaten aufgeteilt. Die Trainingsdaten werden verwendet, um das Neuronale Netzwerk zu trainieren. Die Testdaten dienen dazu, die Leistung des Neuronalen Netzwerks zu bewerten. Damit kann das Phänomen der „Überanpassung“ vermieden werden. Dieses entsteht, wenn ein Neuronales Netz zu genau auf die Testdaten angepasst wird, bei neuen Daten jedoch schlechtere Ergebnisse liefert, als prinzipiell möglich wäre.

Die Aufteilung der Trainings- und Testaufteilungen erfolgt über:

```
data_train_array = data_train.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(data_train_array[:,:4], data_train_array[:,4], test_size=0.2)
```

Dabei werden 20 Prozent (test_size = 0.2) der Daten für den Testdatensatz abgetrennt. Somit stehen also 150 x 0,2 = 30 Testdatensätze zur Verfügung, die beim Training nicht verwendet werden. Diese kommen später für die unabhängige Evaluierung des Netzes zum Einsatz.

Training und Vorhersagen

Nun kann mit dem Trainieren eines Neuronalen Netzwerks begonnen werden. Dazu sind die folgenden Anweisungen auszuführen:

```
mlp = MLPClassifier(hidden_layer_sizes=(6,), max_iter=1000)
mlp.fit(X_train, y_train)
```

Hier zeigt sich sehr eindrucksvoll, wie mächtig die Bibliotheken wie „Scikit-Learn“ tatsächlich sind. Mit nur zwei Codezeilen lässt sich ein leistungsfähiges Neuronales Netzwerk erstellen, das bereits praxisrelevante Aufgaben übernehmen kann.

Der erste Schritt besteht darin, den MLPClassifier mit zwei Parametern zu initialisieren. Der erste Parameter (hidden_layer_sizes) wird verwendet, um die Größe der verborgenen Schichten festzulegen. Hier wird beispielsweise eine Ebene mit sechs Knoten definiert (s. Bild 2).

Durch Variation dieses Parameters kann die Topologie des Netzes verändert werden. Mit

```
mlp = MLPClassifier(hidden_layer_sizes=(6, 8, 5), max_iter=1000)
```

entstünde bereits ein Netz mit drei verborgenen Schichten, das sechs Neuronen in der ersten, acht in der zweiten und fünf Neuronen in der dritten inneren Schicht aufweist. Die Anzahl der Eingabe- (vier) und Ausgabeneuronen (drei) ergibt sich automatisch aus dem Format des Datensatzes.

Der zweite Parameter im MLPClassifier („max_iter“) gibt die Anzahl der Iterationen an, die das Neuronale Netzwerk maximal ausführen soll. Der Algorithmus arbeitet so lange, bis eine intern vorgegebene Genauigkeit oder diese maximale Anzahl von Iterationen erreicht wird.

Der MLP-Klassifizierer bietet darüber hinaus weitere optionale Parameter. Die wichtigsten davon sind:

activation*{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Aktivierungsfunktion für die verborgenen Schichten, u. a.:
'logistic': logistische Sigmoid-Funktion
'relu', lineare Einheitsfunktion (max(0, x))
solver*{'lbfgs', 'sgd', 'adam'}, default='adam'*
Funktion zur Gewichtsoptimierung.
„adam" bezieht sich auf einen gradientenbasierten Optimierer. Dieser funktioniert bei großen Datensätzen meist am besten. Bei kleinen Datensätzen kann „lbfgs" jedoch schneller konvergieren und eine bessere Leistung erzielen.
verbose: *bool, default=False*
Ist „verbose" auf True gesetzt, wird der Iterationsfortschritt fortlaufend ausgegeben

Standardmäßig wird die Aktivierungsfunktion „relu" mit dem Optimierer „adam" verwendet. Diese Werte können jedoch mithilfe der Aktivierungs- bzw. Solver-Parameter angepasst werden. Die Anweisung

```
mlp = MLPClassifier(hidden_layer_sizes=(6,8,5),
activation='logistic', solver='lbfgs',  max_iter=3000,
verbose=True)
```

definiert also wieder das oben angegebene Netz, jetzt jedoch mit den folgenden Parametern:

Aktivierungsfunktion: logistisch
Solver: lbfgs (also Broyden-Fletcher-Goldfarb-Shanno-Algorithmus (BFGS)
Maximale Iterationen: 3000
verbose=True: Der Iterationsfortschritt wird fortlaufend ausgegeben

In der zweiten Zeile wird die Anpassungsfunktion verwendet, um den Algorithmus mit den im letzten Abschnitt erzeugten Trainingsdaten (X_train und y_train) zu trainieren.

Je nach verwendeten Parametern und definierter Netzstruktur kann das Training zwischen wenigen Sekunden und mehreren Minuten in Anspruch nehmen. Nach Abschluss des Trainings wird die benötigte Laufzeit über

```
print('runtime: {:5.3f}s'.format(end_proc-start_
proc)))
```

ausgegeben.

Die entsprechende Zelle im Jupyter-Skript enthält mehrere Varianten. Diese können durch Entfernen der Kommentarzeichen (#) wunschgemäß aktiviert werden.

Um die jeweils benötigten Trainingszeiten erfassen zu können, wurde über

```
start_proc = time.process_time()
```

und

```
print('runtime: {:5.3f}s'.format(end_proc-start_proc))
```

eine Laufzeiterfassung implementiert. Die typischen Trainingszeiten liegen auf einem Raspberry Pi 4 zwischen zwei und bis zu 100 Sekunden.

Das Netz liefert Ergebnisse

Nun kann das fertig trainierte Netz bereits erste Vorhersagen liefern. Zunächst kann man sich noch die Fehlerquote des Trainings ansehen:

```
print("result training: %5.4f" % mlp.score(X_train, y_train))
```

Interessanter als die Trainingsdaten sind die dem Netz bisher „unbekannten" Testdaten. Die Treffgenauigkeit des Netzes erhält man über:

```
print("result test: %5.4f" % mlp.score(X_test,y_test))
```

Diese sollte typischerweise bei 96,67 Prozent liegen. Das heißt, dass im Testdatensatz mit 30 Blumen nur eine falsch kategorisiert wurde:
1 – 1/30 = 0,9667 = 96,67 %

Die Werte können natürlich je nach verwendeter Netzarchitektur von den oben angegeben Ergebnissen abweichen. Zudem können die Ergebnisse auch über verschiedene Trainingsläufe hinweg variieren, da die Funktion „train_test_split" die Daten zufällig in Trainings- und Testsätze aufteilt, sodass das Netzwerk nicht immer mit denselben Daten trainiert oder getestet wird.

Die vorhergesagten Werte können über

```
predictions = mlp.predict (X_test)
```

abgefragt werden. Mit

```
print("No   Wert   Vorhersage")
for i in range(0,30):
print (i,"   ", y_test[i]," ", predictions[i])
```

erfolgt eine Ausgabe in tabellarischer Form. Mit den oben angegebenen Ergebnissen sollten alle Werte – mit einer Ausnahme – übereinstimmen:

No	Wert	Vorhersage
0	1.0	1.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	2.0	2.0
5	0.0	0.0
6	**0.0**	**2.0**
7	2.0	2.0
8	2.0	2.0
…	…	…

Der sechste Datensatz, in diesem Beispiel eine Iris setosa ('species' = 0) wurde hier fälschlicherweise als Iris virginica ('species' = 2) klassifiziert.

Man kann nun auch einzelne Datensätze auswählen und die Irisart dazu ermitteln lassen:

Für den ersten Eintrag (Datensatz 0):

	sepal-length	sepal-width	petal-length	petal-width	species
0	5.1	3.5	1.4	0.2	0

liefert das Neuronale Netz über

```
print(mlp.predict([[5.1,3.5,1.4,0.2]]))
```

das Ergebnis

0

also Iris setosa – und damit ein korrektes Ergebnis. Auch weitere Datensätze werden korrekt klassifiziert:

```
print(mlp.predict([[5.1,3.5,1.4,0.2], [5.9,3.,5.1,1.8], [4.9,3.,1.4,0.2],
[5.8,2.7,4.1,1.]]))
```

liefert das Ergebnis:

[0. 2. 0. 1.]

Nur ein Wert im Testdatensatz zeigt die einzige Fehlklassifizierung:

```
print(mlp.predict([[5.4,3.9,1.7,0.4]]))
```

Resultat:

2

also eine als Virginica identifizierte Setsosa.

Bewertung des Algorithmus

Abschließend kann man über verschiedene Bewertungsmethoden feststellen, wie gut der gewählte Algorithmus arbeitet. Dafür stehen die folgenden Parameter zur Verfügung:

- Wahrheitsmatrix (confusion_matrix)
- Präzision,
- Recall und f1-Wert

Die folgenden Befehle liefern einen Bewertungsbericht für das gewählte Netz:

```
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

Dieser Code generiert das folgende Ergebnis:
1. Die Wahrheitsmatrix (Confusion matrix):

	Iris setosa	Iris virginica	Iris versicolor
Iris setosa	[[12	0	0]
Iris virginica	[0	11	1]
Iris versicolor	[0	0	6]]

Aus dieser Matrix entnimmt man, dass 12 Iris setosa, 11 Iris virginica und 6 Iris versicolor korrekt erkannt wurden. Lediglich eine Iris virginica wurde fälschlicherweise als Iris versicolor einsortiert. Man erkennt also auch hier wieder, dass von den 30 Pflanzen des Testdatensatzes lediglich eine falsch klassifiziert wurde.

2. Daneben werden außerdem weitere Bewertungskriterien ausgegeben:
precision, recall, f1-score und support

Diese Werte liefern jeweils das Verhältnis von korrekt oder falsch vorhergesagten Werten zu den insgesamt möglichen Vorhersagen. Von besonderer Bedeutung ist hier der f1-Wert (f1-score). Die F1-Bewertung ist ein Maß für die „Treffgenauigkeit“ des Netzwerkes. Ein f1-Wert von 1 bedeutet, eine hohe Treffgenauigkeit, null eine niedrige. Der f1-Wert von 0,97 ist vergleichsweise gut, da mit nur 120 Datensätzen trainiert wurde. Insgesamt sollte die Genauigkeit unter den hier gegebenen Bedingungen immer besser als 95 Prozent sein.

Lernkurven

Bei Schülern oder Studenten, die sich mit dem Lernstoff schwertun, spricht man beschönigend davon, dass sie eine „flache Lernkurve“ haben. In der KI-Forschung stellt die Lernkurve dagegen ein wichtiges Bewertungskriterium dar.

Eine Lernkurve zeigt hier die Beziehung zwischen der Anzahl der Trainingsperioden und dem verbleibenden Vorhersagefehler. Es sollte sich daher eine stetig abfallende Kurve ergeben. Je schneller diese abfällt, desto rascher „lernt“ das Netz. Typischerweise zeigt sich zu Beginn des Trainings meist ein steiler Abfall. Das bedeutet, dass das Netz rasche Lernfortschritte macht. Dann flacht die Kurve ab. Das Netz hat sozusagen „ausgelernt“. Weitere Trainingszyklen liefern dann praktisch keine Verbesserungen mehr und das Training kann abgebrochen werden.

Die in Bild 4 dargestellte Lernkurve zeigt den Trainingsverlauf für das in Bild 2 dargestellte Netz. Es werden fast 2000 Trainingszyklen benötigt, um die gewünschte Fehlerquote von 0,1 zu erreichen.

Nun kann man mit den verschiedenen Netzstrukturen und -parametern experimentieren. So kann man überprüfen, ob die Lernkurve durch Hinzufügen weiterer Schichten verbessert wird. Auch die anderen Parameter wie die Aktivierungsfunktion oder der Solver können variiert werden. Ziele können hierbei ein möglichst schnelles Lernen oder ein möglichst geringer Endfehler sein.

Bild 5 zeigt eine Lernkurve für ein Netz mit drei inneren Schichten. Dieses weist jeweils 4, 10 und 3 Neuronen auf. Mit der Eingabe- und der Ausgabeschicht verfügt das Netz also bereits über insgesamt fünf Schichten. Die Lernkurve ist dementsprechend schon nahezu perfekt ausgebildet und erreicht bereits nach 600 Iterationen das Lernziel.

Fazit und Ausblick

In diesem Beitrag zur Reihe KI-Praxis wurde gezeigt, wie ein Neuronales Netz konstruiert und trainiert wird. Als Praxisbeispiel diente ein Datensatz von Iris-Blumenarten. Nach dem Training des Netzes konnten die verschiedenen Pflanzenarten weitgehend korrekt anhand ihrer Blütenblattdaten klassifiziert werden. Aufgrund des relativ geringen Datenumfangs kann hier auch ein Kleinrechner wie der Raspberry Pi schnelle und gute Ergebnisse erzielen.

Der nächste Beitrag geht einen Schritt weiter. Dort sollen dann bereits reale Bilder klassifiziert werden. Am Beispiel von handgeschriebenen Ziffern wird dargelegt, wie ein KI-System Daten aus der realen Welt verarbeiten kann. Hier kann der Raspberry Pi sogar einen besonderen Vorteil ausspielen. Über die am Kamera-Port anschließbare Pi-Cam können direkt Bilddaten verarbeitet werden. Diese Möglichkeit wird in den späteren Themen wie Objekt- und Gesichtserkennung weiterentwickelt. ELV

Material	Artikel-Nr.
Raspberry Pi 4 Model B, 8 GB RAM	250567
Raspberry Pi 4B, 4 GB Starterset	250983

Bild 4: „Lernkurve“ eines Neuronalen Netzes nach Bild 2 (drei Schichten)

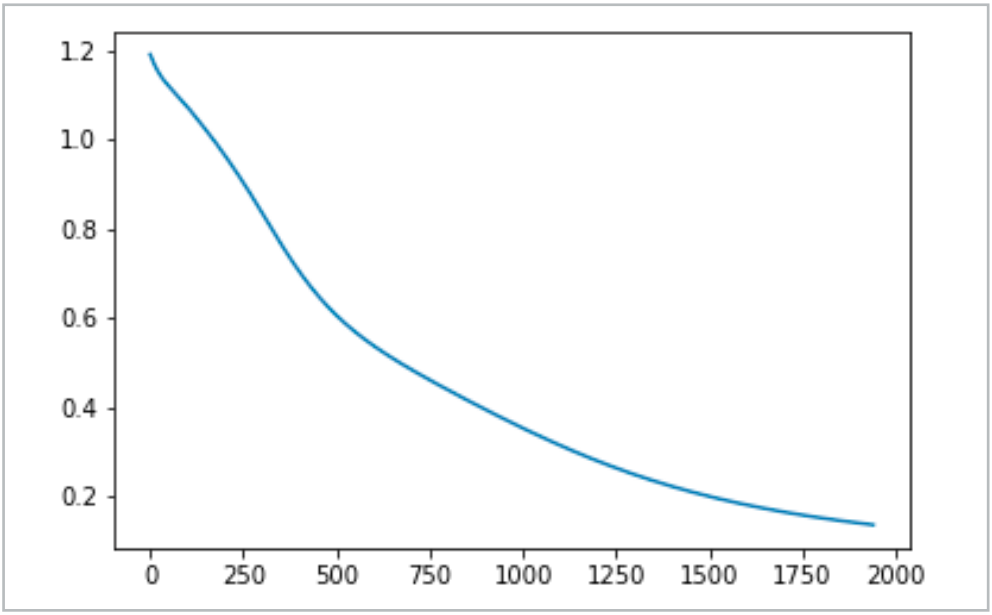
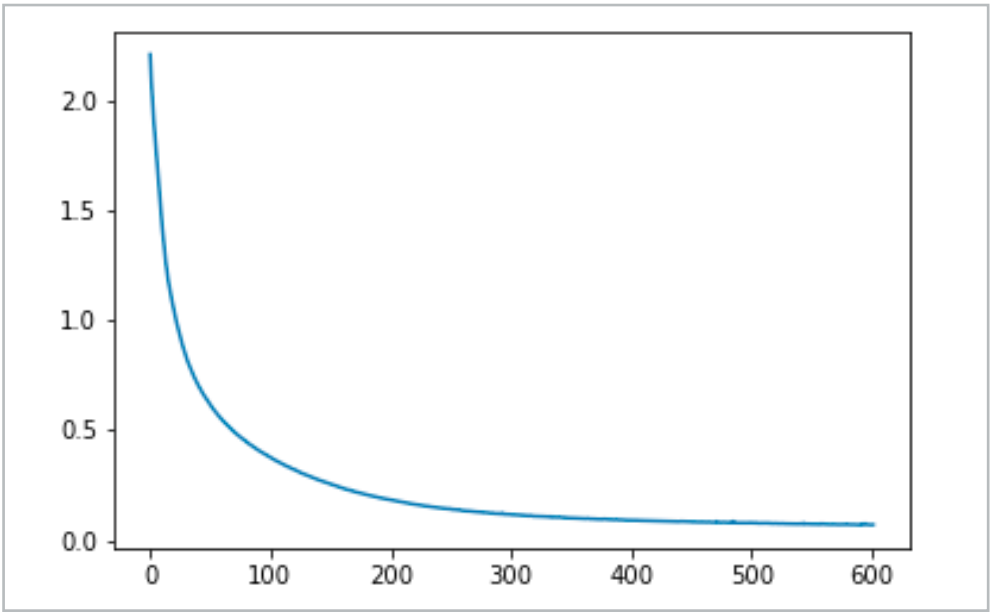


Bild 5: „Lernkurve“ eines Neuronalen Netzes mit insgesamt fünf Schichten



Weitere Infos:

[1] Downloadpaket zu diesem Beitrag: Artikel-Nr. 252174
[2] Filezilla: https://filezilla-project.org/
[3] KI-Praxis I – Einstieg in die Künstliche Intelligenz Teil 1, ELVjournal 3/2021: Artikel-Nr. 252090

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links