

Titel: Schindler, Alexander und Dietz, Alexander: KI im Unterricht mit TensorFlow programmieren

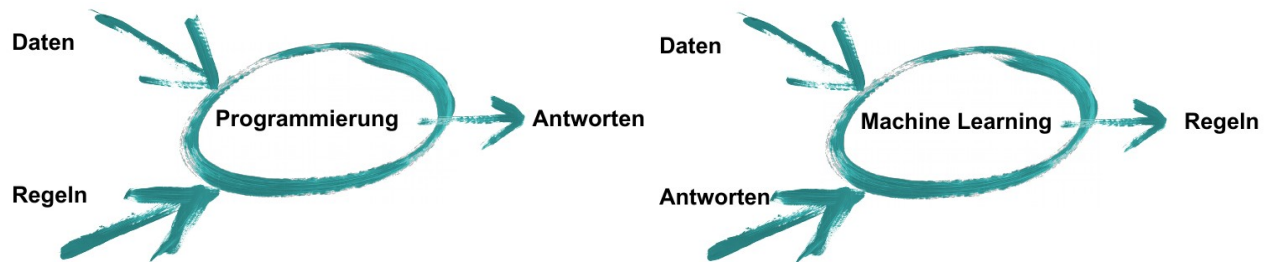


Abb. 1: Klassische Programmierung (links) im Vergleich zu Machine Learning (rechts).

Abstract

TensorFlow bietet als Framework in der KI-Entwicklung einige Unterstützung. Wie die Vorteile dieser Bibliothek im unterrichtlichen Einsatz verwendet werden können, soll der folgende Artikel zeigen. Nach einer kurzen Einführung in die Ideen des Lernens in künstlichen neuronalen Netzwerken (NN) wird die mögliche Umsetzung im Unterricht an zwei Beispielen gezeigt. Das erste Beispiel richtet sich eher an Schüler im Informatikunterricht der Sek I, das zweite an die in Sek II.

TensorFlow ist eine Open Source Bibliothek von Google. Die eigenständige Open Source Deep Learning Bibliothek Keras ist Teil von TensorFlow. Die Bibliothek enthält eine große Auswahl an Trainingsdaten, die für das Lernen von NN genutzt werden können. Es empfiehlt sich, den Lerngruppen eine entsprechend leistungsstarke IDE für Python zur Verfügung zu stellen, wie sie z. B. mit Spyder in Anaconda zu finden ist.

1. Wie ein neuronales Netz arbeitet



Abb. 2. Bekanntes EVA-Prinzip der klassischen Programmierung.

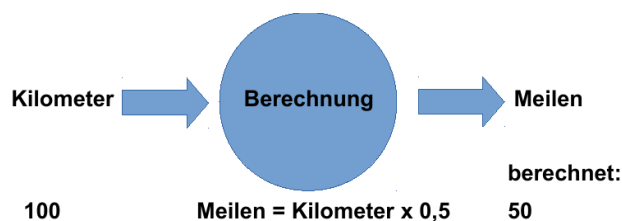


Abb. 3: Lernen im NN, Schritt 1: Umrechnung von Meilen in Kilometer als Schätzung, unter der Annahme, dass der Umrechnungsfaktor nicht bekannt ist. Der Faktor 0,5 wird zufällig gewählt und stellt das Gewicht dar, das zwischen den Neuronen wirkt.

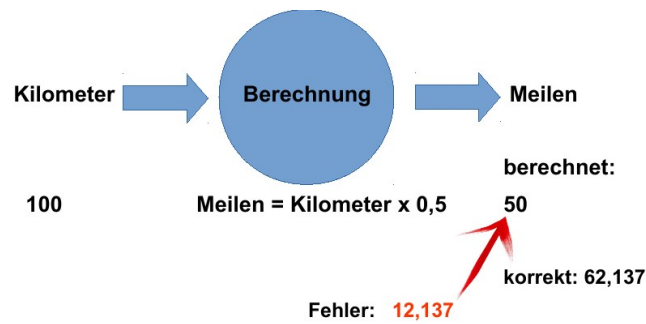


Abb. 4: Lernen im NN, Schritt 2: Die Schätzung 0,5 wird überprüft und der Fehler berechnet.

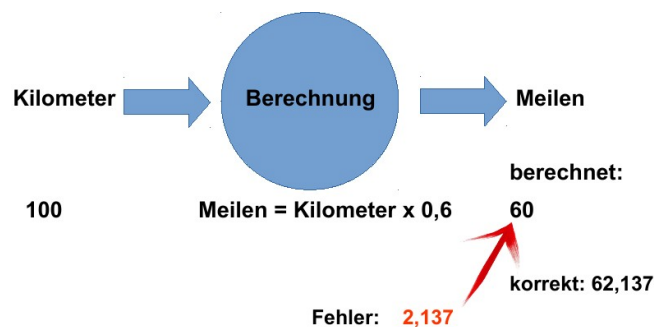


Abb. 5: Lernen im NN, Schritt 3: Aufgrund des in Schritt 2 berechneten Fehlers kann eine verbesserte Schätzung mit 0,6 durchgeführt werden. Schritt 3 wird so oft wiederholt, bis man sich dem Optimum mit einem akzeptablen Fehlerwert angenähert hat.

Die Beispiele in Abb. 3./4. und 5. zeigen in Anlehnung an das EVA-Prinzip (Abb. 2) wie das Lernen in einem NN ablaufen kann, sie sind dem Buch von Tariq Rashid entnommen (siehe Literaturhinweis). Die Beispiele zeigen den Ablauf des *supervised learning* mit nur einem Knoten und der Umrechnung von Kilometern in Meilen mit Hilfe des Trainingspaares *input* = 100 Kilometer und *output* = 62,137 Meilen. Mit jedem Trainingslauf wird das Gewicht so verändert, dass sich das Ergebnis des NN dem *output*, also der optimalen Antwort, annähert. Die Fehlerrate wird bei jedem Lauf verkleinert und je größer die Anzahl der Durchläufe ist (Schritt 2 und 3), umso besser kann das NN Kilometer in Meilen umrechnen.

Üblicherweise werden die Trainingsdaten aus möglichst vielen input-output Paaren gebildet und auch die NN bestehen aus viele Knoten, die in vielen miteinander verbundenen Schichten liegen. Zusätzlich dazu werden noch Aktivierungsfunktionen eingesetzt, so dass die Abbildungen 3./4. und 5. nur das grundsätzliche Prinzip verdeutlichen. Die erhöhte Komplexität realistischer NN und ihrer Trainingsdaten ermöglicht das Lernen auch komplexer Zusammenhänge, wie wir sie z.B. bei Bild- oder Spracherkennung vorfinden.

Überwachte Lernverfahren setzen darauf, aus Daten entsprechende Regeln zu erzeugen, um diese auf zukünftige Daten anzuwenden. Die Regeln werden nicht programmiert, sondern das NN lernt diese Regeln aus den zu den Daten (*input*) passenden Antworten (*output*).

Ergebnisse von NN können die optimale Lösung finden, üblicherweise nähern sie sich der optimalen Lösung aber nur an. Möglicherweise gelangen sie auch nie in die Nähe des Optimums, dann ist das neuronale Netzwerk wertlos und ein neues muss trainiert werden.

2. Hands on Code

Wir werden das Verfahren nun selbst einsetzen. Dazu benutzen wir Python, TensorFlow und Keras, wobei Keras vollständig Teil der TensorFlow Bibliothek ist. TensorFlow benötigt Python 64-Bit ab Version 3.6. Anders als behauptet kann es auch bei höheren Pythonversionen zu Kompatibilitätsproblemen kommen. Es lohnt sich also bei Code, der kompilierbar und ausführbar sein sollte, dies aber nicht ist, die Versionen von TensorFlow und Python zu überprüfen. Hinzu kommt dass TensorFlow noch stark weiterentwickelt wird, aktuell in der Version 2.1 (Frühjahr 2020). Bei Redaktionsschluss gab es TensorFlow 2 nur in einer frühen Version als RC, deswegen basieren die Beispiele hier auf TensorFlow 1.

Wir benötigen:

- Python 64 Bit ab Version 3.6,
- TensorFlow,
- eine passende IDE; empfehlenswert ist z.B. Spyder als Teil von Anaconda.

Hinweis: Die Zahlen in eckigen Klammern beziehen sich auf die Codezeile im Codebeispiel.

3. Rechnen lernen – die Zukunft vorhersagen

3. 1. Unterrichtlicher Einsatz:

Dieses aus wenigen Zeilen Code bestehende Beispiel bietet sich gerade für den Anfängerunterricht in der Sek I an. Selbst mit geringen Pythonkenntnissen können die Schüler hier ein NN selbst ausprobieren. So können sie selbst in einem explorativen Ansatz erproben, welche Änderungen welche Wirkung haben, z.B: Länge der `inputMuster` / `outputMuster`, Anzahl der Epochen, evtl. Anzahl der Knoten. Weiterhin ermitteln die Schüler schnell Ober- und Untergrenzen. Wer erreicht die besten Ergebnisse mit der geringsten Epochenanzahl?

Grundlegende Pythonkenntnisse (z.B. Variablen, Zuweisung) sollten vorhanden sein. Anknüpfungspunkte für den weiterführenden Unterricht könnten sein:

- das Einlesen der drei Muster (Daten: `inputMuster`, Antworten: `outputMuster`, Testdaten: `testMuster`) über eine Nutzereingabe;
- das automatisierte Erstellen des `testMuster` über eine Schleife;
- die Berechnung der Differenz zum erwarteten Ergebnis.

3. 2. Zugrunde liegende Technik

Wir übergeben unserem NN ein `inputMuster` und lassen es mithilfe des `outputMuster` lernen. Mit dem `testMuster` können wir überprüfen, wie gut unser NN trainiert wurde bzw. gelernt hat. Da die Gewichte innerhalb des NN zu Beginn immer zufällig zwischen den Neuronen verteilt sind, unterscheidet sich auch das vorhergesagte Ergebnis bei jedem Durchlauf.

Ein einfaches NN soll nun lernen zu multiplizieren. Das Beispiel mag auf den ersten Blick nicht beeindruckend sein, ist aber gut geeignet, die Konzepte von NN zu veranschaulichen. In unserem Beispiel handelt es sich um eine Multiplikation mit dem Faktor 3. D.h. jedes Element des Arrays `inputMuster` in Zeile [4] wird mit 3 multipliziert so entsteht aus $1 * 3 = 3$, aus $2 * 3 = 6$ usw. Das Ergebnis der Multiplikation steht im Array `outputMuster`. [5] Das neuronale Netzwerk (`model`) wird in [8] erstellt und anschließend mit einem Layer [9] und Parametern [10] versehen. Der Layer besteht hier nur aus einem Knoten [9] und stellt eine Schicht im NN dar. Die genaue Bedeutung der Parameter kann in der TensorFlow-Dokumentation nachgelesen werden. Für die Schüler kann diese Bedeutung problemlos zunächst eine Black Box bleiben.

Das Lernen / Trainieren findet in Zeile [13] statt. Im darauffolgenden Test übergibt man dem

trainierten und compilierten model eine Zahl und lässt sich das Ergebnis des neuronalen Netzwerkes anzeigen.

Das NN versucht einen Wert vorherzusagen, nämlich das zur Testzahl erwartete Ergebnis. Grundsätzlich akzeptiert das neuronale Netzwerk unterschiedliche Datentypen. Hier wurden eindimensionale Listen (inputMuster [4], outputMuster [5] und das testMuster [16]) verwendet.

```
1  import tensorflow as tf
2
3  # Erstellen der Trainingsdaten
4  inputMuster = [1, 2, 4]
5  outputMuster= [3, 6, 12]
6
7  # Aufbau des neuronalen Netzwerkes
8  model = tf.keras.Sequential()
9  model.add(tf.keras.layers.Dense(1, input_shape=[1]))
10 model.compile(optimizer='sgd', loss='mean_squared_error')
11
12 # Trainieren des neuronalen Netzwerkes
13 model.fit(inputMuster, outputMuster, epochs=1000)
14
15 # Testen des neuronalen Netzwerkes mit Testdaten
16 testMuster = [22]
17 print(model.predict(testMuster))
```

Abb 6: Codebeispiel 1.

```
3/3 [=====] - 0s 220us/sample - loss: 7.5569e-05
Epoch 999/1000
3/3 [=====] - 0s 199us/sample - loss: 7.4968e-05
Epoch 1000/1000
3/3 [=====] - 0s 1ms/sample - loss: 7.4373e-05
[[65.88086]]
```

Abb. 7: Ausgabe zu Codebeispiel 1: Für das testMuster wird das Ergebnis 65,88086 berechnet.

3. 3. Mögliche Schüleraufgaben

1. Ändere die Anzahl der Trainingsdaten in den Arrays inputMuster und outputMuster. Beobachte das vorhergesagte Ergebnis. Benenne sinnvolle Ober- und Untergrenzen.
2. Verringere / erhöhe die Epochenanzahl und beobachte das erwartete Ergebnis. Erkläre die Unterschiede.
3. Verändere das inputMuster / outputMuster für verschiedene Berechnungen z.B. Addition, Divisionen usw.
4. Automatisiere das Testen durch viele verschiedene testMuster.
5. Ermittle den Fehlerwert. Benutze dazu Schleifen und zufällige Zahlen.

Man kann am Beispiel erkennen, dass das NN sich dem Ergebnis nur annähert. Dies bietet Anlass

zur Diskussion z.B. wie im NN die Gewichte gesetzt werden.

4. Schriften erkennen Muster in Kategorien einordnen

4. 1. Unterrichtlicher Einsatz

Dieses Beispiel richtet sich an Schüler der SEK II mit Programmiererfahrung, wobei die Erfahrung sich nicht auf Python beziehen muss. Nach einer kurzen Einführung in die grundlegenden Prinzipien des Deep Learning und Python können handschriftliche Zahlen erkannt werden.

4. 2 Zugrunde liegende Technik

In diesem etwas anspruchsvolleren Beispiel wird das NN Zahlen erkennen, die als handschriftliche Graustufenbilder vorliegen. Das NN führt also eine Klassifizierung des Inputmusters durch. Die handschriftlichen Zahlen (70000 Stück, Größe: 28x28 Pixel, Graustufen zwischen 0-weiß und 255-schwarz) selbst sind in TensorFlow enthalten. Die Zahlen werden i. d. R sehr gut erkannt, bei einigen dieser handschriftlichen Zahlen hat man aber selbst als menschlicher Leser Schwierigkeiten bei der Zuordnung. Der MNIST-Datensatz liegt als 2D numpy.ndarray vor.

```
1  import tensorflow as tf
2  import numpy
3  import matplotlib
4
5  # 1. Daten aufbereiten
6  mnist = tf.keras.datasets.mnist
7
8  (trainZiffernBilder, trainZiffernLabels), (testZiffernBilder,
    testZiffernLabels) = mnist.load_data()
9  trainZiffernBilder = trainZiffernBilder / 255.0
10 testZiffernBilder = testZiffernBilder / 255.0
11
12 # 2. Aufbau des NN
13 model = tf.keras.Sequential()
14 model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
15 model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
16 model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
17
18 model.compile(optimizer='adam',
19               loss='sparse_categorical_crossentropy',
20               metrics=['accuracy'])
21
22 # 3. NN trainieren
23 model.fit(trainZiffernBilder, trainZiffernLabels, epochs=5)
24
25 # 4. NN prüfen
26 verlust, genauigkeit = model.evaluate(testZiffernBilder,
    testZiffernLabels)
27 print('Verlust: ', verlust, 'Genauigkeit: ', genauigkeit)
28
29 # 5. Testen des NN mit Testdaten
30 gesuchteZahlIndex = 0
```

```

31 erkennungsRaten = model.predict (testZiffernBilder
    [gesuchteZahlIndex:gesuchteZahlIndex+1])
32 flattendEr = erkennungsRaten.flatten()
33 flattendTZLabels = testZiffernLabels.flatten()
34
35 # 6. Test: Ausgabe der gesuchten Zahl als Bild
36 imageArray = numpy.asfarray (testZiffernBilder
    [gesuchteZahlIndex:gesuchteZahlIndex+1]).reshape((28,28))
37 matplotlib.pyplot.imshow(imageArray, cmap='Greys',
    interpolation='None')
38 matplotlib.pyplot.show()
39 print ("gesuchte Zahl: ", flattendTZLabels[gesuchteZahlIndex])
40
41 # Ausgabe der Erkennungsraten für die Zahlen 0..9
42 counter = 0
43 while counter < 10:
44     readAbleErkennung = flattendEr[counter] * 10000
45     readAbleErkennung = readAbleErkennung.astype(int)
46     readAbleErkennung = readAbleErkennung / 10000
47     print (" Zahl:", counter, " Erkennungsrate:",
        readAbleErkennung)
48     counter = counter + 1

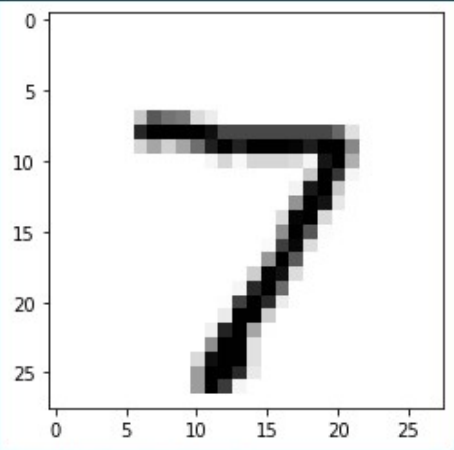
```

Abb. 8.: Codebeispiel 2.

```

Epoch 1/5
60000/60000 [=====] - 2s 33us/sample - loss: 0.2571 - acc: 0.9269
Epoch 2/5
60000/60000 [=====] - 2s 35us/sample - loss: 0.1135 - acc: 0.9663
Epoch 3/5
60000/60000 [=====] - 2s 35us/sample - loss: 0.0774 - acc: 0.9767
Epoch 4/5
60000/60000 [=====] - 2s 33us/sample - loss: 0.0584 - acc: 0.9816
Epoch 5/5
60000/60000 [=====] - 2s 32us/sample - loss: 0.0448 - acc: 0.9860
10000/10000 [=====] - 0s 23us/sample - loss: 0.0804 - acc: 0.9752
Verlust: 0.08036140193331522 Genauigkeit: 0.9752

```



```

gesuchte Zahl: 7
Zahl: 0 Erkennungsrate: 0.0
Zahl: 1 Erkennungsrate: 0.0
Zahl: 2 Erkennungsrate: 0.0
Zahl: 3 Erkennungsrate: 0.0014
Zahl: 4 Erkennungsrate: 0.0
Zahl: 5 Erkennungsrate: 0.0
Zahl: 6 Erkennungsrate: 0.0
Zahl: 7 Erkennungsrate: 0.9984
Zahl: 8 Erkennungsrate: 0.0
Zahl: 9 Erkennungsrate: 0.0

```

Abb. 9: Ausgabe zu Codebeispiel 2.

4.3 Erläuterung des Quelltextes und der Ausgabe

Die folgenden detaillierten Erläuterungen müssen nicht notwendigerweise alle im Unterricht thematisiert werden, bieten aber Möglichkeiten zur Differenzierung.

[8] Laden der MNIST-Datensammlung in vier numpy-Arrays: `trainZiffernBilder` (entsprechen den Daten) und `trainZiffernLabel` (entsprechend den Antworten) bilden 60000 Trainingsdaten, `testZiffernBilder` und `testZiffernLabel` bilden 10000 Testdaten. Die Bilder sind als 2D numpy-Array codiert, die Label sind ein Array mit den Ziffern von 0 bis 9.

[9]/[10] Alle Grauwerte werden durch ihren Maximalwert (255) dividiert, um sie in den Wertebereich zwischen 0 und 1 zu skalieren. Dieser Wertebereich ist für die Aktivierungsfunktionen sinnvoll.

[13-16] Das NN (`model`) wird erstellt. Es besteht aus einem Stapel von Layern. Der erste Layer besteht aus 128 Knoten und der Aktivierungsfunktion `relu`. Die Ausgabeschicht wird aus 10 Knoten und der Aktivierungsfunktion `softmax` gebildet. Das Modell hat somit für jede der 10

möglichen Ziffern einen Knoten in der Outputschicht. Mit Hilfe der Aktivierungsfunktionen `softmax` wird die Wahrscheinlichkeit für die 10 Zielklassen (Ziffern 0 bis 9) festgelegt. Die Summe dieser Wahrscheinlichkeiten ist 1. `input_shape=(28,28)` legt die Eingabeform entsprechend der Bildgröße fest.

[18-20] Beim Kompilieren des NN kann die Art der Verlustfunktion (`loss`), die Art des Optimierers (`optimizer`) und die Art der Metrik-Funktion (`metrics`) gewählt werden. Als Verlustfunktion wurde hier die Kreuzentropie gewählt, die den Unterschied zwischen den vom NN vorhergesagten Wahrscheinlichkeiten und den wirklichen Werten für die einzelnen der 10 Ziffern ermittelt. Diese Werte werden an den Optimierer übergeben. Ein guter Kandidat für einen Optimierer ist oft `adam`, denn er verändert die Lernrate des Modells über den Trainingsverlauf automatisch. Als Metrik, wie gut das trainierte Modell ist, soll die Genauigkeit der Erkennung ausgegeben werden (`accuracy`).

[23] Nachdem das Modell kompiliert ist, muss es nun noch trainiert werden. Da wir sehr viele Trainingsdaten besitzen (60000 Bilder) benötigen wir für eine gute Erkennungsrate nur sehr wenige Durchläufe (`epochs=5`). In diesen fünf Durchläufen versucht das Modell, den wirklichen Werten so nahe wie möglich zu kommen. Dabei werden die vom NN gelieferten Antworten mit den erwarteten Antworten aus `testZiffernLabel` verglichen und die Gewichte zwischen den Neuronen immer wieder angepasst. Während des Trainings werden zwei Werte angezeigt: der Wert der Verlustfunktion (`loss`) und die Korrektklassifizierungsrate (`acc`) für die Trainingsdaten. Die Trainingsdaten werden mit einer Rate von 98,6% erkannt.

[26-27] Überprüfung des NN mit den 10000 Testdaten: Die Klassifizierungsrate für den Testdatensatz beträgt 97.52 %.

[31-33] Das NN soll nun das erste Ziffernbild aus dem Testdatensatz, eine 7, erkennen.

[36-39] Das Ziffernbild mit seinem Label wird ausgegeben. Dies dient nur für unsere eigene Übersicht.

[41-48] Alle Erkennungsraten werden angezeigt, so dass man die Klassifizierungsrate für jede mögliche Ziffer (0-9) nachlesen kann. Unsere Testziffer 7 wird mit einer Rate von 99,84% erkannt.

4. 4 Mögliche Schüleraufgaben

1. Beschreibe das Ergebnis des NN.
2. Erkläre, warum das Ergebnis des NN für das Testmuster bei Aufg. 1 nicht exakt mit dem des Antwortmusters übereinstimmt.
3. Verändere die Anzahl der Epochen. Beschreibe / erkläre die Auswirkungen. Ermittle dazu sinnvolle Ober-/ Untergrenzen.
4. Verändere die Anzahl der Knoten in Zeile 16. Beschreibe die Auswirkungen auf das Ergebnis und erkläre die Veränderungen.
5. Ermittle, wo eine sinnvolle Unter- / Obergrenze für die Knotenanzahl liegt.
6. Entscheide anhand der Erkennungsrate, welche Ziffer erkannt wurde.
7. Überlege, welche Auswirkungen eine zu große Knotenanzahl bei kleinen Datensätzen haben kann.
8. Das Ergebnis soll so aufbereitet werden, dass nur noch 0 und 1 darin vorkommen.
9. Erstelle eigene Bilder in der Größe 28x28 Pixel und versiehe sie mit verschiedenen handschriftliche Zahlen.
10. Fotografiere Zahlen und forme sie auf eine Bildgröße 28x28 Pixel um.
11. Drehe Deine Bilder mit Python um einige Grad.
12. Beobachte bei den selbst erstellten und veränderten Bildern (Aufg 9.-11.) die

Erkennungsrate. Wo liegen Grenzen?
13. Erkennst Du Zahlen besser als Deine KI?

5. Fazit keine Angst vor KI?

Das englische Wort *intelligence* ist nicht gleichzusetzen mit dem deutschen Wort Intelligenz, vielmehr wird das englische *artificial intelligence* besser beschrieben als künstliche Informationsverarbeitung. Genau dafür bietet TensorFlow einen Rahmen.

Wer neuronale Netze einmal selbst trainiert hat, kann deren Risiken, Probleme und Chancen und damit auch mögliche gesellschaftliche Entwicklungen besser einschätzen. Die Schüler entdecken mit den vorliegenden Beispielen dass KI-Systeme häufig gar nicht so intelligent sind, wie vielfach berichtet wird. Aber sie können leistungsfähige Systeme zur spezifischen Problemlösung darstellen. Bis zur technologischen Singularität, also der KI die intelligenter ist als der Mensch ist, ist es also noch ein ganzes Stück, vielleicht ist sie sogar grundsätzlich unmöglich.

Glossar

- accuracy: Anteil der richtigen Vorhersagen.
- Aktivierungsfunktion: Art und Weise in der ein Neuron aufgrund bestimmter Eingaben feuert.
- Epoche: (Trainingslauf) wiederholtes Trainieren derselben Trainingsmuster.
- Gewichte: stärken (oder schwächen) die Verbindungen zwischen Knoten. Die Änderung der Gewichte bildet das Lernen des NN ab.
- Layer: eine Ebene / Schicht des Neuronalen Netzwerkes besteht aus einem oder mehreren Neuronen.
- Lernrate: Höhe der Anpassung der Gewichte in einer Epoche.
- Neuron (auch Knoten): einzelne Verarbeitungseinheit eines Layers
- neuronales Netzwerk (NN): besteht aus mehreren durch Gewichte miteinander verbundenen Layern. In den Codebeispielen treffend als `model` bezeichnet.
- loss value: Abstand zum globalen Minimum
- overfitting: Netzwerk lernt alle Trainingsmuster auswendig.
- overshooting: bei zu hoher Lernrate kann das Fehlerminimum nicht erreicht werden.

Literaturhinweis: Wie man ein NN ohne tensorflow programmiert

- RASHID, Tariq [2017]: Neuronale Netze selbst programmieren, ein verständlicher Einstieg mit Python. Heidelberg

Quellen:

- www.tensorflow.org/api_docs/python/

Weitergehende Datensätze (abgerufen am 15.2.2020)

- MNIST DB, Handschriften: yann.lecun.com/exdb/mnist
- CIFAR-10 / CIFAR-100, 80 Millionen kategorisierte Bilder:
<https://www.cs.toronto.edu/~kriz/cifar.html>

- Kategorisierte Bilder: [www. image-net.org](http://www.image-net.org),
<https://storage.googleapis.com/openimages/web/index.html>
- Blumenbilder: http://download.tensorflow.org/example_images/flower_photos.tgz
- Übersicht über verschieden Datensätze für machine learning:
https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research

Autoren

Alexander Schindler, Hannah-Arendt-Gymnasium Berlin, alexander_schindler@gmx.de

Alexander Dietz, Humboldt-Gymnasium Berlin-Tegel, a.dietz@humboldtschule-berlin.eu