Einführung in das maschinelle Lernen

Da wir `Python` lernen, ist es sicherlich auch wichtig, über maschinelles Lernen zu sprechen. Denn viele seiner Bibliotheken sind in Python geschrieben. Fangen wir damit an, sie zu installieren und ein bisschen damit zu spielen.

TensorFlow

Installiere es.

```
$ pip install tensorflow
ERROR: Es konnte keine Version gefunden werden, die die Anforderung tensorflow erfüllt
ERROR: Keine passende Distribution für tensorflow gefunden

$ type python
python ist ein Alias für `/usr/local/Cellar/python@3.9/3.9.1_6/bin/python3'
```

Allerdings unterstützt `Tensorflow 2` nur `Python 3.5–3.8`. Wir verwenden jedoch `3.9`.

```
 % type python3
python3 ist /usr/bin/python3
% python3 -V
Python 3.8.2
```

Beachte, dass die `python3`-Version auf meinem System `3.8.2` ist. Wo wird das entsprechende `pip` für diese Python-Version installiert?

```
% python3 -m pip -V
pip 21.0.1 aus /Users/lzw/Library/Python/3.8/lib/python/site-packages/pip (python 3.8)
```

Der entsprechende `pip` ist hier. Dann werde ich die `.zprofile`-Datei ändern. Kürzlich habe ich meine `Shell` geändert. `.zprofile` entspricht der vorherigen `.bash_profile`. Füge eine Zeile hinzu.

```
alias pip3=/Users/lzw/Library/Python/3.8/bin/pip3
```

Hinweis: Der obige Code ist ein Shell-Alias, der den Befehl `pip3` auf einen spezifischen Pfad verweist. Da es sich um einen Pfad und einen Befehl handelt, wird dieser nicht übersetzt.

Auf diese Weise verwenden wir `python3` und `pip3`, um mit `Tensorflow` zu arbeiten.

```
% pip3 install tensorflow
...
Erfolgreich installiert: absl-py-0.12.0 astunparse-1.6.3 cachetools-4.2.1 certifi-2020.12.5 chardet-4.0
```

Viele Bibliotheken installiert. Ein Beispiel von der offiziellen Website verwendet.

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Dieser Code lädt die MNIST-Daten und normalisiert die Pixelwerte der Bilder, indem sie durch 255.0 geteilt werden. Dadurch werden die Werte auf einen Bereich zwischen 0 und 1 skaliert.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

predictions = model(x_train[:1]).numpy() print(predictions)

Führe es aus.

```
$ /usr/bin/python3 tf.py
Daten werden heruntergeladen von https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 10s 1us/step
[[ 0.15477428 -0.3877643   0.0994779   0.07474922 -0.26219758 -0.03550266
   0.32226565 -0.37141111  0.10925996 -0.0115255 ]]
```

Es ist ersichtlich, dass der Datensatz heruntergeladen wurde, und anschließend wurden die Ergebnisse ausgegeben.

Als Nächstes betrachten wir ein Beispiel zur Bildklassifizierung.

```
# TensorFlow und tf.keras
import tensorflow as tf
```

Hilfsbibliotheken

import numpy as np import matplotlib.pyplot as plt

```
print(tf.__version__)
```

Fehlermeldung.

```
ModuleNotFoundError: No module named 'matplotlib'
```

Übersetzung:

```
ModuleNotFoundError: Kein Modul namens 'matplotlib' gefunden
```

Installiere es.

```
% pip3 install matplotlib
```

Korrekt.

```
$ /usr/bin/python3 image.py
2.4.1
```

Beispielcode zum Kopieren und Einfügen.

```
# TensorFlow und tf.keras
import tensorflow as tf
```

Hilfsbibliotheken

import numpy as np import matplotlib.pyplot as plt

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
print(train_images.shape)
print(len(train_labels))
```

□□□□□□□□□□□□□ train_images□train_labels□test_images□test_labels□□□□□□□□□□□□□□□□□□□□□□□□□□

```
(60000, 28, 28)
60000
```

Dann versuchen wir, das Bild auszudrucken.

```
print(train_images[0])
```

Schauen wir uns die Ergebnisse an.

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0
    0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
   54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134
  144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178
  107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216
  216 163 127 121 122 146 141  88 172  66]]
  ....
```

Hier ist ein Auszug aus den Ergebnissen.

```
print(len(train_images[0][0]))
```

(Der Code bleibt auf Englisch, da es sich um eine Programmiersprache handelt und keine Übersetzung erforderlich ist.)

Ausgabe von `28`. Es ist also klar, dass dies eine Matrix mit einer Breite von 28 ist. Fahren wir mit dem Drucken fort.

```
    print(len(train_images[0][0][0])
TypeError: Objekt vom Typ 'numpy.uint8' hat keine len()
```

Es ist also ganz klar. Jedes Bild ist ein Array der Größe `28*28*3`. Die letzte Dimension des Arrays speichert die RGB-Werte. Allerdings könnte sich herausstellen, dass unsere Annahme falsch ist.

```
print(train_images[0][1][20])
```

```
0
```

```
print(train_images[0][1])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Jede Abbildung ist ein 28*28-Array. Nach einigem Herumprobieren haben wir endlich das Geheimnis gelüftet.

Schauen wir uns zunächst die Ausgabegrafik an.

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

(Der Code bleibt unverändert, da es sich um eine Programmiersprache handelt und keine Übersetzung erforderlich ist.)

Siehst du die Farbleiste rechts? `0` bis `250`. Ursprünglich war dies ein Farbverlauf zwischen zwei Farben. Aber woher weiß es, welche zwei Farben es sind? Wo haben wir das angegeben?

Dann drucken wir auch das zweite Bild aus.

```
plt.imshow(train_images[1])
```

(Der Code bleibt unverändert, da es sich um eine Programmiersprache handelt und keine Übersetzung benötigt wird.)

Sehr interessant. Ist das die Standardeinstellung der `pyplot`-Abhängigkeitsbibliothek? Ich werde den Code, der auf der offiziellen Website bereitgestellt wird, weiter ausführen.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Beachten Sie, dass hier die Bilder zusammen mit ihren Kategorien angezeigt werden. Endlich haben wir den `cmp`-Parameter verstanden. Wenn bei `cmp` nichts angegeben wird, wird es definitiv die Farbe sein, die wir gerade hatten. Tatsächlich.

```
    plt.imshow(train_images[i])
```

Jetzt suchen wir nach `pyplot cmap`. Wir haben einige Informationen gefunden.

```
    plt.imshow(train_images[i], cmap=plt.cm.PiYG)
```

Hier ist die geänderte Version des Codes:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)    ## Geändert von 2,5 auf 5,5
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.Blues)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Die Änderung besteht darin, dass die `subplot`-Anordnung von `2,5` auf `5,5` geändert wurde, um 25 Bilder in einem 5x5-Raster anzuzeigen.

Es kam jedoch zu einem Fehler.

```
ValueError: num muss im Bereich 1 <= num <= 10 liegen, nicht 11
```

Was bedeutet das? Was genau bedeutet das vorherige `5,5,i+1`? Warum funktioniert es nicht, wenn man es in `2` ändert? Obwohl wir intuitiv verstehen, dass es wahrscheinlich 5 Zeilen und 5 Spalten bedeutet. Aber warum wird dieser Fehler gemeldet? Wie wird `11` berechnet? Was bedeutet `num`? Was bedeutet `10`? Beachten Sie, dass `2*5=10`. Vielleicht tritt der Fehler auf, wenn `i=11` ist. Als wir es in `for i in range(10):` geändert haben, erhielten wir das folgende Ergebnis.

Schauen wir uns die Dokumentation kurz an und erfahren, dass `subplot(nrows, ncols, index, **kwargs)` verwendet wird. Nun, bis hierher ist es uns klar.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    # plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.Blues)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Beachten Sie, dass `0 25` als `xticks` bezeichnet wird. Wenn wir diesen Rahmen vergrößern oder verkleinern, wird dies unterschiedlich dargestellt.

plot_scale

Beachten Sie, dass beim Zoomen und Verkleinern des Rahmens die `xticks` und `xlabels` unterschiedlich angezeigt werden.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Dieser Code kompiliert ein Modell in TensorFlow/Keras. Hier ist die deutsche Übersetzung der Parameter:

- `optimizer='adam'`: Der Optimierer ist auf ‘adam’ gesetzt, einen beliebten Algorithmus für das Training von neuronalen Netzen.
- `loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`: Die Verlustfunktion ist die Sparse Categorical Crossentropy, die häufig für Klassifizierungsprobleme verwendet wird, bei denen die Labels als ganze Zahlen vorliegen. Der Parameter `from_logits=True` gibt an, dass die Ausgaben des Modells Logits sind (d.h., sie wurden nicht durch eine Softmax-Funktion normalisiert).
- `metrics=['accuracy']`: Als Metrik wird die Genauigkeit (Accuracy) verwendet, um die Leistung des Modells zu bewerten.

```
model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)

print('\nTestgenauigkeit:', test_acc)
```

Beachten Sie, wie hier das Modell definiert wird, indem die Klasse `Sequential` verwendet wird. Achten Sie auf diese Parameter: `28,28`, `128`, `relu`, `10`. Beachten Sie, dass `compile` und `fit` benötigt werden. `fit` bedeutet Anpassung. Beachten Sie, dass `28,28` die Größe des Bildes ist.

```
Epoche 1/10
1875/1875 [==============================] - 2s 928us/Schritt - Verlust: 0.6331 - Genauigkeit: 0.7769
Epoche 2/10
1875/1875 [==============================] - 2s 961us/Schritt - Verlust: 0.3860 - Genauigkeit: 0.8615
Epoche 3/10
1875/1875 [==============================] - 2s 930us/Schritt - Verlust: 0.3395 - Genauigkeit: 0.8755
Epoche 4/10
1875/1875 [==============================] - 2s 1ms/Schritt - Verlust: 0.3071 - Genauigkeit: 0.8890
Epoche 5/10
```

```
1875/1875 [==============================] - 2s 1ms/Schritt - Verlust: 0.2964 - Genauigkeit: 0.8927
Epoche 6/10
1875/1875 [==============================] - 2s 985us/Schritt - Verlust: 0.2764 - Genauigkeit: 0.8955
Epoche 7/10
1875/1875 [==============================] - 2s 961us/Schritt - Verlust: 0.2653 - Genauigkeit: 0.8996
Epoche 8/10
1875/1875 [==============================] - 2s 1ms/Schritt - Verlust: 0.2549 - Genauigkeit: 0.9052
Epoche 9/10
1875/1875 [==============================] - 2s 1ms/Schritt - Verlust: 0.2416 - Genauigkeit: 0.9090
Epoche 10/10
1875/1875 [==============================] - 2s 1ms/Schritt - Verlust: 0.2372 - Genauigkeit: 0.9086
313/313 - 0s - Verlust: 0.3422 - Genauigkeit: 0.8798
```

Testgenauigkeit: 0.879800021648407

Das Modell wurde trainiert. Lass uns die Parameter anpassen.

```shell
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),    # 128 -> 28
    tf.keras.layers.Dense(10)
])
```

Ändern Sie den ersten Parameter von `Dense`.

```
Epoche 1/10
1875/1875 [==============================] - 2s 714us/Schritt - Verlust: 6.9774 - Genauigkeit: 0.3294
Epoche 2/10
1875/1875 [==============================] - 1s 715us/Schritt - Verlust: 1.3038 - Genauigkeit: 0.4831
Epoche 3/10
1875/1875 [==============================] - 1s 747us/Schritt - Verlust: 1.0160 - Genauigkeit: 0.6197
Epoche 4/10
1875/1875 [==============================] - 1s 800us/Schritt - Verlust: 0.7963 - Genauigkeit: 0.6939
Epoche 5/10
1875/1875 [==============================] - 2s 893us/Schritt - Verlust: 0.7006 - Genauigkeit: 0.7183
Epoche 6/10
```

```
1875/1875 [==============================] - 1s 747us/Schritt - Verlust: 0.6675 - Genauigkeit: 0.7299
Epoche 7/10
1875/1875 [==============================] - 1s 694us/Schritt - Verlust: 0.6681 - Genauigkeit: 0.7330
Epoche 8/10
1875/1875 [==============================] - 1s 702us/Schritt - Verlust: 0.6675 - Genauigkeit: 0.7356
Epoche 9/10
1875/1875 [==============================] - 1s 778us/Schritt - Verlust: 0.6508 - Genauigkeit: 0.7363
Epoche 10/10
1875/1875 [==============================] - 1s 732us/Schritt - Verlust: 0.6532 - Genauigkeit: 0.7350
313/313 - 0s - Verlust: 0.6816 - Genauigkeit: 0.7230
```

Testgenauigkeit: 0.7229999899864197

```
Beachten Sie, dass sich die `Test accuracy` vor und nach der Änderung unterscheidet. `Epoch` ist ein Te
```

```python
print(train_labels)
[9 0 0 ... 3 0 5]
print(len(train_labels))
60000
```

Das bedeutet, dass diese Klassen mit `0 bis 9` dargestellt werden. Zufälligerweise hat `class_names` auch genau 10 Einträge.

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Lass uns noch ein paar Änderungen vornehmen.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(5)   # 10 -> 5
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Übersetzung:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In diesem Code wird das Modell mit dem Adam-Optimierer kompiliert. Als Verlustfunktion wird `SparseCategoricalCrossentropy` verwendet, wobei `from_logits=True` angegeben ist, da die Modellausgaben Logits sind. Als Metrik wird die Genauigkeit (`accuracy`) verwendet, um die Leistung des Modells zu bewerten.

```
model.fit(train_images, train_labels, epochs=10)
```

Es ist ein Fehler aufgetreten.

```
tensorflow.python.framework.errors_impl.InvalidArgumentError:  Ein Label-Wert von 9 wurde empfangen, der
         [[node sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEn
```

Funktionsaufrufstapel: train_function

```
Ändern Sie den Parameter der dritten Schicht `Dense` in `Sequential` auf `15`. Der Unterschied im Ergebn

```python
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(15)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Dieser Code kompiliert ein Modell in TensorFlow/Keras. Der Optimierer ist auf ‘adam’ gesetzt, die Verlustfunktion ist `SparseCategoricalCrossentropy` (mit `from_logits=True`), und die Metrik, die während des Trainings überwacht wird, ist die Genauigkeit (`accuracy`).

```
model.fit(train_images, train_labels, epochs=15)  # 10 -> 15
```
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)

print('\nTestgenauigkeit:', test_acc)
```

```
Epoch 1/15
1875/1875 [==============================] - 2s 892us/step - loss: 6.5778 - accuracy: 0.3771
Epoch 2/15
1875/1875 [==============================] - 2s 872us/step - loss: 1.3121 - accuracy: 0.4910
Epoch 3/15
1875/1875 [==============================] - 2s 909us/step - loss: 1.0900 - accuracy: 0.5389
Epoch 4/15
1875/1875 [==============================] - 1s 730us/step - loss: 1.0422 - accuracy: 0.5577
Epoch 5/15
1875/1875 [==============================] - 1s 709us/step - loss: 0.9529 - accuracy: 0.5952
Epoch 6/15
1875/1875 [==============================] - 1s 714us/step - loss: 0.9888 - accuracy: 0.5950
Epoch 7/15
1875/1875 [==============================] - 1s 767us/step - loss: 0.8678 - accuracy: 0.6355
Epoch 8/15
1875/1875 [==============================] - 1s 715us/step - loss: 0.8247 - accuracy: 0.6611
Epoch 9/15
1875/1875 [==============================] - 1s 721us/step - loss: 0.8011 - accuracy: 0.6626
Epoch 10/15
1875/1875 [==============================] - 1s 711us/step - loss: 0.8024 - accuracy: 0.6622
Epoch 11/15
1875/1875 [==============================] - 1s 781us/step - loss: 0.7777 - accuracy: 0.6696
Epoch 12/15
1875/1875 [==============================] - 1s 724us/step - loss: 0.7764 - accuracy: 0.6728
Epoch 13/15
1875/1875 [==============================] - 1s 731us/step - loss: 0.7688 - accuracy: 0.6767
Epoch 14/15
1875/1875 [==============================] - 1s 715us/step - loss: 0.7592 - accuracy: 0.6793
Epoch 15/15
1875/1875 [==============================] - 1s 786us/step - loss: 0.7526 - accuracy: 0.6792
313/313 - 0s - loss: 0.8555 - accuracy: 0.6418
```

Testgenauigkeit: 0.6417999863624573

Beachte, dass die Änderung auf 15 keinen großen Unterschied macht. `tf.keras.layers.Dense(88, activatio

```python
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die international einheitlich ist.

Als nächstes folgt die Vorhersage. Beachten Sie, dass `Sequential` dasselbe ist wie oben. Beachten Sie die Parameter `model` und `tf.keras.layers.Softmax()`.

```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_images)
```

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird.

```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted_label = np.argmax(predictions_array)
if predicted_label == true_label:
    color = 'blue'
else:
    color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)
```

Übersetzung:

```
plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                     100*np.max(predictions_array),
                                     class_names[true_label]),
                                     color=color)
```

In diesem Code wird die x-Achsenbeschriftung eines Diagramms festgelegt. Die Beschriftung zeigt den vorhergesagten Klassenname, die Wahrscheinlichkeit in Prozent und den tatsächlichen Klassenname an. Die Farbe der Beschriftung wird durch die Variable `color` bestimmt.

```
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

thisplot[predicted_label].set_color('rot')
thisplot[true_label].set_color('blau')

i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

Dies zeigt, dass das Bild mit einer Wahrscheinlichkeit von 99% ein `Ankle boot` ist. Beachten Sie, dass `plot_image` das linke Bild anzeigt und `plot_value_array` das rechte Diagramm ausgibt.

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
```

```
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

Beachten Sie, dass hier nur weitere Testergebnisse angezeigt werden. Daher ist uns der Ablauf im Großen und Ganzen klar. Wir wissen jedoch noch nicht, wie die Berechnungen im Hintergrund durchgeführt werden. Aber wir wissen, wie man sie verwendet. Im Hintergrund steckt die Infinitesimalrechnung. Wie kann man die Infinitesimalrechnung verstehen?

Angenommen, es gibt eine Zahl zwischen 1 und 100, die du erraten sollst. Jedes Mal, wenn du eine Zahl rätst, sage ich dir, ob sie zu klein oder zu groß ist. Du rätst 50. Ich sage: “Zu klein.” Du rätst 80. Ich sage: “Zu groß.” Du rätst 65. Ich sage: “Zu groß.” Du rätst 55. Ich sage: “Zu klein.” Du rätst 58. Ich sage: “Ja, richtig geraten.”

Maschinelles Lernen simuliert im Grunde einen ähnlichen Prozess, nur etwas komplexer. Es könnte viele `1 bis 100` geben, und es müssen viele Zahlen erraten werden. Gleichzeitig erfordert jedes Raten viele Berechnungen. Und jedes Mal, wenn entschieden wird, ob die Zahl zu groß oder zu klein ist, müssen viele Berechnungen durchgeführt werden.

PyTorch

Installieren Sie es. Dieses unterstützt Python in der Version `3.9`.

```
$ pip install torch torchvision
Collecting torch
  Downloading torch-1.8.0-cp39-none-macosx_10_9_x86_64.whl (120,6 MB)
     |                 | 120,6 MB 224 kB/s
Collecting torchvision
  Downloading torchvision-0.9.0-cp39-cp39-macosx_10_9_x86_64.whl (13,1 MB)
     |                 | 13,1 MB 549 kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.9/site-packages (from torch) (1.20.1)
Collecting typing-extensions
  Downloading typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.9/site-packages (from torchvisio
Installing collected packages: typing-extensions, torch, torchvision
```

```
Successfully installed torch-1.8.0 torchvision-0.9.0 typing-extensions-3.7.4.3
```

Überprüfen wir das.

```
import torch
x = torch.rand(5, 3)
print(x)
```

Es ist ein Fehler aufgetreten.

```
Traceback (zuletzt aufgerufener Befehl):
  Datei "torch.py", Zeile 1, in <module>
    import torch
  Datei "torch.py", Zeile 2, in <module>
    x = torch.rand(5, 3)
AttributeError: Das teilweise initialisierte Modul 'torch' hat kein Attribut 'rand' (höchstwahrscheinli
```

Ich habe die Fehlermeldung gegoogelt. Es stellte sich heraus, dass unsere Datei ebenfalls `torch` hieß. Es gab also einen Namenskonflikt. Nachdem ich den Namen geändert hatte, funktionierte alles korrekt.

```
tensor([[0.5520, 0.9446, 0.5543],
        [0.6192, 0.0908, 0.8726],
        [0.0223, 0.7685, 0.9814],
        [0.4019, 0.5406, 0.3861],
        [0.5485, 0.6040, 0.2387]])
```

Ein Beispiel finden.

```
# -*- coding: utf-8 -*-

import torch
import math
dtype = torch.float
device = torch.device("cpu")
# device = torch.device("cuda:0") # Entkommentieren Sie dies, um auf der GPU zu laufen
```

Zufällige Eingabe- und Ausgabedaten erstellen

x = torch.linspace(-math.pi, math.pi, 2000, device=device, dtype=dtype) y = torch.sin(x)

Gewichte zufällig initialisieren

a = torch.randn((), device=device, dtype=dtype) b = torch.randn((), device=device, dtype=dtype) c = torch.randn((), device=device, dtype=dtype) d = torch.randn((), device=device, dtype=dtype)

```
learning_rate = 1e-6
for t in range(2000):
    # Vorwärtspass: Berechne das vorhergesagte y
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

# Verlust berechnen und ausgeben
loss = (y_pred - y).pow(2).sum().item()
if t % 100 == 99:
    print(t, loss)

# Backpropagation zur Berechnung der Gradienten von a, b, c, d bezüglich des Verlusts
grad_y_pred = 2.0 * (y_pred - y)
grad_a = grad_y_pred.sum()
grad_b = (grad_y_pred * x).sum()
grad_c = (grad_y_pred * x ** 2).sum()
grad_d = (grad_y_pred * x ** 3).sum()

# Gewichte mit Gradientenabstieg aktualisieren
a -= learning_rate * grad_a
b -= learning_rate * grad_b
c -= learning_rate * grad_c
d -= learning_rate * grad_d
```

print(f'Ergebnis: y = {a.item()} + {b.item()} x + {c.item()} x^2 + {d.item()} x^3')

```
Führen Sie es aus.
```

```shell
99 1273.537353515625
199 849.24853515625
299 567.4786987304688
399 380.30291748046875
499 255.92752075195312
599 173.2559814453125
699 118.2861328125
799 81.72274780273438
899 57.39331817626953
999 41.198158264160156
1099 30.41307830810547
1199 23.227672576904297
1299 18.438262939453125
1399 15.244369506835938
1499 13.113286972045898
1599 11.690631866455078
1699 10.740333557128906
1799 10.105220794677734
1899 9.6804780960083
1999 9.39621353149414
Ergebnis: y = -0.011828352697193623 + 0.8360244631767273 x + 0.002040589228272438 x^2 + -0.090383656322
```

Schauen wir uns den Code an, der nur die `numpy`-Bibliothek verwendet.

```
# -*- coding: utf-8 -*-
import numpy as np
import math
```

Zufällige Eingabe- und Ausgabedaten erstellen

x = np.linspace(-math.pi, math.pi, 2000) y = np.sin(x)

Gewichte zufällig initialisieren

a = np.random.randn() b = np.random.randn() c = np.random.randn() d = np.random.randn()

learning_rate = 1e-6 for t in range(2000): # Vorwärtspass: Berechne das vorhergesagte y # y = a + b x + c x^2 + d x^3 y_pred = a + b * x + c * x ** 2 + d * x ** 3

```
# Verlust berechnen und ausgeben
loss = np.square(y_pred - y).sum()
if t % 100 == 99:
    print(t, loss)

# Backpropagation zur Berechnung der Gradienten von a, b, c, d bezüglich des Verlusts
grad_y_pred = 2.0 * (y_pred - y)
grad_a = grad_y_pred.sum()
grad_b = (grad_y_pred * x).sum()
grad_c = (grad_y_pred * x ** 2).sum()
grad_d = (grad_y_pred * x ** 3).sum()

# Gewichte aktualisieren
a -= learning_rate * grad_a
b -= learning_rate * grad_b
c -= learning_rate * grad_c
d -= learning_rate * grad_d

print(f'Ergebnis: y = {a} + {b} x + {c} x^2 + {d} x^3')
```

Beachten Sie, dass dies zwei verschiedene Berechnungsmethoden sind.

Diese beiden Beispiele erzeugen zunächst eine Gruppe von x- und y-Werten. Anschließend wird angenommen, dass es sich um eine kubische Gleichung handelt. Danach werden mit einigen Methoden die Koeffizienten iterativ berechnet. Wie genau funktionieren diese Algorithmen? Es ist zu beachten, dass die Schleife 2000 Mal durchlaufen wird, wobei die Anpassung bei jedem Durchlauf etwas genauer wird. Hier werden wir dies zunächst nicht im Detail untersuchen.

Fazit

Derzeit verstehen wir nicht, wie die Berechnungen hinter dem maschinellen Lernen ablaufen. Das ist jedoch vorerst nicht wichtig. Mit dem oben genannten Wissen können wir bereits viele

Dinge erreichen. Wir können maschinelles Lernen auch zur Verarbeitung von Texten, Audiodateien und mehr verwenden. Es ist nicht zu spät, die Prinzipien zu lernen, nachdem wir Dutzende von Beispielen ausprobiert haben.

Übung

- Die Schüler erkunden wie oben beschrieben.