Woche 10: Programmierung – Neuronale Netzwerke in Python (Teil 2)

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele

- Erklären können, was One-Hot-Codierung ist.
- Die Merkmalswerte eines Datensatzes auf einen Bereich zwischen 0 und 1 normalisieren können.
- Ein einfaches künstliches neuronales Netzwerk mithilfe des Keras-Moduls in Python programmieren können.
- Die Anzahl der Modellparameter berechnen können.

Inhalt

Datennormalisierung

Ein weiterer üblicher Schritt bei der Vorbereitung der Daten ist die Normalisierung. Bei der *Normalisierung* handelt es sich um eine Methode, bei der die Werte numerischer Merkmale so geändert werden, dass sie die gleiche Größenordnung haben. Meistens werden die

Projekt „KI für alle"
HeiCAD – Heine Center for Artificial Intelligence and Data Science

Merkmalswerte auf das Intervall zwischen 0 und 1 normalisiert. Die Normalisierung hilft einerseits den Trainingsprozess zu beschleunigen, andererseits verbessert es die Leistung der künstlichen neuronalen Netze. Wir normalisieren unseren Datensatz mit Hilfe der Funktion *minmax_scale()* des *Scikit-learn*-Untermoduls *Preprocessing and Normalization.* Da der Wertebereich mit 0 bis 1 in der Funktion voreingestellt ist, brauchen wir nichts weiter zu tun, als das Datenarray der Funktion zu übergeben. Um die originalen Werte des DataFrames nicht zu überschreiben, speichern wir den normalisierten Datensatz im *ndarray X_normalized* ab.

Quelle [6]

```
# Normalisiere die Merkmalswerte auf den Bereich [0,1]
X_normalized = preprocessing.minmax_scale(X)
print(X_normalized[:1,:])
```

```
[[1.         0.125      0.         0.01415106 0.         0.
  1.         0.         0.         1.         0.        ]]
```

Wie bei allen Klassifikationsaufgaben muss der Datensatz vor dem Training eines Klassifikationsmodells in eine Trainings- und eine Testmenge zerlegt werden. Dafür benutzen wir die Funktion *train_test_split()* des Untermoduls *Model Selection* von *Scikit-learn*. Dieser Funktion übergeben wir als Eingabe die Datenmatrix mit dem entsprechenden Zielmerkmal. Mit dem Parameter *test_size* legen wir die Größe der Testmenge fest. Diese soll 15 % des gesamten Datensatzes betragen. Weiterhin stellen wir durch die Wertzuweisung des Parameters *random_state* sicher, dass die Beobachtungen im Datensatz auf eine bestimmte Weise durchmischt werden, um die Reproduzierbarkeit unserer Zerlegung sicherzustellen.

```
# Teile den Datensatz in die Trainings- und die Testmenge auf
X_train, X_test, y_train, y_test = train_test_split(
    X_normalized, y, test_size=0.15, random_state=61)
```

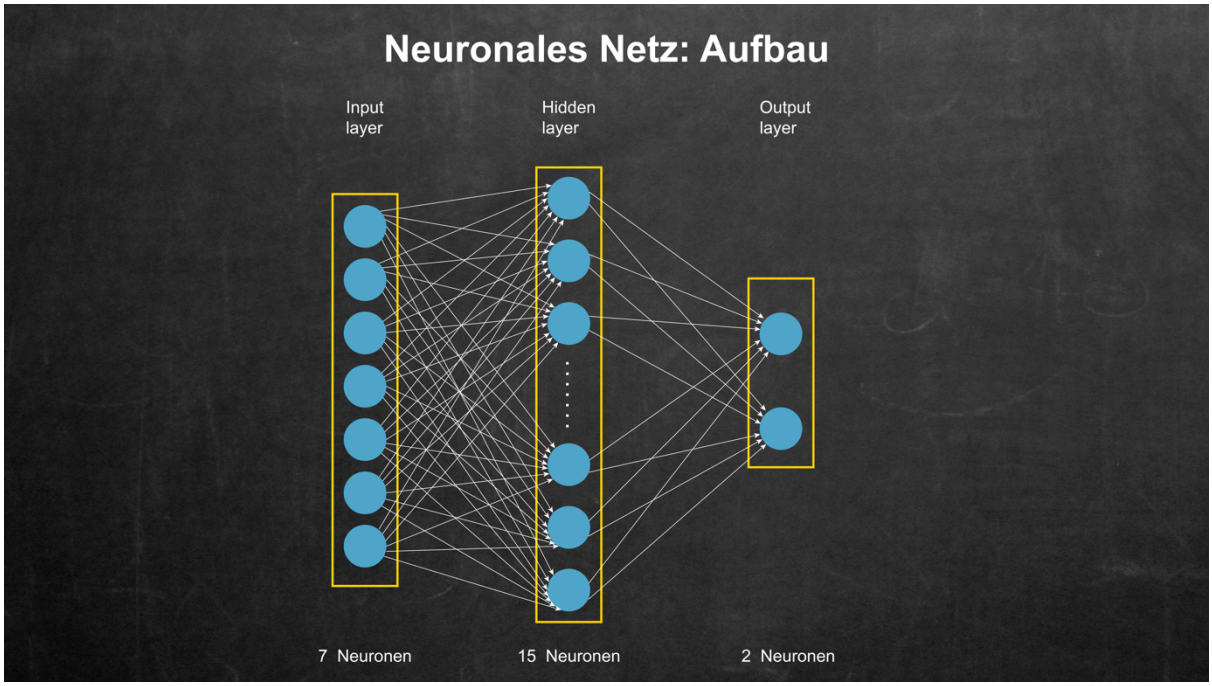
Aufbau eines neuronalen Netzes

Nachdem wir den Datensatz vorbereitet haben, kommen wir zur Implementierung eines neuronalen Netzes für unser Klassifikationsproblem. In Python können künstliche neuronale Netze mit Hilfe des Moduls *Keras* schnell und einfach aufgebaut, trainiert und ausgewertet werden. Deswegen ist Keras sowohl bei Einsteigern als auch Wissenschaftler*innen und erfahrenen Entwickler*innen sehr beliebt.

Quelle [7]

Generell besteht ein neuronales Netzwerk aus drei Teilen: der Eingabeschicht, einigen verborgenen Schichten und der Ausgabeschicht. Für die Vorhersage des Überlebens der Passagiere der Titanic implementieren wir ein sequenzielles Feed-Forward-Netz mit sieben Neuronen in der Eingabeschicht, die den sieben Merkmalen unseres Datensatzes entsprechen. Diese verknüpfen wir mit einer vollständig verbundenen Schicht mit 15

Neuronen. Zur Erinnerung: In einer vollständig verbundenen Schicht ist jedes Neuron mit allen anderen Neuronen benachbarter Schichten verbunden. Die Ausgabeschicht unseres Netzes enthält zwei Neuronen, die unseren beiden Klassen *überlebt* und *nicht überlebt* entsprechen.



Wir kehren zum Jupyter Notebook zurück und fangen mit der Implementierung unseres Netzes an. Zuerst erzeugen wir ein sequentielles Modell mit dem Befehl *keras.Sequential()*. Wir fügen unserem Modell die Input-Schicht mit der Funktion *add()* hinzu. In der Input-Schicht entspricht die Anzahl der Neuronen der Anzahl der Merkmale im Datensatz, die wir mit dem Befehl *shape* ausgeben lassen können, hier 7. Diese weisen wir dem Parameter *shape* der Input-Schicht zu. Nun fügen wir unserem Modell eine verborgene vollständig verbundene bzw. dense Schicht mit der Funktion *add()* hinzu. Wir weisen dem Parameter *units* 15 zu, um festzulegen, dass die vollständig verbundene Schicht 15 Neuronen enthalten soll. Außerdem können wir mit Hilfe des Parameters *activation* die Aktivierungsfunktion festlegen. Hierfür verwenden wir die ReLU-Funktion. Schließlich ergänzen wir unser Modell mit der vollständig verbundenen Ausgabeschicht mit zwei Neuronen, eins pro Klasse. Hier verwenden wir die Aktivierungsfunktion *softmax*, deren Vorteil darin besteht, dass ihre Ausgabe sich als Wahrscheinlichkeiten interpretieren lassen, zu bestimmten Klasse zu gehören.

```
model = keras.Sequential()
model.add(keras.Input(shape=(X_normalized.shape[1],)))
model.add(keras.layers.Dense(units=15, activation="relu"))
model.add(keras.layers.Dense(2, activation = "softmax"))
```

Anzahl der Optimierungsparameter

Nachdem wir das künstliche neuronale Netz implementiert haben, können wir mit der Methode *summary()* die Zusammenfassung des Modells anzeigen lassen. In der Ausgabe kannst du eine detaillierte Beschreibung einzelner Schichten des Modells inklusive ihrer Ausgaben und der Anzahl der trainierbaren Parameter sehen.

```
model.summary()
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 15)	120
dense_1 (Dense)	(None, 2)	32

Total params: 152 (608.00 B)

In der verborgenen vollständig verbundenen Schicht sind 120 trainierbare Parameter. Diese Zahl kommt daher, dass jedes der 15 Neuronen dieser Schicht mit jedem der sieben Neuronen der Eingabeschicht verbunden ist. Dadurch kommen 15 * 7 = 105 Eingaben bzw. Gewichte zustande. Hinzu kommen noch 15 Bias-Terme. Insgesamt ergeben sich 105 + 15, also 120 trainierbare Parameter in dieser Schicht.

Jedes der zwei Neuronen der Ausgabeschicht ist mit jedem der 15 Neuronen der verborgenen Schicht verbunden. Aufgrund dieser Verbindungen beläuft sich die Anzahl der Gewichte in der Ausgabeschicht auf 15 * 2, also 30 trainierbaren Parameter. Hinzu kommen noch zwei Bias-Terme. Insgesamt sind das 32 trainierbare Parameter in der Ausgabeschicht.

Wenn man die Anzahl der Parameter aller Schichten zusammenaddiert, kommt man auf 152 Parameter für das gesamte Modell, die während des Trainings optimiert werden.

Take-Home Message

In diesem Video hast du anhand des Titanic-Datensatzes gesehen, wie man Daten für das Trainieren künstlicher neuronaler Netze vorbereitet. Außerdem hast du gelernt, wie man ein einfaches Feed-Forward-Netz in Python programmiert.

Quellen

Quelle [6] Scikit-learn (2022). Preprocessing and Normalization. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.minmax_scale.html

Quelle [7] Chollet, F., & Others. (2015). Keras. Retrieved from https://keras.io

Weiterführendes Material

https://www.tensorflow.org/tutorials/keras/classification

Disclaimer

Transkript zu dem Video „Woche 10: Programmierung – Neuronale Netzwerke in Python (Teil 2)“, Ludmila Himmelspach.