Inhalt

Anmerkungen:

Eine hilfreiche Einführung in die Programmierung mit Python ist hier zu finden:
https://www.ilsb.tuwien.ac.at/lva/317.530_2017W/01.html

Hilfreiche Informationen zur Programmierung eines neuronalen Netzes sind ergänzend zu den Ausarbeitungen in diesem Projekt hier zu finden:

https://www.youtube.com/playlist?list=PLZbbT5o_s2xrL4F90oKfloWM7ExXT_U_b

https://www.youtube.com/playlist?list=PLZbbT5o_s2xrwRnXk_yCPtnqqo4_u2YGL

1. Was sind neuronale Netze?

Neuronale Netze sind Codestrukturen, die nach dem Vorbild des menschlichen Gehirns designt wurden. Bei neuronalen Netzen wird mit vielen verschiedenen trainierbaren Verknüpfungen gearbeitet, welche ähnlich wie menschliche Neuronen im Gehirn lernen können. Das Ziel ist es Probleme, die für uns Menschen einfach zu lösen sind wie z.B. die Unterscheidung zwischen Hunden und Katzen auf Bildern mit einem Computer lösen zu können.

Problem	Computer	Mensch
Tausende großer Zahlen schnell multiplizieren	Leicht	Schwer
Gesichter auf einem Foto mit einer Menschenmenge heraussuchen	Schwer	Leicht

Abbildung 1 Vergleich Computer mit Mensch (Rashid 2017)

Um diese Art der Datenauswertung zu ermöglichen, muss eine ähnliche Denkstruktur, wie sie ein Mensch besitzt auf eine Maschine übertragen werden.

Abbildung 2: Lernmuster des Menschen (Rashid 2017)

Am einfachsten ist dieses Denkmuster auf Maschinen zu übertragen, indem man iterativ aus der Eingabe die Lösung berechnet und diesen Prozess so lange wiederholt, bis man die kleinste mögliche Abweichung zum Sollergebnis erreicht hat (trainieren eines neuronalen Netzes).

Abbildung 3: Lernmuster eines Computers (Rashid 2017)

Das Training eines neuronalen Netzes kann dabei mit dem Lernen eines Menschen verglichen werden.

2. Aufbau eines Neuronalen Netzes

2.1 Input layer

Die erste Schicht (layer) eines neuronalen Netzes ist immer der input layer. Der input layer besteht aus einer einzelnen Schicht, welche so viele Knoten wie Eingabeparameter aufweisen muss. Der input layer besitzt als einzige layer keine Aktivierungsfunktion. Mit der input layer werden Daten aus Datensätzen in eine für das neuronale Netz passenden Struktur importiert.

Soll z.B. ein Wein auf Basis von 12 Weincharakteristika beschrieben werden, so muss der input layer für jedes Charakteristikum einen eigenen Knoten aufweisen (in Summe also 12 Knoten).

Die Knoten der input layer wird durch Gewichtung mit den Knoten des nächsten layer verbunden.

2.2 Hidden layer

Die Hidden layer bestehen aus einer beliebig großen Anzahl an Knoten, welche über Gewichte mit den vorherigen und nachfolgenden layer verknüpft sind. Jeder hidden layer besitzt eine Aktivierungsfunktion, welche bestimmt, wie das „Neuron“ feuern soll (bei sigmuider Aktivierungsfunktion z.B. sobald der Schwellenwert überschritten wird.

Im Gegensatz zu den input und output layer kann ein neuronales Netz beliebig viele hidden layer aufweisen. Dabei gilt zu beachten, dass viele hidden layer nicht automatisch zu einem besseren Ergebnis, sondern eher zu einem überangepassten Model führen. Außerdem ist zu beachten, dass je mehr hidden layer verwendet werden, desto größer sollte der Trainingsdatensatz sein.

Die hidden layer sind die wichtigsten Bestandteile eines neuronalen Netzes, da sie für das Lernverhalten des neuronalen Netzes entscheidend sind.

Um das neuronale Netz auch ohne vorheriges Training verwenden zu können, können die Gewichte (weights) eines zuvor trainierten neuronalen Netzes gespeichert werden. So sind Vorhersagen ohne vorheriges Training möglich.

2.3 Output layer

Der output layer dient dazu die ermittelten Ergebnisse des neuronalen Netzes an den Benutzer auszugeben. Dabei steht jeder Knoten in der output layer für eine möglich Ausgabe (z.B. eine Weinqualität).

Output layer repräsentieren die letzte Schicht des Neuronalen Netzwerkes, welches das finale Ergebnis des Models über eine Aktivierungsfunktion liefert.

Der Output layer ist in der Regel vollständig mit dem vorherigen hidden layer vernetzt. Soll beispielsweise ein ANN unterscheiden, ob es sich bei einem Unbekannten Wein um einen Weißwein oder Rotwein handelt, so haben wir im Output layer zwei Neuronen, welche repräsentativ für jeweils eines der beiden Weinsorten stehen siehe Abbildung 6.

5. Aktivierungsfunktion

In dieser Arbeit wurde mit drei verschiedenen Aktivierungsfunktionen gearbeitet. Diese sind namentlich die ReLu Funktion, die Sigmuide Funktion und die Softmax Funktion.

Lineare Funktion: Diese Funktion nimmt die Eingaben, multipliziert mit einer Gewichtung für jedes Neuron und erzeugt ein Ausgangssignal. Dieses Signal ist proportional zur Eingabe. Die Lineare Funktion ist in der Lage mehr als nur Ja oder Nein antworten zuzulassen. Allerdings hat sie für unser Projekt den großen Nachteil, dass sie nicht für Backpropagation verwendet werden kann, um das Modell zu trainieren. Grund dafür ist die Ableitungsfunktion, da diese eine Konstante ist und keine Beziehung zur Eingabe X hat. Es ist also nicht möglich, zurückzugehen und zu verstehen, welche Gewichte in den Eingangsneuronen eine bessere Vorhersage liefern können. Eine linearen Aktivierungsfunktion ist in Abbildung 4 zu sehen.

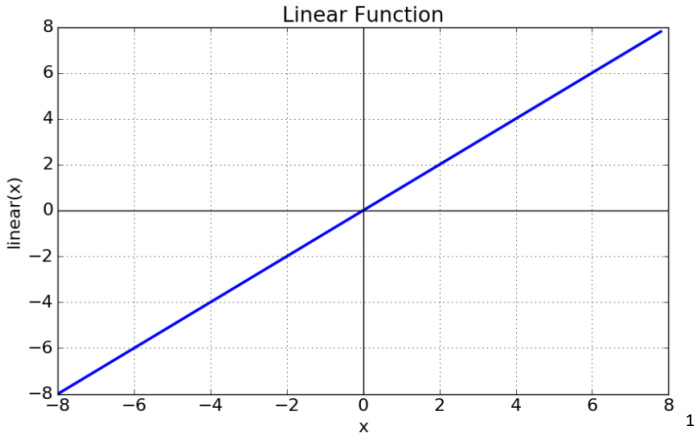
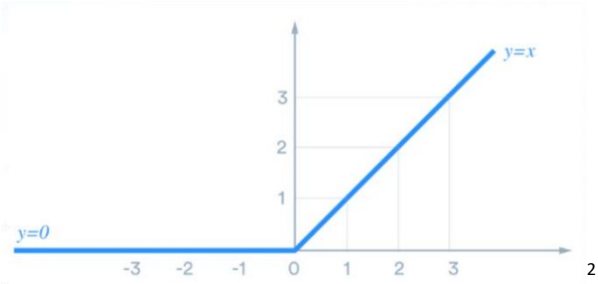


Abbildung 4 Graphische Darstellung einer linearen Aktivierungsfunktion

[1] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

ReLU Funktion: Die gleichgerichtete lineare Aktivierungsfunktion oder kurz ReLU ist eine stückweise lineare Funktion, die Eingabe direkt ausgibt, sofern diese positiv ist, andernfalls wird Null ausgeben. Die ReLu Funktion ist zur Standard-Aktivierungsfunktion für viele Arten von neuronalen Netzen geworden, weil ein Modell damit leicht und mit guten Leistungen trainiert werden kann. Für einen bestimmten Knoten werden die Eingaben mit der Gewichtung in einem Knoten erst multipliziert und dann summiert. Dieser Wert wird als summierte Aktivierung des Knotens bezeichnet. Die summierte Aktivierung wird dann über eine Aktivierungsfunktion transformiert und definiert den spezifischen Ausgang oder die "Aktivierung" des Knotens.

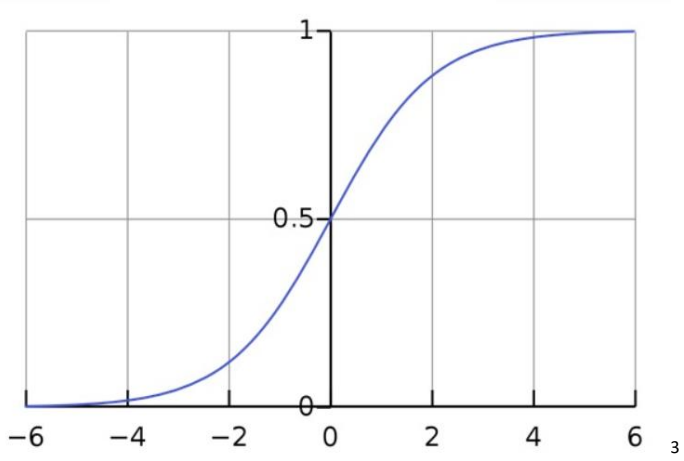
Eine Repräsentation der Kurve kann in Abbildung 5 gesehen werden.



[2]

Abbildung 5 Grafische Darstellung der ReLu Funktion

Sigmuide Funktion: Der Hauptgrund, warum Sigmoide Funktionen verwenden werden, ist, dass sie zwischen (0 und 1) liegt. Daher wird sie besonders für Modelle verwendet, in denen die Wahrscheinlichkeit als Ausgabe vorhergesehen ist. Da die Wahrscheinlichkeit sich immer zwischen 0 und 1 bewegt, ist die Sigmuide Funktion die richtige Wahl. Die Funktion ist differenzierbar, d.h. wir können die Steigung der Sigmuide Kurve an zwei beliebigen Punkten bestimmen. Ein Beispiel einer Sigmuiden Funktion ist in Abbildung 6 erkennbar.



[3]

Abbildung 6 Grafische Darstellung einer Sigmuiden Funktion

[2] https://miro.medium.com/max/2400/1*DfMRHwxY1gyyDmrIAd-gjQ.png

[3] https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg

Softmax Funktion: Sie ist eine mathematische Funktion, die einen Vektor von Zahlen in einen Vektor von Wahrscheinlichkeiten umwandelt, wobei die Wahrscheinlichkeiten jedes Wertes proportional zur relativen Skala jedes Wertes im Vektor sind. Die Softmax-Funktion ist besonders wichtig, um die Ausgaben zu normalisieren und sie von gewichteten Summenwerten in Wahrscheinlichkeiten umzuwandeln, die sich zu eins summieren. Jeder Wert in der Ausgabe der Softmax-Funktion wird als Wahrscheinlichkeit der Mitgliedschaft für jede Klasse interpretiert. Am häufigsten wird sie als Aktivierungsfunktion Output layer verwendet, um Wahrscheinlichkeitsverteilungen vorherzusagen. Ein Softmax Funktion ist grafisch in Abbildung 7 beschrieben.

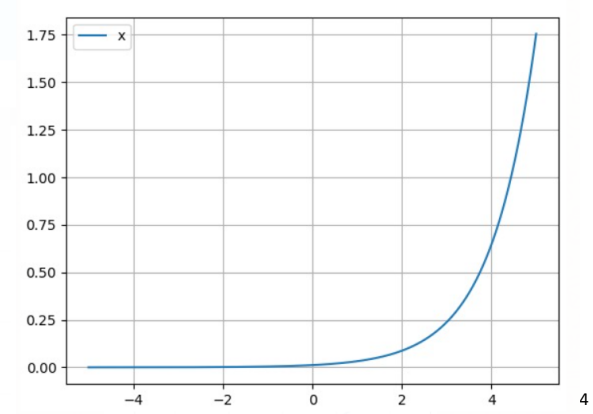


Abbildung 7 Grafische Abbildung einer Softmax Funktion

6. Backpropagation

Der Backpropagation-Algorithmus ist eine Lernmethode für mehrschichtige Neuronale Netzwerke. Das Prinzip des Backpropagation-Ansatzes besteht darin, eine gegebene bekannte Funktion zu modellieren, indem die internen Gewichtungen der Eingangssignale modifiziert werden, um einen bekanntes wahres Ausgangssignal zu erzeugen (Ausgangssignal: Rotwein oder Weißwein). Das System wird so trainiert, dass der Fehler zwischen dem Output layer und einem bekannten erwarteten Wert vergleichen wird und dann zur Modifikation der Gewichtungen im neuronalen Netzwerk verwendet wird. Das Ziel ist es diesen Fehler so gering wie möglich zu halten und somit in möglichst vielen Fällen eine korrekte Zuordnung des bestimmten und des wahren wertes zu erhalten. Da der Backpropagation-Algorithmus Teil der mehrschichtigen Feed-Forward-Neuronalen Netzwerk Familie ist, erfordert es die Definition, dass eine Netzwerkstruktur gegeben sein muss. Diese Netzwerkstruktur besteht aus Schichten, welche vollständig mit der nächsten Schicht verbunden ist. Eine Standard-Netzwerkstruktur besteht aus einem Input layer, mindestens einem hidden layer und einer Output layer. Der Backpropagation- Algorithmus kann sowohl für Klassifizierung als auch für Regressionsprobleme verwendet werden.

[4] https://www.machinecurve.com/wp-content/uploads/2020/01/softmax_logits.png

7. Güte des Modells

7.1 Loss beim Training

Bei der Verwendung eines ANN versuchen wir einen Fehlerterm (loss) zu minimieren. Beispielweise möchten wir einen unbekannten Wein entweder einem Rotwein mit dem label 0 der einem Weißwein mit dem label 1 zuordnen. Der Backpropagation-Algorithmus versucht den Fehler im Netzwerk zu minimieren. Daher wird eine Zielfunktion benötigt, welche auch als Verlustfunktion bezeichnet wird. Der von dieser Verlustfunktion berechnete Wert wird als loss bezeichnet. Sie ist daher so wichtig, da sie alle Aspekte des Modells zu einer einzigen Zahl herunterdestillieren muss, und zwar so, dass Verbesserungen in dieser Zahl ein Zeichen für ein Verbesserung des Modells ist.

7.2 Accuracy

Die Genauigkeit (Accuracy) eines Modells wird über den gesamten Test und Optimierungszeitraum überwacht. Die Anwendung der Genauigkeit kann wie folgt beobachtet werden. Testproben werden dem Modell zugeführt und die Anzahl der Fehler, die das Modell macht, wird nach dem Vergleich mit den wahren Zielvorgaben aufgezeichnet. Dann wird der Prozentsatz der Fehlklassifizierung berechnet. Wenn zum Beispiel die Anzahl der Testproben 1000 beträgt und das Modell 913 davon richtig klassifiziert, dann beträgt die Genauigkeit des Modells 91,3%.

7.3 Lernrate (learning rate)

Die Lernrate ein konfigurierbarer Parameter, der beim Training von neuronalen Netzen verwendet wird und einen kleinen positiven Wert hat, welcher im Bereich zwischen 0 und 1 liegt. Die Lernrate steuert, wie schnell das Modell an das Problem angepasst wird. Kleinere Lernraten erfordern mehr Trainingsepochen, da bei jeder Aktualisierung kleinere Änderungen an den Gewichten vorgenommen werden. Größere Lernraten führen zu schnellen Änderungen und weniger Trainingsepochen, sind damit schneller aber im Vergleich weniger präzise. Eine zu große Lernrate kann dazu führen, dass das Modell zu schnell zu einer suboptimalen Lösung kommt, während eine zu kleine Lernrate dazu führen kann, dass der Prozess stecken bleibt und sehr lange Zeit benötigt.

7.4 Model overfit

Eine Überanpassung des Models (overfit) tritt auf, wenn ein Modell versucht das Rauschen in den Daten zu modellieren. Dies führt dann zu sehr schlechten Vorhersagen bei neu zugeführten Daten. Ein Overfit tritt meist bei Modellen mit komplexer Struktur und zu vielen Parametern auf. Ein überangepasstes Modell ist ungenau, weil der Trend nicht die in den Daten vorhandene Realität widerspiegelt. Man kann eine überangepasstes Model daran erkennen, dass es beim Trainingsdatensatz gute Ergebnisse produziert, aber auf den nicht gesehenen Testdatensatz in Summe schlecht abschneidet. Das Ziel eines maschinellen Lernens ist es, gut von den Trainingsdaten auf alle anderen Daten aus der Problemdomäne schließen zu können. Dies ist sehr wichtig, da wir wollen, dass unser Modell Vorhersagen für die Zukunft auf Daten macht, die es noch nie zuvor gesehen hat.

Ein Anzeichen für einen overfit ist z.B. wenn die accuracy des Trainingsdatensatz steigt, die accuracy eines Validierungsdatensatzes jedoch sinkt.

7.5 Model underfit

Als Gegenstück des overfit gibt es den underfit, dieser Begriff beschreibt ein unterangepasstes Modell. Ein Underfit kann dadurch erkannt werden, dass ein Fortführen des Trainings zu einer weiteren Verkleinerung des Fehlertherms führt. Das deutet dann darauf hin, dass weitere Verbesserungen im Model möglich sind und das Modell nicht ausreichend trainiert wurde.

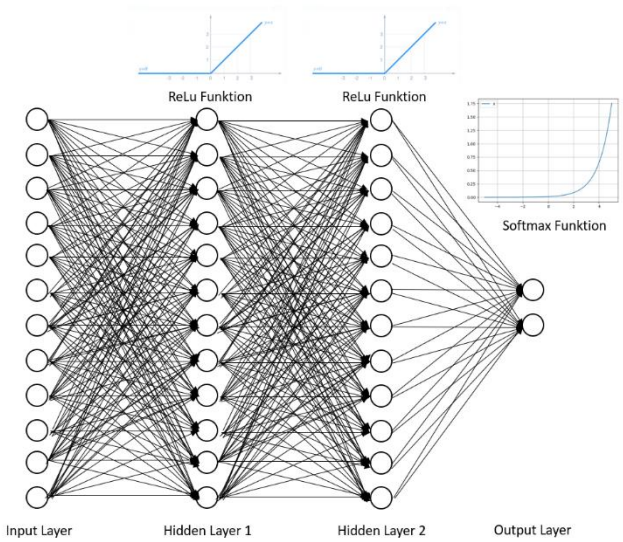


Abbildung 8 Darstellung eines Vollständigen ANN mit einem Input-, zwei hidden- und einem Output layer

Literaturverzeichnis

Rashid, Tariq (2017): Neuronale Netze selbst programmieren. Ein verständlicher Einstieg mit Python. 1. Auflage. Heidelberg: O'Reilly (Animals).