



MATHEMATISCHE GRUNDLAGEN DES MASCHINELLEN LERNENS,
SoSE22

Neuronale Netzwerke und Backpropagation

*Alexander Binkowski, Erdenetuya Undral, Theodora
Omiridou, Xuantong Pan*

Julie 2022

Inhaltsverzeichnis

1 Einleitung	2
2 Umsetzung	2
3 Auswertung	3
3.1 Test der Funktion mit Testdatensatz	3
3.2 Parameterwahl	5
3.3 Test NASA - Nearest Earth Objects	7
4 Fazit	9

1 Einleitung

Umzusetzen ist die Vervollständigung des bereitgestellten Python-Codes MLP, welche einen klassenbasierten Ansatz liefert, um ein neuronales Netzwerk zu implementieren. Das Netzwerk besteht aus zwei abstrakten Klassen, die die Funktionalitäten, wie berechneten Werte, Ableitungen und Ausgaben für die gewünschten Fehler- und Aktivierungsfunktionen liefern. Zentral ist die Multi-LayerPerceptron Klasse, in dem das neuronale Netzwerk lernt (learn), vorhersagt (predict) und verschiedene Ausgaben zur Verfügung stellt. Außerdem existiert die Layer Klasse, welche die Art des Layers charakterisiert und die Datenstruktur, sowie die Anzahl der Neuronen bestimmt. In der zu modifizierten letzten Perceptron Klasse, die unter anderem die zugehörigen Gewichte (weights) und Verzerrung (bias), Ableitungen und den Aktivierungswert abspeichert, soll jetzt die Backpropagation Funktion backprop vervollständigt werden. Backpropagation oder auch Fehlerrückführung / Rückpropagierung, ist ein weit verbreitetes Verfahren, welches für das Einlernen von künstlichen neuronalen Netzen eingesetzt wird. Dieses Verfahren gehört zur Gruppe von überwachten Lernverfahren und ist ein Spezialfall von einem allgemeinen Gradientenverfahren, welches auf dem mittleren quadratischen Fehler basiert.

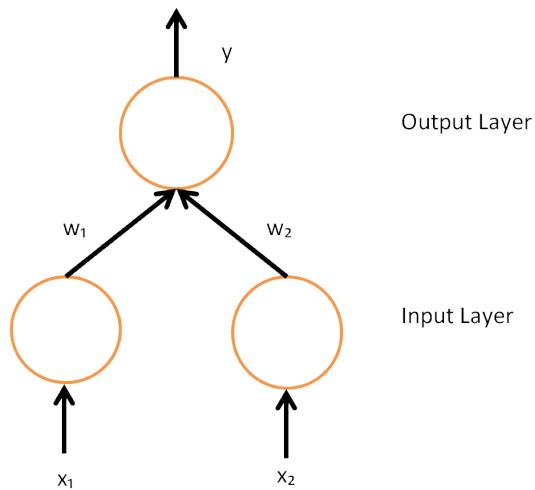


Abbildung 1: ein einfaches neuronales Netzwerk

Quelle: <https://en.wikipedia.org/wiki/Backpropagation>

2 Umsetzung

Das Besondere an dem Verfahren der Rückpropagierung ist das Vermeiden der sonst notwendigen üblichen Berechnung von Ableitungen mittels numerischen Methoden wie den finiten Differenzen. Wie der Name der Methode andeuten lässt, setzt das Verfahren am Output des neuronalen Netzes an. Dort werden zunächst die Ableitungen der Fehlerfunktion an den einzelnen Neuronen berechnet. Da dies bekannte Funktionen sind, muss für jeden Output nur eine Funktion berechnet und der ermittelte Wert im Perceptron hinterlegt werden. Im nächsten Schritt wandert das Verfahren in die darauf folgende Schicht und rechnet für jedes dazugehörige Perceptron die Ableitung der ankommenden Gewichte aus, summiert alle und multipliziert die Lernrate mit den Ableitungen sowie den anliegenden Aktivierungswert des Perceptrons und korrigiert die ankommenden Gewichte mit dem berechneten Produkt. Diese Prozedur wird bis zum Input, für jede Schicht im neuronalen Netz durchgeführt. Ähnlich verhält es sich mit den jeweiligen Verzerrungen der Perceptronen. Da in jedem Schritt die Ableitungen abgespeichert und diese ausschließlich aus einfachen Additionen und Multiplikationen bestehen, kann mit Maschinenpräzision das Gradientenverfahren durchgeführt werden.

Algorithm 1: backprop(precedingLayer, succeedingLayer, index, learningRate)

input: the preceding layer class precdintLayer, the succeeding layer class succeedingLayer, the index integer of the perceptron in the current layer index, the stepsize float of the gradient decent step learningRate

```
1 if precedingLayer = None then
2 | return Null;
3 end
4 if succeedingLayer = None then
5 | d ←  $\frac{\partial c}{\partial p_i}$ ;
6 else
7 | d ← 0;
8 | for length layer n do
9 | | d ← d +  $\frac{\partial p_i}{\partial w_i}$ ;
10 | end
11 end
12 bias ← bias - learningRate × d ×  $\frac{\partial z_i}{\partial b}$ ;
13 weights ← weights - learningRate × d ×  $\frac{\partial z_i}{\partial w_i} \times p$  ;
14 derivatives ← derivatives × d;
```

```
def backprop(self, precedingLayer, succeedingLayer, index, learningRate):
    # do nothing if this is inputlayer
    if precedingLayer is None:
        return
    # get derivative of lossfucntion if this is outputlayer
    if succeedingLayer is None:
        d = self.mlp.lossFunction.getDerivative(index)
    # else eval and sum over all derivitaves with respect to weights
    else:
        d = 0
        for i in range(succeedingLayer.numberOfPerceptrons()):
            d += succeedingLayer.getPerceptron(i).getDerivative(index)

    # do sgd for bias
    self.bias -= learningRate * d * self.activationFunction.getDerivative()
    # get value of perceptrons of preceding layer
    pPerceptrons = precedingLayer.getPerceptrons().astype(np.float)
    # do sgd for weights
    self.weights -= learningRate * d * self.activationFunction.getDerivative() * pPerceptrons
    # update derivaties of perceptron
    self.derivatives *= d
```

Abbildung 2: Backprop Code

3 Auswertung

3.1 Test der Funktion mit Testdatensatz

Zum Test des umgesetzten Verfahrens wird ein selbst erstellter Datensatz verwendet. Dieser besteht aus zwei unterschiedlichen Klassen, welche in Ringen angeordnet sind. Die Punkte, die sich in dem äußenen Kreis befinden, gehören zur gleichen gelb eingefärbten und die im inneren zur blauen eingefärbten Kategorie. Ziel des neuronalen Netzes ist es also, alle Punkte den entsprechenden Kategorien zuzuordnen.

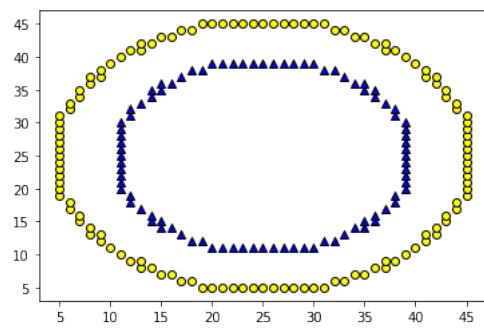


Abbildung 3: Testdatensatz, Kreispunkte: gelb, Dreieckpunkte: blau

Abbildung 4: Klassifikation der Testdaten

Die Animation soll die Klassifikation der Testdaten darstellen. Hierbei wird der Raum, in dem sich die Punkte befinden, je nach Klasse, gelb für Kreispunkte und blau für Dreieckspunkte eingefärbt. In den ersten acht Epochen wird das gesamte Gebiet der Animation abwechselnd gelb oder blau eingefärbt. Erst ab der neunten Epoche kristallisiert sich eine eindeutig erkennbare Aufteilung des Gebietes in blaue sowie gelbe Bereiche. Es bilden sich bis zur 22. Epoche drei Bereiche, einer unten links als gelbe Dreiecksfläche, ein zweiter als blaue Diagonale von oben links bis unten rechts, und ein dritter oben rechts ebenfalls als gelbe Dreiecksfläche. Ab der 30. Epoche reduziert sich die Unterteilung auf einen um den Mittelpunkt gelegenen blauen Dreiecksbereich und einen gelben Restbereich. Im übrigen Verlauf des Verfahrens verbessert sich die Einfärbung, wobei aus dem Dreieck eine Kreisscheibe entsteht. Mit jeder Epoche verringert sich der berechnete Fehler des Netzwerks.

3.2 Parameterwahl

Für mehrere Topologien wurden Schrittweiten von $1.0 \leq \lambda \leq 1.9$ getestet und deren Fehler nach 50 Epochen aufgetragen. Als Verlustfunktion wurde meansquared und als Aktivierungsfunktion die Sigmoidfunktion verwendet. Dabei entstand der kleinste Fehler mit einer Lernrate von $\lambda = 1,9$ und der Topologie [2 10 10 1]. Nur bei einer Lernrate von $\lambda = 1,6$ wich die Genauigkeit von 1.0 ab und lag bei ungefähr 0,748. Der genaue Verlauf des Fehlers kann in Abbildung 5 mit steigender Lernrate abgelesen werden.

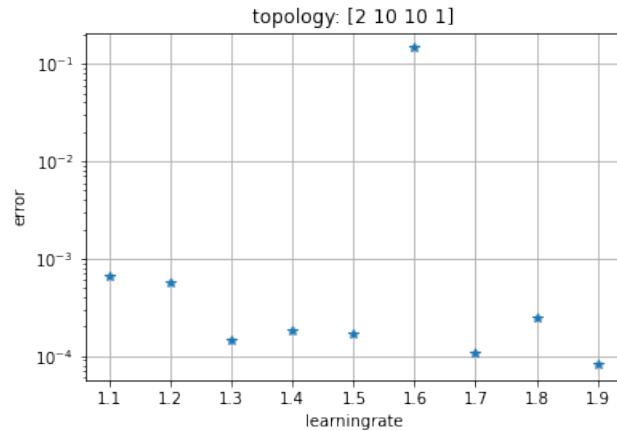
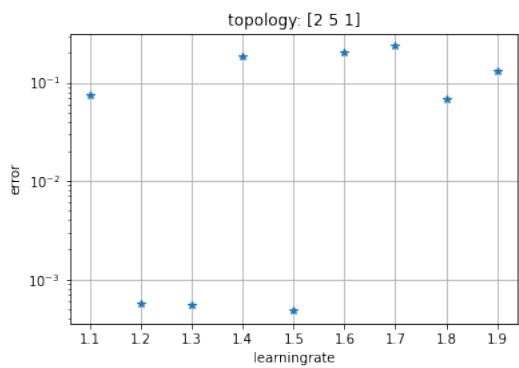
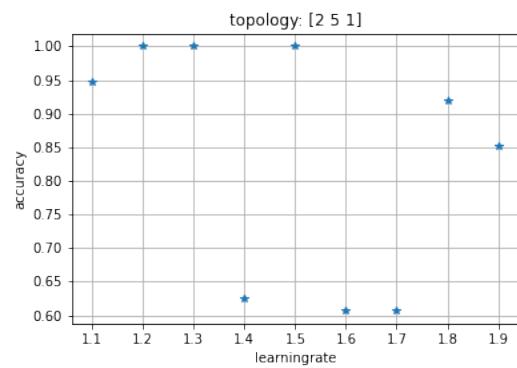


Abbildung 5

Abbildung 6(a) zeigt die Topologie [2 5 1], für die die gleiche Wahl der Fehler, unzureichende Ergebnisse erzielt. Zur Interpretation des Fehlers wird in 6(b) für jede Lernrate die Genauigkeit aufgetragen. Mit den Lernraten $\lambda = \{1.2, 1.3, 1.5\}$ werden minimale Fehler erzeugt, die eine absolute Genauigkeit bei der Vorhersage verursachen. Daraus lässt sich vermuten, dass ein bestimmter Fehler unterschritten werden muss, um eine ideale Genauigkeit zu erreichen.



(a) Fehler



(b) Genauigkeit

Abbildung 6

Abbildung 7: Prediction der Testdaten

3.3 Test NASA - Nearest Earth Objects

Zum Test der Klasse wird ein Datensatz von kaggle.com, mit 90836 Einträgen zu Erdnahen Objekten verwendet, die mitunter ID und Namen, mehrere relevante Kennwerte zur Größe, relativen Geschwindigkeit, Verfehlungsdistanz, geschätztem Durchmesser, sowie absoluter Leuchtkraft und einer Einschätzung zur Gefahr für die Erde beinhaltet. Zusätzliche Informationen zu den Daten unter der URL: https://cneos.jpl.nasa.gov/ca/neo_ca_tutorial.html

	id	name	est_diameter_min	est_diameter_max	relative_velocity	miss_distance	orbiting_body	sentry_object	absolute_magnitude	hazardous
0	2162635	162635 (2000 SS164)	1.198271	2.679415	13569.249224	5.483974e+07	Earth	False	16.73	False
1	2277475	277475 (2005 WK4)	0.265800	0.594347	73588.726663	6.143813e+07	Earth	False	20.00	True
2	2512244	512244 (2015 YE18)	0.722030	1.614507	114258.692129	4.979872e+07	Earth	False	17.83	False
3	3596030	(2012 BV13)	0.096506	0.215794	24764.303138	2.543497e+07	Earth	False	22.20	False
4	3667127	(2014 GE35)	0.255009	0.570217	42737.733765	4.627557e+07	Earth	False	20.09	True

Abbildung 8: NASA - Nearest Earth Objects Dataset

Quelle: <https://www.kaggle.com/datasets/sameepvani/nasa-nearest-earth-objects>

Damit das neuronale Netz trainiert werden kann, müssen zunächst alle relevanten Spalten aus dem Dataframe extrahiert und passend skaliert werden. Dazu wird die Funktion StandardScaler aus sklearn.preprocessing verwendet, welche die Merkmale durch Entfernen des Mittelwerts und Skalierung auf Einheitsvarianz, standardisiert. Zunächst wird noch vor dem eigentlichen Training eine Gegenüberstellung von den Objekten im Datensatz, die als gefährlich bzw. ungefährlich eingestuft worden sind, durchgeführt. Abbildung 9 zeigt, dass ungefähr neunmal mehr ungefährliche Objekte im Datensatz als gefährliche existieren und obwohl das etwas Erfreuliches für die Bevölkerung der Erde bedeutet, kann dieses Ungleichgewicht beim split in Lern und Testdaten verursachen, dass nur ungefährliche Objekte in den Testdaten landen.

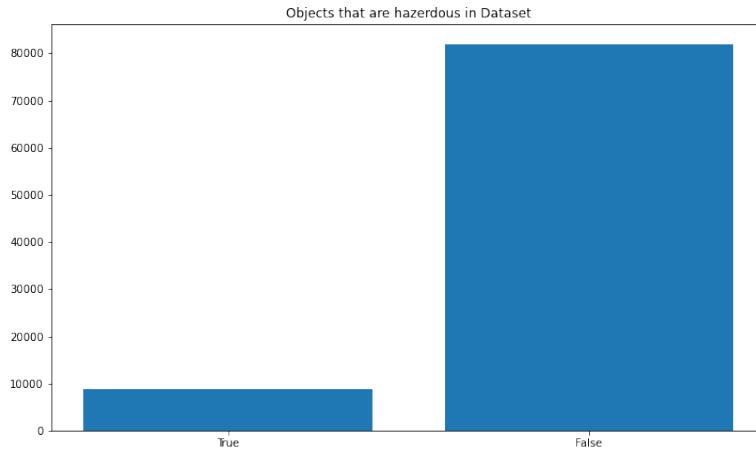


Abbildung 9: Gegenüberstellung der Anzahl als gefährlich und ungefährlich eingestuften Objekte

Wird mit diesen Testdaten ein predict durchgeführt, kann, trotz eventuell hoher erzielter Genauigkeit, keine Aussage über die Qualität des Ergebnisses gemacht werden. Um dieses Dilemma zu umgehen, wird SVMSMOTE Funktion aus imblearn.over_sampling verwendet, welche neue synthetische Daten aus einem Datensatz generiert, sodass eine Gleichverteilung der Gefahr für die Erde entsteht. (siehe Abbildung 9)

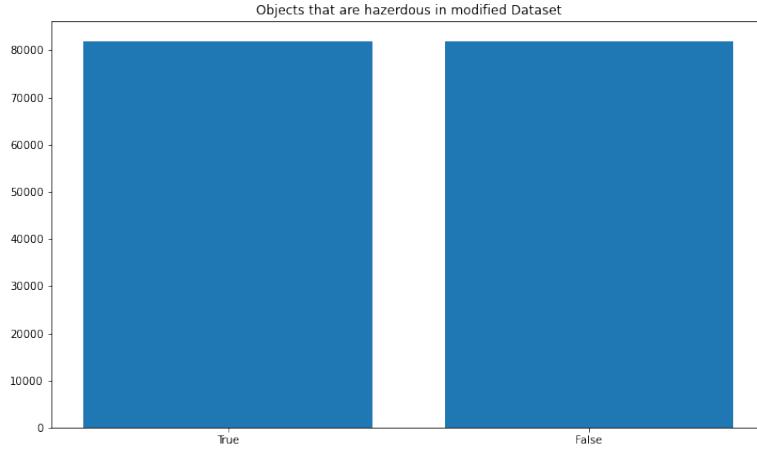


Abbildung 10: Gegenüberstellung der Anzahl als gefährlich und ungefährlich eingestuften Objekte

Nachdem die Verarbeitung der Daten abgeschlossen ist, kann das Training des neuronalen Netzwerks, zur Erkennung von bedrohlichen erdnahen Objekten, beginnen. Das beste Ergebnis wurde mit folgenden Parametern bewirkt. Die Topologie [5 16 8 1], die Lernrate 0.1, der Sigmoid Aktivierungsfunktion, der meansquared Kostenfunktion und einem test train split von 0.8. Nach 1000 Epochen ergab der Wert der Kostenfunktion etwa 0.059, mit einer Genauigkeit von 90% in den Lerndaten und 86,42% in den Testdaten.

```

Structure:
Layer 0:
    Number of Neurons: 5

Layer 1:
    Number of Neurons: 16
    Activation Function: Sigmoid

Layer 2:
    Number of Neurons: 8
    Activation Function: Sigmoid

Layer 3:
    Number of Neurons: 1
    Activation Function: Sigmoid

Loss Function: MeanSquaredLossFunction
Learningrate: 0.1
Epoch: 1000
Loss: 0.05892308226223468
Accuracy: 0.9

```

Abbildung 11: Parameterwahl und Ergebnis nach 1000 Epochen und Genauigkeit der Testdaten von 86,42%

4 Fazit

Die Backpropagation lässt sich in wenigen Zeilen in python-code umsetzen. Das besondere dabei ist, das Berechnen und Abspeichern der Ableitungen ohne auf numerische Methoden, wie den finiten Differenzen, angewiesen zu sein. Dies ermöglicht Schnelligkeit und Maschinengenauigkeit in jeder Iteration des Verfahrens. Hilfreich bei der Auswahl der Aktivierungsfunktion, sowie der Kostenfunktion ist, das genaue Studieren des Datensatzes, da dieser über die Anzahl der Outputs, den Bedarf nach Verarbeitung der Daten, sowie den Einsatz der besten Aktivierungs- und Kostenfunktionen diktieren. Wichtig beim Einsatz der Kreuzentropie als Kostenfunktion ist, dass die Aktivierungsfunktion im Outputlayer linear ist, weil die Genauigkeit des neuronalen Netzwerks sonst deutlich abnimmt. Zumal das Finden der richtigen Parameter die meiste Zeit in Anspruch nimmt, ist der Einbau von Hilfsfunktionen, die die Parameterwahl automatisiert und die Ergebnisse zur genaueren Analyse abspeichert, ebenfalls sinnvoll. Bei der Analyse der Ergebnisse wird deutlich, dass ein besonders kleiner Fehler nicht unbedingt eine hohe Genauigkeit im predict bedeutet. So kann eine zu große Wahl der maximalen Epochen, ein overfit der Trainings-Daten verursachen und das neuronale Netzwerk für andere Daten unbrauchbar machen.