Grundbegriffe des maschinellen Lernens

Clemens Dautermann

9. Januar 2020

Inhaltsverzeichnis

1 Was ist maschinelles Lernen?

Die wohl bekannteste und am häufigsten zitierte Definiton des maschinellen Lernens stammt von Arthur Samuel aus dem Jahr 1959. Er war Pionier auf diesem Gebiet und rief den Begriff „machine learning" ins Leben. So sagte er:

> [Machine learning is the] field of study that gives computers the ability to learn without being explicitly programmed[**1**].
>
> —Arthur Samuel, 1959

Beim maschinellen lernen werden Computer also nicht mit einem bestimmten Algorythmus programmiert um eine Aufgabe zu lösen, sondern lernen eigenständig diese Aufgabe zu bewältigen. Dies geschieht zumeist, indem das Programm aus einer großen, bereits „gelabelten", Datenmenge mit Hilfe bestimmter Methoden, die im Folgenden weiter erläutert werden sollen, lernt, gewisse Muster abzuleiten um eine ähnliche Datenmenge selber „labeln" zu können. Als Label bezeichent man in diesem Fall die gewünschte Ausgabe des Programmes. Dies kann beispielsweise eine Klassifikation sein. Soll das Programm etwa handgeschriebene Ziffern erkennen können, so bezeichnet man das (bearbeitete) Bild der Ziffer als „Input Verctor" und die Information welche Ziffer der Copmputer hätte erkennen sollen, als „Label ". Soll jedoch maschinell erlernt werden, ein simuliertes Auto zu fahren, so bestünde der Input Vector aus Sensorinformationen und das Label würde aussagen, in welche Richtung das Lenkrad hätte gedreht werden sollen, wie viel Gas das Programm hätte geben sollen oder andere Steuerungsinformationen. Der Input Vector ist also immer die Eingabe, die der Computer erhält um daraus zu lernen und das Label ist die richtige Antwort, die vom Programm erwartet wurde. Für maschinelles Lernen wird also vor allem eins benötigt: Ein enormer Datensatz, der bereits gelabelt wurde, damit das Programm daraus lernen kann.
Natürlich werden für maschinelles Lernen trotzdem Algorythmen benötigt. Diese Algorythmen sind jedoch keine problemspezifischen Algorythmen, sondern Algorythmen für maschinelles Lernen. Eine der populärsten Methoden des maschinellen Lernens ist das sogenannte „Neuronale Netz".

3 Neuronale Netze

bei Neuronalen Netzen handelt es sich um eine programminterne Struktur, die für das maschinelle Lernen genutzt wird. Wie der Name bereits vermuten lässt, ist diese Methode ein Versuch das menschliche Lernen nachzuahmen.

3.1 Maschinelles Lernen und menschliches Lernen

Das menschliche Gehirn ist aus sogenannten „Neuronen" aufgebaut. Ein Neuron ist eine Nervenzelle, die elektrische oder chemische Impulse annimmt, und gegebenenfalls einen elektrischen oder chemischen Impuls weitergibt. Die Nervenzellen berühren sich nicht direkt sondern sind nur über die sogenannten Synnapsen verbunden, über die diese Signale übertragen werden, sodass sich ein hoch komplexes Netzwerk von milliarden von Neuronen ergibt.[1] Ein neuronales Netz ist ähnlich aufgebaut. Es besteht aus „Neuronen", die eine theoretisch beliebige Anzahl von Eingaben annehmen können und mit einer entsprechenden Ausgabe reagieren, sowie Verbindungen zwischen den Neuronen. Auch das Lernprinzip entspricht dem eines Menschen. Das Netz nimmt immer Zahlen zwischen 0 und 1 als Eingabe an und berechnet eine entsprechende Ausgabe. Es erhält anschließend die Information, wie die richtige Lösung gelautet hätte und lernt dann aus seinen Fehlern, indem es gewisse Werte, die in die Berechnung einfließen, anpasst. Analog lernt ein Mensch, indem er ausprobiert, gegebenenfalls scheitert, anschließend die richtige Antwort durch eine externe Quelle erhält und somit aus seinem Fehler lernt. Im Menschlichen Gehirn verknüpfen sich Dabei oft genutzte neuronale Verbindungen stärker und weniger benutzte Verbindungen bauen sich ab[**2**]. Die Verstärkung und der Abbau entsprechen dem Ändern

[1] Diese Definition ist stark vereinfacht. Sie enthält ausschließlich die wesentlichen Komponenten um das menschliche Gehirn mit einem neuronalen Netz vergleichen zu können.

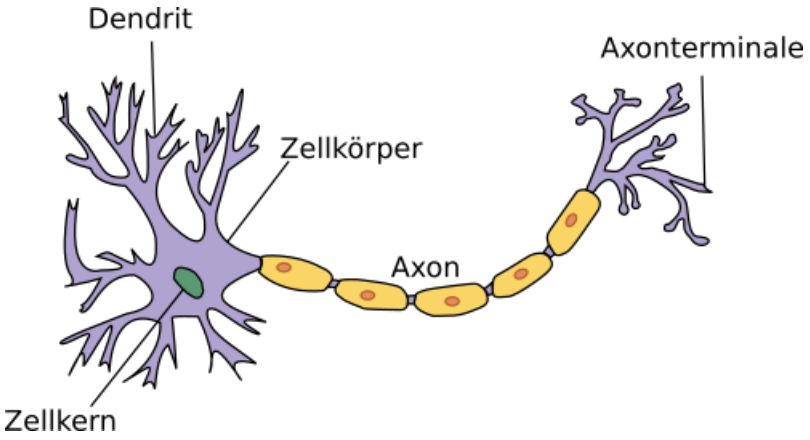

Abbildung 1: Ein Neuron wie es im Gehirn vorliegt

der Gewichtung einer Verbindung im neuronalen Netz. Die Gewichtung ist eine Eigenschaft der Verbindung, die eine zentrale Rolle in der Berechnung spielt und soll im folgenden weiter erläutert werden. Diese Ähnlichkeiten sind kein Zufall, sondern viel mehr Intention. Ein neuronales Netz ist nämlich der gezielte Versuch das menschliche Lernen nachzuahmen um maschinelles Lernen zu ermöglichen.

3.2 Der Aufbau eines neuronalen Netzes

Ein neuronales Netz besteht aus Neuronen und Verbindungen zwischen diesen. Es gibt einen sogenannten „Input Layer", der die Daten, den sogenannten „Input Vector", annimmt, eine beliebige Anzahl von sogenannten „Hidden Layers", in denen das eigentliche Lernen statt findet, und einen sogenannten „Output Layar", der für die Datenausgabe verantwortlich ist. Die Anzahl der Neuronen ist nach oben nicht begrenzt, wird jedoch zumeist der Aufgabe angepasst. Im Input Layer ist meißt ein Neuron pro Pixel des Eingabebildes vorhanden und im Output Layer ein Neuron pro möglicher Ausgabe. Sollen also 28×28 Pixel große Bilder handgeschriebener Ziffern klassifiziert werden, so gibt es 784 Eingabeneuronen, da jedes Bild 784 Pixel groß ist, und 10 Ausgabeneuronen, da es 10 Ziffern gibt. Jedes Neuron hat außerdem eine sogenannte Aktivierungsfunktion, die sich von Neuron zu Neuron unterscheiden kann, und jede Kante eine asoziierte Gewichtung und einen Bias. Ein neuronales Netz besteht also aus:

1. Neuronen mit gegebenenfalls verschiedenen Aktivierungsfunktionen, aufgeteilt in ein Input-, beliebig viele Hidden- und ein Output-Layer.
2. Verbindungen zwischen diesen Neuronen, die jeweils einen eigenen Bias und eine Gewichtung besitzen.

Sind alle Neuronen eines Layers jeweils mit allen Neuronen des nächsten Layers verbunden, wird das Layer als „fully connected layer" bezeichnet.

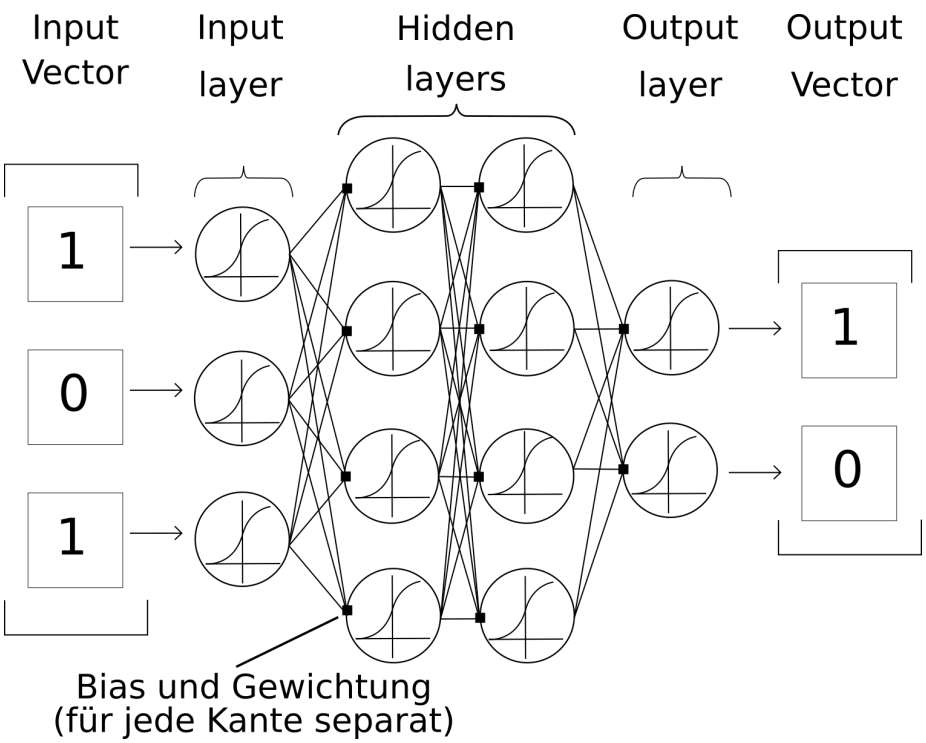

Abbildung 2: Ein einfaches neuronales Netz

3.3 Berechnung des Ausgabevektors

Der Ausgabevektor wird berechnet, indem:

1. Alle Ausgaben aus der vorherigen Schicht mit der Gewichtung der korrespondierenden Kante multipliziert werden
2. Alle gewichteten Eingabewerte summiert werden
3. Der Bias des Neurons hinzuaddiert wird
4. Die Aktivierungsfunktion auf diesen Wert angewandt wird

Die Aktivierungsfunktion hat dabei die Rolle die Werte zu normieren. Sie sorgt also dafür, dass alle Werte innerhalb des Netzes im Intervall $[0, 1]$ bleiben. Es gibt eine Vielzahl von Aktivierungsfunktionen. Die häufigste ist die sogenannte „Sigmoid" Funktion:
Im Gegensatz dazu haben Gewichtungen typischerweise etwa den doppelten Wert der Eingaben. Alle Were werden jedoch automatisch im Lernprozess angepasst.
Der Begriff Eingabe- und Ausgabevektor lassen bereits vermuten, dass es sich bei Neuronalen Netzen um Objekte aus dem Bereich der linearen Algebra handelt. Daher

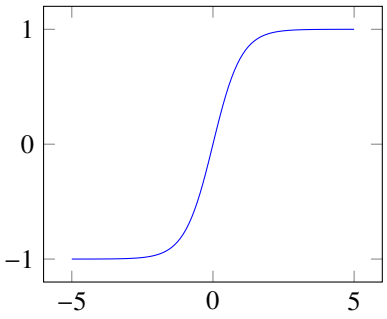

Abbildung 3: Der Plot der Sigmoid Funktion $\sigma(x) = \frac{e^x}{e^x+1}$

wird im Folgenden auch die Notationsweise mit Hilfe von linearer Algebra verwendet. Betrachtet man eine Ausgabe eines Neurons wird diese als $a^{(layer)}_{neuron}$ bezeichnet. Den Ausgabevektor des Input Layers würde man also folgendermaßen schreiben:

$$\begin{bmatrix} a^0_0 \\ a^0_1 \\ a^0_2 \\ \vdots \\ a^0_n \end{bmatrix}$$

Die Gewichtungen w der jeweiligen Kanten werden notiert als $w^{(von\ Layer)}_{(zu\ Neuron,von\ Neuron)}$. „von Layer" bezeichnet dabei das Layer in dem das Neuron liegt, das die Information ausgibt. „zu Neuron" ist der Index des Neurons im nächsten Layer, das die Information annimmt und „von Neuron" der Index des Neurons, das die Information abgibt. Die Gewichtung der Kante, die das zweite Neuron im ersten Layer mit dem dritten Neuron im zweiten Layer verebindet würde also als $w^0_{3,2}$ bezeichnet werden. Dabei wird bei null begonnen zu zählen, sodass das erste Layer und das erste Neuron den Index 0 erhält.
Die Gewichtungen aller Verbindungen eines Layers zum nächsten können also als folgende Matrix geschrieben werden:

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix}$$

Dabei ist n hier die selbe Zahl wie n im Ausgabevektor, da genau so viele Ausgaben vorhanden sein müssen, wie Neuronen in diesem Layer vorhanden sind, da jedes Neuron einen Wert ausgibt.[2]Der Bias Vektor wird genau so wie der Ausgabevektor

[2]Es existieren auch Neuronen, die Daten verwerfen. Diese kommen im hier betrachteten Typ von neuronalem Netz allerdings nicht vor und werden daher der Einfachheit halber außenvor gelassen.

bezeichnet.

$$\begin{bmatrix} b_0^0 \\ b_1^0 \\ b_2^0 \\ \vdots \\ b_n^0 \end{bmatrix}$$

Beachtet man jetzt noch, dass bei jedem Neuron die Aktivierungsfunktion angewandt werden muss ergibt sich folgende Gleichung für die Berechnung des Ausgabevektors $\vec{o}$ aus einem Einbagevektor $\vec{a}$ durch eine Schicht von Neuronen:

$$\vec{o} = \sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^0 \\ a_1^0 \\ a_2^0 \\ \vdots \\ a_n^0 \end{bmatrix} + \begin{bmatrix} b_0^0 \\ b_1^0 \\ b_2^0 \\ \vdots \\ b_n^0 \end{bmatrix}\right)$$

Abbildung 4: Formel zur Berechnung eines Ausgabevektors aus einem Eingabevektor durch ein Layer Neuronen.

Zur Vereinfachung wurde die Funktion hier auf den gesamten Ausgabevektor angewandt. Dies ist korrekt, sofern alle Neuronen eines Layers die selbe Aktivierungsfunktion aufweisen. Dies muss natürlich nicht immer so sein. Sind die Aktivierungsfunktionen der Neuronen eines Layers verschieden, so wird die Aktivierungsfunktion des jeweiligen Neurones separat auf das korrespondierende Element des Vektors $W \cdot \vec{a} + \vec{b}$ angewandt.

3.4 Der Lernprozess

Der Lernprozess gliedert sich in wenige wesentliche Schritte. Zuerst wird unter Verwendung des oben beschriebenen Prozesses aus einem Eingabevektor ein Ausgabevektor berechnet. Diese Berechnung wird im Lernprozess extrem oft durchgeführt, weshalb sich neuronale Netze besonders schnell auf Grafikkarten trainieren lassen. Diese sind für mathematische Operationen im Bereich der linearen Algebra, wie Matritzenmultiplikation oder Addition optimiert und werden daher auch als Vektorprozessoren bezeichnet.
Dieser Ausgabevektor wird nun, mit Hilfe einer Fehlerfunktion, mit dem erwarteten Ausgabevektor verglichen. Je größer dabei die Differenz zwischen erwartetem Ausgabevektor und tatsächlichem Ausgabevektor ist, desto größer ist der Wert der Fehlerfunktion. Der Ausgabewert dieser Fehlerfunktion wird als „Fehler" oder auch als „Kosten" bezeichnet. Wenn also das Minimum dieser Fehlerfunktion bestimmt wird, wird der Fehler minimiert und die tatsächliche Ausgabe des Netzes nähert sich der korrekten Ausgabe immer weiter an.
Eine Methode, die hier erläutert werden soll, dieses Minimum zu finden ist das Gradientenverfahren. Nachdem mit Hilfe dieses Verfahrens der Fehler mimnimiert wurde,

werden die Parameter, also die Gewichtungen und Biases, des neuronalen Netzes entsprechend angepasst. Diesen Prozess der Fehlerminimierung mittels des Gradientenverfahrens und der anschließenden Anpassung der Werte bezeichnet man auch als „Backpropagation". Es existieren auch noch andere Verfahren zur Fehlerminimierung, der Einfachheit halber soll hier aber nur Backpropagation erläutert werden.

3.5 Fehlerfunktionen

Es existiert eine Vielzahl von Fehlerfunktionen, die alle für unterschiedliche Anwendungsgebiete unterschiedlich passend sind. Im Groben lassen sich allerdings Fehlerfunktionen, die für Klassifizierungsprobleme geeignet sind von solchen unterscheiden, die für Regressionsprobleme geeignet sind.

3.5.1 MSE – Durchschnittlicher quadratischer Fehler

Der sogenannte durchschnittliche quadratische Fehler ist eine häufig genutzte Fehlerfunktion für Regressionsprobleme. Die englische Bezeichnung lautet „Mean squared error", woraus sich auch die Abkürzung „MSE loss" ergibt. Sie ist wie in Abbildung 5 dargestellt, definiert.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Abbildung 5: Die Gleichung für den durchschnittlichen quadratischen Fehler

Wie der Name vermuten lässt, gibt diese Fehlerfunktion den Durchschnitt der quadrierten Differenzen zwischen dem vorausgesagten und dem tatsächlichen Ergebnis an. Aufgrund der Quadrierung des Fehlers, werden durch diese Funktion stark abweichende Werte wesentlich stärker gewichtet, als weniger stark abweichende Werte. Ihr Gradient ist außerdem einfach berechenbar, was für das Gradientenverfahren später relevant ist.[**3**]

3.5.2 MAE – Durchschnitztlicher absoluter Fehler

Bei dem durchschnittlichen absoluten Fehler handelt es sich ebenfalls um eine Fehlerfunktion, die für Regressionsprobleme eingesetzt wird. Die englische Bezeichnung lautet „Mean absolute error". Sie ist ähnlich wie der durchschnittliche quadratische Fehler definiert.

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}$$

Abbildung 6: Die Gleichung für den durchschnittlichen absoluten Fehler

Auch hier wird die „Richtung" des Fehlers, in diesem Fall durch die Normierung, verworfen. Außerdem ist diese Fehlerfunktion nicht so anfällig gegenüber Ausreißern in den Daten, da dieser Fehler nicht quadriert wird. Ein Nachteil des durchschnittlichen absoluten Fehlers ist allerdings die höhere Komplexität zur Berechnung des Gradienten.[**3**]

3.5.3 Kreuzentropiefehler

Der Kreuzentropiefehler ist die am häufigsten verwendete Fehlerfunktion für Klassifizierungsprobleme. Sie gibt den Fehler für eine Klassifizierung an, die den gegebenen Klassen Wahrscheinlichkeiten im Intervall $I = [0; 1]$ zuordnet. Dabei steigt der Fehler stärker, je weiter sich die Vorhersage vom tatsächlichen Wert entfernt. Wie aus Abbildung 7 hervorgeht, wird also sicheren, aber falschen Vorhersagen der höhste Fehlerwert zugeordnet.

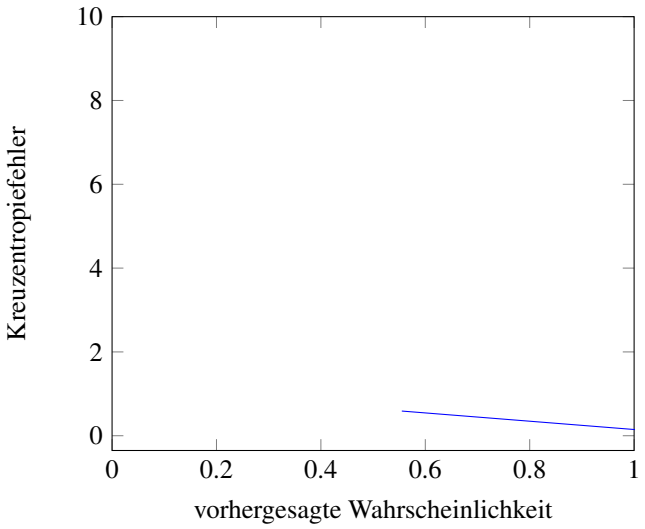

Abbildung 7: Der Graph der Kreuzentropie Fehlerfunktion wenn das tatsächliche Label 1 ist

Der Fehler steigt also mit zunehmender Abweichung der Vorhersage zum tatsächlichen Label rapide an.
Mathematisch ist der Kreuzentropiefehler nach der Funktion in Abbildung 8 definiert, wobei y einen Binärindikator darstellt, der angibt ob das zu klassifizierende Objekt tatsächlich zur Klasse gehört (dann ist er 1) und p die vorausgesagte Wahrscheinlichkeit ob das Objekt zur Klasse gehört, beschreibt.
Hier fällt auf, dass, falls das Label 0 ist, der linke Teil der Gleichung weg fällt und falls es 1 ist, der Rechte. Wenn berechnetes und tatsächliches Label identisch sind, ist der Fehler stets 0.

$$CrossEntropyLoss = -(yln(p) + (1 - y)ln(1 - p))$$

Abbildung 8: Die Gleichung für den Kreuzentropiefehler

Existieren mehr als 2 Klassen, handelt es sich also nicht mehr um eine Binärklassifizierung, müssen die Fehler nach der Gleichung in Abbildung 9 summiert werden.

$$CrossEntropyLoss(M) = -\sum_{c=1}^{M} y_{o,c} ln(p_{o,c})$$

Abbildung 9: Die Gleichung für den durchschnittlichen absoluten Fehler

Dabei gibt M die Anzahl der Klassen an, c das Label für die Klasse und o die berechnete Klassifizierung für diese Klasse.

3.6 Gradientenverfahren und Backpropagation

Das Gradientenverfahren ist ein Verfahren um das Minimum einer Funktion zu finden. Die Funktion, deren Minimum gefunden werden soll ist in diesem Fall die Fehlerfunktion. Diese ist von allen Gewichtungen und Biases des Netzwerkes abhängig, da sie direkt vom Ausgabevektor des Netzes abhängig ist. Der Gradient dieser Funktion ist in Abbildung 10 dargestellt.

$$\nabla C(w_1, b_1, \ldots, w_n, b_n) = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial b_1} \\ \vdots \\ \frac{\partial C}{\partial w_n} \\ \frac{\partial C}{\partial b_n} \end{bmatrix}$$

Abbildung 10: Die Gleichung für den Gradienten der Fehlerfunktion

Um also das Ergebnis „richtiger" zu machen, müssen alle Gewichtungen und Biases negativ zu diesem Gradienten angepasst werden, da der Gradient ja den Hochpunkt angibt.
Diese Anpassung erfolgt, indem das Netz vom Ausgabelayer an, deshalb heißt das Verfahren Backpropagation, durchgegangen wird, und die Gewichtungen und Biases angepasst werden. Dies geschieht in der Regel mit hilfe der „Lernrate" η, mit der die Änderung nach der Formel in Abbildung 11 berechnet wird.
Diese Lernrate ist notwendig um nicht über das Minimum „hinweg zu springen". Sollte sie zu groß sein, passiert genau dies, da die Anpassungen der Parameter in zu großen Schritten erfolgt, sollte sie hingegen zu klein sein, lernt das Netz sehr langsam. Typische Werte sind abhängig von der zu erlernenden Aufgabe, liegen jedoch in der Regel bei etwa 0.01 bis 0.00001

$$w^n_{neu} = w^n_{alt} - \eta \times \frac{\partial C}{\partial w^n}$$

Abbildung 11: Die Gleichung für die Anpassung eines einzelnen Parameters

3.7 Verschiedene Layerarten

3.7.1 Fully connected Layers

3.7.2 Convolutional Layers

3.7.3 Pooling Layers

4 PyTorch

4.1 Datenvorbereitung

4.2 Definieren des Netzes

4.3 Trainieren des Netzes

5 Fallbeispiel I: Ein Klassifizierungsnetzwerk für handgeschriebene Ziffern

5.1 Aufgabe

5.2 Der MNIST Datensatz

5.3 Fragmentbasierte Erkennung

5.4 Ergebnis

6 Fallbeispiel II: Eine selbsttrainierende KI für Tic-Tac-Toe

6.1 Das Prinzip

6.2 Chance-Tree Optimierung

6.3 Lösung mittels eines neuronalen Netzes

6.4 Vergleich

7 Schlusswort

Literatur

[1] Hands-On Machine Learning with Scikit-Learn and TensorFlow
von Aurélien Géron
Veröffentlicht: March 2017 O'Reilly Media, Inc
ISBN: 9781491962282

[2] Die Logistik des Lernens eine Studie der LMU München
Quelle: www.uni-muenchen.de/forschung/news/2013/
f-71-13_kiebler_nervenzellen.html –abgerufen am 16.11.2019

[3] Common Loss functions in machine learning
Von Ravindra Parmar
Veröffentlicht am 02.09.2018, abgerufen am 07.01.2020
Quelle: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

Abbildungsverzeichnis