Grundbegriffe des maschinellen Lernens

Clemens Dautermann

21. 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". Dies wird für die zwei Hauptproblemklassen, die man unterscheidet, verwendet. Klassifizierungs und Regressionsprobleme.

1.1 Klassifizierungsprobleme

Als Klassifizierfung bezeichnet man das Finden einer Funktion, die eine Menge von Eingabevariablen zu einer diskreten Menge von Ausgabevariablen, die auch als Klassen oder Labels bezeichnet werden, zuordnet. Dies kann beispielsweise das Erkennen von Mail Spam sein. Die Eingabevariablen sind die E-Mails und sie sollen den zwei Klassen „Spam" und „nicht Spam" zugeortnet werden.
Das in Dieser Arbeit gegebene Beispiel ist auch ein Klassifizierungsproblem. Die gegebenen Bilder von Ziffern sollen den zehn Klassen „0 bis 9" zugeordnet werden. Die Bilder sind hier die Eingabevariablen und die Klassen null bis neun beschreibt die endliche Menge diskreter Labels.
Das Erste Beispiel würde man als „Binärklassifizierung" bezeichnen, da zwei Klassen

unterschieden werden. Letzteres wird als „Multiklassenklassifizierung" bezeichnet, da mehr als zwei Klassen unterschieden werden. Die Binärklassifizierung ist in Abbildung 1 verbildlicht.

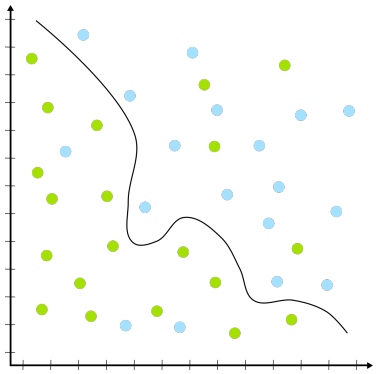

Abbildung 1: Binärklassifizierung

Die zwei Klassen wären hier „grün" und „blau". Die Linie stellt die Klassengrenze dar, die die zwei Klassen unterscheidet. Es sind außerdem einige Ausreißer in den Daten vorhanden.

1.2 Regressionsprobleme

Als Regressionsproblem hingegen bezeichnet man das Finden einer Funktion, die eine Menge von Eingabevariablen einer Stetigen Menge von Ausgabevariablen zuordnet. Wenn beispielsweise ein Bild eines Menschen gegeben ist, könnte ein Regressionsproblem sein, seine Höhe oder sein Gewicht zu bestimmen.

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 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.

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

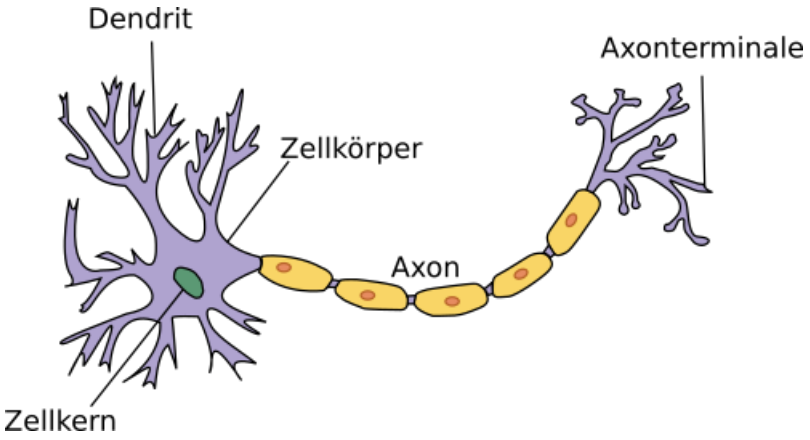

Abbildung 2: Ein Neuron wie es im Gehirn vorliegt

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.

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

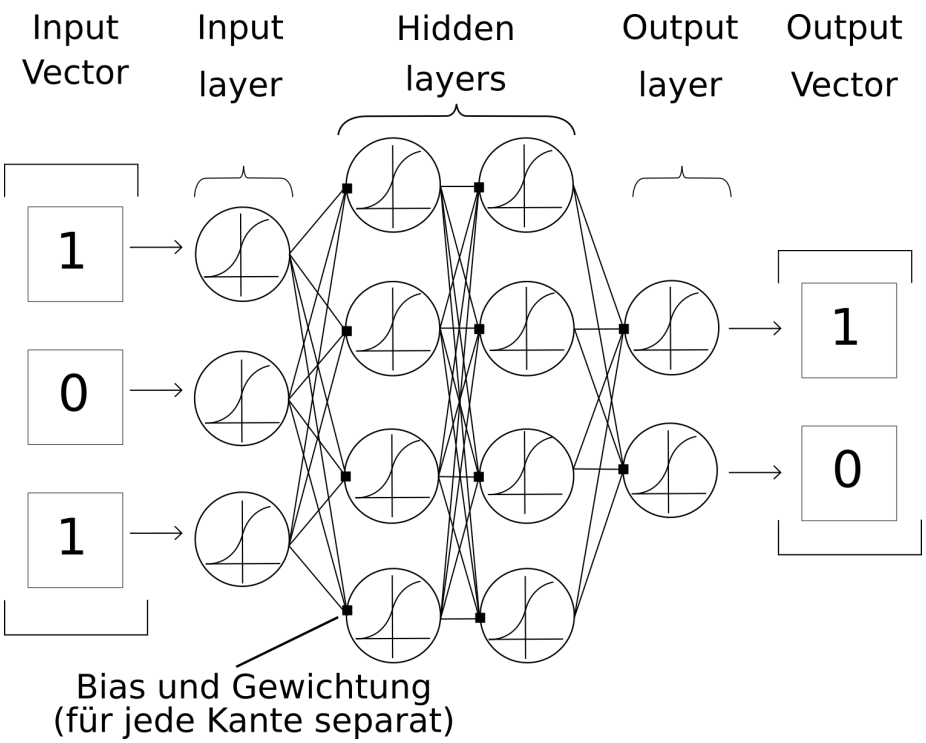

Abbildung 3: Ein einfaches neuronales Netz

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 wird im Folgenden auch die Notationsweise mit Hilfe von linearer Algebra verwendet. Betrachtet man eine Ausgabe eines Neurons wird diese als $a_{neuron}^{(layer)}$ bezeichnet.

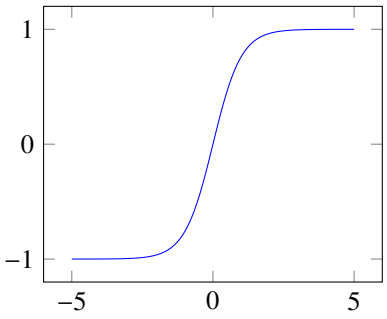

Abbildung 4: Der Plot der Sigmoid Funktion $\sigma(x) = \frac{e^x}{e^x+1}$

Den Ausgabevektor des Input Layers würde man also folgendermaßen schreiben:

$$\begin{bmatrix} a_0^0 \\ a_1^0 \\ a_2^0 \\ \vdots \\ a_n^0 \end{bmatrix}$$

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.

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

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:

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

Abbildung 5: 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 6 dargestellt, definiert.

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

Abbildung 6: 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 7: 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 8 hervorgeht, wird also sicheren, aber falschen Vorhersagen der höhste Fehlerwert zugeordnet.

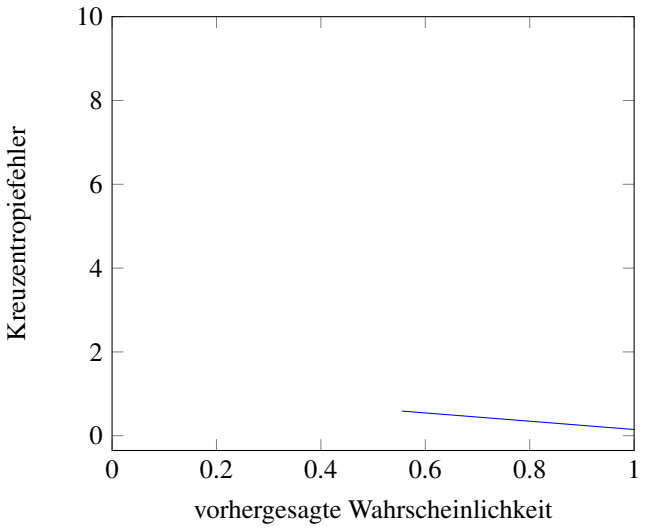

Abbildung 8: Der Graph der Kreuzentropie Fehlerfunktion wenn das tatsächliche Label 1 ist

Der Fehler steigt also mit zunehmender Abweichung der Vorhersage zum tatsächlichen Label rapide an.
Mathematisch ist der Kreuzentropiefehler nach der Funktion in Abbildung 9 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 9: 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 10 summiert werden.

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

Abbildung 10: 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 11 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 11: 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.
Oft wird zur Verbildlichung des Gradientenverfahrens die Analogie eines Balles verwendet, der einen Hügel hinunter rollt. Er findet den Tiefpunkt indem er hinab rollt und dabei immer automatisch eine Kraft nach unten wirkt.

3.6.1 Lernrate

Eine wichtige Rolle dabei spielt die sogenannte „Lernrate" η, mit der die Änderung nach der Formel in Abbildung 12 berechnet wird.

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

Abbildung 12: Die Gleichung für die Anpassung eines einzelnen Parameters

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 0.0001 [3].

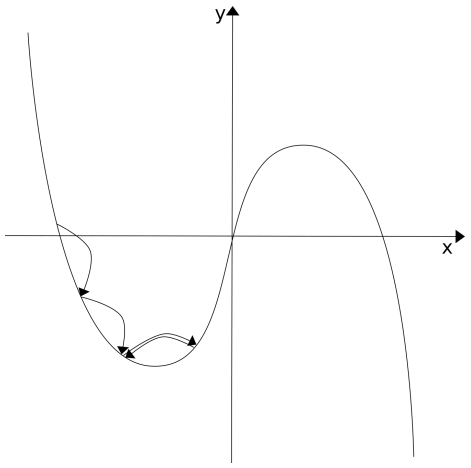

Abbildung 13: η ist hier zu groß gewählt

Abbildung 13 stellt dar, wieso das Minimum nicht erreicht werden kann, falls die Lernrate zu groß gewählt wurde. Es ist zu sehen, dass der Parameter immer gleich viel geändert wird und dabei das Minimum übersprungen wird, da die Lernrate konstant zu groß ist. Dieses Problem kann behoben werden indem eine adaptive Lernrate verwendet wird. Dabei verringert sich die Lernrate im Laufe des Lernprozesses, sodass zu Beginn die Vorzüge des schnellen Lernens genutzt werden können und am Ende trotzdem ein hoher Grad an Präzision erreicht werden kann.

[3] Dies ist ein bloßer Erfahrungswert. Maschinelles Lernen erfordert oft sehr viele Versuche, weshalb nicht genau festgelegt werden kann, wann welche Lernrate optimal ist.

3.7 Verschiedene Layerarten

Mit Hilfe von maschinellem Lernen lassen sich eine Vielzahl von Aufgaben bewältigen. Entsprechend komplex müssen Neuronale Netze aber auch sein. Demzufolge ist es notwendig, Neuronen zu entwickeln, die andere Fähigkeiten aufweisenl, als das einfache oben im sogenannten „Linear Layer" verwendete Neuron. Da man in der Regel nur eine Art von Neuron in einem Layer verwendet, wird das gesamte Layer nach der verwendeten Neuronenart benannt. Die unten beschriebenen Layerarten werden vor allem in einer Klasse von neuronalen Netzen verwendet, die als „Convolutional neural networks" bezeichnet werden. Sie werden meißt im Bereich der komplexen fragmentbasierten Bilderkennung eingesetzt, da sie besonders gut geeignet sind um Kanten oder gewisse Teile eines Bildes, wie zum Beispiel Merkmale eines Gesichtes, zu erkennen.

3.7.1 Convolutional Layers

Convolutional Layers weisen eine fundamental andere Funktionsweise als lineare Layers auf. Sie nehmen zwar ebenfalls rationale Zahlen an und geben rationale Zahlen aus [4], berechnen die Ausgabe jedoch nicht nur mit Hilfe einer Aktivierungsfunktion sondern unter der Verwendung sogenannter „Filter". Diese Filter sind eine $m \times n$ große Matrix, die auch als „Kernel" bezeichnet wird. Der Kernel wird dabei über die Eingabematrix bewegt (daher der Name convolution) und erzeugt eine Ausgabematrix. Dafür wird der betrachtete Abschnitt der Eingabematrix A und des Kernels B skalar multipliziert wobei das Skalarprodukt als Frobenius-Skalarprodukt, also als

$$\langle A, B \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} b_{ij}$$

definiert ist. Die Matritzen werden also Komponentenweise multipliziert und diese Produkte dann summiert.
Dies ist in Abbildung 14 verbildlicht.

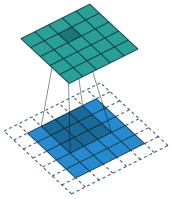

Abbildung 14: Eine Verbildlichung der Vorgänge in einem convolutional Layer. Das blaue Raster stellt die Eingabe dar, das grüne die Ausgabe.

[4] Im Folgenden werden 2 Dimensionale convolutional Layers betrachtet, da diese einfacher vorstellbar sind. Sie nehmen dann eine Matrix rationaler Zahlen an und geben auch eine Matrix rationaler Zahlen aus. Dies korrespondiert mit dem Anwendungsbereich der Erkennung von schwarz weiß Bildern.

Ein Filter kann ganz verschiedene Werte aufweisen. So können Filter der Form

$$\begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Abbildung 15: Erkennt obere horizontale Kanten

$$\begin{bmatrix} -1 & 1 & 0 \\ -1 & 1 & 0 \\ -1 & 1 & 0 \end{bmatrix}$$

Abbildung 16: Erkennt linke vertikale Kanten

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}$$

Abbildung 17: Erkennt untere horizontale Kanten

$$\begin{bmatrix} 0 & 1 & -1 \\ 0 & 1 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

Abbildung 18: Erkennt rechte vertikale Kanten

beispielsweise zur einfachen Kantenerkennung genutzt werden. Zur Veranschaulichung wurden diese Filter auf das Beispielbild in Abbildung 19 angewandt. Das Ergebnis ist in Abbildung 20 dargestellt.

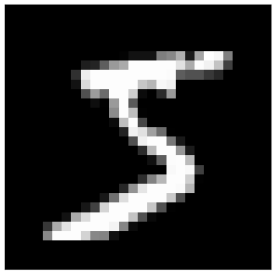

Abbildung 19: Das Beispielbild aus dem Mnist Datensatz

Abbildung 20: Die jeweils oben stehenden Filter wurden auf das Beispielbild angewandt.

Der jeweils dunkel dargestellte Bereich kann als das identifiziert werden, was vom convolutional Layer als Kante erkannt wurde. Hier werden eindeutige Limitationen deutlich: Es kann nur als Kante erkannt werden, was auch eindeutig senkrecht oder waagerecht ist. Außerdem kann es zu Fehlentscheidungen kommen.

Die Kernels werden natürlich nicht per Hand initialisiert und angepasst, sondern setzen sich aus Parameternm zusammen, die im Laufe des Lernprozesses durch das Netz anpassbar sind. Das Netz kann also die Filtermatrix selber verändern. Die Filter werden meißt mit Zufallswerten initialisiert und dann während des Lernens angepasst. Ferner muss ein Kernel auch nicht immer drei Einheiten breit sein, sondern kann jede Größe ≥ 2 annehmen. Je nachdem, wie sich der Kernel über die Eingabematrix bewegt, ist außerdem ein sogenanntes „Padding" nötig, da gegebenenfalls Werte betrachtet werden müssten, die nicht in der Eingabematrix liegen. In der Regel werden daher alle Werte, die nicht in der Eingabematrix vorhanden sind durch 0 ersetzt. Das Padding ist in Abbildung 14 als weiß in der Eingabematrix dargestellt. Es ist eine Art „Rand aus Nullen", der um dasd Bild gelegt wird.

Hintereinander können convolutional Layers auch ganze Elemente eines Bildes erkennen. Erkennt das erste Layer wie oben gezeigt beispielsweise Kanten, so kann das Layer darauf Kombinationen aus diesen, wie beispielsweise Ecken oder Winkel, erkennen. Wie gefilterte Bilder für sogenannte „High-Level-Features" aussehen können ist in Abbilldung 21 dargestellt. Die Ausgabebilder von Convolutional Layers werden als „Feature map" bezeichnet.

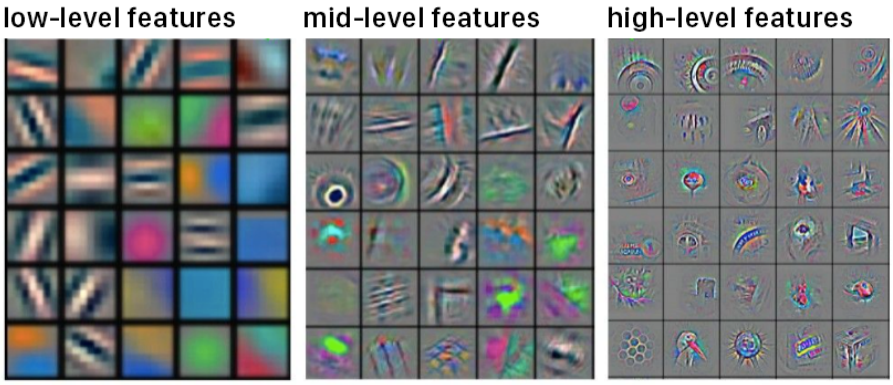

Abbildung 21: Beispiele für low- mid- und high-level Features in Convolutional Neural Nets

Das bemerkenswerte an Convolutional Layers ist vor allem, dass duch ähnliche Optimierungsalgorytmen auch hier maschinelles lernen möglich ist, dass sich ein neuronales Netz diese Filter also selbstständig beibringen kann.

3.7.2 Pooling Layers

Pooling Layers werden ebenfalls hauptsächlich in Convolutional Neural Networks verwendet. Sie werden nach Convolutional Layers genutzt um das Ausgabebild herunterzutakten, also verlustbehaftet zu Komprimieren. Dabei wird die Feature Map im wesentlichen zusammengefasst um die Datenmenge, die das Folgende Convolutional

Layer erhält zu reduzieren und sie Verschiebungen im Originalbild gegenüber immuner zu machen. Das ist deshalb notwendig, da die Convolutional Layers die Features lokal sehr begrenzt erkennen und daher eine kleine Verschiebung des Originalbildes zur Folge haben kann, dass im Folgenden Convolutional Layer die Kombination der Features gegebenenfalls nicht richtig erkannt wird. Das Pooling Layer kann diesem Effekt entgegenwirken, indem es die Features zusammenfasst. So kann aus einer ganzen Kante beispielsweise ein einziger Pixel werden.
Es werden im Wesentlichen zwei Techniken zum Pooling eingesetzt.

1. Max Pooling (Abbildung 22)
2. Average Pooling (Abbildung 23)

Sie unterscheiden sich darin, wie die zu komprimierenden Werte mit einander verrechnet werden, sind ansonsten jedoch identisch.
Beim Pooling wird die Eingabematrix in Submatritzen partitioniert[5]. Jede Submatrix stellt später einen Pixel in der Ausgabematrix dar. Hier unterscheiden sich jetzt Max- und Avarage- Pooling. Beim Max Pooling ist der neue Wert der höhste Wert aus dieser Submatrix, beim Avarage Pooling wird der Durchschnitt aller Werte Der Submatrix gebildet und als Neuer Wert verwendet.

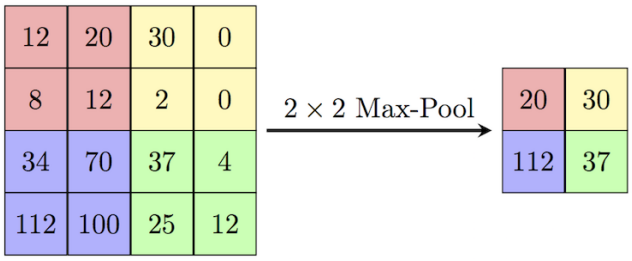

Abbildung 22: Max Pooling mit 2×2 großen Submatritzen

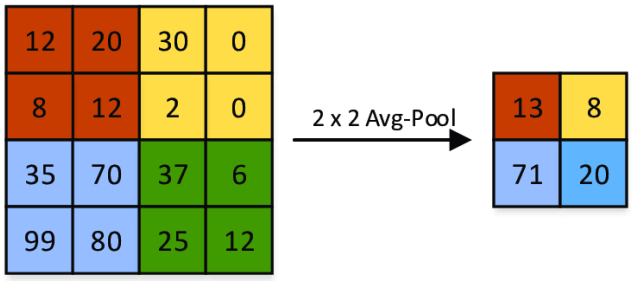

Abbildung 23: Average Pooling mit 2×2 großen Submatritzen

[5]Hier ist die Mengenteorethische Partitionierung gemeint. Eine Menge wird in nicht leere Teilmengen unterteilt, sodass jedes Element der Ausgangsmenge in genau einer der Teilmengen enthalten ist.

Die Dimension der Submatritzen beträgt meißt 2×2. In Abbildung 24 ist dargestellt, wie Pooling konkret auf das im letzten Abschnitt beschriebene Bild abgewandt aussieht. Dafür sind in der ersten Zeile die 28×28 Pixel großen Bilder dargestellt, die das Convolutional Layer mit Hilfe der Kantenerkennungsfilter berechnet hat. In Zeile zwei wurde auf die jeweils darüber stehenden Bilder Max Pooling angewandt, in Zeile drei auf die selben Bilder Average Pooling. Die Bilder sind nach dem Pooling 14×14 Pixel groß.

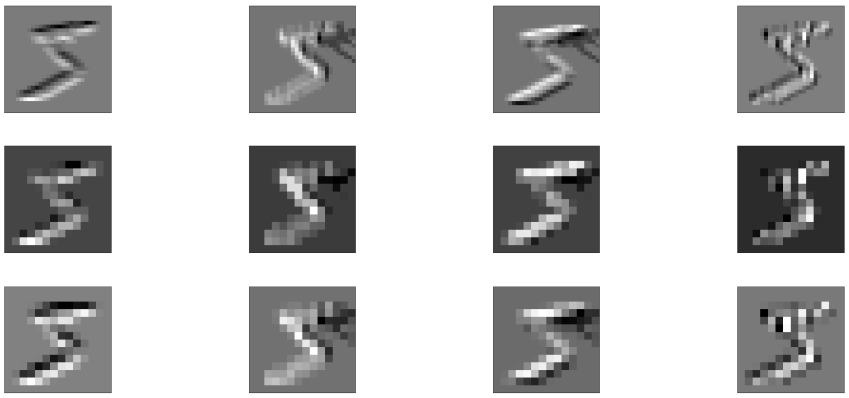

Abbildung 24: Gegenüberstellung von Max und Average Pooling

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