Neuronale Netzwerke

Besondere Lernleistung

Rishab Garg

05. April 2024

Betreut durch Frau Marinescu
Fachbereich Informatik
Gymnasium Buckhorn

Inhaltsverzeichnis

Hinweise

Der Source-Code für das Projekt, sowie der Code für das LaTeX-Dokument dieser schriftlichen Ausarbeitung sind unter `https://github.com/RisGar/bll` verfügbar.

Die Notation hält sich weitestgehend an die Standards der „Beijing Academy of Artificial Intelligence“, verfügbar unter `https://ctan.math.utah.edu/ctan/tex-archive/macros/latex/contrib/mlmath/mlmath.pdf` und des Stanford University CS20-Kurs verfügbar unter `https://cs230.stanford.edu/files/Notation.pdf`, sollte jedoch unabhängig von diesen verständlich sein.

Da der Buchstabe x bereits von diesen Standards genutzt wird, wird als unabhängige Variable der Buchstabe z benutzt, dieser repräsentiert jedoch, anders als üblich, keine komplexe Zahl.

1 Einleitung

In unserer heutigen Welt spielt Künstliche Intelligenz, kurz *KI*, eine immer wichtigere Rolle, vor allem durch Chatbots wie ChatGPT, welche in unterschiedlichsten Bereichen ihren Nutzen finden. Sie werden oftmals als ein „magisches Werkzeug“ gesehen, welches aus jeglicher Aufforderung eine Antwort „herzaubert“, aber wovon niemand wirklich versteht, wie oder warum es funktioniert.

Genau diese „Magie“ hat dieses Thema für mich seit Jahren interessant gemacht. Frühere Versuche, mich mit der genauen Funktionsweise von neuronalen Netzwerken, auf welchen diese fortschrittlichen KIs beruhen, haben mich jedoch aufgrund von fehlendem Grundwissen nicht weiter gebracht.

Aus diesem Grund habe ich mir im Rahmen dieser *besonderen Lernleistung* vorgenommen, mich als Jahresprojekt an das Thema wieder heranzuwagen und unter der Leitfrage **„Wie kann man den Lernprozess eines neuronalen Netzwerkes mathematisch darstellen?“**, die Funktionsweise und den Lernprozess eines neuronalen Netzwerks zu verstehen sowie ein eigenes zu programmieren.

2 Neuronale Netzwerke

„Funktionen beschreiben unsere Welt.“

—Thomas Garrity, *On Mathematical Maturity*

Ähnliche Aussagen hatte ich im Laufe der Jahre im Physikunterricht immer wieder gehört. Sämtliche Aspekte aus unserem Leben wären durch Funktionen beschreibbar, sei es der Strom, der durch unsere Steckdose fließt oder der Schall, den wir mit unseren Ohren hören [1]. Neuronale Netzwerke bringen diesen Satz jedoch auf ein neues Extrem.

Ein künstliches neuronales Netz, kurz *ANN* (von dem englischen „Artificial Neural Network“) ist ein mathematisches Modell der inter- und intraneuronalen Vorgänge in einem biologischen, also „natürlichen“ neuronalen Netz, wie es auch im Gehirn eines Menschen vorzufinden ist [2, S. 3].

Ein biologisches neuronales Netz hat seinen Namen davon, dass es eine Vernetzung von Zellen, genannt Neuronen, ist, welche mit Synapsen miteinander verknüpft sind. Sie kommunizieren lediglich durch elektrische Impulse und jedes Neuron kann basierend auf der Stärke der Impulse der verknüpften Zellen selbst „entscheiden“ ob es einen Impuls weiterleitet [3, S. 1]. Diese Separierbarkeit von einzelnen Neuronen lässt sie unahängig von einem ganzen Netz, als *künstliche* Neuronen, im Laufe dieses Textes einfach halber nur als „Neuronen“ bezeichnet, mathematisch als eine Funktion modellieren [2, S. 3]. Das ganze ANN ist also eine Verknüpfung einzelner Funktionen, was es in seiner Gesamtheit auch zu einer Funktion macht. Es ergibt sich also die Frage, wie eine Funktion komplexe Probleme aus unserer Welt lösen kann.

2.1 Das McCulloch-Pitts-Neuron

Das erste mathematische Modell eines biologischen Neurons ist das McCulloch-Pitts-Neuron von Warren McCulloch und Walter Pitts. In ihrer Publikation [3], veröffentlicht in 1943, beschreiben sie ein Modell, welches mit binären Signalen arbeitet, basierend auf dem Alles-oder-nichts-Gesetz, welches besagt, dass ein Neuron bei dem Erreichen eines bestimmten Aktionspotenzials, also einer bestimmten

Schwelle, mit maximaler Stärke ausgelöst wird. Das Neuron kann also eine Ausgabe a von 1 für ausgelöst oder 0 für nicht ausgelöst vorhersagen [4].

Neuronen bestehen in diesem Modell aus zwei Teilen: Die im vorigen Absatz erwähnte Schwellenfunktion f, welche ein Signal ausgeben kann sowie eine Summation $\sum$ der Eingangs-Impulse $\vec{x}$. Diese werden als Eingangsmenge in Form eines Vektors $\vec{x}$ dargestellt, wobei x_i der i-te Eingangs-Impuls ist. Dabei kann ein solcher Impuls, wie die Ausgabe a, nur 0 oder 1 sein [4].

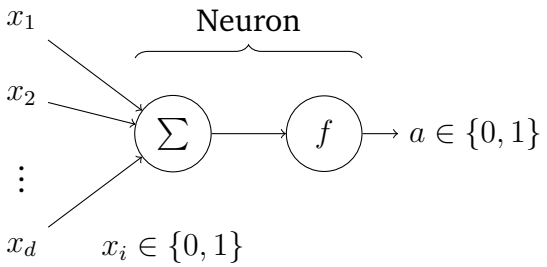

Abbildung 1: Funktionsweise eines McCulloch-Pitts-Neurons

Eine solche Schwellenfunktion mit einem Sprung von 0 zu 1 lässt sich als Verschiebung der Einheitsstufenfunktion Θ, auch *Heaviside-Funktion* genannt, auch der x-Achse darstellen.

$$\Theta(z) = \begin{cases} 1 & \text{für } z \geq 0 \\ 0 & \text{für } z < 0 \end{cases}$$

$$\Theta(z - n) = \begin{cases} 1 & \text{für } z \geq n \\ 0 & \text{für } z < n \end{cases}$$

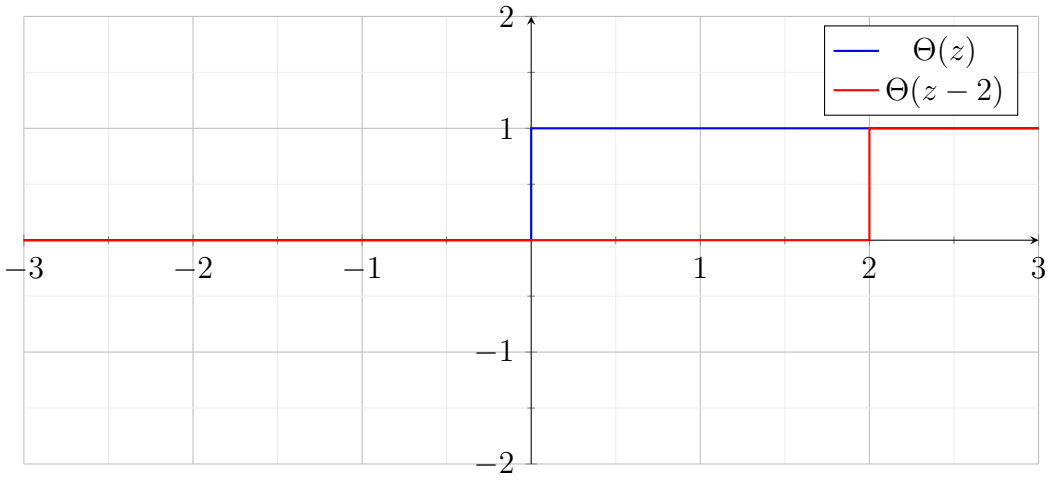

Abbildung 2: Einheitsstufenfunktion mit und ohne Verschiebung

Die vorhergesagte Ausgabe a des Neurons mit einer Eingabe $\vec{x}$ der Dimension d und dem Schwellenwert, genannt *Bias* b und einheitlicher Weise das Negativ des eigentlichen Schwellenwerts, ist also durch folgende Formel definiert:

$$a(\vec{x}) = \Theta\left(\left(\sum_{i=1}^{d} x_i\right) + b\right)$$

Ein Beispiel für die Ausgabe eines McCulloch-Pitts-Neurons mit dem Bias $b = -2$ und der Eingabedimensionalität $d = 5$ ist:

$$\vec{x} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$
$$a(\vec{x}) = \Theta((3) + (-2)) = \Theta(1) = 1$$

2.2 Das Perzeptron

Die offensichtliche Limitation eines McCulloch-Pitts-Neuronens ist, dass es nur mit binären Werten arbeitet. Ein fortgeschritteneres ANN, das *Perzeptron*, über das Frank Rosenblatt in einem in 1957 publizierten Bericht [5] schrieb, überkam diese Limitation und bildet auch heute immer noch die Grundlage von ANNs.

Ein Perzeptron kann entscheiden, ob eine Eingabe einer bestimmten Kategorie, welche als *Klasse* bezeichnet wird, angehört. Es besteht aus mehreren Neuronen,

welche in zwei Schichten aufgeteilt sind: die Eingangs-Schicht und die Ausgangs-Schicht. Die Eingangs-Neuronen übernehmen je eine Zahl aus der Eingangsmenge $\vec{x}$, welche, anders als beim McCulloch-Pitts-Neuron, aus Zahlen aus dem reellen Zahlenbereich $\mathbb{R}$ besteht. Da das Netz mehrere Neuronen in verschiedenen Schichten besitzt, gibt es auch mehrere Ausgänge dieser $a_i^{[l]}$ wobei die Expontentenziffer $[l]$ die aktuelle Schicht darstellt und die Neuronnummer i die Nummer des Neurons darstellt. Die Ausgangs-Neuronen funktionieren insofern wie das McCulloch-Pitts-Neuron, als dass sie die eingehenden Werte der Synapsen summieren und das Ergebnis in die Einheitsstufenfunktion einspeisen. Diese nimmt dann die *Klassifizierung* vor, sagt also aus, ob, wenn das Ergebnis 1 ist, die Eingabe der Klasse angehört, oder, wenn das Ergebnis 0 ist, die Eingabe der Klasse nicht angehört.

Die wichtigste Eigenschaft, die mit der Einführung von reellen Zahlen plausibel wird, ist die individuelle Anpassung jeder Synapse mit Gewichtungen $\vec{w}$. Der Wert, den ein Ausgabe-Neuron von einem Eingabe-Neuron $a_i^{[1]}$ als Signal bekommt, ist nun nicht nur die Ausgabe dessen, sondern wird multipliziert mit der Gewichtung w_i der Synapse, welche die beiden Neuronen verbindet.

Der Bias b wird hier als getrennte Gewichtung für ein Eingabe-Neuron mit dem konstanten Wert 1 implementiert. Wegen des Faktors 1 kann der Bias einfacher Weise, nach der Summierung der Eingangswerte, getrennt addiert werden [6, S. 4–5], [2, S. 3].

Welchen vorhergesagten Ausgangswert $\hat{y}$ ein Perzeptron am Ende für einen bestimmten Eingangswert $\vec{x}$ hat, ist also abhängig von den Gewichtungen und dem Bias. Deswegen bezeichnet man sie als die *Parameter* eines ANNs. Abbildung 3 zeigt ein Modell eines Perzeptrons mit der Eingabedimension $d = 2$ und der Ausgabedimension $d_o = 1$ an.

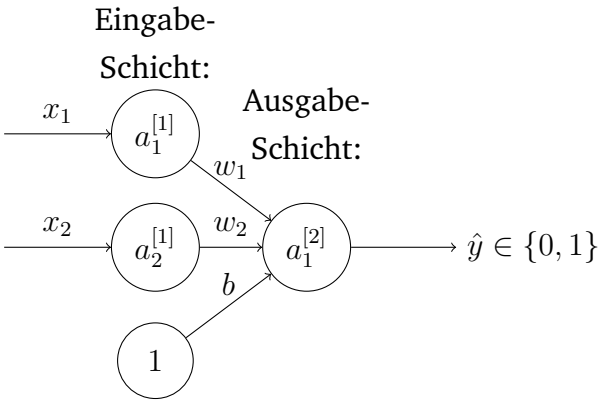

Abbildung 3: Ein Perzeptron

Da die Eingabewerte $\vec{x}$ und die Gewichtungen $\vec{w}$ beide Vektoren sind, kann die Multiplikation der einzelnen Werte auch als Skalarprodukt von den Vektoren angegeben werden:

$$a(\vec{x}) = \Theta\left(\sum_{i=1}^{d} x_i w_i + b\right)$$
$$a(\vec{x}) = \Theta(\vec{x} \cdot \vec{w} + b)$$

Es stellt sich jedoch die Frage, ob mit einem so simplen Aufbau jede Aufgabe lösbar ist. Die Limitation eines Perzeptrons lässt sich gut an einem Perzeptron mit einer zwei-dimensionalen Eingangsmenge $\{x, y\}$ darstellen:

$$\begin{aligned}
a(x, y) &= \Theta(w_1 x + w_2 y + b) \\
0 &\leq w_1 x + w_2 y + b && | - b \\
-b &\leq w_1 x + w_2 y && | : w_2 \\
-\frac{b}{w_2} &\leq \frac{w_1 x}{w_2} + y && | - \frac{w_1 x}{w_2} \\
-\frac{b}{w_2} - \frac{w_1 x}{w_2} &\leq y \\
y &\geq -\frac{w_1}{w_2} x - \frac{b}{w_2}
\end{aligned}$$

Die resultierende Funktionsungleichung ist linear. Für alle Werte über dem Graphen

dieser ist diese Ungleichung wahr, die Ausgabe des Perzeptrons ist also $\hat{y} = 1$ und für alle Werte unter dem Graphen ist sie falsch, die Ausgabe also $\hat{y} = 0$.

Die Limitation mit dem Aufbau eines Perzeptrons ist also, dass es nur lineare Funktionen annehmen kann und deshalb die Eingaben durch eine Hyperebene, im zwei-dimensionalen Raum also eine Gerade, in „nicht der Klasse angehörend“ und „der Klasse angehörend“ aufteilbar sein müssen. Diese Eigenschaft wird *lineare Separierbarkeit* genannt [2, S. 7]. Ein Beispiel für eine linear separierbare Funktion ist das OR-Gatter, zu sehen in Abbildung 4 mit einer separierenden Gerade. Ein Beispiel für eine Funktion, die dies hingegen nicht ist, ist das XOR-Gatter, zu sehen in Abbildung 5, hier lässt sich keine Gerade durch die Werte ziehen, sodass sie in ihre Klassen eingeteilt sind [6, S. 9].

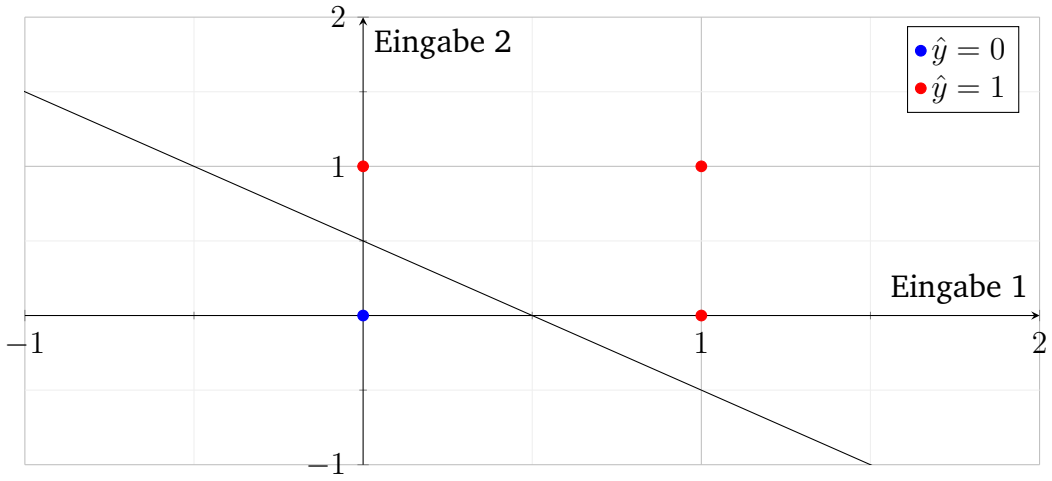

Abbildung 4: Das OR-Gatter

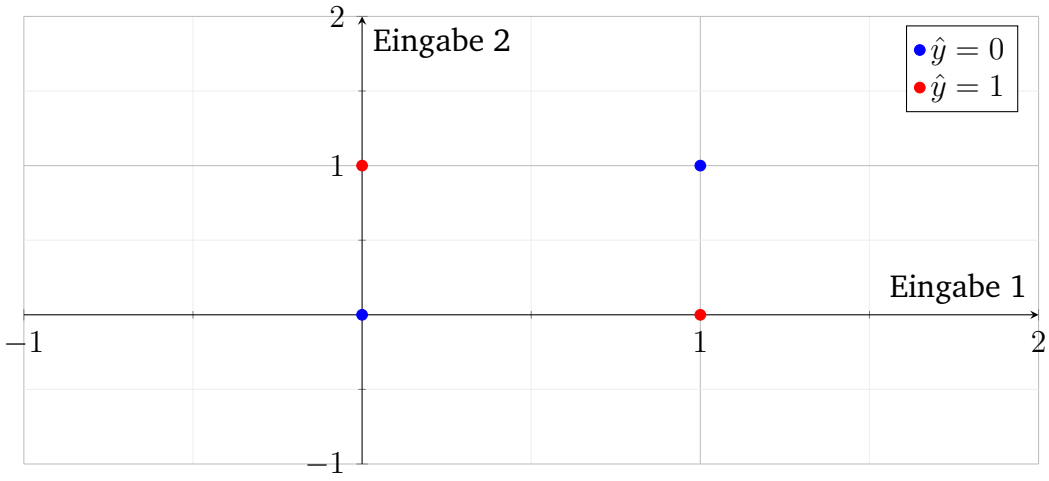

Abbildung 5: Das XOR-Gatter

3 Feedforward-Netzwerke

3.1 Die logistische Funktion

Eine Alternative zur Einheitsstufenfunktion eines Perzeptrons ist die logistische Funktion σ, welche auch Grenzen von 0 und 1 besitzt, aber statt einem Sprung eine S-Kurve, zu sehen in Abbildung 6, besitzt [6, S. 17]. Allgemein werden diese Funktionen, die nach der Summation in einem Neuron verwendet werden, als *Aktivierungsfunktion* bezeichnet. Ein ANN mit einer logistischen Funktion ist nun kein Perzeptron mehr, sondern ein allgemeines *feedforward-Netz*, danach benannt, dass die Signale im Netz vorwärts von einer Schicht zur nächsten transportiert werden. Der Vorteil an dieser Funktion ist, dass man durch einen reellen Ausgabewert sehen kann, wie sicher sich das ANN mit der Vorhersage ist und wie sich das Ändern der Parameter auf die Funktion auswirkt [7, Kapitel 1]. Wie man „richtige“ Parameter bestimmt, wird in Kapitel 4.2 erläutert.

Die Funktionsgleichung der logistischen Funktion ist:

$$\sigma(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}$$

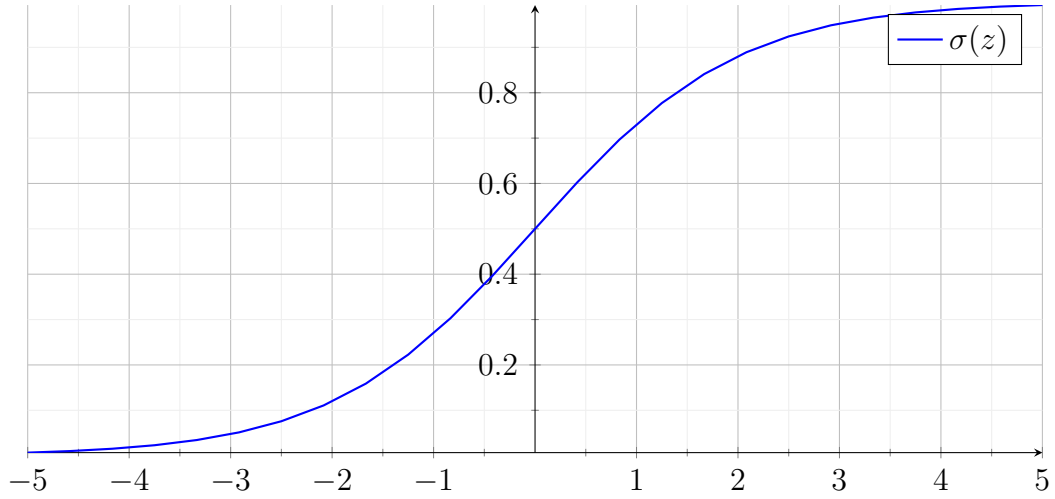

Abbildung 6: logistische Funktion

3.2 Mehrklassige Klassifikation

Ein Perzeptron mit einem Neuron in der Ausgabe-Schicht kann nur zwischen zwei verschiedenen Klassen differenzieren. Ein solcher Algorithmus wird deswegen als *binäre Klassifizierung* bezeichnet [8].

In diesem Fall hat jedes Ausgabe-Neuron eigene Synapsen, welche es wie in der binären Klassifikation mit jedem Neuron in der Eingabe-Schicht verbindet. Es gibt nun also nicht nur so viele Gewichtungen wie die Eingabedimension d, sondern diese Zahl multipliziert mit der Ausgabedimension d_o. Da jeweils das Eingabe- und Ausgabe-Neuron einer Gewichtung wichtig ist, kann sie als w_{ij} dargestellt werden, wobei i das Eingabeneuron $a_i^{[1]}$ und j die Ausgabeneuron $a_j^{[2]}$ bezeichnet. Anstatt eines Gewichtungs-Vektors $\vec{w}$ wird also eine Gewichtungs-Matrix W verwendet:

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d_o} \\ a_{21} & w_{22} & \cdots & w_{2d_o} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dd_o} \end{bmatrix}$$

Da ein Ausgabe-Neuron auch einen Bias braucht, wird hier ein Vektor $\vec{b}$ für die Biase verwendet, wobei b_i der Bias von $a_i^{[2]}$ ist. In der Abbildung 7, sowie den folgenden

Abbildungen, werden die Biase sowie die Bezeichnungen für die Gewichtungen aus Platzgründen weggelassen, sie werden jedoch weiterhin im Modell verwendet.

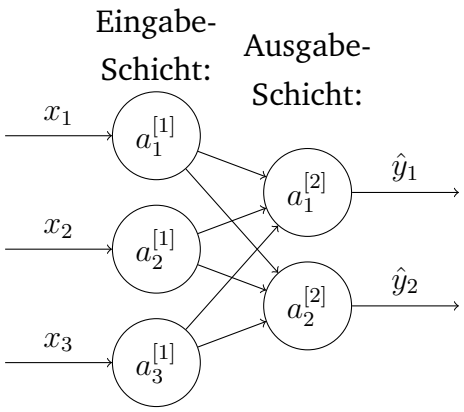

Abbildung 7: Mehrklassiges Perzeptron mit $d_o = 2$

In dem ANN der Abbildung 7 ist zu sehen, dass durch die zwei Ausgabe-Neuronen $a_i^{[2]}$ nun auch zwei Ausgaben gibt, welche den Vektor $\hat{\mathbf{y}}$ bilden, hier nicht mit einem Vektorpfeil notiert, da dieser mit dem Zirkumflex interferiert. Diese Ausgaben könnten verschieden interpretiert werden, zum Beispiel bei einer Einheitsstufenfunktion als eine quaternäre Klassifizierung, wo jede der vier Kombinationen der Ausgaben eine Klasse darstellt.

3.3 Die Softmax-Funktion

Eine einheitliche Benutzung dieser Ausgaben wird mit der Softmax-Funktion ermöglicht. Die Softmax-Funktion ist eine vektorwertige Funktion mit einem Vektor als Argument, was bedeutet, dass sie sowohl ihre Definitions- als auch ihre Zielmenge ein Vektorraum ist, hier mit der Dimension der Ausgaben d_o, da diese verarbeitet werden sollen. Die Funktion konvertiert einen Vektor in eine Wahrscheinlichkeitsverteilung, wobei jedes Element $\hat{y}_i$ Wahrscheinlichkeit ist, mit der das Netzwerk sagt, dass dies die korrekte Klasse sei. Sie kann also als eine logistische Funktion für mehrklassige Klassifikation gesehen werden, sichtbar an der S-Kurve in der Abbildung 8.

Die Ähnlichkeit zur logistischen Funktion sieht man auch an der Funktionsgleichung, bei dieser auch die e-Funktion durch sich selbst geteilt wird, bei der Softmax-Funktion jedoch durch die Summe aller e-Funktionen des Vektors:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{d_o} e^{z_j}}$$

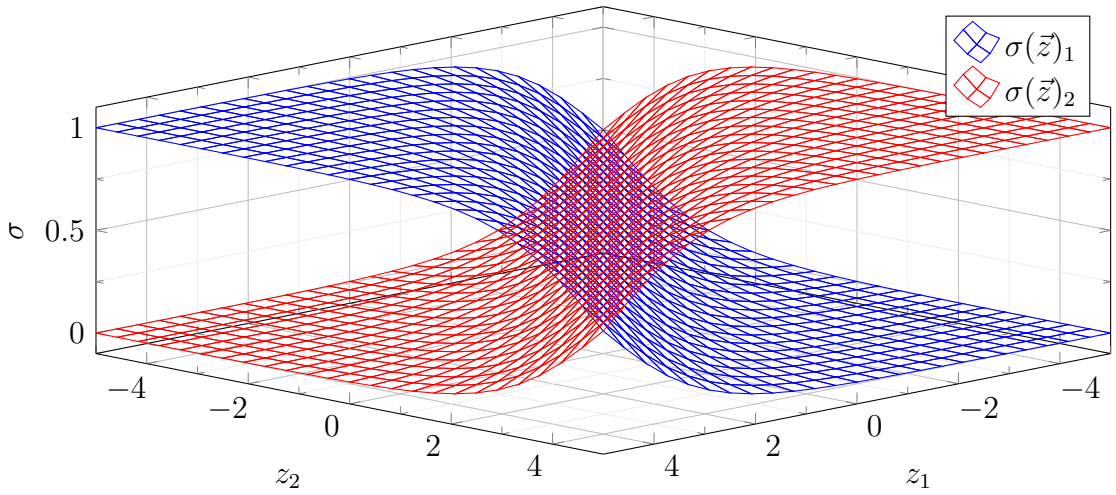

Abbildung 8: Softmax-Funktion von einem zwei-dimensionalem Vektor $\vec{z}$

3.4 Mehrschichtige Netzwerke

Um die Limitation des Perzeptrons, nur linear separierbare Probleme lösen zu können, zu überwinden, können zwischen der Eingabe- und der Ausgabeschicht weitere Schichten platziert werden, siehe Abbildung 10, dessen Neuronen mit einer Aktivierungsfunktion eine nicht-lineare Funktion g verwenden und somit die Linearität des Perzeptrons vermeiden [6, S. 18]. Diese Schichten werden als *versteckte Schichten* bezeichnet, da sie weder vom Ausgang noch vom Eingang „sichtbar“ sind.

Die versteckten Neuronen agieren, mit Ausnahme einer verschiedenen Aktivierungsfunktion, gleich wie die Ausgangs-Neuronen und brauchen so, siehe Abbildung 10, auch ihre eigenen Gewichtungen und Biase. Statt nur einem Gewichtungs-Vektor und nur einem Bias-Vektor werden also mehrere benötigt, welche jeweils, wie

die Ausgaben einer Schicht $a^{\vec{[l]}}$, mit der Nummer der Schicht $[l]$ im Exponenten angegeben werden.

Hierbei ist jedoch zu beachten, dass die Parameter jeweils nur zwischen zwei Schichten benötigt. Die Gewichtungen $W^{[l]}$ und die Biase $b^{\vec{[l]}}$ werden also zwischen den Schichten $[l]$ und $[l+1]$ benutzt und so gibt es weniger Parameter gibt als die gesamte Anzahl der Schichten L.

3.5 Die ReLU-Funktion

Es gibt für die Aktivierungsfunktion der versteckten Schichten g unendlich viele Möglichkeiten; sie soll nur nicht linear sein. Die ReLU-Funktion R hat sich jedoch als die effektivste erwiesen. Sie ist eine einfache Modifikation der Ursprungsgeraden $y = x$, um das ANN nicht mehr linear zu machen. Für alle Werte $x > 0$ wird der Funktionswert gleich null gesetzt, zu sehen in Abbildung 9 [9]. Als Formel kann dies wie folgt dargestellt werden:

$$R(z) = \begin{cases} z & \text{für } z \geq 0 \\ 0 & \text{für } z < 0 \end{cases}$$
$$= \max(0, z)$$

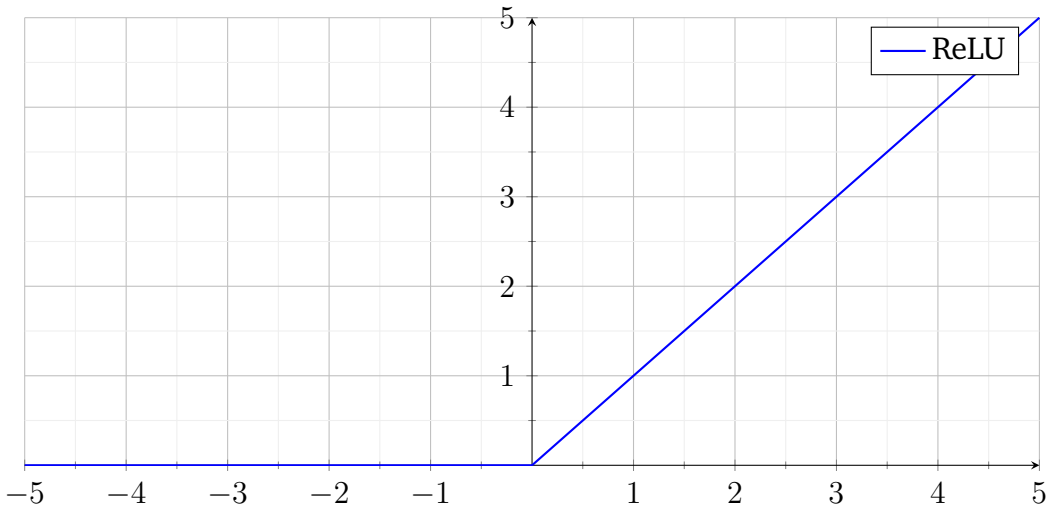

Abbildung 9: ReLU-Funktion

3.6 Die Feedforward-Architektur

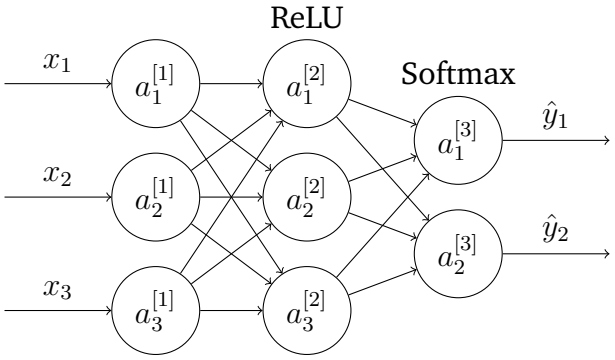

Abbildung 10: Feedforward-Netzwerk mit einer versteckten Schicht

Der Ausgang einer versteckten Schicht wird als Eingang der nächsten versteckten Schicht oder, wenn keine mehr vorhanden ist, als Eingang für die Ausgabe-Schicht verwendet. Je mehr Schichten ein ANN also hat, desto länger wird die Formel. Für die Abbildung 10 kann die Formel für die Ausgabe $\hat{\mathbf{y}}$ abhängig von der Eingabe $\vec{x}$ wie folgt erlangt werden:

$$
\begin{aligned}
\hat{\mathbf{y}} &= \vec{a}^{[3]} \\
\vec{a}^{[3]} &= \sigma(W^{[2]}\vec{a}^{[2]} + \vec{b}^{[2]}) \\
\vec{a}^{[2]} &= \sigma(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) \\
\vec{a}^{[1]} &= \vec{x}
\end{aligned}
$$

Die komplette Formel lautet also:

$$
\hat{\mathbf{y}} = \sigma(W^{[2]}(\sigma(W^{[1]}\vec{x} + \vec{b}^{[1]})) + \vec{b}^{[2]})
$$

In den folgenden Kapiteln wird von einem ANN ähnlich zur Abbildung 10 ausgegangen, welches in den versteckten Schichten die ReLU-Aktivierungsfunktion und in der Ausgangs-Schicht die Softmax-Aktivierungsfunktion verwendet.

4 Evaluieren und Trainieren eines Netzwerks

Nachdem ich die Struktur und Funktionsweise eines Feedforward-Netzwerks verstanden hatte, konnte ich mich mit meiner Leitfrage beschäftigen. Erstmal stellte sich mir die Frage, wie ein ANN überhaupt „lernt“. In Kapitel 2.2 wurde definiert, dass der Ausgang des Netzwerks von den Parametern von diesem abhängt. Damit ein ANN „funktioniert“ müssen also Parameter gewählt werden, die eine korrekte Ausgabe produzieren.

Um wie biologische Netzwerke lernen zu können, muss ein ANN jedoch erst einmal wissen, was die korrekte Ausgabe ist. Es wird ein Datensatz S mit mehreren Eingaben $\vec{x}$ und den gewünschten Ausgaben $\vec{y}$, bezeichnet als *Ziele*. Je größer dieser Datensatz ist, desto mehr „Erfahrung“ sammelt das ANN und desto besser kann es korrekte Ausgaben produzieren [7, Kapitel 1].

Da die Softmax-Funktion eine Wahrscheinlichkeitsverteilung ist, aber nur ein Wert korrekt ist, hat der Zielvektor $\vec{y}$ also nur für ein Element den Wert 1, für den Rest den Wert 0. Eine solche Kodierung eines Vektors wird als *One-Hot-Kodierung* bezeichnet [10].

Wichtig ist hier auch, dass die Eingaben eine gewisse Variation haben. Bei einem Netzwerk zum Klassifizieren des XOR-Gatters gibt es nur vier mögliche Eingaben, bei einer Klassifizierung einer Zeichnung eines Hundes mit hunderten von Pixeln ist es jedoch unmöglich jede Möglichkeit in den Datensatz aufzunehmen einen Hund zu zeichnen. Deshalb sollten also verschiedene Weisen einen solchen zu zeichnen in den Datensatz aufgenommen werden, sodass das Netz nicht nur lernt, wie ein Hund auf eine bestimmte Weise gezeichnet werden kann, sondern wie er auf verschiedene Weisen gezeichnet werden kann.

4.1 Die Verlustfunktion

Um sehen zu können, wie gut ein ANN funktioniert, benötigt man eine Verlustfunktion ℓ. Diese vergleicht die vorhergesagten Ausgaben des Netzwerks mit den erwarteten Ausgaben. Dabei repräsentiert ein Verlust von 0 eine perfekte Ausgabe; diesem Wert sollte sich also genährt werden.

Die einfachste Verlustfunktion ist der Durchschnitt der betraglichen Differenz $|\hat{\mathbf{y}} - \vec{y}|$. Dies wird als der mittlere betragliche Fehler, kurz *MAE* von dem englischen „Mean Absolute Error“, bezeichnet [11]. Da beiden Vektoren die Dimension der Ausgangs-Schicht d_o haben, sieht die Formel für MAE wie folgt aus:

$$\ell(\hat{\mathbf{y}}, \vec{y}) = \frac{1}{d_o} \sum_{i=1}^{d_o} |\hat{y}_i - y_i|$$

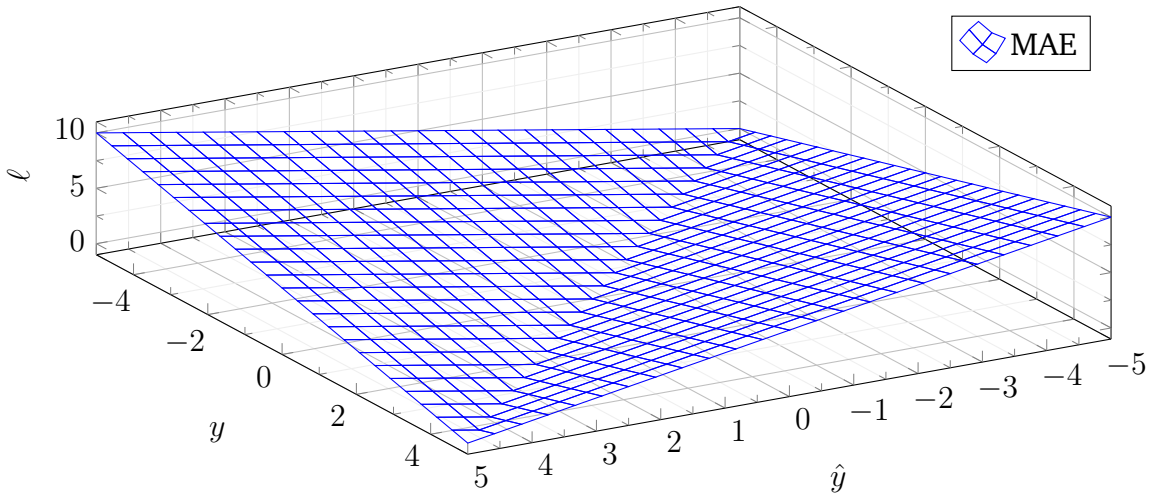

Abbildung 11: Mittlerer betraglicher Fehler von zwei Skalaren $\hat{y}$ und y

4.2 Optimierung

Wie in Kapitel 4.1 beschrieben, funktioniert ein ANN besser, je näher die Verlustfunktion an dem idealen Wert von 0 liegt. Dies ist jedoch für viele Eingaben eines Trainingsdatensatzes nicht realistisch, auch wenn die Ausgabe der Softmax-Funktion die höchste Wahrscheinlichkeit für die richtige Ausgabe hat, da die Verlustfunktion mit den eigentlichen Werten der Ausgabe, nicht mit der Interpretation der Wahrscheinlichkeitsverteilung, rechnet. Es wird also lediglich ein Minimum der Verlustfunktion gesucht.

Die Minima der Funktion sind jedoch nicht algebraisch mithilfe einer einzigen Ableitung bestimmbar, da ein ANN mit vielen Parametern arbeitet, welche jeweils ihren eigenen Einfluss auf das Endergebnis haben. Die Richtung der stärksten Steigung einer Funktion mit mehreren Argumenten kann durch den *Gradienten* ∇ angegeben werden.

Dieser ist ein Vektor und hat die einzelnen partiellen Ableitungen der Argumente als Komponenten. Um ein lokales Minimum der Verlustfunktion ℓ zu finden, wird also der negative Gradient $-\nabla\ell$ gesucht, der die Richtung angibt, in der die Funktion am stärksten sinkt. [12, Kapitel 2], [13, S. 3].

Mit diesem negativen Gradienten $-\nabla\ell$ kann der Algorithmus des *Gradientenverfahren* verwendet werden. Beim Gradientenverfahren werden die Parameter des ANN schrittweise verändert, sodass die Funktion sich in Richtung eines Minimums bewegt.

Einer Gewichtung $W^{[l]}$ würde also die partielle Ableitung $-\frac{\partial\nabla\ell}{\partial W^{[l]}}$ hinzugefügt werden. Wichtig ist hier, dass diese partielle Ableitung nicht unverändert hinzugefügt werden kann, da die Größe der partiellen Ableitung den Einfluss der Variable auf den gesamten Gradienten angibt, nicht aber die Entfernung zum Minimum. Es sollte also in kleinen Schritten verändert werden; der Faktor, mit dem die partielle Ableitung multipliziert wird, sollte unter 1 liegen. Die Größe dieses Faktors wird als die *Lernrate* η bezeichnet [12, Kapitel 2]. Eine Änderung einer Gewichtung sieht also wie folgt aus:

$$W^{[l]} = W^{[l]} - \eta\frac{\partial\nabla\ell}{\partial W^{[l]}}$$

4.3 Modifikationen am Gradientenverfahren

Eine kostante Lernrate η würde dazu führen, dass die Variablen, sobald sie nah sind an Werten, die in einem Minimum der Verlustfunktion resultieren, über diese Werte hinaus gehen könnten. Damit sich die Variablen in diese Werte einpendeln können, sollte die Lernrate nach jedem Schritt, als *Epoche* t bezeichnet, reduziert werden. Dies nennt man *Verfall der Lernrate* [14]. Die Rate dieses Verfalles ist die *Verfallsrate* λ. Es werden deswegen verschiedene Lernraten in der Notation

benötigt; die Lernrate für eine Epoche wird als η_t bezeichnet, wobei η_0 die originale Lernrate ist. Die Formel für die Lernrate einer Epoche ist wie folgt:

$$\eta_t = \eta_0 \cdot \frac{1}{1 + \lambda \cdot t}$$

Wenn wir von der originalen Lernrate $\eta_0 = 0.1$ und einer Verfallsrate von $\lambda = 1 \cdot 10^{-3}$ ausgehen, wäre die Lernrate bei der Epoche $t = 1000$ also:

$$\eta_{1000} = 0.1 \cdot \frac{1}{1 + 1 \cdot 10^{-3} \cdot 1000} = 0.05$$

4.4 Die Kreuzentropie-Verlustfunktion

Das Problem mit linearen Verlustfunktionen wie MAE ist jedoch, dass sie mit logistischen Aktivierungsfunktionen wie der Softmax-Funktion ab einem gewissen Punkt nicht mehr gute Minima finden kann. Also wird eine Verlustfunktion verwendet, die für eine Wahrscheinlichkeitsverteilung wie die Softmax-Funktion vorhergesehen ist. Eine solche ist die *Kreuzentropie* H [7, Kapitel 3], [11]:

$$H(\hat{\mathbf{y}}, \vec{y}) = -\sum_{i=1}^{d_o} y_i \ln(\hat{y}_i)$$

5 Der Gradient

Auch wenn das Prinzip des Lernprozesses klar ist, fehlt immer noch der Gradient, den dieser Prozess ja braucht. Wie in Kapitel 4.2 erwähnt, werden speziell nur die partiellen Ableitungen für die veränderbaren Parameter des Netzwerks, also den Gewichtungen und den Biasen, gebraucht.

Um den Gradienten für jeweils eine Schicht l der Parameter herauszufinden, muss die Verlustfunktion jeweils so ausgeschrieben werden, dass alle Parameter dieser Schicht darin vorhanden sind. Zur Einfachheit wird hier das ANN aus Abbildung 10 mit einer Kreuzentropie-Verlustfunktion verwendet:

$$\begin{aligned}
\ell &= H(\hat{\mathbf{y}}, \vec{y}) \\
&= H(\vec{a}^{[3]}, \vec{y}) \\
&= H(\sigma(W^{[2]}\vec{a}^{[2]} + \vec{b}^{[2]}), \vec{y}) && \leftarrow \text{Schicht 2} \\
&= H(\sigma(W^{[2]}R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]}), \vec{y}) && \leftarrow \text{Schicht 1}
\end{aligned}$$

Nun können zuerst die partiellen Ableitungen für die Schicht 1 berechnet werden. Um die Formeln kürzer zu halten, wird die neue Variable $z^{[l]} = (W^{[l]}\vec{a}^{[l]} + \vec{b}^{[l]})$ definiert:

$$\begin{aligned}
\frac{\partial \ell}{\partial \vec{a}^{[1]}} &= \frac{\partial}{\partial \vec{a}^{[1]}} H(\sigma(W^{[2]} R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]})) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial}{\partial \vec{a}^{[1]}} \sigma(W^{[2]} R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot \frac{\partial}{\partial \vec{a}^{[1]}} (W^{[2]} R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial}{\partial \vec{a}^{[1]}} R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}} \cdot \frac{\partial}{\partial \vec{a}^{[1]}} (W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}} \cdot W^{[1]}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \ell}{\partial W^{[1]}} &= \frac{\partial}{\partial W^{[1]}} H(\sigma(W^{[2]} R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]})) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial}{\partial W^{[1]}} \sigma(W^{[2]} R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot \frac{\partial}{\partial W^{[1]}} (W^{[2]} R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial}{\partial W^{[1]}} R(W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}} \cdot \frac{\partial}{\partial W^{[1]}} (W^{[1]}\vec{a}^{[1]} + \vec{b}^{[1]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}} \cdot \vec{a}^{[1]}
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial \ell}{\partial \vec{b}^{[1]}} &= \frac{\partial}{\partial \vec{b}^{[1]}} H(\sigma(W^{[2]} R(W^{[1]} \vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]})) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial}{\partial \vec{b}^{[1]}} \sigma(W^{[2]} R(W^{[1]} \vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot \frac{\partial}{\partial \vec{b}^{[1]}} (W^{[2]} R(W^{[1]} \vec{a}^{[1]} + \vec{b}^{[1]}) + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial}{\partial \vec{b}^{[1]}} R(W^{[1]} \vec{a}^{[1]} + \vec{b}^{[1]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}} \cdot \frac{\partial}{\partial \vec{b}^{[1]}} (W^{[1]} \vec{a}^{[1]} + \vec{b}^{[1]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial \vec{z}^{[2]}} \cdot W^{[2]} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}}
\end{aligned}
$$

Aufgrund der Verkettung der Funktionen wurde mehrmals die Kettenregel verwendet, was dazu führt, dass alle drei partielle Ableitungen dieser Schicht einen gleichen Faktor haben und bis auf den letzten Term gleich aussehen. Wie sich dies noch einfacher darstellen lässt, wird nach dem Ableiten der Parameter der zweiten Schicht klar:

$$
\begin{aligned}
\frac{\partial \ell}{\partial \vec{a}^{[2]}} &= \frac{\partial}{\partial \vec{a}^{[2]}} H(\sigma(W^{[2]} \vec{a}^{[2]} + \vec{b}^{[2]})) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial}{\partial \vec{a}^{[2]}} \sigma(W^{[2]} \vec{a}^{[2]} + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \partial \vec{a}^{[3]}}{\partial z^{[2]}} \cdot \frac{\partial}{\partial \vec{a}^{[2]}} (W^{[2]} \vec{a}^{[2]} + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial z^{[2]}} \cdot W^{[2]}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \ell}{\partial W^{[2]}} &= \frac{\partial}{\partial W^{[2]}} H(\sigma(W^{[2]} \vec{a}^{[2]} + \vec{b}^{[2]})) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial}{\partial W^{[2]}} \sigma(W^{[2]} \vec{a}^{[2]} + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \partial \vec{a}^{[3]}}{\partial z^{[2]}} \cdot \frac{\partial}{\partial W^{[2]}} (W^{[2]} \vec{a}^{[2]} + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial z^{[2]}} \cdot a^{[2]}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \ell}{\partial \vec{b}^{[2]}} &= \frac{\partial}{\partial \vec{b}^{[2]}} H(\sigma(W^{[2]}\vec{a}^{[2]} + \vec{b}^{[2]})) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial}{\partial \vec{b}^{[2]}} \sigma(W^{[2]}\vec{a}^{[2]} + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \partial \vec{a}^{[3]}}{\partial z^{[2]}} \cdot \frac{\partial}{\partial \vec{b}^{[2]}} (W^{[2]}\vec{a}^{[2]} + \vec{b}^{[2]}) \\
&= \frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial z^{[2]}}
\end{aligned}
$$

Es lässt sich erkennen, dass der Term $\frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}} \cdot \frac{\partial \vec{a}^{[3]}}{\partial z^{[2]}} \cdot W^{[2]}$, welcher auch in den Ableitungen der ersten Schicht vorhanden ist, die partielle Ableitung von $\vec{a}^{[2]}$ ist. Die Terme für die Ableitungen der ersten Schicht können also definiert werden als:

$$
\begin{aligned}
\frac{\partial \ell}{\partial \vec{a}^{[1]}} &= \frac{\partial \ell}{\partial \vec{a}^{[2]}} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}} \cdot W^{[1]} \\
\frac{\partial \ell}{\partial W^{[1]}} &= \frac{\partial \ell}{\partial \vec{a}^{[2]}} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}} \cdot \vec{a}^{[1]} \\
\frac{\partial \ell}{\partial \vec{b}^{[1]}} &= \frac{\partial \ell}{\partial \vec{a}^{[2]}} \cdot \frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}}
\end{aligned}
$$

Nun werden noch die Ableitung der Kreuzentropie-Funktion $\frac{\partial H(\vec{a}^{[3]})}{\partial \vec{a}^{[3]}}$, die Ableitung der Softmax-Funktion $\frac{\partial \partial \vec{a}^{[3]}}{\partial z^{[2]}}$ und die Ableitung der ReLU-Funktion $\frac{\partial \vec{a}^{[2]}}{\partial \vec{z}^{[1]}}$ benötigt.

5.1 Ableitung der Softmax- und Kreuzentropie-Funktionen

Weil die Softmax-Funktion und die Kreuzentropie-Funktion nur als $H(\sigma(z))$ vorkommen, kann die Ableitung auch zusammen berechnet werden:

$$
\frac{d}{d\vec{z}} H(\sigma(\vec{z})) = \frac{dH(\sigma(\vec{z}))}{d\sigma(\vec{z})} \cdot \frac{d\sigma(\vec{z})}{d\vec{z}}
$$

Da die Softmax-Funktion einen Vektor als Funktionsargument nimmt und auch einen Vektor als Funktionswert zurückgibt, werden hier wie bei der Verlustfunktion mehr als eine Ableitung benötigt. Es wird hier also ein Operator ähnlich zum Gradient ∇ benötigt. Da die Softmax-Funktion aber auch einen Vektor als Funktionswert zurückgibt, wird anstatt eines Vektors eine ganze Matrix an partiellen Ableitungen benötigt. Diese Matrix wird als die Jacobi-Matrix J bezeichnet [15].

Die Jakobi-Matrix für eine Funktion $\vec{f} : \mathbb{R}^n \to \mathbb{R}^m$ mit dem Vektor $\vec{z}$ als Funktionsargument ist wie folgend definiert:

$$J = \frac{\partial \vec{f}(\vec{z})}{\partial \vec{z}}$$
$$j_{ij} = \frac{\partial f(z_i)}{\partial z_j} = \frac{\partial f_i}{\partial z_j}$$

Nach der Summenregel ist die Ableitung einer Summe die Summe aller einzelnen Ableitungen. Die Jacobi-Matrix für die Kreuzentropie von der Softmax-Funktion lautet also:

$$\begin{aligned}\frac{\partial H(\sigma_i)}{\partial z_j} &= \frac{\partial}{\partial z_j}\left(-\sum_{i=1}^{d_o} y_i \cdot \ln(\sigma_i)\right) \\ &= -\sum_{i=1}^{d_o} y_i \cdot \frac{\partial}{\partial z_j} \ln(\sigma_i)\end{aligned}$$

Zuerst der Term $\frac{\partial}{\partial z_j} \ln(\sigma_i)$:

$$\begin{aligned}\frac{\partial \ln(\sigma_i)}{\partial z_j} &= \frac{\partial}{\partial z_j} \cdot \ln\left(\frac{e^{z_i}}{\sum_{l=1}^{n} e^{z_l}}\right) \\ &= \frac{\partial}{\partial z_j} \cdot \left(z_i - \ln\left(\sum_{l=1}^{n} e^{z_l}\right)\right) \\ &= \frac{\partial z_i}{\partial z_j} - \frac{\partial}{\partial z_j} \ln\left(\sum_{l=1}^{n} e^{z_l}\right) \\ &= \frac{\partial z_i}{\partial z_j} - \frac{1}{\sum_{l=1}^{n} e^{z_l}} \cdot \frac{\partial}{\partial z_j}\left(\sum_{l=1}^{n} e^{z_l}\right) \\ &= \frac{\partial z_i}{\partial z_j} - \frac{1}{\sum_{l=1}^{n} e^{z_l}} \cdot \frac{\partial}{\partial z_j}(e^{z_1} + e^{z_2} + \cdots + e^{z_j} + \cdots + e^{z_n})\end{aligned}$$

Alle Summanden außer e^{z_j} leiten zu 0 ab:

$$
\begin{aligned}
&= \frac{\partial z_i}{\partial z_j} - \frac{1}{\sum_{l=1}^{n} e^{z_l}} \cdot e^{z_j} \\
&= \frac{\partial z_i}{\partial z_j} - \frac{e^{z_j}}{\sum_{l=1}^{n} e^{z_l}} \\
&= \frac{\partial z_i}{\partial z_j} - \sigma_j
\end{aligned}
$$

$\frac{\partial}{\partial z_j} z_i$ ergibt immer 0, außer wenn z_i und z_j identisch sind, also $i = j$, dann ergibt der Term 1. Dies kann mit dem Kronecker-Delta δ vereinfacht dargestellt werden:

$$
\begin{aligned}
\frac{\partial z_i}{\partial z_j} &= \begin{cases} 1 & \text{für } i = j \\ 0 & \text{sonst} \end{cases} \\
\delta_{i,j} &= \begin{cases} 1 & \text{für } i = j \\ 0 & \text{sonst} \end{cases} \\
\frac{\partial z_i}{\partial z_j} &= \delta_{i,j}
\end{aligned}
$$

Also ist die vereinfachte Ableitung des Terms wie folgt:

$$
\frac{\partial \ln(\sigma_i)}{\partial z_j} = \delta_{i,j} - \sigma_j
$$

Dieser kann nun wieder die gesamte Formel eingesetzt werden:

$$
\begin{aligned}
\frac{\partial H(\sigma_i)}{\partial z_j} &= -\sum_{i=1}^{d_o} y_i \cdot \frac{\partial}{\partial z_j} \ln(\sigma_i) \\
&= -\sum_{i=1}^{d_o} y_i \cdot (\delta_{i,j} - \sigma_j) \\
&= \sum_{i=1}^{d_o} y_i \cdot \sigma_j - \sum_{i=1}^{d_o} y_i \cdot \delta_{i,j}
\end{aligned}
$$

Es gilt $\sum_{i=1}^{d_o} y_i \cdot \delta_{i,j} = y_j$, da der Rest die restlichen Summanden 0 ergeben:

$$\begin{aligned}\sum_{i=1}^{d_o} y_i \cdot \delta_{i,j} &= y_1 \cdot \delta_{1,j} + \cdots + y_j \cdot \delta_{j,j} + \cdots + y_c \cdot \delta_{d_o,j} \\ &= y_1 \cdot 0 + \cdots + y_j \cdot 1 + \cdots + y_{d_o} \cdot 0 \\ &= y_j\end{aligned}$$

Die Ableitung lautet nun also:

$$\begin{aligned}\frac{\partial H(\sigma_i)}{\partial z_j} &= \sum_{i=1}^{d_o} y_i \cdot \sigma_j - y_j \\ &= \sigma_j \sum_{i=1}^{d_o} y_i - y_j\end{aligned}$$

Da bei der Klassifizierung $\vec{y}$ immer nur eine Klasse den Wert 1 hat, ist $\sum_{i=1}^{d_o} y_i = 1$:

$$\begin{aligned}\frac{\partial H(\sigma_i)}{\partial z_j} &= \sigma_j \sum_{i=1}^{d_o} y_i - y_j \\ &= \sigma_j - y_j\end{aligned}$$

Die Finale Ableitung für die Kreuentropie der Softmax-Funktion in der letzten Schicht des ANNs lautet also:

$$\frac{dH(\hat{\mathbf{y}}, \vec{y})}{d\vec{z}} = \hat{\mathbf{y}} - \vec{y}$$

5.2 Ableitung der ReLU-Funktion

Die Ableitung der ReLU-Funktion $R(z)$ ist simpler als die der vorigen, da sie nur mit einer Definitions- und Zielmenge von $\mathbb{R}$ arbeitet, hat jedoch die Besonderheit, dass sie, siehe Abbildung 12, bei der Stelle $z = 0$ einen Sprung hat und somit an dieser Stelle keine Ableitung besitzt. Im Kontext des ANN hat dies jedoch keinen großen Einfluss auf das Netz, somit wird hier $R'(0) = 0$ angenommen. Die Ableitung lautet also wie folgt:

$$
\begin{aligned}
R(z) &= \begin{cases} z & \text{für } z \geq 0 \\ 0 & \text{für } z < 0 \end{cases} \\
&= \max(0, z) \\
\frac{dR(z)}{dz} &= \begin{cases} 1 & \text{für } z > 0 \\ 0 & \text{für } z < 0 \end{cases} \\
&\approx z > 0
\end{aligned}
$$

$z > 0$ stellt hier einen Wert im Sinne der booleschen Algebra dar, ist also 0 wenn nicht zutreffend, und 1 wenn zutreffend.

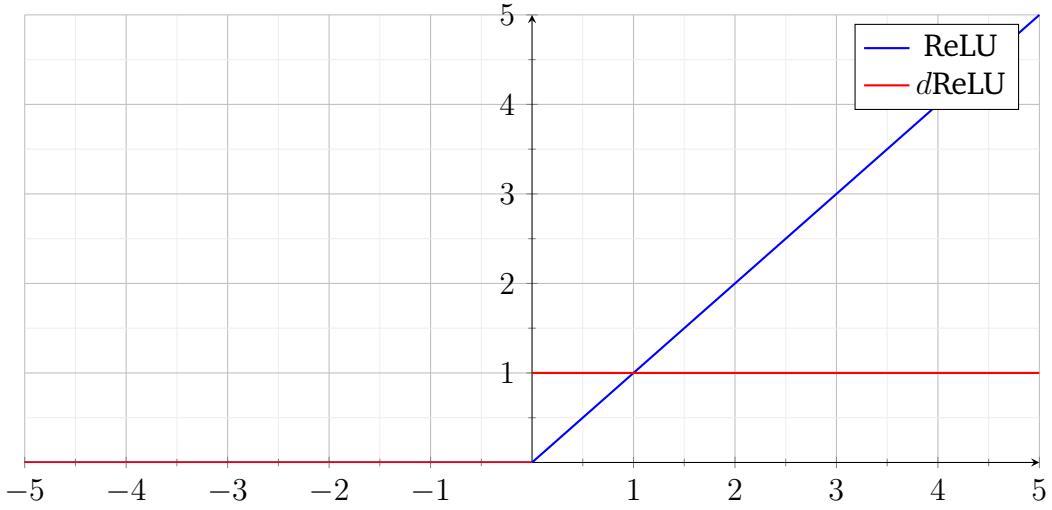

Abbildung 12: Die ReLU-Funktion und ihre Ableitung

5.3 Finaler Gradient

Mit den Ableitungen von den Aktivierungsfunktionen kann nun der Gradient vollständig mathematisch definiert werden. Hierbei wird zwischen den Parametern der letzten Schicht, der Ausgangsschicht, und den der vorigen, den versteckten Schichten, unterschieden, da die letzteren, wie in der Formel zu sehen, jeweils eine partielle Ableitung der nächsten Schicht als Faktor haben.

Die partiellen Ableitungen der Ausgangsschicht sind:

$$\begin{aligned}
\frac{\partial \ell}{\partial \vec{a}^{[L-1]}} &= \frac{\partial H(\vec{a}^{[L]})}{\partial \vec{a}^{[L]}} \cdot \frac{\partial \vec{a}^{[L]}}{\partial z^{[L-1]}} \cdot W^{[L-1]} &&= (\vec{a}^{[L]} - \vec{y}) \cdot W^{[L-1]} \\
\frac{\partial \ell}{\partial W^{[L-1]}} &= \frac{\partial H(\vec{a}^{[L]})}{\partial \vec{a}^{[L]}} \cdot \frac{\partial \vec{a}^{[L]}}{\partial z^{[L-1]}} \cdot \vec{a}^{[L-1]} &&= (\vec{a}^{[L]} - \vec{y}) \cdot \vec{a}^{[L-1]} \\
\frac{\partial \ell}{\partial \vec{b}^{[L-1]}} &= \frac{\partial H(\vec{a}^{[L]})}{\partial \vec{a}^{[L]}} \cdot \frac{\partial \vec{a}^{[L]}}{\partial z^{[L-1]}} &&= (\vec{a}^{[L]} - \vec{y})
\end{aligned}$$

Die partiellen Ableitungen der versteckten Schichten können durch folgende Terme beschrieben werden:

$$\begin{aligned}
\frac{\partial \ell}{\partial \vec{a}^{[l]}} &= \frac{\partial \ell}{\partial \vec{a}^{[l+1]}} \cdot \frac{\partial \vec{a}^{[l+1]}}{\partial \vec{z}^{[l]}} \cdot W^{[l]} &&= \frac{\partial \ell}{\partial \vec{a}^{[l+1]}} \cdot (\vec{z}^{[l]} > 0) \cdot W^{[l]} \\
\frac{\partial \ell}{\partial W^{[l]}} &= \frac{\partial \ell}{\partial \vec{a}^{[l+1]}} \cdot \frac{\partial \vec{a}^{[l+1]}}{\partial \vec{z}^{[l]}} \cdot \vec{a}^{[l]} &&= \frac{\partial \ell}{\partial \vec{a}^{[l+1]}} \cdot (\vec{z}^{[l]} > 0) \cdot \vec{a}^{[l]} \\
\frac{\partial \ell}{\partial \vec{b}^{[l]}} &= \frac{\partial \ell}{\partial \vec{a}^{[l+1]}} \cdot \frac{\partial \vec{a}^{[l+1]}}{\partial \vec{z}^{[l]}} &&= \frac{\partial \ell}{\partial \vec{a}^{[l+1]}} \cdot (\vec{z}^{[l]} > 0)
\end{aligned}$$

Letzendlich können die partiellen Ableitungen also als ein einfacher Algorithmus dargestellt werden, welcher jeweils auf den Gradienten der nächsteren Schicht basiert. Es ist hier also sinnvoll, bei einer Implementation dieses Algorithmuses bei der letzten Schicht anzufangen und sich „zurückzuarbeiten“. Deswegen wird diese Strategie, zusammen mit der in Kapitel 4.2 beschriebenen Optimierung, *Rückpropagierung* genannt und bildet den letzten wichtigen Schritt im Verständnis des Lernprozesses eines ANNs [7, Kapitel 2].

6 Fazit und Selbstreflexion

Nachdem ich meine Leitfrage also in Form der obigen sechs Formeln beantworten konnte und über die letzten Monate die Funktionsweise des Feedforward-Netzwerks verstanden hatte, konnte ich mich also an die Implementation eines solchen machen. Die Analyse des Quelltextes überschreitet den Rahmen dieser Ausarbeitung, dieser ist jedoch auf `https://github.com/RisGar/bll` einzusehen.

Das Ergebnis meiner Implementation ist ein Netzwerk, welches aus einem Satz von 10.000 28×28 Pixel großen schwarz-weiß Bildern von Kleidungsartikeln aus dem „Fashion MNIST“-Datensatz von Zalando Research, verfügbar unter `https://github.com/zalandoresearch/fashion-mnist`, nach etwa 1000 Epochen eine Genauigkeit von 86% richtigen Klassifikationen erzielt. Nach mehr Epochen schien sich diese Genauigkeit jedoch nicht groß zu verändern, was darauf schließen lässt, dass für ein solches Problem ein komplexeres ANN benötigt wird.

Meine Leitfrage und Ausarbeitung explizit auf die theoretische und mathematische Seite von ANNs zu beziehen, hat mir bei der technischen Implementation eines ANNs dabei geholfen, den Code den ich schrieb, auch wirklich zu verstehen und nicht, wozu man oft in Versuchung kommt, Quelltext aus diversen Online-Quellen zu beziehen, auch wenn man diesen nicht vollständig versteht. Deswegen werde ich mich auf jeden Fall weiter mit dem Thema beschäftigen und meine programmatische Implementation weiterführen.

Neuronale Netzwerke werden in unserer heutigen Zeit immer wichtiger und sind nicht nur interessant für große Cloud-Anbieter von AI-Diensten wie OpenAI, aber auch für lokale Anwendungszwecke. Zwar sind moderne neuronale Netzwerke wie GPT-4, mit dem ChatGPT läuft, unglaublich komplex, aber wie mit dieser Ausarbeitung gezeigt wurde, ist die Essenz von neuronalen Netzwerken sehr simpel und das reine Verarbeiten von Eingaben zu Ausgaben in einem ANN kann so selbst direkt auf sogar den kleinsten Geräten durchgeführt werden, die Möglichkeiten sind also unendlich.

Literaturverweise

[1] „On Mathematical Maturity," präsentiert bei 27th Annual PCMI Summer Session (Park City, Utah), 17. Juli 2017. Adresse: https://www.youtube.com/watch?v=zHU1xH6Ogs4 (besucht am 13. 01. 2024).

[2] K. Suzuki, *Artificial Neural Networks - Methodological Advances and Biomedical Applications*. 11. Apr. 2011, ISBN: 978-953-307-243-2.

[3] W. S. McCulloch und W. Pitts, „A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, Jg. 5, Nr. 4, S. 115–133, Dez. 1943, ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02478259. Adresse: http://link.springer.com/10.1007/BF02478259 (besucht am 23. 02. 2024).

[4] A. L. Chandra. „McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron," Towards Data Science. (27. Sep. 2022), Adresse: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1 (besucht am 30. 03. 2024).

[5] F. Rosenblatt, „The Perceptron, a Perceiving and Recognizing Automaton (Project PARA)," Cornell Aeronautical Laboratory, Cornell Aeronautical Laboratory, 1957. Adresse: https://books.google.de/books?id=P_XGPgAACAAJ.

[6] H. Abdi, „A Neural Network Primer," *Journal of Biological Systems*, Jg. 02, Nr. 03, S. 247–281, Sep. 1994, ISSN: 0218-3390, 1793-6470. DOI: 10.1142/S0218339094000179. Adresse: https://www.worldscientific.com/doi/abs/10.1142/S0218339094000179 (besucht am 31. 03. 2024).

[7] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. Adresse: http://neuralnetworksanddeeplearning.com (besucht am 29. 03. 2024).

[8] S. Sharma. „What the Hell is Perceptron?" Towards Data Science. (11. Okt. 2019), Adresse: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53 (besucht am 31. 03. 2024).

[9] S. Sharma. „Activation Functions in Neural Networks,“ Towards Data Science. (20. Nov. 2022), Adresse: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (besucht am 01. 04. 2024).

[10] J. Brownlee. „Ordinal and One-Hot Encodings for Categorical Data,“ Machine Learning Mastery. (11. Juni 2020), Adresse: https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/ (besucht am 02. 04. 2024).

[11] B. Fortuner, *Machine Learning Glossary*. Adresse: https://ml-cheatsheet.readthedocs.io/en/latest/index.html (besucht am 01. 04. 2024).

[12] G. Sanderson. „3Blue1Brown.“ (), Adresse: https://www.3blue1brown.com/topics/3blue1brown.com (besucht am 30. 03. 2024).

[13] Y. Lecun, „A Theoretical Framework for Back-Propagation,“ *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, D. Touretzky, G. Hinton und T. Sejnowski, Hrsg., S. 21–28, 1988.

[14] V. Haswani. „Learning Rate Decay and methods in Deep Learning,“ Analytics Vidhya. (30. Mai 2021), Adresse: https://medium.com/analytics-vidhya/learning-rate-decay-and-methods-in-deep-learning-2cee564f910b (besucht am 03. 04. 2024).

[15] E. Bendersky. „The Softmax Function and Its Derivative - Eli Bendersky's Website,“ Eli Bendersky's website. (), Adresse: https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/ (besucht am 02. 04. 2024).