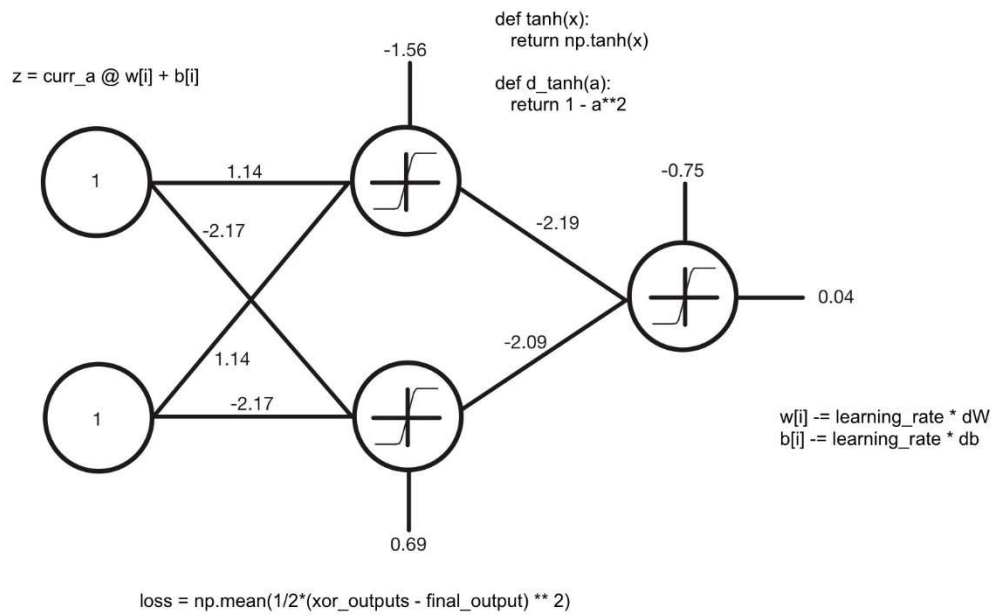


Neuronale Netze und nichtlineare Klassifikation

-Facharbeit im Fach Mathematik-



Verfasser: Marvin Jäger

Fach: Mathematik

Ort: Gladbeck

Jahr: 2026

Inhaltsverzeichnis	
1. Einleitung	3
2. Mathematische Modellierung eines künstlichen Neurons	4
2.1 Definition eines künstlichen Neurons	4
2.2 Vektorielle Darstellung	4
2.3 Der Bias-Term als Parameter einer affinen Transformation	5
2.4 Funktionskomposition	6
2.5 Mathematischer Nachweis: Das Problem der Linearität	6
3. Lineare Trennbarkeit und das XOR-Problem	7
3.1 Formale Definition der linearen Separierbarkeit	7
3.2 Geometrische Repräsentation der XOR-Logik	7
3.3 Algebraischer Beweis: Ein Neuron kann XOR nicht lösen	7
3.4 Die Notwendigkeit mehrschichtiger Architekturen	8
4. Aktivierungsfunktionen in einem neuronalen Netzwerk	8
4.1 Mathematisches Anforderungsprofil	8
4.2 Aktivierungsfunktionen	9
○ 4.2.1 Sigmoid	9
○ 4.2.2 Tanh	9
○ 4.2.3 ReLU	9
4.3 Bedeutung der Steigung für den Gradientenabstieg	10
5. Der Backpropagation-Algorithmus	10
5.1 Die Fehlerfunktion	10
5.2 Partielle Ableitungen und die Bildung des Gradientenvektors	11
5.3 Mehrdimensionale Kettenregel zur Fehlerfortpflanzung	11
5.4 Das Gradientenabstiegsverfahren	13
6. Versuch	13
6.1 Backpropagation implementiert am XOR-Problem	13
6.2 Erweiterte Architektur für komplexere Funktionen	14
7. Fazit	14
8. Literatur- und Quellenverzeichnis	16
9. Anhang	18

1. Einleitung

Das Thema dieser Facharbeit hatte seinen Ursprung schon vor ungefähr einem Jahr, als ich ein selbstfahrendes Auto entwickelt habe, mit einem eigenen neuronalen Netzwerk, jedoch ohne wirklich zu wissen, warum genau das Netz lernen kann, wie es zu lenken hat oder wann es beschleunigen und bremsen soll, auf Basis konkreter Datensätze und vergleichsweise simpler Mathematik.

Ziel dieser Arbeit ist es, die Funktionsweise neuronaler Netzwerke aus der mathematischen Perspektive zu untersuchen und später mit einem Versuch in die Praxis umzusetzen. Der Schwerpunkt liegt auf der Modellierung eines künstlichen Neurons und darauf, wo die Grenzen dieser Modelle liegen und wie diese mit Aktivierungsfunktionen überwunden werden können. Zudem werden ein Lernalgorithmus sowie die Auswirkungen verschiedener Aktivierungsfunktionen auf die Konvergenz und den Ausgabewert untersucht.

Untersucht werden ausschließlich der mathematische Aufbau und die Funktionsweise neuronaler Netze sowie deren Trainingsverfahren. Aspekte wie biologischer Hintergrund, praktische Anwendungsfelder oder gesellschaftliche Auswirkungen werden bewusst nicht behandelt, da der Fokus auf der mathematischen Struktur liegt.

Die Arbeit basiert auf der Herleitung zentraler mathematischer Zusammenhänge und Modellierungen durch Konzepte der linearen Algebra und Analysis, Vektorräume, affine Abbildungen, Funktionskompositionen und partielle Ableitungen. Abschließend wird ein einfaches neuronales Netz implementiert, um das XOR-Problem zu lösen und die theoretischen Ergebnisse und Herleitungen praktisch zu veranschaulichen.

Im Hauptteil wird zuerst die mathematische Modellierung eines einzelnen Neurons vorgenommen und dessen lineare Eigenschaften werden untersucht. Anschließend wird das XOR-Problem, welches das am Ende zu lösende Problem sein wird, analysiert und die Notwendigkeit mehrschichtiger Architekturen begründet. Darauf folgend kommt eine Analyse der verschiedenen Aktivierungsfunktionen und das Problem des Vanishing-Gradient, welches damit einhergeht, wird erläutert, sowie die Herleitung des Backpropagation-Algorithmus. Zum Abschluss kommen zwei Versuche, implementiert in Python mit NumPy, in denen das XOR-Problem untersucht wird, mit verschiedenen Aktivierungsfunktionen und Parametern, um deren Unterschiede praktisch zu veranschaulichen.

2. Mathematische Modellierung eines künstlichen Neurons

2.1 Definition eines künstlichen Neurons

Ein künstliches Neuron j lässt sich als mathematische Funktion seiner n Eingänge i durch drei grundlegende Komponenten definieren:

Gewichtung (w_{ij}): Jedem Eingang i des Neurons j ist ein Gewicht zugeordnet, welches den Einfluss des jeweiligen Eingangssignals auf den späteren Ausgabewert skaliert. Dabei gilt: Je kleiner der Betrag des Gewichts ist, desto geringer ist die Auswirkung auf den Output. Das Vorzeichen eines Gewichts wirkt auf eine Eingabe entweder erregend, sofern dieses positiv ist (*exzitatorisch*), oder hemmend bei einem negativen Wert (*inhibitorisch*). Bei einem Gewicht von Null wirkt sich die Eingabe nicht auf die Ausgabe des Neurons aus, weil es das Fehlen einer Verbindung signalisiert.¹

Übertragungsfunktion (Σ): Die Übertragungsfunktion bereitet die Eingaben des Neurons für die Aktivierungsfunktion vor. Hierbei wird jeder Eingangswert mit seinem entsprechenden Gewicht multipliziert und die Produkte werden mit dem Bias addiert. Die Summe wird dann anschließend der Aktivierungsfunktion übergeben.¹

Aktivierungsfunktion (φ): Die Summe der Übertragungsfunktion wird schließlich in eine Aktivierungsfunktion eingesetzt, was die endgültige nichtlinear gefilterte Ausgabe des Neurons ist. Es gibt verschiedenste Aktivierungsfunktionen, deren wesentliche Aufgabe darin besteht, Nichtlinearität in den Ausgabewert zu bringen, wodurch sich nichtlineare Probleme lösen lassen.¹

2.2 Vektorielle Darstellung

Um die Arbeitsweise eines Neurons mit mehreren Eingängen mathematisch zu präzisieren, werden die Eingabewerte als Vektor dargestellt. Dies dient einer kompakteren Notation, aber auch, da Grafikkarten auf parallele Matrix-Vektor-Operationen optimiert sind. Jeder Eingang des Neurons wird als Komponente eines Eingangsvektors x dargestellt, welcher üblicherweise ein Spaltenvektor ($1 \times n$) ist.²

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_R \end{bmatrix}$$

¹ Vgl.: Unbekannter Autor unbekanntes Jahr: Künstliches Neuron, online

² Vgl.: Hagan Martin T.; Demuth Howard B.; Beale Mark H.; De Jesús Orlando unbekanntes Jahr, S. 42-43

Ebenso lassen sich auch die Gewichte als Vektoren darstellen, wobei die erste tiefgestellte Zahl das Neuron beschreibt, zu welchem das Gewicht gehört und die zweite tiefgestellte Zahl den dazugehörigen Eingang. Wegen der Funktionsweise der Matrix-Vektor-Multiplikation, ist es notwendig, dass der Gewichtsvektor ein Zeilenvektor ($1 \times n$) ist, wenn der Gewichtsvektor: $w_1 = [w_{1,1}, w_{1,2}, w_{1,3}, \dots, w_{1,R}]$ mit dem Eingangsvektor multipliziert wird.¹

Dieser Wechsel der Orientierung heißt Transponierung, was einen Spaltenvektor in einen Zeilenvektor umwandelt.²

Wie in Kapitel 2.1 erwähnt, ist die Kernaufgabe eines Neurons die Berechnung der gewichteten Summe anstatt jede Multiplikation einzeln zu berechnen, kann auch das Skalarprodukt beider Vektoren berechnet werden: $w_1^T * x = \sum_{i=1}^n w_{1,i} * x_i$.³

Damit diese Gleichung vollständig ist, wird noch ein Bias b addiert, wodurch die endgültige Gleichung im Neuron wie folgt lautet: $n = w_1^T * x + b_1$.¹

2.3 Der Bias-Term als Parameter einer affine Transformation

Das Bias-Neuron ist besonders innerhalb eines Netzes, es hat keine Propagierungs- oder Aktivierungsfunktion. Stattdessen wird sein Ausgangswert immer mit eins multipliziert. Es dient dazu, den Schwellenwert einer Aktivierungsfunktion zu verschieben ohne dabei die Funktion anzupassen.^{4,5}

Ohne Bias-Term ist die Entscheidungsgrenze des Neurons linear und geht immer durch den Ursprung. Der Bias-Term ermöglicht eine Verschiebung dieser Grenze im Raum. Es wirkt sich auch auf die Aktivierungsfunktion aus, wodurch auch im Aktivierungsfunktionsraum die Entscheidungsgrenze verschoben werden kann.^{5,6}

Das ermöglicht dem Neuron, mathematische Zusammenhänge abzubilden, die eine Translation der Kurve erfordern. Der Bias wird zur gewichteten Summe addiert, bevor die Aktivierungsfunktion angewendet wird, er verschiebt somit den Schwellenwert. Das entspricht einer affinen Transformation, zuerst werden die Eingaben mit den Gewichten multipliziert und anschließend wird der Bias addiert: $f(x) = w^T * x + b$.^{7,8}

¹ Vgl.: Hagan, Martin T.; Demuth, Howard B.; Beale, Mark H.; De Jesús, Orlando 2014, S. 42

² Vgl.: Kneusel, Ronald T. 2022, S. 113

³ Vgl.: Goodfellow, Ian; Benigo, Yoshua; Courville, Aaron 2016, online, S. 32-35

⁴ Vgl.: Kriesel, David 2005, S. 46-47

⁵ Vgl.: Unbekannter Autor unbekanntes Jahr: Künstliches Neuron, online

⁶ Vgl.: Wang, Shengjie; Zhou, Tianyi; Bilmes, Jeffery A. 2019, online, S. 5

⁷ Vgl.: Montesinos-Lopez, Osval A.; Montesinos, Abelardo; Crossa, Jose 2022, online

⁸ Vgl.: Cabello, Julia Garcia 2022, online

2.4 Funktionskomposition

Die affine Transformation ist in der Regel nicht die finale Ausgabe eines Neurons. Stattdessen ist die Gesamtfunktion eines Neurons eine Komposition aus zwei Funktionen: einmal die Berechnung der Netzeingabe und die anschließende Transformation durch eine Aktivierungsfunktion, der finale Ausgabewert.¹

Innere Funktion (n): Die Berechnung der Netzeingabe: $n = w^T * x + b$.

Äußere Funktion (φ): Die Anwendung der Aktivierungsfunktion auf die Netzeingabe: $a = \varphi(n) = \varphi(w^T * x + b)$.

Daraus ergibt sich dann folgende Funktion: $a(x) = \varphi(w^T * x + b)$.

Die wiederholte Funktionskomposition über mehrere Schichten hinweg ermöglicht die Approximation komplexer Funktionen mit beliebiger Genauigkeit.²

2.5 Mathematischer Nachweis: Das Problem der Linearität

Die Notwendigkeit nichtlinearer Aktivierungsfunktionen lässt sich an einem Beispiel verdeutlichen. Ein neuronales Netz mit zwei aufeinanderfolgenden Schichten, wobei zur Vereinfachung auf den Bias-Term verzichtet wird.

Die erste Schicht, berechnet aus dem Inputvektor x und der Gewichtsmatrix W_1 , ergibt den Vektor y : $y = W_1 * x$.

Schicht 2 nimmt y und berechnet mit der Gewichtsmatrix W_2 den Output n :

$$n = W_2 * y.$$

Wenn die Gleichung für y in die Gleichung für n eingesetzt wird, ergibt sich folgende Funktionskomposition: $n = W_2 * (W_1 * x)$.

Unter Anwendung des Assoziativgesetzes für die Matrixmultiplikation lassen sich die Klammern umstellen: $n = (W_2 * W_1) * x$.

Da das Produkt zweier Matrizen W_2 und W_1 wiederum eine Matrix W_{neu} ist, lässt sich die gesamte Operation als einzige lineare Transformation darstellen:

$$n = W_{neu} * x.^3$$

Dieser Nachweis zeigt, dass bei n -vielen Schichten, solange nur lineare Operationen durchgeführt werden, der Ausgabewert mathematisch dem Ergebnis einer einzelnen Schicht entspricht. Ohne eine nichtlineare Aktivierungsfunktion,

¹ Vgl.: Unbekannter Autor unbekanntes Jahr: Künstliches Neuron, online

² Vgl.: Unbekannter Autor unbekanntes Jahr: Universal approximation theorem, online

³ Vgl.: Kneusel, Ronald T. 2022, S. 121

sind mehrschichtige Netze unnötig und können keine komplexen Zusammenhänge klassifizieren, wie es auch eine lineare Regression könnte.¹

3. Lineare Trennbarkeit und das XOR-Problem

3.1 Formale Definition der linearen Separierbarkeit

Zwei Mengen von Punkten werden als linear separierbar bezeichnet, wenn diese durch eine Hyperebene im n -dimensionalen Vektorraum vollständig voneinander getrennt werden können. Da das XOR-Problem eine Abbildung im zweidimensionalen Raum darstellt, müsste eine Gerade existieren zur korrekten Klassifizierung der beiden Mengen.²

3.2 Geometrische Repräsentation der XOR-Logik

Beim OR-Fall gibt es drei positive Fälle, welche mit einer Geraden vom negativen Fall getrennt werden können. Der Bias bestimmt dabei den Achsenabschnitt dieser Hyperebene.³

Die XOR-Logik umfasst zwei positive Fälle und zwei negative Fälle, welche sich nicht mit einer Trenngeraden trennen lassen, da beide Fälle geometrisch gesehen über Kreuz liegen, womit sich das XOR-Problem nicht mit einem einfachen Neuron trennen lässt. Stattdessen benötigt das Neuron eine nichtlineare Aktivierungsfunktion, damit die Ausgabewerte nicht linear sind, was auch in der Abbildung A1 im Anhang sichtbar wird.³

3.3 Algebraischer Beweis: Ein Neuron kann XOR nicht lösen

Bei einem Neuron mit einer Schwellenwertfunktion, bei dem das Neuron bei $x \geq 0$ feuert und bei $x < 0$ nicht feuert, ergeben sich folgende Bedingungen für die XOR-Logik:

- Punkt A(0,0) mit der Ausgabe 0: $w_1(0) + w_2(0) + b \rightarrow b < 0$
- Punkt B(0,1) mit der Ausgabe 1: $w_1(0) + w_2(1) + b \rightarrow w_2 + b \geq 0$
- Punkt C(1,0) mit der Ausgabe 1: $w_1(1) + w_2(0) + b \rightarrow w_1 + b \geq 0$
- Punkt D(1,1) mit der Ausgabe 0: $w_1(1) + w_2(1) + b \rightarrow w_1 + w_2 + b < 0$

Um den Widerspruch aufzuzeigen, werden die Erkenntnisse aus den Bedingungen (2) und (3) kombiniert:

¹ Vgl.: Unbekannter Autor unbekanntes Jahr: Künstliches Neuron, online

² Vgl.: Unbekannter Autor unbekanntes Jahr: Lineare Separierbarkeit, online

³ Abbildung A1: Darstellung des XOR-Problems

Aus (2) folgt: $w_2 \geq -b$, aus (3) folgt: $w_1 \geq -b$, und addiert ergeben sie: $w_1 + w_2 \geq -2b$

Um auf die Übertragungsfunktion aus der Bedingung (4) zu kommen, wird der Bias b auf beiden Seiten der Gleichung addiert: $w_1 + w_2 + b \geq -b$

Die erste Bedingung besagt, dass b negativ ist, also muss $-b$ positiv sein, woraus folgt: $w_1 + w_2 + b > 0$

Diese mathematische Schlussfolgerung steht im direkten Widerspruch zur vierten Bedingung, da $w_1 + w_2 + b$ nicht gleichzeitig positiv wie auch negativ sein kann. Das zeigt, dass es für ein einzelnes Neuron, selbst mit Aktivierungsfunktion, unmöglich ist, die zwei Mengen zu trennen, was die mehrschichtige Architektur von Netzen unumgänglich macht.

3.4 Die Notwendigkeit mehrschichtiger Architekturen

Da ein einfaches Netzwerk nicht in der Lage ist, das XOR-Problem zu lösen, ist eine Erweiterung der Architektur des Netzwerkes erforderlich. Die Lösung ist ein mehrschichtiges Netzwerk mit mindestens einer zusätzlichen Schicht, dem sogenannten Hidden-Layer. Diese transformiert den Eingaberaum des Netzwerkes durch eine Folge nichtlinearer Abbildungen, wodurch zuvor nicht linear trennbare Mengen separierbar werden. Der Hidden-Layer besteht dabei üblicherweise nicht nur aus einer Schicht an Neuronen. Für das XOR-Problem reicht eine Konfiguration aus insgesamt drei Schichten aus: Der Input-Layer, welcher die Eingaben entgegennimmt, dann der Hidden-Layer mit einer Schicht zur nichtlinearen Transformation sowie ein Output-Layer, welcher die Ergebnisse aus dem Hidden-Layer zur Klassifikation zusammenführt.¹

4. Aktivierungsfunktionen in einem Neuronalen Netzwerk

4.1 Mathematisches Anforderungsprofil

Die wichtigste Anforderung ist die Nichtlinearität, weil, wie in Kapitel 2.5 gezeigt, sich sonst mehrere Schichten in einer zusammenfassen lassen. Funktionen, die stückweise linear sind, erfüllen auch die Anforderung, weil sich diese Operation nicht durch eine einfache Matrixmultiplikation beschreiben lässt.²

Da das Training auf dem Gradientenabstiegsverfahren basiert, muss die Aktivierungsfunktion nicht zwingend im gesamten Definitionsbereich differenzierbar sein. Die erste Ableitung hat einen direkten Einfluss auf die

¹ Vgl.: Lee, Minhyeok 2023, online, S. 3-4

² Vgl.: Cabello, Julia Garcia 2022, online

Gewichtsaktualisierung, da diese bestimmt, in welche Richtung die Fehlerfunktion minimiert werden muss.¹

Idealerweise ist die Funktion monoton steigend. Das sorgt dafür, dass die Richtung des Gradienten stabil bleibt und nicht unvorhersehbar springt, da bei einer monotonen Funktion die erste Ableitung entweder positiv oder negativ ist, was ein stabiles Lernverhalten begünstigt.¹

4.2 Aktivierungsfunktionen

4.2.1 Sigmoid

Die Sigmoid-Funktion $f(x) = \frac{1}{1+e^{-x}}$ hat einen Wertebereich von $[0,1]$ und die erste Ableitung lautet: $f'(x) = f(x)(1 - f(x))$. Der Ausgabewert ist für höhere und niedrigere Eingaben gesättigt, was beim Lernen zum Vanishing-Gradient-Problem führen kann. Dabei nähert sich der Gradient der Zielfunktion Null an, wodurch die Parameter-Updates im stochastischen Gradientenabstiegsverfahren praktisch verschwinden, wodurch das Training stagniert. Zudem ist sie nicht nullzentriert, da ihre Ausgaben immer positiv sind. Dies kann zu ineffizienteren Gewichts Anpassungen und langsameren Konvergenzen führen. Die Sigmoid-Funktion war eine der ersten und wird oft noch in der Ausgabeschicht genutzt, aber in dem Hidden-Layer wird sie meist schon durch modernere und bessere ersetzt.²

4.2.2 Tanh

Die Tanh-Funktion $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ hat einen Wertebereich von $[-1,1]$ und die erste Ableitung lautet: $f'(x) = 1 - f(x)^2$. Die Nachteile wie das Vanishing-Gradient-Problem oder hohe Rechenkomplexität treten, wie auch bei der Sigmoid-Funktion, auch bei der Tanh-Funktion auf. Dafür ist die Funktion nullzentriert, was das Lernen oft beschleunigen kann. Wie die Sigmoid-Funktion wird die Tanh-Funktion sehr häufig noch genutzt und war eine der ersten.²

4.2.3 ReLU

Die ReLU-Funktion $f(x) = \max(0, x)$ hat einen Wertebereich von $[0, \infty[$ und die erste Ableitung lautet: $f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$. Aufgrund ihrer einfachen Berechnung, schnelleren Konvergenz und teilweisen Lösung des Vanishing-Gradient-Problems ist sie die heute meistgenutzte Aktivierungsfunktion. Für positive Inputs hat die ReLU-Funktion einen konstanten Gradienten von eins, wodurch Gradienten durch

¹ Vgl.: Cabello, Julia Garcia 2022, online

² Vgl.: Dubey, Shiv Ram; Singh, Satish Kumar; Chaudhuri, Bidyut Baran 2022, online

viele Schichten effektiv zurückpropagiert werden können. Dadurch ist das Training oft stabiler und schneller als bei der Sigmoid- oder Tanh-Funktion. Ein Nachteil ist die Nichtverwendung negativer Werte, was zu dem Dying-ReLU-Problem führen kann, bei dem die Neuronen dauerhaft den Wert Null ausgeben und keinen Beitrag mehr zum Lernprozess leisten. Diese Problem wird durch angepasste Varianten wie Leaky ReLU, Parametric ReLU oder ELU abgeschwächt.¹

4.3 Bedeutung der Steigung für den Gradientenabstieg

Die Sigmoid- und Tanh-Funktion werden im Hidden-Layer immer seltener benutzt, da ihre weitreichenden Sättigungsbereiche das gradientenbasierte Lernen sehr erschweren können. Bei der Sigmoid-Funktion lernt ein Neuron nur um den Schwellenwert $x \approx 0$ wo die Steigung ihr Maximum erreicht. Sobald das Neuron hohe oder niedrige Werte erreicht und somit in die Sättigung geht, reagiert es kaum noch auf Fehlerkorrekturen, weil sich die Steigung dem Wert Null nähert. Dies führt zum Vanishing-Gradient-Problem. Es entsteht, wenn bei dem Backpropagation-Verfahren durch die Kettenregel wiederholt Werte kleiner als eins multipliziert werden, wodurch der Gradient über viele Schichten exponentiell abnimmt. Da die erste Ableitung der Sigmoid-Funktion maximal 0.25 beträgt, verschwindet das Korrektursignal für die vorderen Schichten fast. Deswegen wird die ReLU-Funktion bevorzugt, weil wie in 4.2.3 beschrieben, die Steigung für alle positiven Werte konstant eins ist, weshalb das Netzwerk stabiler lernt und nicht durch eine Sättigung ausgebremst wird.²

5. Der Backpropagation-Algorithmus

5.1 Die Fehlerfunktion

Damit ein neuronales Netz lernen kann, braucht es eine mathematische Größe, die zeigt, wie stark die Ausgabe von dem gewünschten Zielwert abweicht. Diese Größe ist die Fehlerfunktion. In dieser Arbeit wird der quadratische Fehler (*Quadratic-Loss*) genutzt, da diese stets positive Werte liefert und größere Fehler überproportional stark bestraft. Bei der Funktion ist y der Zielwert und a die Ausgabe des Netzes: $L = \frac{1}{2}(y - a)^2$. Angenommen, der Zielwert ist $y = 1$ und die Ausgabe des Netzes ist $a = 0.7$, dann ergibt sich: $L = \frac{1}{2}(1 - 0.7)^2 = 0.045$. Wenn die Ausgabe mit dem Zielwert übereinstimmt, ergibt sich Null, während der Fehler bei größeren Abweichungen quadratisch ansteigt. Der Faktor $\frac{1}{2}$ dient nur zur mathematischen Vereinfachung, sodass bei der ersten Ableitung der Faktor

¹ Vgl.: Dubey, Shiv Ram; Singh, Satish Kumar; Chaudhuri, Bidyut Baran 2022, online

² Vgl.: Goodfellow, Ian; Benigo, Yoshua; Courville, Aaron 2016, online, S. 191-192, 286-287

wegfällt. Die erste Ableitung lautet: $\frac{dL}{da} = a - y$, welche für den Backpropagation-Algorithmus gebraucht wird, um festzustellen, in welche Richtung die Gewichte angepasst werden müssen.¹

5.2 Partielle Ableitung und Bildung des Gradientenvektors

Die Fehlerfunktion hängt von vielen Parametern gleichzeitig ab, wie den Gewichten und Biases, weshalb partielle Ableitungen verwendet werden, um den Einfluss jedes einzelnen Parameters auf den Gesamtfehler zu bestimmen. Jede partielle Ableitung beschreibt, wenn ein bestimmtes Gewicht oder ein Bias angepasst wird, während alle anderen Parameter konstant bleiben, wie sich der Fehler verändert. Mit der mehrdimensionalen Kettenregel wird der Fehler schrittweise vom Output-Layer zurück durch den Hidden-Layer propagiert. Die erste Ableitung der Aktivierungsfunktion bestimmt dabei, wie stark Fehleränderungen an vorherige Neuronen weitergegeben werden. Alle partiellen Ableitungen bilden den Gradientenvektor. Dieser zeigt in die Richtung des stärksten Anstiegs der Fehlerfunktion und bildet die Grundlage für das anschließende Gradientenabstiegsverfahren.^{2,3}

5.3 Mehrdimensionale Kettenregel zur Fehlerfortpflanzung

Zur Veranschaulichung wird ein Netzwerk mit der Architektur 2-2-1 und der Sigmoid-Funktion betrachtet, wie auch in Abbildung A2 im Anhang dargestellt. An diesem Beispiel werden alle relevanten Ableitungen hergeleitet und erläutert.⁴

Der Fehler L hängt nicht direkt vom Bias b_2 ab, sondern über mehrere Zwischenschritte. Die Abhängigkeit verläuft entlang des Rechenweges im Netz: $b_2 \rightarrow z_2 \rightarrow a_2 \rightarrow L$. Der Bias b_2 beeinflusst zunächst die gewichtete Summe z_2 . Diese bestimmt über die Aktivierungsfunktion die Ausgabe a_2 , welche schließlich in die Fehlerfunktion L eingeht. Da alle Zwischenschritte berücksichtigt werden müssen, muss die mehrdimensionale Kettenregel angewendet werden. Gesucht ist die partielle Ableitung des Fehlers nach dem Bias b_2 und mit der Kettenregel ergibt sich:

$$\frac{\partial L}{\partial b_2} = \left(\frac{\partial L}{\partial a_2} \right) \left(\frac{\partial a_2}{\partial z_2} \right) \left(\frac{\partial z_2}{\partial b_2} \right).$$

Der erste Faktor $\left(\frac{\partial L}{\partial a_2} \right)$ ergibt sich aus der Ableitung der Fehlerfunktion nach a_2 und lautet $\left(\frac{\partial L}{\partial a_2} \right) = a_2 - y$. Der zweite Faktor $\left(\frac{\partial a_2}{\partial z_2} \right)$ beschreibt die Steigung der

¹ Vgl.: Kneusel, Ronald T. 2022, S. 245-246

² Vgl.: Kneusel, Ronald T. 2022, S. 186-187, 246-249

³ Vgl.: Vgl.: Goodfellow, Ian; Benigo, Yoshua; Courville, Aaron 2016, online, S. 200-207

⁴ Abbildung A2: Beispiel eines neuronalen Netzwerks

Aktivierungsfunktion und ist: $\left(\frac{\partial a_2}{\partial z_2}\right) = a_2(1 - a_2)$. Der letzte Faktor $\left(\frac{\partial z_2}{\partial b_2}\right)$ stammt aus der gewichteten Summe $z_2 = w_4 a_0 + w_5 a_1 + b_2$, welche nach b_2 abgeleitet $\left(\frac{\partial z_2}{\partial b_2}\right) = 1$ ergibt.

Insgesamt ergibt sich:

$$\frac{\partial L}{\partial b_2} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial b_2}\right) = (a_2 - y) a_2 (1 - a_2) * 1$$

Für die Gewichte des Output-Neurons gilt:

$$\frac{\partial L}{\partial w_4} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial w_4}\right) = (a_2 - y) a_2 (1 - a_2) a_0 * 1$$

$$\frac{\partial L}{\partial w_5} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial w_5}\right) = (a_2 - y) a_2 (1 - a_2) a_1 * 1$$

Für Neuron a_1 gilt:

$$\frac{\partial L}{\partial b_1} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial a_1}\right) \left(\frac{\partial a_1}{\partial z_1}\right) \left(\frac{\partial z_1}{\partial b_1}\right) = (a_2 - y) a_2 (1 - a_2) w_5 a_1 (1 - a_1) * 1$$

$$\frac{\partial L}{\partial w_1} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial a_1}\right) \left(\frac{\partial a_1}{\partial z_1}\right) \left(\frac{\partial z_1}{\partial w_1}\right) = (a_2 - y) a_2 (1 - a_2) w_5 a_1 (1 - a_1) x_0$$

$$\frac{\partial L}{\partial w_3} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial a_1}\right) \left(\frac{\partial a_1}{\partial z_1}\right) \left(\frac{\partial z_1}{\partial w_3}\right) = (a_2 - y) a_2 (1 - a_2) w_5 a_1 (1 - a_1) x_1$$

Für Neuron a_2 gilt:

$$\frac{\partial L}{\partial b_0} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial a_0}\right) \left(\frac{\partial a_0}{\partial z_0}\right) \left(\frac{\partial z_0}{\partial b_0}\right) = (a_2 - y) a_2 (1 - a_2) w_4 a_0 (1 - a_0) * 1$$

$$\frac{\partial L}{\partial w_0} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial a_0}\right) \left(\frac{\partial a_0}{\partial z_0}\right) \left(\frac{\partial z_0}{\partial w_0}\right) = (a_2 - y) a_2 (1 - a_2) w_4 a_0 (1 - a_0) x_0$$

$$\frac{\partial L}{\partial w_2} = \left(\frac{\partial L}{\partial a_2}\right) \left(\frac{\partial a_2}{\partial z_2}\right) \left(\frac{\partial z_2}{\partial a_0}\right) \left(\frac{\partial a_0}{\partial z_0}\right) \left(\frac{\partial z_0}{\partial w_2}\right) = (a_2 - y) a_2 (1 - a_2) w_4 a_0 (1 - a_0) x_1$$

Diese Ableitungen zeigen, wie der Fehler schrittweise durch die Schichten zurückpropagiert und auf die einzelnen Parameter verteilt wird.

Der Gradientenvektor fasst alle partiellen Ableitungen der Fehlerfunktion nach sämtlichen Gewichten und Biases zusammen. Jeder Eintrag zeigt, wie sich das Loss ändert, wenn ein Parameter verändert wird. Die Richtung bestimmt, wie die

Parameter verändert werden müssen, und der Betrag, wie stark (vgl. Abbildung A3 im Anhang).^{1,2,3}

5.4 Das Gradientenabstiegsverfahren

Nachdem der Fehler mit Backpropagation durch das Netzwerk zurückgeführt wurde, erfolgt die eigentliche Optimierung des Modells durch das Gradientenabstiegsverfahren. Ziel ist es, die Gewichte und Biases so anzupassen, dass die Kostenfunktion minimiert wird. Die Anpassung geht gegen die Richtung des steilsten Anstiegs der Kostenfunktion. Da der Gradient angibt, wie sich die Kosten bei einer Änderung der Parameter erhöhen, führt eine Subtraktion zur Verringerung des Fehlers. Die Lernrate η fungiert als Skalierungsfaktor für die Korrekturschritte.⁴

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i}, \quad b_i \leftarrow b_i - \eta \frac{\partial L}{\partial b_i}$$

6. Versuch

6.1 Backpropagation implementiert am XOR-Problem

Der Backpropagation-Algorithmus, die Kostenfunktion und die Architektur des Netzes wurden, wie in der Arbeit behandelt, implementiert (siehe Anhang A5). Da das XOR-Problem nicht linear separierbar ist, eignet es sich für diesen Versuch. Das Ziel ist, bei den vier möglichen Inputs die korrekten Ausgaben [0, 1, 1, 0] zu approximieren. Die folgende Tabelle zeigt, wie verschiedene Aktivierungsfunktionen bei nahezu gleichen Parametern abschneiden:

Funktion	Iterationen	Lernrate	Vorzeichen	Max. Betrag	Loss \approx
Sigmoid	25000	0.1	\pm	0.5	0.0008
Tanh	1000	0.1	\pm	0.5	0.0006
ReLU	200	0.1	+	1	0.000001

Diese Ergebnisse zeigen, dass alle Aktivierungsfunktionen das Problem lösen konnten. Das Netzwerk konvergierte mit der Sigmoid-Funktion nahezu immer zuverlässig, während bei Tanh- und insbesondere ReLU-Funktion oft mehrere Versuche gebraucht wurden, aufgrund der zufälligen Initialisierung der Gewichte und Biases, um das Problem zu lösen. Dafür zeigte die ReLU-Funktion die schnellste Konvergenz. Zudem wird deutlich, dass die Tanh-Funktion trotz ähnlicher Kurvenform wie die Sigmoid-Funktion deutlich schneller und besser

¹ Vgl.: Hagan, Martin T.; Demuth, Howard B.; Beale, Mark H.; De Jesús, Orlando 2014

² Vgl.: Kneusel, Ronald T. 2022, S. 184-185

³ Abbildung A3: Darstellung eines Gradientenvektors

⁴ Vgl.: Kneusel, Ronald T. 2022, S. 272-279

abschneidet, weil, wie in Kapitel 4.2.2 beschrieben, die Funktion nullzentriert ist, was den Lernprozess beschleunigt. Bei der ReLU-Funktion trat oft das Dying-ReLU-Problem auf, weswegen bei der Funktion auch die Parameter angepasst sind.¹

6.2 Erweiterte Architektur für komplexere Funktionen

Als weiterführenden Versuch wird die Architektur des Netzes erweitert. Beim ersten Versuch war die Netzwerkarchitektur $2 - 2 - 1$, was die Mindestanforderung zum Lösen des XOR-Problems ist, wie auch in Kapitel 3 nachgewiesen wurde. In Kapitel 2.4 wird zudem erläutert, dass durch die wiederholte Funktionskomposition mit mehreren Schichten komplexere Zusammenhänge approximiert werden können. Im zweiten Versuch wird die Tanh-Funktion verwendet, um diese Eigenschaft von tieferen neuronalen Netzwerken zu untersuchen.

Zur Untersuchung dieses Effekts wurden drei unterschiedlich tiefe Architekturen betrachtet: $2 - 2 - 1$, $2 - 25 - 25 - 25 - 25 - 25 - 1$ und $2 - 150 - 150 - 150 - 150 - 150 - 150 - 150 - 1$. Die daraus entstandenen Entscheidungsgrenzen des Netzwerks sind in Abbildung A4 im Anhang visualisiert. Während das einfache Netzwerk zwei einfache Trennlinien erzeugt, zeigen die beiden komplexeren Netzwerke zunehmend komplexere, stark gekrümmte Entscheidungsgrenzen. Dadurch können deutlich kompliziertere Funktionen approximiert werden.²

Jede zusätzliche Schicht führt eine weitere nichtlineare Transformation des Eingaberaumes durch, wodurch hochkomplexe Entscheidungsflächen entstehen können. Damit bestätigt dieser Versuch experimentell die in Kapitel 2.4 beschriebene Eigenschaft tiefer neuronaler Netze.

7. Fazit

Ziel dieser Arbeit war es, das XOR-Problem als nichtlineares Klassifikationsproblem mithilfe von Vektor- und Differentialrechnung mathematisch zu beschreiben und zu lösen. Durch die Modellierung eines künstlichen Neurons wurde gezeigt, dass dieses allein nicht in der Lage ist, nichtlinear separierbare Probleme zu lösen.

Durch die Einführung nichtlinearer Aktivierungsfunktionen und mehrschichtiger Architekturen wurde diese Einschränkung überwunden. Mithilfe des Versuchs wurde gezeigt, dass ein Netzwerk mit einer $2 - 2 - 1$ Architektur in der Lage ist, das XOR-Problem korrekt zu klassifizieren. Die Kapitel über die

¹ Versuch A5: Quellcode und ausführbare Datei

² Abbildung A4: Entscheidungsgrenze bei verschiedenen Netzwerkarchitekturen

Aktivierungsfunktionen und den Backpropagation-Algorithmus verdeutlichen, wie Gradienten mithilfe von partiellen Ableitungen berechnet werden und zur Optimierung der Gewichte und Biases eingesetzt werden können.

Die Ergebnisse aus dem Versuch bestätigen die theoretisch erarbeiteten Konzepte. Alle behandelten Aktivierungsfunktionen waren in der Lage, das XOR-Problem zu lösen, unterscheiden sich jedoch deutlich in der Konvergenz und Stabilität.

Kritisch anzumerken ist, dass die behandelten Netzwerke sehr klein und die Datensätze stark vereinfacht sind. Weswegen sich der Versuch nicht auf Skalierbarkeit und Generalisierung ableiten lässt. Zudem wurden nicht alle Aktivierungsfunktionen sowie moderne Optimierungsverfahren behandelt, sondern die klassischen, die mit dem XOR-Problem verbunden sind.

Dennoch zeigt die Arbeit, wie sich komplex erscheinende Lernprozesse neuronaler Netze auf überschaubare mathematische Strukturen zurückführen lassen.

8. LITERATURVERZEICHNIS

Goodfellow, Ian; Benigo, Yoshua; Courville, Aaron 2016: Deep Learning. The MIT Press. Online im Internet, URL <https://www.deeplearningbook.org>, Zugriff am 31.01.2026

Hagan, Martin T.; Demuth, Howard B.; Beale, Mark H.; De Jesús, Orlando 2014:

Neural Network Design, 2nd Edition. Oklahoma State University, University of Colorado. PDF <https://hagan.okstate.edu/NNDesign.pdf>

Kneusel, Ronald T. 2022: Math for deep learning. What you need to know to understand neural networks. San Francisco: no starch press

Montesinos-Lopez, Osval A.; Montesinos, Abelardo; Crossa, Jose 2022: Multivariate Statistical Machine Learning Methods for Genomic Prediction. Springer. Online im Internet, URL <https://www.ncbi.nlm.nih.gov/books/NBK583971/>, Zugriff am 09.02.2026

INTERNETSEITEN

Cabello, Julia Garcia 2022: Mathematical Neural Networks. Online im Internet, URL <https://www.mdpi.com/2075-1680/11/2/80>, Zugriff am 27.01.2026

Dubey, Shiv Ram; Singh, Satish Kumar; Chaudhuri, Bidyut Baran 2022:

Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark. Online im Internet, URL <https://arxiv.org/pdf/2109.14545>, Zugriff am 28.01.2026

Kriesel, David 2005: Neuronale Netze. Online im Internet, URL https://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-2col-dkrieselcom.pdf, Zugriff am 24.01.2026

Lee, Minhyeok 2023: The Geometry of Feature Space in Deep Learning Models: A Holistic Perspective and Comprehensive Review. Online im Internet, URL <https://scholarworks.bwise.kr/cau/bitstream/2019.sw.cau/69887/1/The%20Geometry%20of%20Feature%20Space%20in%20Deep%20Learning%20Models%3B%20A%20Holistic%20Perspective%20and%20Comprehensive%20Review.pdf>, Zugriff am 10.02.2026

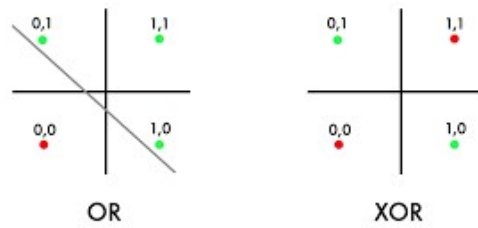
Wang, Shengjie; Zhou, Tianyi; Bilmes, Jeffery A. 2019: Bias Also Matters: Bias Attribution for Deep Neural Network Explanation. In Proceedings of the 36th International Conference on Machine Learning. Online im Internet, URL <https://proceedings.mlr.press/v97/wang19p/wang19p.pdf>, Zugriff am 08.02.2026

Unbekannter Autor unbekanntes Jahr: Künstliches Neuron. Online im Internet, URL https://de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron, Zugriff am 18.01.2026

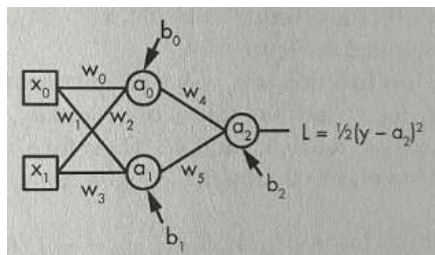
Unbekannter Autor unbekanntes Jahr: Lineare Separierbarkeit. Online im Internet, URL https://de.wikipedia.org/wiki/Lineare_Separierbarkeit, Zugriff am 24.01.2026

Unbekannter Autor unbekanntes Jahr: Universal approximation theorem. Online im Internet, URL https://en.wikipedia.org/wiki/Universal_approximation_theorem, Zugriff am 18.01.2026

9. ANHANG



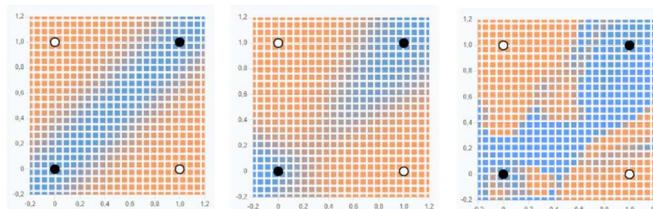
A1: Darstellung des XOR-Problems, URL <https://media2.dev.to/dynamic/image/width=1280,height=720,fit=cover,gravity=auto,format=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Fkl02223oqhlac1jetz.png>, Zugriff am 08.02.2026



A2: Beispiel eines neuronalen Netzwerks, Kneusel, Ronald T., S. 245

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_0} \\ \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial w_3} \\ \frac{\partial L}{\partial w_4} \\ \frac{\partial L}{\partial w_5} \\ \frac{\partial L}{\partial b_0} \\ \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial b_2} \end{bmatrix}$$

A3: Darstellung eines Gradientenvektors



A4: Entscheidungsgrenze bei verschiedenen Netzwerkarchitekturen, Eigene Messdaten



Facharbeit_Versuch.zip

A5: Quellcode und ausführbare Datei

Der Quellcode sowie die .exe-Datei sind zudem unter folgendem Link verfügbar:

https://github.com/Fusraumspinne/Neural_Network_Versuch

Zusätzlich wird der Code unter folgendem Link gehostet: <https://neural-network-versuch.onrender.com/>

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die im Literatur- und Quellenverzeichnis angegebenen Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe. Jede Nutzung von KI-basierten Hilfsmitteln (ChatGPT oder ähnliche) wurde vor dem Verfassen der Facharbeit mit dem Fachlehrer/der Fachlehrerin abgesprochen und ist in der Facharbeit z.B. durch eine Fußnote deutlich gekennzeichnet. Mir ist bewusst, dass ein Verstoß gegen diese Regeln zu einer Bewertung mit der Note "ungenügend" führen kann.

Ort / Datum

Unterschrift