St.-Gotthard-Gymnasium

der Benediktiner Niederalteich

ST.-GOTTHARD-GYMNASIUM
NIEDERALTEICH

Künstliche Neuronale Netze

vorgelegt
von

Julian Harrer

am
10.11.2020

Qualifikationsphase 2019-2021

W-Seminar: Mathematik – Digitalisierung von Musik und Sprache

Seminarleiter: Frau Isolde Mazzucco

Inhaltsverzeichnis

Abbildungsverzeichnis

Alle Grafiken wurden selbst mit den folgenden Open-Source Programmen erstellt:

- Inkscape (inkscape.org)
- Geogebra (geogebra.org)
- Audacity (audacityteam.org)

Als Vorlage und Inspiration dienten die Illustrationen in der im Literaturverzeichnis genannten Literatur.

1 Einleitung

Die Menge der Menschen, die digitale Sprachassistenten in ihrem Alltag nutzen, hat seit dem Jahr 2016 fast stetig zugenommen[1]. Die Technologie, die diesem Fortschritt zu verantworten hat, wird „maschinelles Lernen“ oder auch „Deep Learning“ genannt. Jedoch können die Meisten mit diesen Begriffen nur wenig anfangen.

Aus diesem Grund stellt sich die Frage, was es mit dem Begriff der „Künstlichen Neuronalen Netze“ bzw. des „Deep Learnings“ auf sich hat. Wie sind diese vereinfacht auf mathematischer Ebene aufgebaut und welche grundsätzlichen Varianten gibt es? Wie sieht das gängigste Verfahren für das Trainieren dieser Netze aus und was sind die Voraussetzungen dafür? Wo können sie eingesetzt werden und wie setzt man diese in der Praxis um?

Daher ist es das Ziel dieser Arbeit, dem Leser einen kurzen Einblick in die Welt des maschinellen Lernens mit künstlichen neuronalen Netzen und deren Verwendung bei der digitalen Verarbeitung von Sprache zu geben. Sie behandelt dabei jedoch nur die gängigsten Vorgehensweisen auf rudimentärer Ebene. Dabei wird sie sich auf Grund des Faches und Themas des Seminars, sowohl auf die mathematische, als auch auf die praktische Umsetzung beziehen. Der Rahmen der Arbeit umfasst also

- die Definition und Erklärung von künstlichen neuronalen Netzen,
- die mathematischen Herangehensweisen an dadurch entstehende Probleme,
- den Wert des Konzeptes im Bezug auf die digitale Verarbeitung und das Verständnis von Sprache und
- die konkrete Umsetzung des gelernten in einem praktischen Beispiel.

Die gerade genannte Vorgehensweise soll dem Leser auf eine möglichst unkomplizierte Weise ein grobes Verständnis dafür geben, wie der Kern dieser Alltagstechnologie aufgebaut ist.

Der Grund für die rudimentäre Herangehensweise ist hauptsächlich durch zwei Gegebenheiten begründet: Zum einen die vorausgesetzte Kürze des vorliegenden Dokuments und zum anderen die Tiefe des benötigten mathematischen Wissens, das für größere Ausführungen benötigt werden würde, jedoch nicht dem Wissensstand der 12. Jahrgangsstufe entspricht. Dinge, wie die partielle Ableitung oder die gehobene Notation für mathematische Ausdrücke, mussten eigenständig erarbeitet werden, um die schiere Menge an Fachliteratur zu verstehen.

1.1 Begriffserklärung und Definitionen

Ein Künstliches Neuronales Netz (KNN) ist, wie es der Name schon vermuten lässt, eine künstliche Nachahmung der natürlichen neuronalen Netze bzw. deren grundlegenden Aufbau mit Neuronen und Synapsen. D.h. es wird versucht, das Verhalten von kognitiven Vorgängen in Organismen mathematisch als Funktion im Computer zu reproduzieren, um

[1] Volker Wittpahl, Künstliche Intelligenz Technologie | Anwendung | Gesellschaft, S.8 dritter Absatz

damit vielseitige Lernaufgaben zu bewältigen. Dabei werden genau diese Lernaufgaben bei hoher Intensität und Komplexität des KNNs als Deep Learning -also tiefes Lernen- bezeichnet. Mit tiefem Lernen sind hier Netzwerke mit „einigen bis vielen Schichten“ (Wittpahl, 2019) gemeint[2] (Genaueres folgt zu einem späteren Zeitpunkt in dieser Arbeit.). Grundsätzlich werden Künstliche Neuronale Netze immer gerne eingesetzt, wenn es um die Verarbeitung von großen Datenmengen und um die Erkennung von Mustern in Datensätzen geht[3].

1.2 Abgrenzung und Inspiration von der Natur

Obwohl diese Technologie von der Natur inspiriert wurde, muss im Zuge von Fortschritten in der Neurobiologie die Unterschiede zwischen einem KNN und einem neuronalen Netz aus eben diesem Bereich klar abgegrenzt werden. Diese unterscheiden sich sowohl in der Art und Weise, wie die Neuronen untereinander verbunden sind, als auch in dem, wie diese die Wichtigkeit von Impulsen bewerten[4]. Jedoch besteht eine grundlegende Gemeinsamkeit der neuronalen Netze in der Natur und deren in der Informatik. Beide wurden prinzipiell für das Suchen bzw. Erkennen von Mustern in der Umgebung geschaffen und werden für diesen Zweck demnach am erfolgreichsten eingesetzt.

3 Das künstliche neuronale Netz

3.1 Grundlegender Aufbau

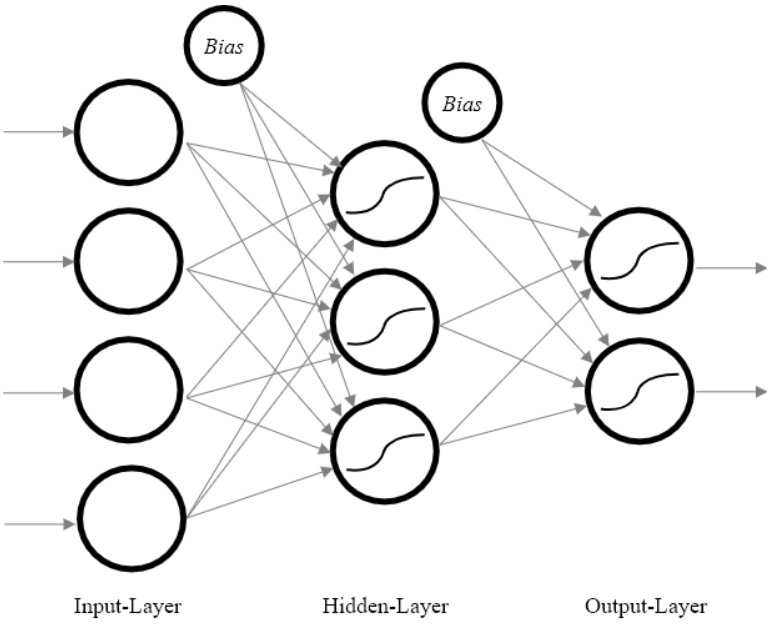

Abbildung 1: Illustration eines künstlichen neuronalen Netzes

[2] Prof. Dr. Volker Wittpahl Künstliche Intelligenz Technologie | Anwendung | Gesellschaft S. 29
[3] ebenda S. 32, zweiter Absatz
[4] ebenda S. 30,31

Ein künstliches neuronales Netzt besteht grundlegend aus imaginären Neuronen (*u*) -auch Units genannt-, welche in unterschiedlichen Schichten angeordnet sind. Jedes Neuron einer Schicht ist mit jedem Neuron in der nächsten Schicht und einem Bias (*b*) verbunden (vgl. Abb. 1). Um die Relevanz jeder Verbindung zu bewerten, besitzt jede Verbindung ein Gewicht *w*. Dabei wird zwischen den *feed forward* und den *recursiven* Netzwerken unterschieden. Bei Ersteren laufen alle Verbindungen in eine Richtung, während bei Letzterem auch manche in die entgegengesetzte Richtung wieder in das Netz fließen können. Diese Arbeit wird sich jedoch nur mit den *feed forward networks* (*vorwärts durchlaufend*) beschäftigen, da dies die gängigste Architektur ist[5]. Bei den Layern (*dt. Schicht*) wird zwischen drei Typen unterschieden: Dem Input-Layer, in die der Input *x* eingegeben wird, den Hidden-Layer und den Output-Layer mit ihrer Ausgabe *out*. Jedes dieser Neuronen ist eine Zusammensetzung aus drei Funktionen, mit dessen Hilfe es seine Eingabewerte in einen Ausgabewert übersetzt[6]:

Netzeingabefunktion: $f_{net}(x_n, w_n) = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$

➔ Diese multipliziert die eingehenden Werte (x_n) mit den dazugehörigen Gewichtungen (w_n) und addiert diese Produkte.[7]

Aktivierungs- und Ausgabefunktion: $f_{out}\left(f_{act}\left(f_{net}(x_n, w_n)\right)\right) = out_u$

Sowohl für die Aktivierungsfunktion f_{act}, als auch für die Ausgabefunktion f_{out} gibt es diverse Möglichkeiten. Die meist verwendete ist die *Sigmoid-Funktion*, welche den errechneten Wert der Netzeingabefunktion in eine Zahl zwischen 0 und 1 konvertiert und zudem differenzierbar ist, was sich in späteren Kapiteln noch als wichtiges Kriterium erweisen wird[8]. Je nach Anwendungsfall sind aber auch beispielsweise Lineare- oder Logarithmusfunktionen möglich[9].

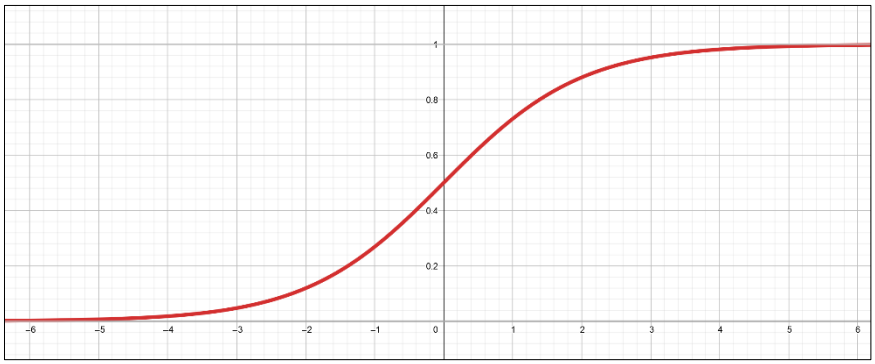

Abbildung 2: Sigmoidfunktion

[5] Einführung in die Automatische Spracherkennung an der Universität München, Vertiefung ANN, Seite 51 oben, Ansichtsdatum 5.11.2020
[6] Rudolf Kruse u. weitere, Computational Intelligence Eine methodische Einführung S.34 und Thamm, S. (2019), Einführung in Neuronale Netze, Universität Köln, S. 5
[7] ebenda S.36 unten
[8] ebenda S.64 Abschnitt 5.6
[9] Artikel www.ai-united.de/aktivierungsfunktionen-neuronale-netze/ am 01.11.2020 um 19:50

3.2 Neuronen und Matrizen

Es hat sich durchgesetzt, große Berechnungen, wie die der drei Funktionen in einem künstlichen neuronalen Netz in Form von Matrizen zu realisieren. Dies sorgt vor Allem für eine Erhöhung der Verarbeitungsgeschwindigkeit auf der Ebene des Computers. Somit kann eine große Funktion in kleinere Berechnungen in Form von Matrizenmultiplikationen gespalten werden. Dies ist deshalb vom Vorteil, da so die Maschine die Rechenarbeit parallelisieren kann, was heißt, dass viele Rechenprozesse gleichzeitig laufen können[10].

Alle Gewichtungen *w* der Verbindungen zwischen den Neuronen zweier Schichten, werden zu einer Matrix zusammengefasst. Das Gleiche geschieht mit den Werten aus der vorausgehenden Neuronenschicht. Beide Matrizen multipliziert, ergeben dann eine einzelne Matrix, welche die folgende Neuronenschicht repräsentiert[11].

$$\begin{pmatrix} w_{u_1u_1} & \cdots & w_{u_1u_r} \\ \vdots & \ddots & \vdots \\ w_{u_ru_1} & \cdots & w_{u_ru_r} \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_r \end{pmatrix} = \begin{pmatrix} (u_r * w_{u_1u_1}) + (u_1 * w_{u_1u_r}) \\ \vdots \\ (u_r * w_{u_ru_1)} + (u_1 * w_{u_ru_r}) \end{pmatrix}$$

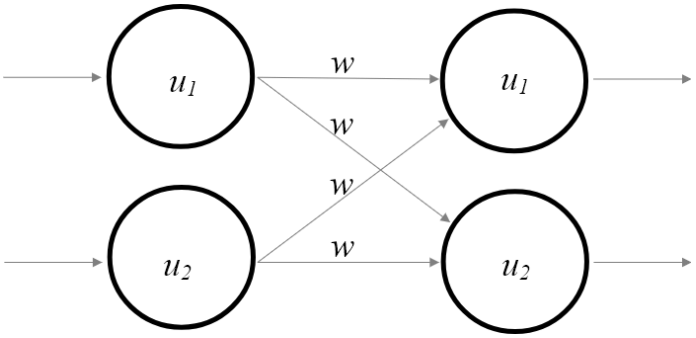

Abbildung 3: Beispiel für ein KNN zur Matrixmultiplikation

Moderne Computer enthalten sogenannte GPUs, welche durch ihre hohe Anzahl an Rechenkernen besonders viele Aufgaben parallel berechnen können. Dies ist einer der Hauptgründe, warum die Matrizendarstellung im Bezug auf dieses Thema eine große Relevanz genießt[12].

4 Das Lernen

4.1 Grundlagen für das Lernen

Das Lernen oder auch das Training von künstlichen neuronalen Netzen ist einer der zentralen Bestandteile dieser Technologie, da es sich andernfalls nur um ein Gebilde von Zufallszahlen in Form der Gewichte handeln würde. Mit dem Training werden genau diese

[10] Volker Wittpahl, Künstliche Intelligenz Technologie | Anwendung | Gesellschaft, Seite 38 oben
[11] ebenda S.37 unten
[12] ebenda S. 38 oben

Zahlen zu dem, was die „Intelligenz“ des Systems ausmacht[13]. Wenn dies nicht geschieht, würde das Netz nur zufällige Werte ausgeben, welche keinerlei Muster oder Sinn aufweisen. Um dies zu erreichen, also das Erlernen eines Musters und die Anpassung des Netzes, gibt es diverse Verfahren. Diese Arbeit wird sich jedoch mit den beiden Grundlegendsten und Häufigsten beschäftigen. Jedes dieser so genannten *Lernverfahren* folgt jeweils einer *Lernaufgabe*, bei denen zwischen *frei* und *fest* bzw. *überwacht* (*engl. supervised*[14]) und *unüberwacht* (*engl. unsupervised*[10]) unterschieden wird.

Feste Lernaufgaben (*supervised*): Hierfür muss zu den eingegebenen Daten *x* (*engl. input*) eine gewünschte Ausgabe *o* (*engl. output*) feststehen. Die Lernaufgabe ist dann erfüllt, wenn die Eingabe zur Ausgabe passt. Als Beispiel ist hier ein klassisches KNN zur Spracherkennung zu nennen, bei welchem, vereinfacht gesehen, die Eingabe *x* aus einem gesprochenen Wort und die Ausgabe *o* aus dem dazu passenden Transkript besteht. Die Ausgabe ist somit fest. Zudem handelt es sich hierbei um die gängigste Lernmethode[15].

Freie Lernaufgaben (*unsupervised*): Bei diesem Verfahren steht die gewünschte Ausgabe nicht ganz fest[16], sondern besitzt eine gewisse Streuung. Es soll also „für ähnliche Eingaben ähnliche Ausgaben liefern“ (Rudolf Kruse, 2015).

Beide Lernaufgaben dienen dazu, die Akkuratheit des KNN zu bestimmen und Abweichungen (*e*) zu minimieren, also ein Optimum zu finden. Tiefergehende Ausführungen wird diese Arbeit jedoch nur dem überwachten Lernen widmen[17].

4.1.1 Die Fehlerfunktion

Die Aufgabe der *Fehlerfunktion* (*engl. cost function*) ist es, die Abweichung der Ausgabe des Netzes vom Soll-Zustand zu berechnen, um einen konkreten Anhaltspunkt zur Optimierung des Netzes zu haben[18].

$$e_u = (o_u - out_u)^2$$

Hier ist ein Beispiel für eine Fehlerfunktion zu sehen, wobei *o* die erwartete und *out* die tatsächliche Ausgabe des Neurons *u* ist. Der Fehlerwert ist also die Differenz zwischen beiden Werten (*o* und *out*) und kann bei mehreren Ausgangsneuronen auch als Vektor ausgedrückt werden. Die Quadratur dient zudem dazu, Vorzeichen im Fehler *e* zu eliminieren, da diese sonst Werte aufheben könnten. Der *Betrag* wird nicht verwendet, da dieser im Punkt 0 keine Ableitung besitzt und daher nicht vollständig differenzierbar ist[19].

Dies lässt bereits auf die Absicht der folgenden Lernverfahren schließen: Das Ziel dieser Verfahren ist es nämlich, genau diesen Wert des Fehlers *e* weitestgehend zu minimieren, d.h. ein **möglichst** globales Minimum zu finden. Meist ist dies jedoch in der Praxis nicht

[13] Thamm, S. (2019), Einführung in Neuronale Netze, Universität Köln, S. 27
[14] Volker Wittpahl, Künstliche Intelligenz, Erschienen 2019, Seite 24 unten (Maschinelles Lernen)
[15] ebenda, Seite 163 unten (Übung macht den Meister)
[16] ebenda, Seite 26 letzter Absatz
[17] ebenda, Seite 32 zweiter Absatz
[18] Rudolf Kruse u. weitere, Computational Intelligence Eine methodische Einführung, Seite 41 oberer Term
[19] Rudolf Kruse u. weitere, Computational Intelligence Eine methodische Einführung S. 41 oben

ganz möglich, da mit zunehmender Größe und Komplexität des Netzes auch die Menge an lokalen Minima zunimmt und das Finden des globalen Minimums fast unmöglich macht[20].

Eine gängige Methode, um ein Minimum in einer Funktion mit *n* Variablen zu finden, ist das *Gradientenabstiegsverfahren (engl. gradient descent)*, welches die Grundlage für die folgenden Lernverfahren bildet. Zunächst soll jedoch erklärt werden, wie dieses Verfahren im groben funktioniert[21].

4.2 Gradientenabstiegsverfahren

Die Funktion des Gradientenabstiegsverfahrens (*engl. gradient descent*) ist es, die Variablen, also die Gewichte und Biases, des Netzes so anzupassen, dass die Ausgabewerte der *Fehlerfunktion* (*engl. cost function*) weitestgehend minimiert wird und somit die Sicherheit des Netzes in seiner Entscheidungsfindung zu erhöhen[22]. Das Verfahren bildet die Grundlage für den Fehlerrückführungs-Algorithmus (*engl. error backpropagation*), welcher zu einem späteren Zeitpunkt in dieser Arbeit noch genauer erläutert wird. Er geht dabei iterativ vor, was einen großen Variationsspielraum für die Art und Weise der Umsetzung zulässt (Batch- und Online-Training[23]).

Gradient Definition: Ein Gradient ist ein Spaltenvektor der ersten partiellen Ableitung einer Funktion mit *n* Variablen. Dieser beschreibt die Richtung und Stärke der größten Steigung in einem Punkt p in Form eines Vektors[24].

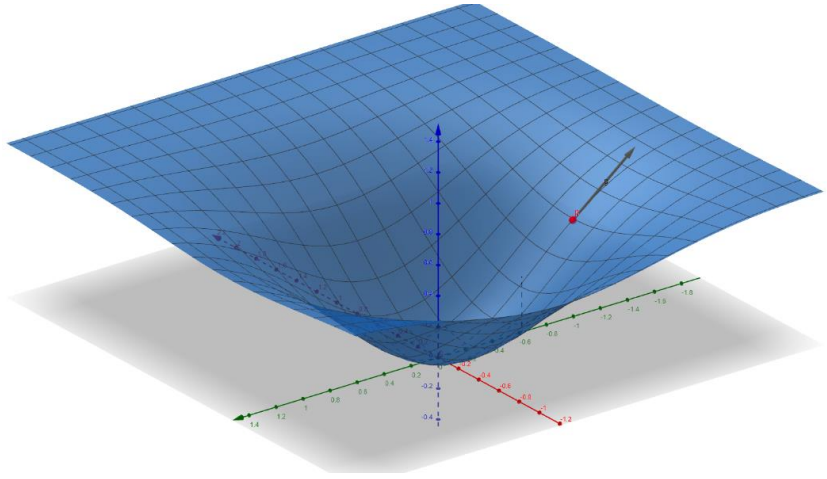

Abbildung 4: Gradient in einem Punkt einer Funktion mit zwei Variablen

Erläuterung der Funktionsweise des Verfahrens

[20] ebenda, S. 139 erster Absatz
[21] Thamm, S. (2019), Einführung in Neuronale Netze, Universität Köln, S. 19
[22] Rudolf Kruse u. weitere, Computational Intelligence Eine methodische Einführung S. 58 Kapitel 5.6
[23] Rudolf Kruse u. weitere, Computational Intelligence Eine methodische Einführung, Seite 24, erster und zweiter Absatz
[24] https://www.calculushowto.com/gradient/ Ansichtsdatum: 5.11.2020

Um den Gradienten der Funktion *f* zu erhalten, wird diese zunächst partiell abgeleitet. D. h. es wird die Funktion für jede Variable einzeln betrachtet und nach dieser abgeleitet. Dabei werden die restlichen Variablen wie eine konstante Zahl behandelt[25].

Beispiel: Hierfür dient die Funktion *f* der oben gezeigten Abbildung,

$$f(x,y) = -e^{-x^2-y^2}$$

welche nun partiell abgeleitet wird.

$$f'_x(x,y) = -e^{-x^2-y^2} * -2x$$
$$f'_y(x,y) = -e^{-x^2-y^2} * -2y$$

Nun wird der Gradient mithilfe des Nabla-Operators in einem Vektor dargestellt.

$$\nabla f(x,y) = \begin{pmatrix} -e^{-x^2-y^2} * -2x \\ -e^{-x^2-y^2} * -2y \end{pmatrix}$$

Der Punkt P(0.5, 0.5, z) kann nun in den Gradienten eingesetzt werden, um den Richtungsvektor der größten Steigung zu erhalten.

$$\nabla f(0.5, 0.5) \approx \begin{pmatrix} 0{,}607 \\ 0{,}607 \end{pmatrix}$$

Um den Punkt P jetzt in Richtung des Minimums zu verschieben, wird der *Gegenvektor* benötigt, welcher auf den Punkt P addiert wird. Um den *Gegenvektor* zu erhalten, muss der ursprüngliche Vektor nur mit -1 multipliziert werden.

Fazit zum Verfahren

Es wird klar, warum der Betrag für die Fehlerfunktion aufgrund seiner schlechten Ableitbarkeit nicht geeignet ist, denn diese Ableitung ist eine Voraussetzung für das Gradientenabstiegsverfahren. Dieses Verfahren findet später bei der Fehlerrückführung noch tiefere Anwendung, ist jedoch grundlegend für die meisten Lernverfahren im Bereich der künstlichen neuronalen Netze, wenn eine Fehlerfunktion aufgrund der Lernaufgabe vorhanden ist.

4.3 Delta Lernregel

Die Deltalernregel ist ein einfaches und gängiges Verfahren zur Lösung des Lernproblems und lässt sich auf *Bernard Widrow* und *Marcian Hoff* zurückführen[26]. Dies ist eine vereinfachte Form der Fehlerrückführung und setzt ebenso auf eine Art des Gradientenabstiegsverfahrens. Jedoch kann der Algorithmus nur bei Netzen mit einer Ebene verwendet werden. D.h. bei Netzen, welche keine versteckten Schichten bzw. Hidden-

[25] https://studyflix.de/mathematik/gradient-berechnen-1350, Ansichtsdatum: 5.11.2020
[26] Strecker Stefan, Künstliche Neuronale Netze – Aufbau und Funktionsweise, Ansichtsdatum: 06.11.2020

Layers besitzen[27].

Vorgehensweise

Es wird zunächst die Fehlerdifferenz für jedes Ausgabeneuron bestimmt.

$$\delta_i = (o_u - out_u)$$

Der ausgegebene Fehlerwert dient im Folgenden als Faktor zur Bestimmung der Änderungsrate des jeweiligen Gewichtes w_{ij},

$$\Delta w_{ij} = \varepsilon * \delta_j * x_i$$

wobei ε die sog. *Lernrate* darstellt. Dies ist ein experimentell bestimmter Wert, welcher die Schrittweite des Gradientenabstiegs bestimmt und im Intervall *I* liegt. Hierbei stellt *x* den Input des Neurons *i* in das Netz dar.

$$I =]\,0\,;\,1\,]$$

Der letzte Schritt dieses Delta-Algorithmuses besteht dann nur noch darin, die Änderungsrate Δw_{ji} mit dem gegebenen Gewicht w_{ij} zwischen dem jeweiligen Neuron der Input- und der Output-Schicht zu addieren. Dabei ist dieser Algorithmus ebenfalls iterativ und sorgt nur für eine Annäherung an das lokale Minimum.

$$w^{(neu)}{}_{ij} = w^{(alt)}{}_{ij} + \Delta w_{ij}$$

Somit ist das gerade gezeigte Verfahren eine Grundlage für das maschinelle Lernen und die folgenden Verfahren[28].

4.4 Backpropagation bzw. Fehlerrückführung

Bei der Backpropagation bzw. Fehlerrückführung handelt es sich um eine Erweiterung der Delta-Lernregel. Sie ist, im Gegensatz zur dieser, nicht auf die Input- und Output-Neuronen beschränkt. Es handelt sich um einen Algorithmus, welcher die Fehlerfunktion des Outputs in einen Gradienten umformt und die Gewichtungen und Biases dann schrittweise zurück (*engl. back*) durch das Netz propagiert[29].

Um die Mathematik hinter dem Fehlerrückführungsverfahren besser verstehen zu können, werden sich die Erläuterungen in den folgenden Zeilen nur auf ein „Netzwerk“ mit **einem** Neuron pro Schicht beziehen.

Begonnen wird mit der Definition der schon bekannten skalaren Fehlerfunktion am Output des Netzes, in dem die tatsächlichen Ausgaben *out* von der gewünschten Ausgaben *o* subtrahiert und die Differenzen dann addiert werden[30].

[27] Patrick Werner und Ploner Moritz Klaukien, Referatszusammenfassung Forschungsseminar Neuronale Netze, Kapitel 9.1
[28] Rudolf Kruse u. weitere, Computational Intelligence Eine methodische Einführung S. 26
[29] ebenda, S. 64 Abschnitt 5.6
[30] Beschreibung des Backpropagation Lab von Martin Reiche Abschnitt 7.2.1 (Ansichtsdatum: 3.11.2020)

$$E = (o - out)^2$$

Diese Funktion umfasst nun auch indirekt alle Rechnungen, welche zuvor in dem Neuronalen Netz durchgeführt wurden. Dies kann auch folgendermaßen ausgedrückt werden:

$$o = f_{act}(z)$$

In der Fehlerfunktion: $E = (f_{act}(z) - out)^2$

$$z = w * o^{(Vorgänger)} + b$$

Die Variable $o^{(Vorgänger)}$ beinhaltet wiederholt den selben Ausdruck, nur, dass dieser zum Vorgängerneuron gehört, wobei *b* wieder den einfließenden Bias und *w* das Gewicht der Verbindung repräsentiert.

Aus diesem Schema lässt sich erkennen, dass ein KNN nur eine tief ineinander verschachtelte Funktion ist, bei welcher somit die *Kettenregel* angewandt werden muss, um sie partiell abzuleiten, da jeder Teil von dem Vorherigen abhängig ist. Diese Gegebenheit kann in differenzierter Form so dargestellt werden:

$$\frac{\partial E}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial f_{act}(z)}{\partial z} \frac{\partial E}{\partial f_{act}(z)}$$

Nun kann jeder Faktor einzeln abgeleitet werden.

$$\frac{\partial z}{\partial w} = {f_{act}}^{(Vorgänger)}$$

$$\frac{\partial E}{\partial f_{act}(z)} = 2(f_{act}(z) - out)$$

$$\frac{\partial f_{act}(z)}{\partial z} = {f'}_{act}(z)$$

Vorläufiger Gradient: $\frac{\partial E}{\partial w_{pu}} = {f_{act}}^{(Vorgänger)} * {f'}_{act}(z) * 2(o - out)$

Die erhaltene Funktion beschreibt einen Gradienten, welcher in Richtung des größten Anstiegs zeigt. D.h., um die Korrekturrichtung für die Funktion des KNNs zu erhalten, muss dieser noch invertiert bzw. umgekehrt werden. Am Ende des Verfahrens sollte eine Funktion stehen, welche den Änderungswert bzw. im erweiterten Fall den Änderungsvektor für die Gewichtungen *w* zurückgibt.

$$\Delta w = a * {f_{act}}^{(Vorgänger)} * {f'}_{act}(z) * 2(o - out) * (-1)$$

Wobei *a* die Lernrate beschreibt, also die Schrittweite, mit welcher der Algorithmus die Variablen bei jeder Iteration anpasst. Dieser Wert liegt, wie bei der Deltaregel, im Intervall I =] 0 ; 1] und sollte weder zu groß, noch zu klein ausfallen. In der Praxis liegt dieser häufig bei ca. 0,01.

Um die Änderungen nun wirksam zu machen, werden der Änderungsvektor $\overrightarrow{\Delta w}$ und der Vektor der aktuellen Gewichte $\vec{w}$ miteinander addiert. Dieser Vorgang kann nun weiter durch das Netz gegeben werden. Er wird somit zurückpropagiert (*daher back propagation*).

Eine fast identische Vorgehensweise, muss nun auch noch für den Bias wiederholt werden. Jedoch wird diese Arbeit aus Platzgründen darauf nicht eingehen.

Danach kann das Netz mit neuen Beispielen „gefüttert“ (*engl. to feed*) werden und der Prozess beginnt von Vorne.

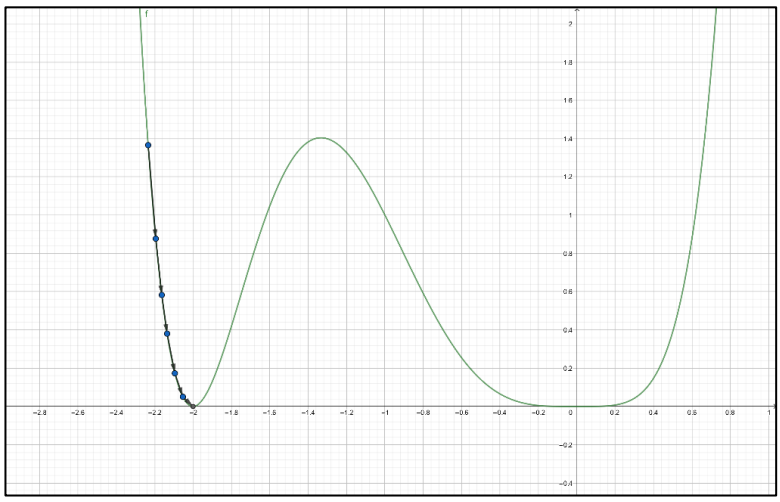

Abbildung 5: Grafische Darstellung des Gradientenabstiegsverfahrens bei einer eindimensionalen Funktion im Optimalfall bei niedriger Lernrate.

Das gezeigt Vergehen ist in der Literatur *Rudolf Kruse u. weitere, Computational Intelligence Eine methodische Einführung* im Kapitel 5.5 und 5.6 zu sehen.

4.5 Evolution als Beispiel für unüberwachtes Lernen

Die evolutionären Lernverfahren sind eine weitere Art des Lernens, welche dem Grundprinzip des Nachahmens der Natur eine noch größere Bedeutung geben. Diese Lernverfahren setzen auf das darwin‘sche Prinzip der Evolution. Dabei werden Evolutionsvoraussetzungen wie Mutation, Selektion und Populationen mit Algorithmen künstlich im Computer nachgestellt. Sie bilden eine weitere Art der Lösungsfindung beim Optimierungsproblem, sind dabei aber auch nur Näherungsverfahren[31].

Zudem gehören sie zu dem Zweig des unüberwachten Lernens (*engl. unsupervised lerning*), was bedeutet, dass sie eine freie Lernaufgabe besitzen. Es werden also Daten in das Netz „gefüttert“ (vgl. *feed forward*), aber es gibt keine korrespondierenden Ausgabedaten. Es soll nur Ähnliches mit Ähnlichem zusammengebracht werden.

[31] Rudolf Kruse u. weitere, Computational Intelligence Eine methodische Einführung Kapitel 10.2 und 10.3

5 Künstliche Neuronale Netze und Sprache bzw. Audio

5.1 Anwendung

Eine äußerst beliebte Anwendung für die Technologie des Deep Learning ist die virtuelle Verarbeitung von Sprache und Audio. Dies rührt daher, dass Sprache das natürliche Kommunikationsmittel des Menschen ist und es daher naheliegt, es auch zur intuitiven Kommunikation mit dem Computer zu verwenden. Beispielsweise ist es so möglich, der Maschine einen Befehl zu geben, welcher nicht von einer direkten Nähe abhängt. D.h., es ist kein direktes physisches Interagieren wie z.B. beim Druck einer Taste notwendig. Dies ist dabei keine Zukunftstechnologie, sondern findet bereits tägliche Anwendung: Sei es beim Sprechen mit den bekannten Sprachassistenten „Siri“ oder „Alexa“ auf mobilen Endgeräten oder beim Beantworten von häufig gestellten Fragen bei einer Hotline. Auch kann es beim Einsatz des unüberwachten Lernens in der Trainingsphase völlig neue Möglichkeiten der Kreativität eröffnen, wie beispielsweise beim Komponieren eines Musikstückes durch ein KNN. Im Folgenden wird sich jedoch aus Platzgründen nur auf Ersteres kurz bezogen[32].

5.2 Sprache aus digitaler Perspektive

Zunächst ist zu klären, wie Sprache aus Sicht des Computers aussieht und wie dieser Muster erkennen soll.

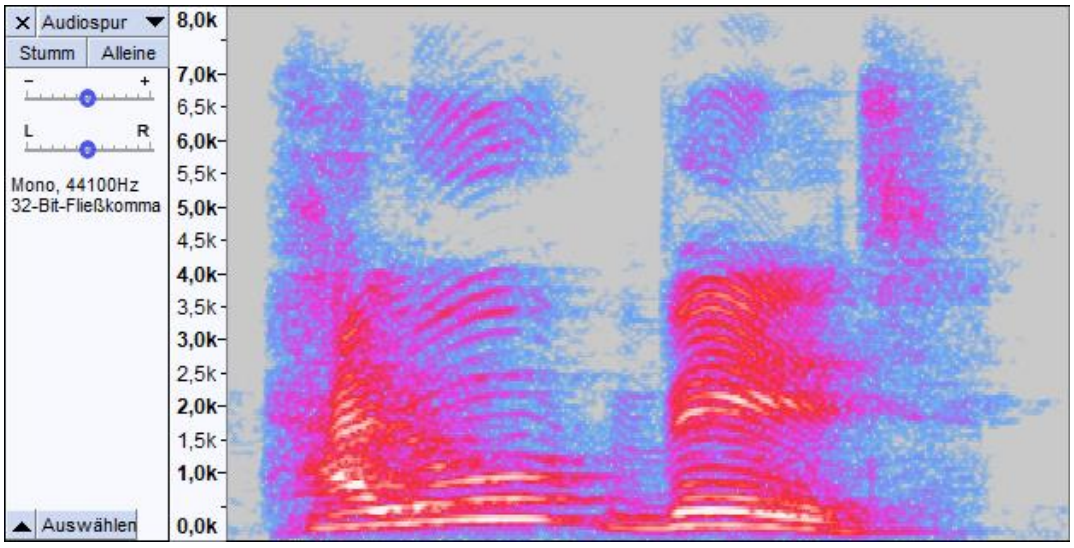

Abbildung 6: Spektrogramm einer digitalen Aufnahme der Wörter "Hallo Welt"

Hier ist ein so genanntes Spektrogramm zu sehen. Es zeigt die Stärke einer Frequenz (vgl. y-Achse) in einer Aufnahme mit Hilfe von Farbtönen an. Dabei bedeuten niedrige Frequenzen tiefe Töne und hohe Frequenzen hohe Töne. Mit solch einem Spektrogramm lässt sich auf die Klangcharakteristik jedes einzelnen Wortes schließen, da jeder

[32] Volker Wittpahl, Künstliche Intelligenz Technologie | Anwendung | Gesellschaft, S.155

Laut der Sprache aus bestimmten Frequenzen besteht, welche sich mit einem Fingerabdruck vergleichen lassen[33].

5.3 Aritificial Speech Recognition (ASR)

Nun kommt es zum Einsatz der mittlerweile gut bekannten künstlichen neuronalen Netze. Die Aufgabe dieses Netzes ist es aus dem oben gezeigten Spektrogramm die so genannten Phoneme, aus denen die menschliche Sprache besteht, zu erzeugen. Diese Phoneme werden dann mit Hilfe einer Art Wörterbuch in Buchstaben und Worte interpretiert.

Es gibt viele verschiedene Herangehensweisen, wenn es zur Konstruktion eines Netzes für diese Aufgabe kommt. Jedoch ist die Folgende eine der Beliebtesten: Aus dem Spektrogramm wird ein Inputvektor für das KNN generiert. Dieser wird in die Eingabeschicht des neuronalen Netzes eingegeben. Das Netz sollte dabei klassischer Weise und aufgrund der Komplexität der Aufgabe, mindestens ein Hidden-Layer (*dt. versteckte Schicht*) haben. Die Ausgabeschicht besitzt dann die gleiche Menge an Neuronen, wie Phoneme in der aufgenommenen Sprache vorhanden sind. So kann durch das „Aufleuchten“ eines bestimmten Neurons das gesprochene Phonem identifiziert bzw. klassifiziert werden[34].

Die gängigste Form des Trainierens eines Netzes auf die gegebene Sprache ist hier, wie so oft, die *error backpropagation*, was bedeutet, dass es sich um einen überwachten Lernprozess mit einer festen Lernaufgabe handelt. Das bedeutet, dass Trainingsdaten benötigt werden. Diese bestehen wiederum aus einer digitalen Audioaufnahme und dem dazugehörigen, durch einen Menschen erstelltes Transkript[35].

5.4 Weitere Aspekte und Methoden

Hinzuzufügen sei, dass die oben genannten Interpretationen der aufgenommenen Sprache natürlich auch auf eine gewisse stochastische Einordnung stützt. Damit konkret gemeint ist, dass der Computer versucht, auf seine Sinnhaftigkeit zu überprüfen, um die Wahrscheinlichkeit auf das korrekte Erkennen von Wörtern zu erhöhen. D.h. es werden stochastische Untersuchungen unternommen, um die Wahrscheinlichkeit zu berechnen, wie oft Wörter miteinander in einem Satz oder hintereinander auftreten[36].

Beispiel:

Der Satz „Im Ofen werden Kekse gebacken.“ ist semantisch wahrscheinlicher als der Satz „In Kühlschrank Hunde bellen groß.“, welcher zudem auch syntaktisch falsch ist.

[33]Einführung in die Automatische Spracherkennung an der Universität München, Grundlagen 1, Seite 8, Ansichtsdatum 4.11.2020
[34] ebenda, Vertiefung ANN, Seite 52 oben, Ansichtsdatum 4.11.2020
[35] ebenda, Vertiefung ANN, Seite 52 unten, Ansichtsdatum 4.11.2020
[36] ebenda, Grundlagen 2, Seite 24, Ansichtsdatum 4.11.2020

Quelle: Beispiel inspiriert durch Einführung in die Automatische Spracherkennung an der Universität München, Grundlagen 2, Seite 24 Sprachmodellierung, Ansichtsdatum: 4.11.2020

5 Praktische Umsetzung

5.1 Verwendete Bibliotheken und Programmiersprache

Dieses Praxisbeispiel wird mit Hilfe der Open-Source-Programmiersprache *Python* umgesetzt. Python ist eine der populärsten Programmiersprachen der Welt, da sie einfach zu erlernen und gleichzeitig flexibel einsetzbar ist. Besonders häufig wird sie im Zusammenhang mit der Verarbeitung großer Datenmengen oder zur Programmierung intelligenter Systeme verwendet.

Um die Realisierung der Implementation eines künstlichen neuronalen Netzes für diese Arbeit zu vereinfachen, werden einige Bibliotheken eingebunden. Diese dienen dabei als eine Art Werkzeug. Zur einfacheren Ansicht wird das Online-Code-Dokumentationstool *Colab* von Google eingesetzt.

Das folgende Praxisbeispiel ist unter dem genannten Link einsehbar:
https://tinyurl.com/y5bg9vgw

Zuerst werden die benötigten Bibliotheken in das Programm importiert

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

Danach wird der bekannte MNIST-Datensatz heruntergeladen. Dabei handelt es sich um eine Beispieldatenbank, welche häufig für Experimente mit neuronalen Netzen verwendet wird.

```
(trainingsdaten, trainingslabels), (testdaten, testlabels) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

Die nächsten vier Zeilen konstruieren das eigentliche Netz. Das Netz wird als virtuelles Objekt mit dem Namen „*model*“ gespeichert. Dieses Netzwerk wird aus drei verschiedenen Schichten bestehen. Die erste Schicht ist die Input-Schicht (*vgl. „Flatten“*). Sie hat die Aufgabe, die zweidimensionalen Bilder bzw. Datenmatrizen in einen Zeilenvektor zu konvertieren. Die zweite Schicht (vgl. Z. 3) ist das Hidden-Layer mit 128 Neuronen. Als Aktivierungsfunktion für sie wird die „*ReLu*“-Funktion verwendet. Die letzte Schicht (*die Output-Schicht*) besteht aus zehn Ausgabeneuronen, die zu den Nummern 0 bis 9 gehören. Danach muss das Netz (*model*) nur noch kompiliert werden. Dabei wird die Fehlerfunktion (*vgl. „loss“*) noch definiert.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Nun muss das erstellte Netz nur noch trainiert werden, dabei gibt „*epochs*“ die Menge der Trainingsiterationen mit dem gegebenen Datensatz an.

```
model.fit(trainingsdaten,trainingslabels, epochs=3)
```

```
Epoch 1/3
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1204 - accuracy: 0.9649
Epoch 2/3
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0799 - accuracy: 0.9757
Epoch 3/3
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0583 - accuracy: 0.9822
<tensorflow.python.keras.callbacks.History at 0x7f6d54227b70>
```

Um nun das Netz auf den Testdatensatz anzuwenden muss zuerst ein Wahrscheinlichkeits-Model erstellt werden, welches jedes Bild im Testdatensatz bewertet. Die Zahl in den eckigen Klammern ist dabei der Index des Bildes, dessen Bewertung ausgegeben werden soll.

```
wahrscheinlichkeits_model = tf.keras.Sequential([model,tf.keras.layers.Softmax()])
```

```
ausgabe = wahrscheinlichkeits_model.predict(testdaten)
```

```
np.argmax(ausgabe[0])
```

```
7
```

Dank der äußerst umfangreichen Bibliothek *Tensorflow* ist es nichtmehr nötig, alle Teile eines künstlichen neuronalen Netzes selbst zu programmieren, da dies durch die Bibliothek bereits realisiert wird.

Das gezeigte Praxisbeispiel beruht auf dem Anfängerbeispiel für die Tensorflow-Bibliothek von Google: https://www.tensorflow.org/tutorials/keras/classification

6 Schluss

Am Ende dieser Arbeit lässt sich folgendes zusammenfassen: Es wurde versucht, einen groben Einblick in die Welt des maschinellen Lernens, sowohl von theoretischer, als auch von praktischer Sicht zu geben.

Dabei wurde mit der Definition eines künstlichen neuronalen Netzes begonnen. Demnach handelt es sich bei solch einem Netz um eine mathematische Funktion, welche die Aufgabe beinhaltet, intelligentes Verhalten zur Mustererkennung im Computer nachzustellen. Zudem sind sie von den natürlichen neuronalen Netzen aus der Natur abgeleitet.

Aufgebaut sind diese wiederum aus unterschiedlichen Schichten, den sog. „Layern“, welche aus Neuronen bestehen, die untereinander verbunden sind. Jedes dieser Verbindungen ist mit einem Gewicht belegt, welches die Wichtigkeit dessen bewertet und die Intelligenz des Systems ausmacht. Es wurde geklärt, dass bei diesen Netzen zwischen den *feed forward* und den *recursiven* unterschieden wird. Jedoch konnte nur auf ersterer der Beiden aufgrund der begrenzten Seitenzahl eingegangen werden.

Ferner wurde auch die Frage nach dem gängigsten Lernverfahren für ein solches System beantwortet. Dabei wurde festgestellt, dass es zwei gängige Herangehensweisen an das sog. Trainieren (*bzw. Lernen*) gibt: Das *Überwachte*, auf das mit der Fehlerrückführung stärker eingegangen wurde und das *Unüberwachte*, für welches das kurze Beispiel der evolutionären Netzwerke behandelt wurde. Das Ergebnis war, dass für das überwachte Lernen dem Netzwerk ein fester Datensatz, bestehend aus Eingabedaten und korrespondierenden Ausgabedaten, vorgegeben ist, während beim unüberwachten Lernen nur ähnliche Ausgaben zu den gegebenen Eingabedaten generiert werden müssen. Mit dem Gradientenabstiegsverfahren wurde wiederum die Funktionsweise der Fehlerrückführung des überwachten Lernens auf methodischer Weise geklärt.

Als Antwort für ein mögliches Einsatzgebiet dieser Technologie wurde die automatische Spracherkennung angeführt und mithilfe des bereits Erlernten erklärt.

Das letzte Kapitel behandelte das gewünschte praktische Beispiel, in welchem die automatisierte Erkennung von handgeschriebenen Zahlen realisiert wurde. Als Grundlage und zur Vereinfachung der Programmierung eines Netzwerkes, diente an dieser Stelle die bekannte Python-Bibliothek *Tensorflow*, mit deren Hilfe nur wenige Code-Zeilen benötigt werden.

Die Vorliegenden Ergebnisse haben einen wertvollen Einblick in die gängigsten Methoden des maschinellen Lernens und deren Einsatzgebiete gegeben. Es konnte jedoch oft nicht intensiv ins Detail gegangen werden, da, wie schon am Anfang der Arbeit bemerkt, einerseits die Zeit bzw. der Platz für größere Ausführungen fehlte und andererseits dies den Horizont des mathematischen Verständnisses eines Schülers der 12. Jahrgangsstufe überstiegen hätte. Jedoch ist die Wahrscheinlichkeit hoch, dass dieses Thema zu einem späteren Zeitpunkt noch einmal wieder aufgegriffen werden wird.

Probleme gab es zu Anfang auf Grund der Coronakrise des Jahres 2020, welche das Beschaffen physischer Fachliteratur oder Antworten durch Fachpersonal deutlich erschwerte.

Literaturverzeichnis

Kruse, P. D., & Dockhorn, A. (kein Datum). *Neuronale Netze.* Otto von Geuricke Universität Magdeburg.

Ploner, P. W., & Klaukien, M. (2009). *Referatszusammenfassung des Forschungsseminars Neuronale Netze.* Universität Innsbruck.

Reiche, M. (2019). *Backpropagation Lab.* Tübingen.

Rudolf Kruse, C. B. (2015). *Computational Intelligence.* Wiesbaden: Springer Vieweg.

Schiel, F. (2014/15). *Einführung in die automatische Spracherkennung.* Ludwig-Maximilians-Universität München.

Strecker, S. (1997). *Künstliche Neuronale Netze - Aufbau und Funktionsweise.* Universität Mainz.

Thamm, S. (2019). *Einführung in Neuronale Netze.* Universität Köln.

Wittpahl, V. (2019). *Künstliche Intelligenz Technologie | Anwendung | Gesellschaft.* Berlin, Deutschland: SpringerVieweg.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe. Alle Stellen der Arbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Das Gleiche gilt für die beigegebenen Zeichnungen, Kartenskizzen und Darstellungen.

Moos, 10.11.2020 Unterschrift