UNIVERSITÄT SIEGEN

Clientseitiges Deep Learning durch Klassifizierung von deutschsprachigen Clickbaits

Ugur Tigu

Master-Thesis

zur Erlangung des akademischen Grades Master of Science (M.Sc.)

Studiengang Wirtschaftsinformatik

Fakultät IV - Institut für Wissensbasierte Systeme und Wissensmanagement

Universität Siegen

8. Dezember 2020

Betreuer

Prof. Dr.-Ing. Madjid Fathi, Universität Siegen

Johannes Zenkert, Universität Siegen

Tigu, Ugur:
Clientseitiges Deep Learning durch Klassifizierung von deutschsprachigen Clickbaits / Ugur Tigu. –
Master-Thesis, Aachen: Universität Siegen, 2020. 12 Seiten.

Tigu, Ugur:
Client-side deep learning through classification of German clickbaits / Ugur Tigu. –
Master Thesis, Aachen: University of Siegen, 2020. 12 pages.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Universität Siegen öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Aachen, 8. Dezember 2020

Ugur Tigu

Abstract

Clientseitiges Deep Learning durch Klassifizierung von deutschsprachigen Clickbaits

Ein im Internet weit verbreitetes Phänomen sind *Clickbaits-Nachrichten* (auf deutsch „Klickköder“). Ziel dieser Arbeit ist die Entwicklung eines Deep Learning Verfahrens, welches deutsche Clickbait Nachrichten automatisch erkennen soll. Die Arbeit stellt einen Datensatz vor, welches aus zwei Klassen von Nachrichten Überschriften besteht und zum trainieren eines Deep Learning Ansatzes verwendet wird. Dieser Datensatz wird durch Web Scraping erstellt und gelabelt. Das Ergebnis dieser Arbeit ist ein Modell für die Textklassifizierung, entwickelt in TensorFlow.js. Dieses Modell wird vollständig clientseitig in den Browser eingebettet und benötigt somit keinen Server.

Client-side deep learning through classification of German clickbaits

A widespread phenomenon on the Internet are clickbaits. The aim of this thesis is the development of a deep learning model which should automatically recognize German clickbait titles. The thesis presents a data set, which consists of two classes of news headlines and is used to train a deep learning approach. This data set is created using web scraping and hand-labeled. The result of this work is a model for text classification, developed in TensorFlow.js. This model is completely embedded in the browser on the client side and therefore does not require a server.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Kapitel 2

Deep Learning

2.1 Einleitung

Künstliche neuronale Netze stellen eine Klasse von Modellen des maschinellen Lernens dar, die vom Zentralnervensystem von Säugetieren inspiriert sind. Jedes Netz besteht aus mehreren miteinander verbundenen „Neuronen“, die in „Schichten“ organisiert sind. Neuronen in einer Schicht leiten Nachrichten an Neuronen in der nächsten Schicht weiter (sie „feuern“ im Jargon). Erste Studien wurden in den frühen 50er Jahren mit der Einführung des „Perzeptrons“ [1] begonnen, eines zweischichtigen Netzwerks, das für einfache Operationen verwendet wird, und in den späten 60er Jahren mit der Einführung des „Back-Propagation-Algorithmus“ (effizientes mehrschichtiges Netzwerktraining) (gemäß [2], [3]) weiter ausgebaut. Einige Studien argumentieren, dass diese Techniken Wurzeln haben, die weiter zurückreichen als normalerweise zitiert [4].

Neuronale Netze waren bis in die 80er Jahre ein Thema intensiver akademischer Studien. Zu diesem Zeitpunkt wurden andere, einfachere Ansätze relevanter. Ab Mitte der 2000er Jahre ist das Interesse jedoch wieder gestiegen, hauptsächlich aufgrund von drei Faktoren: einem von G. Hinton [3], [5] vorgeschlagenen bahnbrechenden Algorithmus für schnelles Lernen, die Einführung von GPUs um 2011 (für massive numerische Berechnungen) und die Verfügbarkeit großer Datenmengen.

Diese Verbesserungen eröffneten den Weg für modernes „Deep Learning“, eine Klasse neuronaler Netze, die durch eine erhebliche Anzahl von Neuronenschichten gekennzeichnet ist, die in der Lage sind, auf der Grundlage progressiver Abstraktionsebenen komplexe Modelle zu erlernen. Sie werden als „tief“ bezeichnet, als es

vor einigen Jahren damit begannen wurde, 3-5 Schichten zu verwenden. Jetzt sind Netzwerke mit mehr als 200 Schichten vorstellbar.

Das Lernen durch progressive Abstraktion ähnelt Visionsmodellen, die sich über Millionen von Jahren im menschlichen Gehirn entwickelt haben. In der Tat ist das menschliche visuelle System in verschiedene Schichten unterteilt. Erstens sind unsere Augen mit einem Bereich des Gehirns verbunden, der als visueller Kortex (V1) bezeichnet wird und sich im unteren hinteren Teil unseres Gehirns befindet. Dieser Bereich ist vielen Säugetieren gemeinsam und hat die Aufgabe, grundlegende Eigenschaften wie kleine Änderungen der visuellen Ausrichtung, der räumlichen Frequenzen und der Farben zu unterscheiden.

Es wird geschätzt, dass V1 aus etwa 140 Millionen Neuronen besteht, zwischen denen zig Milliarden Verbindungen bestehen. V1 wird dann mit anderen Bereichen (V2, V3, V4, V5 und V6) verbunden, wobei die Bildverarbeitung zunehmend komplexer wird und komplexere Konzepte wie Formen, Gesichter, Tiere und vieles mehr erkannt werden. Es wird geschätzt, dass es 16 Milliarden menschliche kortikale Neuronen gibt und etwa 10-25% des menschlichen Kortexes dem Sehen gewidmet sind [6]. Deep Learning hat sich von dieser schichtbasierten Organisation des menschlichen visuellen Systems inspirieren lassen: Höhere künstliche Neuronenschichten lernen grundlegende Eigenschaften von Objekten, während tiefere Schichten komplexere Konzepte dieser Objekte lernen (siehe Abbildung 2.1).

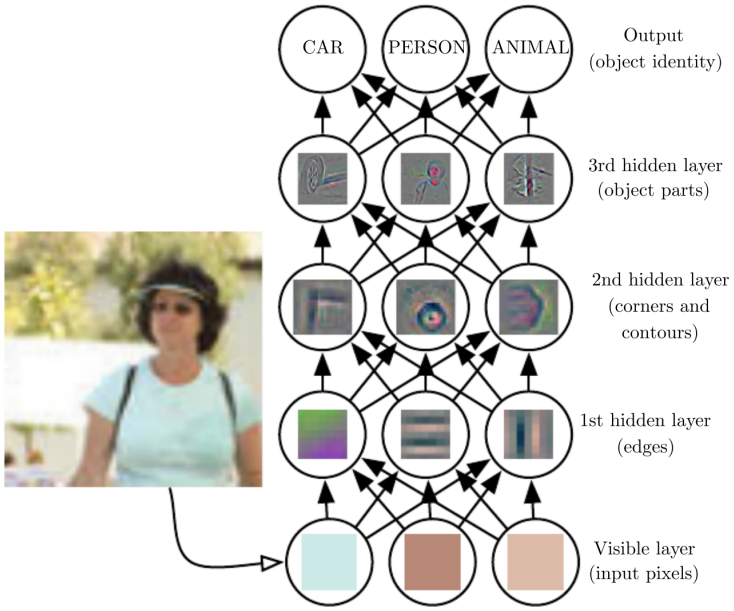

Abbildung 2.1: Illustration eines Deep-Learning-Modells aus [7] und [8]: Ein Computer kann ohne weiteres das Bild in dieser Abbildung nicht erfassen, da es keine sensorischen Rohdaten verstehen kann. Das Bild in dieser Abbildung ist nur eine Sammlung von Pixelwerten. Die Funktionszuordnung von einem Satz von Pixeln zu einer Objektidentität ist sehr kompliziert. Deep Learning löst diese Schwierigkeit, indem das gewünschte komplizierte „Mapping“ in eine Reihe verschachtelter einfacher Mappings aufgeteilt wird, die jeweils durch eine andere Ebene des Modells beschrieben werden. Die Eingabe wird auf der „sichtbaren Ebene“ (visible layer) dargestellt. Diese Schicht wird so genannt, weil sie die Variablen enthält, die wir beobachten können. Dann extrahieren eine Reihe „versteckter Ebenen“ (hidden layer) zunehmend abstrakte Merkmale aus dem Bild. Diese Ebenen werden als „versteckt“ bezeichnet, da ihre Werte nicht in den Daten angegeben sind. Stattdessen muss das Modell bestimmen, welche Konzepte zur Erklärung der Beziehungen in den beobachteten Daten nützlich sind. Angesichts der Pixel kann die erste Schicht Kanten nur leicht identifizieren, indem die Helligkeit benachbarter Pixel vergleicht. Angesichts der Beschreibung der Kanten durch die erste verborgene Ebene kann die zweite verborgene Ebene nach Ecken und erweiterten Konturen suchen. Angesichts der Beschreibung des Bildes durch die zweite verborgene Ebene in Bezug auf Ecken und Konturen kann die dritte verborgene Ebene ganze Teile bestimmter Objekte erkennen, indem bestimmte Konturen und Ecken gefunden werden. Schließlich kann diese Beschreibung des Bildes in Bezug auf die darin enthaltenen Objektteile verwendet werden, um die im Bild vorhandenen Objekte zu erkennen.

2.2 Das Perzeptron

Das Perzeptron kann ins deutsche mit dem Begriff der „Wahrnehmung“ übersetzt werden. Das Perzeptron ist ein einfacher Algorithmus mit einem Eingabevektor x mit m Werten $(x_2, ..., x_m)$. Es wird oft wird als „Eingabe-Features“ oder einfach als „Features“ bezeichnet und zurückgegeben wird entweder eine 1 „Ja“ oder eine 0 „Nein“ (siehe Formel 2.1).

In Formel 2.1 ist w ein Vektor welches das Gewicht darstellt, und wx das Punktprodukt aus $\sum_{j=1}^{m} w_j x_j$, b ist der Bias. Aus $wx + b$ ist die Grenzhyperebene definiert, die die Position gemäß den w und b zugewiesenen Werten ändert.

$$fx = \begin{cases} 1 & wx + b > 0 \\ 0 & ansonsten \end{cases} \tag{2.1}$$

Mit anderen Worten, ist dies ein sehr einfacher, aber effektiver Algorithmus. Beispielsweise kann das Perzeptron bei drei Eingabemerkmalen (Rot, Grün und Blau) unterscheiden, ob die Farbe weiß ist oder nicht. Es soll beachtet werden, dass das Perzeptron keine „Vielleicht"-Antwort ausdrücken kann. Es kann mit „Ja" (1) oder „Nein" (0) antworten. Das Perzeptron-Modell kann also benutzt werden, indem durch Anpassung von w und b, das Modell „trainiert" wird.

2.3 Mehrschichtiges Perzeptron

In der Vergangenheit war „Perzeptron" der Name eines Modells mit einer einzigen linearen Schicht. Wenn es mehrere Schichten hat, wurde es daher als mehrschichtiges Perzeptron (Multi-layer perceptron / MLP) bezeichnet. Die Eingabe- und Ausgabeebene ist von außen sichtbar, während alle anderen Ebenen in der Mitte ausgeblendet sind - daher der Name ausgeblendete Ebenen (hidden layers). In diesem Zusammenhang ist eine einzelne Schicht einfach eine lineare Funktion, und der MLP wird daher erhalten, indem mehrere einzelne Schichten nacheinander gestapelt werden (siehe Abbildung 2.2).

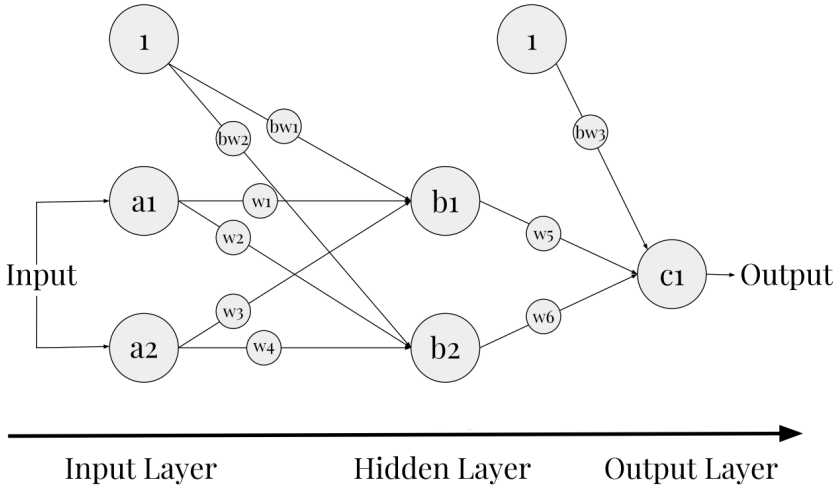

Abbildung 2.2: Ein Beispiel für ein mehrschichtiges Perzeptron in Anlehnung an [9]: Jeder Knoten in der ersten verborgenen Schicht empfängt eine Eingabe und „feuert“ eine 0 oder 1 gemäß den Werten der zugehörigen linearen Funktion. Dann wird die Ausgabe der ersten verborgenen Schicht an die zweite Schicht übergeben, wo eine andere lineare Funktion angewendet wird, deren Ergebnisse an die endgültige Ausgabeschicht übergeben werden. Die letzte Schicht besteht nur aus einem einzelnen Neuron. Es ist interessant festzustellen, dass diese geschichtete Organisation vage der Organisation des menschlichen Sichtsystems ähnelt, wie zuvor besprochen.

Was sind die besten Entscheidungen für das Gewicht w und den Bias b? Um diese Frage zu beantworten, wird nur ein einzelnes Neuron (ein einzelner Knoten) betrachtet.

Im Idealfall werden eine Reihe von Trainingsbeispielen bereitgestellt und der Computer muss das Gewicht w und den Bias b so einstellen, dass die in der Ausgabe erzeugten Fehler minimiert werden.

Um dies etwas konkreter zu machen, wird angenommen, dass es eine Reihe von Katzenbildern vorhanden sind und eine weitere separate Reihe von Bildern, die keine Katzen enthalten. Angenommen, jedes Neuron empfängt Eingaben vom Wert eines einzelnen Pixels in den Bildern. Während der Computer diese Bilder verarbeitet, möchten wir, dass unser Neuron seine Gewichte und seine Vorspannung so anpasst, dass immer weniger Bilder falsch erkannt werden. Dieser Ansatz scheint sehr intuitiv zu sein, erfordert jedoch eine kleine Änderung der Gewichte (oder des Bias), um nur eine kleine Änderung der Ausgänge zu bewirken. Wenn wir einen großen Leistungssprung haben, können wir nicht progressiv lernen. Es wird gewünscht, wie ein „Kind“ zu lernen, nach und nach. Das Perzeptron zeigt jedoch

dieses „Stück für Stück“-Verhalten nicht. Ein Perzeptron gibt entweder eine 0 oder eine 1 zurück und das ist ein großer Sprung, der beim Lernen nicht hilft.

2.4 Sigmoid-Neuron

Das Verhalten des Perzeptron ist sehr „uneben“, sodass ein „glatteres“ nötig ist. Wir brauchen eine Funktion, die sich ohne Diskontinuität schrittweise von 0 auf 1 ändert. Mathematisch bedeutet dies, dass wir eine stetige Funktion benötigen, mit der wir die Ableitung berechnen können.

Dieses Problem kann überwunden werden, indem einen neuer Typ eines künstlichen Neurons eingeführt wird, der als Sigmoid-Neuron. Sigmoidneuronen ähneln Perzeptronen, sind jedoch so modifiziert, dass kleine Änderungen ihres Gewichts und ihres Bias nur eine geringe Änderung ihrer Leistung bewirken. Dies ist die entscheidende Tatsache, die es einem Netzwerk von Sigmoidneuronen ermöglicht, zu lernen [10, S. 8].

Genau wie ein Perzeptron hat das Sigmoid-Neuron die Eingaben $x_1, x_2, ...$, aber anstatt nur 0 oder 1 zu sein, können diese Eingänge auch beliebige Werte zwischen 0 und 1 annehmen. Also zum Beispiel 0,123 welches eine gültige Eingabe für ein Sigmoid-Neuron ist. Ebenso wie ein Perzeptron hat das Sigmoid-Neuron Gewichte für jede Eingabe, $w_1, w_2, ...$ und einen Bias, b. Die Ausgabe ist jedoch nicht 0 oder 1, stattdessen ist es σ, $(wx + b)$, wobei σ als Sigmoidfunktion bezeichnet wird und durch Formel 2.2 definiert ist.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

2.5 Aktivierungsfunktionen

Ohne eine Aktivierungsfunktion (auch als Nichtlinearität bezeichnet) würde die dichte Schicht (dense layer) nuraus zwei linearen Operationen bestehen - einem Punktprodukt und einer Addition: $Ausgabe = Punkt(w, Eingabe)+b$. Die Schicht konnte also nur lineare Transformationen (affine Transformationen) der Eingabedaten lernen. Um Zugang zu einem viel umfangreicheren Hypothesenraum zu erhalten, wird eine Nichtlinearitäts- oder Aktivierungsfunktion benötigt [11, S. 72]. Es

gibt weitaus mehr Aktivierungsfunktionen, als die in die in diesem Abschnitt beschriebenen. Es sollen hier nur die gängigsten 3 vorgestellt werden.

2.5.1 Sigmoid

Die Sigmoidfunktion wurde bereits mit der Formel 2.2 definiert und in der Abbildung 2.3 dargestellt. Sie hat kleine Ausgangsänderungen im Bereich (0, 1), wenn der Eingang im Bereich $(-\infty, \infty)$ variiert. Mathematisch ist die Funktion stetig. Ein Neuron kann das Sigmoid zur Berechnung der nichtlinearen Funktion $\sigma(z = wx + b)$ verwenden. Wenn $z = wx + b$ sehr groß und positiv ist, dann wird $e^z \to 0$ also $\sigma(z) \to 1$, während wenn $z = wx + b$ sehr groß und negativ ist dann wird $e^{-z} \to 0$ also $\sigma(z) \to 0$. Mit anderen Worten, ein Neuron mit Sigmoidaktivierung hat ein ähnliches Verhalten wie das Perzeptron, aber die Änderungen sind allmählich und Ausgabewerte wie 0,54321 oder 0,12345 sind vollkommen legitim. In diesem Sinne kann ein Sigmoid-Neuron „vielleicht“ antworten [12, S. 10].

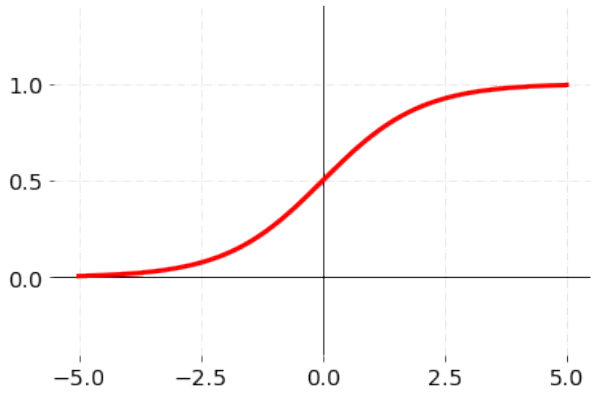

Abbildung 2.3: Darstellung der Sigmoid-Aktivierungsfunktion (eigene Darstellung)

2.5.2 Tanh

Die Tanh-Aktivierungsfunktion wird mit der Formel 2.3 definiert. Sie hat ihre Ausgangsänderungen im Bereich (-1, 1). Sie hat eine Struktur, die der Sigmoid-Funktion sehr ähnlich ist. Der Vorteil gegenüber der Sigmoidfunktion besteht darin, dass ihre

Ableitung steiler ist, was bedeutet, dass sie mehr Wert erhalten kann (vergleiche Abbildung 2.4).

$$tanh(z) = \frac{e^z - e^{-z}}{e^z - e^{-z}} \tag{2.3}$$

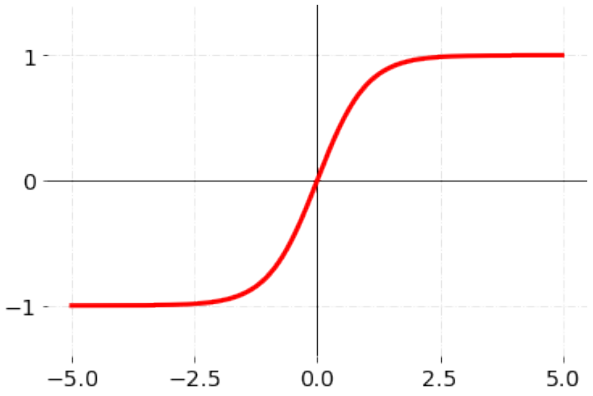

Abbildung 2.4: Darstellung der Tanh-Aktivierungsfunktion (eigene Darstellung)

2.5.3 ReLu

Vor kurzem wurde eine sehr einfache Funktion namens ReLU (REctified Linear Unit) sehr beliebt, da sie dazu beiträgt, einige bei Sigmoiden beobachtete Optimierungsprobleme zu lösen [12, S. 11]. Eine ReLU wird relativ einfach in und wird in der Formel 2.4 definiert. Wie in Abbildung 2.5 zu sehen, ist die Funktion für negative Werte Null und wächst für positive Werte linear. Die ReLU ist auch sehr einfach zu implementieren (im Allgemeinen reichen drei Anweisungen aus), während das Sigmoid einige Größenordnungen mehr benötigt.

$$f(x) = \begin{cases} 0 & \textit{für } x < 0 \\ x & \textit{für } x \geq 0 \end{cases} \tag{2.4}$$

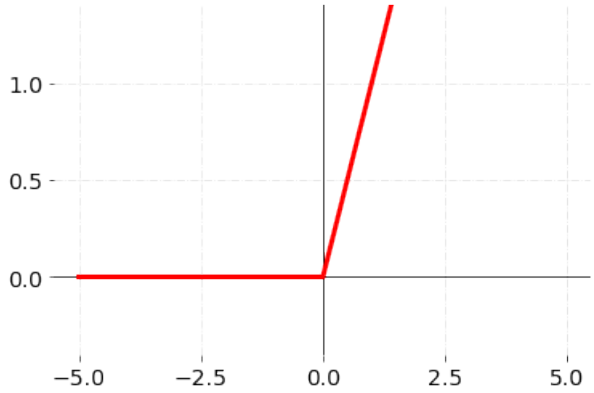

Abbildung 2.5: Darstellung der ReLu-Aktivierungsfunktion (eigene Darstellung)

2.6 Verlustfunktion und Kreuzentropie

Innerhalb eines neuronalen Netzwerks wandelt eine Verlustfunktion alle möglichen Fehler, in eine Zahl um, die den Gesamtfehler des Netzwerks darstellt. Im Wesentlichen ist es ein Maß dafür, wie falsch ein Netzwerk ist. Auf einer technischeren Ebene werden ein Ereignis oder Werte einer oder mehrerer Variablen einer reellen Zahl zugeordnet. Diese reelle Zahl stellt den „Verlust“ oder die „Kosten“ dar, die mit dem Ereignis oder den Werten verbunden sind [9, S. 41].

2.6.1 Mittlere quadratische Abweichung

Wie der Name sagt, wird mit der mittleren quadratischen Abweichung (MSE), der Verlust berechnet, indem der Mittelwert der quadratischen Differenzen zwischen tatsächlichen Ziel- und vorhergesagten Werten genommen wird. Sie wird in Formel 2.5 definiert.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(d-y)^2 \tag{2.5}$$

Diese Funktion ist der Durchschnitt aller Fehler, die in jeder Vorhersage gemacht wurden. Wenn eine Vorhersage weit vom wahren Wert entfernt ist, wird dieser Abstand durch die „Quadrierungsoperation“ deutlicher. Außerdem kann das Quadrat den Fehler addieren, unabhängig davon, ob ein bestimmter Wert positiv oder negativ ist [12, S. 17].

Grundsätzlich kann diese Verlustfunktion verwendet werden, wenn die Ausgabe eine reelle Zahl ist. Die MSE-Verlustfunktion wird meistens für Regressionsaufgaben verwendet.

2.6.2 Binäre Kreuzentropie

2.6.3 Kategoriale Kreuzentropie

Abkürzungsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Literatur

[1] F Rosenblatt, „THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN 1“, Techn. Ber. 6, S. 19–27.

[2] Paul J. Werbos, „Backpropagation Through Time: What It Does and How to Do It“, *Proceedings of the IEEE*, Jg. 78, Nr. 10, S. 1550–1560, 1990. DOI: 10.1109/5.58337.

[3] Geoffrey E Hinton und Simon Osindero, „A Fast Learning Algorithm for Deep Belief Nets Yee-Whye Teh“, Techn. Ber.

[4] Jürgen Schmidhuber, „Deep Learning in Neural Networks: An Overview“, Techn. Ber., 2014. arXiv: 1404.7828v4. Adresse: http://www.idsia.ch/%CB%9Cjuergen/DeepLearning8Oct2014.texCompleteBIBTEXfile.

[5] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams, „Learning representations by back-propagating errors“, *Nature*, Jg. 323, Nr. 6088, 1986. DOI: 10.1038/323533a0.

[6] Suzana Herculano-Houzel, *The human brain in numbers: A linearly scaled-up primate brain*, 2009. DOI: 10.3389/neuro.09.031.2009.

[7] Aaron Courville Ian Goodfellow, Yoshua Bengio, *Deep Learning*. 2016, S. 1–10.

[8] Faiz Bukhari Mohd Suah, „Preparation and characterization of a novel Co(II) optode based on polymer inclusion membrane“, *Analytical Chemistry Research*, Jg. 12, S. 40–46, 2017. DOI: 10.1016/j.ancr.2017.02.001.

[9] Michael Taylor, *The Math of Neural Networks*. 2017.

[10] Michael Nielsen, „Neural Networks and Deep Learning“, Techn. Ber. Adresse: http://neuralnetworksanddeeplearning.com.

[11] Francois Chollet, *Deep learning with Python*. 2017. DOI: 10.23919/ICIF.2018.8455530.

[12] Antonio Guili; Amita Kapoor; Sujit Pal, *Deep Learning with TensorFlow 2 and Keras: Regression, ConvNets, GANs, RNNs, NLP, and More with TensorFlow 2 and the Keras API*. 2019.