Eine praxisorientierte Einführung in **künstliche neuronale Netze, Gesichtserkennung** und Co.

SCIENCE ON STAGE DEUTSCHLAND

THE EUROPEAN NETWORK FOR SCIENCE TEACHERS

# Machine Learning in der Schule

**Herausgeber**
Science on Stage Deutschland e.V.
Am Borsigturm 15
13507 Berlin

**Projektmanagement und Redaktion**
Karoline Kirschner, Projektmanagerin
Science on Stage Deutschland e.V.
Stefanie Schlunk, Geschäftsführerin
Science on Stage Deutschland e.V.

**Autor**
Dr. Daniel Janssen, Gymnasium Dionysianum Rheine

**Text- und Bildnachweise**
Der Autor hat die Bildrechte für die Verwendung in dieser Publikation nach bestem Wissen geprüft und ist für den Inhalt seiner Texte verantwortlich.

**In Kooperation mit**

VECTOR STIFTUNG

**Hauptförderer von Science on Stage Deutschland e.V.**

think ING.
Die Initiative für Ingenieurnachwuchs

**Gestaltung**
WEBERSUPIRAN.berlin

**Illustration**
Rupert Tacke, Tricom Kommunikation und Verlag GmbH

**Kontakt**
www.science-on-stage.de
info@science-on-stage.de

ISBN 978-3-942524-71-1

Diese Publikation steht unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz:
https://creativecommons.org/licenses/by-sa/4.0/

CC BY SA

3. korrigierte Auflage, April 2022
© Science on Stage Deutschland e.V.

# Inhalt





# Über den Autor

**Dr. Daniel Janssen** studierte Informatik und Sport für die Lehrämter der Sekundarstufen I und II an der Westfälischen Wilhelms-Universität Münster. Nach dem Studium war er vorerst als wissenschaftlicher Mitarbeiter im Arbeitsbereich Trainingswissenschaft der Westfälischen Wilhelms-Universität Münster und in der Abteilung für Trainings- und Bewegungswissenschaft der Johannes Gutenberg-Universität Mainz tätig. Im Anschluss absolvierte er sein Referendariat am Studienseminar Osnabrück und war zuerst Studienrat am Kopernikus-Gymnasium Ratingen und später am Gymnasium Martinum Emsdetten. Seit 2015 ist er Informatik- und Sportlehrer am Gymnasium Dionysianum in Rheine. Neben seiner Lehrertätigkeit promovierte er 2016 am Fachbereich Sportwissenschaft der Johannes Gutenberg-Universität Mainz. Zu Science on Stage kam Janssen als Teilnehmer des Nationalen Science on Stage Festivals 2018, auf dem er sein Unterrichtsprojekt „Machine Learning in der Schule“ präsentierte. Vor Ort qualifizierte er sich damit als eines von elf deutschen Projekten, die am Europäischen Science on Stage Festival 2019 in Portugal mit insgesamt 450 MINT-Lehrkräften aus über 30 Ländern teilnehmen durften.

# Vorwort

Liebe Leserinnen und Leser,

wie begeistert man Schülerinnen und Schüler für MINT-Themen? Auf gute Lehrkräfte kommt es an! Als größtes europäisches Netzwerk für MINT-Lehrkräfte bietet Science on Stage ihnen deshalb eine Bühne, um eigene innovative Unterrichtsideen vorzustellen, sich darüber auszutauschen und voneinander zu lernen. Pädagoginnen und Pädagogen in ihrer Aus- und Weiterbildung zu unterstützen, um so das Interesse bei jungen Menschen für Naturwissenschaften, IT und Ingenieurwissenschaften zu wecken und sie optimal zu fördern, ist auch das Anliegen der Vector Stiftung.

Daher kooperierten Science on Stage und die Vector Stiftung erstmalig beim Nationalen Science on Stage Festival 2018. Alle zwei Jahre kommen hier 100 ausgewählte Lehrkräfte aus dem gesamten Bundesgebiet zusammen, um auf einem Bildungsmarkt, in Kurzvorträgen und Workshops ihre Unterrichtsprojekte zu präsentieren und engagierte Kolleginnen und Kollegen kennenzulernen. Das Projekt *Machine Learning in der Schule* des Teilnehmers Dr. Daniel Janssen hat die Fachjury vor Ort überzeugt. Damit seine Idee weitere Verbreitung findet, stellen wir sie in der vorliegenden Broschüre vor.

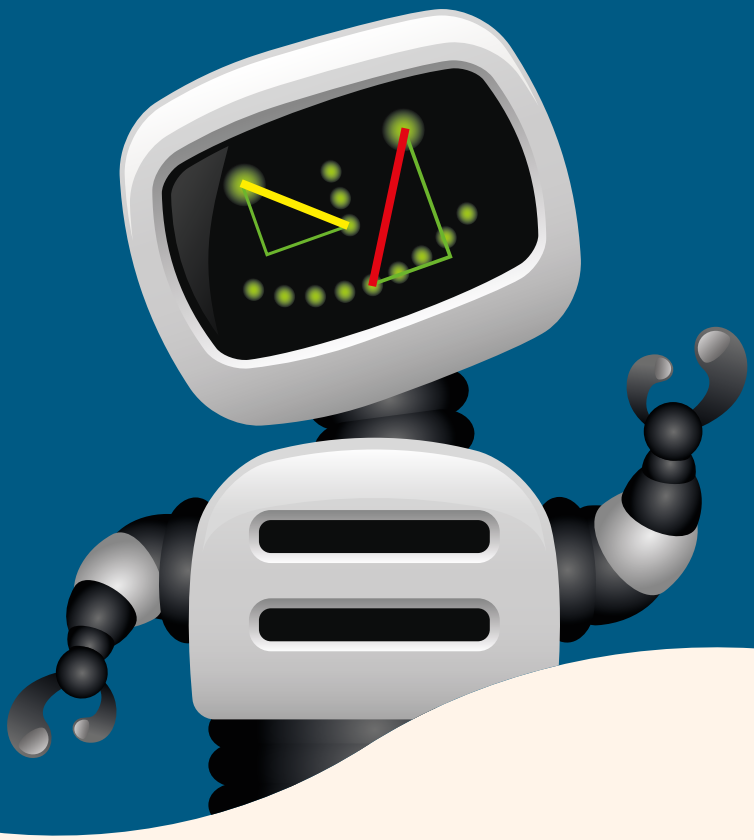
Mit diesem in der Praxis erprobten Unterrichtsmaterial möchten wir zeigen, wie Sie Ihren Schülerinnen und Schülern komplexe Inhalte des maschinellen Lernens verständlich näherbringen. Als Teilgebiet künstlicher Intelligenz hält damit ein hochaktuelles und gesellschaftsrelevantes Thema im Klassenzimmer Einzug.

Wir danken herzlich unserem Autor Dr. Daniel Janssen für die Arbeit und Zeit, die er neben seiner Tätigkeit als Lehrer in die Aufbereitung seines Unterrichtsprojektes investiert hat! Nun wünschen wir Ihnen und Ihren Schülerinnen und Schülern viel Freude bei der Umsetzung der Materialien. Bei Fragen und Anregungen können Sie sich gerne per E-Mail an die Geschäftsstelle von Science on Stage Deutschland wenden (info@science-on-stage.de).

**Stefanie Schlunk**
Geschäftsführerin Science on Stage Deutschland e.V.

**Edith Wolf**
Vorstand Vector Stiftung

# Einführung für Lehrkäfte



**UNTERRICHTSFÄCHER** Informatik
**DAUER** 6–8 Doppelstunden
**SCHLAGWÖRTER** maschinelles Lernen, neuronale Netze
**KLASSENSTUFE** 9–10 (Sekundarstufe I)
**ERFORDERLICHE VORKENNTNISSE** keine (Kenntnisse zu Schaltnetzen aus der technischen Informatik evtl. vorteilhaft)
**HARDWARE** Computer mit Windows-Betriebssystem
**SOFTWARE** MemBrain, GIMP, Texteditor, Browser

*Neural Networks, Deep Learning, Machine Learning* (zu Deutsch: Maschinelles Lernen) – das sind Schlagworte, die in den letzten Jahren ein gewaltiges Medienecho hervorriefen. Oftmals ist die Rede von künstlicher Intelligenz, die in der Lage ist, Erstaunliches zu leisten, wie bspw. die Spracherkennung in modernen Smartphones. Dabei sind die Begriffe künstliche Intelligenz und künstliche neuronale Netzwerke – z. B. Deep Learning – nicht gleichzusetzen, da letztere nur einen Teilbereich der Forschung im Bereich künstlicher Intelligenz darstellen. Dennoch dringt das Thema immer tiefer in die Gesellschaft ein, sodass eine Beschäftigung hiermit im Bereich Schule unumgänglich ist. Aufgrund der Komplexität des Themas wird in der vorgestellten Unterrichtsreihe an mehreren Stellen didaktisch reduziert, was jedoch dem Gesamtverständnis in keiner Weise schadet. Stellvertretend für einen Teil des maschinellen Lernens werden künstliche neuronale Netze vorgestellt, denn diese nehmen einen immer größer werdenden Anteil im Gesamtbereich künstlicher Intelligenz ein. Das Arbeitsmaterial ist auf die Sekundarstufe I ausgelegt und kann dort vor Beginn der Oberstufe in einigen Doppelstunden bearbeitet werden. Es sollen dabei nicht nur die technische Seite, sondern auch die gesellschaftlichen, politischen und ethischen Auswirkungen dieser Entwicklung am Rande beleuchtet werden. Zudem wird bewusst auf Programmierung verzichtet, sodass die Unterrichtsreihe das reine Modellieren in den Vordergrund stellt. Da u. a. mit Python und Tensorflow inzwischen genug High-Level-Programmiermöglichkeiten existieren, kann hier im Anschluss beliebig erweitert werden, um ggf. das Thema um aktuell viel diskutierte Deep Learning-Netze zu ergänzen.

Machine Learning ist vielseitig und ein Teilgebiet der Forschungsrichtung, die sich mit künstlicher Intelligenz befasst. Als besonders populäre Vertreter des maschinellen Lernens gelten künstliche neuronale Netze. Schülerinnen und Schüler lernen, wie künstliche neuronale Netze aufgebaut sind und wie diese selbstständig lernen können. Daher werden sie sich in dieser Reihe mit dem Aufbau von künstlichen Neuronen, dem Zusammenschalten zu neuronalen Netzen und dem Lernen in diesen Netzen beschäftigen. Sie arbeiten mit dem Modellierungswerkzeug MemBrain[A], klassifizieren mit ihren eigenen neuronalen Netzen Boolesche Funktionen und den Iris-Datensatz. Darüber hinaus entwerfen sie anhand des Yale Face-Datensatzes und der Bildbearbeitungssoftware GIMP ein eigenes Gesichtsmodell, mit dem sie neuronale Netze darauf trainieren, Gesichter zu erkennen und Erkennungsraten zu berechnen.

Das nachfolgende Material ist für die Bearbeitung durch Ihre Schülerinnen und Schüler gedacht. Ergänzt wird es zwischen den Aufgaben durch entsprechende Hintergrundinformationen. Zudem finden Sie online[B] weiterführendes Material zu den Aufgaben, auf das an den jeweiligen Stellen verwiesen wird. Die Lösungen können dort ebenfalls heruntergeladen werden. Die Reihe behandelt bis zum Arbeitsmaterial 6 zur Gesichtserkennung die Grundlagen des Themas. Das danach aufbereitete Material stellt eine optionale Weiterführung dar.

## Zusatzmaterialien

- → Anleitung Simulation eines Netzes in MemBrain
- → Anleitung Klassifikation mit einer Booleschen Funktion in MemBrain
- → Anleitung zur Gesichtserkennung mit MemBrain für Schülerinnen und Schüler
- → Anleitung zur Gesichtserkennung mit MemBrain für Lehrkräfte
- → Iris-Beispiel
- → Lösungen

A www.membrain-nn.de (19.07.2019)
B Alle Zusatzmaterialien können unter folgendem Link heruntergeladen werden: www.science-on-stage.de/machinelearning

# Arbeitsmaterial für Schülerinnen und Schüler

# Arbeitsmaterial 1

## in Partnerarbeit

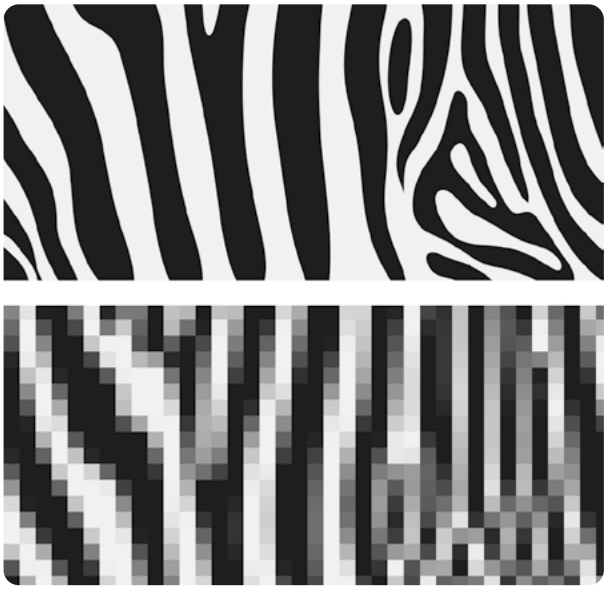
① Öffnet auf einem PC ein Zeichenprogramm (Paint o. ä.). Eine/r von euch zeichnet nun mithilfe der Maus ein Bild, das den Namen einer Lehrkraft an eurer Schule beschreibt. Der oder die andere Schüler/in versucht, den Namen zu erraten. Gibt es z. B. eine Frau Müller, so kann eine Mühle gezeichnet werden; oder gibt es einen Herrn Bäumer, so könnten mehrere grob skizzierte Bäume dabei helfen, den richtigen Namen zu erraten.

② Überlegt einmal unvoreingenommen und kleinschrittig, wie euer Gegenüber auf die Lösung kommen konnte. Welches Wissen und welche Fähigkeiten waren dazu notwendig? Diskutiert im Anschluss die Frage, warum und wodurch wir Menschen überhaupt in der Lage sind, eine solche Rateleistung zu vollbringen.

③ In der Abbildung oben seht ihr sechs Socken und sechs Schuhe, die Benutzer von Google Quickdraw[C] mit der Hand gemalt haben. Ein Computer hat bei allen zwölf Zeichnungen richtig erkannt, ob es Socken oder Schuhe waren. Überlegt zusammen anhand welcher Merkmale ihr erkennt, ob es sich um eine Socke oder einen Schuh handelt. Erarbeitet umgangssprachlich ganz grob einen Algorithmus, mit dem ein Computer erkennen könnte, um was es sich handelt – in der Form: „wenn … dann … und … oder wenn …“ usw.

## Mensch versus Computer

Der Computer kann im sogenannten Dualsystem blitzschnell Rechenaufgaben lösen. Dazu wandelt er Zahlen in „Strom an“ (1) und „Strom aus“ (0) um. Mithilfe von Transistoren und Schaltungen kann der Computer so die Aufgabe milliardenfach schneller lösen als der Mensch. Der Mensch hingegen ist ein Experte im Erkennen von Mustern. Er verfügt über ein hochkomplexes System, Zusammenhänge und Muster zu erfassen. Unsere Lernfähigkeit gepaart mit der enormen Erfahrung, die wir in unserem Leben sammeln, befähigt uns, in Sekundenbruchteilen schwierige Aufgaben zu lösen wie z. B. auf einem Bild zu erkennen, dass dort ein Zebra abgebildet ist.



Ein Zebramuster wie in der Abbildung kann ein Mensch vermutlich direkt als Ausschnitt aus einem Bild mit einem Zebra identifizieren. Das kann ein Computer nicht leisten. Für einen Computer ist diese Aufgabe auf den ersten Blick schier unlösbar, da er nur ein zweidimensionales Gitter an Pixeln „sieht“, die entweder an (schwarz) oder aus (weiß) sind. Zwar könnte man ihn darauf programmieren, eine bestimmte Reihenfolge aus unterschiedlichen schwarz-weißen Pixeln als Zebra zu erkennen, da aber jedes Zebra anders aussieht, würde das nicht gut oder gar nicht funktionieren. In der Abbildung erkennt man die einzelnen Pixel in der Vergrößerung sehr gut. Schnell leuchtet ein, dass Regeln der Form „Wenn das erste Pixel weiß ist, das nächste schwarz und das darunter wieder weiß ist, …, … dann ist es ein Zebra.“ nicht funktionieren werden, um ganz allgemein ein Zebra zu erkennen.

Könnte man dem Computer beibringen, solche Dinge auf dieselbe Art wie ein Mensch zu interpretieren, hätte man ein mächtiges Werkzeug, da der Computer diese vielleicht sogar weitaus schneller lösen würde. Tatsächlich gibt es bereits seit

C https://quickdraw.withgoogle.com (19.07.2019)

den 1940er Jahren Bestrebungen, Teile des menschlichen Gehirns künstlich nachzubilden, zuerst nur als mathematisches Modell, später mit dem Aufkommen des Computers auch als elektronische Software. Seit den 1980er Jahren erlebte diese Forschungsrichtung einen regelrechten Aufschwung, weil Computer immer leistungsfähiger wurden. Doch erst seit den 2010er Jahren geht die Forschung auf diesem Gebiet erfolgreich voran. Gründe dafür waren der Einstieg der großen Unternehmen wie Google, Facebook, Apple usw. in das Geschäft und die Verfügbarkeit von massenweise günstigen hochleistungsfähigen Grafikprozessoren, auf denen heutzutage riesige künstliche neuronale Netzwerke massiv parallel mit Daten gefüttert werden können. Dass gerade Grafikprozessoren so interessant sind, liegt in ihrer Struktur. Statt nur eines Kerns bzw. weniger Kerne (wie eine CPU eines Rechners) besitzen sie sehr viele programmierbare Einheiten und sind durch ihre parallele Verschaltung überaus schnell. Ob sie deswegen auch wirklich – im menschlichen Sinne – „intelligent" sind, steht auf einem anderen Blatt.

# Arbeitsmaterial 2

## in Partnerarbeit

Öffnet einen Browser und navigiert zu quickdraw.withgoogle.com. Auf dieser Seite gibt es ein kleines interaktives Programm, das versucht, zu erraten, was ihr am Bildschirm mit der Maus gemalt habt. Dazu bekommt ihr nacheinander sechs Aufgaben gestellt, wie z. B. „Zeichne eine Straßenlaterne in 19 Sekunden." In der zur Verfügung stehenden Zeit versucht das Programm so schnell wie möglich, das von euch Gezeichnete richtig zu erkennen. Spielt dieses je zweimal durch und probiert, möglichst viel Gezeichnetes vom Programm richtig erraten zu lassen. Wer die höchste Punktzahl erreicht, hat gewonnen.

### Überall maschinelles Lernen

Computer sind gut darin, mit exakten Daten zu arbeiten. Eine Rechenaufgabe wie $1.234 \cdot 4.321 = 5.332.114$ ist in Sekundenbruchteilen gelöst. Dafür würde selbst ein Kopfrechnen-Genie deutlich länger benötigen. Eine Zeichnung der korrekten Klasse zuzuordnen wie es Quickdraw kann („Das, was du gemalt hast, war ein Schuh!"), ist jedoch eine sehr viel kompliziertere und anspruchsvollere Aufgabe, die bisher von Menschen deutlich schneller und besser durchgeführt werden konnte. Doch der Computer wird immer besser im Umgang mit diesen nicht-exakten, vielleicht sogar verrauschten, ungenauen Informationen. Zu verdanken ist dies Forschenden, die seit vielen Jahren versuchen, vom Gehirn des Menschen zu lernen und den Computer zu befähigen, Strukturen zur Aufgabenlösung zu nutzen, die bisher nur Menschen vorbehalten waren. Google Quickdraw arbeitet z. B. mit sogenanntem maschinellen Lernen, genauer: mit künstlichen neuronalen Netzen, die schon ziemlich gut funktionieren. Quickdraw ist nur ein Beispiel dafür, wie maschinelles Lernen hilft, Informationen zu verarbeiten, die unscharf und ungenau sind. So ist es auch egal, ob man einen Schuh links, rechts, oben, unten, groß, klein, nach links oder nach rechts ausgerichtet malt. Meist wird er trotzdem erkannt. So viele Wenn-Dann-Regeln könnte man gar nicht programmieren, wollte man einen Schuh-Erkennungs-Algorithmus entwerfen. Das riesige künstliche neuronale Netz von Quickdraw kann es einfach so – und zudem auch gleichzeitig alle anderen Klassen von Zeichnungen meist richtig zuordnen (siehe quickdraw.withgoogle.com/data für einen Einblick in den riesigen Datensatz, mit dem das künstliche neuronale Netz trainiert wurde).

# Arbeitsmaterial 3

## in Partnerarbeit

Neben Quickdraw oder digitalen Assistenten wie Siri[D] und Cortana[E] begegnen uns im Alltag bereits sehr viele Anwendungen, in denen Maschinen befähigt werden, zu lernen, mit unscharfen, nicht-exakten Informationen umzugehen. Fertigt eine Liste mit zehn Punkten an, für die ihr vermutet (!), dass dort ähnlich mächtige Werkzeuge eingesetzt werden wie bei der Zeichnungserkennung von Quickdraw oder der Spracherkennung von digitalen Assistenten (zwei dieser zehn Punkte sind bereits ausgefüllt):

- Erkennen von Zeichnungen (Google Quickdraw)
- Spracherkennung (Siri, Cortana)
- …

## Das biologische Neuron

Forschende haben sich in der Vergangenheit bei der Entwicklung von Modellen künstlicher neuronaler Netze grob an Überlegungen orientiert, wie das menschliche Gehirn aufgebaut sein könnte und aus welchen Bestandteilen es besteht. Da bis heute nicht vollends geklärt ist, wie genau unser Gehirn eigentlich in der Lage ist, Informationen zu speichern und zu verarbeiten, hat man sich seit jeher mit sehr vereinfachten Modellen zufriedengegeben. Und tatsächlich haben diese Modelle bereits ausgereicht, um leistungsstarke künstliche neuronale Netze wie z. B. Google Quickdraw zu entwerfen. Die Modelle, die sich an einem sehr vereinfachten biologischen Verständnis des Gehirns orientieren, werden im Folgenden vorgestellt. Dabei wird nur Rücksicht auf das Notwendigste genommen, das zum Verständnis von künstlichen neuronalen Netzen erforderlich ist.

Das menschliche Gehirn hat ca. $10^{11}$ Nervenzellen (auch Neuronen genannt). Wie diese (grob) aufgebaut sind, wird nachfolgend beschrieben: Die Nervenzellen sind die Grundbausteine der Informationsverarbeitung im Gehirn. Die genauen Prozesse, die im Gehirn zwischen den Neuronen ablaufen, sind sehr komplex und zum Teil noch unerforscht. Schon vor längerer Zeit haben Forschende ein Modell entwickelt, mit dem sie versuchen zu erklären, wie Nervenzellen miteinander „kommunizieren“ bzw. wie die Informationsverarbeitung über sie abläuft. Der prinzipielle und hier vereinfachte Aufbau einer Nervenzelle wird dabei so beschrieben, dass diese aus dem Zellkörper, dem Zellkern, den Dendriten, dem Axon und den Synapsen besteht. Hierbei ist zu beachten, dass die Neuronen untereinander verbunden sind. Auch wenn es verschiedene Arten von Nervenzellen gibt, die sich z. T. in ihrem Aufbau unterscheiden und es neben aktivierenden Eingangssignalen auch hemmende Eingangssignale gibt, soll die hier stark vereinfachte Darstellung des Neurons für unsere Zwecke reichen:

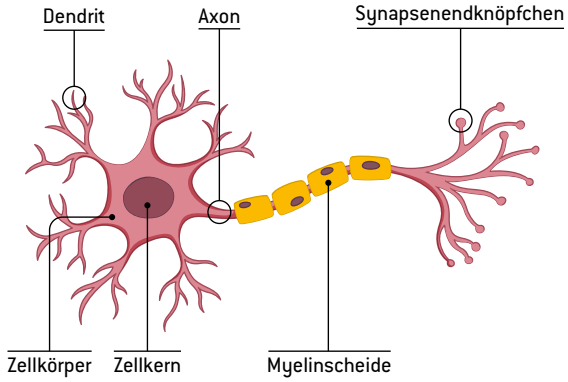



Abbildung 1

- Der **Zellkörper** enthält den Zellkern und u. a. die Mitochondrien, die für die weitere Betrachtung jedoch nicht relevant sind. Er steuert und regelt alle Vorgänge in einer Zelle.
  Der Zellkern hat dabei die Aufgabe, die Eingangssignale zu verarbeiten und ggf. ein Ausgangssignal zu erzeugen.
  Man spricht dann auch davon, dass die Zelle „feuert“.
- Die **Dendriten** sind meist stark verästelte Fortsätze der Zellen, über die das Neuron seine Eingangssignale empfängt.
- Über das **Axon** „feuert“ die Nervenzelle, d. h. hier werden die Ausgangssignale der Nervenzelle weitergeleitet.
- Die **Synapsenendknöpfchen** übernehmen dabei die Funktion der Kontaktstellen zu den Dendriten anderer Nervenzellen, die mit der Nervenzelle verbunden sind, oder zu Muskel- und Drüsenzellen.

D https://de.wikipedia.org/wiki/Siri_(Software) (23.09.2019)
E https://de.wikipedia.org/wiki/Cortana_(Software) (23.09.2019)

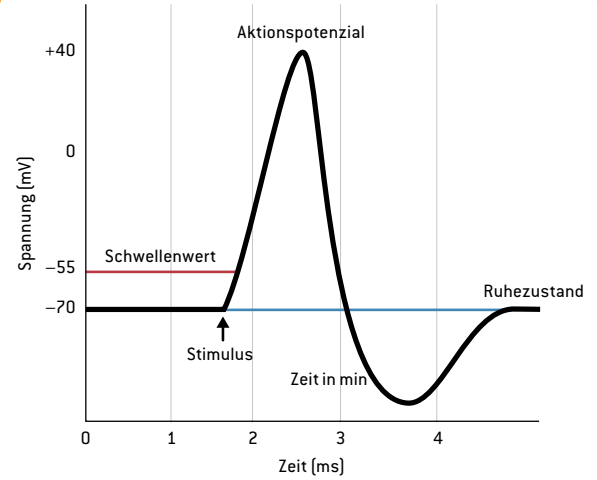



Abbildung 2

Man kann sich ein Neuron ganz vereinfacht wie einen Schalter vorstellen. Das Neuron „feuert" bzw. sendet einen Impuls über sein Axon, sobald genug Eingangssignale anliegen, die in der Summe einen Schwellenwert überschreiten. Das geschieht durch elektrochemische Prozesse. Das Axon leitet die Aktionspotenziale in Richtung Synapsenendknöpfchen weiter. Im Detail sei dazu auf Fachliteratur aus der Biologie verwiesen. Abbildung 2 visualisiert den Vorgang lediglich grob: Eingangssignale vorgeschalteter Neuronen sorgen für eine Spannungsänderung, indem ein in der Zelle vorhandener Schwellenwert überschritten wird, die Zelle ein Aktionspotenzial ausbildet und damit „feuert", da die Spannungsänderung über das Axon abgeleitet wird. Ein solches Aktionspotenzial dauert bei Nervenzellen ca. 1–2 Millisekunden. Es schließt sich danach eine kurze Ruhephase an, bevor der ursprüngliche Spannungszustand wiederhergestellt ist und die Zelle erneut „feuern" kann.

# Arbeitsmaterial 4

## Für die Schnellen: Zusatzaufgabe

Überlegt, welche Bestandteile ein künstliches Neuron benötigen könnte und skizziert ein stark vereinfachtes Modell mit möglichst wenigen Bestandteilen. Bedenkt dabei, dass später evtl. viele Neuronen zu einem Netz(werk) zusammengefügt werden, und dass dieses Netz dann einen bis mehrere Eingänge sowie Ausgänge besitzen kann. Überlegt weiterhin, wie ihr folgende Fragen lösen könnt:

- Wie könnte ein Dendrit modelliert werden?
- Eine Synapse bestimmt im Prinzip in diesem Modell, wie stark eine Verbindung zwischen zwei Neuronen ist. Wie könnte man unterschiedliche Stärken an einer Verbindung (bzw. an einem Dendriten) modellieren?
- Wie könnte …
  - … der Zellkörper mit dem Schwellenwert modelliert werden?
  - … man modellieren, dass der Schwellenwert nur übertroffen wird, wenn es genug Eingangssignale von feuernden Vorgängerneuronen gibt?
  - … ein Axon modelliert werden?
  - … man das Feuern des Neurons modellieren?

### Künstliche Neuronen

Aus dem biologischen Modelloriginal (s. Abbildung 1) wird ein vereinfachtes informatisches Modell, ebenso bestehend aus Synapsen, Dendriten, dem Zellkörper und einem Axon.

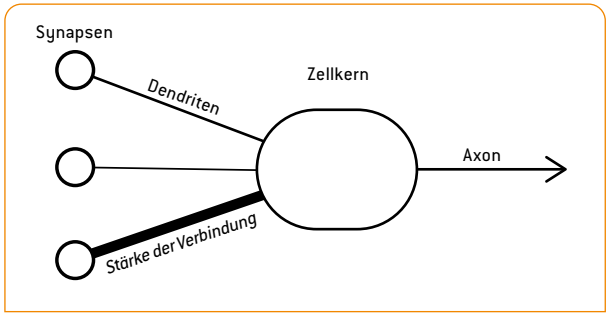



Abbildung 3

Aus diesem Modell lässt sich wiederum unmittelbar ein künstliches Neuron modellieren, wie es bereits in den 1940er Jahren mathematisch beschrieben wurde. Das folgende Neuron (s. Abbildung 4) wurde beispielhaft mit Zahlen versehen. Was diese bedeuten, wird nachstehend erläutert.

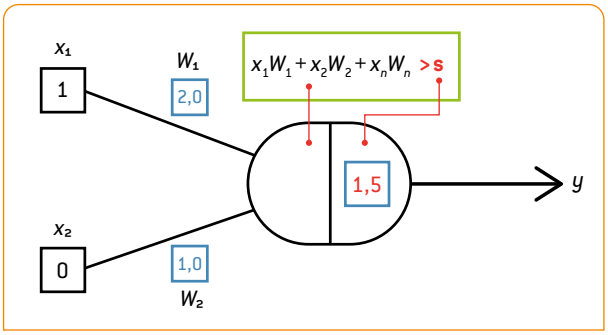



Abbildung 4

Beginnen wir mit den Eingaben: Diese sind ganz einfach die Ausgaben der davor verbundenen Neuronen. In unserem vereinfachten Modell sollen diese zunächst binär, also nur 0 oder 1 sein. Daher kann ein Eingang entsprechend aktiv (1) oder nicht aktiv (0) sein. In diesem Beispiel ist der obere Eingang 1 und der untere 0. Alle Eingaben zusammen bezeichnet man als Eingabevektor $x$ (es wird im Folgenden der Begriff Eingabekombination verwendet). Von oben nach unten sind dies also die Eingaben $x_1$, $x_2$, ..., $x_n$ bei bis zu $n$ möglichen Eingängen (in diesem Beispiel gibt es jedoch nur zwei Eingänge).

Weiter geht es mit den Synapsen: Diese sind die Kontaktstellen zwischen den Neuronen und werden in unserem vereinfachten Modell durch sogenannte Gewichte an den Eingangsverbindungen (also an den Dendriten) modelliert. Ein Gewicht ist eine Dezimalzahl, die auch negativ sein kann und neben eine Eingangskante gezeichnet wird. Die Zahl symbolisiert die Stärke der Verbindung. In der Beispielgrafik wird dies durch die beiden Zahlen 2,0 und 1,0 symbolisiert. Da jede Eingangskante ein Gewicht haben muss, sind hier zwei Gewichte zu finden: 2,0 für die obere Kante und 1,0 für die untere Kante. Das bedeutet auch, dass die obere Verbindung „stärker ins Gewicht" fällt, weil das Kantengewicht höher ist als bei der unteren Kante. Aus biologischer Sicht sind Verbindungen mit einem höheren Gewicht als Wege zu betrachten, wo die Synapsen gut ausgeprägt sind, z. B., weil dieser Weg öfter benutzt wird („vom Feldweg zur Datenautobahn", je größer der Wert wird). Alle (Synapsen-)gewichte einer Schicht bezeichnet man zusammen auch als Gewichtsvektor mit dem Buchstaben $W$. Von oben nach unten sind dies also $W_1$, $W_2$, ..., $W_n$ bei bis zu $n$ möglichen Gewichten.

Betrachten wir nun den Zellkörper: Der linke Teil der Zelle symbolisiert die Aktivierung des Neurons. Auch wenn die Formel kompliziert aussieht, ist sie es in Wahrheit nicht. Man multipliziert für jeden Dendriten (also jede Kante) den Wert des Eingangs mit dem Wert des Gewichts und addiert diese Zahlen. In unserem Beispiel aus der Abbildung 4 also: $1 \cdot 2 + 0 \cdot 1$, was als Ergebnis 2 annimmt. Die Aktivierung des Neurons ist also 2. Mit einer Aktivierung von 2 wird der Schwellenwert übertroffen, der in der Abbildung im rechten Teil der Zelle abgebildet ist und den Wert 1,5 hat. Ist die Aktivierung größer (>) als der Schwellenwert, feuert das Neuron. Dementsprechend sendet das Neuron in diesem Beispiel für die beiden anliegenden Eingaben als Ausgabe ($y$) eine 1 über sein Axon, die an anderen verbundenen Neuronen wiederum als Eingangswert fungieren kann.

## Was kann das künstliche Neuron?

Schauen wir uns an, was passiert, wenn unterschiedliche Eingangssignale in das Netz gelangen. Für die Eingabe (1,0), also $x_1 = 1$ und $x_2 = 0$, wissen wir schon, was passiert: Das Neuron produziert eine 1 als Ausgabe. Wie verhält sich das Netz für die anderen drei möglichen Kombinationen?

- Sind beide Eingänge 0, also (0,0), so ist die Ausgabe ebenso 0, denn $0 \cdot 2 + 0 \cdot 1 = 0$, was kleiner ist als der Schwellenwert.
- Ist $x_1 = 0$ und $x_2 = 1$, dann ist die Ausgabe 0, da $0 \cdot 2 + 1 \cdot 1 = 1$, was kleiner ist als der Schwellenwert.
- Sind beide Eingänge 1, also (1,1), so ist die Ausgabe 1, denn $1 \cdot 2 + 1 \cdot 1 = 3$, was größer ist als der Schwellenwert.

Dies lässt sich tabellarisch darstellen:

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Eine etwas andere Darstellung besteht darin, ein Koordinatensystem zu definieren, in dem die Eingangskanäle die Achsen aufspannen, also $x_1$ die eine Achse und $x_2$ die andere Achse ist. Die Eingabe (0,0) wird durch den Punkt (0,0) dargestellt und daher trägt man an diese Stelle den Wert für die Ausgabe ein, nämlich 0. Die Eingabe (0,1) wird dementsprechend durch den Punkt (0,1) dargestellt und auch dort wird die Ausgabe des Neurons für (0,1) eingetragen, die eben

auch 0 war. Die beiden blau gefärbten Einsen kommen zustande, da die Ausgabe $y$ für die Kombinationen (1,0) und (1,1) jeweils den Wert 1 annimmt.

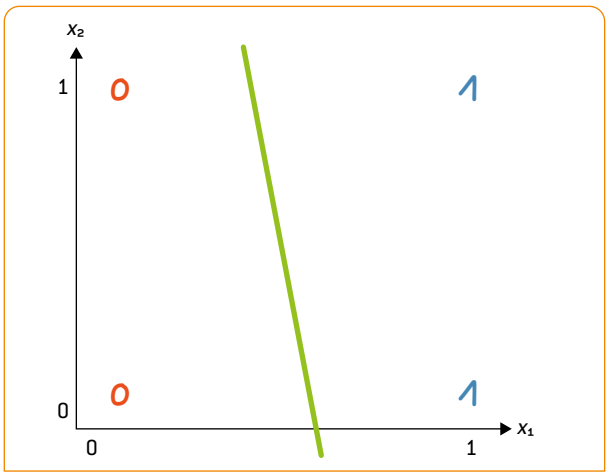



Abbildung 5

Nun lässt sich grafisch sehr anschaulich durch eine Linie zeigen, was das Netz leistet, nämlich die Nullen von den Einsen zu trennen. Das Trennen von Klassen ist die Kernaufgabe einer Klassifikation, wie wir sie im Folgenden noch verwenden werden. Anstatt es so zu betrachten, dass das Netz nun für alle Eingaben der in der Tabelle abgebildeten zweistelligen Booleschen Funktion die korrekte Ausgabe $y$ aus der Tabelle liefert (was der Fall ist), kann man es auch anders betrachten: Es gibt in diesem Beispiel vier Eingabekombinationen von $x_1$ und $x_2$. Die vier Möglichkeiten sind (0,0), (0,1), (1,0) und (1,1). Dabei gehören die ersten beiden Eingabekombinationen zur Klasse 0 und die beiden letzten zur Klasse 1 (s. Tabelle, S. 13). Das Netz hat nun so passende Gewichte und einen so passenden Schwellenwert, dass es eine Trennebene (in diesem Fall eine Trennlinie) gefunden hat, sodass alle Eingabekombinationen, die zur Klasse 0 gehören links von der Linie liegen und alle Eingabekombinationen, die zur Klasse 1 gehören rechts davon. Das sieht auf den ersten Blick trivial aus, ist aber eine zentrale und sehr mächtige Eigenschaft maschinellen Lernens, die wir später noch benötigen werden.

## Trennen von Klassen

Ein Beispiel aus dem Tierreich soll nachfolgend genutzt werden, um das Prinzip der Klassifikation im Allgemeinen zu erläutern. Es ist absichtlich vereinfacht, um an das Beispiel mit der Booleschen Funktion direkt anzuknüpfen.

In die folgende Grafik (Abbildung 6) sind Raupen (rot) und Marienkäfer (grün) eingetragen, deren Breiten und Längen gemessen wurden, indem auf der x-Achse ihre Breite und auf der y-Achse ihre Länge dargestellt ist. Bitte beachten: Die Farben dienen hier nur zur Verdeutlichung. Stellvertretend sind in der Abbildung noch die Breiten und Längen jeweils einer Raupe und eines Marienkäfers angegeben.

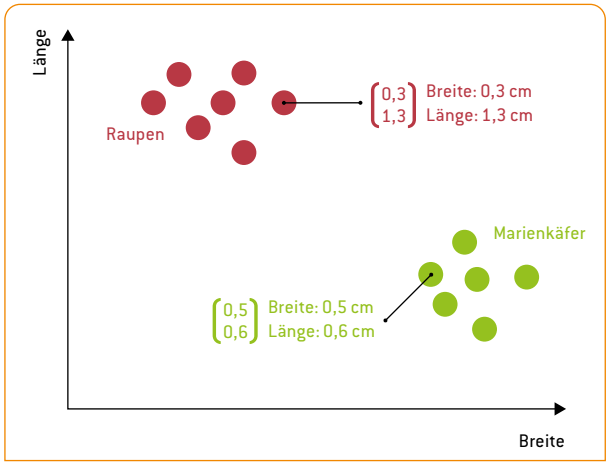



Abbildung 6[F]

Auch ohne die Farben würden wir als Menschen recht schnell erkennen, dass die Daten des beschrifteten grünen Punktes zu einem Marienkäfer und die Daten des beschrifteten roten Punktes zu einer Raupe gehören. Ein Computer kann das aber nicht ohne Weiteres erkennen, vor allem, sobald es wie im folgenden Bild (Abbildung 7) etwas komplizierter – weil realistischer – wird, wenn noch mehr Tierarten hinzukommen.

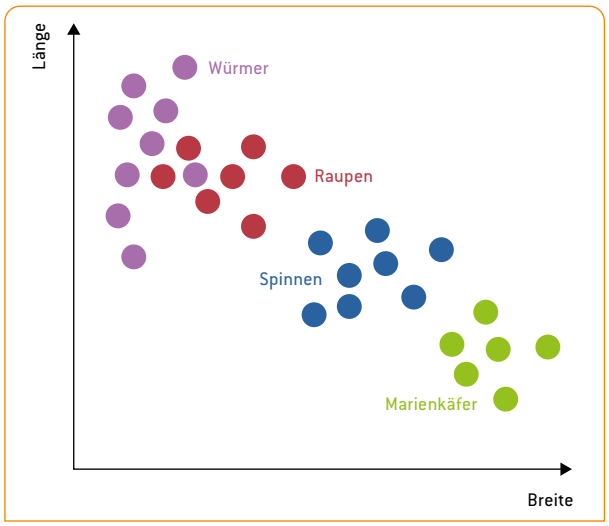



Abbildung 7[F]

F Abb. mod. nach Rashid, Tariq (2017), Neuronale Netze selbst programmieren: Ein verständlicher Einstieg mit Python, S. 9ff.

Die Aufgabe für uns Menschen – und später auch für unseren zu trainierenden Computer – besteht nun darin, eine Trennung zwischen diesen verschiedenen Typen zu finden. Reduzieren wir der Einfachheit halber diese Trennung noch einmal auf zwei Typen:

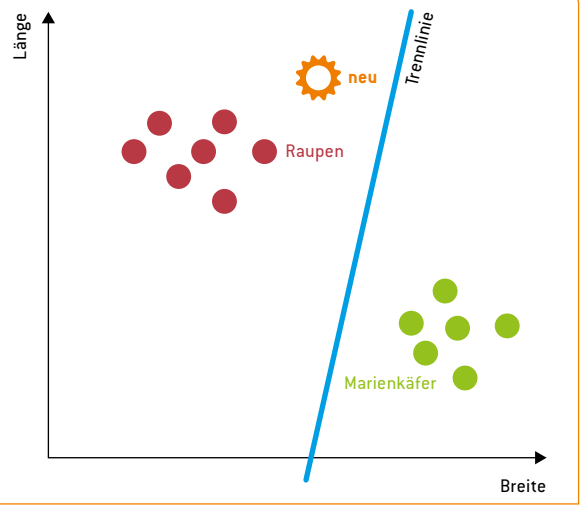



Abbildung 8[F]

Die Linie kennzeichnet, woran wir Menschen ganz intuitiv Raupen und Marienkäfer (anhand der erhobenen Messwerte) unterscheiden bzw. trennen würden: Raupen links und Marienkäfer rechts der Linie. Diese Trennung ist extrem nützlich. Denn, wenn sie gut gewählt wurde, können ab jetzt bisher unbekannte Tiere ganz einfach in die Klassen Raupe und Marienkäfer eingeteilt werden. Hat man Breite und Länge eines Tiers, trägt man den Wert ein und weiß, was für ein Tier es ist, je nachdem, auf welcher Seite der Linie die Werte des Tieres liegen. So wird in Abbildung 8 das als „neu" gekennzeichnete Tier in unserem Modell als Raupe klassifiziert – selbst wenn es in Wahrheit ein ganz anderes Tier ist.

Die Aufgabe, diese beiden Klassen zu trennen, kann man mit einem künstlichen neuronalen Netz gut lösen – auch für den Fall, dass man ganz viele Klassen hat und nicht nur zwei. Wir reduzieren jedoch zunächst im folgenden Beispiel dazu das Problem wieder ganz allgemein darauf, die Klassen 0 und 1 zu trennen. 1 könnte dabei für Raupen stehen und 0 für Marienkäfer (oder andersherum, je nachdem wie man es definiert).

Würde man ein künstliches neuronales Netz dazu bringen, diese Trennlinie zu finden, hätte man die Klassen 0 und 1 sauber voneinander getrennt. Wir vereinfachen das dargestellte Problem jedoch erst einmal noch weiter, sodass nur genau vier Werte eingetragen werden können. $x_1$ ($x$-Achse) kann an Position 0 entweder 0 oder 1 und an Position 1 entweder 0 oder 1 sein. Für $x_2$ ($y$-Achse) gilt dasselbe: Auf der $x_2$-Achse können nur zwei Werte an den Stellen 0 und 1 eingetragen werden, die jeweils auch 0 oder 1 sind. In Tabellenform ist dies etwas übersichtlicher und man erkennt, dass es sich um eine zweistellige Boolesche Funktion handelt. Die Boolesche Funktion ist genau dann 1, wenn $x_2$ 1 ist (Vergleich Abbildung 5).

| $x_1$ | $x_2$ | $y$ | |
|---|---|---|---|
| 0 | 0 | 0 | (die rote 0 links unten, wo $x_1$ und $x_2$ jeweils 0 sind) |
| 0 | 1 | 0 | (die rote 0 links oben, wo $x_1$ 0 ist und $x_2$ den Wert 1 hat) |
| 1 | 0 | 1 | (die blaue 1 unten rechts, wo $x_1$ den Wert 1 hat und $x_2$ 0 ist) |
| 1 | 1 | 1 | (die blaue 1 oben rechts, wo $x_1$ und $x_2$ den Wert 1 haben) |

Für diese Funktion funktioniert folgendes Netz, wenn man die Gewichte und den Schwellenwert passend wählt:

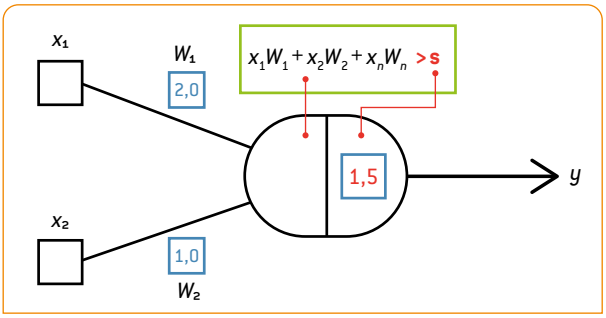



Abbildung 9

Die schwarzen Kästchen bei $x_1$ und $x_2$ stehen dabei immer als Platzhalter für eine mögliche Eingabe, die 0 oder 1 sein kann. $y$ steht für die Ausgabe.

- Sind beide Eingaben 0, so ist die Aktivierung $0 \cdot 2 + 0 \cdot 1 = 0$ und damit kleiner als der Schwellenwert.
- Ist $x_1 = 0$ und $x_2 = 1$, so ist die Aktivierung $0 \cdot 2 + 1 \cdot 1 = 1$ und damit kleiner als der Schwellenwert.
- Ist $x_1 = 1$ und $x_2 = 0$, so ist die Aktivierung $1 \cdot 2 + 0 \cdot 1 = 2$ und damit ist die Ausgabe 1.
- Sind $x_1$ und $x_2$ jeweils 1, so ist die Aktivierung $1 \cdot 2 + 1 \cdot 1 = 3$ und damit ist die Ausgabe 1.

F Abb. mod. nach Rashid, Tariq (2017), Neuronale Netze selbst programmieren: Ein verständlicher Einstieg mit Python, S. 9ff.

# Arbeitsmaterial 5

## in Kleingruppenarbeit

① Erstellt für alle folgenden zweistelligen Booleschen Funktionen – außer F6 und F9 – ein künstliches Neuron, das die Muster 0 und 1 trennt, indem ihr das Neuron mit passenden Gewichten und Schwellenwerten bestimmt und zwei Bilder wie in Abbildung 10 und 11 zeichnet: einmal das künstliche Neuron mit passenden Gewichtswerten und passendem Schwellenwert und ein Bild, in dem ihr die Nullen und Einsen, die sich aus der Funktion ergeben, durch eine einzige gerade Linie trennt.

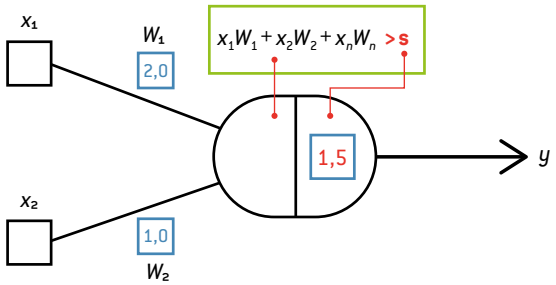



Abbildung 10

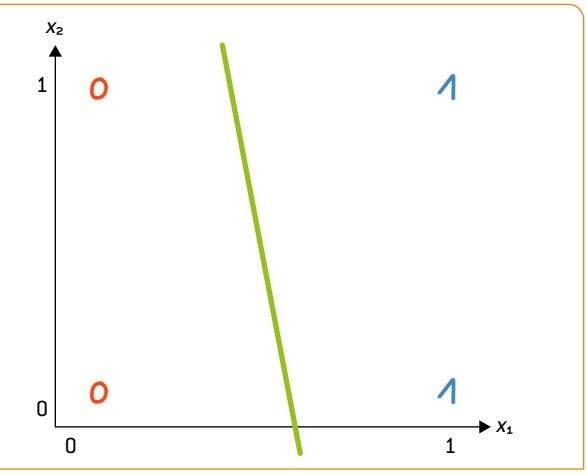



Abbildung 11

② Die Funktionen F6 und F9 lassen sich nicht mit nur einer geraden Linie trennen. Dies kann man bereits erkennen, wenn man dazu ein Bild wie in Abbildung 11 zeichnet. Es ist nicht möglich, mit einer einzigen Geraden alle Nullen auf die eine Seite der Linie zu bringen und alle Einsen auf die andere Seite. Um die Einsen daher von den Nullen trennen zu können, ist ein komplizierteres Netz mit mehr als einer Schicht notwendig. Nutzt dazu eines der folgenden mehrschichtigen Netze, die ihr mit richtigen Werten ausfüllen müsst. Wählt passende Schwellenwerte für die Neuronen (Kreise) und passende Gewichtswerte für jede Kante. Dazu dürft ihr beliebige Dezimalzahlen verwenden.

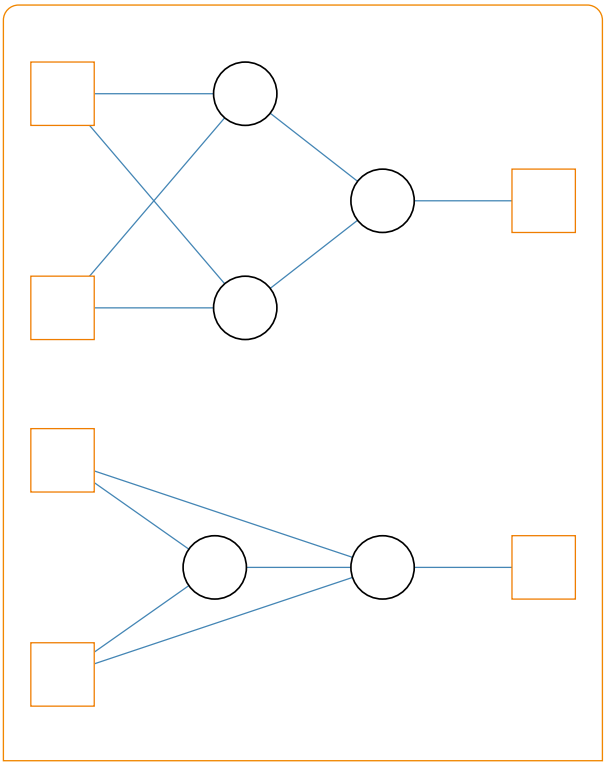
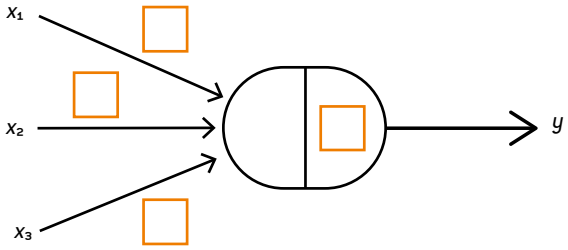


Abbildung 12

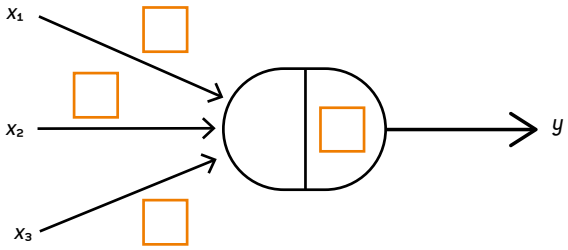
| $x_1$ | $x_2$ | F0 | F1 | F2 | F3 | F4 | F5 | F7 | F8 | F10 | F11 | F12 | F13 | F14 | F15 | F6 | F9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Tabelle mit Funktionen

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |




| $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |




| $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

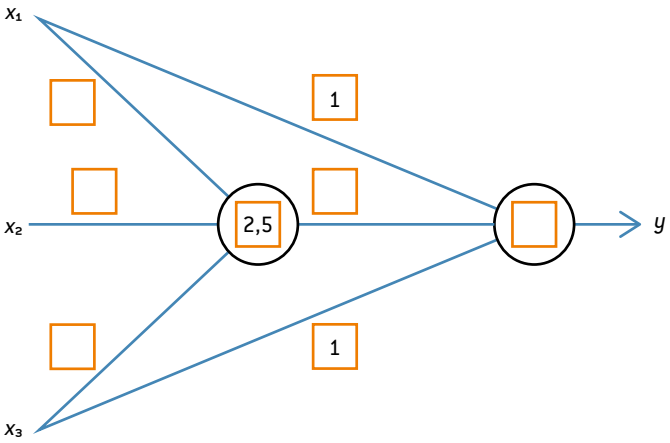



Abbildung 13

**Tipp:** Ihr könnt euer Ergebnis in der Modellierungssoftware MemBrain überprüfen, die unter www.membrain-nn.de kostenlos zum Download zur Verfügung steht. Nutzt dabei die von eurer Lehrkraft bereitgestellte Anleitung *Simulation eines Netzes in MemBrain*.

③ Ergänzt die fehlenden Werte in der oberen Abbildung, sodass künstliche neuronale Netze entstehen, die die neben stehenden Funktionen linear trennen (also korrekt umsetzen). Dazu können beliebige Dezimalzahlen verwendet werden. Überprüft eure Lösungen anschließend in MemBrain.

## Wie lernt ein neuronales Netz?

Bisher haben wir für gegebene Boolesche Funktionen versucht, die Schwellenwerte und Gewichte so zu wählen, dass das Netz die Funktion korrekt abbildet. Doch wie lernen neuronale Netze das von alleine, wenn man die Gewichte und Schwellenwerte nicht selbst bestimmt, sondern diese – wie üblich – zufällig initialisiert werden? Betrachte dazu die folgende Abbildung, in der die Startwerte für die Gewichte bei 0,5 und –0,5 liegen und der Schwellenwert 2,1 beträgt. Es ist leicht durch Überprüfen festzustellen, dass das Netz die abgebildete Funktion aus der Wahrheitstafel nicht korrekt abbildet (z. B. ist die Aktivierung $1 \cdot 0{,}5 + 1 \cdot (-0{,}5) = 0$ für die Eingabekombination (1,1) und damit kleiner als der Schwellenwert, weswegen das Neuron nicht feuert).

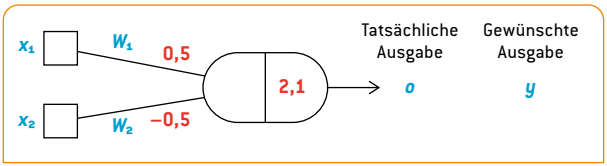



Abbildung 14

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Auch hier vereinfachen wir die Antwort, weil es keine Einschränkung im Verständnis darstellt, wenn wir uns mit einer sehr einfachen Lernregel beschäftigen, da diese immer noch die Grundidee des künstlichen Lernens beinhaltet. Im Kern wird diese Idee auch in ähnlicher Form in modernen, sehr viel komplizierteren Netzen verwendet – wenn auch mit leistungsfähigeren Varianten. Die Grundidee der hier vorgestellten Perzeptron-Lernregel[G] als Erweiterung der Hebbschen Lernregel[H] ist, dass ein Gewicht zwischen zwei Neuronen erhöht wird, wenn diese Verbindung häufig genutzt, dieses Gewicht also vom Datenfeldweg zur Datenautobahn ausgebaut wird. Diese Lernregel sieht nur auf den ersten Blick kompliziert aus.

G https://de.wikipedia.org/wiki/Perzeptron (23.09.2019)
H https://de.wikipedia.org/wiki/Hebbsche_Lernregel (23.09.2019)

Die Lernregel (hier vereinfacht) lautet:

$$W_{neu} = W_{alt} + \eta * (y - o) * x_n$$

- $x_1..x_n$: Eingaben
- $y$: gewünschte Ausgabe
- $o$: tatsächliche Ausgabe
- $\eta$: Lernrate, hier der Einfachheit halber ein fester Wert, nämlich 0,5
- $W_1..W_n$: Gewicht, das geändert wird

Das bedeutet nun, dass jedes Gewicht des Gewichtsvektors nach dieser Regel einen neuen Wert bekommt. Der neue Wert steht links vom Gleichheitszeichen (schwarz). Er ergibt sich aus dem alten, bisherigen Gewicht (blau) plus noch etwas dazu (in rot). Dieses „noch etwas" setzt sich wie folgt zusammen: Man subtrahiert die tatsächliche Ausgabe des Neurons von der gewünschten Ausgabe und multipliziert das Ergebnis mit der Lernrate $\eta$ (eta; hier mit festem Wert 0,5) und (Achtung!) mit der Eingabe (0 oder 1), die gerade an dem zu verändernden Gewicht anliegt. Dabei muss man aufpassen, dass man dies für jeden Dendriten einzeln macht und jeweils immer nur die Eingabe nimmt, die an dem jeweiligen Dendriten anliegt.

Das folgende Beispiel zeigt für die ersten vier Schritte, wie sich durch diese Lernregel die Gewichte im Laufe der Zeit ändern. Die Strategie ist immer dieselbe: Man geht die Wahrheitstafel (s. Abbildung 15) mehrmals von oben nach unten solange Zeile für Zeile durch, bis am Ende alle Gewichte so passen, dass das Netz die Funktion gelernt hat. Also erst Zeile 1 (ggf. Gewichte $W_1$ und $W_2$ ändern), dann Zeile 2 (ggf. Gewichte $W_1$ und $W_2$ ändern), dann Zeile 3 (ggf. Gewichte $W_1$ und $W_2$ ändern), dann Zeile 4 (ggf. Gewichte $W_1$ und $W_2$ ändern), und dann wieder von vorne (Zeile 1, Zeile 2, ... usw.). Am Ende hat Gewicht $W_1$ den Wert 2,5 und Gewicht $W_2$ den Wert 0,5. Wie man leicht überprüfen kann, passt es dann.

Die folgende Abbildung zeigt dies für die ersten vier Schritte: Zunächst wird die violette Zeile betrachtet. Die Eingaben sind 0, also ist die Aktivierung des Neurons 0 und das Neuron feuert nicht – was auch so sein soll. Im nächsten Schritt (grüne Zeile) ist es ebenso. Die Aktivierung von –0,5 übertrifft den Schwellenwert nicht. Da das Neuron für die Eingaben jedoch auch nicht feuern soll, muss nichts geändert werden. Im dritten Schritt (orange markiert) finden die ersten Änderungen statt. Die tatsächliche weicht von der gewünschten Ausgabe ab und dementsprechend ändert sich nur das obere Gewicht $W_1$ um 0,5, da hier der Eingang 1 war $(0{,}5 + 0{,}5 \cdot (1 - 0) \cdot 1)$.

Eine Änderung findet auch im vierten Schritt statt (türkis markiert): Hier tragen beide Eingaben dazu bei, dass gewünschte und tatsächliche Ausgabe nicht übereinstimmen, weswegen nach der Formel beide Gewichte um jeweils 0,5 verändert – in diesem Fall erhöht – werden. Da das Neuron immer noch nicht für alle Eingaben die gewünschten Ausgaben liefert, geht dieses Prozedere wieder von vorne los – bis die Gewichte passen und schließlich die Werte 2,5 und 0,5 haben.

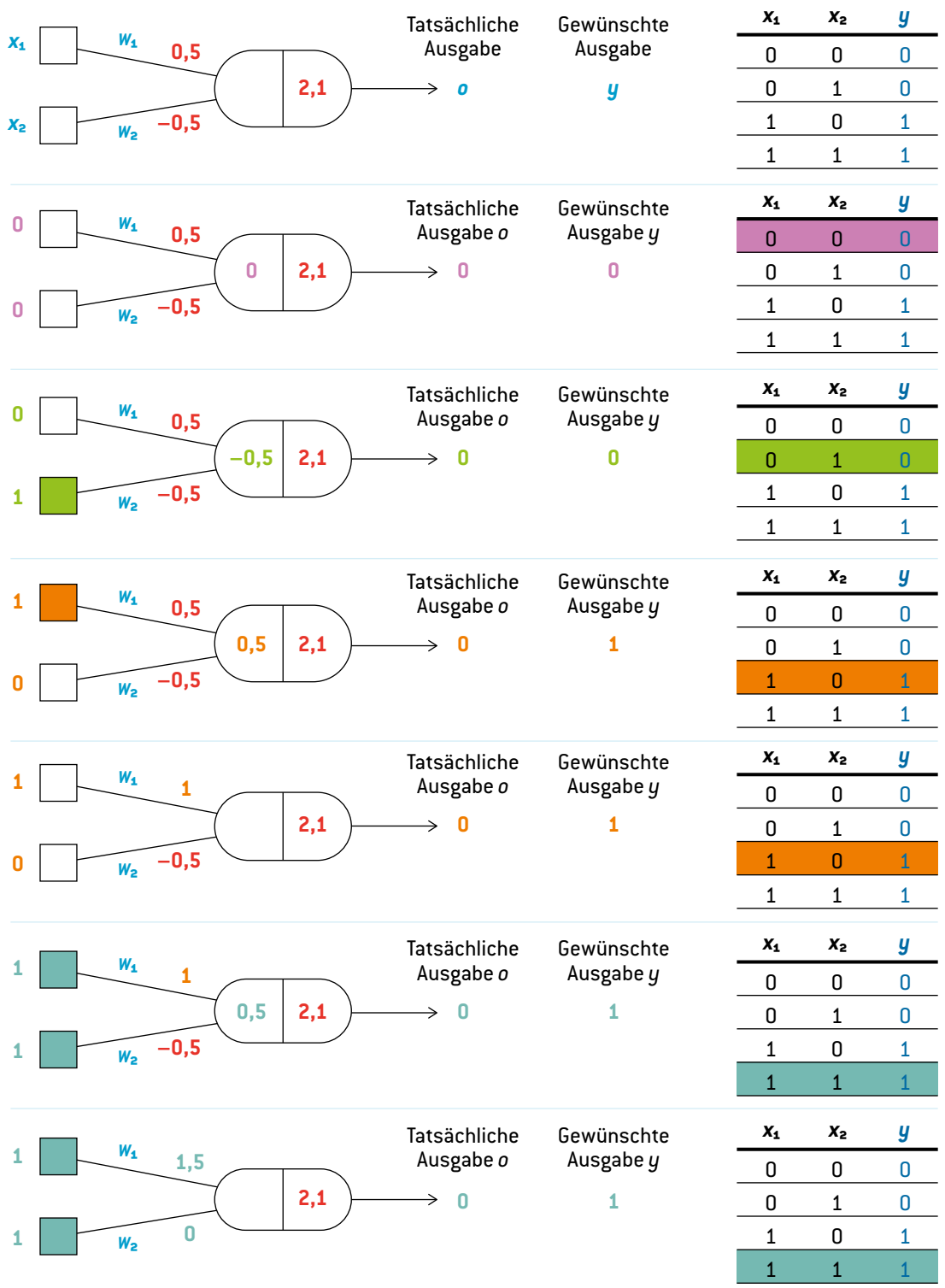



Abbildung 15

# Arbeitsmaterial 6

① Lade MemBrain[1] herunter und installiere es auf deinem PC. Nutze die von deiner Lehrkraft bereitgestellte Anleitung *Klassifikation mit einer Booleschen Funktion in MemBrain* und simuliere zunächst von Hand die Gewichtsänderungen für das folgende Netz mit den gegebenen Startgewichten mittels der Lernregel. Simuliere anschließend in MemBrain mithilfe der Anleitung, wie das Netz dies automatisch lernt.

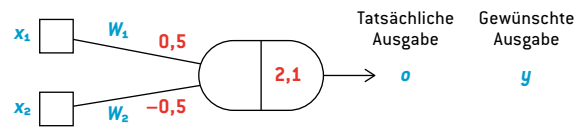



Abbildung 16

② Wie sehen die Gewichte und Schwellenwerte des künstlichen Neurons aus der Abbildung 17 aus, nachdem es mit der Lernregel die folgende Boolesche Funktion gelernt hat ($\eta = 0{,}5$)? Rechne zunächst von Hand die Gewichtsänderungen aus, indem du die neben stehende Tabelle ergänzt und simuliere dies danach in MemBrain, um deine Lösung zu überprüfen.

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

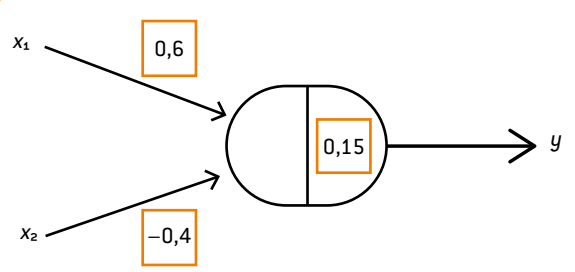



Abbildung 17

Die Lernregel lautet:

$$W_{neu} = W_{alt} + \eta * (y - o) * x_n$$

Ergänze dazu die folgende Tabelle:

- → $x_1$ und $x_2$: Eingaben
- → $y$: gewünschte Ausgabe
- → $o$: tatsächliche Ausgabe
- → $W_1$, $W_2$: Gewichte, hier jeweils die neuen Werte nach der Berechnung

| $x_1$ | $x_2$ | $y$ | $o$ | $W_1$ | $W_2$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | | | |
| 0 | 1 | 1 | | | |
| 1 | 0 | 0 | | | |
| 1 | 1 | 1 | | | |
| | | | | | |
| | | | | | |

## Aktivierungsfunktionen bei komplizierteren neuronalen Netzen

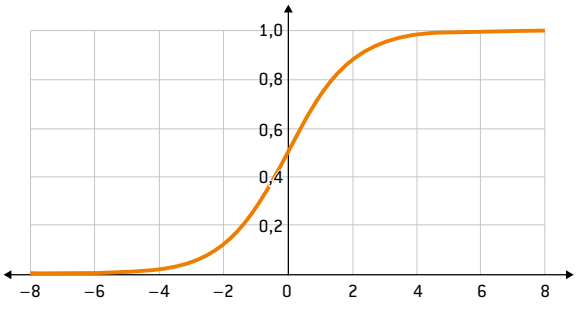



Abbildung 18

Als Erweiterung der binären Schwellenwertfunktion (0 = nichts; 1 = feuern) wird die Aktivierung eines künstlichen Neurons in performanteren Netzen durch eine Aktivierungsfunktion modelliert, die nicht ab einem bestimmten übertroffenen Schwellenwert 1 ist und sonst 0, sondern den anliegenden

1 www.membrain-nn.de (19.07.2019)

Aktivierungswert eines Neurons (auf der $x$-Achse dargestellt) – wie in der Abbildung 18 angegeben – in den entsprechenden $y$-Wert umwandelt. Man erzielt hiermit in der Praxis bessere Resultate. Ein weiterer Vorteil liegt darin, dass nun der gesamte Wertebereich [0..1] genutzt werden kann und das Netz so viel feinere Abstufungen durch kontinuierliche Werte zulässt.

## Datennormierung

Generell ist es eine gute Idee, die Daten so aufzubereiten, dass jedes Merkmal im Wertebereich [0..1] abgebildet wird. Würde man z. B. Gewicht und Größe einer Person als zwei Merkmale für eine Eingabekombination nehmen, hätte man für $x_1$ Werte von z. B. 40 bis 160 [in kg] und für $x_2$ Werte von ca. 1,4 bis 2,0 [in m] oder 140 bis 200 [in cm]. Da die Gewichts- und Größenmerkmale gar nicht in vergleichbaren Größenordnungen liegen, könnte es passieren, dass das neuronale Netz ein Merkmal als wichtiger behandelt, weil die Werte in absoluten Zahlen größer sind. Daher sollten die Wertebereiche ungefähr in der gleichen Größenordnung liegen. Man kann dies erreichen, indem man beide Werte auf den Bereich [0..1] abbildet. Das geht z. B. in der einfachsten Form für das Gewicht von 0,2 bis 0,8 (durch 200 teilen) und für die Größe von 0,7 bis 1,0 (durch 2 teilen). Selbstverständlich kann man in verschiedenen Tabellenkalkulationsprogrammen (z. B. Excel oder Numbers) auch bessere Normierungsfunktionen, die den gesamten Eingabebereich eines Merkmals auf den Wertebereich [0..1] abbilden, verwenden oder selbst programmieren. Für unsere Belange reicht an dieser Stelle die einfache Vorgehensweise aus.

## Ein Klassifizierungsbeispiel aus der realen Welt mit dem Iris-Datensatz

Der Statistiker Ronald Fisher machte durch eine Veröffentlichung 1936 den Iris-Datensatz weltbekannt. Bis heute wird dieser Datensatz gerne verwendet, um statistische Methoden zu demonstrieren. Hierzu gehören im weitesten Sinne auch künstliche neuronale Netze.

Der Datensatz besteht aus 150 Eingabekombinationen aus drei Klassen. Statt Breite und Länge eines Tiers zu vermessen oder $x_1$ und $x_2$ bei einer Booleschen Funktion zu benutzen (wie in unseren Eingangsbeispielen), verwendet der Datensatz vier Merkmale mit realen Messdaten. Diese vier Merkmale sind die gemessenen Werte in Zentimetern bei unterschiedlichen Iris-Blumen, und zwar jeweils die Kelchblattlänge, Kelchblattbreite, Kronblattlänge und Kronblattbreite. Jene vier Werte wurden für 50 *Iris setosa,* 50 *Iris versicolor* und 50 *Iris virginica* erhoben. Insgesamt besteht der Datensatz also aus einer Tabelle mit 150 Eingabekombinationen (aus drei Klassen) mit jeweils vier Merkmalen. Damit die Klassen der jeweiligen Eingabekombination zugeordnet werden können, befindet sich als fünfter Eintrag hinter den vier gemessenen Werten das sogenannte Klassenlabel, sodass das neuronale Netz beim Training weiß, zu welcher der drei Klassen die Blume gehört. Im Folgenden wird für jede Blume ein beispielhafter Eintrag aus dem Datensatz angegeben:

| | | | | |
|---|---|---|---|---|
| 0.5 | 0.35 | 0.13 | 0.03 | 0 |
| 0.55 | 0.26 | 0.44 | 0.12 | 0.5 |
| 0.67 | 0.31 | 0.56 | 0.24 | 1 |

Die Klasse 0 soll dabei für die Setosapflanze stehen, die 0.5 für die Versicolor- und die 1 für die Virginicapflanze. Eine solche Kodierung ist üblich, um die drei Klassen optimal auf den Wertebereich des Ausgabeneurons von [0..1] zu verteilen. Die vier Merkmale sind von links nach rechts: Kelchblattlänge, Kelchblattbreite, Kronblattlänge und Kronblattbreite.

## Die Idee der Klassifizierung

MemBrain erlaubt es, den Datensatz über den „Lesson Editor“ als CSV-Datei einzulesen. Es muss jedoch darauf geachtet werden, dass die Werte in den Bereich [0..1] normiert und mit Punkt statt Komma als Dezimaltrennzeichen in der Datei gespeichert werden. Daher wurden die Daten aus dem Originaldatensatz hier bearbeitet und normiert. Zusätzlich muss beachtet werden, dass eine CSV-Datei die Werte durch Komma bzw. Semikolon trennt und es eine Titelzeile geben muss, die die exakt gleichen Namen der Ein- und Ausgabeneuronen in MemBrain enthält (s. dazu ggf. die zugehörige *Anleitung zur Gesichtserkennung mit MemBrain*). Eine Titelzeile könnte dementsprechend so aussehen, wenn die Namen der Neuronen ebenso in MemBrain benannt werden:

```
in1;in2;in3;in4;out1
```

Um zu überprüfen, ob das Netz auch wirklich in der Lage ist, eine Trennfunktion für die drei Klassen zu finden, wird der Datensatz gesplittet. Dazu kann man z. B. jeweils 40 Blumen

pro Klasse für das Training benutzen und im Anschluss daran mit den verbleibenden 10 Blumen pro Klasse testen, ob diese alle den richtigen Klassen zugeordnet werden können. Das von der Lehrkraft bereitgestellte *Iris-Beispiel* bestehend aus MemBrain-Netz, Trainings- und Testdatensatz kann verwendet werden, um eine Klassifikation durchzuführen. Auch wenn zu Beginn des Trainings alle Gewichtswerte des Netzes randomisiert werden, ist das Netz meistens in der Lage, ein erfolgreiches Training durchzuführen. Sobald der Fehler des Netzes nach einigen 100 Trainingsschritten klein genug ist, kann das Training abgebrochen und es kann getestet werden, wie gut die Testdaten den (richtigen) Klassen zugeordnet werden können. In diesem Beispiel kann es passieren, dass das Netz stellenweise hängen bleibt, im Regelfall sollte aber eine hundertprozentige Zuordnung möglich sein. Dass hier zumeist 100 Prozent Erkennungsrate erzielt werden, liegt daran, dass nur wenig Testdaten benutzt wurden bzw. nur ein sehr kleiner Datensatz und evtl. zu lange trainiert wurde. In einer „echten" Klassifikation (z. B. echte Objekterkennung mit Millionen von Datensätzen) ist eine Performance von 80–90 Prozent schon ein sehr guter Erfolg. Es bedeutet jedoch auch immer, dass ein größerer Teil an Falschklassifikationen zu erwarten ist, was eben nicht ausgeschlossen werden kann und in Kauf genommen werden muss. Das sollte man immer im Gedächtnis behalten, wenn man sich mit künstlichen neuronalen Netzen beschäftigt.

## Vorgehensweise

① Lade das Netz „iris.mbn" in MemBrain. Tipp: Beachte, dass im Menü „Teach" unter „Teacher Manager" nicht die Hebbsche Lernregel mit der Lernrate 0,5 ausgewählt ist. Das würde nicht funktionieren. Erstelle stattdessen eine neue Lernregel mit den Standardeinstellungen (z. B. „BP with Momentum full loopback support").

② Öffne den „Lesson Editor" (im Menü „Teach") und klicke im Menü „Raw CSV Files" auf den Eintrag „Import Current Lesson (Raw CSV)", um die Datei „iris_train.csv" zu laden.

③ Schließe den „Lesson Editor" und klicke anschließend im Menü „Teach" zunächst auf „Net Error Viewer" und dann auf „Start Teacher (Auto)", um das Training zu starten.

④ Stoppe das Training, wenn sich die Fehlerkurve der 0 annähert und öffne wiederum den „Lesson Editor".

⑤ Importiere auf die gleiche Weise wie oben die Datei „iris_test.csv" und navigiere mit den Pfeil-Buttons durch die 30 Test-Patterns, um bei jedem Muster im Lesson Editor mit dem Button „Think on Input" zu überprüfen, ob das Muster am ehesten der Klasse 0, 0,5 oder 1 zugeordnet wird.

## Aufgabe

Ändere das Netz, indem du z. B. Neuron 4 löschst (auswählen und ENTF auf der Tastatur drücken). Analysiere, welche Auswirkungen dies auf die Klassifikation hat, indem du wiederum alle 30 Testmuster durchklickst und mittels „Think on Input" bewerten lässt.

## Lösung

Wenn du nicht gerade genau das falsche Neuron eliminiert hast, ist die Performance des Netzes zwar schlechter als zuvor, jedoch immer noch ganz passabel. Im Notfall könnte man dem Netz mit der neuen Anzahl an Neuronen von vorne beibringen, die Muster zu lernen und es würde wieder gut funktionieren. Dies wird selbstverständlich schlechter, je mehr Neuronen du löschst. Würde man so etwas mit einigen Transistoren auf der Hauptplatine eines PCs machen oder aus einem Schaltnetz ein Gatter herausreißen, würde vermutlich nichts mehr funktionieren. Bei einem neuronalen Netz ist es jedoch anders, nämlich eher wie mit unserem Gehirn. Auch wenn einzelne Nervenzellen ausfallen oder absterben sollten, verlernen wir nicht gleich alles.

In den folgenden Aufgaben wird eine Gesichtserkennung durchgeführt, die prinzipiell genauso funktioniert wie die Erkennung der Blumen im Iris-Datensatz.

# Arbeitsmaterial 7

## Gruppenarbeit

① Nutzt die von eurer Lehrkraft bereitgestellte *Anleitung zur Gesichtserkennung mit MemBrain* und bearbeitet diese in der Gruppe. Welche Erkennungsrate schafft ihr?

② Diskutiert zunächst in der Gruppe und anschließend in der Klasse, welche Gefahren bestehen, wenn maschinelles Lernen missbräuchlich eingesetzt wird. Worauf müssen wir zukünftig achten? Welche Probleme könnten auf uns zukommen? Vielleicht helfen euch die folgenden Stichworte: Deep Fake Videos, autonomes Fahren, Moral Machine, Tay (Chatbot von Microsoft), Fake News, Militärschläge der USA im Nahen Osten, Kreditfähigkeitsbestimmung, Personaleinstellung mittels künstlicher Intelligenz, Compas (correctional offender management profiling for alternative sanctions), Watson Therapieempfehlung, Skynet (der NSA). Findet noch weitere Beispiele!

## Eine weitere Anwendung: Interpolation

Künstliche neuronale Netze sind hervorragend darin, zwischen Daten, mit denen sie trainiert wurden, zu „interpolieren". Interpolieren bedeutet „abschätzen". Dieses Abschätzen zeigt sich u. a. darin, dass in unseren bisherigen neuronalen Netzen für die Testdaten jeweils eine Klassenzugehörigkeit (korrekte Ausgabe) bestimmt werden konnte, obwohl diese Testdaten gar nicht im Trainingsdatensatz waren. Das Maß anhand dessen z. B. eine unbekannte Blume der richtigen Klasse zugeordnet wird, hat stark mit Ähnlichkeit zu tun. Sind die vier Merkmale einer unbekannten Blume ähnlich zu denen einer bereits bekannten Blume, so kann die Klassenzugehörigkeit gut abgeschätzt werden. Diese Fähigkeit der Interpolation kann man auch nutzen, um aus (wenigen) Trainingsbeispielen eine Funktion $f(x)$ abzuschätzen, für die das Netz auch alle $y$-Werte interpolieren kann, deren $x$-Werte nicht zum Trainingsdatensatz gehörten. Ein dreischichtiges neuronales Netz kann dabei sogar jede beliebige mathematische Funktion annähern. Die folgende Grafik zeigt das beispielhaft für die Funktion $f(x) = x^2$.

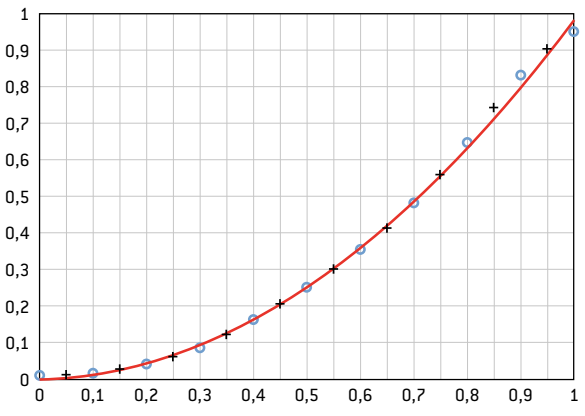



Abbildung 19

Dazu wurde in diesem Fall ein vierschichtiges Netz aus 1-2-4-1 Neuronen konstruiert – mit einem Eingabe- ($x$) und einem Ausgabeneuron ($y$). Als Trainingsdaten wurden folgende $x/y$-Kombinationen gewählt:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ***x*:** | 0 | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | 0,7 | 0,8 | 0,9 | 1,0 |
| ***y*:** | 0 | 0,01 | 0,04 | 0,09 | 0,16 | 0,25 | 0,36 | 0,49 | 0,64 | 0,81 | 1,0 |

Die Trainingsdaten wurden genutzt, um das Netz zu trainieren. Im Anschluss wurde das Netz zunächst wieder mit den Trainingsdaten getestet. Es wurde geschaut, welche $y$-Werte das Netz nun wirklich für die gelernten Daten ausgibt. Diese $y$-Werte wurden als blaue Kreise in die Grafik eingezeichnet. Wie man sehen kann, liegen sie relativ genau auf der in Rot eingezeichneten Funktion $x^2$. Im Anschluss wurde die Interpolationsfähigkeit des Netzes getestet. Dazu wurden die Ausgaben des Netzes für die $x$-Werte 0,05, 0,15, 0,25, 0,35, 0,45, 0,55, 0,65, 0,75, 0,85 und 0,95 berechnet und als schwarze Kreuze in die Grafik eingetragen. Diese $x$-Werte kannte das Netz noch nicht aus dem Training. Auch hier ist keine perfekte Genauigkeit vorhanden, aber eine sehr schön zu beobachtende Interpolation, die zeigt, dass das Netz die Funktion $x^2$ recht gut gelernt hat.

# Arbeitsmaterial 8

Modelliere ein künstliches neuronales Netz in MemBrain mit einem Eingabe- und einem Ausgabeneuron sowie beliebig vielen versteckten Neuronen. Erstelle im „Lesson Editor“ ca. zehn Muster mit den $x$- und $y$-Werten einer beliebigen Funktion (als Ein- und Ausgabe). Auch wenn dies in MemBrain theoretisch anders möglich ist, beschränke den $x$- und $y$-Wertebereich hier auf [0..1]. Sollte dir keine geeignete Funktion einfallen, kannst du es mit $f(x) = x$ oder mit $f(x) = x^2$ versuchen.

## Eine weitere Anwendung: Autoencoder

Künstliche neuronale Netze lassen sich nicht nur zur Klassifikation bzw. zum Erkennen von Subjekten oder Objekten, sondern auch für viele andere Anwendungen verwenden. So werden spezielle neuronale Netze, die man Autoencoder nennt, u. a. dafür genutzt, Daten zu komprimieren. Eine beispielhafte Anwendung findet sich im Folgenden, nachdem erläutert wird, wie ein solcher Autoencoder aufgebaut ist.

Ein neuronales Netz wird als Autoencoder bezeichnet, wenn es einen Eingabevektor wie z. B. (0.1 0.2 0.3 0.4) nicht auf einen einer Klasse zugehörigen Wert wie 0 oder 1 abbildet, sondern auf den gleichen Eingabevektor (0.1 0.2 0.3 0.4). Es besteht somit folgende Funktionalität:

$AE(x) = x$, wobei AE der Autoencoder ist und $x$ der Eingabevektor.

Ein solches Netz muss also gleich viele Eingabe- wie Ausgabeneuronen haben und ist zumeist symmetrisch aufgebaut. Das Netz wird darauf trainiert, für einen Eingabevektor $x$ den nahezu gleichen Eingabevektor als Ausgabe zu reproduzieren. Das klingt auf den ersten Blick verwirrend, weil das Netz hier keine Klassifikationsleistung vollbringt wie wir es bislang kennengelernt haben. Bisher wurde z. B. ein Netz darauf trainiert, für den Eingabevektor (0.1 0.1 0.1) die Ausgabe 1 zu generieren und für den Eingabevektor (0.9 0.9 0.9) die Ausgabe 0. Ein solches neuronales Netz ist aber trotzdem im Bereich Machine Learning neben anderen Verfahren wie der Hauptkomponentenanalyse (auch PCA, Principal Component Analysis, genannt) eine wichtige Methode, um Daten und Vektoren mit großer Eingabedimension – also vielen anliegenden Werten – auf wenige Dimensionen (Werte) zu komprimieren. Wie das funktioniert, zeigt die folgende Abbildung:

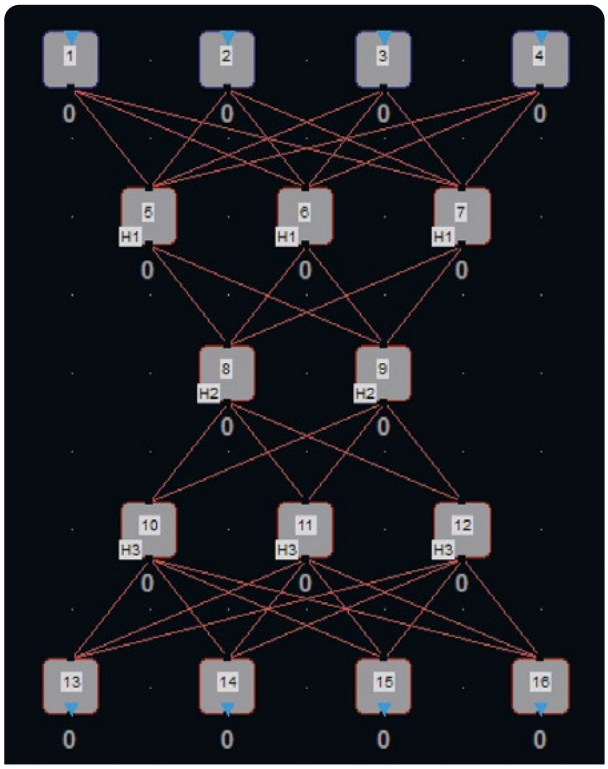



Abbildung 20

Wenn nun der Autoencoder solange trainiert wird, bis er für jeden Eingabevektor auch in etwa dasselbe als Ausgabe produziert, also z. B. für $x$ = (0.1 0.2 0.3 0.4) auch ungefähr $y$ = (0.1 0.2 0.3 0.4), ist er in der Lage, die Eingabe auf die Ausgabe „zu kodieren“ (daher z. T. auch der Name). Weil ein solcher Eingabevektor schichtweise durch das Netz fließt, befinden sich die Eingabedaten in der mittleren Schicht in einem Zustand, der nur zwei Neuronen zur Repräsentation des Vektors benötigt. Und da vier Neuronen die Eingabe repräsentieren und auch erfolgreich wieder auf vier Neuronen abgebildet werden können, gehen in der mittleren Schicht nicht viele Informationen verloren. Zu genau diesem Zeitpunkt sind die Informationen jedoch so komprimiert, dass nicht mehr vier Werte nötig sind, um den Eingabevektor zu beschreiben, sondern zwei Neuronen ausreichen. Es findet hier eine Datenkomprimierung statt wie sie auch beim Komprimieren von einigen Bildformaten in ähnlicher Form genutzt wird. Der Vorteil liegt auf der Hand: Möchte man bspw. das Körpergewicht und die Körpergröße zur Visualisierung in

ein Koordinatensystem eintragen, geht dies in einem zweidimensionalen Koordinatensystem ganz gut. Nimmt man jedoch noch die Schuhgröße hinzu, benötigt man schon ein dreidimensionales Koordinatensystem, was sich ausgedruckt auf Papier nur noch schlecht darstellen lässt. Bei mehr als drei Merkmalen (Bauchumfang, Alter, Beinlänge, Haarfarbe etc.) kann man die Daten gar nicht mehr darstellen. Mit einem Autoencoder hingegen ist es theoretisch möglich, beliebig viele Merkmale so zu reduzieren, dass man diese auf zwei Neuronen abbildet und deren Aktivierung (zwischen 0 und 1) dann in ein normales zweidimensionales Koordinatensystem einträgt. Zwar hat man so zwei Merkmale, die irgendwie irgendwelche Merkmale zusammenfassen und sich nicht mehr isoliert beschreiben lassen, aber diese Vorgehensweise bietet zumindest den Vorteil, dass ähnliche Eingaben auch in ähnlichen Regionen des Koordinatensystems abgebildet werden und man einen guten Überblick über die Gesamtdaten erhält.

Auf diese Weise könnte man den Iris-Datensatz auf zwei Merkmale reduzieren. Die vier Merkmale für jede Blume (Kelchblattlänge, Kelchblattbreite, Kronblattlänge und Kronblattbreite) lassen sich nicht derart in ein Koordinatensystem eintragen, dass man auf Anhieb erkennt, wie sich die drei Klassen verteilen. Man kann höchstens paarweise immer zwei Merkmale miteinander vergleichen. Mittels Autoencoder ist es jedoch möglich, einen Eingabevektor wie (0.51 0.35 0.14 0.02) auf z. B. (0.34 0.07) zu reduzieren und diesen als Punkt in ein zweidimensionales Koordinatensystem einzutragen. Für alle 150 Eingabevektoren ergibt sich dadurch folgendes Bild, in dem die drei unterschiedlichen Klassen deutlich zu erkennen sind (die Achsen stellen jeweils die Aktivierungen der beiden mittleren Neuronen dar):

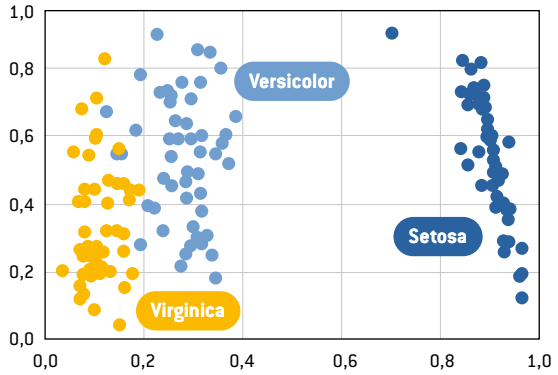



Abbildung 21

# Arbeitsmaterial 9

Die folgende Aufgabe sieht vor, einen Fragebogen mit mindestens fünf Fragen zu entwerfen und die Antworten auf ein zweidimensionales Koordinatensystem abzubilden, um einen Überblick zu erhalten. Dabei sollte jede Frage erfordern, die Antwort auf einer Skala zwischen 0 und 1 einzutragen (z. B. „gefällt mir gar nicht“ bis „gefällt mir sehr gut“). Für jede Person, die daran teilnimmt, erhält man dann einen Eingabevektor, der wie folgt aussehen könnte:

(0.3 0.3 0.2 0.7 0.8)

Wird das Autoencoder-Netz mit allen Kombinationen aus Eingabe- und Ausgabevektoren – die ja identisch sind – trainiert, kann man im Anschluss für jeden Eingabevektor die Aktivierungswerte der mittleren beiden Neuronen in MemBrain ablesen und diese in ein zweidimensionales Koordinatensystem eintragen. So erhält man eine zweidimensionale Grafik mit einer Übersicht, welche Personen ähnliche Einschätzungen abgegeben haben.

Entwickelt einen solchen Fragebogen und erhebt damit ca. 15–20 Datensätze.

Tragt die Werte in eine CSV-Datei ein und orientiert euch an dem nachfolgenden Schema. Die Neuronen müssen dieselben Namen tragen wie in der CSV-Datei in der ersten Zeile angegeben; die zehn Werte sind je zwei Mal die fünf erhobenen Daten hintereinander kopiert.

in1;in2;in3;in4;in5;ou1;ou2;ou3;ou4;ou5
0.3;0.3;0.2;0.7;0.8;0.3;0.3;0.2;0.7;0.8

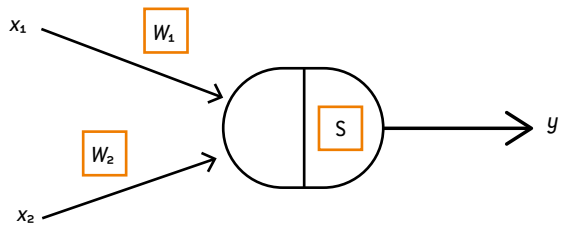
Erstellt danach zu zweit einen Autoencoder in MemBrain, importiert die Trainingsdaten als CSV-Datei und trainiert das

Netz. Für jede Person können mittels „Think On Input“ (mit den gleichen Daten des Trainingsdatensatzes) die Aktivierungen der beiden mittleren Neuronen abgelesen und in eine Tabellenkalkulation eingetragen werden. So lässt sich später daraus ein Diagramm erstellen.

# Arbeitsmaterial 10

① Analysiere die folgenden Varianten desselben Netzes und begründe anhand eines Beispiels, weshalb es prinzipiell irrelevant ist, ob man ein neuronales Netz durch Variante A oder Variante B modelliert:

**Variante A**




**Variante B**

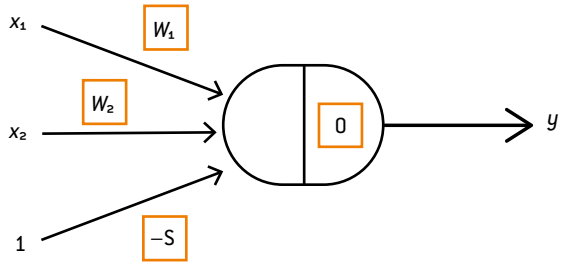



Abbildung 22

**Hintergrund:** Um die Umsetzung eines neuronalen Netzes in einer Programmiersprache so einfach wie möglich zu gestalten, werden Schwellenwerte häufig als zusätzliche Eingänge modelliert und der Schwellenwert auf den Wert 0 gesetzt. Dies funktioniert ebenso gut.

② Bei künstlichen neuronalen Netzen werden die Gewichte eines Netzes, das meist aus vielen einzelnen künstlichen Neuronen besteht, vor dem Lernen mit zufälligen Werten initialisiert. Beurteile mithilfe der Lernregel, warum es für das Verhalten des Netzes prinzipiell egal ist, mit welchen „Startwerten“ es angelegt wird.

③ Maschinelles Lernen ermöglicht einem künstlichen neuronalen Netz in Millisekundenschnelle Gesichter wiederzuerkennen, die es vorher erlernt hat. Diese Technik kann natürlich nicht nur auf Gesichter angewendet werden, sondern auch auf menschliche Gangmuster. Die Art, wie wir gehen, ist so individuell wie ein Fingerabdruck und eignet sich bestens dafür, Personen anhand ihrer Gangmuster biometrisch zu erkennen oder auch in Kategorien wie „jung – alt“ oder „gesund – krank“ einzuordnen. Dazu muss das Netz einmal mit genug Daten – z. B. über eine Videokamera mit entsprechender „kluger“ Software – trainiert werden, sodass es neue, fremde Daten in das zugrunde liegende Schema einordnen kann. Politiker X hat von dieser Methode gehört und fordert, die Technik weiter auszubauen. So könnten seiner Meinung nach Netze entwickelt werden, die unter anderem Gangmuster in die Kategorien „terrorverdächtig – nicht terrorverdächtig“ einordnen.

Nimm begründet Stellung zu der Idee, zu den Möglichkeiten, Grenzen, Gefahren und den sich daraus ergebenden Konsequenzen!

④ Neuronale Netze wie sie für Bild- und Spracherkennung von Apple, Facebook, Google & Co. verwendet werden, besitzen nicht nur wesentlich mehr Neuronen und Schichten als in unseren Beispielen, sondern werden auch mit millionenfach mehr Daten trainiert, damit es nicht zum sogenannten Overtraining[J] kommt. Informiere dich über das Thema Over- bzw. Undertraining bei neuronalen Netzen im Internet und begründe, warum die von uns durchgeführte Gesichtserkennung mit ziemlicher Sicherheit ein Overtraining provoziert hat.

J https://de.wikipedia.org/wiki/Überanpassung (23.09.2019)

⑤ Die Performance und schlussendlich der Erfolg eines künstlichen neuronalen Netzwerks hängen von der Auswahl der Trainingsdaten ab. Diskutiert in der Gruppe, welche Problematik sich daraus ergeben könnte.

⑥ Der Mensch denkt nicht nur in Mustern – er kann bspw. Unsinn und Humor in Bildern erkennen. Diskutiert in der Gruppe, ob künstliche neuronale Netze nur „gute Augen" bzw. „Mustererkenner" sind oder inwiefern man ihre Leistung als intelligent bezeichnen kann. Betrachtet in diesem Zusammenhang noch einmal die Hebbsche Lernregel. Ist sie die ultimative Intelligenzformel? Begründet eure Ausführungen.

⑦ Ist ein so unglaublich komplexes Netz wie das Quickdraw-Netz von Google intelligent oder nur ein Blendwerk an Komplexität? Schaue abschließend noch einmal in die Daten hinein, indem du am Ende eines Spiels auf ein nicht erkanntes Bild in Google Quickdraw klickst. Es zeigt dir an, wofür es deine Zeichnung gehalten hat. Versuche, mit deinem erworbenen Wissen ein Urteil zu fällen.

Wir haben einen kleinen Einblick in die Welt des maschinellen Lernens erhalten. Zwar erfolgte das sehr didaktisch reduziert, indem wir nur einfache künstliche neuronale Netze kennengelernt haben, dennoch sollte der Grundgedanke klar geworden sein: Im Gegensatz zum Entwurf eines Algorithmus, in dem die Programmiererin bzw. der Programmierer genau definiert, wie eine Eingabe $x$ verarbeitet wird, damit eine Ausgabe $y$ berechnet werden kann, lernen überwachte künstliche neuronale Netze dies selbst, indem ihnen Paare aus $x$ und $y$ präsentiert werden und das Netz den Zusammenhang zwischen $x$ und $y$ selbstständig lernt und später dazwischen interpolieren kann.

Wie das jedoch genau abläuft, wie also der interne „Algorithmus“ aussieht, der $x$ nach $y$ überführt, ist nahezu nicht nachvollziehbar, weil er sich als Gewichtsvektor-Zahlen irgendwo im Netz versteckt. Das hat gravierende Auswirkungen auf die Leistung des Netzes, da seine Güte auch stark von der Datenbasis abhängt. Ein Netz, das zur Personaleinstellung geeignete Kandidaten ermitteln soll und hauptsächlich mit Daten weißer männlicher Angestellter trainiert wurde, wird vermutlich weibliche Bewerberinnen mit anderer Hautfarbe benachteiligen. Ähnlich könnte es aussehen, wenn ein neuronales Netz im medizinischen Sektor Diagnosen stellen soll und bestimmte seltene Krankheiten im Trainingsdatensatz unterrepräsentiert sind – möglicherweise werden diese dann später weniger gut diagnostiziert.

Da maschinelles Lernen jedoch bereits breit gefächert eingesetzt wird und noch längst nicht alle Probleme gelöst sind, die diese Technik mit sich bringt, ist Vorsicht geboten. Hier gilt es, sich nicht davon blenden zu lassen, dass hier eine höhere künstliche Intelligenz am Werk sei, die nach menschlichen Maßstäben kreativ und klug ist. Nicht nur beim Zeichnen mit Google Quickdraw können wir feststellen, dass das neuronale Netz manchmal nach menschlichen Maßstäben ziemlich „dumme“ Fehler oder Einschätzungen macht. Auch eine Internetrecherche zu Fehlern von künstlichen neurona-

# Fazit und Ausblick

len Netzen fördert aberwitzige Irrtümer zutage, die ganze Blogs[K] füllen. So wird aus der semantischen Bilderkennung – „Das ist mit 90-prozentiger Wahrscheinlichkeit eine Katze auf dem Foto." – plötzlich zu 85-prozentig ein Auto, wenn ein für das Auge nicht sichtbares Rauschmuster über das Bild gelegt wird, das jedoch auf Pixelebene das neuronale Netz völlig zu verwirren scheint.

Um scheinbar magische Fähigkeiten neuronaler Netze, aber auch solche Irrtümer sowie weitere Beispiele verstehen und richtig deuten zu können, ist hoffentlich im Ansatz diese Reihe hilfreich, die die Welt des maschinellen Lernens ein wenig entmystifizieren sollte.

## Quellen


A www.membrain-nn.de
B Alle Zusatzmaterialien können unter folgendem Link heruntergeladen werden: www.science-on-stage.de/machinelearning
C www.quickdraw.withgoogle.com
D https://de.wikipedia.org/wiki/Siri_(Software)
E https://de.wikipedia.org/wiki/Cortana_(Software)
F Rashid, Tariq (2017), Neuronale Netze selbst programmieren: Ein verständlicher Einstieg mit Python (Animals)
G https://de.wikipedia.org/wiki/Perzeptron
H https://de.wikipedia.org/wiki/Hebbsche_Lernregel
I www.membrain-nn.de
J https://de.wikipedia.org/wiki/Überanpassung
K www.kaspersky.de/blog/ai-fails/14540/


## Weiterführende Hinweise

- Unter www.cs.utexas.edu/~teammco/misc/perceptron kann man ein einfaches neuronales Netz (ein Perzeptron) trainieren. Zwei Klassen (rote und blaue Punkte) lassen sich per Klick auf den Bildschirm erstellen und linear trennen. Ein Testen der Daten ist auch möglich.
- Unter playground.tensorflow.org kann man ein neuronales Netz trainieren. Hier kann die Performance des Netzes untersucht werden, wenn alle möglichen Änderungen am Netz vorgenommen werden. Die Seite bietet eine sehr schöne Visualisierung.
- Unter www.cs.stanford.edu/people/karpathy/convnetjs kann man mehrere neuronale Netze trainieren, u. a. zu den Themen Ziffernerkennung, Bilderkennung und Autoencoder.
- Unter www.simbrain.net gibt es das kostenlose, plattformunabhängige Tool Simbrain, mit dem sich neuronale Netze simulieren lassen. Das Programm bietet viele Möglichkeiten und Visualisierungen und funktioniert für Windows, Mac OS X und Linux.
- Unter www.medien-in-die-schule.de/unterrichtseinheiten/machine-learning-intelligente-maschinen/ finden Sie weitere kostenfreie Unterrichtsmaterialien zum maschinellen Lernen.

K www.kaspersky.de/blog/ai-fails/14540/ (23.09.2019)

VECTOR STIFTUNG

## Die Vector Stiftung: Gemeinsam Richtung Zukunft.

2011 wurde die Vector Stiftung als unternehmensverbundene Stiftung von den Unternehmensgründern der Vector Informatik GmbH Eberhard Hinderer, Martin Litschel und Dr. Helmut Schelling gegründet.

Die Vector Stiftung will den Wirtschafts- und Hochtechnologiestandort Baden-Württemberg mitgestalten und dem Fachkräftemangel in den MINT-Berufen (Mathematik, Informatik, Naturwissenschaften und Technik) entgegenwirken. Sie konzentriert ihr Tun auf die Förderbereiche Forschung, Bildung und Soziales Engagement.

### Forschung

Eine innovationsstarke und wettbewerbsfähige Forschungslandschaft ist der Motor für eine technologiebasierte, erfolgreiche Entwicklung und gesicherte Zukunft des Standortes Baden-Württemberg. Im Mittelpunkt der Forschungsförderung der Vector Stiftung steht die Forschung an den Schnittstellen von Technologie und Umwelt. Die Vector Stiftung fördert Forschungsprojekte in Baden-Württemberg in den Bereichen Innovationen in den MINT-Disziplinen und Klimaschutz im Verkehr.

### Bildung

Ziel der Vector Stiftung im Bildungsbereich ist es, mehr Menschen in technische Berufe zu bringen und damit dem Fachkräftemangel im MINT-Bereich entgegenzuwirken. Dafür setzt die Vector Stiftung früh an und fördert MINT-Interesse bei Schülerinnen und Schülern sowie die MINT-Lehrerbildung.

Um junge Menschen optimal fördern zu können, braucht es gute Lehrkräfte. Die Vector Stiftung fördert MINT-Lehramtsstudierende während ihres Studiums. Die Förderung von Stipendienprogrammen für Lehramtsstudierende und Lehr-Lern-Labore in den MINT-Fächern tragen zu einer praxisnahen Lehramtsausbildung und Unterstützung angehender MINT-Lehrkräfte bei. Außerdem fördert die Stiftung Mentoring für Lehramtsstudierende und MINT-Fachdidaktik im Rahmen von Stiftungsprofessuren.

Die Vector Stiftung unterstützt zudem Lehrkräfte dabei, MINT-Interesse bei Schülerinnen und Schülern hervorzurufen. Fortbildungen für Lehrkräfte, die Entwicklung von Unterrichtsmaterialien in den MINT-Fächern und die Vernetzung engagierter Lehrkräfte tragen zu einer guten MINT-Bildung bei.

Um Begeisterung für MINT zu wecken, fördert die Vector Stiftung schulische Projekte und Arbeitsgemeinschaften im MINT-Bereich. Die Stiftung bietet mit „Mkid – Mathe kann ich doch!“ außerdem ein Programm, das den Interessenverlust an MINT stoppen will. Das Programm richtet sich an Kinder der 6. Klasse, die Potenzial für Mathematik und Naturwissenschaften haben, dieses aber nicht nutzen.

Neben der MINT-Bildungsförderung setzt sich die Vector Stiftung dafür ein, Kindern unabhängig von ihrer Herkunft gute Bildungschancen zu ermöglichen. Gefördert werden Projekte, die sozial benachteiligte Mädchen und Jungen dabei unterstützen, ihre Potenziale zu erkennen und Kompetenzen auszubauen.

### Soziales Engagement

Die Vector Stiftung fördert im Bereich Soziales Engagement Projekte in der Metropolregion Stuttgart, um nachhaltige Veränderungen für sozial benachteiligte Menschen zu erzielen. Dabei konzentriert sich die Stiftung auf die Bekämpfung von Wohnungslosigkeit und die Integration chancenarmer junger Erwachsener in die Gesellschaft.

**Weitere Informationen zur Vector Stiftung finden Sie unter: www.vector-stiftung.de**

SCIENCE ON STAGE DEUTSCHLAND

## Science on Stage Deutschland – The European Network for Science Teachers

… ist ein Netzwerk von Lehrkräften für Lehrkräfte aller Schularten, die Mathematik, Informatik, Naturwissenschaften und Technik (MINT) unterrichten.
… bietet eine Plattform für den europaweiten Austausch anregender Ideen und Konzepte für den Unterricht.
… sorgt dafür, dass MINT im schulischen und öffentlichen Rampenlicht steht.

Science on Stage Deutschland e.V. wird maßgeblich gefördert von think ING., der Initiative für den Ingenieurnachwuchs des Arbeitgeberverbandes GESAMTMETALL.

**Machen Sie mit!**
www.science-on-stage.de
www.facebook.com/scienceonstagedeutschland
www.twitter.com/SonS_D

**Newsletter**
www.science-on-stage.de/newsletter

SCIENCE ON STAGE EUROPE

**Science on Stage Deutschland e.V. ist Mitglied im Netzwerk Science on Stage Europe e.V.**
www.science-on-stage.eu
www.facebook.com/scienceonstageeurope
www.twitter.com/ScienceOnStage

## Weitere Materialien


**Coding im MINT-Unterricht**
Hands-on-Beispiele für den Einstieg ins Programmieren im MINT-Unterricht


**Die perfekte Winterjacke**
Praktische Stationenarbeit zum Thema Wärmelehre


**Teachers + Scientists**
Unterrichtsmaterialien zu aktuellen Forschungsthemen

Die Broschüren stehen zum freien Download zur Verfügung unter
**www.science-on-stage.de/unterrichtsmaterialien**

Ein Projekt von

SCIENCE ON STAGE DEUTSCHLAND

Hauptförderer von
Science on Stage Deutschland

think ING.
Die Initiative für Ingenieurnachwuchs

In Kooperation mit

VECTOR STIFTUNG

www.science-on-stage.de