Künstliche neuronale Netze - ein Teilgebiet der künstlichen Intelligenz

Lennart Venus

Künstliche neuronale Netze - ein Teilgebiet der künstlichen Intelligenz

Schule: Goethe-Gymnasium Sebnitz

Schuljahr: 2021/2022

Verfasser: Lennart Venus

Betreuender Fachlehrer: Herr Wilke

Abgabetermin: 02.03.2022

Inhaltsverzeichnis:

1 Einführung

Diese Arbeit beschäftigt sich mit Grundstrukturen künstlicher neuronaler Netze. Mein Ziel ist es, ein künstliches neuronales Netz zu finden, mit dem möglichst gute Ergebnisse bei der Erkennung von gezeichneten Zahlen und Motiven auf Fotos erzielt werden können. Dieses Netzwerk wird dann in einer von mir entwickelten App getestet.

Das Prinzip künstlicher neuronaler Netze, wie es in der Informatik angewandt wird, hat seinen Ursprung in der Biologie. Neuronale Netze kommen bei Tieren und Menschen in verschiedenen Formen und Größen vor. Die Anzahl der Neuronen kann dabei stark variieren. Während der Fadenwurm *Caenorhabditis elegans* nur ca. 300 Nervenzellen besitzt, geht man davon aus, dass das menschliche Gehirn über etwa 86 Milliarden Neuronen verfügt. Ein neuronales Netz besteht aus einzelnen Nervenzellen, welche durch Dendriten/Axone verbunden sind. Dabei empfangen die Dendriten die Signale und die Axone leiten sie bei Überschreitung eines Schwellenwertes als Aktionspotenzial weiter.
Diese grundlegenden Merkmale findet man auch in den meisten künstlichen neuronalen Netzen wieder. Dabei sind die Nervenzellen oft nur Klassen oder vergleichbare Objekte in der Programmierung. Anders als bei biologischen neuronalen Netzen gibt es hier jedoch verschiedene Arten der Neuronen: Input-Neuronen, Hidden-Neuronen und Output-Neuronen [1]. Auf den Aufbau und die Funktion von diesen wird genauer im nächsten Kapitel eingegangen. Die Neuronen sind auch bei künstlichen neuronalen Netzen untereinander verbunden, jedoch wird das Signal nicht als Aktionspotenzial, sondern als ein Wert übertragen, in der Regel einfach als eine Zahl.
Der genaue Aufbau künstlicher neuronaler Netze bestimmt die Netzwerktopologie, also die Netzwerkart.
Einfache Netzwerkarten, wie das einlagige Perzeptron, wurden schon zwischen 1940 und 1950 entwickelt. Zehn Jahre später wurden die ersten Netze mit einer solchen Topologie teilweise auch für einfache Anwendungen wie die Echtzeit-Echofilterung bei Analogtelefonen kommerziell genutzt. Als 1969 jedoch die damals genutzten Netzwerkarten an ihre Grenzen kamen, wurden viele Forschungsgelder gestrichen und der erste „KI-Winter” begann. Später erholte sich das Forschungsgebiet langsam wieder, wobei es immer wieder Hoch- und Tiefphasen gab.
Ab etwa 1975 wurden dann viele weitere Netzwerkarten entwickelt und die bereits vorhandenen verbessert.

Heute gibt es viele verschiedene Netzwerktopologien, welche meist für einen bestimmten Anwendungsfall optimiert sind. Ich werde darum nur eine engere Auswahl genauer vorstellen. Der Schwerpunkt wird dabei auf ein- und mehrschichtigen Feedforward-Netzen liegen, die ich auch in meiner App nutzen werde. An manchen Stellen führe ich auch Ausschnitte aus dem Programmiercode, in der Programmiersprache Java, an.

Quellen: 1, 2, 3, 4, 9, 10, 11, 26, 27, 16, 23, 74

2 Künstliche neuronale Netzwerke [2,7]

Künstliche neuronale Netzwerke (Zur Vereinfachung wird im Weiteren nur der Begriff „neuronales Netz" verwendet) können zur Umsetzung des tiefen Lernens (Deep Learning) genutzt werden. Deep Learning ist ein Teilgebiet des maschinellen Lernens (Machine Learning), welches wiederum ein Teil der künstlichen Intelligenz (artificial intelligence) ist.
In diesem Kapitel werden die Grundstrukturen und Lernalgorithmen eines Feedforward-Netzes beschrieben.
Dabei gibt es mehrlagige Netze, bestehend aus Input-Neuronen, Hidden-Neuronen und Output-Neuronen, welche untereinander verbunden sind, und einlagige, bei denen die Hidden-Neuronen fehlen.

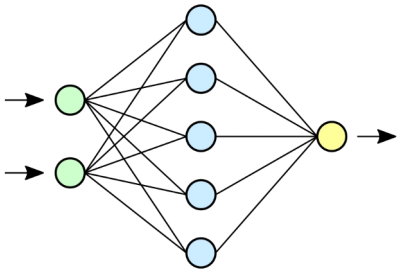

Abbildung 1 (Dake, Mysid, Neural network, CC BY 1.0 [3])

Die in der Abbildung 1 grün gekennzeichneten Input-Neuronen bilden die Input-Schicht (→ „Eingabeschicht"), der die Eingabewerte zugewiesen werden.
Die blau dargestellten Hidden-Neuronen bilden die Hidden-Schicht (→ „versteckte Schicht"). Der Anzahl an Hidden-Schichten sind zwar theoretisch keine Grenzen gesetzt, jedoch steigt der Berechnungsaufwand für den Prozessor mit jeder hinzugefügten Schicht stark an. Dafür kann das Netz mit zusätzlichen Schichten meist bessere Ergebnisse erzielen.
In der Regel gibt die gelb gekennzeichnete Output-Schicht (→ Ausgabeschicht) das wahrscheinlichste Ergebnis für die eingegebenen Werte aus. Das Output-Neuron, welches den höchsten Wert ausgibt, steht dabei häufig für das wahrscheinlichste Ergebnis.
Die einzelnen Schichten sind bei den meisten Arten von Netzwerken immer nur mit der nächsten Schicht verbunden, es gibt also keine sogenannten „Shortcuts". Deshalb wird die Input-Schicht nur mit der vordersten und die Output-Schicht nur mit der hintersten Hidden-Schicht verbunden. Bei Netzen mit insgesamt nur drei Schichten sind Input-Schicht und Output-Schicht mit derselben Hidden-Schicht verbunden und bei einschichtigen Netzen sind Input- und Output-Schicht direkt miteinander verbunden.

Die Pfeile verdeutlichen in der Grafik die eingegebenen und ausgegebenen Werte. Man spricht von einem Feedforward-Netz, weil die Werte immer nur nach vorne gegeben werden.

Quellen: 8, 9, 11, 12, 13, 17, 18, 23, 50, 72

2.1 Bestandteile

2.1.1 (künstliches) Neuron

Ein Neuron in einem neuronalen Netz verarbeitet im Allgemeinen mehrere eingegebene Werte und gibt einen Ergebniswert aus.

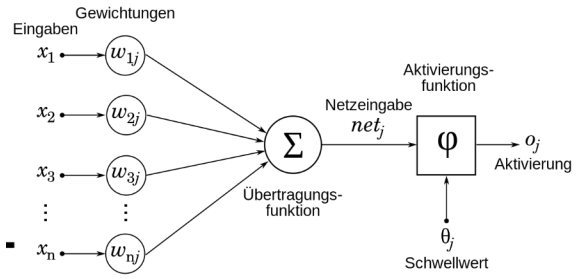

Abbildung 2 (Perhelion, NeuronModel deutsch, CC BY-SA 3.0[4])

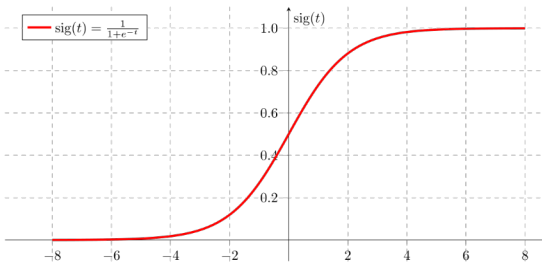

Abbildung 3 (Sigmoid-function-2, CC0 1.0 [5])

Dabei werden zuerst die Werte der verbundenen Neuronen der vorherigen Schicht addiert und danach in eine Aktivierungsfunktion gegeben. Im Fall der Sigmoidfunktion, welche ich in meinem neuronalen Netzwerk nutze, bleibt der ausgegebene Wert immer zwischen 0 und 1 (Abbildung 3). Teilweise muss für eine Weiterleitung des Wertes auch ein Schwellenwert überschritten werden. Die Formel für die Berechnung des Ausgabewertes ist:

$$o_j = f(net_j)$$ [13]

mit

$$net_j = \sum_{i=1}^{n} x_i \cdot w_{ij}$$ [13]

Dabei ist o_j der Ausgabewert; net_j ist die Netzeingabe; f ist die Aktivierungsfunktion; n ist die Anzahl der Eingaben; x_i ist die Eingabe i und w_{ij} ist das Gewicht (weight) zwischen den Neuronen i und j.

Nachfolgend eine mögliche Umsetzung dieser Formel für ein Hidden-Neuron oder ein Output-Neuron:

[21]

```
private float neuronValue = 0;

@Override
public float getValue() {
    float summe = 0;
    for (Connection con : connections) {
        summe += con.getValue();
   }
   neuronValue = activationFunction.activation(summe);

   return neuronValue;
}
```

Bei Input-Neuronen ist dies nicht nötig, da sie keine Werte von anderen Neuronen erhalten und ihre Ausgabe gleich ihrer Eingabe, beziehungsweise ihrem zugewiesenen Wert, ist:

[21]

```
@Override
public float getValue() {
   return value;
}
```

Quellen: 8, 9, 10, 11, 14, 29, 35

2.1.2 Weights (Gewichte)

Die eingegebenen Werte werden mit Gewichten multipliziert, bevor sie vom Neuron verarbeitet werden. Durch deren Anpassung „lernt" ein neuronales Netz. Wenn ein Gewicht null ist, bedeutet es bei den meisten Netzwerkarten, dass die beiden Neuronen nicht verbunden sind. Bevor der Lernvorgang gestartet wird, werden die Gewichte initialisiert.
Ein naheliegender Ansatz wäre, alle Gewichte mit dem Wert null zu initialisieren. Jedoch lernen dann manche Neuronen aufgrund der gleichen Anfangswerte möglicherweise das Gleiche. Für die meisten Anwendungsfälle ist das nicht erstrebenswert, weshalb das Prinzip der Initialisierung mit zufälligen Werten zwischen -1 und +1 bevorzugt wird.
Eine Möglichkeit, die Gewichte eines voll vermaschten Netzwerkes mit einer Hidden-Schicht zu setzen, sind zwei verschachtelte for Schleifen:
[21]

```
int index = 0;

for (WorkingNeuron hidden : hiddens) {
   for (InputNeuron in : inputs) {
       hidden.addConnection(new Connection(in, weights[index++]));
   }
}

for (WorkingNeuron out : outputs) {
   for (WorkingNeuron hidden : hiddens) {
       out.addConnection(new Connection(hidden,
               weights[index++]));
   }
}
```

Bei dieser Implementierung werden die Verbindungen mit den Neuronen der vorherigen Schicht in einem Neuron gespeichert.

Quellen: 9, 15, 73, 74

2.1.3 Input-Schicht (Eingabeschicht)

In die Input-Schicht werden die zu verarbeitenden Werte eingegeben.
Bei Bildern als Eingabewerte wird es oft so gehandhabt, dass jeder Pixel einem Input-Neuron entspricht und die Helligkeit dabei der Eingabewert für dieses Neuron ist. So erhält das neuronale Netzwerk jedoch keine Informationen über die Intensität der einzelnen Grundfarben (Rot, Grün und Blau). Das Bild wird also lediglich in Graustufen erfasst. Bei simplen Anwendungsfällen, wie dem Erkennen von handschriftlich geschriebenen Zahlen reicht das vollkommen aus. Wenn jedoch das Motiv zum Beispiel eines Farbfotos erkannt werden soll, reicht die Verarbeitung in Graustufen für manche Motive möglicherweise nicht aus. Ein Bild in Schwarz-Weiß von einer Landschaft mit blauem Himmel ist von einem Bild, welches die gleiche Landschaft mit einem Sonnenuntergang zeigt, kaum zu unterscheiden (Abbildung 4). Wenn man nur die beiden Schwarz-Weiß-Bilder betrachtet, kann man nicht sagen, welches der beiden den Sonnenuntergang zeigt.

Abbildung 4 [6]

Um dieses Problem zu beheben, werden die Intensitäten von den Grundfarben Rot, Grün und Blau in individuelle Neuronen eingegeben. Dabei müssen die Neuronen für die einzelnen Pixel nicht nebeneinanderliegen. Die einzige Bedingung ist, dass jedes Neuron immer für die gleiche Farbe eines Pixels steht.
Wenn das eingegebene Bild ein Farbbild ist, werden die Eingabewerte deshalb folgendermaßen gesetzt:

[21]

```
for (int x = 0; x < res; x++) {
    for (int y = 0; y < res * 3; y++) {
        inputs[x][y].setValue(currentImage.data[x][y]);
        y++;
        inputs[x][y].setValue(currentImage.data[x][y]);
        y++;
        inputs[x][y].setValue(currentImage.data[x][y]);
    }
}
```

Die Eingabe der Pixelwerte von Bildern in Graustufen ist einfacher :

[21]

```
for (int x = 0; x < res; x++) {
   for (int y = 0; y < res; y++) {
       inputs[x][y].setValue(currentImage.data[x][y]);
   }
}
```

Sprach- und Musikdaten oder Texte sind schwerer zu kodieren. Hier reicht es oft nicht aus, wenn man Zahlenwerte aus einer Audiodatei, die Sprache repräsentieren, den Neuronen zuweist.

Die Qualität der Ergebnisse wird nicht nur von der Kodierung der Daten, sondern auch von der Datenmenge beeinflusst. Eine zu hohe Auflösung bei der Eingabe von Bildern hat viele unnötige Berechnungen zur Folge, wodurch das Netz langsamer lernt. Andererseits können wichtige Details bei einer zu geringen Auflösung nicht erkannt werden. Bei der Erkennung von Gesichtern sollte man zum Beispiel die Auflösung so wählen, dass sich mindestens 64 Pixel zwischen den Augen befinden.

Die den Input-Neuronen zugewiesenen Werte werden dann an die Hidden-Schicht weitergegeben.

Quellen: 9, 19, 20, 21

2.1.4 Hidden-Schicht (versteckte Schicht)

Die versteckte(n) Schicht(en) in einem neuronalen Netzwerk werden gebraucht, um auch komplexere Zusammenhänge verarbeiten zu können. Bei manchen einfachen Anwendungsfällen, wie dem Unterscheiden zwischen einem Kreis und einem Strich, sind versteckte Schichten eher nicht nötig. Sie können sogar zu schlechteren Ergebnissen führen, da das Lernen mit versteckten Schichten einen erhöhten Berechnungsaufwand darstellt. Bevor der Lernprozess gestartet wird, sollte deshalb überlegt werden, wie viel versteckte Schichten wirklich benötigt werden.
Ich habe mich für ein neuronales Netz mit einer Hidden-Schicht entschieden, sowohl für die Erkennung von handschriftlichen Ziffern und Rechenzeichen, als auch für die Erkennung von Motiven auf Farbbildern. Bei der Erkennung von handschriftlichen Ziffern war der Unterschied zwischen keiner und einer Hidden-Schicht nur gering. Die Erkennungsraten mit einer Hidden-Schicht waren trotzdem etwas besser. Deshalb habe ich mich letztendlich dazu entschieden, die Hidden-Schicht mitzubenutzen, auch wenn der zusätzliche Zeitaufwand zur Berechnung des Ergebnisses bei der Echtzeiterkennung bemerkbar war. Bei zwei versteckten Schichten wurde die Erkennungsrate schlagartig schlechter. Die möglichen Gründe werden im Kapitel „Probleme“ genauer erläutert.
Eine mögliche Implementation der Initialisierung der Hidden-Schicht ist folgende:
Zuerst werden die Hidden-Neuronen erstellt:

[21]

```
public void createHiddenNeurons(int a) {
    for (int i = 0; i < a; i++) {
        hiddens.add(new WorkingNeuron());
    }
}
```

Danach wird die erste Hidden-Schicht mit der Input-Schicht verbunden:

[21]

```
int index = 0;

for (WorkingNeuron hidden : hiddens) {
    for (InputNeuron in : inputs) {
        hidden.addConnection(new Connection(in, weights[index++]));
    }
}
```

Weitere Hidden-Schichten würden auch noch untereinander verbunden werden.

Quellen: 9, 18, 73

2.1.5 Output-Schicht (Ausgabeschicht)

Die Neuronen der Output-Schicht geben das Ergebnis des neuronalen Netzwerkes aus. Bei manchen Netzwerkarten, wie den Generative Adversarial Networks, kann das Ergebnis der Output-Schicht auch an ein anderes neuronales Netz weitergegeben werden.
Die Anzahl der Output-Neuronen muss passend zu den möglichen Ergebnissen gewählt werden.
Bei der Erkennung von Ziffern zwischen Null und Neun wäre es zum Beispiel ungünstig, wenn die Output-Schicht nur aus einem Output-Neuron besteht, welches Werte zwischen Null und Neun annehmen kann. Dies würde bedeuten, dass die Ziffern Null und Eins näher beieinanderliegen als die Ziffern Null und Acht. Zwar ist dies mathematisch gesehen der Fall, jedoch unterscheiden sich die grafischen Repräsentationen beider Ziffern stark.
Eine bessere Möglichkeit der Implementation wäre, jeder Ziffer ein eigenes Output-Neuron zuzuordnen. Bei den Ziffern von null bis neun gäbe es also zehn Output-Neuronen. Es gilt die Ziffer als erkannt, deren Neuron den höchsten Wert ausgibt.
Ein weiter Vorteil ist, dass auch die Ziffern abgerufen werden können, die nicht an erster Stelle [8] kommen. Damit können auch komplexere Abfragen getätigt werden, wie zum Beispiel die Wahrscheinlichkeit, mit der das neuronale Netzwerk eine bestimmte Ziffer ausschließen kann.
Um in meiner App das wahrscheinlichste Ergebnis abzurufen, werden die Ausgabewerte der Output-Neuronen absteigend sortiert und gespeichert. Steht das erste Neuron für den richtigen Wert, ist die Ausgabe des Netzes richtig.

[21]

```
ImageProbability[] imageProbabilities = new
ImageProbability[categories.size()];
for (int k = 0; k < imageProbabilities.length; k++) {
   imageProbabilities[k] = new ImageProbability(k,
outputs[k].getValue());
}

Arrays.sort(imageProbabilities, Collections.reverseOrder());

if (i == imageProbabilities[0].category) {
   correct++;
} else {
   incorrect++;
}
```

Quellen: 9, 22, 33, 73

2.2 Lernen

Nach dem Initialisieren eines neuronalen Netzwerkes sind die Ergebnisse zufällig, das Netz „rät" also. Bevor das neuronale Netz produktiv eingesetzt werden kann, muss es deshalb „lernen". Üblicherweise wird das Netz in Epochen trainiert. In jeder Epoche wird die Fehlerquote, also der Anteil an falschen Ergebnissen, tendenziell minimiert. Zwischen zwei Epochen kann die Fehlerquote manchmal aber wieder größer werden. Die Gründe dafür werden später in diesem Kapitel erklärt.

Wie groß die Fehlerquote sein darf, hängt von dem Anwendungsgebiet ab. Bei der Erkennung von Motiven auf Fotos oder von handgeschriebenen Ziffern ist eine Erkennungsrate von 95 % möglicherweise ausreichend. Wenn jedoch beim autonomen Fahren entschieden werden muss, ob gebremst werden soll, wird eine Fehlerquote von 5 % in der Regel zu groß sein.[9]

Für verschiedene Anwendungsfälle gibt es auch verschiedene Lernverfahren.

Hat man zum Beispiel nur ungelabelte Lerndaten, also Daten, wo man das richtige Ergebnis nicht kennt, dann kann man das Verfahren des unüberwachten Lernens (unsupervised learning) nutzen. Dabei gibt man nur das zu lernende Muster ein und das neuronale Netz passt sich diesem automatisch an. Da hier in Echtzeit gelernt wird, nutzt man diese Art des Lernens zum Beispiel, um das Kaufverhalten von Nutzern im Internet zu analysieren.

Wenn zumindest bekannt ist, ob die Ausgabe des neuronalen Netzes gute oder schlechte Auswirkungen, aber keinen zu dem Eingabedatensatz passenden Ausgabedatensatz hat, dann sollte man das bestärkende Lernen (reinforced learning) nutzen. Hier bekommt das neuronale Netz Belohnungen, je nachdem, ob das Ergebnis gut oder schlecht war [10]. Ein Beispiel für die Anwendung dieses Lernverfahrens wäre das Einparken eines Fahrzeuges mittels neuronaler Netze. Das neuronale Netz erhält als Eingabe die umgebenden Objekte und gibt die Fahrtrichtung und ob gebremst oder beschleunigt werden soll, aus. Verursacht das neuronale Netzwerk Kollisionen, dann ist dies in der Regel nicht das gewollte Ergebnis und das Netz erhält eine negative Belohnung. Sobald das Fahrzeug eingeparkt wurde, bekommt das neuronale Netzwerk eine (positive) Belohnung. Nach der Zuweisung der Belohnung werden die Gewichte angepasst.

Hat man aber einen zu dem Eingabedatensatz passenden Ausgabedatensatz, wird das überwachte Lernen (supervised learning) genutzt. Da ich diese Art des Lernens in meiner App einsetze, wird in diesem Kapitel lediglich das überwachte Lernen mittels Backpropagation [11] genauer erläutert.

Nutzt man das überwachte Lernen, kann die Abweichung der Ausgabe (Fehler) des Netzes von dem eigentlich richtigen Ergebnis (Sollwert) direkt berechnet werden.

Der Wert des Fehlers der gesamten Ausgabe wird zum Beispiel folgendermaßen berechnet:

$$E = \frac{1}{n} \sum_{i=1}^{n} (t_i - o_i)^2$$ [12]

Dabei ist E der Fehler; n ist die Anzahl der Ausgabemöglichkeiten; t_i ist der Sollwert von dem Neuron an der Stelle i und o_i ist die eigentliche Ausgabe des Neurons an der Stelle i.
Das Problem dabei ist, dass der Gesamtfehler durch das Quadrieren sehr groß wird, wenn einer der Ausgabewerte sehr von dem Sollwert abweicht.
Eine weitere Möglichkeit ist, den Fehler mittels des Betrages zu berechnen:

$$E = \frac{1}{n} \sum_{i=1}^{n} |t_i - o_i|$$ [12]

Der Vorteil bei der Bildung des Betrages ist, dass eine große Abweichung des Wertes eines Neurons vom Sollwert keinen zu großen Einfluss auf den Gesamtfehler hat.
Wenn der Fehler null ist, hat das neuronale Netz die „perfekte" Ausgabe für die Eingabewerte ausgegeben. Dies ist in der Praxis meist nicht der Fall, da es immer eine gewisse Unsicherheit gibt.

Quellen: 9, 25, 28, 29, 31, 35, 37

2.2.1 Delta-Lernregel und Backpropagation

Eine Möglichkeit, die Gewichte bei einem neuronalen Netz ohne Hidden-Schicht anzupassen und damit den Fehler zu verringern, ist die Delta-Lernregel. Backpropagation, als Verallgemeinerung der Delta-Lernregel, kommt bei Netzwerken mit einer oder mehreren Hidden-Schichten zum Einsatz. Beide Lernregeln sind recht einfach und werden schon lange genutzt.

Die Delta-Lernregel setzt sich aus zwei einzelnen Formeln zusammen:

$$\Delta w_{ij} = \varepsilon \cdot \delta_j \cdot o_i$$ [13]

mit

$$\delta_j = o_j - t_j$$ [13]

Dabei ist $\Delta_{W_{ij}}$ die Gewichtsänderung des Gewichtes zwischen den Neuronen i und j. Hier ist das Neuron j das Output-Neuron.

ε ist die Lernrate mit einem Wert zwischen null und eins, welcher die Stärke der Gewichtsänderung angibt und t_j der Sollwert des Neurons j.

Die Herleitung der Formeln der Delta-Lernregel würde den Rahmen diese Arbeit sprengen.

In meiner App passe ich alle Gewichte eines Neurons folgendermaßen an:

[21]

```
public void deltaLearning(float epsilon, float should) {
    smallDelta = (should - getValue());
    for (Connection connection : connections) {
        float bigDelta = epsilon * smallDelta *
                connection.getNeuron().getValue();
        connection.addWeight(bigDelta);
    }
}
```

Auch wenn ich mit einem Netz ohne Hidden-Schichten, das mit der Delta-Lernregel lernt, bereits gute Ergebnisse erzielen konnte, blieb nach circa 50 Epochen die Erkennungsrate [14] bei durchschnittlich 88 Prozent:

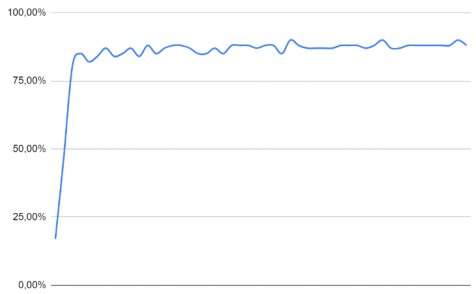

Abbildung 5 [15]

Eine Möglichkeit, die Erkennungsrate zu verbessern, wäre das Hinzufügen einer zusätzlichen Schicht. Leider ist der Sollwert eines Neurons in der Hidden-Schicht unbekannt und damit die Delta-Lernregel für diese nicht anwendbar. Hier kommt das Verfahren der Backpropagation zum Einsatz, wo der Sollwert an die versteckten Schichten „zurückgegeben" wird. Dabei ändert sich nur die Berechnung von δ_j.

δ_j wird für das Output-Neuron j folgendermaßen berechnet:

$$\delta_j = f'(net_j) \cdot (o_j - t_j)$$ [13]

f' ist dabei die Ableitung der Aktivierungsfunktion, welche aufgrund des Kommutativgesetzes in der Mathematik auch ein Faktor bei der Berechnung von $\Delta_{W_{ij}}$ sein kann und dafür bei der Berechnung von δ_j weggelassen wird. Dies kann die Implementation effektiver gestalten.

Für ein Neuron in einer Hidden-Schicht nutzt man zur Berechnung von δ_j folgende Formel:

$$\delta_j = f'(net_j) \cdot \sum_k \delta_k \cdot w_{jk}$$ [13]

Dabei ist das Neuron k das Neuron, welches hinter Neuron j kommt. Bei einem neuronalen Netz mit nur einer Hidden-Schicht wäre das Neuron k, wenn j ein Hidden-Neuron ist, also ein Output-Neuron.

Eine Möglichkeit der Implementation von Backpropagation in Java ist folgende:

Zuerst berechnet man δ_k von der Output-Schicht:

[21]

```
for (int i = 0; i < shoulds.length; i++) {
  outputNeurons.get(i).calculateOutputDelta(shoulds[i]);
}
```

```
public void calculateOutputDelta(float should) {
  smallDelta = (should - getValue());
}
```

Dann gibt man alle δ_k, multipliziert mit dem Gewicht w_{jk}, an die letzte Hidden-Schicht:

[21]

```
if (hiddenNeurons.size() > 0) {
  for (int i = 0; i < outputNeurons.size(); i++) {
    outputNeurons.get(i).backpropagateSmallDelta();
  }
}
```

```
public void backpropagateSmallDelta() {
  for (Connection c : connections) {
    Neuron n = c.getNeuron();
    if (n instanceof WorkingNeuron) {
      WorkingNeuron wn = (WorkingNeuron) n;
      wn.smallDelta += this.smallDelta * c.getWeight();
    }
  }
}
```

Zuletzt berechnet man, ähnlich wie bei einem neuronalen Netz ohne Hidden-Schichten, $\Delta_{W_{ij}}$:

[21]

```
for (int i = 0; i < shoulds.length; i++) {
  outputs.get(i).deltaLearning(epsilon);
}

for (WorkingNeuron neuron : hiddens) {
  neuron.deltaLearning(epsilon);
}
```

```
public void deltaLearning(float epsilon) {
  float derivative  = activationFunction.derivative(getValue());
  for (Connection connection : connections) {
    float bigDelta = epsilon * smallDelta *
        connection.getNeuron().getValue() * derivative;
    connection.addWeight(bigDelta);
  }
}
```

Bei weiteren Hidden-Schichten würde man die letzten Schritte für jede zusätzliche Schicht wiederholen, nur dass Hidden-Neuronen statt Output-Neuronen ihr δ_j weitergeben.

Eine Möglichkeit, sich Backpropagation vorzustellen, wäre folgende:
Wenn man ein einzelnes Gewicht und dessen Auswirkung auf den Fehler des Netzwerkes betrachtet, kann man dies in einem Graphen (Abbildung 6) visualisieren:

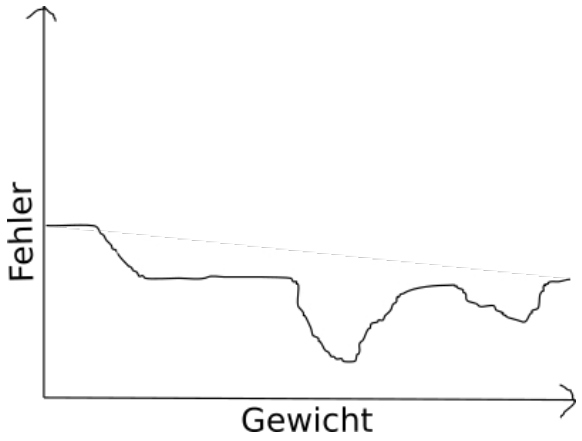

Abbildung 6

Je nachdem, wie man das Gewicht ändert, ändert sich auch der Fehler. Mithilfe der Backpropagation wird der Anstieg des Graphen an der aktuellen Stelle des Gewichtes berechnet, wodurch man die Richtung erhält, in die man das Gewicht ändern muss, um den Fehler zu reduzieren. Je größer dabei der Anstieg des Graphen ist, desto größer ist auch die Gewichtsanpassung.
Ziel ist es, ein möglichst gutes Minimum zu finden, optimalerweise das globale. Dies ist jedoch meist aufgrund der Komplexität der Fehlerfunktion nicht möglich, da man bei einem lokalen Minimum „stecken” bleibt. Dies wird als Gradientenabstiegsverfahren bezeichnet.

Betrachtet man Netzwerke mit mehreren Gewichten, hat die Fehlerfunktion eine höhere Dimensionalität. Abbildung 7 zeigt eine mögliche Fehlerfunktion für zwei Gewichte.

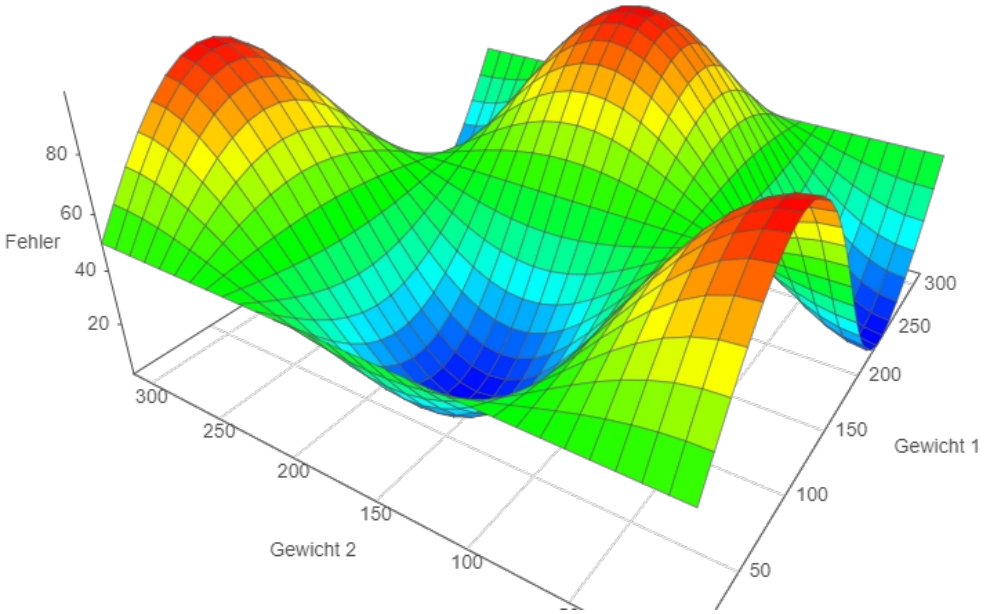

Abbildung 7 [16]

Quellen: 8, 9, 23, 24, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38

2.2.2 Momentum (variabler Trägheitsterm)

In der Praxis haben die Fehlerfunktionen der meisten Gewichte jedoch große „Plateaus". Selbst bei einer großen Änderung des Gewichtes in die richtige Richtung wird der Fehler kaum kleiner.

Um diese „Plateaus" schneller zu überwinden und damit die Erkennungsrate zu verbessern, kann man einen variablen Trägheitsterm, auch Momentum genannt, nutzen.

Dabei werden auch die vorhergegangenen Gewichtsanpassungen in die Berechnung der aktuellen Gewichtsänderung mit einbezogen.

Eine Möglichkeit der Berechnung der Gewichtsänderung mit Momentum wäre folgende:

$$\Delta w_{ij} = \Delta w_{ij} + (\alpha \cdot \Delta w_{ij}^{t-1})$$ [17]

Dabei ist $\Delta_{W_{ij}}$ die, mittels Backpropagation berechnete, Gewichtsänderung zwischen den Neuronen i und j. Auf $\Delta_{W_{ij}}$ wird die Gewichtsänderung von der vorherigen Gewichtsanpassung Δw_{ij}^{t-1} addiert, nachdem diese mit dem Faktor α multipliziert wurde.

α hat dabei oft einen Wert um 0.9 und verhindert, dass das Momentum (Δw_{ij}^{t-1}) zu groß wird.

Bei einer schwankenden Gewichtsänderung (manchmal positiv und manchmal negativ), hat das Momentum einen Wert um null. Wenn die Gewichtsänderung immer positiv ist, ist das Momentum groß, bei einer negativen Gewichtsänderung ist das Momentum klein beziehungsweise negativ.

Die Nutzung von Momentum beschleunigt das Lernen und führt meist zu besseren Ergebnissen.

Dies sieht man auch in den Erkennungsraten von Motiven in meiner App. Nach ca. 50 Epochen blieb die Erkennungsrate bei durchschnittlich 92 %. Ohne Momentum lag diese bei 88 %.

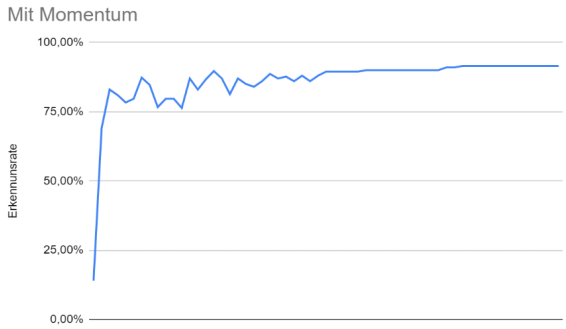

Abbildung 8 [18]

In der App habe ich das Momentum folgendermaßen implementiert:

[21]

```
public void addWeight(float weightDelta) {
 weightDelta += momentum * alpha;
 weight += weightDelta;
 momentum = weightDelta;
}
```

Quellen: 29, 35, 39, 40

2.2.3 Batch-Learning

Falls selbst die Nutzung von Momentum die Ergebnisse nicht ausreichend verbessert, kann, anstelle des bis jetzt vorgestellten Online-Learning, auch Batch-Learning genutzt werden.
Dabei werden die Gewichte nicht nach jedem eingegebenen Bild angepasst, sondern es werden mehrere Bilder eingegeben, wobei sich das neuronale Netz für jedes Bild die eigentlichen Gewichtsanpassungen abspeichert. Danach werden diese aufeinander addiert und die Gewichte werden erst mit den neuen Gewichtsanpassungen angepasst.
Eine mögliche Implementierung sieht so aus:
[21]

```
public void addWeight(float weightDelta) {
  weightAdd += weightDelta;
}
```

In diese Methode wird, genau wie vorher bei der Backpropagation, die Gewichtsanpassungen der einzelnen Gewichte gegeben.
Die richtige Gewichtsanpassung erfolgt jedoch erst später, in diesem Fall aller sieben Bilder:
[21]

```
if (trainingSample % 7 == 0) {
  for (WorkingNeuron output : outputs) {
    output.applyBatch();
  }
  for (WorkingNeuron working : hiddenNeuronsTwo) {
    working.applyBatch();
  }
  for (WorkingNeuron hidden : hiddens) {
    hidden.applyBatch();
  }
}

trainingSample++;
```

```
public void applyBatch() {
  weightAdd += momentum * alpha;
  momentum = weightAdd;
  weight += weightAdd;
  weightAdd = 0;
}
```

Quellen: 24, 33, 42

2.3 Probleme

2.3.1 Vanishing Gradient Problem

Nutzt man tiefe neuronale Netze, also neuronale Netze bestehend aus mehreren Hidden-Schichten, mit bestimmten Aktivierungsfunktionen, wie zum Beispiel der Sigmoidfunktion, kann es passieren, dass das neuronale Netzwerk schlechtere Ergebnisse erzielt. Dies kann an dem Problem mit dem verschwindenden Gradienten (vanishing gradient problem) liegen. Dabei werden die Gewichte der vorderen Schichten nur sehr geringfügig bis gar nicht angepasst.

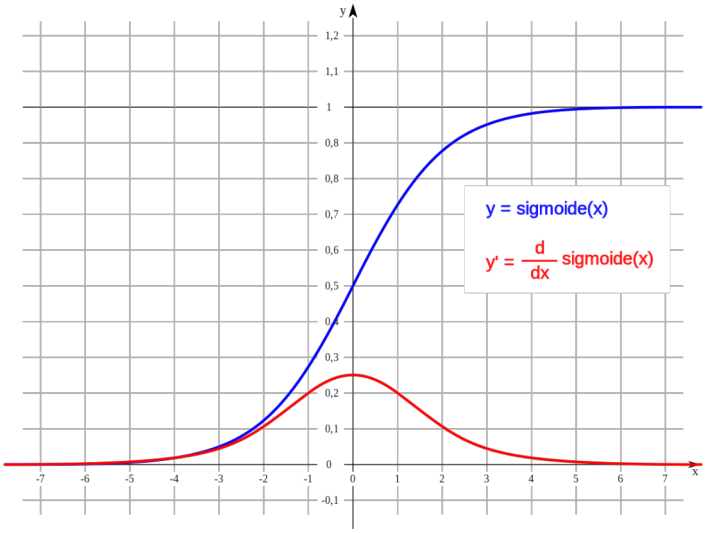

Abbildung 9 (Dnu72, Función sigmoide 02, CC BY-SA 3.0 [19])

Dies liegt daran, dass f'(net$_j$) (die rote Funktion in Abbildung 9) bei größeren Werten gegen null geht und damit auch δ_j sehr klein wird. Da δ_j auch für die Gewichtsänderungen der davorliegenden Schichten gebraucht wird, wird δ_j, und damit auch die gesamte Gewichtsänderung, immer kleiner, je weiter vorn ein Neuron im neuronalen Netz liegt.
Bei neuronalen Netzen mit wenigen Hidden-Schichten ist das weniger häufig ein Problem, da die Gewichtsänderungen immer noch groß genug sind und der Lernprozess einfach nur etwas länger dauert. Wie viele Hidden-Schichten mit Funktionen wie der Sigmoidfunktion ohne größere Probleme genutzt werden können, kommt dabei immer auf den Anwendungsfall an.
Das Nutzen von Funktionen wie der ReLu-Funktion (ReLu steht für Rectified Linear Unit) als Aktivierungsfunktion kann dieses Problem beheben. Die ReLu-Funktion ist im negativen Be-

reich 0 und im positiven mit f(x)=x identisch. Damit ist f'(net_j) für alle positiven Werte 1 und für alle negativen 0.
In meiner App hat ein neuronales Netz mit der ReLu-Funktion als Aktivierungsfunktion für die Neuronen der Hidden-Schicht etwas bessere Ergebnisse erzielt als mit der Sigmoidfunktion.

Quellen: 8, 9, 14, 41

2.3.2 Probleme durch Trainingsdaten

Auch durch Trainingsdaten können Probleme entstehen. Es kann zum Beispiel sein, dass die Trainingsdaten ein bestimmtes Muster aufweisen, welches jedoch in der “echten Welt” nicht vorhanden ist. Ein Beispiel dafür ist, wenn man alle Bilder einer Kategorie bei der Erkennung von Motiven an einem sonnigen Tag aufgenommen hat. Auch wenn das neuronale Netz bei der Erkennung dieser Bilder gute Ergebnisse erzielt, kann es Bilder, die bei schlechtem Wetter aufgenommen wurden, möglicherweise schlechter erkennen.
Bei der Gesichtserkennung ist dies teilweise auch erkennbar. Viele Datenbanken, mit denen die Programme trainiert werden, führen mehr Gesichter von hellhäutigen Personen als von dunkelhäutigen auf. Dadurch werden dunkelhäutige Personen oft schlechter erkannt.
Zu viele Hidden-Schichten können dieses Problem auch vergrößern, da sie mehr „falsche” Muster in den Trainingsdaten finden als Netzwerke mit weniger Hidden-Schichten. So könnte ein Netz, welches Katzen identifizieren soll, das Halsband als Teil der Katze erkennen und somit eine Katze ohne Halsband nicht als Katze erkennen. Einem Netz mit weniger Hidden-Schichten würde das Halsband wahrscheinlich gar nicht „auffallen”.
Solche Probleme lassen sich einfach durch ein größeres und diverseres Trainingsset lösen. Trotzdem sollte man sich genau überlegen, wie viele Hidden-Schichten wirklich nötig sind.

Quellen: 9, 18, 43, 64

3 Weitere Arten künstlicher neuronaler Netze

Manche dieser Probleme lassen sich auch durch die Nutzung anderer Netzwerktopologien lösen. Dabei gibt es bis jetzt keine „beste” Netzwerkart, die alle Probleme auf einmal löst.
Für viele Anwendungsgebiete gibt es Netzwerktopologien, die speziell für dieses optimiert wurden.
Manche Netzwerkarten, wie gepulste neuronale Netze, welche zusätzlich den Aspekt der Zeit integrieren, orientieren sich mehr an biologischen neuronalen Netzen, andere wiederum orientieren sich eher an der Mathematik. Es gibt neuronale Netze, die, zum Beispiel durch das Hinzufügen einzelner Neuronen, „wachsen” können. Beispiele für solche Netze sind das Growing Neural Gas oder Cascade-Correlation-Netze. Wieder andere Netze nutzen spezielle Lernverfahren.
In den nächsten Kapiteln werden zwei weitere Arten neuronaler Netze genauer vorgestellt.

Quellen: 6, 7, 50

3.1 Rekurrente neuronale Netze

Rekurrente neuronale Netze nutzt man, wenn die Eingabedaten in einer Sequenz vorliegen. Eines der besten Beispiele dafür ist die Übersetzung eines Textes in eine andere Sprache. Dabei hängt die Bedeutung eines Wortes unter anderen von den Worten ab, welche davor stehen. Aber auch bei vielen anderen Anwendungen, wo Sprache eine Rolle spielt, wie Spracherkennung oder Sprachsynthese, kommen rekurrente neuronale Netzwerke zum Einsatz.
Klassische Feedforward-Netzwerke können solche Aufgaben nicht zufriedenstellend lösen.
Rekurrente Netze schaffen dies, da dort der Output zum vorherigen Zeitpunkt, zusätzlich zum aktuellen Input, in das Netz zum aktuellen Zeitpunkt eingeben wird. Somit kann ein neuronales Netz mit einem „Gedächtnis” ausgestattet werden.
Da rekurrente neuronale Netzwerke im Gegensatz zu Feedforward-Netzen wesentlich schwerer zu trainieren sind, wird der Lernprozess hier nicht erklärt. Ein weiterer Nachteil dieser Netze ist, dass sie längere Datensequenzen mit bestimmten Aktivierungsfunktionen, wie der ReLu oder Tangens hyperbolicus, nicht richtig verarbeiten können. Außerdem haben rekurrente neuronale Netze auch Probleme mit dem verschwindendem Gradienten, da der Lernalgorithmus auf der Backpropagation und dem Gradientenabstiegsverfahren basiert.
Da bei der Bilderkennung die aktuelle Ausgabe jedoch nicht von den Vorherigen abhängt, habe ich keine rekurrente Netzwerkstruktur genutzt und führe hier keinen Programmiercode auf.

Quellen: 9, 11, 17, 44, 45, 47, 48, 49

3.2 Deep Belief Networks

Einen Lösungsansatz für das Problem mit dem verschwindendem Gradienten bieten „Deep Belief Networks”. Diese Netzwerkart erzielt im Vergleich zu den meisten Feedforward-Netzen auch bei weniger Trainingsdaten bessere Ergebnisse.
Sie hat zwar den gleichen Aufbau wie ein normales Feedforward-Netz, wird jedoch mit einem anderen Lernalgorithmus trainiert.
Eine Restricted Boltzmann Maschine, also ein Deep Belief Network mit zwei Schichten, trainiert man mit dem Contrastive Divergence Algorithmus. Dabei wird die Eingabe-Schicht als die sichtbare Einheit bezeichnet und die zweite Schicht als versteckte Einheit.
Zuerst werden die Daten der sichtbaren Einheit, ähnlich wie bei einem Feedforward-Netz, welches mit Backpropagation trainiert wird, an die versteckte Einheit gegeben (Forward Pass). Danach werden die Werte der versteckten Einheit, multipliziert mit den gleichen Gewichten wie bei dem Forward Pass, an die sichtbare Einheit gegeben (Backward Pass). Dort werden die Eingabe-Werte mit den beim Backward Pass erhaltenen Werten verglichen und die Gewichte so angepasst, dass die beiden Werte für jedes Neuron möglichst wenig voneinander abweichen. Die Gewichtsänderungen werden mit der Delta-Lernregel berechnet, wobei die Eingabe-Werte die Sollwerte und die beim Backward Pass erhaltenen Werte die Ist-Werte sind.
Um ein tieferes Netz zu trainieren, kann man mehrere Restricted Boltzmann Maschinen miteinander verbinden, wobei die versteckte Einheit der vorderen Restricted Boltzmann Maschine immer die sichtbare Einheit der nächsten bildet.
Lernt das Netz auf diese Art, müssen die Lerndaten zwar nicht unbedingt gelabelt sein, um die Gewichte anzupassen, jedoch weiß man dann nicht, welches Output-Neuron für welche Ausgabe steht, da das neuronale Netz „selbst entscheidet”, welche Aspekte der Input-Daten wichtig sind. Bei manchen Anwendungen ist das auch nicht nötig. Wenn jedoch Motive erkannt werden sollen, muss man wissen, welches Motiv erkannt wurde. Das erreicht man, indem man mit ein paar wenigen gelabelten Lerndaten eine Feinabstimmung der Gewichte vornimmt und damit letztendlich zuordnet, welche Ausgaben zu welchen Motiven gehören. Die Anzahl der gelabelten Lerndaten kann jedoch im Vergleich zu der Anzahl der Ungelabelten sehr gering sein, was den Aufwand beim Sammeln der Daten senkt. Trotzdem erzielte ich mit dieser Netzwerktopologie keine Verbesserungen bei den Erkennungsraten.

Quellen: 51, 52, 53, 54, 55, 56, 57, 58

4 Anwendung

Künstliche neuronale Netze, egal welcher Art, können überall eingesetzt werden. Bekannte Anwendungen sind unter anderen die Gesichts-, Objekt- und Spracherkennung oder Bots in Computerspielen. Dies stellt jedoch nur einen sehr kleinen Teil von den eigentlichen Einsatzgebieten neuronaler Netze dar. Sie werden aber auch bei Frühwarnsystemen, medizinischer Diagnostik oder verschiedenen Algorithmen im Internet genutzt. Davon bekommen nur die wenigsten Leute etwas mit. Google zum Beispiel nutzt tiefe neuronale Netze, unter anderem bei Gmail, für die Filterung von Spam, oder bei Google Translate für eine „natürlichere" Übersetzung.

Bei komplexeren Anwendungen, wie selbstfahrenden Autos, werden mehrere neuronale Netze genutzt. Die einen zur Erkennung von Objekten, wie Ampeln, Fußgänger oder anderen Gegenständen im Straßenverkehr. Die erkannten Objekte können dann in ein anderes neuronales Netz gegeben werden, das dann entscheidet, welches Manöver das Fahrzeug ausführen muss. Dieses kann dann auch unabhängig vom Ersten mit Daten einer Simulation trainiert werden, womit es wesentlich mehr Erfahrung sammeln kann als mit Daten aus der „echten" Welt.

Auch in vielen Apps werden künstliche neuronale Netze genutzt. Vor allem bei populäreren Social-Media-Apps wird über die Konsequenzen des Einsatzes von neuronalen Netzen stark diskutiert. Das Problem ist, dass versucht wird vorauszusagen, welche Informationen dem Benutzer gefallen könnten. Dadurch erhält der Nutzer möglicherweise nur noch Informationen, die dem eigenen Standpunkt entsprechen.

Bei vielen anderen Apps ist der Einsatz von neuronalen Netzen weniger problematisch. Dies können Apps zur Erkennung von Pflanzen, wie die App „PlantNet", oder zum Durchsuchen eigener Bilder nach bestimmten Motiven sein.

Quellen: 9, 46, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68

4.1 App: „MyNet”

In meiner App „MyNet” nutze ich neuronale Netze, um verschiedene Motive auf Bildern und handgeschriebene Ziffern von 0 bis 9 zu erkennen [20]. Mein Ziel war es dabei möglichst hohe Erkennungsraten durch Verbesserungen des neuronalen Netzes zu erreichen.
Angefangen habe ich mit einem neuronalen Netz ohne Hidden-Schichten, Momentum oder Batch-Learning und Schwarz-Weiß-Bildern. Dabei blieben die Erkennungsraten bei der Motiverkennung meist unter 50 %. Um sie zu verbessern, wechselte ich, wie im Kapitel „Input-Schicht (Eingabeschicht)” beschrieben, zu Farbbildern. Dadurch stiegen die Erkennungsraten durchschnittlich auf bis zu 90 %. Eine weitere Verbesserung konnte ich durch die Nutzung von Momentum erzielen. Bei Testdaten erhielt ich damit teilweise sogar Erkennungsraten von 100 %. Batch-Learning und Hidden-Schichten konnten nur bei der Erkennung von Ziffern zwischen 0 und 9 weitere Verbesserungen erzielen. Insgesamt waren die Erkennungsraten dort jedoch immer circa fünf Prozentpunkte geringer. Auch andere Netzwerktopologien verbesserten die Ergebnisse nicht.
Um beim Lernen Parameter wie Anzahl der einzelnen Neuronen in jeder Schicht, die Lernrate oder die zu lernenden Motive schnell anpassen zu können, werden diese in einer Klasse aufgelistet:

[21]

```
ArrayList<String> categories = new ArrayList<>();
categories.add("Forest");
categories.add("Landscape");
categories.add("Document");
categories.add("Fire");
categories.add("Sunset");
categories.add("Window");
categories.add("Winter");
int res = 21;
int numHidden = 0;
int numHiddenTwo = 0;
boolean isColor = false;
float epsilon = 0.05f;
float epsilonFactor = 0.99f;
```

Die Parameter werden dann an eine andere Klasse gegeben, welche den eigentlichen Lernprozess steuert, indem sie das neuronale Netz erst nach den gegebenen Vorgaben initiali-

sieren lässt, dann die Lerndaten der nächsten Klasse im richtigen Format übergibt und den Lernfortschritt testet.
Die nächste Klasse speichert, welche Neuronen untereinander verbunden sind und lässt die Neuronen der Output- und Hidden-Schicht(en) die Delta-Lernregel beziehungsweise Backpropagation ausführen. Der Code dafür wurde schon teilweise in den vorherigen Kapiteln angeführt. Außerdem setzt diese Klasse das neuronale Netz zurück, wenn zwischen den verschiedenen Funktionen der App gewechselt wird (zum Beispiel zwischen „Taschenrechner" und „Motiverkennung"). Dadurch entsteht auch die Verzögerung beim Laden.
Ich habe das neuronale Netz auf einem Laptop trainiert, da die Rechenleistung dort höher als auf einem Handy ist. Danach wurden die Gewichte als Datei in die App eingebunden, wo der Lernfortschritt besser eingeschätzt werden kann.

.apk-Datei für Android-Geräte: https://leancoding.co/0IXPYQ (Link gekürzt)

Quellcode der Version für PCs/Laptops: https://leancoding.co/ZUSBO6 (Link gekürzt)

5 Zusammenfassung und Prognose

Von den vorgestellten Netzwerktopologien hat sich ein einfaches neuronales Netz mit bis zu einer Hidden-Schicht, welches nur mit Backpropagation und Momentum arbeitet, als am effektivsten erwiesen. Dieser Typ konnte auch in meiner App sehr gute Ergebnisse erzielen. Andere kompliziertere Topologien erbrachten für meinen Anwendungsfall weniger gute Erkennungsraten. Sie jedoch werden in anderen, komplexeren, Anwendungsgebieten bessere Ergebnisse erzielen können.

Wie sich die Technologie der neuronalen Netze weiterentwickeln wird, ist schwer zu sagen, da es sehr viele Arten gibt, welche sich auch stark voneinander unterscheiden.
Schaut man zurück in die Vergangenheit, kann man erkennen, dass mit steigender Rechenleistung der Computer auch immer neue Netzwerkarten entwickelt wurden, die diese höhere Rechenleistung benötigen. Mittlerweile gibt es sogar Prozessoren, die auf ein möglichst effizientes Training von neuronalen Netzen spezialisiert sind. Diese Entwicklung wird sich wahrscheinlich auch weiter fortsetzen.
Im Allgemeinen lässt sich sagen, dass neuronale Netze in Zukunft vermehrt eingesetzt und verbessert werden, egal ob in der Industrie oder im alltäglichen Leben.
Meiner Meinung nach hat diese Technologie, obwohl sie schon lange genutzt wurde, immer noch viel Zukunftspotenzial.

Quellen: 9, 69, 70, 71

6 Ergänzungen

[1]: Die entsprechenden deutschen Begriffe dafür sind „Eingabeneuronen”, „versteckte Neuronen” und „Ausgabeneuronen". Hier, aber auch bei den meisten anderen Wörtern, werden jedoch in der Regel eher die englischen Begriffe aufgrund ihrer häufigeren Verwendung (zum Beispiel im Internet) verwendet.

[2]: Die Begriffe „(künstliches) neuronales Netz” und „(künstliches) neuronales Netzwerk” werden hier synonym verwendet.

[3]: Dake, Mysid (https://commons.wikimedia.org/wiki/File:Neural_network.svg), „Neural network“, https://creativecommons.org/licenses/by/1.0/legalcode

[4]: Perhelion (https://commons.wikimedia.org/wiki/File:NeuronModel_deutsch.svg), „NeuronModel deutsch“, https://creativecommons.org/licenses/by-sa/3.0/legalcode

[5]: MartinThoma (https://commons.wikimedia.org/wiki/File:Sigmoid-function-2.svg), „Sigmoid-function-2“, https://creativecommons.org/publicdomain/zero/1.0/legalcode

[6]: Beide Bilder sind aus meinem selbst erstellten Datensatz für die Erkennung von verschiedenen Motiven entnommen, welchen ich in meiner App nutze.

[7]: Dieses Kapitel bezieht sich vor allem auf feedforward-Netze. Rekurrente Netze werden im dritten Kapitel genauer erläutert.

[8]: Kommt eine Ausgabe nicht „an erster Stelle”, heißt das die Ausgabe wird vom neuronalen Netz als weniger wahrscheinlich betrachtet.
Ein Beispiel dafür wäre, wenn man die Ziffer 6 als Bild eingibt, dann sollte die Ausgabe für die Ziffer 6 optimalerweise an erster Stelle kommen. An zweiter Stelle könnte dann eine Ziffer wie 0 oder 8 stehen, da diese der 6 grafisch gesehen sehr ähnlich sind.

[9]: Bei einer Erkennungsrate von 95 % ist die Fehlerquote 5 %.

[10]: Belohnungen sind eine Art der Rückmeldung an ein neuronales Netz. Sie können sowohl negativ als auch positiv sein.

[11]: Backpropagation ist ein Verfahren, um neuronale Netze mit überwachtem Lernen zu trainieren.

[12]: Grafik von https://www.zahlen-kern.de/editor/; Formel von https://www.theaidream.com/post/loss-functions-in-neural-networks

[13]: Grafik von https://www.zahlen-kern.de/editor/; Formel von https://de.wikipedia.org/wiki/Backpropagation

[14]: Hier ist der Anteil der richtig erkannten Motive auf Bildern gemeint. Die Motive sind Wald, Landschaft, Dokument, Feuer, Winter, Sonnenuntergang und Fenster.

[15]: Grafik mit https://docs.google.com/spreadsheets erstellt; y-Achse: Erkennungsrate; x-Achse: Epochen (0-52)

[16]: Grafik mit https://almende.github.io/chap-links-library/js/graph3d/playground/ erstellt

[17]: Grafik von https://www.zahlen-kern.de/editor/; Formel in leicht abgewandelter Form von https://visualstudiomagazine.com/Articles/2017/08/01/Neural-Network-Momentum.aspx

[18]: Grafik mit https://docs.google.com/spreadsheets erstellt; y-Achse: Erkennungsrate (durchschnitt); x-Achse: Epochen (0-60)

[19]: Dnu72 (https://commons.wikimedia.org/wiki/File:Función_sigmoide_02.svg), „Función sigmoide 02“, https://creativecommons.org/licenses/by-sa/3.0/legalcode

[20]: Der Erkennung von den Ziffern wurde die „Taschenrechner”-Funktion hinzugefügt, um die Nutzung abwechslungsreicher zu gestalten.

[21]: Teile des Programmiercodes wurden von https://pastebin.com/5pKQ1PcE, https://www.youtube.com/playlist?list=PLgomWLYGNl1dL1Qsmgumhcg4HOcWZMd3k, https://www.youtube.com/watch?v=YIqYBxpv53A&list=PL58qjcU5nk8sUq97ze6MZB8aHflF0uNgK und https://dzone.com/articles/designing-a-neural-network-in-java übernommen.

Literatur- und Quellenverzeichnis

Textquellen

1: Wikipedia-Autoren. (2002, 8. Februar). *Neuronales Netz*. Wikipedia. Abgerufen am 2. Februar 2022, von https://de.wikipedia.org/wiki/Neuronales_Netz

2: Wikipedia-Autoren. (2003, 23. Dezember). *Caenorhabditis elegans*. Wikipedia. Abgerufen am 2. Februar 2022, von https://de.wikipedia.org/wiki/Caenorhabditis_elegans

3: J.H. (2020, 6. August). *Wie viele Neuronen enthält das menschliche Gehirn?* Richard Dawkins Foundation für Vernunft & Wissenschaft. Abgerufen am 2. Februar 2022, von https://de.richarddawkins.net/articles/wie-viele-neuronen-enthaelt-das-menschliche-gehirn

4: Peichl, P. D. L. (2015, 16. April). *Wie viele Nervenzellen hat das Gehirn?* Helmholtz-Gemeinschaft Deutscher Forschungszentren. Abgerufen am 2. Februar 2022, von https://www.helmholtz.de/newsroom/artikel/wie-viele-nervenzellen-hat-das-gehirn/

5: Wikipedia-Autoren. (2002b, November 27). *Nervenzelle*. Wikipedia. Abgerufen am 2. Februar 2022, von https://de.wikipedia.org/wiki/Nervenzelle

6: Wikipedia contributors. (2022, 20. Januar). *Spiking neural network*. Wikipedia. Abgerufen am 2. Februar 2022, von https://en.wikipedia.org/wiki/Spiking_neural_network

7: Wikipedia-Autoren. (2015, 11. Mai). *Gepulste neuronale Netze*. Wikipedia. Abgerufen am 2. Februar 2022, von https://de.wikipedia.org/wiki/Gepulste_neuronale_Netze

8: *Lizenzhinweisgenerator*. (o. D.). Lizenzhinweisgenerator. Abgerufen am 9. Februar 2022, von https://lizenzhinweisgenerator.de/

9: Wikipedia-Autoren. (2003a, April 11). *Künstliches neuronales Netz*. Wikipedia. Abgerufen am 3. Februar 2022, von https://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz

10: Wikipedia-Autoren. (2005, 23. Juli). *Künstliches Neuron*. Wikipedia. Abgerufen am 3. Februar 2022, von https://de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron

11: Heuler, M. (1997, 7. Januar). *Neuronale Netze*. Neuronale Netze. Abgerufen am 3. Februar 2022, von http://duechs.com/stud/pendel/html_netz/node7.html

12: Ceron, R. (2021, 8. März). *AI, machine learning and deep learning: What's the difference?* Servers & Storage. Abgerufen am 3. Februar 2022, von https://www.ibm.com/blogs/systems/ai-machine-learning-and-deep-learning-whats-the-difference/

13: Wikipedia-Autoren. (2004, 6. März). *Maschinelles Lernen*. Wikipedia. Abgerufen am 3. Februar 2022, von https://de.wikipedia.org/wiki/Maschinelles_Lernen

14: Wikipedia-Autoren. (2006, 18. Januar). *Sigmoidfunktion*. Wikipedia. Abgerufen am 3. Februar 2022, von https://de.wikipedia.org/wiki/Sigmoidfunktion

15: Goyal, C. (2021, 6. Juli). *Weight Initialization Techniques in Neural Networks*. Analytics Vidhya. Abgerufen am 4. Februar 2022, von https://www.analyticsvidhya.com/blog/2021/05/how-to-initialize-weights-in-neural-networks/

16: Wikipedia-Autoren. (2004a, Februar 28). *Perzeptron*. Wikipedia. Abgerufen am 3. Februar 2022, von https://de.wikipedia.org/wiki/Perzeptron

17: Wikipedia contributors. (2022b, Januar 30). *Feedforward neural network*. Wikipedia. Abgerufen am 3. Februar 2022, von https://en.wikipedia.org/wiki/Feedforward_neural_network

18: Gad, A. (2018, 27. Juni). *Beginners Ask "How Many Hidden Layers/Neurons to Use in Artificial Neural Networks?"*. Medium. Abgerufen am 5. Februar 2022, von https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e

19: Danis, N. (o. D.). *Kairos: Best Practices for Developing with Face Recognition*. Kairos. Abgerufen am 5. Februar 2022, von https://www.kairos.com/docs/api/best-practices

20: *Neuronale Netze - Eine Einführung - Netzaufbau*. (o. D.). Neuronale Netze Eine Einführung. Abgerufen am 3. Februar 2022, von http://www.neuronalesnetz.de/farbkonstanz2.html

21: T.M., R.W., A.C. & M.P. (2013). *Influence of low resolution of images on reliability of face detection and recognition*. https://link.springer.com/content/pdf/10.1007/s11042-013-1568-8.pdf

22: Wikipedia-Autoren. (2017, 23. Juli). *Generative Adversarial Networks*. Wikipedia. Abgerufen am 7. Februar 2022, von https://de.wikipedia.org/wiki/Generative_Adversarial_Networks

23: *Neuronale Netze – Sebastian Dörn*. (o. D.). Sebastian Dörn. Abgerufen am 3. Februar 2022, von https://sebastiandoern.de/neuronale-netze/

24: S.H. (2018). *Stochastisches Lernen*. https://www.uni-muenster.de/AMM/num/Vorlesungen/Seminar_Wirth_Master_WS18_b/handouts/Hein.pdf

25: Wikipedia-Autoren. (2004b, März 4). *Bestärkendes Lernen*. Wikipedia. Abgerufen am 6. Februar 2022, von https://de.wikipedia.org/wiki/Best%C3%A4rkendes_Lernen

26: Wikipedia-Autoren. (2019, 30. Mai). *KI-Winter*. Wikipedia. Abgerufen am 2. Februar 2022, von https://de.wikipedia.org/wiki/KI-Winter

27: Redaktion, D. (2021, 10. November). *KI-Geschichte: Vom „KI Winter“ zum endgültigen Durchbruch*. brutkasten. Abgerufen am 2. Februar 2022, von https://brutkasten.com/ki-geschichte-venionaire/

28: datasolut GmbH. (2020, 19. September). *Unsupervised Learning: Definition, Arten & Beispiele - datasolut Wiki*. Abgerufen am 6. Februar 2022, von https://datasolut.com/wiki/unsupervised-learning/

29: Wikipedia-Autoren. (2004a, Februar 28). *Backpropagation*. Wikipedia. Abgerufen am 6. Februar 2022, von https://de.wikipedia.org/wiki/Backpropagation

30: Wikipedia-Autoren. (2004e, Juni 30). *Gradientenverfahren*. Wikipedia. Abgerufen am 6. Februar 2022, von https://de.wikipedia.org/wiki/Gradientenverfahren

31: N.S.C. (2021, 20. September). *Loss Functions in Neural Networks*. The AI Dream. Abgerufen am 6. Februar 2022, von https://www.theaidream.com/post/loss-functions-in-neural-networks

32: *Delta-Regel*. (o. D.). Hochschule für Technik und Wirtschaft Dresden. Abgerufen am 6. Februar 2022, von https://www2.htw-dresden.de/%7Eboehme/Neuroinformatik/GNI_Prakt_TUI/ni_grundlagen_prak/deltalearning/deltalearning.html

33: Brotcrunsher. (2017, 16. Februar). *Neuronale Netze* [Video]. YouTube. https://www.youtube.com/playlist?list=PL58qjcU5nk8sUq97ze6MZB8aHflF0uNgK

34: *Neuronale Netze - Eine Einführung - Delta-Regel*. (o. D.). Neuronale Netze Eine Einführung. Abgerufen am 3. Februar 2022, von http://www.neuronalesnetz.de/delta.html

35: *LaTeX-Formeleditor*. (o. D.). LaTeX-Formeleditor. Abgerufen am 7. Februar 2022, von https://www.zahlen-kern.de/editor/

36: *Backpropagation – Biologie*. (o. D.). Biologie - Die Wissenschaft vom Leben. Abgerufen am 6. Februar 2022, von https://www.biologie-seite.de/Biologie/Backpropagation

37: Reynolds, M. (2021, 24. Dezember). *Backpropagation: Intuition and Explanation - Towards Data Science*. Medium. Abgerufen am 6. Februar 2022, von https://towardsdatascience.com/backpropagation-intuition-and-derivation-97851c87eece

38: *Graph 3D - Playground*. (o. D.). ALMENDE. Abgerufen am 7. Februar 2022, von https://almende.github.io/chap-links-library/js/graph3d/playground

39: J.M.C. (2017, 15. August). *Neural Network Momentum Using Python* -. Visual Studio Magazine. Abgerufen am 6. Februar 2022, von https://visualstudiomagazine.com/articles/2017/08/01/neural-network-momentum.aspx

40: *What does momentum mean in neural networks?* (o. D.). Quora. Abgerufen am 6. Februar 2022, von https://www.quora.com/What-does-momentum-mean-in-neural-networks

41: Wang, C. (2021, 7. Dezember). *The Vanishing Gradient Problem - Towards Data Science*. Medium. Abgerufen am 6. Februar 2022, von https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

42: James McCaffrey, J. (2014, 1. August). *Understanding Neural Network Batch Training: A Tutorial* -. Visual Studio Magazine. Abgerufen am 6. Februar 2022, von https://visualstudiomagazine.com/Articles/2014/08/01/Batch-Training.aspx

43: *Does adding more layers always result in more accuracy in convolutional neural networks?* (o. D.). Quora. Abgerufen am 6. Februar 2022, von https://www.quora.com/Does-adding-more-layers-always-result-in-more-accuracy-in-convolutional-neural-networks

44: Wikipedia contributors. (2022c, Februar 2). *Recurrent neural network*. Wikipedia. Abgerufen am 7. Februar 2022, von https://en.wikipedia.org/wiki/Recurrent_neural_network

45: Education, I. C. (2021, 7. April). *Recurrent Neural Networks*. IBM. Abgerufen am 7. Februar 2022, von https://www.ibm.com/cloud/learn/recurrent-neural-networks

46: Montegriffo, N. (2019, 7. März). *Fünf tolle KI-Apps, die Spaß machen und nützlich sind*. NextPit. Abgerufen am 8. Februar 2022, von https://www.nextpit.de/beste-ki-apps

47: GeeksforGeeks. (2018, 3. Oktober). *Introduction to Recurrent Neural Network*. Abgerufen am 7. Februar 2022, von https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

48: Wang, T. (o. D.). *Recurrent Neural Network*. University of Toronto. Abgerufen am 7. Februar 2022, von https://www.cs.toronto.edu/~tingwuwang/rnn_tutorial.pdf

49: Wikipedia-Autoren. (2003a, Januar 25). *Sprachsynthese*. Wikipedia. Abgerufen am 7. Februar 2022, von https://de.wikipedia.org/wiki/Sprachsynthese

50: Wikipedia contributors. (2021, 4. November). *Types of artificial neural networks*. Wikipedia. Abgerufen am 7. Februar 2022, von https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks

51: Wikipedia-Autoren. (2014, 20. April). *Boltzmann-Maschine*. Wikipedia. Abgerufen am 7. Februar 2022, von https://de.wikipedia.org/wiki/Boltzmann-Maschine

52: Wikipedia contributors. (2021b, November 27). *Restricted Boltzmann machine*. Wikipedia. Abgerufen am 7. Februar 2022, von https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine

53: *Contrastive Divergence*. (2019, 16. Oktober). My Research Wiki. Abgerufen am 7. Februar 2022, von https://wiki.haowen-xu.com/Deep_Learning/Confronting_Partition_Function/Contrastive_Divergence/

54: Serrano.Academy. (2020, 7. Juli). *Restricted Boltzmann Machines (RBM) - A friendly introduction* [Video]. YouTube. https://www.youtube.com/watch?v=Fkw0_aAtwIw

55: DeepLearning.TV. (2015, 17. Dezember). *Deep Belief Nets - Ep. 7 (Deep Learning SIMPLIFIED)* [Video]. YouTube. https://www.youtube.com/watch?v=E2Mt_7qked0

56: DeepLearning.TV. (2015a, Dezember 15). *Restricted Boltzmann Machines - Ep. 6 (Deep Learning SIMPLIFIED)* [Video]. YouTube. https://www.youtube.com/watch?v=puux7KZQfsE

57: Nayak, M. (2021, 9. Dezember). *An Intuitive Introduction Of Restricted Boltzmann Machine (RBM)*. Medium. Abgerufen am 7. Februar 2022, von https://medium.datadriveninvestor.com/an-intuitive-introduction-of-restricted-boltzmann-machine-rbm-14f4382a0dbb

58: *The Algorithm*. (1995, 28. November). Eberhard Karls Universität Tübingen. Abgerufen am 8. Februar 2022, von http://www.ra.cs.uni-tuebingen.de/SNNS/UserManual/node166.html#SECTION001081100000000000000

59: *Was sind neuronale Netze und wie werden sie eingesetzt?* (o. D.). Retresco. Abgerufen am 8. Februar 2022, von https://www.retresco.de/lexikon/neuronale-netze/

60: Protalinski, E. (2015, 9. Juli). *Google now uses an artificial neural network to fight spam, debuts Gmail Postmaster Tools to cut false positives*. VentureBeat. Abgerufen am 8. Februar 2022, von https://venturebeat.com/2015/07/09/google-launches-gmail-postmaster-tools-to-help-companies-ensure-their-emails-arent-marked-as-spam/

61: Kopp, O. (2021, 19. November). *Die Rolle von Machine Learning in der Google-Suche*. Searchmetrics SEO & Content Marketing Blog. Abgerufen am 8. Februar 2022, von https://blog.searchmetrics.com/de/rolle-machine-learning-google/

62: *Waymo Driver*. (o. D.). Waymo. Abgerufen am 8. Februar 2022, von https://waymo.com/waymo-driver/

63: Wikipedia contributors. (2022d, Februar 4). *Self-driving car*. Wikipedia. Abgerufen am 8. Februar 2022, von https://en.wikipedia.org/wiki/Self-driving_car

64: Dr. Orwat, C. (2020, 8. Juli). *Risks of Discrimination through the Use of Algorithms*. Federal Anti-Discrimination Agency. Abgerufen am 7. Februar 2022, von https://www.antidiskriminierungsstelle.de/EN/homepage/_documents/download_diskr_risiken_verwendung_von_algorithmen.pdf

65: Peck, B. (2021, 9. Dezember). *Monitoring Content — The Good and Bad of How Social Media Platforms Influence Our Perspectives*. Medium. Abgerufen am 7. Februar 2022, von https://medium.datadriveninvestor.com/monitoring-content-the-good-and-bad-of-how-social-media-platforms-bias-structures-perspectives-a11223b989ce

66: R.P. (2016, 14. November). *Social media is blinding us to other points of view*. CBC. Abgerufen am 7. Februar 2022, von https://www.cbc.ca/news/science/u-s-election-social-media-biases-1.3848120

67: Gould, W. R. (2019, 21. Oktober). *Are you in a social media bubble? Here's how to tell*. NBC News. Abgerufen am 7. Februar 2022, von https://www.nbcnews.com/better/lifestyle/problem-social-media-reinforcement-bubbles-what-you-can-do-about-ncna1063896

68: Wikipedia-Autoren. (2011, 2. Mai). *Filterblase*. Wikipedia. Abgerufen am 7. Februar 2022, von https://de.wikipedia.org/wiki/Filterblase

69: Lakra, S., Prasad, T. V. & Ramakrishna, G. (2012, Februar). *The Future of Neural Networks*. ResearchGate. Abgerufen am 8. Februar 2022, von https://www.researchgate.net/publication/230899572_The_Future_of_Neural_Networks

70: *Neural Processor - WikiChip*. (o. D.). WikiChip. Abgerufen am 8. Februar 2022, von https://en.wikichip.org/wiki/neural_processor

71: Landman, F. (2019, 26. Januar). *Everything You Need to Know About the Future of Neural Networks*. ReadWrite. Abgerufen am 8. Februar 2022, von https://readwrite.com/everything-you-need-to-know-about-the-future-of-neural-networks/

72: *NeuralNetwork - Pastebin.com*. (2018, 8. Februar). Pastebin. Abgerufen am 9. Februar 2022, von https://pastebin.com/5pKQ1PcE

73: Finn Eggers. (2017, 26. Mai). *NN - Fully Connected Tutorial* [Video]. YouTube. https://www.youtube.com/playlist?list=PLgomWLYGNl1dL1Qsmgumhcg4HOcWZMd3k

74: Kolarova, D. (2019, 21. September). *Designing a Neural Network in Java From a Programmer's Perspective*. Dzone.Com. Abgerufen am 9. Februar 2022, von https://dzone.com/articles/designing-a-neural-network-in-java

Bildquellen

Dnu72 (https://commons.wikimedia.org/wiki/File:Función_sigmoide_02.svg), „Función sigmoide 02“, https://creativecommons.org/licenses/by-sa/3.0/legalcode

MartinThoma (https://commons.wikimedia.org/wiki/File:Sigmoid-function-2.svg), „Sigmoid-function-2“, https://creativecommons.org/publicdomain/zero/1.0/legalcode

Perhelion (https://commons.wikimedia.org/wiki/File:NeuronModel_deutsch.svg), „NeuronModel deutsch“, https://creativecommons.org/licenses/by-sa/3.0/legalcode

Dake, Mysid (https://commons.wikimedia.org/wiki/File:Neural_network.svg), „Neural network“, https://creativecommons.org/licenses/by/1.0/legalcode

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende schriftliche Arbeit ohne fremde Hilfe angefertigt und nur die im Literatur- und Quellenverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Neustadt in Sachsen, 26.02.2022; Lennart Venus