Informatik-GK

Q4 Künstliche Intelligenz

Thomas Klein

Februar 2023

Inhaltsverzeichnis

Vorwort

Ein »Neuronales Netz« ist ein mathematisches Modell, das lose an den Aufbau unseres Gehirns erinnert. Dieses wurde bereits 1956 in den USA entwickelt und schon 1958 gab es mit dem »Mark I Perceptron« einen Computer, der mit einem Bildsensor handgeschriebene Ziffern erkennen konnte. Dies waren die Anfänge der sog. »künstlichen Intelligenz«:

Definition 1 *Der Begriff der* **Künstlichen Intelligenz** *lässt sich aktuell nicht definieren, da der Begriff der »Intelligenz« nicht definiert ist. Allgemein meint man damit Algorithmen, die ein »intelligentes Verhalten« simulieren sollen.*

Durch die heute verfügbaren großen Datenmengen (»Big Data«) und schnellen Rechner, erleben neuronale Netze einen neuen Aufschwung. Das aktuell (Anfang 2023) berühmteste neuronale ist ist ChatGPT, eine künstliche Intelligenz, die die menschliche Sprache versteht und scheinbar intelligente Antworten auf Anfragen generieren kann. Es ist davon auszugehen, dass die Entwicklung in den nächsten Jahren noch deutlich an Fahrt aufnehmen wird.

In dieser Unterrichtseinheit geht es darum zu verstehen, was ein neuronales Netz ist und vor allem, wie es in der Lage ist, selbstständig anhand von Trainingsdaten zu lernen.

1 Das Regressionsproblem

Im Gegensatz zu klassischen Algorithmen werden KIs nicht programmiert, sondern anhand von Daten *trainiert*. Man sagt auch, dass eine KI selbstständig lernt, ein Problem zu lösen. Dieses selbstständige Lernen bezeichnet man auch als **Maschinelles Lernen**.

In diesem ersten Kapitel wollen wir anhand eines (relativ) einfachen Beispiels erkunden, wie maschinelles Lernen funktioniert.

Beispiel 1.1 Ein Unternehmen möchte ein neues Produkt auf den Markt bringen (z. B. einer Schrank) und muss einen Preis für das Produkt festlegen. Der Absatz (= verkaufte Einheiten des Produkts) hängt natürlich vom Preis ab, ein niedriger Preis wird zu einem höheren Absatz führen als ein hoher Preis.

Nun werden für einen Monat in vier unterschiedlichen Geschäften die Schränke testweise zum Verkauf angeboten. Man erhält z. B. folgende Daten:

Preis in € (x)	50	60	70	80
Absatz ($f(x)$)	27	16	14	5

Nun geht es darum, die Funktion $f(x)$ zu bestimmen und zwar anhand der obigen Trainingsdaten. Wir suchen also eine Funktion f, die den Zusammenhang zwischen Preis und Absatz möglichst gut modelliert. #

Information 1.2 Unter dem **Regressionsproblem** versteht man das Finden einer Funktion f, die eine gegebene Anzahl von Punkten möglichst gut annähert.

Dazu muss ein **Ansatz** für die gesuchte Funktion f gemacht werden. Im einfachsten Fall geht man davon aus, dass f eine lineare Funktion ist, d.h., es gilt

$$f(x) = mx + b$$

In diesem Fall müssen die Parameter m und b so bestimmt werden, dass f die gegebenen Punkte möglichst gut annähert.

Beispiel 1.3 Nun stellt sich die Frage, was »möglichst« gut bedeuten soll. **Abb. 1** zeigt 4 verschiedene Geraden, die durch die gegebenen Punkte gelegt wurden.

»Offensichtlich« ist die Gerade in (b) die »beste« Annäherung an die Punkte. Um aber überhaupt sagen zu können, dass eine Näherung besser ist als eine andere, muss man die Näherungen bewerten.

Wir benötigen also eine weitere Funktion, die eine Näherung bewerten kann, man sagt auch, dass eine solche Funktion die »Kosten« der Annäherung misst. #

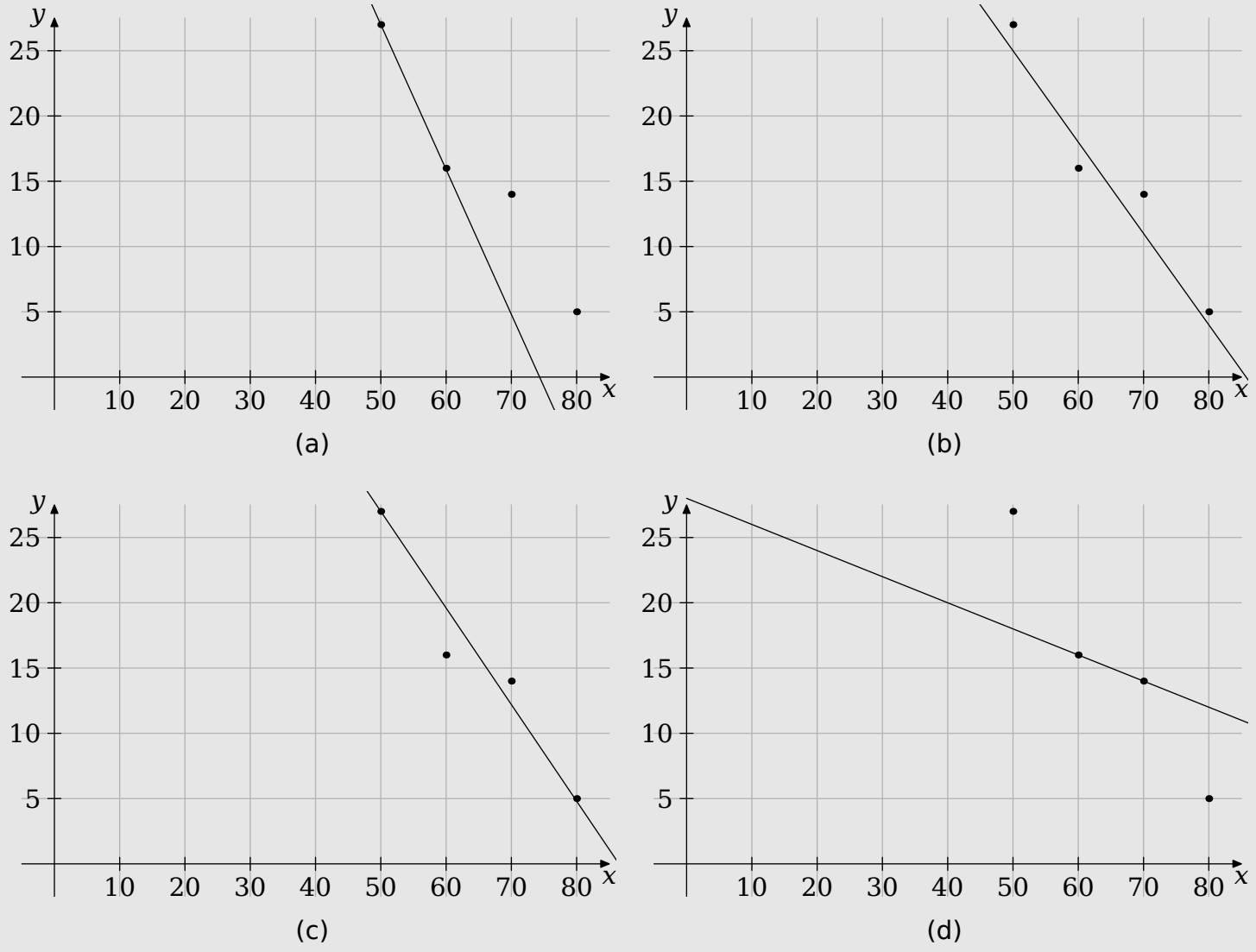


Abbildung 1

Definition 1.4 *Die Funktion*

$$C(f) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

heißt **Kostenfunktion** *zu den Punkten* $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ *(englisch:* **Cost-Function***).*

Sie misst, wie weit die Funktion f *im Mittel von den Punkten* $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ *entfernt ist. Je kleiner* $C(f)$*, desto besser approximiert* f *die Punkte.*

Satz 1.5 Die Kostenfunktion C hat die folgenden Eigenschaften:

(a) Sie ist niemals negativ.

(b) Je kleiner $C(f)$ desto besser ist die Annäherung.

(c) Gilt $C(f) = 0$, dann werden die Punkte von f perfekt getroffen, d.h. f ist dann die beste denkbare Annäherung.

Beweis: Übung! □

Information 1.6 Um eine möglichst gute Näherung zu erhalten, muss man den Tiefpunkt der Kostenfunktion finden.

Beispiel 1.7 Da in unserem Beispiel die Funktion $f(x) = mx + b$ lautet, können wir die Kostenfunktion als Funktion von m und b angeben:

$$C(m, b) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(mx_i + b - y_i)^2$$

Nun können wir die gegebenen Punkte $(x_i \mid y_i)$ einsetzen und erhalten

$$C(m, b) = \frac{1}{4}\cdot\left((50m + b - 27)^2 + (60m + b - 16)^2 + (70m + b - 14)^2 + (80m + b - 5)^2\right)$$

Gesucht ist nun m und b, sodass C minimal wird. Dazu müssen die Ableitungen von C beide gleichzeitig 0 werden. Wir bilden nun die Ableitungen von C nach m und nach b:

Zunächst die Ableitung nach m:

$$\begin{aligned}\frac{\partial C}{\partial m} &= \frac{1}{4}\cdot\Big(2\cdot(50m+b-27)\cdot 50 + 2\cdot(60m+b-16)\cdot 60\\ &\quad + 2\cdot(70m+b-14)\cdot 70 + 2\cdot(80m+b-5)\cdot 80\Big)\\ &= \frac{1}{4}\Big(5000m+100b-2700+7200m+120b-1920\\ &\quad + 9800m+140b-1960+12800m+160b-800\Big)\\ &= \frac{1}{4}\Big(34800m+520b-7380\Big)\\ &= 8700m+130b-1845\end{aligned}$$

Nun die Ableitung nach b:

$$\begin{aligned}\frac{\partial C}{\partial b} &= \frac{1}{4}\cdot\Big(2\cdot(50m+b-27)+2\cdot(60m+b-16)\\ &\quad + 2\cdot(70m+b-14)+2\cdot(80m+b-5)\Big)\\ &= \frac{1}{4}\Big(100m+2b-54+120m+2b-32\\ &\quad + 140m+2b-28+160m+2b-10\Big)\\ &= \frac{1}{4}\Big(520m+8b-124\Big)\\ &= 130m+2b-31\end{aligned}$$

Um das Minimum zu finden, müssen *beide* Ableitungen gleichzeitig 0 sein. Das ergibt das LGS

$$\left|\begin{array}{rcr} 8700m+130b & = & 1845\\ 130m+2b & = & 31\end{array}\right|$$

Dieses LGS kann man per Hand oder per Rechner lösen und erhält als Lösung $m=-0{,}68$ und $b=59{,}7$.

Die gesuchte Funktion ist also $f(x)=-0{,}68x+59{,}7$. Damit kann man nun vorhersagen, wie viele Schränke bei einem bestimmten Preis verkauft werden. #

Bemerkung 1.8 Die obige Rechnung erscheint auf den ersten Blick furchteinflößend komplex. Das Wesentliche ist aber, dass sie automatisch von einem Computer durchgeführt werden kann.

Das bedeutet, dass der Computer selbstständig anhand der Trainingsdaten (die Punkte $(x_i \mid y_i)$) einen Algorithmus erlernen kann! Würden beispielsweise weitere Trainingsda-

ten hinzukommen, kann das Verfahren wieder angewendet werden und der Algorithmus wird weiter verbessert. #

Information 1.9 Zusammenfassung maschinelles Lernen:

Das Ziel ist, eine Funktion f zu finden, die die Trainingsdaten möglichst gut annähert.

Gegeben:	Punkte $(x_i \mid y_i)$ (»Trainingsdaten«)
Gesucht:	Funktion f, die die Trainingsdaten möglichst gut annähert
Lösung:	(1) Mache einen Ansatz für f.
	(2) Formuliere die Kostenfunktion $C(f)$.
	(3) Stelle die Parameter von f so ein, dass die Kostenfunktion möglichst klein wird.

Das größte Problem dabei stellt Schritt (3) dar, wie wir im Beispiel gesehen haben. Daher werden wir uns im nächsten Kapitel das Gradientenverfahren ansehen, ein Algorithmus, der automatisch einen Tiefpunkt einer Funktion finden kann.

2 Das Gradientenverfahren

Im letzten Kapitel haben wir gesehen, dass maschinelles Lernen im Endeffekt bedeutet, einen Tiefpunkt der Kostenfunktion zu finden. Wir haben auch gesehen, dass der algebraische Weg (Ableitung gleich 0 setzen) sehr aufwändig ist und auch nur in ganz besonders einfachen Fällen funktioniert.

Das Gradientenverfahren dagegen ist ein numerisches Verfahren, das einen (lokalen) Tiefpunkt einer Funktion finden kann.

2.1 Das Gradientenverfahren für Funktionen mit 1 Unbekannten

Wir machen uns die Idee zunächst an einem Beispiel klar:

Beispiel 2.1 Gesucht ist der Tiefpunkt der Funktion $f(x) = 3x^2 - 10x + 12$. Zunächst bilden wir die Ableitung

$$f'(x) = 6x - 10$$

Nun führen wir folgende Schritte durch:

(1) Wähle einen zufälligen Startwert x, z. B. $\boxed{x = 3}$

(2) Bilde $f'(3) = 8$ und $f(3) = 9$

(3) Da die Steigung positiv ist, gehen wir ein Stück weiter nach links, z. B. zu 2 und berechnen $f(2) = 4$. Da der Wert kleiner geworden ist, setze $\boxed{x = 2}$.

(4) Bilde $f'(2) = 2$

(5) Da die Steigung immer noch positiv ist, gehen wir wieder ein Stück weiter nach links, z. B. zu 1. Berechne $f(1) = 5$. Da dieser Wert schlechter ist als bei 2, sind wir zu weit gegangen. Wir gehen stattdessen zu $1{,}5$. Hier gilt $f(1{,}5) = 3{,}75$ und dieser Wert ist besser. Also merken wir uns $\boxed{x = 1{,}5}$. Außerdem gehen wir nur noch $0{,}5$er-Schritte.

(6) Bilde $f'(1{,}5) = -1 < 0 \Rightarrow$ gehe nach rechts!

(7) Teste $f(2) = 4$. Da der Wert schlechter ist, reduzieren wir die Schrittweite auf $0{,}25$ und testen $f(1{,}75) = 3{,}6875$. Dieser Wert ist wieder besser, also $\boxed{x = 1{,}75}$

Dies wiederholen wir nun so lange, bis sich der x-Wert nicht mehr genug ändert.

Zum Vergleich, der exakte Tiefpunkt liegt bei $x = \frac{5}{3} \approx 1{,}666...$ #

Beim »echten« Gradientenverfahren geht man zusätzlich so vor, dass man umso weiter nach links/rechts geht, je positiver/negativer die Ableitung ist:

Information 2.2 Das Gradientenverfahren für eine Funktion mit einer Variablen:

Gegeben: Funktion f mit 1 Unbekannten

Gesucht: (ein) Tiefpunkt von f

Lösung:

(1) Wähle einen Startwert x und einen Schrittweitenfaktor s (z. B. $s = 1$).

(2) Berechne $m = f'(x)$ und $y = f(x)$.

(3) Finde einen besseren Wert:

- Teste $f(x - s \cdot m)$: Ist der Wert kleiner als y? Falls ja, weiter mit Schritt (4), andernfalls geht es hier weiter:
- Halbiere s. Ist jetzt $|s \cdot m|$ klein genug, beende das Verfahren. Andernfalls gehe zum vorherigen Schritt zurück.

(4) Setze x auf $x - s \cdot m$. Zurück zu Schritt (2).

Beispiel 2.3 Gesucht wird ein lokaler Tiefpunkt der Funktion

$$f(x) = 0{,}1x^4 - x^2 + 4$$

Es gilt $f'(x) = 0{,}4x^3 - 2x$. **Abb. 2** zeigt die Ausgabe des Algorithmus (mit gerundeten Zahlen). Zum Vergleich: Die exakten Tiefstellen sind $x_{1/2} = \pm\sqrt{5} \approx \pm 2{,}2361$. #

Bemerkung 2.4 Wenn die Funktion mehrere Tiefpunkte hat, findet das Gradientenverfahren nur einen davon, abhängig vom Startwert. Um einen möglichst kleinen Tiefpunkt zu erreichen, muss man das Verfahren mehrfach mit verschiedenen Startwerten durchführen. #

```
Gradientenverfahren mit Startwert x=4 und Schrittweitenfaktor s=1
****Neuer Durchgang****
x=4, y=13.6, ableitung=17.6, s=1
  Teste x=-13.6: f(x)=3240.0601600000014
    Ist schlechter. Halbiere Schrittweitenfaktor
  Teste x=-4.8: f(x)=34.04416
    Ist schlechter. Halbiere Schrittweitenfaktor
  Teste x=-0.4: f(x)=3.84256
    Ist besser! Nächster Durchgang.
****Neuer Durchgang****
x=-0.4, y=3.84256, ableitung=0.7744, s=0.25
  Teste x=-0.5936: f(x)=3.66
    Ist besser! Nächster Durchgang.
****Neuer Durchgang****
x=-0.5936, y=3.66, ableitung=1.103535, s=0.25
  Teste x=-0.8694838: f(x)=3.30115
    Ist besser! Nächster Durchgang.
****Neuer Durchgang****
x=-0.8694838, y=3.30115, ableitung=1.476, s=0.25
  Teste x=-1.2384926: f(x)=2.70141
    Ist besser! Nächster Durchgang.
****Neuer Durchgang****
x=-1.2384926, y=2.70141, ableitung=1.717113, s=0.25
  Teste x=-1.667771: f(x)=1.9922
    Ist besser! Nächster Durchgang.
****Neuer Durchgang****
x=-1.667771, y=1.9922, ableitung=1.48, s=0.25
  Teste x=-2.03777: f(x)=1.57182
    Ist besser! Nächster Durchgang.
****Neuer Durchgang****
x=-2.03777, y=1.57182, ableitung=0.69079, s=0.25
  Teste x=-2.21047: f(x)=1.5013
    Ist besser! Nächster Durchgang.
****Neuer Durchgang****
x=-2.21047, y=1.5013, ableitung=0.1, s=0.25
  Teste x=-2.2356: f(x)=1.5
    Ist besser! Nächster Durchgang.
****Neuer Durchgang****
x=-2.2356, y=1.5, ableitung=0.001, s=0.25
  Teste x=-2.236: f(x)=1.50000001
    Ist schlechter. Halbiere Schrittweitenfaktor
Schrittweite klein genug, FERTIG! Ergebnis: x=-2.2356
```

Abbildung 2

2.2 Das Gradientenverfahren für multivariate Funktionen

In unserem Fall bringt uns diese Form des Gradientenverfahrens allerdings recht wenig, weil unsere Kostenfunktion von mehr als 1 Unbekannten abhängt. Daher muss man das Verfahren auf alle Unbekannten gleichzeitig anwenden. Dabei kommt der »Gradient« und die sogenannten »partiellen Ableitungen« ins Spiel:

Definition 2.5 *Gegeben sei eine Funktion*

$$f(x_1, x_2, ..., x_n) = f(\vec{x})$$

die von mehreren Unbekannten x_1, x_2, ..., x_n abhängt. Diese Unbekannten werden in Vektorschreibweise zusammengefasst:

$$\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

(a) Mit $\frac{\partial f}{\partial x_i}$ bezeichnen wir die **partielle Ableitung** *der Funktion f nach x_i.*

(b) Der **Gradient** *$\vec{\nabla} f$ ist der Vektor, der aus allen partiellen Ableitungen besteht:*

$$\vec{\nabla} f(\vec{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Beispiel 2.6 Für $f(x_1, x_2) = 5x_1 \cdot x_2^2 + x_1^3$ gilt

$$\frac{\partial f}{\partial x_1} = 5x_2^2 + 3x_1^2 \qquad \text{und} \qquad \frac{\partial f}{\partial x_2} = 10x_1x_2$$

und damit

$$\vec{\nabla} f(x_1, x_2) = \begin{pmatrix} 5x_2^2 + 3x_1^2 \\ 10x_1x_2 \end{pmatrix}$$

Somit gilt bspw. $\vec{\nabla} f(-1, 2) = \begin{pmatrix} 5 \cdot 4 + 3 \cdot 1 \\ 10 \cdot (-1) \cdot 2 \end{pmatrix} = \begin{pmatrix} 23 \\ -20 \end{pmatrix}$. Das bedeutet, dass f in x_1-Richtung um 23 steigt und in x_2-Richtung um -20 sinkt. #

Mit Hilfe des Gradienten lässt sich das Gradientenverfahren auch auf Funktionen mit mehr als 1 Unbekannten anwenden:

Information 2.7 Das Gradientenverfahren für eine multivariate Funktion:

Gegeben: Funktion f mit mehreren Unbekannten

Gesucht: (ein) Tiefpunkt von f

Lösung:

(1) Wähle einen Startvektor $\vec{x}$ und einen Schrittweitenfaktor s (z. B. $s = 1$).

(2) Berechne den Gradienten $\overrightarrow{\nabla} = \overrightarrow{\nabla} f(\vec{x})$ und $y = f(\vec{x})$.

(3) Finde einen besseren Wert:

- Teste $f(\vec{x} - s \cdot \overrightarrow{\nabla})$: Ist der Wert kleiner als y? Falls ja, weiter mit Schritt (4), andernfalls geht es hier weiter:
- Halbiere s. Ist jetzt $|s \cdot \overrightarrow{\nabla}|$ klein genug, beende das Verfahren. Andernfalls gehe zum vorherigen Schritt zurück.

(4) Setze $\vec{x}$ auf $\vec{x} - s \cdot \overrightarrow{\nabla}$. Zurück zu Schritt (2).

Bemerkung 2.8 Mit $|s \cdot \overrightarrow{\nabla}|$ ist in diesem Fall die Länge des entsprechenden Vektors gemeint. #

3 Neuronale Netze

Nachdem wir nun in der Lage sind, Kostenfunktionen zu minimieren, benötigen wir noch ein geeignetes Modell für unsere KI. Die sog. »Neuronalen Netze« haben sich als geeignet herausgestellt.

3.1 Aufbau eines neuronalen Netzes

Information 3.1

- Ein **Neuronales Netz** besteht aus mehreren (mindestens 2) durchnummerierten **Schichten**. Die erste Schicht ist die **Eingabeschicht**, die letzte Schicht ist die **Ausgabeschicht**. Die Schichten dazwischen werden auch als **hidden layers** bezeichnet.
- Jede Schicht besteht wiederum aus einer bestimmten Anzahl von **Neuronen**, wobei die Anzahl in jeder Schicht unterschiedlich sein können.
- Ein **Neuron** a ist eine einzelne reelle Zahl, normalerweise im Bereich von 0 bis 1.

  Die Idee dahinter ist: Je höher der Wert eines Neurons ist, desto stärker ist es aktiviert.
- Zwischen jedem Neuron einer Schicht und jedem Neuron der benachbarten Schichten gibt es eine **Verbindung**, die wiederum durch eine reelle Zahl w (beliebiger Größe) dargestellt wird.

  Die Idee hierbei ist, dass eine positive Verbindung dafür sorgt, dass die beiden Neuronen stärker miteinander verbunden sind (ist eines aktiviert, dann auch das andere und umgekehrt). Eine negative Verbindung bedeutet das Gegenteil (ist eines aktiviert, dann das andere eher nicht).
- Jedes Neuron außer denen der Eingabeschicht besitzt außerdem einen sogenannten **Bias** b, eine reelle Zahl beliebiger Größe. Der Bias legt fest, wie schwer oder einfach es ist, das Neuron zu aktivieren.

Abb. 3 zeigt ein neuronales Netz mit 4 Schichten.

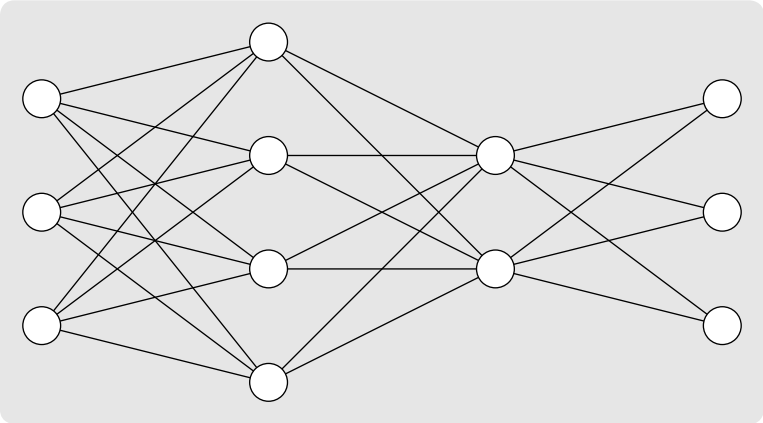


Abbildung 3

3.2 Funktionsweise eines neuronalen Netzes

Information 3.2

- Ein neuronales Netz erhält die Neuronen seiner Eingabeschicht als Eingabe.
- Die Neuronen einer Schicht bestimmten nach einem festgelegten Algorithmus die Neuronen der nachfolgenden Schicht.
- Dadurch werden die Neuronen aller Schichten nacheinander berechnet.
- Die Neuronen der Ausgabeschicht stellen das Ergebnis des Neuronalen Netzes dar.

Nun soll es darum gehen, wie man die Neuronen einer Schicht aus den Neuronen der vorangegangenen Schicht berechnen kann:

Information 3.3 Für ein neuronales Netz mit den Schichten $k = 0, 1, 2, ..., N$ legen wir folgende Bezeichnungen fest:

- Die k-te Schicht besitzt n_k viele Neuronen.
- Die Neuronen der k-ten Schicht werden mit $a_1^{(k)}$, $a_2^{(k)}$, ..., $a_{n_k}^{(k)}$ bezeichnet. Zusammengefasst können wir sie als Vektor darstellen:

$$a^{(k)} = \begin{pmatrix} a_1^{(k)} \\ \vdots \\ a_{n_k}^{(k)} \end{pmatrix}$$

- Die **Biasse** der Neuronen der k-ten Schicht ($k \geq 1$) werden mit $b_1^{(k)}$, $b_2^{(k)}$, ..., $b_{n_k}^{(k)}$ bezeichnet. Zusammengefasst können wir auch sie als Vektor darstellen:

$$b^{(k)} = \begin{pmatrix} b_1^{(k)} \\ \vdots \\ b_{n_k}^{(k)} \end{pmatrix}$$

- Die Verbindung zwischen dem Neuron $a_i^{(k)}$ und dem Neuron $a_j^{(k-1)}$ aus der Vorgänger-Schicht bezeichnen wir mit $w_{i,j}^{(k)}$. Zusammengefasst können wir alle Verbindungen der k-ten Schicht zu ihrer Vorgänger-Schicht als Matrix darstellen:

$$W^{(k)} = \begin{pmatrix} w_{1,1}^{(k)} & w_{1,2}^{(k)} & \dots & w_{1,n_{k-1}}^{(k)} \\ w_{2,1}^{(k)} & w_{2,2}^{(k)} & \dots & w_{2,n_{k-1}}^{(k)} \\ \dots & \dots & \dots & \dots \\ w_{n_k,1}^{(k)} & w_{n_k,2}^{(k)} & \dots & w_{n_k,n_{k-1}}^{(k)} \end{pmatrix}$$

  Diese Matrix hat so viele Zeilen, wie die k-te Schicht Neuronen besitzt (= n_k) und so viele Spalten wie die Vorgänger-Schicht Neuronen besitzt (= n_{k-1}).

  Die Zahlen $w_{i,j}^{(k)}$ werden auch als **Gewichte** des Netzes bezeichnet.

- Die **Sigmoid**-Funktion ist definiert als

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- Wenn wir die Sigmoid-Funktion auf einen Vektor anwenden, soll dies bedeuten, dass die Funktion auf jede einzelne Komponente angewendet wird:

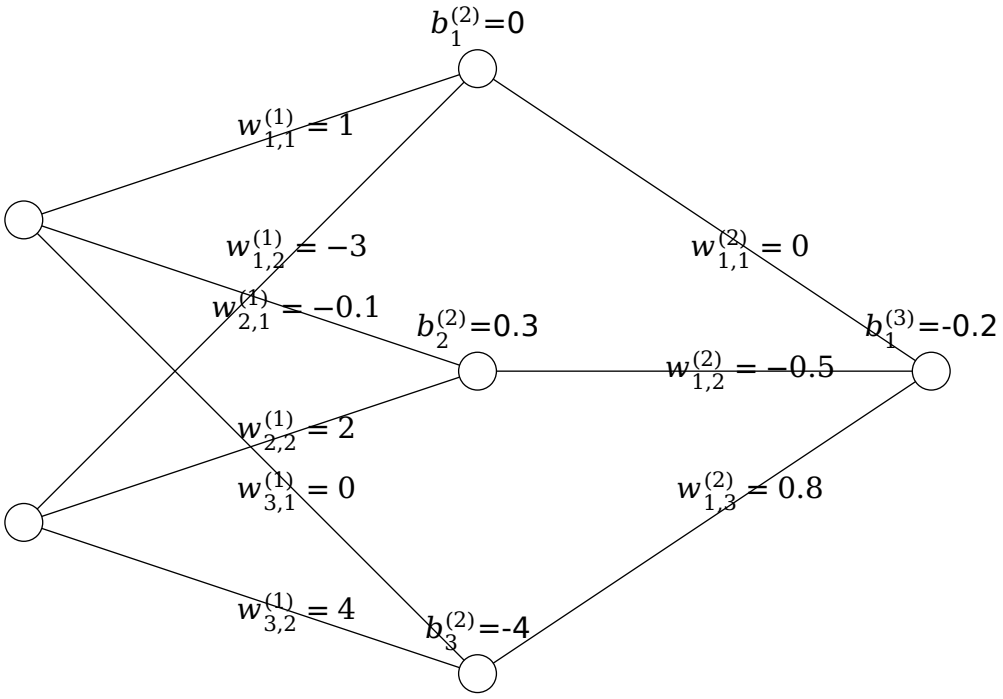
$$\sigma(\vec{x}) = \begin{pmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_n) \end{pmatrix}$$

Mit diesen Bezeichnungen kann man formulieren, wie die Neuronen der k-ten Schicht aus denen der Vorgänger-Schicht bestimmt werden können:

$$a^{(k)} = \sigma\left(W^{(k)} \cdot a^{(k-1)} + b^{(k)}\right)$$

Dabei bezeichnet $W^{(k)} \cdot a^{(k-1)}$ das Produkt zwischen einer Matrix und einem Vektor.

Beispiel 3.4 Wir betrachten ein neuronales Netz mit 3 Schichten:



Dieses Netz besitzt die folgenden Eigenschaften:

- Biasse:

$$b^{(1)} = \begin{pmatrix} 0 \\ 0{,}3 \\ -4 \end{pmatrix} \qquad b^{(2)} = (-0{,}2)$$

- Verbindungsmatrizen:

$$W^{(1)} = \begin{pmatrix} 1 & -3 \\ -0{,}1 & 2 \\ 0 & 4 \end{pmatrix} \qquad W^{(2)} = \begin{pmatrix} 0 & -0{,}5 & 0.8 \end{pmatrix}$$

Bei einer Eingabe von $a^{(0)} = \begin{pmatrix} 0{,}9 \\ 0{,}6 \end{pmatrix}$ ergibt sich dann

- $a^{(1)} = \sigma\left(W^{(1)} \cdot a^{(0)} + b^{(1)}\right) = \sigma\begin{pmatrix} -0{,}9 \\ 0{,}3 \\ 2{,}4 \end{pmatrix} \approx \begin{pmatrix} 0{,}289 \\ 0{,}5744 \\ 0{,}9168 \end{pmatrix}$
- $a^{(2)} = \sigma\left(W^{(2)} \cdot a^{(1)} + b^{(2)}\right) = \sigma(0{,}4462) \approx 0{,}61$

Für die Eingabe $\begin{pmatrix} 0{,}9 \\ 0{,}6 \end{pmatrix}$ erhalten wir also das Ergebnis $0{,}61$. #

4 Backpropagation

Nun haben wir prinzipiell alle Zutaten zusammen, um ein neuronales Netz trainieren zu können:

(1) Erzeuge ein neuronales Netz.

(2) Wende das Gradientenverfahren auf die Kostenfunktion an, um die Parameter (Gewichte und Biasse) des Netzes so einzustellen, dass ein (lokales) Minimum der Kostenfunktion gefunden wird.

4.1 Die Kostenfunktion

Die Kostenfunktion soll messen, wie gut unser neuronales Netz N die Trainingsdaten trifft. Die Trainingsdaten sind wiederum in der Form

$$(\vec{x}_1 \mid \vec{y}_1),\ (\vec{x}_2 \mid \vec{y}_2),\ ...,\ (\vec{x}_n \mid \vec{y}_n)$$

als Paare von Vektoren gegeben. (Mit der Bedeutung: Eine Eingabe von $\vec{x}_i$ sollte möglichst zu einer Ausgabe von $\vec{y}_i$ führen.)

Definition 4.1 *Für ein neuronales Netz f legen wir fest:*

- *Es hat die Schichten von $k = 0$ bis $k = N$,*
- *n_k ist die Anzahl der Neuronen der k-ten Schicht,*
- *$a_i^{(k)}$ ist das i-te Neuron der k-ten Schicht ($i = 1, 2, ..., n_k$),*
- *$b_i^{(k)}$ ist der Bias des i-ten Neurons der k-ten Schicht ($i = 1, 2, ..., n_k$),*
- *$w_{i,j}^{(k)}$ ist das Gewicht der Verbindung zwischen dem Neuron $a_i^{(k)}$ und dem Neuron $a_j^{(k-1)}$ aus dessen Vorgänger-Schicht ($i = 1, 2, ..., n_k,\ j = 1, 2, ..., n_{k-1}$).*

Außerdem seien n Trainingsdaten in der Form

$$(\vec{x}_1 \mid \vec{y}_1),\ (\vec{x}_2 \mid \vec{y}_2),\ ...,\ (\vec{x}_n \mid \vec{y}_n)$$

gegeben.

Die **Kostenfunktion** *C des neuronalen Netzes f zu diesen Trainingsdaten ist dann gegeben durch*

$$C = \frac{1}{n}\sum_{m=1}^{n}\left(f(\vec{x}_m) - \vec{y}_m\right)^2 = \frac{1}{n}\sum_{m=1}^{n} C_m$$

Dabei sind

$$C_m = \left(f(\vec{x}_m) - \vec{y}_m\right)^2 = \sum_{\ell=1}^{n_N}\left(a_\ell^{(N)} - (\vec{y}_m)_\ell\right)^2$$

die Kosten, die das m-te Trainingsdatum verursacht, wobei $(\vec{y}_m)_\ell$ die ℓ-te Komponente des Vektors $\vec{y}_m$ ist.

4.2 Die Ableitung der Kostenfunktion

Die Kostenfunktion ist ein ziemlich heftiges Biest. Um das Gradientenverfahren anwenden zu können, benötigen wir die Ableitung (genauer: den Gradienten) dieser Funktion, die von den Gewichten $w_{i,j}^{(k)}$ und Biassen $b_i^{(k)}$ abhängt.

Das Erstaunliche: Obwohl wir C nicht als Term aufschreiben können, können wir trotzdem die partiellen Ableitungen berechnen.

Satz 4.2 Mit den obigen Bezeichnungen und

$$z^{(k)} = W^{(k)} \cdot a^{(k-1)} + b^{(k)}$$

gelten für $k = 1, 2, ..., N$ die Rekursionsformeln

$$\frac{\partial C_m}{\partial w_{i,j}^{(k)}} = \sigma'\left(z_i^{(k)}\right) \cdot a_j^{(k-1)} \cdot \frac{\partial C_m}{\partial a_i^{(k)}}$$

$$\frac{\partial C_m}{\partial b_i^{(k)}} = \sigma'\left(z_i^{(k)}\right) \cdot \frac{\partial C_m}{\partial a_i^{(k)}}$$

$$\frac{\partial C_m}{\partial a_i^{(k)}} = \begin{cases} 2(a_i^{(N)} - (\vec{y}_m)_i) & \text{für } k = N \\ \sum_{\ell=1}^{n_{k+1}} \sigma'(z_\ell^{(k+1)}) \cdot w_{\ell,i}^{(k+1)} \cdot \frac{\partial C_m}{\partial a_\ell^{(k+1)}} & \text{für } k < N \end{cases}$$

Beweis: Um den Gradienten zu bestimmen, müssen wir die partiellen Ableitungen nach den Gewichten und den Biassen bestimmen.

Zunächst berechnen wir einige Ableitungen, die wir mehrfach benötigen werden. Nach unseren Definitionen gilt

$$a_\ell^{(k)} = \sigma\left(w_{\ell,1}^{(k)} \cdot a_1^{(k-1)} + w_{\ell,2}^{(k)} \cdot a_2^{(k-1)} + ... + w_{\ell,n_{k-1}}^{(N)} \cdot a_{n_{k-1}}^{(k-1)} + b_\ell^{(k)}\right) = \sigma(z_\ell^{(k)})$$

Damit erhalten wir

$$\frac{\partial a_\ell^{(k)}}{\partial z_\ell^{(k)}} = \sigma'\left(z_\ell^{(k)}\right)$$

$$\frac{\partial z_\ell^{(k)}}{\partial a_i^{(k-1)}} = \frac{\partial}{\partial a_i^{(k-1)}}\left(w_{\ell,1}^{(k)} \cdot a_1^{(k-1)} + w_{\ell,2}^{(k)} \cdot a_2^{(k-1))} + \ldots + w_{\ell,n_{k-1}}^{(k)} \cdot a_{n_{k-1}}^{(k-1)} + b_\ell^{(k)}\right) = w_{\ell,i}^{(k)}$$

$$\frac{\partial z_\ell^{(k)}}{\partial w_{i,j}^{(k)}} = \frac{\partial}{\partial w_{i,j}^{(k)}}\left(w_{\ell,1}^{(k)} \cdot a_1^{(k-1)} + w_{\ell,2}^{(k)} \cdot a_2^{(k-1)} + \ldots + w_{\ell,n_{k-1}}^{(N)} \cdot a_{n_{k-1}}^{(k-1)} + b_\ell^{(k)}\right) = \begin{cases} a_j^{(k-1)} & \text{falls } \ell\text{=i} \\ 0 & \text{sonst} \end{cases}$$

$$\frac{\partial z_\ell^{(k)}}{\partial b_i^{(k)}} = \frac{\partial}{\partial b_i^{(k)}}\left(w_{\ell,1}^{(k)} \cdot a_1^{(k-1)} + w_{\ell,2}^{(k)} \cdot a_2^{(k-1)} + \ldots + w_{\ell,n_{k-1}}^{(N)} \cdot a_{n_{k-1}}^{(k-1)} + b_\ell^{(k)}\right) = \begin{cases} 1 & \text{falls } \ell\text{=i} \\ 0 & \text{sonst} \end{cases}$$

Damit können wir die gesuchten Ableitungen nach den Gewichten bestimmen. Dazu bauen wir mit Hilfe der Kettenregel die Ableitungen nach den Neuronen ein und müssen diese summieren:

$$\begin{aligned} \frac{\partial C_m}{\partial w_{i,j}^{(k)}} &= \sum_{\ell=1}^{N_k} \frac{\partial C_m}{\partial a_\ell^{(k)}} \cdot \frac{\partial a_\ell^{(k)}}{\partial z_\ell^{(k)}} \cdot \frac{\partial z_\ell^{(k)}}{\partial w_{i,j}^{(k)}} \\ &= \frac{\partial C_m}{\partial a_i^{(k)}} \cdot \sigma'\left(z_i^{(k)}\right) \cdot a_j^{(k-1)} \end{aligned}$$

Dabei blieb von der Summe nur der Summand mit $\ell = i$ übrig. Genauso können wir die Ableitung nach den Biassen bestimmen:

$$\begin{aligned} \frac{\partial C_m}{\partial b_i^{(k)}} &= \sum_{\ell=1}^{N_k} \frac{\partial C_m}{\partial a_\ell^{(k)}} \cdot \frac{\partial a_\ell^{(k)}}{\partial z_\ell^{(k)}} \cdot \frac{\partial z_\ell^{(k)}}{\partial b_i^{(k)}} \\ &= \frac{\partial C_m}{\partial a_i^{(k)}} \cdot \sigma'\left(z_i^{(k)}\right) \cdot 1 \end{aligned}$$

Damit sind die ersten beiden Formeln gezeigt. Die letzte Formel erhalten wir prinzipiell genauso: Für $k < N$ gilt:

$$\begin{aligned} \frac{\partial C_m}{\partial a_i^{(k)}} &= \sum_{\ell=1}^{n_{k+1}} \frac{\partial C_m}{\partial a_\ell^{(k+1)}} \cdot \frac{\partial a_\ell^{(k+1)}}{\partial z_\ell^{(k+1)}} \cdot \frac{\partial z_\ell^{(k+1)}}{\partial a_i^{(k)}} \\ &= \sum_{\ell=1}^{n_{k+1}} \frac{\partial C_m}{\partial a_\ell^{(k+1)}} \cdot \sigma'\left(z_\ell^{(k+1)}\right) \cdot w_{\ell,i}^{(k+1)} \end{aligned}$$

Für $k = N$ endet die Rekursion: Nach Definition ist

$$C_m = \left(a_1^{(N)} - (\vec{y}_m)_1\right)^2 + \left(a_2^{(N)} - (\vec{y}_m)_2\right)^2 + \ldots + \left(a_{n_N}^{(N)} - (\vec{y}_m)_{n_N}\right)^2$$

Wenn man diesen Ausdruck nach $a_i^{(N)}$ ableitet, fallen alle Terme bis auf den i-ten weg, weil kein $a_i^{(N)}$ in ihnen vorkommt:

$$\frac{\partial C_m}{\partial a_i^{(N)}} = 0 + 0 + \ldots + 0 + \frac{\partial}{\partial a_i^{(N)}} \left(a_i^{(N)} - (\vec{y}_m)_i\right)^2 + 0 + \ldots + 0$$
$$= 2 \cdot \left(a_i^{(N)} - (\vec{y}_m)_i\right) \cdot 1$$

Das bedeutet, die Kostenänderung, die ein Neuron der letzten Schicht verursacht, entsprechen dem Doppelten der Differenz des Neurons zum gewünschten Wert. Das macht perfekt Sinn.

□

4.3 Zusammenfassung

Damit können wir nun einen Algorithmus formulieren, mit dem ein neuronales Netz anhand von Traingsdaten trainiert werden kann:

Information 4.3

Gegeben: Neuronales Netz $f = \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)} \ldots, W^{(N)}, b^{(N)}\}$ mit $N+1$ Schichten (n_k Neuronen in der k-ten Schicht);
Trainingsdaten $(\vec{x}_1 \mid \vec{y}_1), \ldots, (\vec{x}_n \mid \vec{y}_n)$

Gesucht: Werte für die Gewichte $W^{(k)}$ und die Biasse $b^{(k)}$, für die das neuronale Netz die Trainingsdaten möglichst gut annähert (die Kostenfunktion minimiert)

Lösung:

(1) Erzeuge ein neuronales Netz mit zufälligen Werten für die Gewichte und Biasse.

(2) Verbessere die Gewichte und Biasse mit Hilfe des Gradientenverfahrens:

  (1) Schrittweite $w = 1$

  (2) Wiederhole unendlich:

  - ▷ Wähle eine zufällige Auswahl der Trainingsdaten (z. B. 200 Stück).
  - ▷ Berechne die Kosten von f für diese Trainingsdaten.
  - ▷ Erzeuge ein neues neuronales Netz g (für »Gradient«) als Kopie von f, wobei aber alle Gewichte und Biasse auf 0 eingestellt werden.
  - ▷ Wiederhole für alle ausgewählten Trainingsdaten m:

Wiederhole für $k = N$ bis $k = 1$:

- Berechne $\frac{\partial C_m}{\partial a_i^{(k)}} = \begin{cases} 2(a_i^{(N)} - (\vec{y}_m)_i) & \text{für } k = N \\ \sum_{\ell=1}^{n_{k+1}} \sigma'(z_\ell^{(k+1)}) \cdot w_{\ell,i}^{(k+1)} \cdot \frac{\partial C_m}{\partial a_\ell^{(k+1)}} & \text{für } k < N \end{cases}$
- Berechne damit $\frac{\partial C_m}{\partial w_{i,j}^{(k)}} = \sigma'\left(z_i^{(k)}\right) \cdot a_j^{(k-1)} \cdot \frac{\partial C_m}{\partial a_i^{(k)}}$ und $\frac{\partial C_m}{\partial b_i^{(k)}} = \sigma'\left(z_i^{(k)}\right) \cdot \frac{\partial C_m}{\partial a_i^{(k)}}$
- Ändere die Gewichte von g in der k-ten Schicht um $\frac{\partial C_m}{\partial w_{i,j}^{(k)}}$ und die Biasse von g in der k-ten Schicht um $\frac{\partial C_m}{\partial b_i^{(k)}}$

▷ Setze `fertig = false`

▷ Solange `fertig = false`:

- Erzeuge ein neues Neuronales Netz t als exakte Kopie von f.
- Wiederhole für $k = 1$ bis N:
  - Subtrahiere das w-fache der Gewichtsmatrizen der k-ten Schicht von g von den Gewichtsmatrizen von t.
  - Subtrahiere das w-fache der Biasse der k-ten Schicht von g von den Biassen von t.
- Berechne die Kosten für t für die ausgewählten Trainingsdaten.
- Wenn die Kosten für t niedriger sind als die Kosten für f: Ersetze f durch t. Setze `fertig = true`.
- Andernfalls: Halbiere w. Ist w klein genug: Beende den Algorithmus.