Architektur Neuronaler Netze für Generative KI

Grundlagen Neuronaler Netze 2

Versteckte Schichten, Feed Forward, Aktivierungsfunktionen, etc.

Hochschule Worms • Fachbereich Informatik

Prof. Dr. Stephan Kurpjuweit

Wiederholung

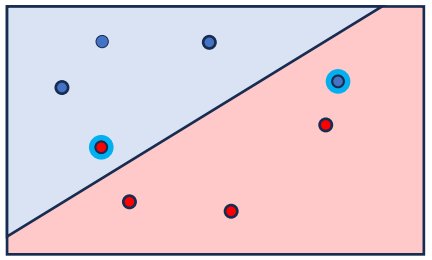
Aus Teil 1…

1 Binäre Klassifikation

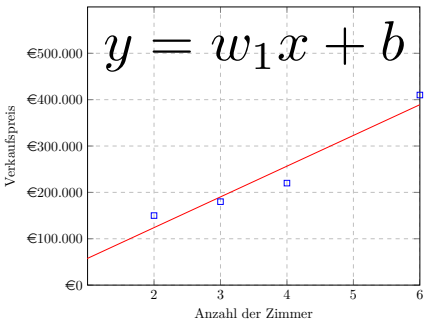
$$\begin{cases} 1 & \text{für } Katze \\ 0 & \text{für } keine\ Katze \end{cases}$$

x y

2



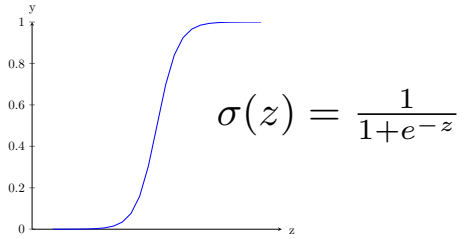
3 Lineare Regression



4 allgemein

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_n x_n + b$$

5 Logistische Funktion (Sigmoid)



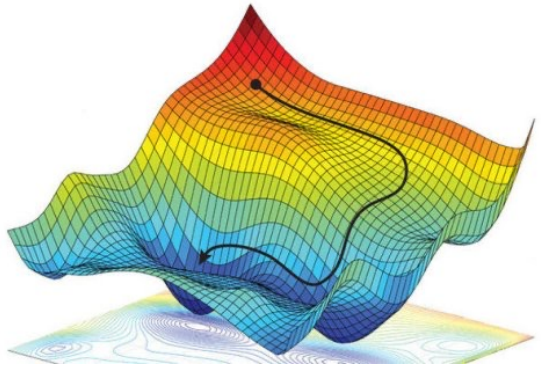
6 Logistische Regression

$$\hat{y} = \sigma(w^T x + b) = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_{n_x} x_{n_x} b)$$

7 Fehlerfunktion bei der logistischen Regression

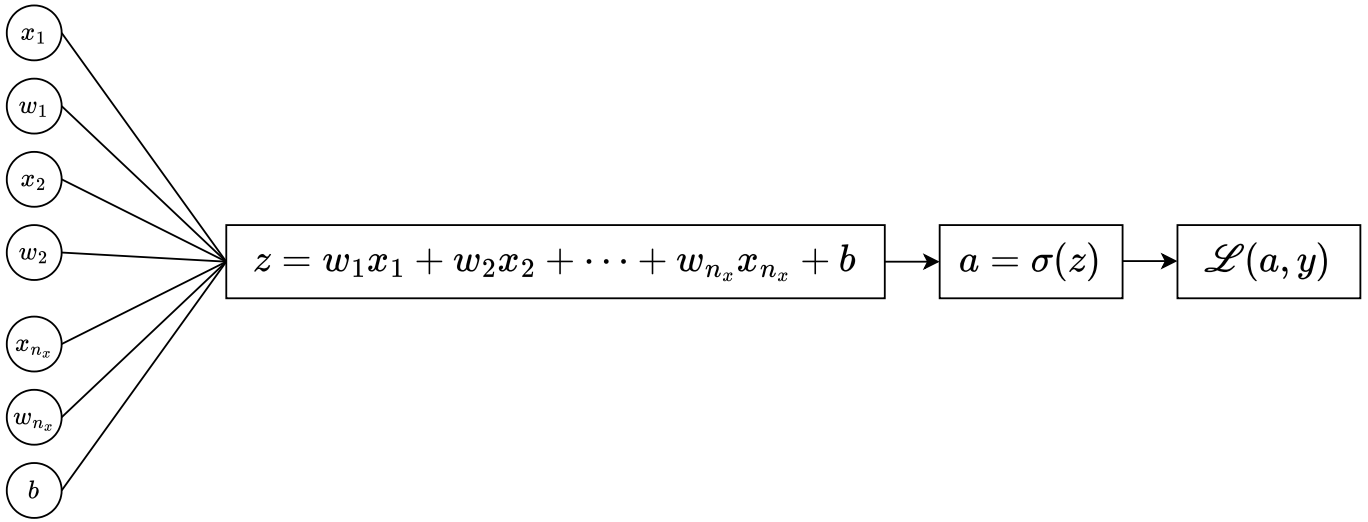
$$\mathscr{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

8 Gradientenabstieg

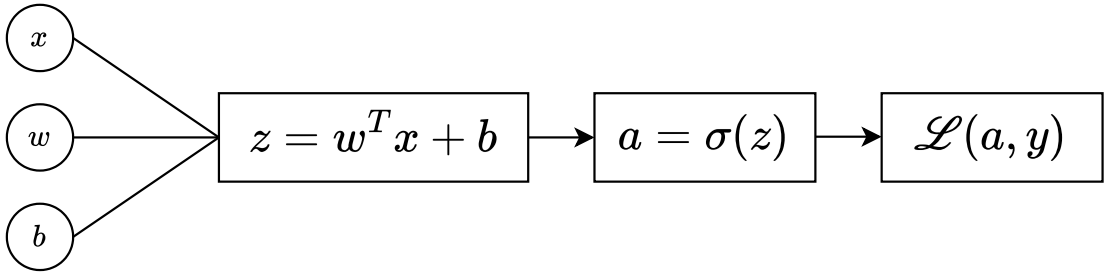


Logistische Regression im Berechnungsgraph

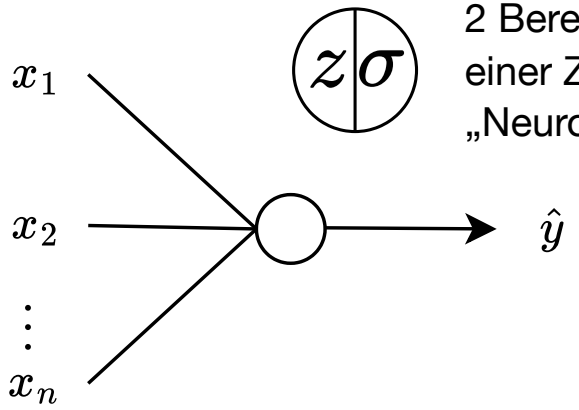
Berechnungsgraph:



Mit Vektoren:



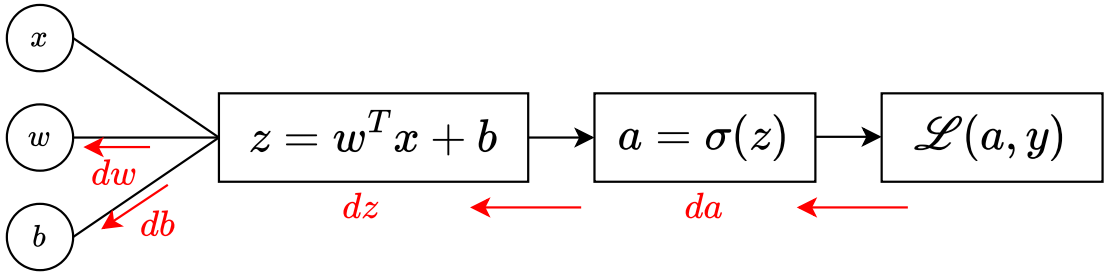
Ganz kompakt:



2 Berechnungsschritte in einer Zelle / einem „Neuron“ (Feed Forward)

Gradientenabstieg bei der logistischen Regression

Berechnung der Ableitung im Berechnungsgraph




```
Für jede Iteration:
    // Initialisierung
    I = 0
    dw_1 = 0
    dw_2 = 0
    db = 0

    // durch alle Trainingsbispiele gehen
    Für i = 1 bis m:
        // Prediction a für i-tes Trainingsbeispiel berechnen
        z = vektormult(w, x[i]) + b
        a = sigmoid(z)

        // Fehler berechnen
        I = I - (y[i] * log(a) + (1 - y[i])*log(1 - a))

        // dz berechnen
        dz = a - y[i]

        // partielle Ableitungen berechnen und aufsummieren
        dw_1 = dw_1 + x[i][1] * dz
        dw_2 = dw_2 + x[i][2] * dz
        db = db + dz
```

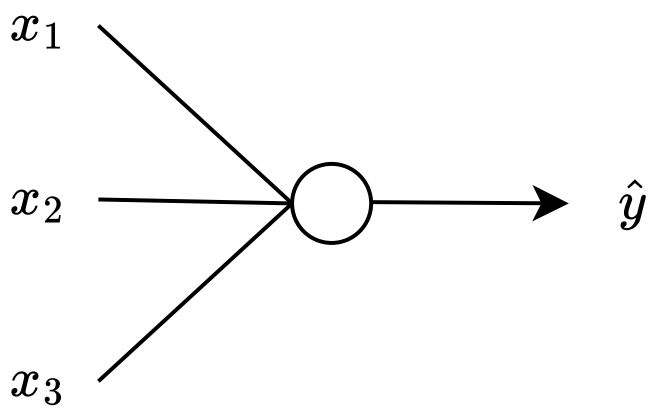
```
// Durchschnitt über alle Trainingsbeispiele berechnen
I = I / m
dw_1 = dw_1 / m
dw_2 = dw_2 / m
db = db / m

// Parameter aktualisieren
w_1 = w_1 - alpha * dw_1
w_2 = w_2 - alpha * dw_2
b = b - alpha * db
```

Themen

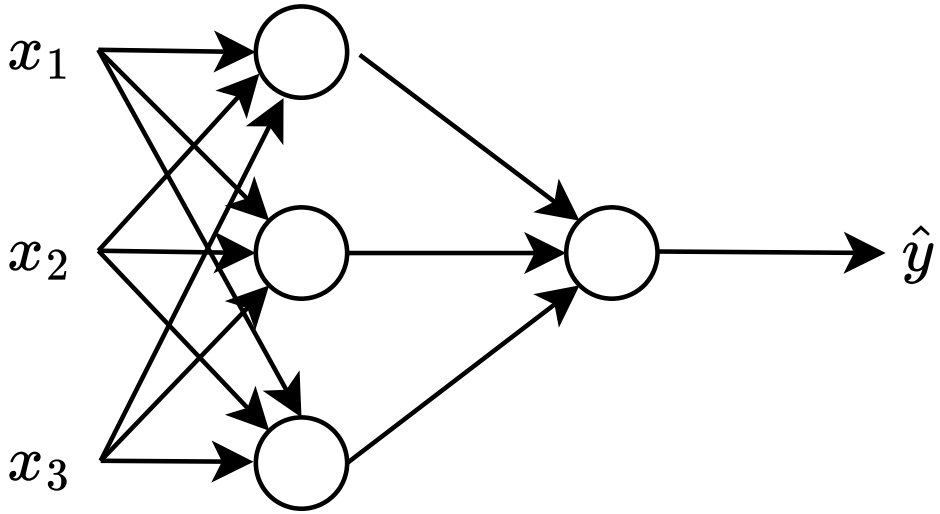
- Von der logistischen Regression zum neuronalen Netz
- Architekturvarianten
- Vorwärtskopplung (Feed Forward)
- Aktivierungsfunktionen (Activation functions)
- Ausgabefunktionen (Output Functions)
- Entwurfsentscheidungen bei der Gestaltung neuronaler Netze
- Gradientenabstieg in neuronalen netzen

Von der logistischen Regression zum neuronalen Netz

Idee: Mehrere Schichten



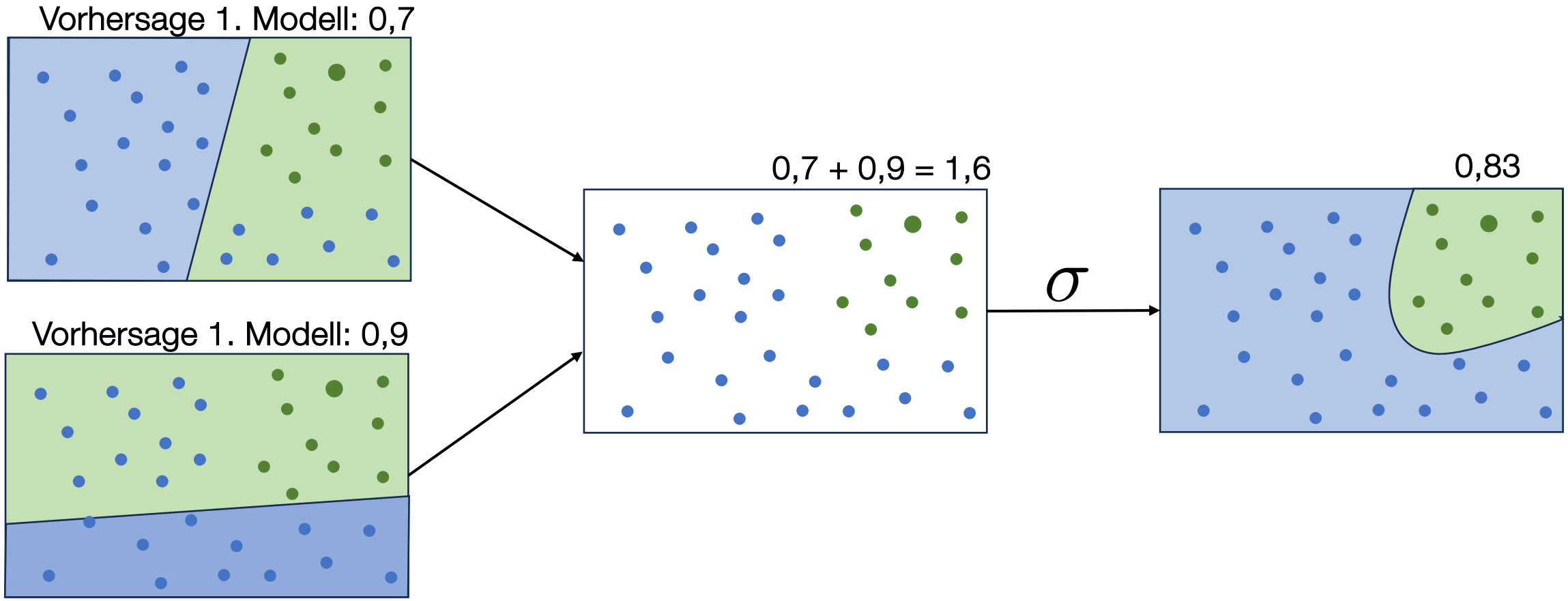
1 Neuron



2 Schichten
mit insgesamt 4 Neuronen

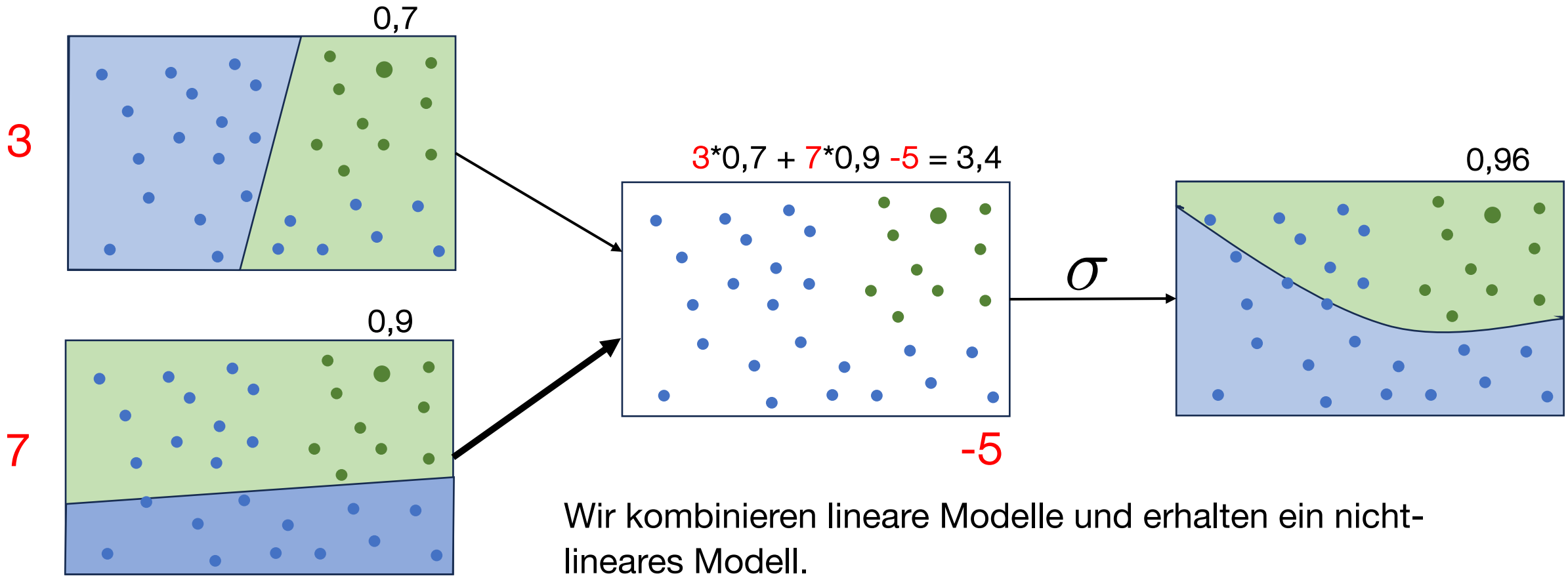
Veranschaulichung: Kombination von Modellen (1)

Addieren von Modellen



Veranschaulichung: Kombination von Modellen (2)

Addieren von Modellen mit Gewichtung und Verschiebung (Bias)



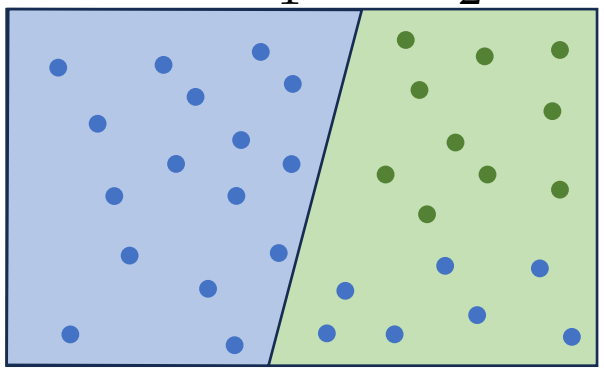
Wir kombinieren lineare Modelle und erhalten ein nicht-lineares Modell.

Veranschaulichung: Kombination von Modellen (3)

Beispiel:

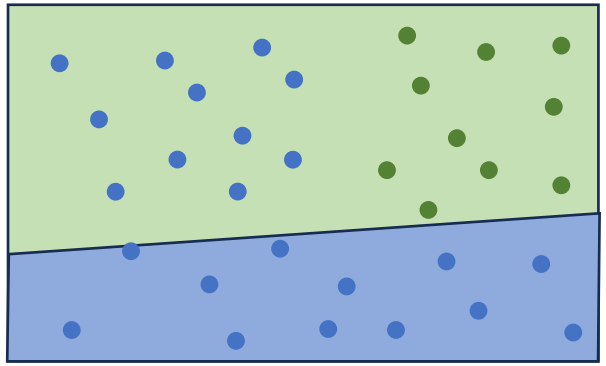
3

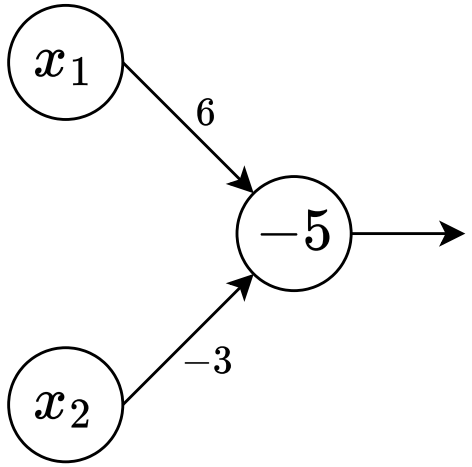
$6x_1 - 3x_2 - 5$

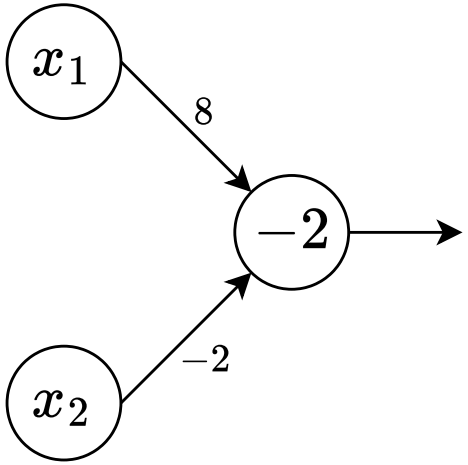


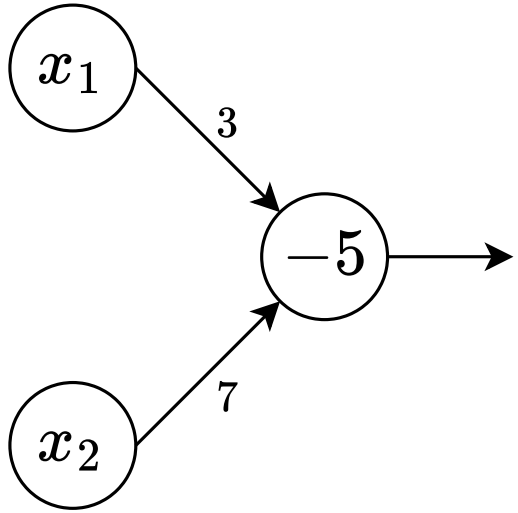
7

$8x_1 - 2x_2 - 2$





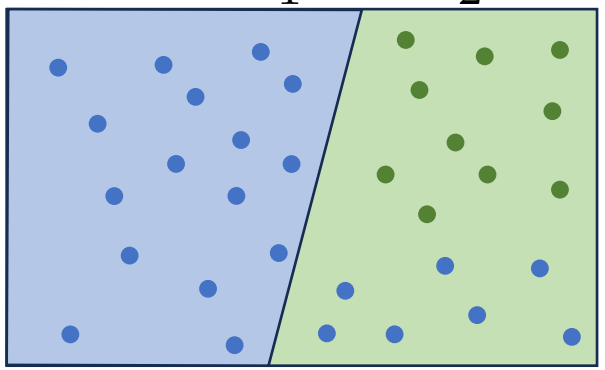




Veranschaulichung: Kombination von Modellen (4)

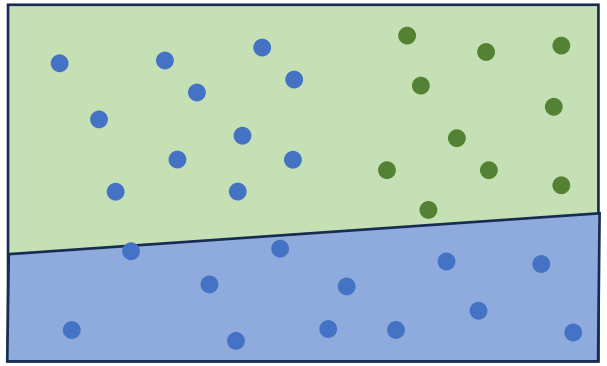
3

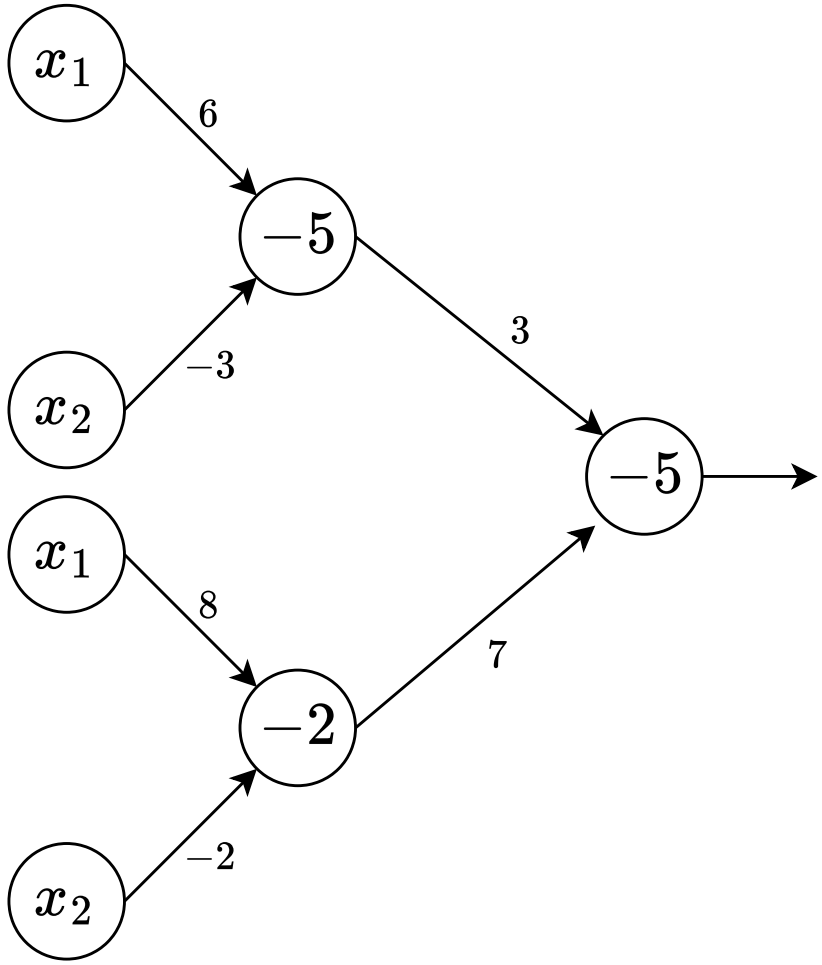
$6x_1 - 3x_2 - 5$



7

$8x_1 - 2x_2 - 2$

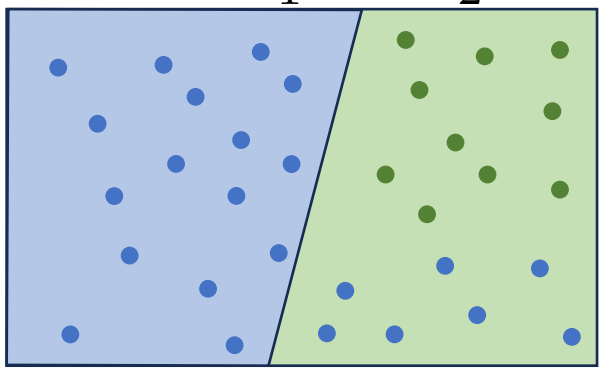




Veranschaulichung: Kombination von Modellen (5)

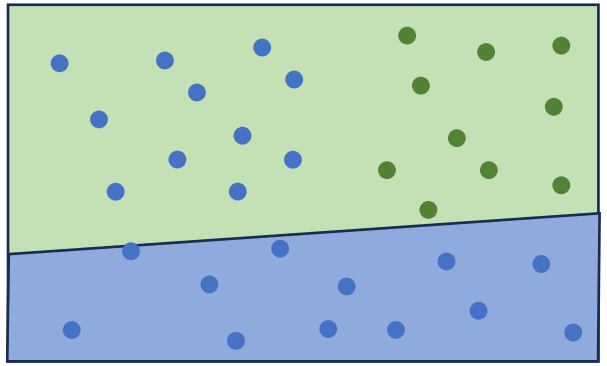
3

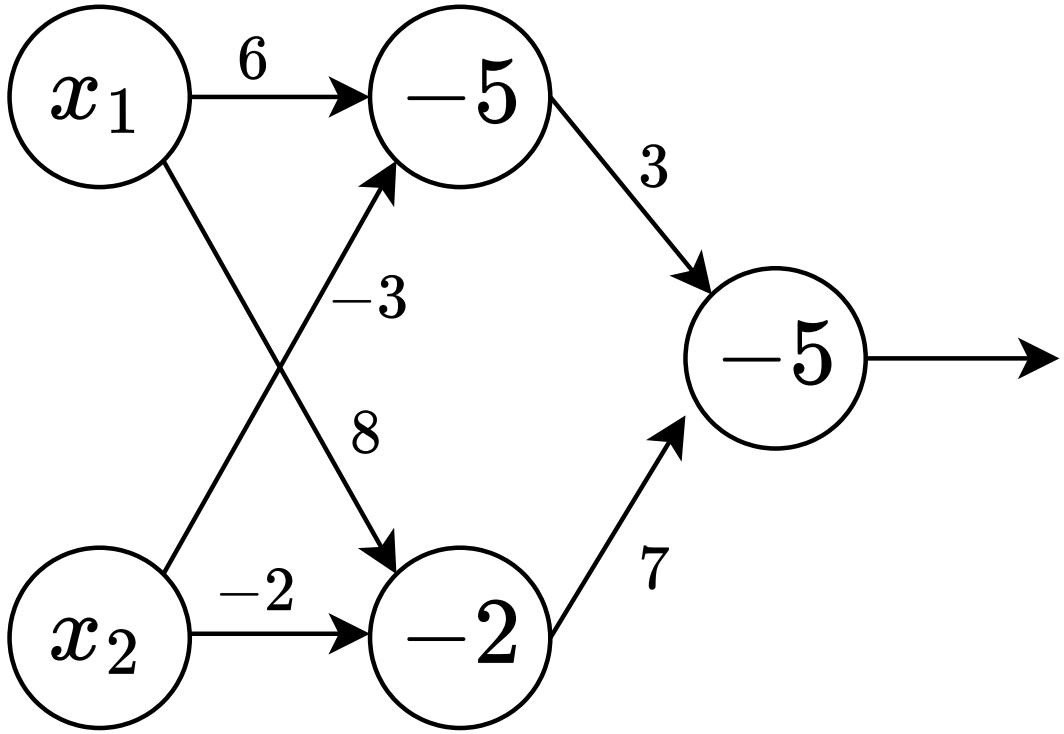
$6x_1 - 3x_2 - 5$



7

$8x_1 - 2x_2 - 2$

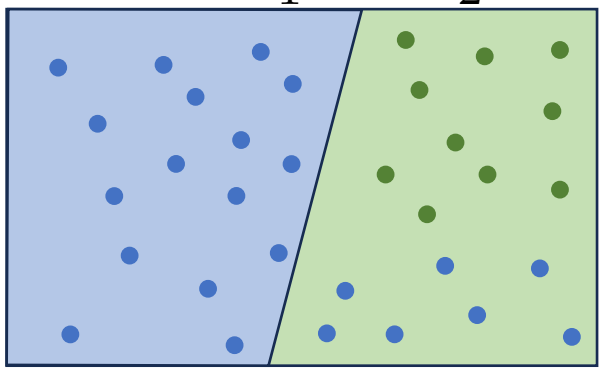




Alternative Darstellung

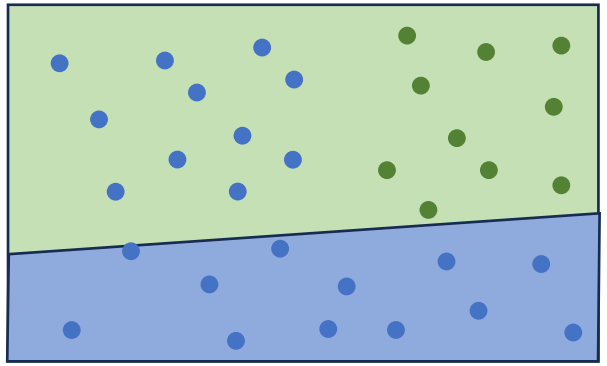
3

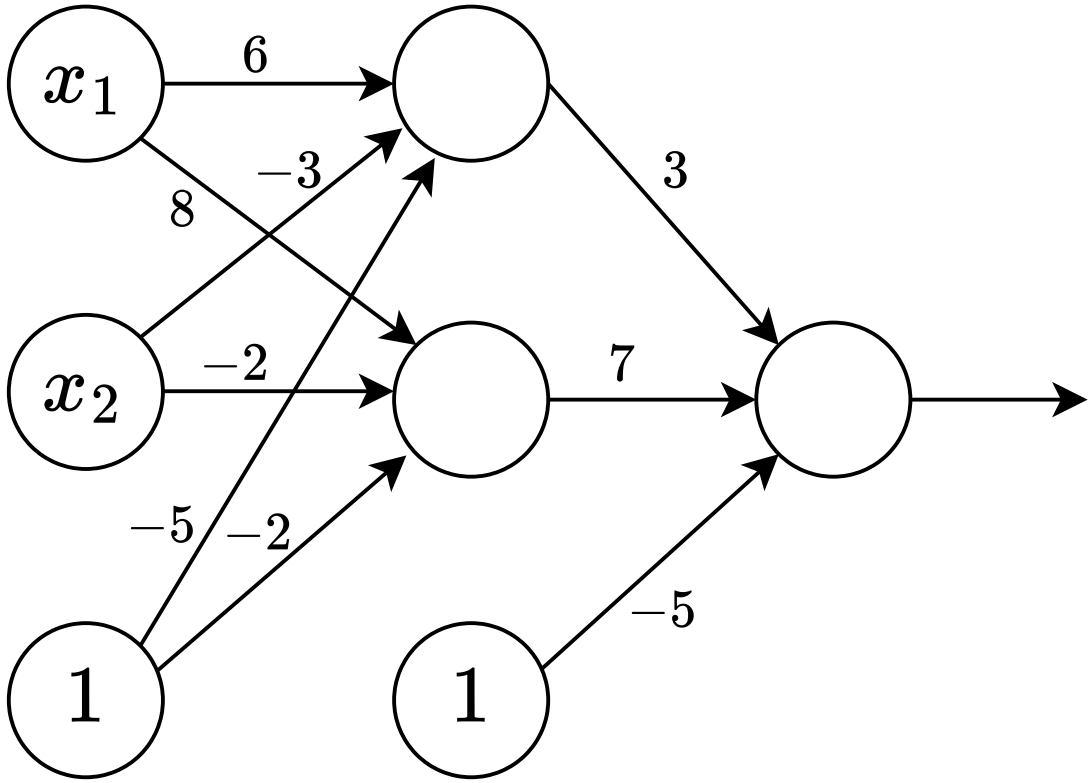
$6x_1 - 3x_2 - 5$



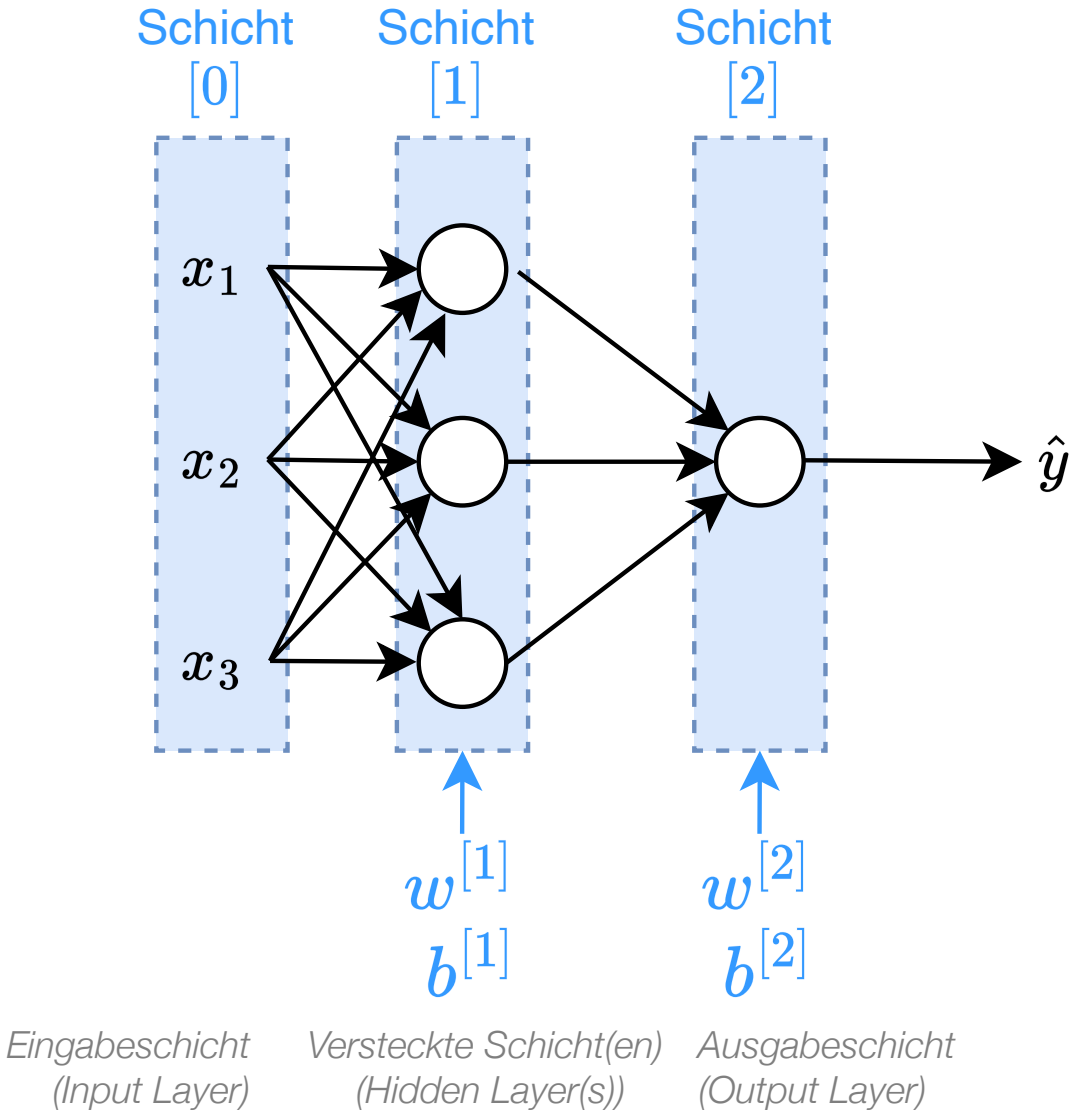
7

$8x_1 - 2x_2 - 2$





Mehrere Schichten formalisiert



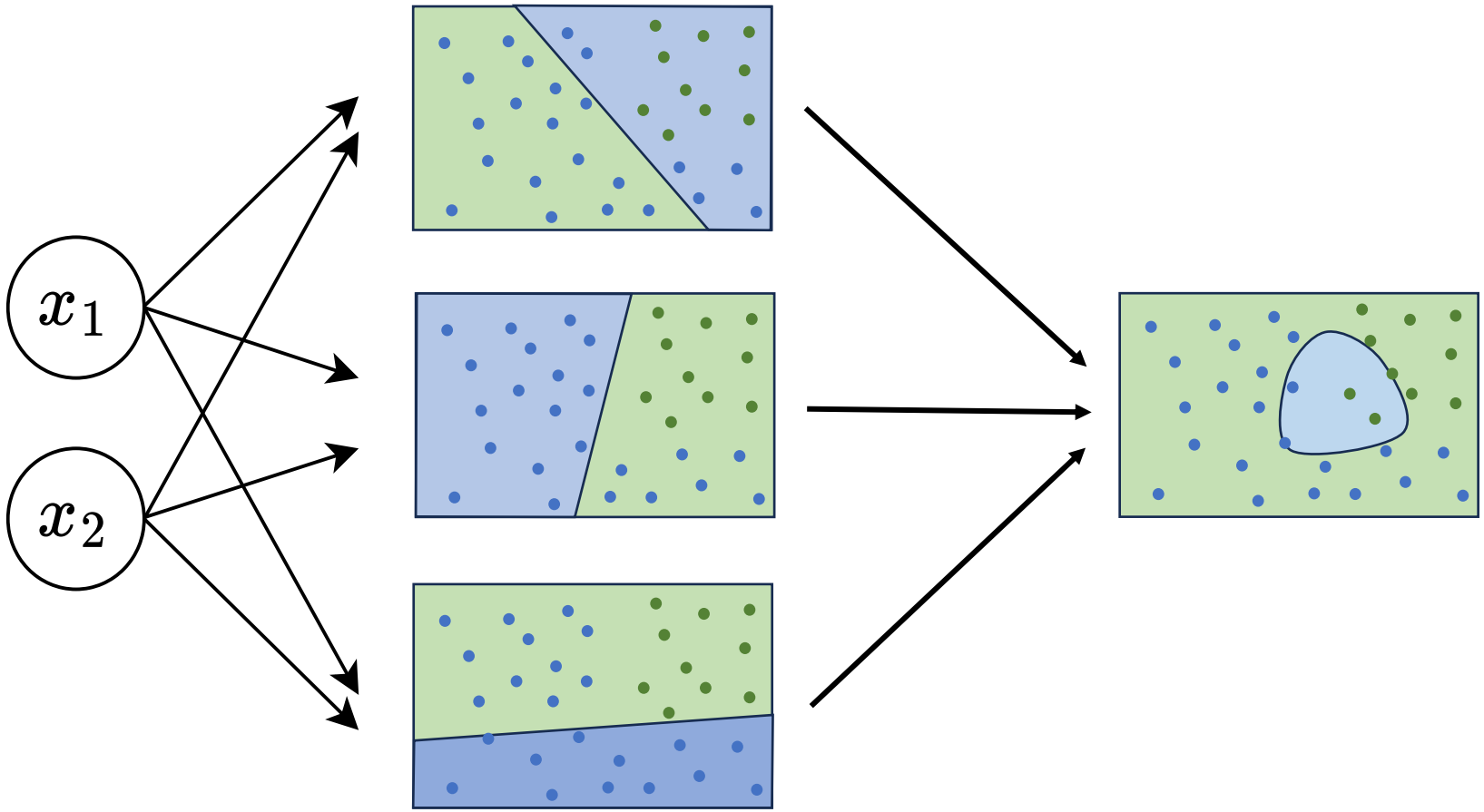
Warum heißt die mittlere Schicht „versteckte Schicht“ (hidden layer)?

Die Eingabewerte und die Ausgabewerte gehören zum Trainingsdatensatz, die Werte dazwischen nicht, sie sind versteckt.

Architekturvarianten

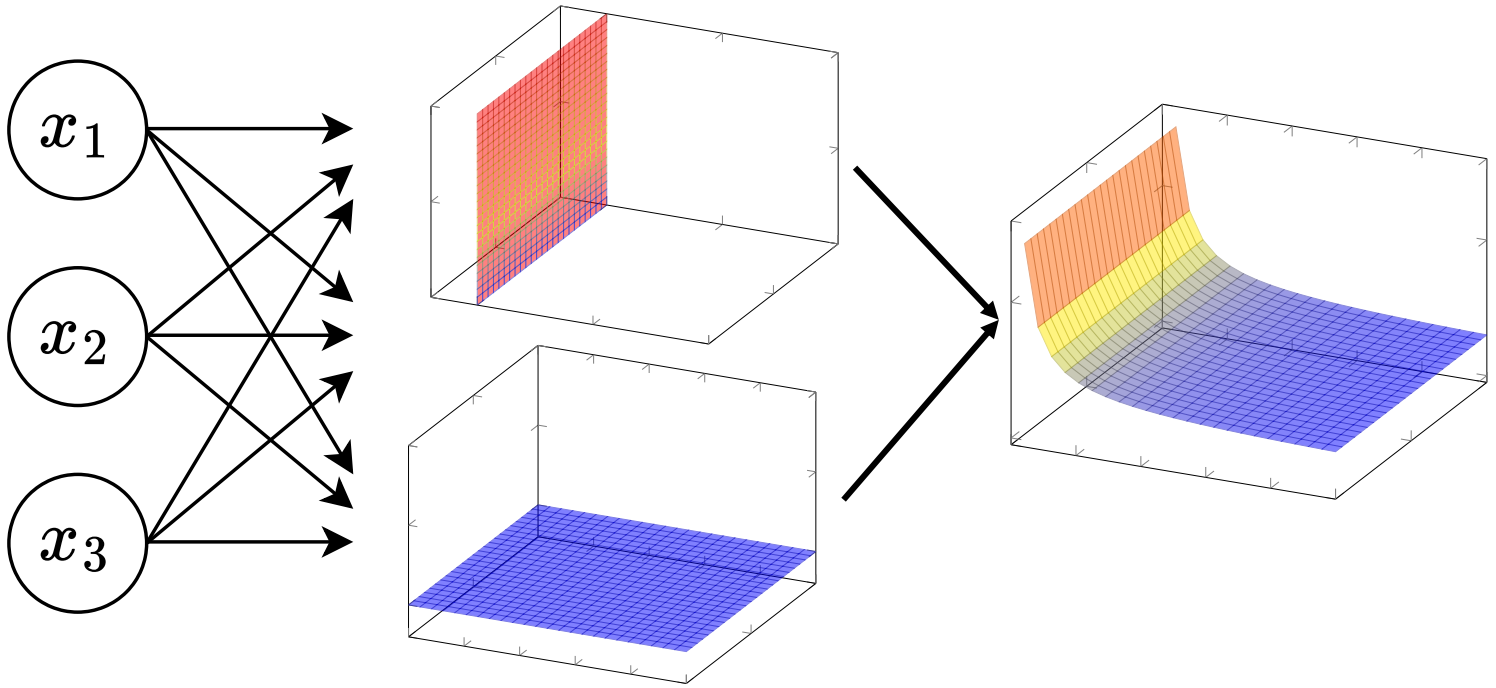
Mehr Knoten in der versteckten Schicht

Durch mehr Knoten in der versteckten Schicht werden komplexere Entscheidungsgrenzen ermöglicht.



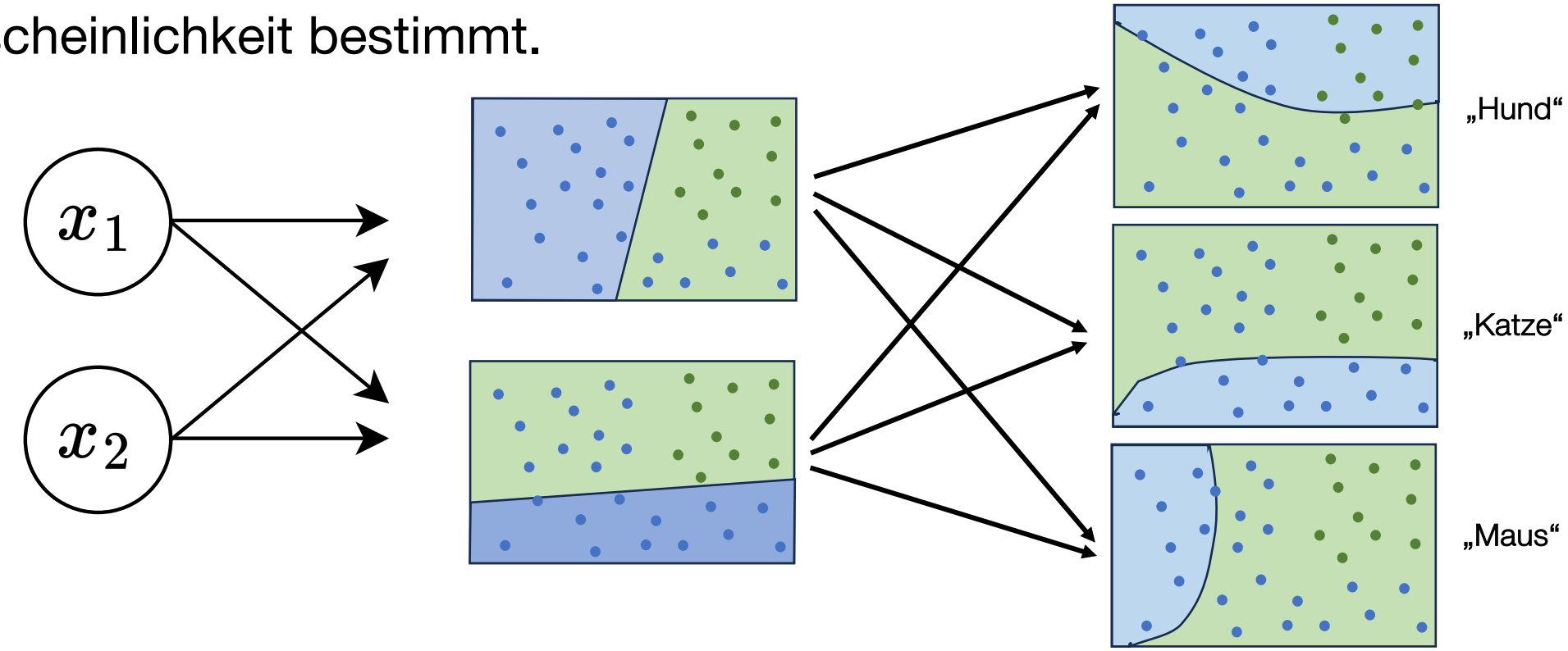
Mehr Eingabeknoten

- Wenn wir mehr (z.B. n) Eingabeknoten haben, ergeben sich höherdimensionale (n-dimensionale) Daten.
- **Beispiel:** 3-dimsonsionale Daten



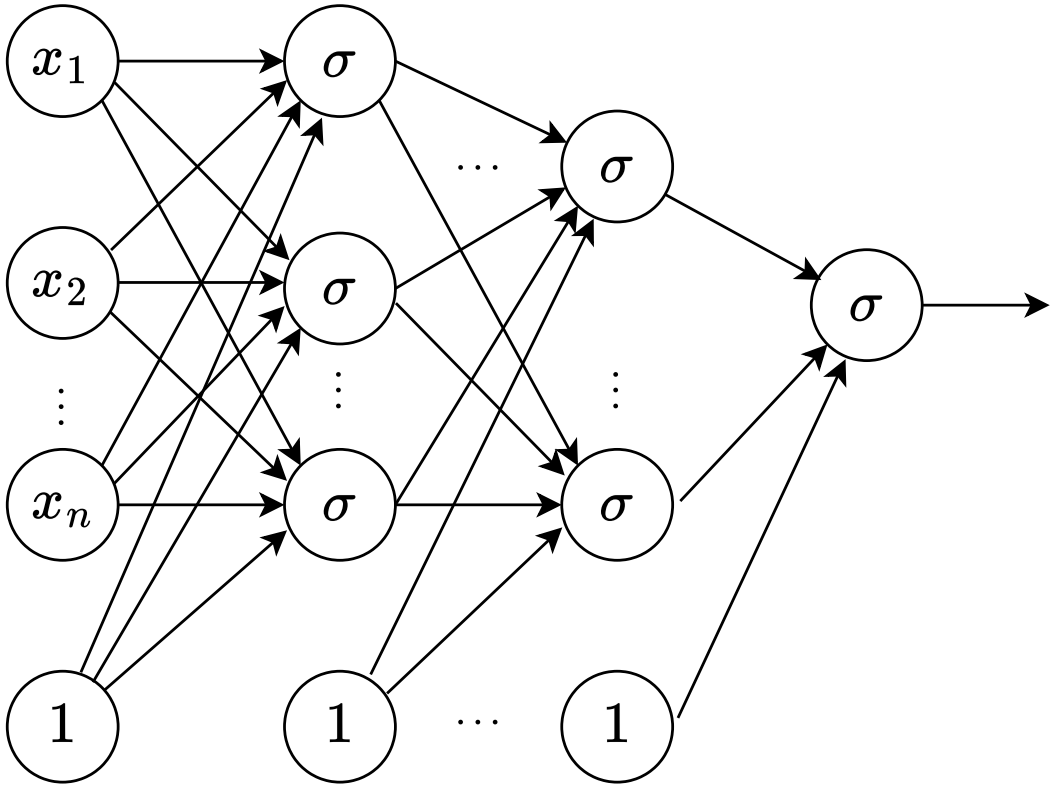
Mehr Ausgabeknoten

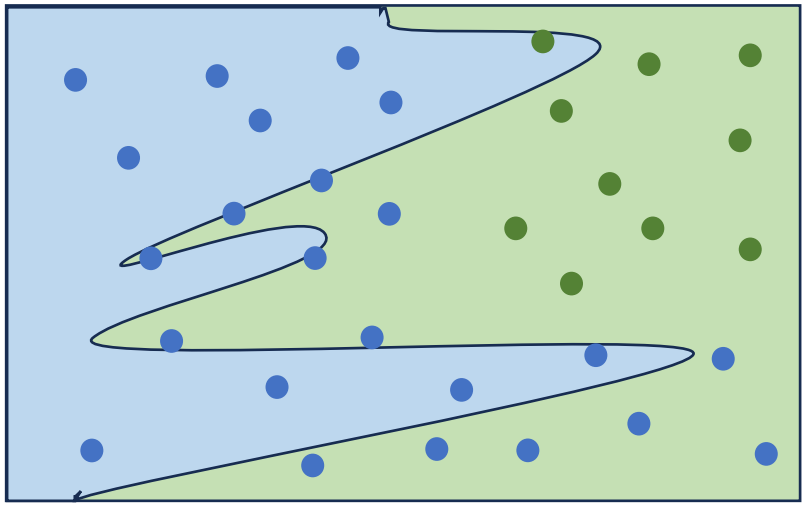
Durch mehrere Ausgabekonten ergibt sich ein Mehrklassen-Klassifikator, der für jede der möglichen Ausgabeklassen eine Wahrscheinlichkeit ausgibt. Über eine Softmax-Funktion wird dann die Klasse mit der höchsten Wahrscheinlichkeit bestimmt.



Mehr versteckte Schichten

Durch mehr versteckte Schichten entstehen tiefe neuronale Netze (Deep Neural Networks). Diese ermöglichen die Repräsentation komplexer nichtlinearer Modelle.





Vorwärtskopplung (Feed Forward)

Definition: Vorwärtskopplung (Feed Forward)

Vorwärtskopplung (Feed Forward) beschreibt einen Prozess, bei dem Informationen durch die Schichten eines neuronalen Netzes in nur eine Richtung fließen – von der Eingabeschicht über die verborgenen Schichten (falls vorhanden) zur Ausgabeschicht.

- Jede Schicht empfängt Eingangssignale, führt Berechnungen durch (z.B. durch Anwenden von Gewichten, Bias-Additionen und Aktivierungsfunktionen) und gibt das Ergebnis an die nächste Schicht weiter, bis die Ausgabeschicht erreicht ist.
- keine Rückkopplungen oder Schleifen, Informationen bewegen sich stetig vorwärts
  ➔ einfache und effektive Modellierung von Beziehungen zwischen Eingabe- und Ausgabedaten

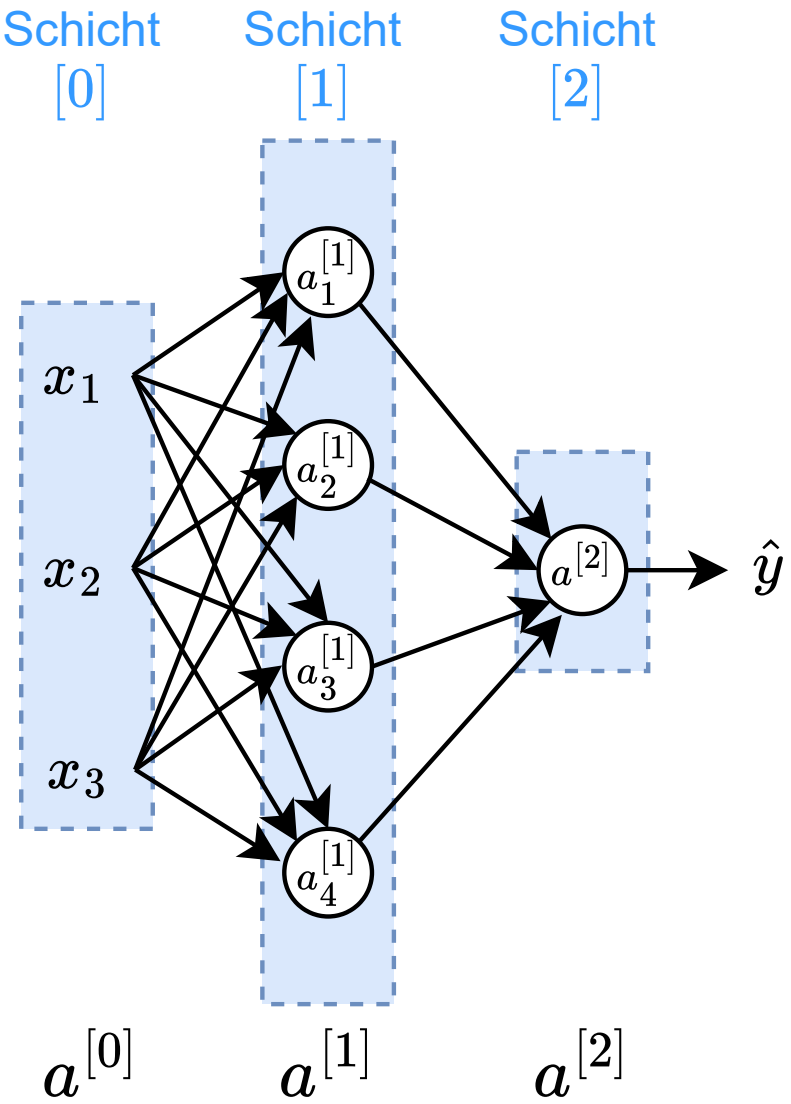
Notation

- **Beispiel:** Neuronales Netz mit 2-Schichten
  (Die Eingabeschicht wird nicht mitgezählt.)

- a steht für „Activations“ (die Werte, die von einer Schicht an die nächste weitergegeben werden)

$$a_i^{[l]}$$

l: Schicht (layer)
i: Knoten / Neuron innerhalb der Schicht



$$a^{[0]} = x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

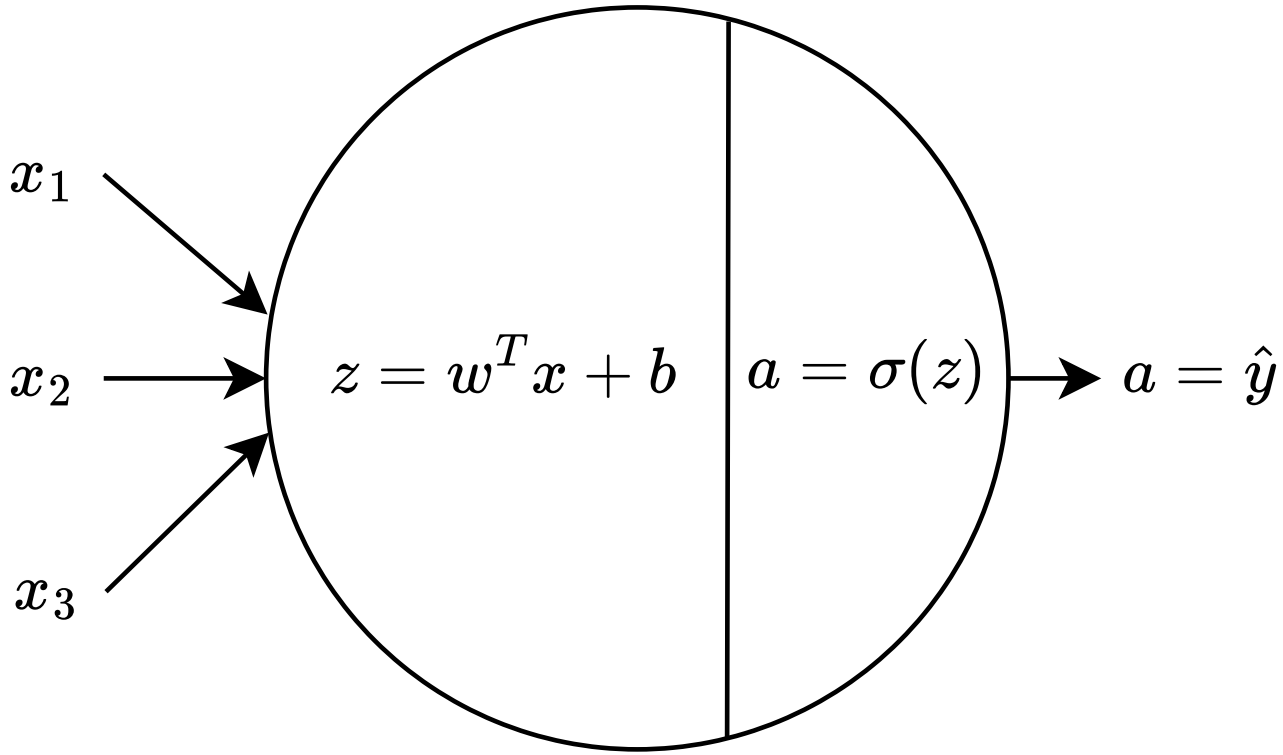
$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

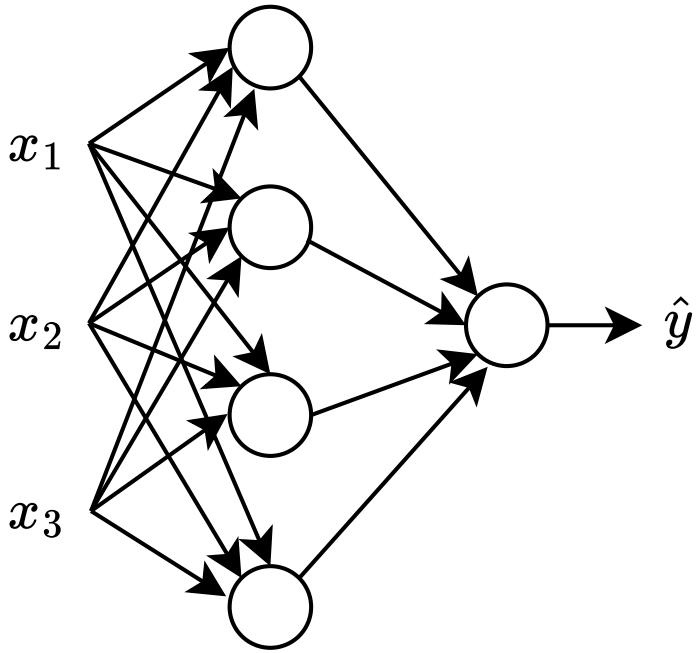
$$a^{[2]} = \hat{y}$$

Berechnung des Ausgabewertes eines Neurons

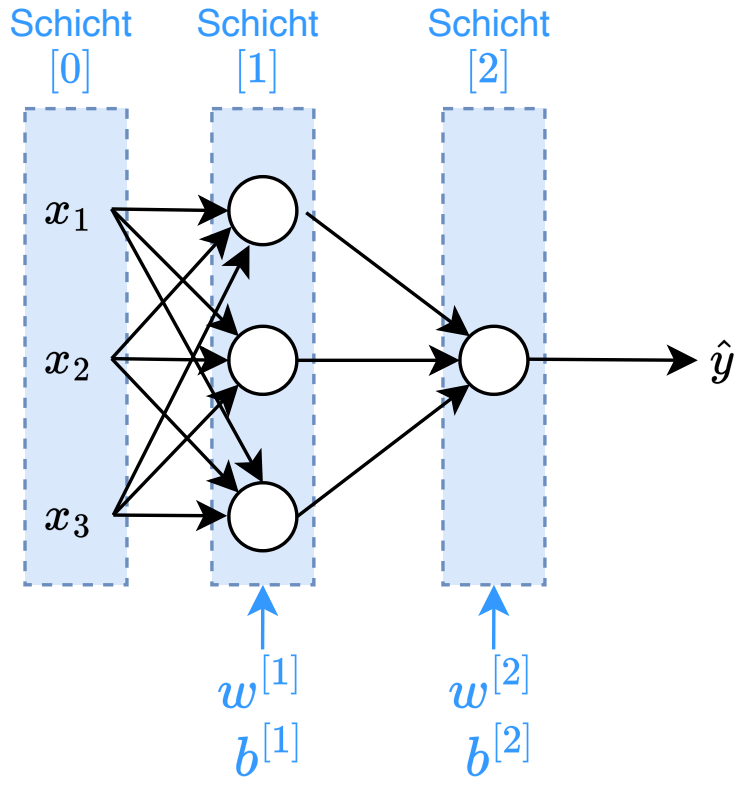
Jedes Neuron (d.h. jeder Kreis) besteht aus zwei Funktionen:

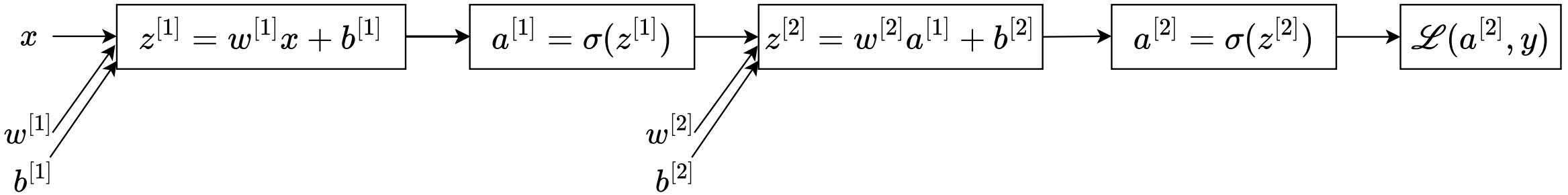
1) die Berechnung der Gewichteten Summen der Eingaben plus Bias (z)
2) die Berechnung der Aktivierungsfunktion (a)



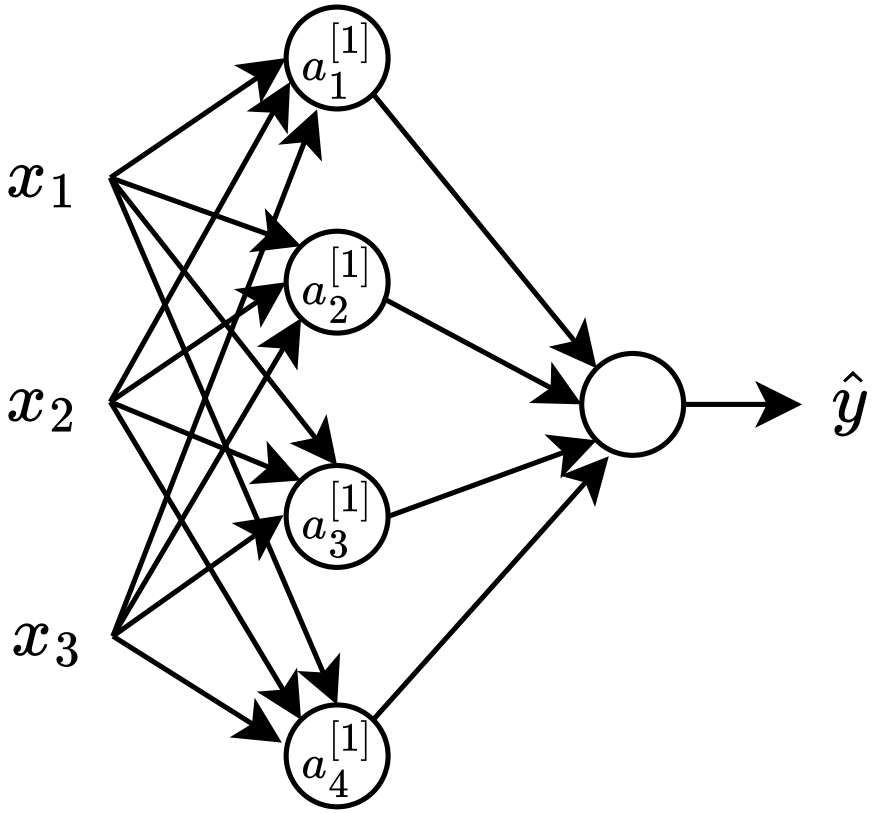


Berechnungsgraph über 2 Schichten





Berechnung der Ausgabewerte der versteckten Schicht



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma(z_4^{[1]})$$

…als Matrixoperationen

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$= \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$$= \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & w_{23}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} & w_{43}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

Gewichte des 1. Neurons der versteckten Schicht

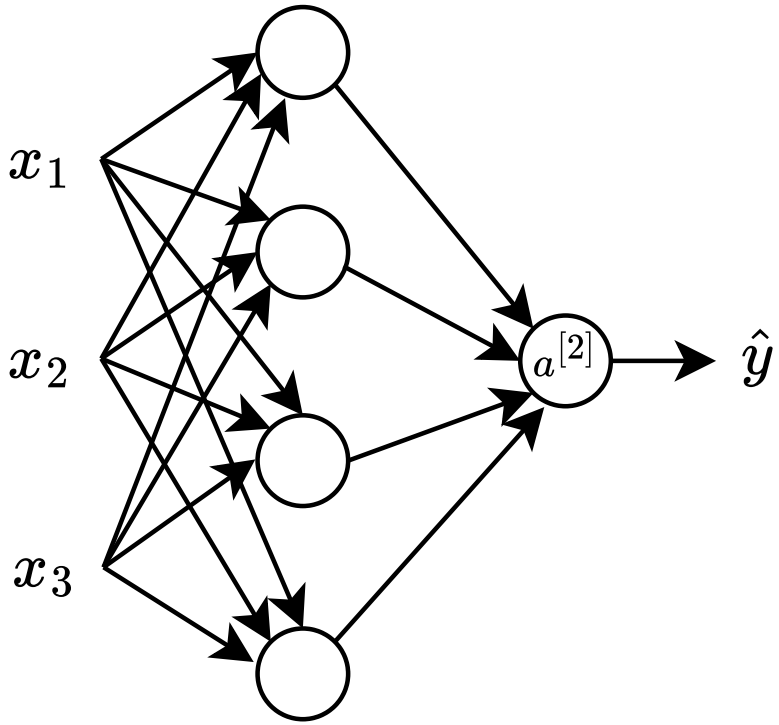
$$= \begin{bmatrix} w_1^{[1]T}x + b_1^{[1]} \\ w_2^{[1]T}x + b_2^{[1]} \\ w_3^{[1]T}x + b_3^{[1]} \\ w_4^{[1]T}x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \\ \sigma(z_4^{[1]}) \end{bmatrix} = \sigma(z^{[1]})$$

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

Berechnung des Ausgabewerts der Ausgabeschicht



$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Quiz

Welche Dimensionen haben die verschiedenen Vektoren und Matrizen in unserem neuronalen Netz mit 2 Schichten aus dem Beispiel?

Beachte: Die Matrix $\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix}$ hat die Dimensionen (3,2).

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
(,) (,) (,) (,)

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
(,) (,) (,) (,)

$$a^{[1]} = \sigma(z^{[1]})$$
(,) (,)

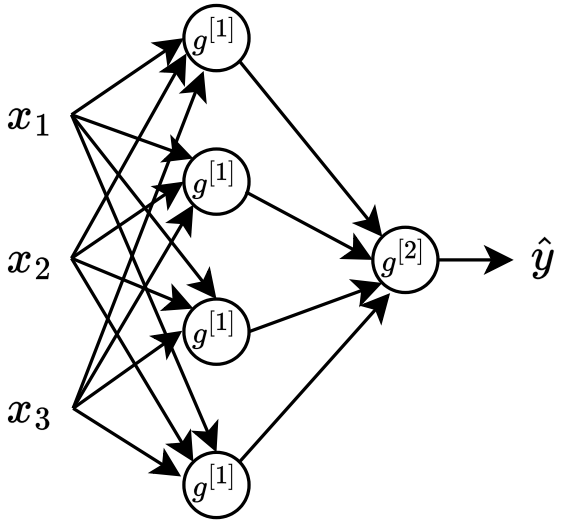
$$a^{[2]} = \sigma(z^{[2]})$$
(,) (,)

Aktivierungsfunktionen
(Activation Functions)

Aktivierungsfunktion

Eine Aktivierungsfunktion ist eine mathematische Funktion, die in einem künstlichen Neuron verwendet wird, um die gewichtete Summe der Eingaben und den Bias in einen Ausgabewert zu transformieren, der dann an die nächsten Schichten im Netzwerk weitergeleitet wird.

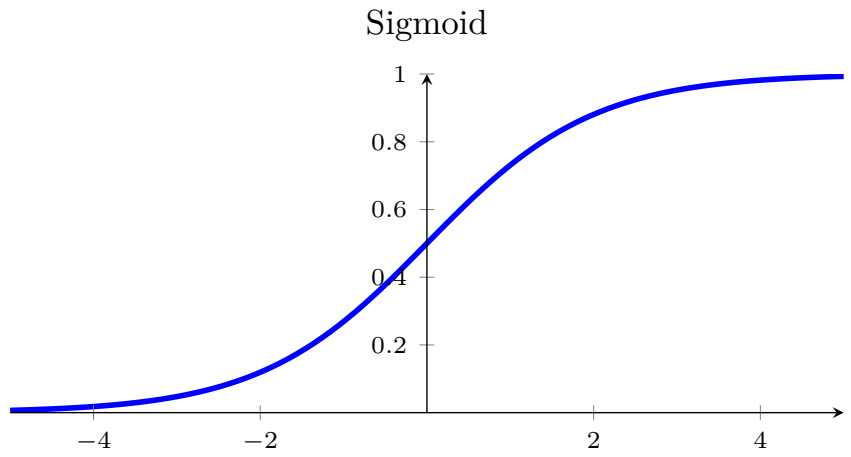
- Wir haben bisher als Aktivierungsfunktion die logistische Funktion (sigmoid) verwendet. Dies ist jedoch nicht die einzige Option.
- Auch ist es möglich, in einem neuronalen Netzwerk für die einzelnen Schichten unterschiedliche Aktivierungsfunktionen zu verwenden.



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$

$g^{[i]}$ bezeichne die Aktivierungsfunktion der i. Schicht.

Option 1: Sigmoid (Logistische Funktion)



Die logistische Funktion (Sigmoid) ist definiert als

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

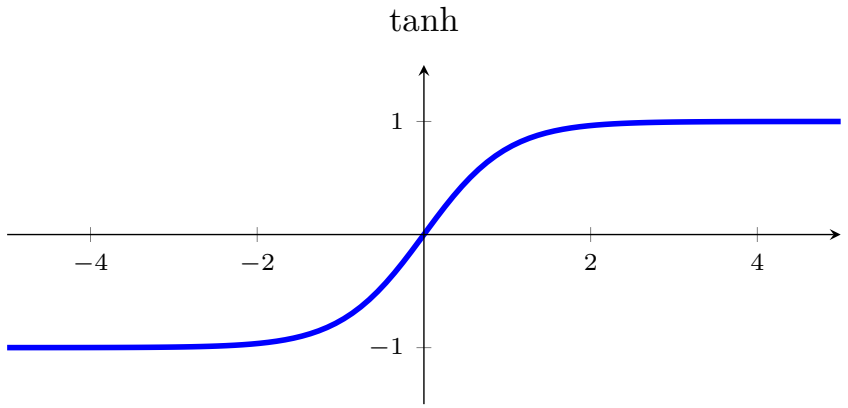
- Die Sigmoid-Funktion gibt Werte im Bereich zwischen 0 und 1 zurück ➔ Sie ist daher besonders nützlich in der Ausgabe-Schicht für Modelle, bei denen eine Wahrscheinlichkeit als Ausgabe erwartet wird.
- Sie wird jedoch seltener in den verstecken Schichten verwendet, da sie zu Problemen wie dem verschwindenden Gradienten (Vanishing Gradient, siehe unten) führen kann und es effizientere Funktionen für das Training gibt .

Verschwindender Gradient (Vanishing Gradient)

Das Phänomen des verschwindenden Gradienten (Vanishing Gradient) kann in neuronalen Netzwerken mit vielen Schichten (Deep Neural Networks) auftreten.

- **Grund:** Aktivierungsfunktionen (wie sigmoid) mit Bereichen, in denen ihre Ableitungen sehr klein sind (nahe 0). Wenn der Eingabewert einer solchen Funktion sehr groß oder sehr klein ist, wird die Ableitung der Funktion fast null.
- Die wiederholte Multiplikation dieser kleinen Werte im Rahmen der Kettenregel führt dazu, dass der Gradient für die vorderen Schichten des Netzwerks sehr klein werden.
- ➔Dies führt wiederum dazu, dass die Gewichte in den vorderen Schichten des Netzwerks kaum aktualisiert werden, was das Lernen effektiv verlangsamt oder sogar stoppt.

Option 2: tanh (Hyperbolische Tangensfunktion)

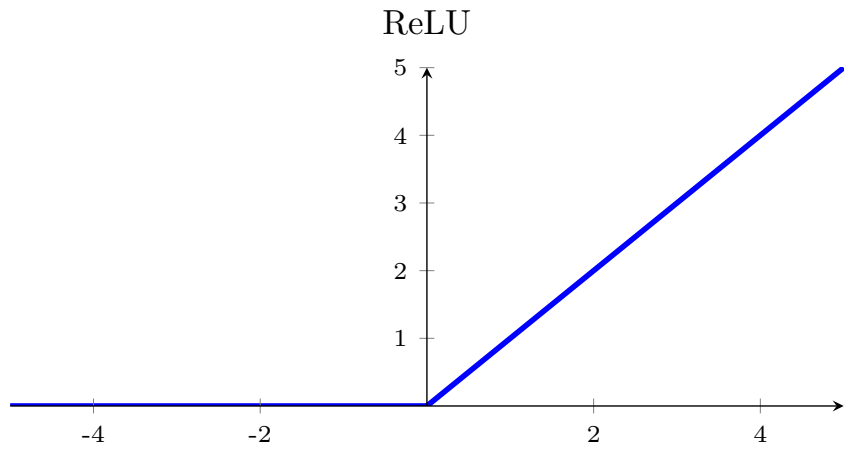


Die Hyperbolische Tangens-Funktion ist definiert als

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

- Die tanh-Funktion gibt Werte zwischen -1 und 1 zurück.
- Sie ist eine skalierte Version der Sigmoid-Funktion und tendiert dazu, in mittleren Schichten von Netzwerken besser zu funktionieren, da sie datenzentriert (um die Null herum) ist.
- Das Problem des verschwindenden Gradienten kann trotzdem auftreten.

Option 3: ReLU (Rectified Linear Unit)

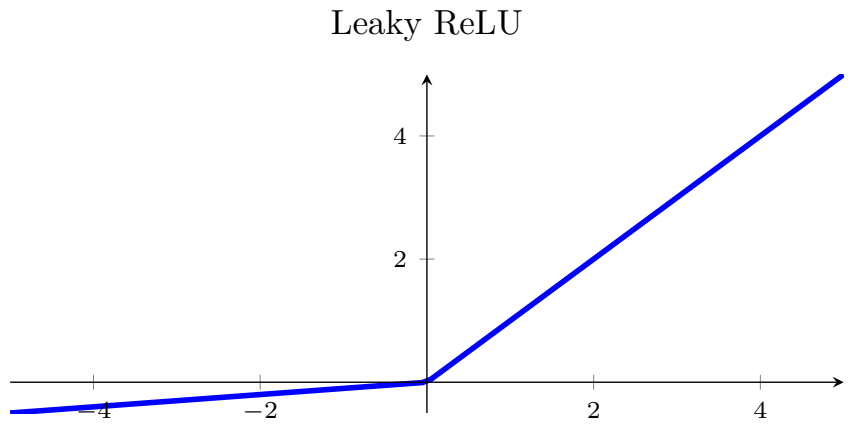


Die ReLU (Rectified Linear Unit) Funktion wird definiert als:

$$\mathrm{ReLU}(x) = \max(0, x)$$

- ReLU wird häufig verwendet, da die Funktion einfach und effizient berechnet werden kann und beim Training gut funktioniert.
- Die Ableitung für x=0 ist mathematisch nicht definiert. Dies spielt in der Praxis jedoch keine Rolle, man setzt den Ableitungswert für x=0 auf 1 oder 0.

Option 4: Leaky ReLU



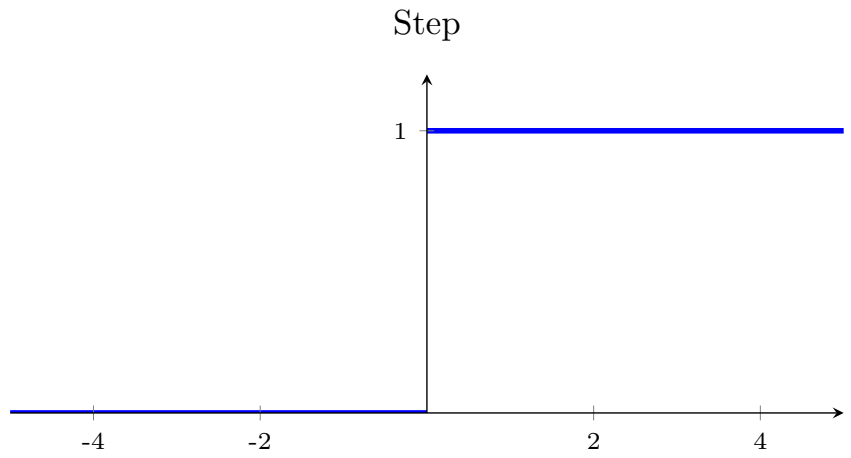
Die Leaky ReLU-Funktion ist definiert als

$$f(x) = \begin{cases} x & \text{für } x > 0, \\ \alpha x & \text{für } x \leq 0, \end{cases}$$

wobei α ein kleiner positiver Faktor ist, typischerweise $\alpha = 0.01$.

- Leaky ReLU funktioniert oft etwas besser als ReLU, wird in der Praxis jedoch selten eingesetzt.

Option 5: Step

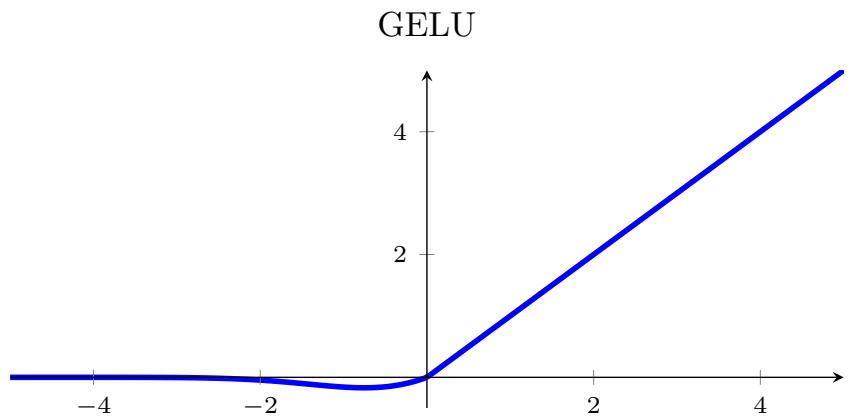


Die Step-Funktion $H(x)$, auch bekannt als Heaviside-Funktion, wird wie folgt definiert:

$$H(x) = \begin{cases} 0 & \text{für } x < 0 \\ 1 & \text{für } x \geq 0 \end{cases}$$

- Die Step-Funktion wurde historisch in den ersten neuronalen Netzen (z.B. dem Perzeptron) eingesetzt, spielt heute jedoch als Aktivierungsfunktion keine Rolle mehr.
- Nachteil: Da die Step-Funktion eine binäre Ausgabe liefert, gehen alle Informationen über die Stärke der Eingabe (außer dass sie über oder unter einem bestimmten Schwellenwert liegt) beim Training verloren.

Option 6: GELU (Gaussian Error Linear Unit)



Die GELU (Gaussian Error Linear Unit) Funktion wird definiert als:

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

wobei

$$\Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

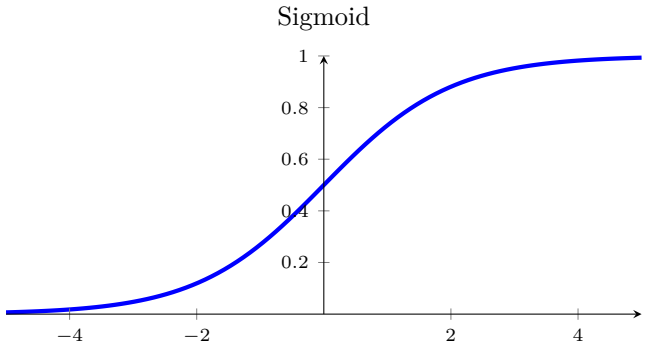
Die Fehlerfunktion (error function), bezeichnet als erf, ist definiert als:

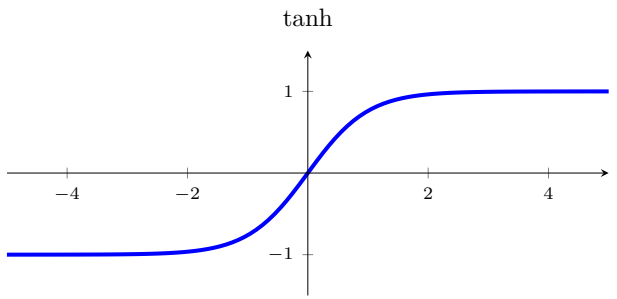
$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

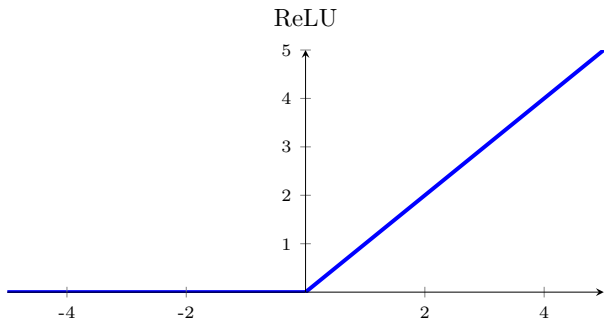
- GELU ist eine Aktivierungsfunktion, die sich vor allem in NLP-Anwendungen und Transformer-Modellen bewährt hat.
- Nachteil ist ein hoher Rechenaufwand.

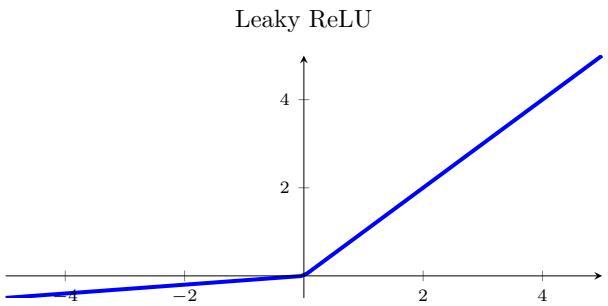
Zusammenfassung: Auswahl an Aktivierungsfunktionen

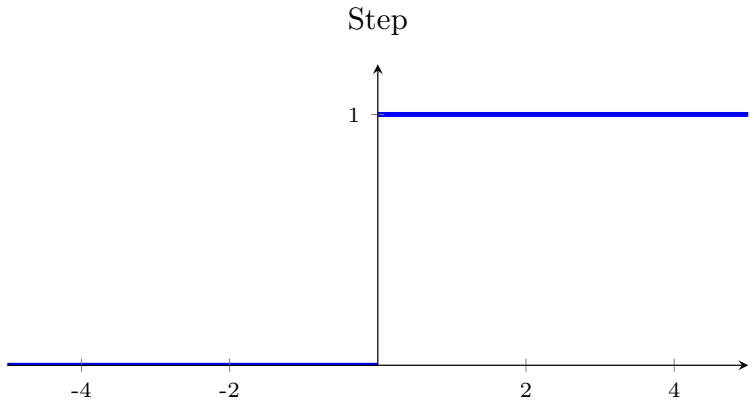
- Es gibt eine Reihe möglicher Aktivierungsfunktionen. Die wichtigsten sind:

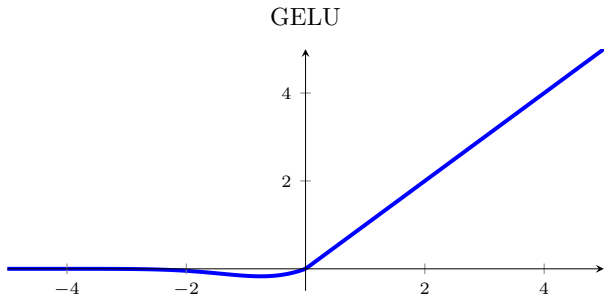












- Bei der Auswahl der Aktivierungsfunktion sollte man etwas experimentieren und ausprobieren, wie sie für die vorliegenden Daten funktionieren.

Eigenschaften von Aktivierungsfunktionen

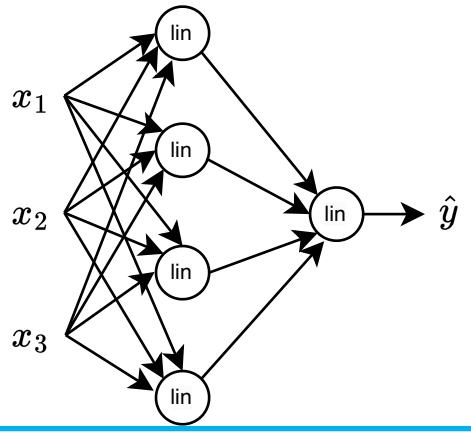
Aktivierungsfunktionen sollten folgende Eigenschaften aufweisen:

1) Nichtlinearität
2) Stetige Differenzierbarkeit
3) Monotone Steigung
4) Identitätsfunktion am Ursprung

Eigenschaft 1: Nichtlinearität

Warum sollte die Aktivierungsfunktion keine lineare Funktion sein?

- Wenn alle Aktivierungsfunktionen linear wären, würde sich als Gesamtfunktion einfach wieder ein lineares Modell ergeben. Komplexe, nichtlineare Daten könnten nicht modelliert werden.
- **Beispiel:** Wir verwenden als Aktivierungsfunktion g(x) = x (also die Identitätsfunktion als lineare Aktivierungsfunktion)



$$
\begin{aligned}
a^{[1]} &= z^{[1]} = W^{[1]}x + b^{[1]} \\
a^{[2]} &= z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} \\
&= W^{[2]}W^{[1]}x + (W^{[2]}b^{[1]} + b^{[2]}) \\
&= W'x + b'
\end{aligned}
$$

➔ Verstecke Schichten sollten (bis auf einige spezielle Anwendungsfälle) keine lineare Aktivierungsfunktionen verwenden.

Eigenschaft 2: Stetige Differenzierbarkeit

Warum sollte die Aktivierungsfunktion stetig differenzierbar sein (d.h. sie sollte keine „Ecken“ oder „Spitzen“ haben)?

- Dies ist notwendig, damit der Gradientenabstieg (Gradient Descent) funktioniert.

Eigenschaft 3: Monotone Steigung

Warum sollte die Aktivierungsfunktion eine monoton steigende Funktion sein, d.h. $\text{für } x_1 \text{ und } x_2 \text{ mit } x_1 < x_2 \text{ gilt: } g(x_1) <= g(x_2)$?

- Dies muss gelten, da ansonsten im Lernprozess Mehrdeutigkeiten und Oszillationen auftreten können: Es könnte Bereiche geben, in denen eine höhere Eingabe zu einer niedrigeren Ausgabe führen würde.
- Das könnte zu Situationen führen, in denen das Netzwerk keine eindeutige Richtung für das Update der Gewichte hat oder der Lernalgorithmus könnte zwischen verschiedenen Gewichten hin und her springt, ohne eine konvergente Lösung zu finden.

Eigenschaft 4: Identitätsfunktion am Ursprung

Warum sollte die Aktivierungsfunktion um den Ursprung herum der Identitätsfunktion $g(x) = x$ entsprechen?

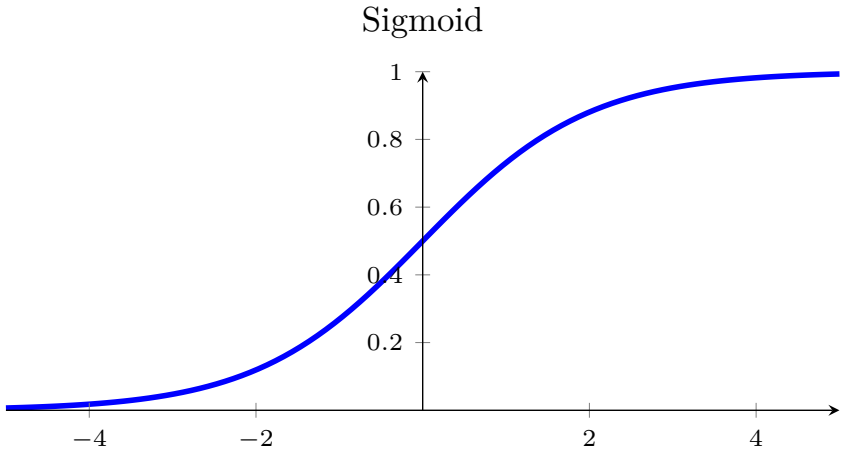
Dies bietet folgende Vorteile:

- **Effizienz im Lernen:** Lineare Aktivierung in der Nähe von Null hält Gradienten konstant, fördert effektive Gewichtsanpassungen in frühen Trainingsphasen.
- **Reduzierung des Verschwindenden Gradienten**: Mindert das Problem des verschwindenden Gradienten in tiefen Netzwerken, indem es den Informationsfluss durch die Schichten unterstützt.
- **Symmetrische Behandlung:** Gleiche Behandlung von positiven und negativen Eingaben verhindert Voreingenommenheit und ermöglicht ausgewogenes Training.
- **Neutralisierung bei Null-Eingabe:** Null als Eingabe führt zu Null als Ausgabe, was unnötige Aktivierungen reduziert und als implizite Regularisierung* dient.

* kommt noch in einer der nächsten Vorlesungen

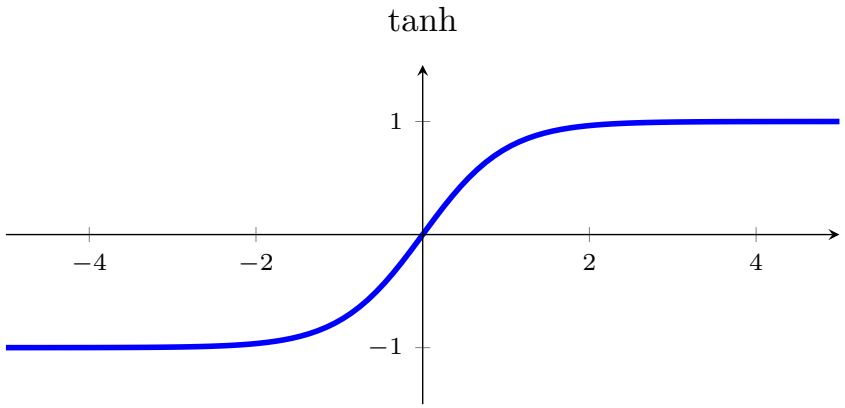
Ableitung der Aktivierungsfunktionen (1/2)

Zur Aktualisierung der Gewichte während des Trainingsprozesses muss die Ableitung der jeweiligen Aktivierungsfunktion berechnet werden. Diese ergibt sich durch die folgenden Formeln:



$$
\begin{aligned}
g(z) &= a = \sigma(z) \\
&= \frac{1}{1+e^{-z}} \\
g'(z) &= \frac{d}{dz} g(z) \\
&= g(z)(1-g(z)) \\
&= a(1-a)
\end{aligned}
$$

(Herleitung siehe unten)

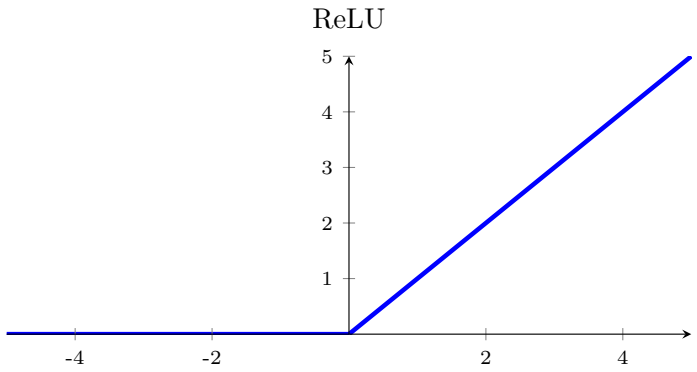


$$
\begin{aligned}
g(z) &= a = \tanh(z) \\
&= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \\
g'(z) &= \frac{d}{dz} g(z) \\
&= 1-(g(z))^2 \\
&= 1-a^2
\end{aligned}
$$

(Herleitung siehe unten)

Ableitung der Aktivierungsfunktionen (2/2)

Zur Aktualisierung der Gewichte während des Trainingsprozesses muss die Ableitung der jeweiligen Aktivierungsfunktion berechnet werden. Diese ergibt sich durch die folgenden Formeln:



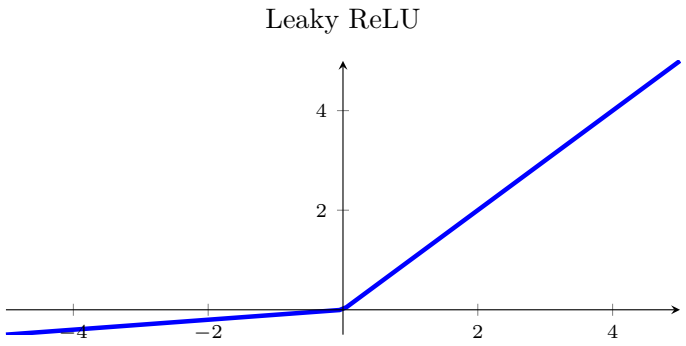
Die ReLU-Funktion wird definiert als:

$$g(z) = \max(0, z)$$

Die Ableitung der ReLU-Funktion ist:

$$g'(z) = \begin{cases} 0 & \text{falls } z < 0 \\ 1 & \text{falls } z \geq 0 \end{cases}$$

$g'(z)$ ist für $x = 0$ mathematisch eigentlich undefiniert. In der Praxis kann dies jedoch ignoriert werden.



Die Leaky ReLU-Funktion wird definiert als:

$$g(z) = \max(\alpha z, z)$$

Die Ableitung der Leaky ReLU-Funktion ist:

$$g'(z) = \begin{cases} \alpha & \text{falls } z < 0 \\ 1 & \text{falls } z \geq 0 \end{cases}$$

Ableitung der Sigmoid-Funktion

Die Sigmoid-Funktion ist definiert als:

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

Um die Ableitung $\sigma'(x)$ zu finden, differenzieren wir $\sigma(x)$ nach x:

$$\sigma'(x) = \frac{d}{dx}\left(\frac{1}{1 + e^{-x}}\right) \tag{2}$$

$$= \frac{d}{dx}\left(1 + e^{-x}\right)^{-1} \tag{3}$$

$$= -1 \cdot (1 + e^{-x})^{-2} \cdot \frac{d}{dx}\left(e^{-x}\right) \tag{4}$$

$$= -1 \cdot (1 + e^{-x})^{-2} \cdot (-e^{-x}) \tag{5}$$

$$= \frac{e^{-x}}{(1 + e^{-x})^2} \tag{6}$$

$$= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \tag{7}$$

$$= \frac{1}{1 + e^{-x}} \cdot \left(1 - \frac{1}{1 + e^{-x}}\right) \tag{8}$$

$$= \sigma(x) \cdot (1 - \sigma(x)) \tag{9}$$

Somit ist die Ableitung der Sigmoid-Funktion $\sigma(x)$ gegeben durch:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x)) \tag{10}$$

Ableitung der tanh-Funktion

Die hyperbolische Tangensfunktion ist definiert als:

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

Um die Ableitung $\tanh'(x)$ zu finden, differenzieren wir $\tanh(x)$ nach x:

$$\tanh'(x) = \frac{d}{dx}\left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right) \tag{2}$$

$$= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \tag{3}$$

$$= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} \tag{4}$$

$$= \frac{e^{2x} + 2 + e^{-2x} - (e^{2x} - 2 + e^{-2x})}{(e^x + e^{-x})^2} \tag{5}$$

$$= \frac{4}{(e^x + e^{-x})^2} \tag{6}$$

$$= 4 \cdot \left(\frac{1}{e^x + e^{-x}}\right)^2 \tag{7}$$

$$= \left(\frac{2}{e^x + e^{-x}}\right)^2 \tag{8}$$

$$= \left(1 - \frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2 \tag{9}$$

$$= (1 - \tanh(x))^2 \tag{10}$$

$$= 1 - \tanh^2(x) \tag{11}$$

Somit ist die Ableitung der hyperbolischen Tangensfunktion $\tanh(x)$ gegeben durch:

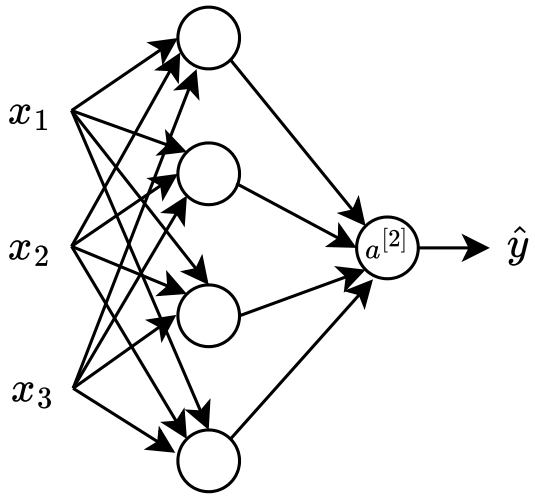
$$\tanh'(x) = 1 - \tanh^2(x) \tag{12}$$

Ausgabefunktionen
(Output Functions)

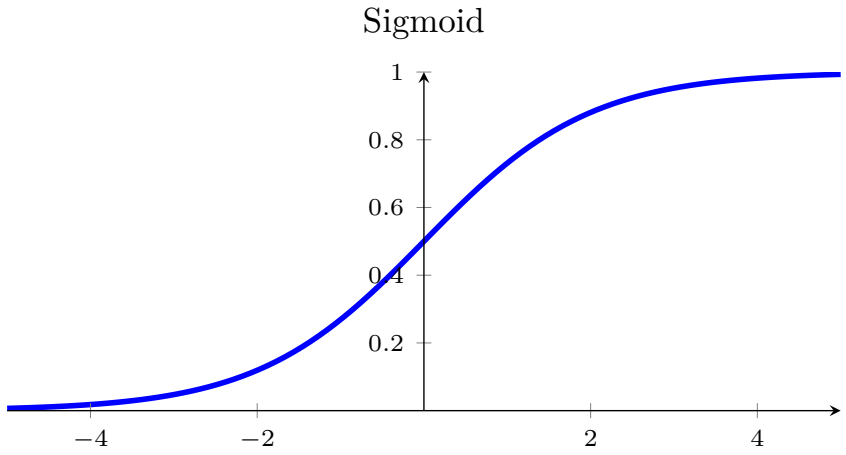
Ausgabefunktion

- Die Ausgabefunktion (d.h. die Aktivierungsfunktion der Ausgabeschicht) bestimmt
  1. das Format der Ausgabe (Einzelwert, Vektor, Matrix, Tensor)
  2. Dem Wertebereich der Ausgabe (Wahrscheinlichkeitswert zwischen 0 und 1, reelle Zahl)

  des neuronalen Netzes.
- Die Ausgabe sollte das gleiche Format und den gleichen Wertebereich haben wie die Labels der Trainingsdaten.



Ausgabefunktion 1: Sigmoid (Logistische Funktion)



Die logistische Funktion (Sigmoid) ist definiert als

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

Diese Funktion gibt Werte im Bereich zwischen 0 und 1 zurück ➔ Sie ist daher besonders nützlich in der Ausgabe-Schicht für Modelle, bei denen eine Wahrscheinlichkeit als Ausgabe erwartet wird (z.B. bei der binären Klassifizierung).

Ausgabefunktion 2: Softmax

Die Softmax-Funktion für einen Vektor $\mathbf{z}$ ist definiert als:

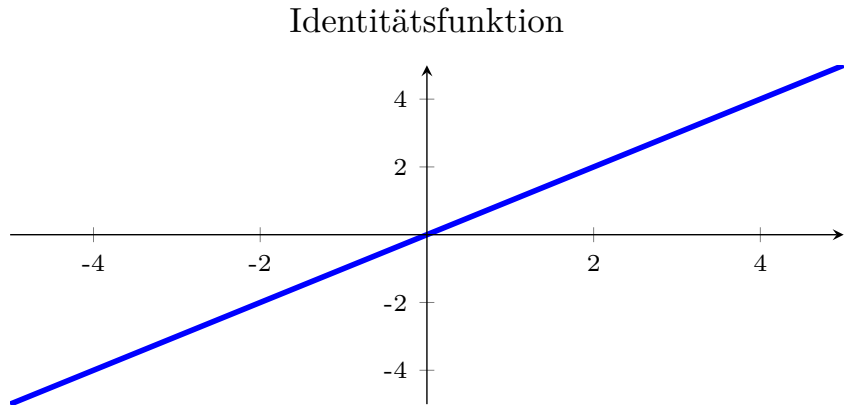
$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

wobei:

- $\sigma(\mathbf{z})_i$ ist der Wert der Softmax-Funktion für die Komponente i des Vektors $\mathbf{z}$.
- k ist die Anzahl der Klassen oder die Länge des Vektors $\mathbf{z}$.

- Softmax wandelt einen Vektor mit Zahlen in einen Vektor mit Wahrscheinlichkeiten um. Die Summer der Wahrscheinlichkeiten ist 1.
- Die Softmax-Funktion kann damit genutzt werden, um bei einer Mehrklassen-Klassifikation die Wahrscheinlichkeitsverteilung über die Ausgabeklassen zu berechnen.
- Mit Hilfe der Argmax-Funktion kann das Vektorelement mit dem höchsten Wert ausgegeben werden. Dies entspricht dann beispielsweise der Ausgabeklasse mit der höchsten Wahrscheinlichkeit.

Ausgabefunktion 3: Identitätsfunktion



Die Identitätsfunktion ist definiert als

$$f(x) = x.$$

- Die Identitätsfunktion kann als Ausgabefunktion genutzt werden, wenn ein Zahlenwert im Rahmen einer Regressionsanalyse ausgegeben werden soll. (z.B. bei der Vorhersage von Immobilienpreisen).
- Der Wertebereich der Identitätsfunktion kann nach oben und/oder unten mit $\min$ oder $\max$ beschränkt werden. Die ReLU-Funktion ist beispielsweise eine solche nach unten beschränkte Identitätsfunktion.

Historische Begriffe

- Perzeptron (Perceptron): Ein Neuron, das jedoch keinen Wahrscheinlichkeitswert ausgibt wie bei der logistischen Regression, sondern einfach nur 0 oder 1, d.h. es verwendet als Ausgabefunktion die Step-Funktion.

- Ein mehrschichtiges Perzeptron (Multi-layer Perceptron) ist ein spezielles neuronales Netz (ein fully-connected network), welches als Ausgabefunktion ebenfalls die Step-Funktion verwendet.

Entwurfsentscheidungen bei der Gestaltung neuronaler Netze

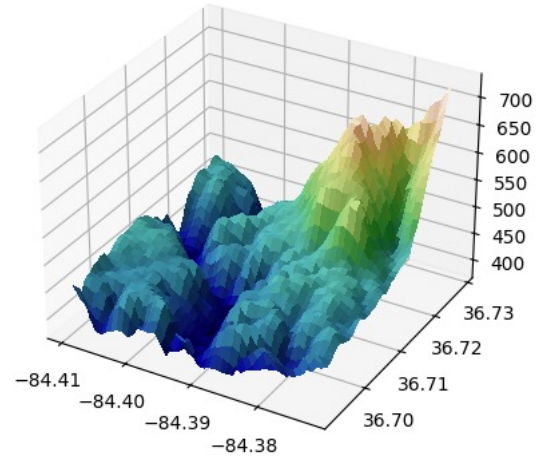
Problemtypen

	Binäre Klassifikation	Mehrklassen-Klassifikation	Regression
Ziel	Klassifikation in 0 oder 1	Klassifikation in eine von k Klassen	Vorhersage eines numerischen Wertes
Beispiel	Klassifikation, ob eine E-Mail Spam ist oder nicht	Klassifikation, welches Verkehrszeichen auf einem Bild zu sehen ist	Vorhersage des Preises einer Immobilie
Ausgabe	Wahrscheinlichkeit der Klasse 1	Vektor der Länge k (der k. Wert gibt die Wahrscheinlichkeit für die k. Klasse an)	vorhergesagter Wert
Aktivierungsfunktion in der Ausgabeschicht	*Sigmoid*	*Softmax* (die Summe aller Wahrscheinlichkeitswerte ergibt 1)	*Identitätsfunktion* oder *ReLU* (wenn man negative Werte vermeiden möchte)
Verlustfunktion (Loss Function)	*Binary Cross Entropy*	*Multi-class Cross Entropy*	*Mean Squared Error* (mittlere quadratische Abweichung) oder *Mean Absolute Error* (mittlere absolute Abweichung)
Kommentar		Die Klasse mit der höchsten Wahrscheinlichkeit wird ausgewählt.	

Die Kombination der Verlustfunktion und der Aktivierungsfunktion der Ausgabeschicht wird meist durch den Problemtyp bestimmt.

Entscheidungsgrenzen (Decision Boundaries)

Die Entscheidungsgrenze (Decision Boundary) unterteilt den Raum der Eingabevariablen (Feature-Raum) eines Klassifikationsmodells in Bereiche, die verschiedenen vorhergesagten Klassen zugeordnet sind. Es handelt sich also um die Grenze, an der die Wahrscheinlichkeit der Zugehörigkeit zu den angrenzenden Klassen gleich hoch ist.



https://matplotlib.org/stable/gallery/mplot3d/custom_shaded_3d_surface.html

- In einem zweidimensionalen Feature-Raum (mit zwei Eingabevariablen) kann eine Entscheidungsgrenze eine Linie sein, im dreidimensionalen Raum eine Fläche und in höherdimensionalen Räumen eine Hyperfläche (Hyperplane).
- Allgemein: Ein n-dimensionaler Feature-Raum hat eine (n-1)-dimensionale Entscheidungsgrenze.

- Die Komplexität der Entscheidungsgrenze hängt vom verwendeten Modell ab:
  - Lineare Modelle wie die logistische Regression haben gerade Linien oder Ebenen als Entscheidungsgrenzen.
  - Komplexere Modelle wie künstliche neuronale Netze können nichtlineare und somit viel kompliziertere Entscheidungsgrenzen erzeugen. Diese entstehen durch die Kombination der linearen Modelle der einzelnen Neuronen in den verschiedenen Ebenen.
- Die Visualisierung der Entscheidungsgrenzen in hochdimensionalen Feature-Räumen (mit oft mehreren Hundert Dimensionen) ist schwierig. Daher wird meist auf eine vereinfachte, 3-dimensionale Visualisierung zurückgegriffen (siehe z.B. die Darstellung rechts).

Quiz

Quiz: Welche Charakteristika des Datensatzes legen nahe, dass zur Modellierung der Entscheidungsgrenze ein tiefes neuronales Netz benötigt wird?

☐ lineare Trennbarkeit der Daten

☐ kleine Anzahl von Eingabevariablen / Features

☐ komplexe Zusammenhänge zwischen Parametern

☐ große Anzahl von Trainingsbeispielen

- Die Daten geben die Form der Entscheidungsgrenze vor. Neuronale Netze sind in der Lage, Entscheidungsgrenzen zu finden, die hochdimensional und nichtlinear sind, indem sie die Entscheidungsgrenzen der verborgenen Neuronen kombinieren.
- Selbst in Fällen, in denen wir unsere Entscheidungsgrenze nicht einfach visualisieren können, gibt die Kenntnis der ungefähren Komplexität unserer Entscheidungsgrenze Aufschluss darüber, wie groß unser Modell sein muss.

Quelle: Udacity

Quiz: Autonomes Fahren – Einführung

Sie sind KI-Ingenieur bei einem Unternehmen, das selbstfahrende Autos herstellt, und Ihr Management hat Sie gebeten, drei neuronale Netze für das nächste Fahrzeugmodell zu entwickeln:

1) Ein System zur Erkennung von einer Gefahrensituation auf der Straße

2) Ein System zur Klassifizierung von Straßenschildern

3) Ein System zur Vorhersage der Geschwindigkeit im fließenden Verkehr

Jedes dieser drei Modelle hat unterschiedliche Anforderungen, ist aber in einem großen System miteinander verbunden. Die folgenden Fragen beziehen sich auf dieses System.

Quelle: Udacity

Quiz: Autonomes Fahren – Frage 1

1) Welche Aktivierungsfunktion schlagen Sie vor für die drei Modelle?

Modell	Aktivierungsfunktion
Erkennung von einer Gefahrensituation auf der Straße	
Klassifizierung von Straßenschildern	
Vorhersage der Geschwindigkeit im fließenden Verkehr	

Quelle: Udacity

Quiz: Autonomes Fahren – Frage 2

2) Welches Risiko ist mit dem Modell zur Vorhersage der Geschwindigkeit im fließenden Verkehr verbunden?

- ◯ Der Ausgabewert ist unbegrenzt und könnte extrem hohe Geschwindigkeiten vorhersagen.
- ◯ ReLU gibt keine negativen Werte zurück und kann somit nicht berücksichtigen, falls der Verkehr rückwärts fließt.
- ◯ Die Mindestgeschwindigkeit kann nicht konfiguriert werden.
- ◯ Es konkurriert direkt mit den anderen Modellen wie dem Modell zur Erkennung von Gefahrensituationen.

Quelle: Udacity

Quiz: Autonomes Fahren – Frage 3

Denken Sie an das System zur Klassifizierung von Verkehrszeichen. Nur eine kleine Anzahl von Schildern ist für ein autonomes Fahrzeug unmittelbar einsatzfähig: Geschwindigkeitsbegrenzung, Stoppschild, Vorfahrt gewähren, usw. Viele andere sind rein informativ, und einige sind extrem selten. Hinzu kommt, dass sich die Schilder von Land zu Land unterscheiden, so dass jede Region ihr eigenes Modell benötigt. Der Ausgabewert ist unbegrenzt und könnte extrem hohe Geschwindigkeiten vorhersagen.

3) Wie sollte man mit dieser Herausforderung umgehen?

- ◯ Alle Verkehrsschilder aus allen Ländern zusammentragen und klassifizieren
- ◯ Die Kontrolle an den Fahrer übergebe, falls ein Verkehrsschild nicht erkannt wird
- ◯ Berücksichtige die wichtigsten Verkehrsschilder und führe eine Klasse „unbekannt“ ein

Quelle: Udacity

Gradientenabstieg in neuronalen Netzen

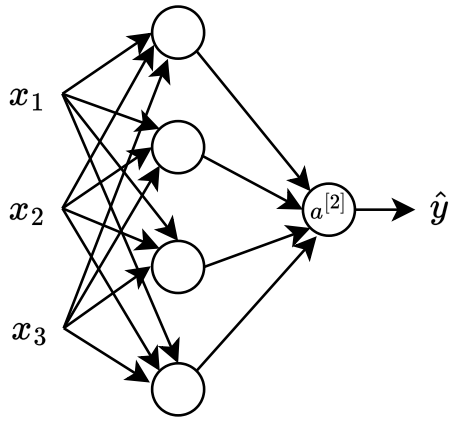
Gradientenabstieg

Neuronales Netz mit 2 Schichten, binäre Klassifikation

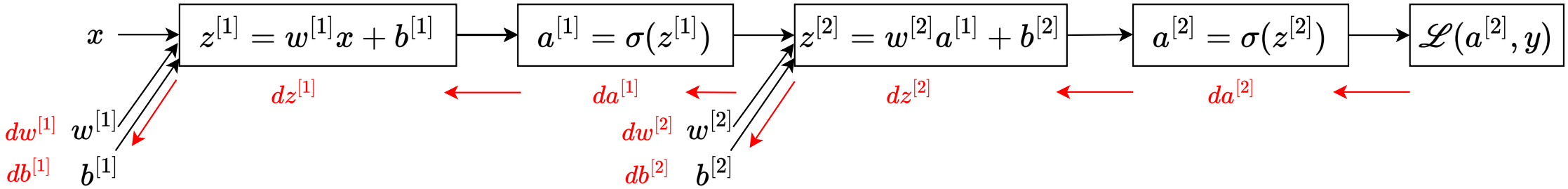
Anzahl:
der Eingabe-Features: $n_x = n^{[0]}$
der Knoten / Neuronen in der hidden Layer: $n_x = n^{[1]}$
der Ausgabe-Knoten: $n_x = n^{[2]}$

Parameter:
$w^{[1]}$ mit Dimensionen $(n^{[1]}, n^{[0]})$
$b^{[1]}$ mit Dimensionen $(n^{[1]}, 1)$
$w^{[2]}$ mit Dimensionen $(n^{[2]}, n^{[1]})$
$b^{[2]}$ mit Dimensionen $(n^{[2]}, 1)$

Kostenfunktion:
$\mathcal{J}(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$




```
Initialisiere die Parameter (Zufallswerte)
Wiederhole:
    1) Feed Forward: Berechne die Predictions
    2) Backpropagation: Berechne die Gradienten
    3) Aktualisiere die Parameter
```



1) Feed Forward: Berechne die Predictions

Für ein Trainingsbeispiel:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$$

Für alle m Trainingsbeispiele (vektorisiert):

$$Z^{[1]} = W^{[1]}X + b^{[1]} \text{ mit } Z^{[1]} = [z^{[1](1)}, z^{[1](2)}, \ldots, z^{[1](m)}]$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

2) Backpropagation: Berechne die Gradienten

Für ein Trainingsbeispiel:

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} A^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]^T} dz^{[2]} \odot g^{[1]'}(z^{[1]})$$
(elementweises Produkt)

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

Für alle m Trainingsbeispiele (vektorisiert):

$$dZ^{[2]} = A^{[2]} - Y \text{ mit } Y = (y^{(1)}, y^{(2)}, \ldots, y^{(m)})$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m} \text{Vektorsumme}(dZ^{[2]})$$

$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} \odot g^{[1]'}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} \text{Vektorsumme}(dZ^{[1]})$$

3) Aktualisiere die Parameter

$$W^{[1]} := W^{[1]} - \alpha \times dW^{[1]}$$

$$b^{[1]} := b^{[1]} - \alpha \times db^{[1]}$$

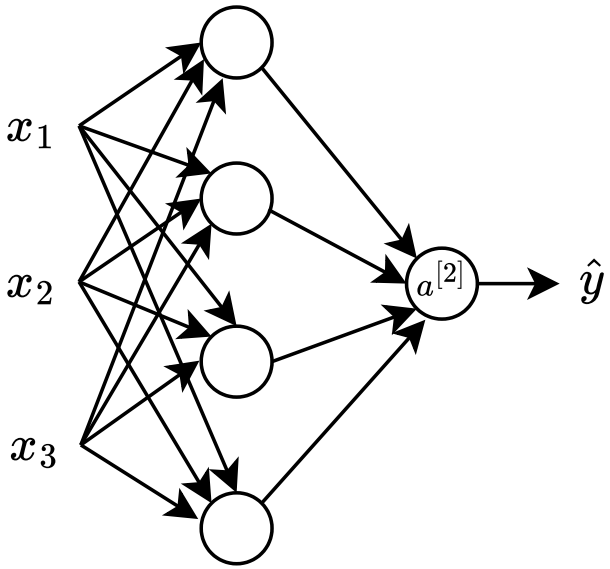
$$W^{[12]} := W^{[2]} - \alpha \times dW^{[2]}$$

$$b^{[12} := b^{[2]} - \alpha \times db^{[2]}$$

Zufällige Initialisierung der Parameter

Die Parameter W müssen mit Zufallszahlen initialisiert werden. Warum?

- Wenn alle Parameter der Matrix W beim gleichen Wert starten (z.B. 0), berechnen alle Knoten die gleiche Funktion und entwickeln sich im Trainingsprozess nicht auseinander.



Initialisierung der Parameter mit kleinen Werten

Die Parameter W sollten mit kleinen Werten initialisiert werden. Warum?

- Je nach Aktivierungsfunktion sollten die Parameter nicht in einem Bereich starten, in dem die Ableitung nahe 0 ist. Ansonsten hat man das gleiche Problem wie beim verschwindenden Gradienten (Vanishing Gradient) ➔ Das Training läuft sehr langsam oder kommt ganz zum Erliegen.