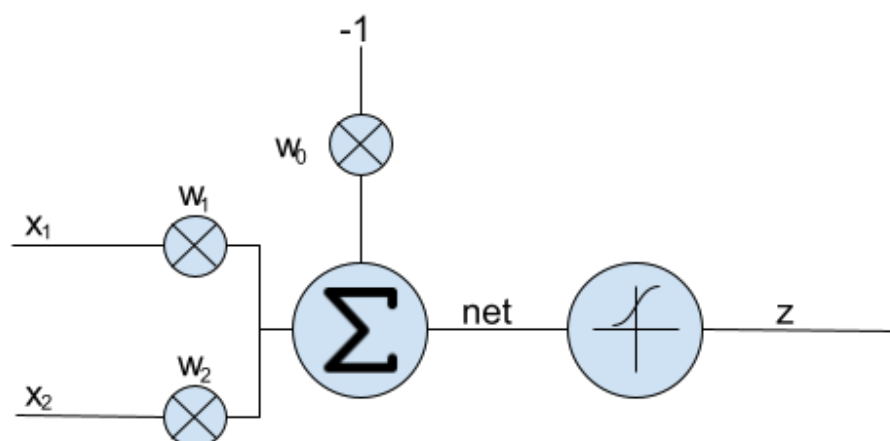

DIE MATHEMATIK NEURONALER NETZE

EINE EINFÜHRUNG

BY
UNKNOWN



2021
NO PUBLISHER

Inhaltsverzeichnis

1	Einführung	2
2	Die Mathematik neuronaler Netze	3
2.1	Das Perceptron	3
2.1.1	Lernverfahren	4
2.1.2	Das Problem mit XOR und nichtlinearen Funktionen	6
2.2	Neuronale Netze	7
2.2.1	Lernverfahren mit Gradientenabstieg	8
2.2.2	Lernverfahren mit Backpropagation	11
2.2.3	XOR und die Lösung	15
	Abbildungsverzeichnis	16
	Glossar	17
3	Anhang	18
3.1	Die Ableitung 1. Grades	18
3.2	Die partielle Ableitung	18
3.3	Die Sigmoidfunktion	19
3.4	Der Gradient	21
3.5	Das Gradientenabstiegsverfahren	22

Kapitel 1

Einführung

In Anbetracht der Tatsache, dass neuronale Netzwerke in Zukunft eine grössere Rolle spielen, wird in diesem Dokument das Ziel verfolgt, die Mathematik dahinter verständlich zu erklären. Es werden einige Vorkenntnisse im Bereich der neuronalen Netze vorausgesetzt. Diese werden in diesem Dokument nicht näher erläutert. Für Leser, welche sich im Gebiet der Differentialrechnung und Optimierung noch nicht so gut auskennen, sei hier an dieser Stelle geraten, den Anhang zu lesen.

Grundsätzlich bildet ein neuronales Netz einen gegebenen Input auf einen Output ab, ist also nichts weiteres als eine Funktion. Diese Funktion wiederum ist mehrdimensional, hat also mehrere Input-Variablen und bildet diese wiederum auf einen mehrdimensionalen Output um $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$. Die Input-Variablen sind die Gewichte, die Output-Werte die Klassen¹. Grundsätzlich werden nun die Gewichte dieses neuronalen Netzes so trainiert, dass eine gewählte Fehlerfunktion möglichst minimiert wird. Man befindet sich hier im Bereich der Optimierung. In Abbildung 1.1 ist eine solche lineare Funktion (in rot) gegeben, welche die Datenpunkte möglichst optimal annähert.

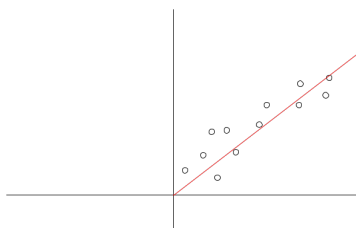


Abbildung 1.1: Annäherung an Datenpunkte über lineare Funktion

¹Sollen z.B. in einem Bild Hunde und Katzen erkannt werden, so gibt es zwei Klassen „Hund“ und „Katze“. In dem Fall ist die Output-Dimension zweidimensional.

Kapitel 2

Die Mathematik neuronaler Netze

2.1 Das Perceptron

Zu Beginn steht das Perceptron. Es wird hier wie in Abbildung 2.1 veranschaulicht. Es besteht aus der Summe mehrerer gewichteter Eingabewerte wie auch einem Bias, welcher die Schwelle einer Aktivierung verschiebt. Die Gewichte werden mit w_i , die Eingabewerte mit x_i bezeichnet. Diese Summe, im folgenden als *net* bezeichnet, wird in eine Aktivierungsfunktion, hier eine Sigmoide, gegeben. Der resultierende Wert wird als z bezeichnet. Der Bias hat den fixen Inputwert -1 , er wird wiederum über ein Gewicht w_0 trainiert/eingestellt.

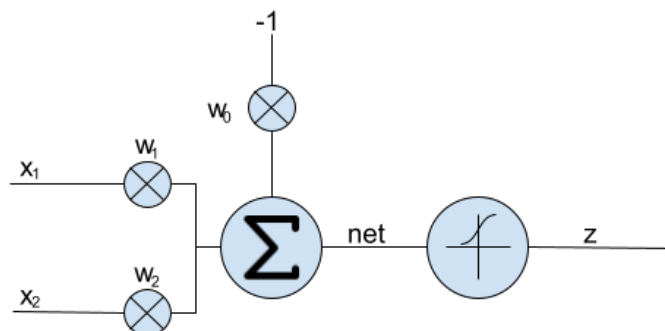


Abbildung 2.1: Das Perceptron

Am Ende dieses Perceptrons steht die Berechnung eines Fehlers. An dieser Stelle wird die quadratische Distanz gewählt. Man berechnet also die quadratische Differenz zwischen dem erwarteten Wert d sowie dem effektiven Wert z .

$$P_{err} = \sum_i^n (d_i - z_i)^2 \quad (2.1)$$

Im Falle des Perceptrons gibt es nur einen Output-Wert. Daher lässt sich die Fehlerfunktion wie in Gleichung 2.2 darstellen.

$$P_{err} = (d - z)^2 \quad (2.2)$$

2.1.1 Lernverfahren

Um nun die Gewichte entsprechend ihrem Anteil am Fehler zu korrigieren, wird das Gradientenabstiegsverfahren¹ verwendet. Dazu muss die Ableitung der Fehlerfunktion bekannt sein. Um diese Ableitung nun einfacher zu gestalten, wird die Fehlerfunktion mit einem konstanten Faktor $\frac{1}{2}$ multipliziert. Dieser Faktor hat keinen Einfluss auf Minima oder Maxima, da er an jedem Punkt der Funktion angewendet wird.

$$P_{err} = \frac{1}{2} \cdot (d - z)^2 \quad (2.3)$$

Nun hat eine Maximierung ebenfalls einen vereinfachenden Vorteil aufgrund der Kettenregel bei der Ableitungsbildung, weswegen die Fehlerfunktion gedreht wird. Es ergibt sich die endgültige Fehlerfunktion.

$$P_{err} = -\frac{1}{2} \cdot (d - z)^2 \quad (2.4)$$

Entsprechend lautet die Ableitung unter Anwendung der Kettenregel:

$$\frac{\delta P_{err}}{\delta z} = -\frac{1}{2} \cdot 2 \cdot (d - z) \cdot -1 \quad (2.5)$$

$$\frac{\delta P_{err}}{\delta z} = (d - z) \quad (2.6)$$

Die Gewichte können anhand des Gradienten korrigiert werden.

$$(w_{0,neu} \quad w_{1,neu} \quad w_{2,neu}) = (w_{0,alt} \quad w_{1,alt} \quad w_{2,alt}) + \lambda \cdot \vec{\nabla} \quad (2.7)$$

Dieser Gradient setzt sich nun aus der partiellen Ableitungskette der verschiedenen Funktionen nach den einzelnen Gewichten zusammen, welche nacheinander aufgerufen und jeweils als Input für die nächste dienen. Dazu wird das Perceptron aus Abbildung 2.1 von hinten her aufgerollt. Der Gradient lautet demnach:

$$\vec{\nabla} = \left(\frac{\delta P_{err}}{\delta w_0} \quad \frac{\delta P_{err}}{\delta w_1} \quad \frac{\delta P_{err}}{\delta w_2} \right) \quad (2.8)$$

An dieser Stelle wird nun die Ableitungskette für w_0 näher erläutert. In einem ersten Schritt muss also die Ableitung der Fehlerfunktion nach w_0 gebildet werden. Wie bereits erwähnt, werden die Funktionen nacheinander aufgerufen und dienen sich gegenseitig als Input. Von hinten her aufgerollt lautet die Aufrufreihenfolge:

$$P_{err} = (d - z(net(w_0, w_1, w_2))) \quad (2.9)$$

Für z und net gilt:

$$z = \frac{1}{1 + e^{-net}} \quad (2.10)$$

$$net = -1 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2 \quad (2.11)$$

¹Siehe Kapitel 3.5

Um nun den Wert des Gradienten für w_0 zu bilden, muss die Funktion P_{err} für w_0 partiell abgeleitet werden. Dies geschieht durch konsequentes Anwenden der Kettenregel.

$$\frac{\delta P_{err}}{\delta z(w_0)} = (d - z) \quad (2.12)$$

$$\frac{\delta z(w_0)}{\delta net(w_0)} = z(1 - z) \quad (2.13)$$

$$\frac{\delta net(w_0)}{\delta w_0} = -1 \quad (2.14)$$

Die gesamte Ableitungskette nach Anwendung der Kettenregel lautet nun:

$$\frac{\delta P_{err}}{\delta w_0} = \frac{\delta P_{err}}{\delta z(w_0)} \cdot \frac{\delta z(w_0)}{\delta net(w_0)} \cdot \frac{\delta net(w_0)}{\delta w_0} \quad (2.15)$$

$$\frac{\delta P_{err}}{\delta w_0} = (d - z) \cdot z \cdot (1 - z) \cdot -1 \quad (2.16)$$

Mit dem Ausdruck 2.16 lässt sich das Gewicht w_0 in einer Iteration korrigieren. Dasselbe Verfahren wird für die Gewichte w_1 sowie w_2 angewendet, es resultiert:

$$\frac{\delta P_{err}}{\delta w_1} = (d - z) \cdot z \cdot (1 - z) \cdot x_1 \quad (2.17)$$

$$\frac{\delta P_{err}}{\delta w_2} = (d - z) \cdot z \cdot (1 - z) \cdot x_2 \quad (2.18)$$

2.1.2 Das Problem mit XOR und nichtlinearen Funktionen

Mittels eines Perceptrons kann also eine Funktion implementiert werden, die ab einer gewissen Schwelle den Ausgabewert 1 liefert. Geometrisch kann dies als eine lineare Separation angesehen werden. Es können z.B. logische Operatoren wie „AND“, „OR“ oder auch „NAND“ abgebildet werden. In der Abbildung 2.2 werden die genannten Operatoren gezeigt. Auf den X-Y-Achsen sind jeweils die Input-Variablen angegeben. Die Output-Dimension wird als Kreis dargestellt. Da der Input auf die Paare $(x, y) \rightarrow (00), (10), (01), (11)$ beschränkt ist, befinden sich die Output-Werte ebenfalls nur an diesen Positionen. Wird an einer Stelle als Ausgabe eine 1 erwartet, so ist der Kreis rot ausgefüllt. Wird eine 0 erwartet, ist der Kreis nicht ausgefüllt. In

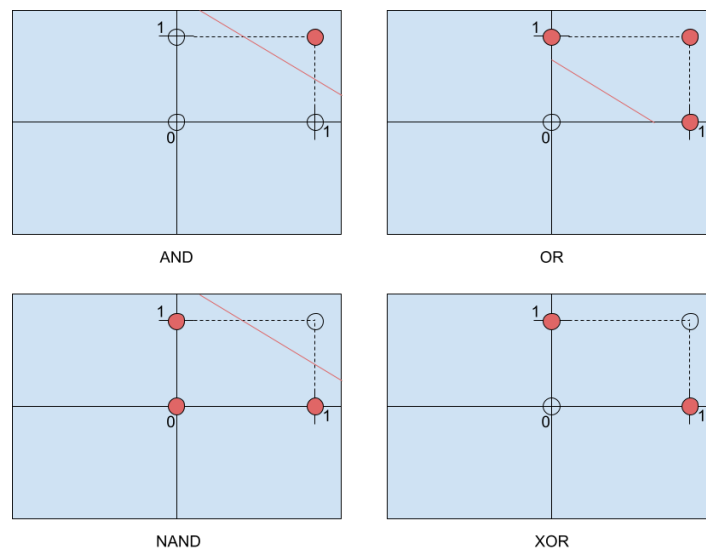


Abbildung 2.2: Logische Operatoren und ihre Separierbarkeit

Abbildung 2.2 wird nun gezeigt, was mit einem Perceptron möglich ist. Es können nur Funktionen implementiert werden, die die einzelnen Klassen voneinander linear separieren können (durch die eingezeichnete rote Gerade). Dies ist im Falle von „XOR“ nicht möglich. Der Leser kann sich selbst überlegen, wie eine solche Gerade auszusehen vermag, um eine Grenze zu ziehen. Diese Problematik wird nun durch ein neuronales Netz gelöst, wo viele dieser Perceptronen miteinander verbunden werden. Dadurch ergeben sich weitere Möglichkeiten, um die Werte in bestimmten Clustern zu klassifizieren. Es können also auch nichtlineare Funktionen abgebildet werden.

2.2 Neuronale Netze

Ein neuronales Netz ist ein Zusammenschluss aus mehreren Perceptronen in verschiedenen Layern. Als Beispiel sei an dieser Stelle die Abbildung 2.3 gegeben. Dieses wird ebenso verwendet, um die Ableitungskette für ein Gewicht in diesem Fall zu zeigen. Anzumerken sei hier, dass die Kombination der gewichteten Summen mit der Aktivierungsfunktion ein Neuron darstellt.

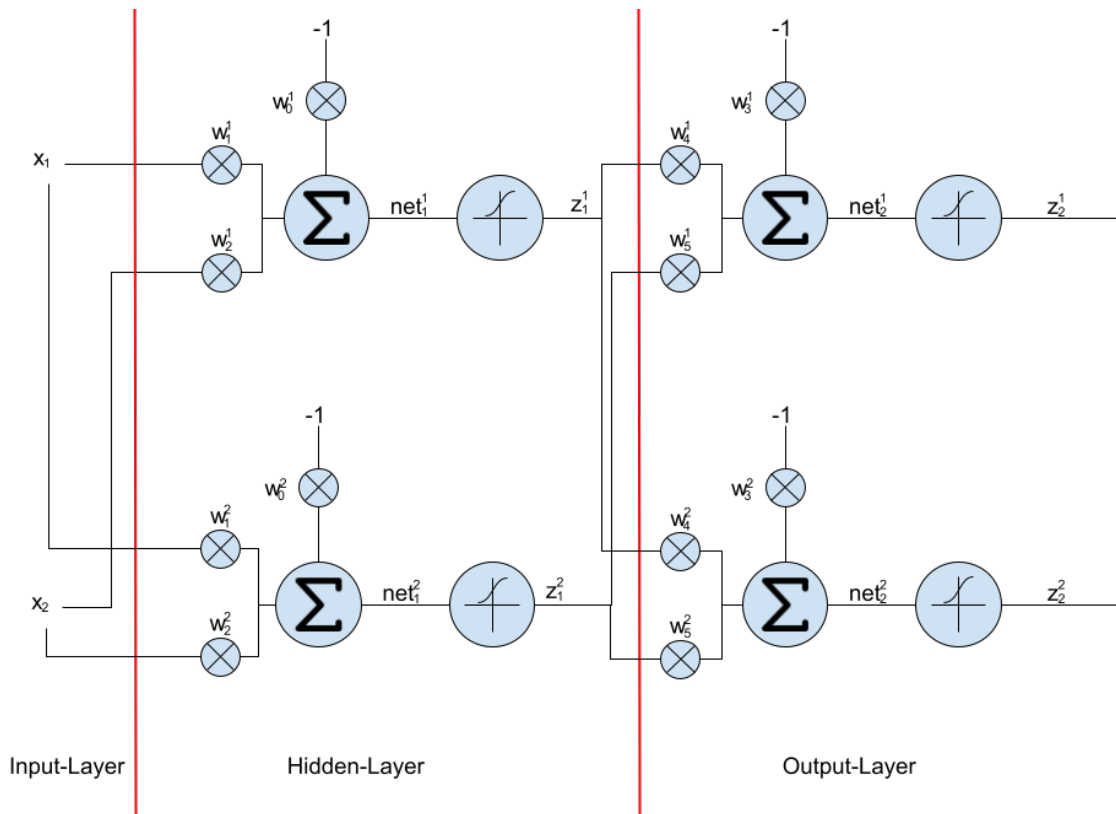


Abbildung 2.3: Ein neuronales Netz mit einem Hidden-Layer

2.2.1 Lernverfahren mit Gradientenabstieg

Wie beim Perceptron auch soll nun an dieser Stelle die Komponente des Gradienten für das Gewicht w_1^1 gezeigt werden. Für dieses Beispiel wird die Abbildung 2.4 hinzugezogen. Die Feh-

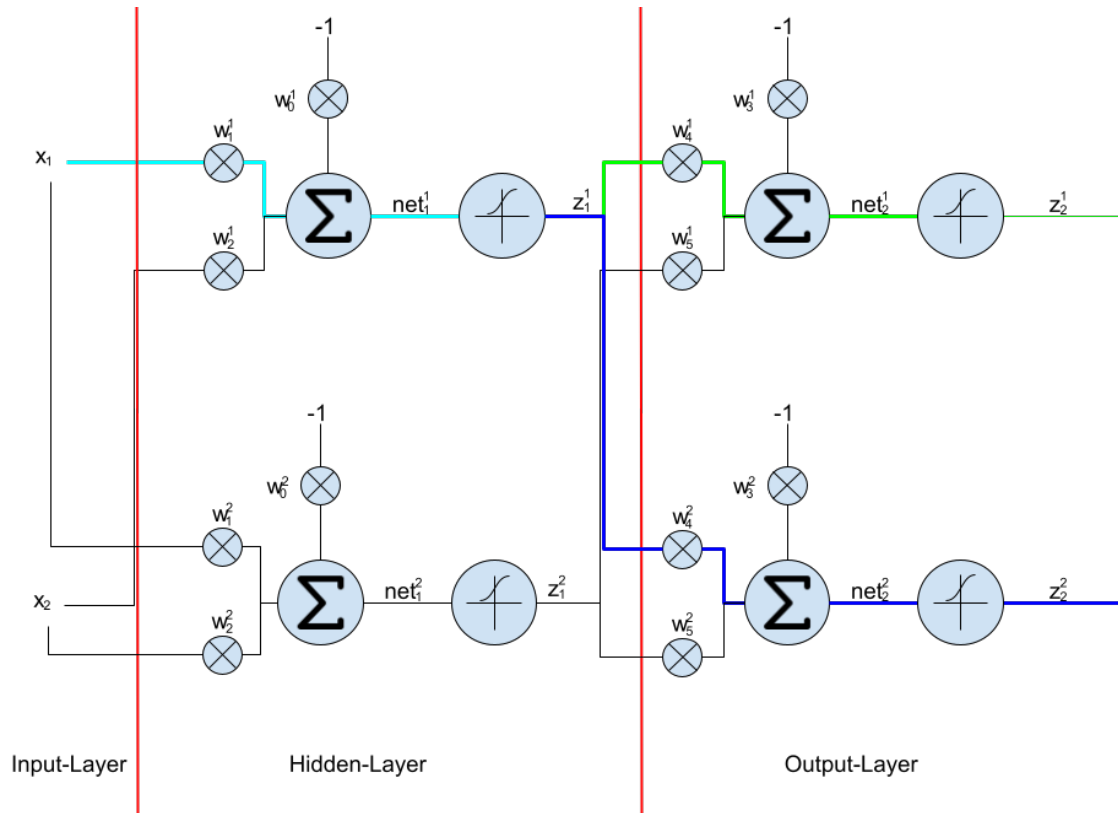


Abbildung 2.4: Der Pfad zum Gewicht w_1^1 im neuronalen Netz

lerfunktion ist wie beim Perceptron dieselbe $P_{err} = -\frac{1}{2} \cdot \sum_i^2 (d^i - z_2^i)^2$. In diesem Fall steht die Komponente i für den Index und die 2 für den Layer. Zur Abbildung 2.4 ist wichtig zu erkennen, dass für die Korrektur des Gewichts w_1^1 mehrere Wege relevant sind. Es kann einmal der grüne sowie der blaue Weg verfolgt werden. Der Teil, welcher bei beiden gleich ist, wird cyan

markiert. Weiterhin sind folgende Werte bekannt:

$$P_{err} = -\frac{1}{2} \cdot \sum_i^2 (d^i - z_2^i)^2 \quad (2.19)$$

$$net_1^1 = x_1 \cdot w_1^1 + x_2 \cdot w_2^1 - w_0^1 \quad (2.20)$$

$$z_1^1 = \frac{1}{1 + e^{-net_1^1}} \quad (2.21)$$

$$net_1^2 = x_1 \cdot w_1^2 + x_2 \cdot w_2^2 - w_0^2 \quad (2.22)$$

$$z_1^2 = \frac{1}{1 + e^{-net_1^2}} \quad (2.23)$$

$$net_2^1 = z_1^1 \cdot w_4^1 + z_1^2 \cdot w_5^1 - w_3^1 \quad (2.24)$$

$$z_2^1 = \frac{1}{1 + e^{-net_2^1}} \quad (2.25)$$

$$net_2^2 = z_1^1 \cdot w_4^2 + z_1^2 \cdot w_5^2 - w_3^2 \quad (2.26)$$

$$z_2^2 = \frac{1}{1 + e^{-net_2^2}} \quad (2.27)$$

Der Gradient für die Gewichte setzt sich wie nachfolgend zusammen:

$$\vec{\nabla} = (w_0^1, w_1^1, w_2^1, w_0^2, w_1^2, w_2^2, w_3^1, w_4^1, w_5^1, w_3^2, w_4^2, w_5^2) \quad (2.28)$$

Die Korrektur in einem Schritt kann wiederum über den Vektor der Gewichte \vec{w} erfolgen.

$$\vec{w}_{neu} = \vec{w}_{alt} + \lambda \cdot \vec{\nabla} \quad (2.29)$$

Wie bereits erwähnt wird nun nur die Komponente w_1^1 aus dem Vektor $\vec{\nabla}$ berücksichtigt. Um diesen herzuleiten, müssen alle Pfade zum Gewicht berücksichtigt werden. In dem Fall gibt es, wie in Abbildung 2.4 entnommen werden kann, deren zwei. Die Funktionskette ist in Gleichung 2.30 ersichtlich. Es werden nur die wirklich relevanten Aufrufe gezeigt.

$$P_{err} = -\frac{1}{2} \cdot (d^1 - z_2^1 (net_2^1 (w_4^1 \cdot z_1^1 (net_1^1 (w_1^1 \cdot x_1, w_2^1 \cdot x_2, -w_0^1)), z_1^2 \cdot w_5^1, -w_3^1)))^2 + \\ -\frac{1}{2} \cdot (d^2 - z_2^2 (net_2^2 (w_4^2 \cdot z_1^1 (net_1^1 (w_1^1 \cdot x_1, w_2^1 \cdot x_2, -w_0^1)), z_1^2 \cdot w_5^2, -w_3^2)))^2 \quad (2.30)$$

Dementsprechend werden die beiden Pfade für die Komponente des Gradienten für w_1^1 berücksichtigt.

$$\frac{\delta P_{err}}{\delta w_1^1} = \frac{\delta P_{err}}{\delta z_2^1(w_1^1)} \cdot \frac{\delta z_2^1(w_1^1)}{\delta net_2^1(w_1^1)} \cdot \frac{\delta net_2^1(w_1^1)}{\delta z_1^1(w_1^1)} \cdot \frac{\delta z_1^1(w_1^1)}{\delta net_1^1(w_1^1)} \cdot \frac{\delta net_1^1(w_1^1)}{\delta w_1^1} + \\ \frac{\delta P_{err}}{\delta z_2^2(w_1^1)} \cdot \frac{\delta z_2^2(w_1^1)}{\delta net_2^2(w_1^1)} \cdot \frac{\delta net_2^2(w_1^1)}{\delta z_1^1(w_1^1)} \cdot \frac{\delta z_1^1(w_1^1)}{\delta net_1^1(w_1^1)} \cdot \frac{\delta net_1^1(w_1^1)}{\delta w_1^1} \quad (2.31)$$

Anders ausgedrückt lautet demnach die Komponente:

$$w_{1,korr}^1 = (d^1 - z_2^1) \cdot z_2^1 (1 - z_2^1) \cdot w_4^1 \cdot z_1^1 (1 - z_1^1) \cdot x_1 + \\ (d^2 - z_2^2) \cdot z_2^2 (1 - z_2^2) \cdot w_4^2 \cdot z_1^1 (1 - z_1^1) \cdot x_1 \quad (2.32)$$

An dieser Stelle sei nun angemerkt, dass dieses Verfahren durchaus funktioniert, jedoch wachsen die Ableitungsterme exponentiell mit der Anzahl Neuronen in den Layern. Aus diesem Grund ist der Gradientenabstieg so nicht brauchbar in der Praxis. Was jedoch auffällt ist die Tatsache, dass einmal berechnete Ableitungsketten wiederverwendet werden können. Gedanklich kann an dieser Stelle z.B. die Ableitungskette für w_2^1 hinzugezogen werden. Man erkennt relativ schnell, dass viele Berechnung, welche schon für w_1^1 gemacht wurden, wiederauftreten. Auf Grundlage dieser Idee der dynamischen Programmierung wurde der Backpropagation Algorithmus entwickelt.

2.2.2 Lernverfahren mit Backpropagation

Der grosse Vorteil dieses Verfahrens ist die Berechnungszeit, welche im Vergleich zum reinen Gradientenabstieg nur noch linear mit den Anzahl der Gewichte wächst. So werden einmal berechnete Ableitungsketten gespeichert und wiederverwendet. Der Input-Layer wird nicht beachtet, dieser ist für die Berechnung irrelevant. Die verbleibenden Layer (Hidden-, Output-Layer) bilden nun die zwei Fälle des Algorithmus.

Zuerst betrachtet man den Fall, wo es sich um den Output-Layer handelt. Um dies zu veranschaulichen, kann die Abbildung 2.5 betrachtet werden. Auf der rechten Seite der Abbildung sind die Werte ersichtlich, die zu diesem Zeitpunkt bekannt sind. In dem Fall gibt es keine folgende Schicht, da es sich selbst um den Output-Layer handelt und das einzig Bekannte ist somit der Fehlerwert P_{err} . Hierbei stehen nun die Angaben r für die Schicht, i für die Anzahl der Neuronen der Schicht und j für die Anzahl der Gewichte (Inputs), welche pro Neuron existieren. In diesem Beispiel ist $j = 3$. Die Gewichte erhalten dadurch mit $i = 2$ die folgende Nummerierung: $w_1^r, w_2^r, w_3^r, w_4^r, w_5^r, w_6^r$. Ebenfalls relevant ist hierbei, dass es sich um ein Fully-Connected Netzwerk handelt, also alle jeweiligen Outputs mit allen Neuronen der folgenden Schicht verbunden sind.

Am Ende sollen auch hier die Gewichte korrigiert werden, es wird nun aber ein Zwischenschritt geschaltet. Dieser Zwischenschritt besteht aus der Berechnung des Terms δ_i^r . Dieser Term beschreibt die Ableitungskette $\frac{\delta P_{err}}{\delta z_i^r} \cdot \frac{\delta z_i^r}{\delta net_i^r}$. Man beachte, dass es genau so viele Elemente δ_i^r wie Neuronen in der Schicht r gibt. Im Falle des Perceptrons gibt es z.B. nur ein Element, im Falle eines neuronalen Netzes gibt es im Output-Layer so viele Elemente wie Output-Klassen (Neuronen). Zu diesem Zeitpunkt gibt es noch keine Korrektur der Gewichte. Es kann also festgehalten werden, dass gilt:

$$\delta_i^r = (d_i^r - z_i^r) \cdot z_i^r(1 - z_i^r). \quad (2.33)$$

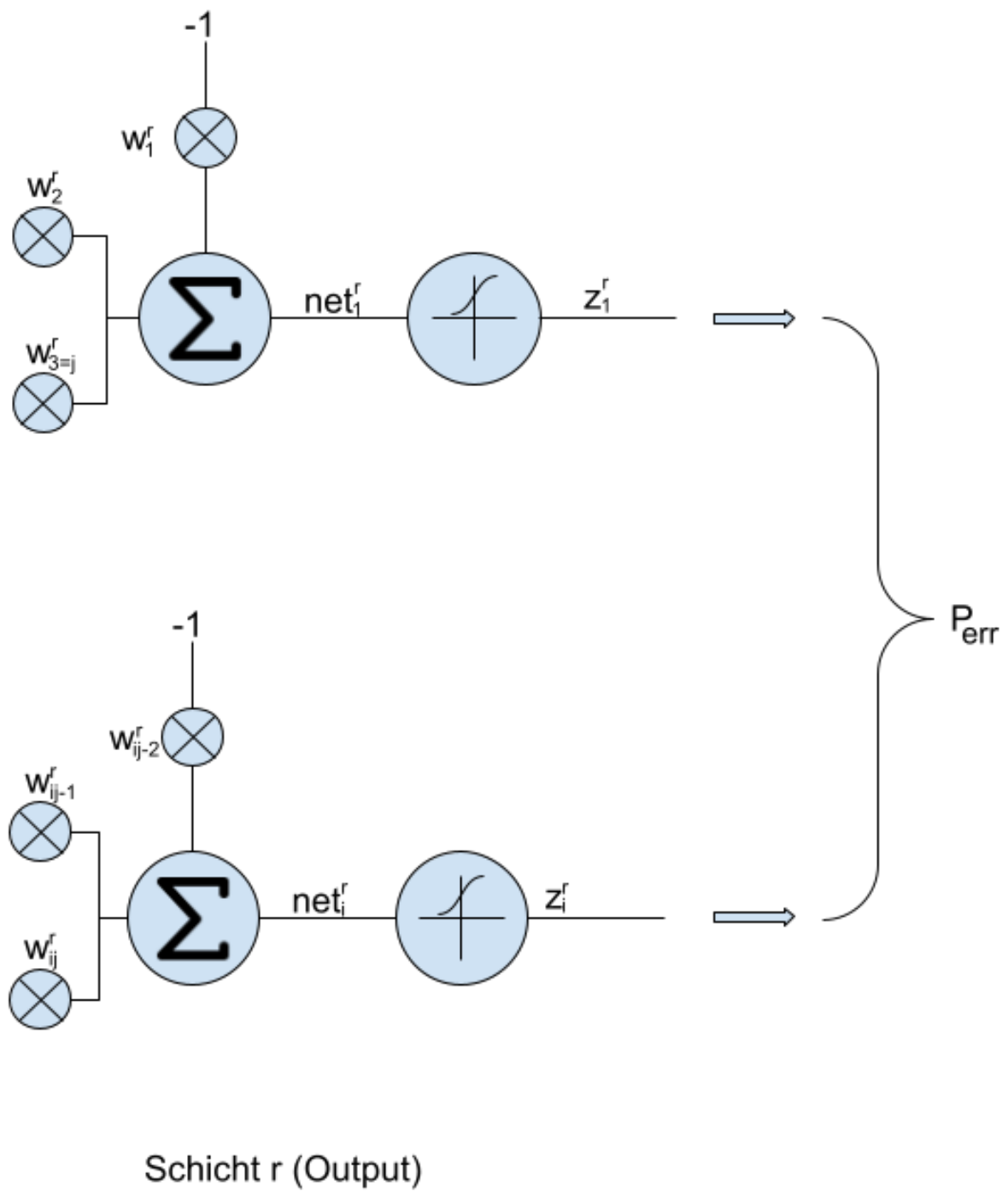


Abbildung 2.5: Die Output-Schicht eines neuronalen Netzes für Backpropagation

Beim zweiten Fall handelt es sich um einen Hidden-Layer. Dazu kann Abbildung 2.6 hinzugezogen werden. Wie vorhin auch steht hier r für die Schicht, welche berechnet werden soll, i für die Anzahl der Neuronen der Schicht r , j für die Anzahl der Gewichte (Inputs) pro Neuron der Schicht r , k für die vorher berechnete Schicht, l für die Anzahl der Neuronen der Schicht k , m für die Anzahl der Gewichte der Schicht k (bei einem Full-Connected Netzwerk gilt $m = i + 1$). Auf der rechten Seite können ebenfalls die bekannten Werte abgelesen werden. Diese setzen sich nun aus den bereits vorhin gesehenen Termen $\delta_l^{k,2}$ zusammen. Es sind die Ableitungen nach net_l^k . Wie bereits erwähnt gibt es so viele Terme, wie Neuronen der Schicht k .

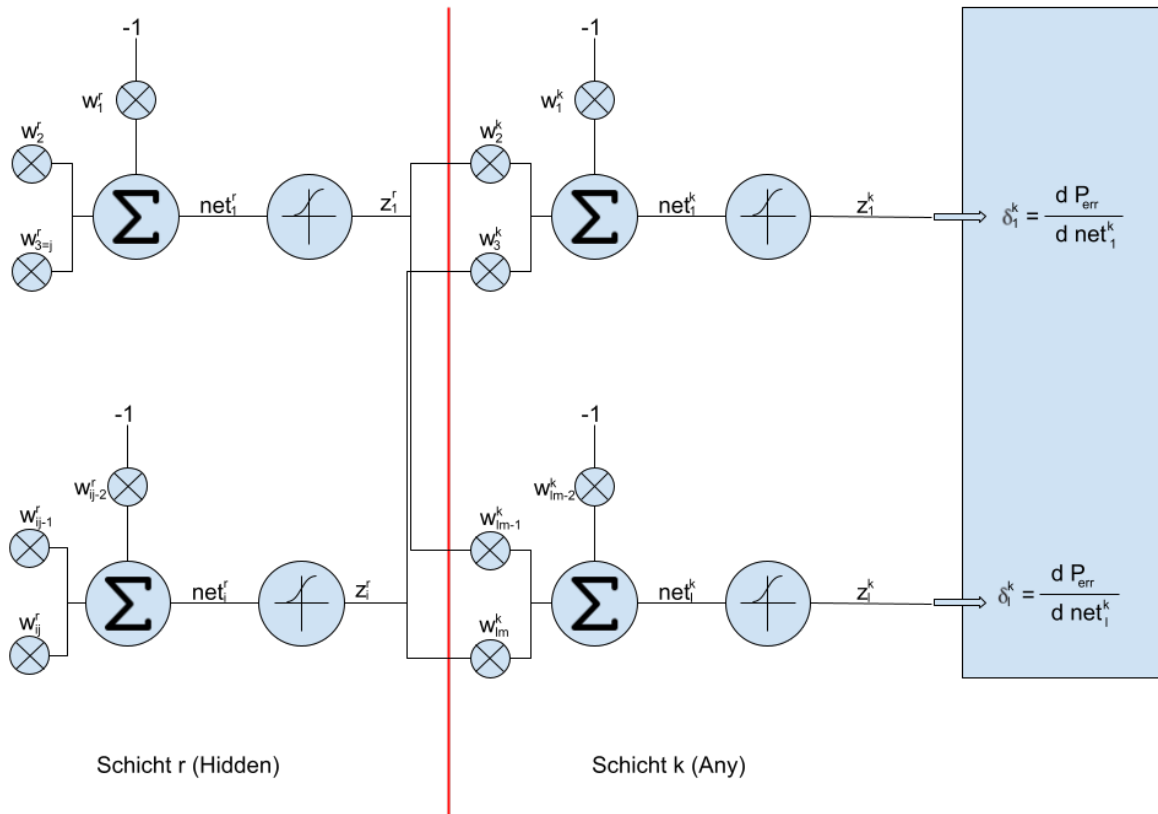


Abbildung 2.6: Zwei Schichten eines neuronalen Netzes für Backpropagation

Auch in diesem Fall geht es um die Berechnung des Terms δ_i^r anhand der genannten bekannten Terme. Es müssen wiederum alle Pfade, welche zu dem Neuron führen, betrachtet

²Anzumerken sei hier, dass es sich bei l nur um einen Platzhalter handelt. Die Nummerierung geht von 1 bis l , es existieren also l δ .

werden. Der Term δ_i^r setzt sich wie nachfolgend zusammen³:

$$\delta_1^k = \frac{\delta P_{err}}{\delta net_1^k} \quad (2.34)$$

$$\delta_l^k = \frac{\delta P_{err}}{\delta net_l^k} \quad (2.35)$$

$$\delta_i^r = \frac{\delta z_i^r}{\delta net_i^r} \cdot \delta_1^k \cdot w_3^k + \frac{\delta z_i^r}{\delta net_i^r} \cdot \delta_l^k \cdot w_{lm}^k \quad (2.36)$$

$$\delta_i^r = \frac{\delta z_i^r}{\delta net_i^r} \cdot (\delta_1^k \cdot w_3^k + \delta_l^k \cdot w_{lm}^k) \quad (2.37)$$

$$\delta_i^r = \frac{\delta z_i^r}{\delta net_i^r} \cdot \sum_n^l (\delta_n^k \cdot w_{nm}^k) \quad (2.38)$$

$$\delta_i^r = z_i^r (1 - z_i^r) \cdot \sum_n^l (\delta_n^k \cdot w_{nm}^k) \quad (2.39)$$

Da der Term δ_i^r nun für beide Fälle bekannt ist, können zuletzt die Gewichte korrigiert werden. Man nehme nun an, dass es bei den Gewichten der Schicht r die Inputs x_{ij}^{r-1} der vorherigen Schicht $r - 1$ gibt, die ebenso wie die Gewichte nummeriert sind. In dem Fall würde die Korrektur für w_{ij}^r lauten:

$$\delta_i^r = \frac{\delta P_{err}}{\delta net_i^r} \quad (2.40)$$

$$w_{ij,neu}^r = w_{ij,alt}^r + \lambda \cdot \delta_i^r \cdot \frac{net_i^r}{\delta w_{ij}^r} \quad (2.41)$$

$$w_{ij,neu}^r = w_{ij,alt}^r + \lambda \cdot \delta_i^r \cdot x_{ij}^{r-1} \quad (2.42)$$

³Aus der Gleichung 2.39 ist ersichtlich, dass es sich bei nm um eine Multiplikation handelt. n läuft von 1 bis l . Wird n mit der Anzahl Gewichte m pro Neuron der Schicht k multipliziert, wird also jeweils das letzte Gewicht eines Neurons betrachtet. Dieses führt immer zu z_i^r .

Zu guter Letzt kann noch der abschliessende Algorithmus 1 angegeben werden.

```

for  $o = last$  downto 2 do
  if  $o == last$  then
    for  $t = 1$  to  $i$  do
       $\delta_t^o = (d_t^o - z_t^o) \cdot z_t^o (1 - z_t^o)$ 
    end
  end
  else
    for  $t = 1$  to  $i$  do
       $\delta_t^o = z_t^o (1 - z_t^o) \cdot \sum_n^l (\delta_n^{o+1} \cdot w_{nm}^{o+1})$ 
    end
  end
  for  $(t, h) = (1, 1)$  to  $(i, j)$  do
     $\delta w_{th}^o = \delta_t^o \cdot w_{th}^o$ 
     $w_{th, neu}^o = w_{th, alt}^o + \lambda \cdot \delta w_{th}^o$ 
  end
end

```

Algorithm 1: Backpropagation Algorithmus

2.2.3 XOR und die Lösung

Abbildungsverzeichnis

1.1	Annäherung an Datenpunkte über lineare Funktion	2
2.1	Das Perceptron	3
2.2	Logische Operatoren und ihre Separierbarkeit	6
2.3	Ein neuronales Netz mit einem Hidden-Layer	7
2.4	Der Pfad zum Gewicht w_1^1 im neuronalen Netz	8
2.5	Die Output-Schicht eines neuronalen Netzes für Backpropagation	12
2.6	Zwei Schichten eines neuronalen Netzes für Backpropagation	13
3.1	Steigung an einem bestimmten Punkt der Funktion $f(x)$	18
3.2	Die Sigmoid-Funktion	19
3.3	Die Ableitung der Sigmoid-Funktion	21
3.4	Gradient an der Position (1, 1)	21

Glossar

Kapitel 3

Anhang

3.1 Die Ableitung 1. Grades

Die Ableitung 1. Grades beschreibt die Steigung an einem bestimmten Punkt der Funktion. In der Abbildung 3.1 wird eine Funktion $f(x)$ (in grün) gegeben. Die Steigung $f'(x)$ an einem bestimmten Punkt x ist rot markiert. Die Ableitung selbst ist wiederum eine Funktion und kann über diverse Ableitungsregeln aufgrund der gegebenen Funktion selbst gebildet werden.

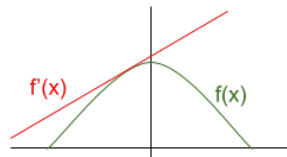


Abbildung 3.1: Steigung an einem bestimmten Punkt der Funktion $f(x)$

Die Ableitungsregeln, welche im Zuge der Erklärung des Lernprozesses eines neuronalen Netzwerks benötigt werden, sind nachfolgend ersichtlich.

Potenzregel $f(x) = x^n \longrightarrow f'(x) = n \cdot x^{n-1}$

Kettenregel $f(x) = u(v(x)) \longrightarrow f'(x) = u'(v(x)) \cdot v'(x)$

Es existieren viele weitere Ableitungsregeln, auf die hier nicht weiter eingegangen wird. Die Schreibweise der Ableitung einer Funktion nach einer Variablen lautet $f'(x) = \frac{\delta f(x)}{\delta x}$.

3.2 Die partielle Ableitung

Ist der Input einer Funktion mehrdimensional, das heisst, die Funktion f ist abhängig von mehreren Variablen, dann kann die Ableitung jeweils lediglich nach einer Variablen gebildet werden. Die übrigen Variablen werden als konstant angesehen. In dem Fall beschreibt die partielle Ableitung die Steigung an einem bestimmten Punkt der abgeleiteten Dimension. Es sei als Beispiel die Funktion $f(x, y) = x^2 + y^2 + 10$ gegeben. Diese wird nun partiell nach x sowie

nach y abgeleitet.

$$f^x(x, y) = 2x \quad (3.1)$$

$$f^y(x, y) = 2y \quad (3.2)$$

Die hierbei angewendete Ableitungsregel ist die Potenzregel, welche bereits im vorangegangenen Kapitel erwähnt wurde.

3.3 Die Sigmoide

Als Aktivierungsfunktion wird die Sigmoidfunktion benutzt. Diese wird heutzutage meist nicht mehr eingesetzt aufgrund des schlechten Lernverhaltens in einigen Bereichen der Funktion. Die erwähnte Eigenschaft wird bei der Betrachtung der Ableitung ersichtlich.

$$\text{sig}(t) = \frac{1}{1 + e^{-t}} \quad (3.3)$$

Geometrisch lässt sich die Funktion wie in Abbildung 3.2¹ so interpretieren, dass eine gewisse Schwelle existiert, ab der die Funktion den Eingabewert auf 1 abbildet, in dem Jargon der Neuronen also „feuert“. Das Resultat lautet entweder 0 oder 1.

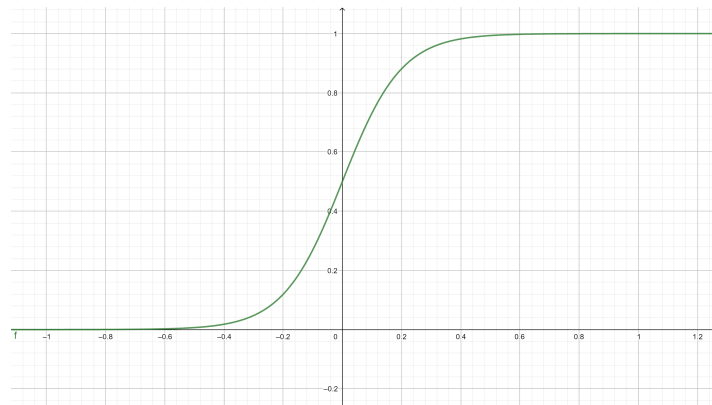


Abbildung 3.2: Die Sigmoid-Funktion

Für die weitere Verwendung ist nun vor allem die Ableitung der Sigmoide interessant, welche in den nachfolgenden Zeilen behandelt wird. Zuerst wird der Bruch durch eine andere Schreibweise (Exponent -1) dargestellt. Weiterhin lässt sich nun die Sigmoide als Verkettung von zwei Funktionen f und y schreiben.

$$\text{sig}(t) = (1 + e^{-t})^{-1} \longrightarrow y(t) = 1 + e^{-t}, f(t) = y(t)^{-1} \quad (3.4)$$

¹Die Sigmoid-Funktion kann über einen Parameter modifiziert werden. Die eigentliche Funktion lautet $\text{sig}(t) = \frac{1}{1 + e^{-a \cdot t}}$. Für die Abbildungen der Sigmoide wie deren Ableitung wurde ein Parameter $a = 10$ gewählt. Für die Herleitung der Ableitungskette der neuronalen Netze wie auch bei der weiteren Erklärung in diesem Kapitel wird der Wert auf 1 belassen. Für die Abbildung wurde ein höherer Wert gewählt, um den Effekt der Ableitung nahe 0 zu verdeutlichen.

Es handelt sich also um eine äussere und innere Funktion, wobei nun die Ableitungsregel 3.1 angewendet wird. Für die innere Ableitung wird nun noch die Regel $f(x) = e^x \rightarrow f'(x) = e^x$ verwendet.

$$\frac{\delta f(t)}{\delta t} = (-1) \cdot (y(t))^{-2} \quad (3.5)$$

$$\frac{\delta y(t)}{\delta t} = (e^{-t}) \cdot (-1) \quad (3.6)$$

Nach der Kettenregel resultiert:

$$\frac{\delta sig(t)}{\delta t} = (-1) \cdot (1 + e^{-t})^{-2} \cdot (e^{-t}) \cdot (-1) \quad (3.7)$$

$$\frac{\delta sig(t)}{\delta t} = \frac{e^{-t}}{(1 + e^{-t})^2} \quad (3.8)$$

Nun wird $\frac{1}{1+e^{-t}}$ ausgeklammert.

$$\frac{\delta sig(t)}{\delta t} = \frac{1}{1 + e^{-t}} \cdot \frac{e^{-t}}{1 + e^{-t}} \quad (3.9)$$

Es wird $\frac{1}{1+e^{-t}}$ dazuaddiert und abgezogen, damit der Faktor umgeformt werden kann.

$$\frac{\delta sig(t)}{\delta t} = \frac{1}{1 + e^{-t}} \cdot \frac{e^{-t} + 1 - 1}{1 + e^{-t}} \quad (3.10)$$

$$\frac{\delta sig(t)}{\delta t} = \frac{1}{1 + e^{-t}} \cdot \left(\frac{1 + e^{-t}}{1 + e^{-t}} - \frac{1}{1 + e^{-t}} \right) \quad (3.11)$$

Es resultiert die Ableitung in bekannter Form 3.12

$$\frac{\delta sig(t)}{\delta t} = sig(t) \cdot (1 - sig(t)) \quad (3.12)$$

Auch hier kann eine geometrische Interpretation in Abbildung 3.3 erfolgen. Ersichtlich wird nun, dass die Steigung nur in einem sehr kleinen Intervall stark ungleich 0 ist. Dies bedeutet, dass sich im Falle eines solchen Variablenwerts, wo die Steigung fast 0 ist, kaum eine Korrektur durch den Gradienten² durchzuführen ist. Daher werden heutzutage Aktivierungsfunktionen wie die „ReLU“ oder „LeakyReLU“ verwendet, welche dieses Problem beheben.

²Siehe Kapitel 3.4

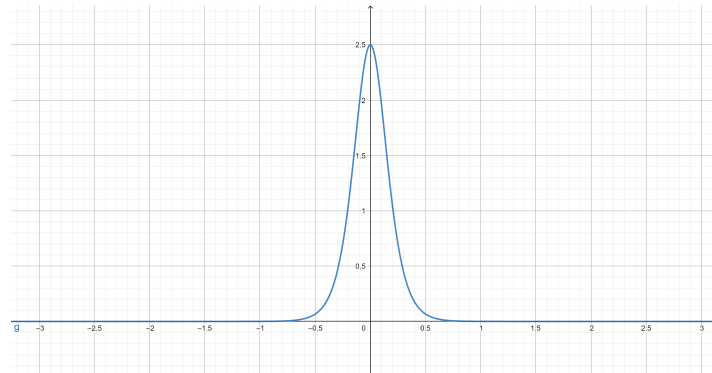


Abbildung 3.3: Die Ableitung der Sigmoid-Funktion

3.4 Der Gradient

Der Gradient beschreibt einen Vektor, welcher in Richtung des steilsten Anstiegs einer Funktion zeigt. Die Komponenten des Gradientenvektors bestehen aus den partiellen Ableitungen der Funktion an der jeweiligen Variablen.

$$\nabla f(x, y) \longrightarrow \left(\frac{\delta f(x, y)}{\delta x} \quad \frac{\delta f(x, y)}{\delta y} \right) \quad (3.13)$$

Geometrisch kann dies an der Position $(x = 1, y = 1)$ für die Funktion $f(x, y) = x^2 + y^2$ wie in Abbildung 3.4 aussehen. Zu beachten sei hier, dass der eigentliche Gradient in der XY-Ebene liegt (blau). Der schwarze Vektor soll lediglich anzeigen, was eine Verschiebung in dieser Richtung bei der Eingabe der Variablen für den Ausgabewert der Funktion bedeutet. Die Länge des Gradienten entspricht der Stärke der Steigung an diesem Punkt.

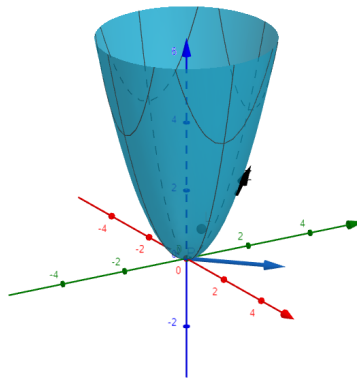


Abbildung 3.4: Gradient an der Position (1,1)

3.5 Das Gradientenabstiegsverfahren

Hierbei handelt es sich um ein Optimierungsverfahren, um einen Maximal- oder Minimalwert³ einer gegebenen Zielfunktion zu finden. Es wird in dem Fall der Gradient, wie in Kapitel 3.4 besprochen, verwendet. Die Idee ist, dass man bei einer Maximierung in kleinen Schritten in Richtung des Gradienten folgt. Als Beispiel wird eine Funktion angegeben, die von zwei Variablen x, y abhängig ist. Beim Wert λ handelt es sich um die Lernrate, welche die Länge der zu gehenden Schritte beeinflusst. Der $\vec{\nabla}$ steht hierbei für den Gradienten.

$$(x_{neu} \quad y_{neu}) = (x_{alt} \quad y_{alt}) + \lambda \cdot \vec{\nabla} \quad (3.14)$$

Geometrisch kann wiederum die Abbildung 3.4 hinzugezogen werden. Dort würde man in einem ersten Schritt in Richtung des schwarzen Vektors gehen, respektive in Richtung des Blauen, wenn man nur die Variablen beachtet.

³Obwohl das Verfahren Gradientenabstieg heisst, kann ebenso ein Gradientenaufstieg durchgeführt werden. Dies ist auch der Ansatz, der in diesem Kapitel erwähnt wird. Beim Gradientenabstieg wird lediglich der Gradientenvektor in die umgekehrte Richtung verfolgt, man sucht also ein Minimum. Die Formel lautet in dem Fall $w_{neu} = w_{alt} - \lambda \cdot \vec{\nabla}$, wobei der Vektor \vec{w} für die anzupassenden Variablen steht.