Deep Learning: Künstliche neuronale Netze

Rückblick

- Warum werden zunehmend neuronale Netze als Modelle verwendet?
- Was sind wesentliche Unterschiede zwischen überwachtem und unüberwachtem Lernen?
- Nenne ein Anwendungsbeispiel für selbstüberwachtes Lernen
- Wie unterscheiden sich Regression und Klassifikation?
- Wozu werden OneHotEncoder verwendet?
- np.ones(120).reshape(-1,10): Was bedeutet die -1? Welche Dimensionen entstehen?
- Was sind die zwei beliebtesten Bibliotheken für Deep Learning?

Ein neuronales Netz

Achtung: Verschiedene Merkmale (Dimensionen) sind als Zeilen dargestellt, nicht als Spalten!

1. Merkmal $X^{(1)}$

2. Merkmal $X^{(2)}$

$w_{1,1}^{(1)}$ $w_{1,2}^{(4)}$

$$y_1^{(1)} = g_1\left(w_{1,1}^{(1)}X^{(1)} + w_{1,2}^{(1)}X^{(2)} + b_1^{(1)}\right)$$

$$y_1^{(2)} = g_1\left(w_{1,1}^{(2)}X^{(1)} + w_{1,2}^{(2)}X^{(2)} + b_1^{(2)}\right)$$

$$y_1^{(3)} = g_1\left(w_{1,1}^{(3)}X^{(1)} + w_{1,2}^{(3)}X^{(2)} + b_1^{(3)}\right)$$

$$y_1^{(4)} = g_1\left(w_{1,1}^{(4)}X^{(1)} + w_{1,2}^{(4)}X^{(2)} + b_1^{(4)}\right)$$

$w_{2,1}^{(1)}$ $w_{2,4}^{(1)}$

Prognostiziertes Label

$$y_2^{(1)} = g_2\left(\begin{matrix} w_{2,1}^{(1)}y_1^{(1)} + w_{2,2}^{(1)}y_1^{(2)} + \\ w_{2,3}^{(1)}y_1^{(3)} + w_{2,4}^{(1)}y_1^{(4)} + b_2^{(1)} \end{matrix}\right)$$

$y_1^{(1)}$ ← Dimension, ← Schicht

$w_{1,1}^{(1)}$ ← Zieldimension, ← Schicht, Urdimension

Eingangsknoten (Input features)

Verdeckte Neuronen (Hidden layer)

Ausgangsknoten (Output layer)

Gewichtungen und Bias-Werte

Das neuronale Netz trainiert indem es die Gewichtungen w aller Verbindungen, sowie die Bias-Werte b anpasst.

Jedes Neuron hat seine eigenen Gewichtungen und Bias-Werte.

Die Gewichtung gibt an wie stark ein Neuron von einem Merkmal der vorderen Schicht abhängt.

Die Bias-Werte verschieben den Wert des Neurons. Der Wert des Neurons kann aufgrund des Bias ungleich null sein, selbst wenn alle Eingänge null sind.

Deklarationen

X = Werte der ursprünglichen Merkmale

y = Werte der Neuronen

w = Gewichtung (Steigung)

b = Bias (Konstante)

g = Aktivierungsfunktion

$y_1^{(1)}$ ← Dimension
← Schicht

$w_{1,1}^{(1)}$ ← Zieldimension
← Schicht, Urdimension

$$\underbrace{y_1}\quad \underbrace{\boldsymbol{w}_1 X}\quad \underbrace{b_1}$$

$$y_1^{(1)} = g_1\left(w_{1,1}^{(1)}X^{(1)} + w_{1,2}^{(1)}X^{(2)} + b_1^{(1)}\right)$$

$$y_1^{(2)} = g_1\left(w_{1,1}^{(2)}X^{(1)} + w_{1,2}^{(2)}X^{(2)} + b_1^{(2)}\right)$$

$$y_1^{(3)} = g_1\left(w_{1,1}^{(3)}X^{(1)} + w_{1,2}^{(3)}X^{(2)} + b_1^{(3)}\right)$$

$$y_1^{(4)} = g_1\left(w_{1,1}^{(4)}X^{(1)} + w_{1,2}^{(4)}X^{(2)} + b_1^{(4)}\right)$$

$$y_1 = g_1(w_1 X + b_1)$$

Matrix-Schreibweise

$$y_1^{(i)} = g_1\left(\sum_{j=1..D} w_{1,j}^{(i)} X^{(j)} + b_1^{(i)}\right) \qquad y_1 = g_1(\boldsymbol{w}_1 X + b_1)$$

$$y_1 = \begin{bmatrix} y_1^{(1)} \\ y_1^{(2)} \\ y_1^{(3)} \\ y_1^{(4)} \end{bmatrix} \quad \boldsymbol{w}_1 = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{1,1}^{(3)} & w_{1,2}^{(3)} \\ w_{1,1}^{(4)} & w_{1,2}^{(4)} \end{bmatrix} \quad X = \begin{bmatrix} X^{(1)} \\ X^{(2)} \end{bmatrix} \quad b_1 = \begin{bmatrix} b_1^{(1)} \\ b_1^{(2)} \\ b_1^{(3)} \\ b_1^{(4)} \end{bmatrix}$$

Die verschiedenen Merkmale (Zeilen!) werden in Matrizen zusammengefasst

Matrizenmultiplikation:

1. Zeile von $\boldsymbol{w}_1$: $y_1^{(1)} = g_1\left(w_{1,1}^{(1)} X^{(1)} + w_{1,2}^{(1)} X^{(2)} + b_1^{(1)}\right)$

2. Zeile von $\boldsymbol{w}_1$: $y_1^{(2)} = g_1\left(w_{1,1}^{(2)} X^{(1)} + w_{1,2}^{(2)} X^{(2)} + b_1^{(2)}\right)$

...

Die Summe über alle Eingangsparameter ergibt sich durch Matrixmultiplikation

Von Vektor zu Vektor

Matrix multipliziert mit Vektor ergibt Vektor

$$\boldsymbol{A} = \begin{bmatrix} A_1^{(1)} & A_2^{(1)} \\ A_1^{(2)} & A_2^{(2)} \\ A_1^{(3)} & A_2^{(3)} \\ A_1^{(4)} & A_2^{(4)} \end{bmatrix} \quad B = \begin{bmatrix} B^{(1)} \\ B^{(2)} \end{bmatrix} \quad \boldsymbol{A} \cdot B = C = \begin{bmatrix} A_1^{(1)} B^{(1)} + A_2^{(1)} B^{(2)} \\ A_1^{(2)} B^{(1)} + A_2^{(2)} B^{(2)} \\ A_1^{(3)} B^{(1)} + A_2^{(3)} B^{(2)} \\ A_1^{(4)} B^{(1)} + A_2^{(4)} B^{(2)} \end{bmatrix}$$

Jede Schicht eines neuronalen Netzes konvertiert einen Eingangsvektor in einen Ausgangsvektor. Beispiel:

Erste Schicht: Eingangsvektor = X (2 Zeilen × 1 Spalte), Ausgangsvektor = y_1 (4 × 1), Gewichtsmatrix = $\boldsymbol{w}_1$ (4 × 2) $\quad y_1 = \mathrm{g}_1(\boldsymbol{w}_1 X + b_1)$

Zweite Schicht: Eingangsvektor = y_1 (4 × 1), Ausgangsvektor = y_2 (1 × 1), Gewichtsmatrix = $\boldsymbol{w}_2$ (1 × 4)

Anwendung auf gesamten Datensatz

Typischerweise wird der gesamte Datensatz auf einmal berechnet

Anzahl der Datenpunkte: N, Anzahl der Eingangsmerkmale: D, Anzahl der Neuronen: H_1

Die Merkmale an den Eingangsknoten stellen eine 2D-Matrix dar, mit der Größe (N, D)

Die Werte der Neuronen der Hiddenschicht (y_1) bilden eine Matrix der Größe (N, H_1)

Die Gewichtungen bilden eine Matrix der Größe (D, H_1)

Die Biaswerte bilden einen Vektor der Größe (H_1)

$$y_1 = g_1(X\boldsymbol{w}_1 + b_1)$$

Berechnung der Werte der 1. Hiddenschicht (y_1) bei N Datenpunkten, D = 2, H_1= 4:

$$X = \begin{array}{cc} \text{1. Merkmal} & \text{2. Merkmal} \\ \begin{bmatrix} X_1^{(1)} & X_1^{(2)} \\ X_2^{(1)} & X_2^{(2)} \\ X_3^{(1)} & X_3^{(2)} \\ \dots & \dots \\ X_n^{(1)} & X_n^{(2)} \end{bmatrix} \end{array}$$

$$w_1 = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,1}^{(2)} & w_{1,1}^{(3)} & w_{1,1}^{(4)} \\ w_{1,2}^{(1)} & w_{1,2}^{(2)} & w_{1,2}^{(3)} & w_{1,2}^{(4)} \end{bmatrix}$$

$$b_1 = \begin{bmatrix} b_1^{(1)} & b_1^{(2)} & b_1^{(3)} & b_1^{(4)} \end{bmatrix}$$

$$y_1 = g \begin{array}{cccc} \text{1. Neuron} & \text{2. Neuron} & \text{3.} & \text{4.} \\ \left[\begin{matrix} X_1^{(1)} w_{1,1}^{(1)} + X_1^{(2)} w_{1,2}^{(1)} + b_1^{(1)} & X_1^{(1)} w_{1,1}^{(2)} + X_1^{(2)} w_{1,2}^{(2)} + b_1^{(2)} & \dots & \dots \\ X_2^{(1)} w_{1,1}^{(1)} + X_2^{(2)} w_{1,2}^{(1)} + b_1^{(1)} & X_2^{(1)} w_{1,1}^{(2)} + X_2^{(2)} w_{1,2}^{(2)} + b_1^{(2)} & \dots & \dots \\ X_3^{(1)} w_{1,1}^{(1)} + X_3^{(2)} w_{1,2}^{(1)} + b_1^{(1)} & X_3^{(1)} w_{1,1}^{(2)} + X_3^{(2)} w_{1,2}^{(2)} + b_1^{(2)} & \dots & \dots \\ \dots & \dots & \dots & \dots \\ X_n^{(1)} w_{1,1}^{(1)} + X_n^{(2)} w_{1,2}^{(1)} + b_1^{(1)} & X_n^{(1)} w_{1,1}^{(2)} + X_n^{(2)} w_{1,2}^{(2)} + b_1^{(2)} & \dots & \dots \end{matrix}\right] \end{array}$$

➡ Hier sind verschiedene Merkmale wieder wie gewohnt als Spalten dargestellt

Mehrschichtiges Perzeptron in sklearn

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# MLP-Classifier mit 2 Hidden-Layer mit 7, 4 Neuronen
# (7 Neuronen in der ersten Hiddenschicht, 4 Neuronen in der zweiten Hiddenschicht)
clf = MLPClassifier(hidden_layer_sizes=(7, 4), max_iter=10000)
clf.fit(X_train, y_train)

print(accuracy_score(y_test, clf.predict(X_test)))
```

Teamwork

Gruppe A: Wie verändern sich die Dimensionen der Gewichtungen (coefs_), wenn die Anzahl der verdeckten Schichten in 'MLP Iris.py' von 2 auf 3 erhöht (eine Schicht mit 6 Neuronen hinzugefügt) wird? Wie liegen die coefs_ vor? Wie hängt die Anzahl mit der Anzahl der Hiddenlayer zusammen?

Gruppe B: Wie verändern sich die Dimensionen der Gewichtungen (coefs_), wenn die Anzahl der Merkmale in 'MLP Iris.py' von 4 auf 2 reduziert wird?

Gruppe C: Wie verändern sich die Dimensionen der Gewichtungen (coefs_), wenn die Anzahl der Iris-Klassen in 'MLP Iris.py' von 3 auf 2 reduziert wird?

Bis 12:15 Uhr

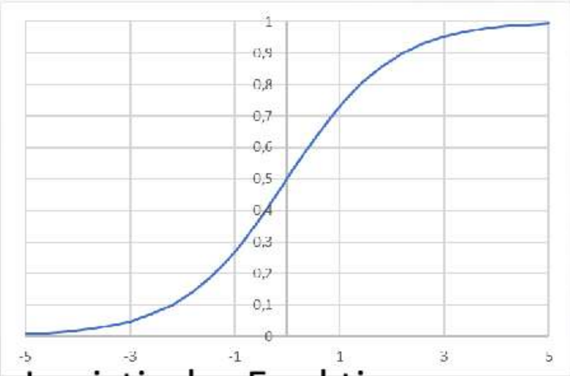
Die Aktivierungsfunktion

Die Aktivierungsfunktion ist in der Regel nichtlinear und sorgt damit für Stabilität des neuronalen Netzes. Erst durch die Nichtlinearität sind Kombinationen mehrerer Schichten sinnvoll.

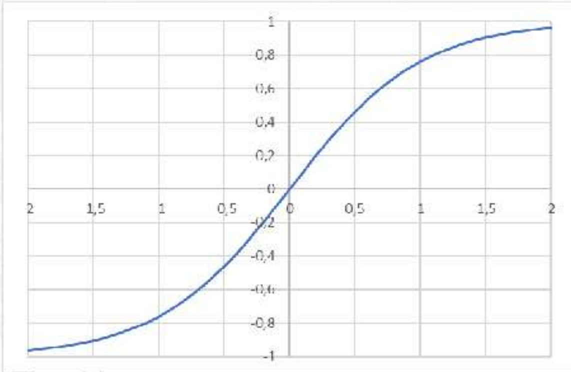
Beispiel wenn Aktivierungsfunktion linear wäre: $y_1 = 5x + 1$ $y_2 = 2y_1 + 2$
y_2 kann auch direkt erreicht werden, ohne die Zwischenschicht: $y_2 = 10x + 4$

Die Aktivierungsfunktion sorgt dafür, dass die Neuronen nur bestimmte Werte annehmen können (z. B. zwischen 0 und 1).

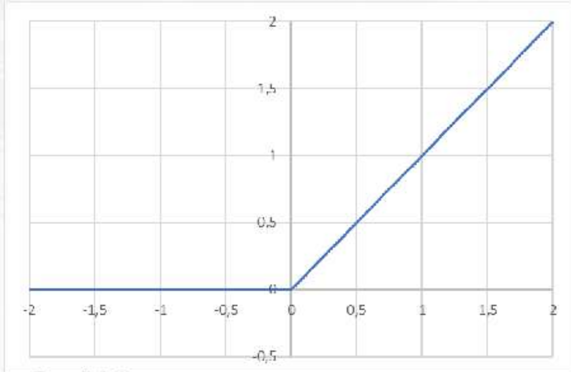
Die Aktivierungsfunktion der Ausgabeschicht hängt von der Aufgabe des Netzes ab (Regression oder Klassifikation).



Logistische Funktion (Sigmoid)



TanH



ReLU

https://en.wikipedia.org/wiki/Activation_function

Komplettes neuronales Netz

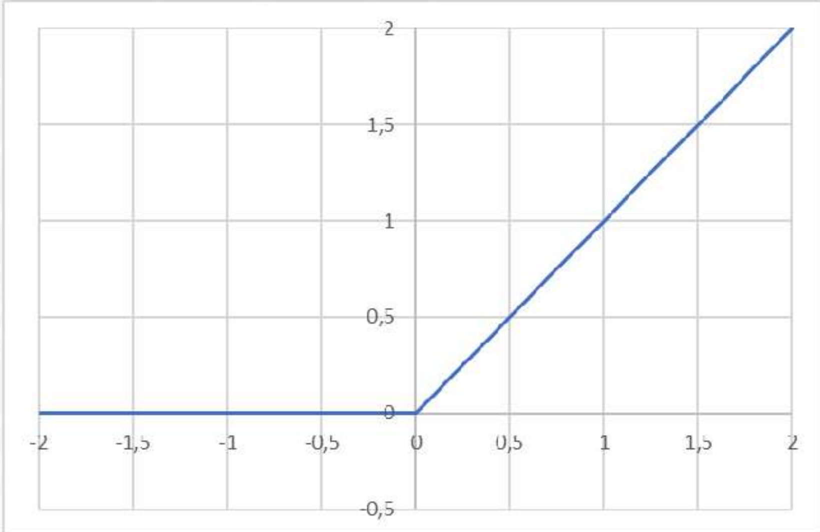
Erste Schicht: $y_1 = g_1(\boldsymbol{w}_1 X + b_1)$

Zweite Schicht: $y_{pred} = y_2 = g_2(\boldsymbol{w}_2 y_1 + b_2)$

```
from sklearn.datasets import load_iris
import numpy as np

iris = load_iris()
X=iris.data[:,0:2]
y=iris.target[:, np.newaxis]

w1 = np.random.randn(2, 4)
w2 = np.random.randn(4, 1)

b1 = np.random.randn(1, 4)
b2 = np.random.randn(1, 1)

y1 = np.maximum(X.dot(w1) +b1, 0)
y_pred = np.maximum(y1.dot(w2)+b2, 0)
loss = np.square(y_pred - y).sum()
```

ReLU

np.maximum([4, -7, 3],0) → [4, 0, 3]



Gradientenabstieg um loss zu minimieren → Ableitungen müssen durch das Netz zurückgerechnet werden

Was sind die „bestmöglichen“ Parameter?

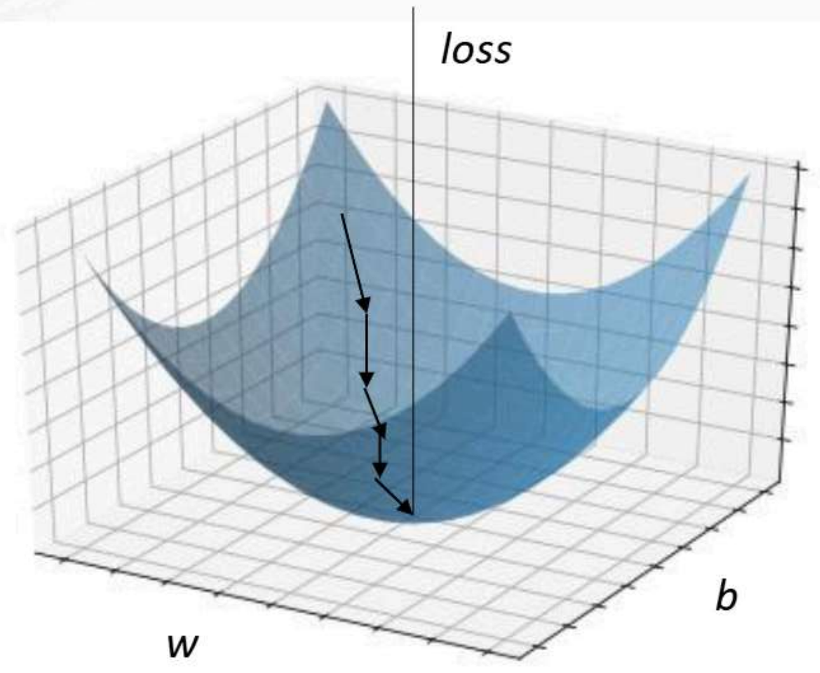
Die bestmöglichen Parameter sind gegeben durch die geringste Abweichung zwischen Vorhersage und wirklichen Labels

$$loss = \frac{1}{n}\sum\left(y - y_{pred}\right)^2$$

n = Anzahl der Datensätze
Optimum, wenn $loss$ = min

⇩

w und b geschickt variieren bis $loss$ = min



https://de.wikipedia.org/wiki/Methode_der_kleinsten_Quadrate

Gradientenabstieg bei linearer Regression

1. Zufälligen Startpunkt wählen w_0, b_0
2. Loss-Funktion definieren

$$loss = \frac{1}{N} \sum_{i=1..N} [y_i - (wX_i + b)]^2$$

dloss/dw= d((y-wx-b)^2)/dw
dloss/dw=dg(u)/du * du(w)/dw
g(u)=u^2,   u(w)=y-wx-b
dg(u)/du=2u,   du(w)/dw=-x
dloss/dw=-2ux = -2x(y-wx-b)

3. Ableitungen der loss-Funktion bilden

$$\frac{dloss}{dw} = \frac{1}{N} \sum_{i=1..N} -2X_i[y_i - (wX_i + b)] \qquad \frac{dloss}{db} = \frac{1}{N} \sum_{i=1..N} -2[y_i - (wX_i + b)]$$

4. Mit Hilfe der Lernrate α neue Steigung w^* und Achsenabschnitt b^* berechnen

$$w^* = w - \alpha \frac{dloss}{dw} \qquad b^* = b - \alpha \frac{dloss}{db}$$

5. Wiederhole die Schritte 3 und 4 bis ein Abbruchkriterium erfüllt ist (z.B. Änderung von *loss* < 0.01)

Backpropagation

$$y_1 = g_1(X\boldsymbol{w}_1 + b_1) \qquad y_{pred} = y_2 = g_2(y_1\boldsymbol{w}_2 + b_2) \quad loss = \sum (y - y_{pred})^2$$

- Der Loss hängt von der gesamten Struktur des neuronalen Netzes ab
- Um den Loss zu minimieren, müssen alle Werte in $\boldsymbol{w}_1$ und $\boldsymbol{w}_2$ sowie in b_1 und b_2 angepasst werden
- Der Gradientenabstieg verändert die Werte jeweils in der steilsten Richtung, um schnellstmöglich das Minimum von loss zu erreichen

$$\boldsymbol{w}_1^* = \boldsymbol{w}_1 - \alpha \frac{dloss}{d\boldsymbol{w}_1} \qquad b_1^* = b_1 - \alpha \frac{dloss}{db_1}$$

$$\boldsymbol{w}_2^* = \boldsymbol{w}_2 - \alpha \frac{dloss}{d\boldsymbol{w}_2} \qquad b_2^* = b_2 - \alpha \frac{dloss}{db_2}$$

- Die Lernrate α gibt die Schrittweite an, wie stark sich die Werte in $\boldsymbol{w}_1$ und $\boldsymbol{w}_2$ sowie in b_1 und b_2 verändern können
- Optimierung so lange, bis ein Abbruchkriterium erfüllt ist

Ableitungen von Hand

$$y_1 = g_1(\mathbf{w}_1 X + b_1) \qquad y_{pred} = y_2 = g_2(\mathbf{w}_2 y_1 + b_2) \qquad loss = \left(y_{pred} - y\right)^2$$

- Berechnung von $\frac{dloss}{d\mathbf{w}_2}$ nach Kettenregel $f(x) = u\big(v(x)\big) \rightarrow \frac{df}{dx} = \frac{du}{dv}\frac{dv}{dx}$

$$\frac{dloss}{d\mathbf{w}_2} = \frac{d((y_{pred}-y)^2)}{\mathrm{d}\mathbf{w}_2} \rightarrow \frac{du}{dv}\frac{dv}{d\mathbf{w}_2}$$

$$u(v) = v^2 \qquad v(\mathbf{w}_2) = y_{pred} - y$$

$$\frac{du(v)}{dv} = 2v \qquad \frac{dv(\mathbf{w}_2)}{d\mathbf{w}_2} = \frac{d(y_{pred}-y)}{d\mathbf{w}_2}$$

$$\frac{dloss}{d\mathbf{w}_2} = 2\left(y_{\mathrm{pred}} - \mathrm{y}\right) \frac{d(y_{pred}-y)}{d\mathbf{w}_2}$$

$$\frac{d(y_{pred} - y)}{d\mathbf{w}_2} = \frac{d(g_2(\mathbf{w}_2 y_1 + b_2) - y)}{\mathrm{d}\mathbf{w}_2} \rightarrow \frac{du}{dv}\frac{dv}{d\mathbf{w}_2}$$

$$u(v) = g_2(v) = \mathrm{relu}(v) \qquad v(\mathbf{w}_2) = \mathbf{w}_2 y_1 + b_2$$

$$g_2(v) = \begin{cases} 0 \text{ wenn } v \leq 0 \\ v \text{ wenn } v > 0 \end{cases} \qquad \frac{dv(\mathbf{w}_2)}{d\mathbf{w}_2} = y_1$$

$$\frac{du(v)}{dv} = \begin{cases} 0 \text{ wenn } v \leq 0 \\ 1 \text{ wenn } v > 0 \end{cases}$$

$\Rightarrow$ $\frac{dloss}{d\mathbf{w}_2} = 2\left(y_{pred} - \mathrm{y}\right) y_1$ bzw. $\frac{dloss}{d\mathbf{w}_2} = 0$ wenn $\mathbf{w}_2 y_1 + b_2 < 0$

Ableitungen für mehrere Schichten

Ableitung nach der 2. Schicht mit ReLU-Aktivierung

$$\frac{dloss}{d\boldsymbol{w}_2} = 2y_1\left(y_{pred} - y\right), 0 \qquad \frac{dloss}{db_2} = 2\left(y_{pred} - y\right), 0$$

Ableitung nach der 1. Schicht mit ReLU-Aktivierung

$$\frac{dloss}{d\boldsymbol{w}_1} = 2X\left[\left(y_{pred} - y\right)\boldsymbol{w}_2\right], 0 \qquad \frac{dloss}{db_1} = 2\left(y_{pred} - y\right)\boldsymbol{w}_2, 0$$

Übungsvorschläge

- Matrixmultiplikation nachvollziehen
- Berechnung von neuronalen Netzen nachvollziehen
- Welche Dimensionen haben die Modellparameter?
- Welche Aufgabe hat die Aktivierungsfunktion in neuronalen Netzen?
- Multilayer-Perzeptron in sklearn benutzen, um Iris-Datensatz zu klassifizieren (Welcher Score kann durch Hyperparameter-Tuning erreicht werden, wenn nur das 1. und 2. Merkmal verwendet werden?)
- Im Beispiel 'numpy neuronales Netz.py' zusätzlichen Bias hinzufügen
- In 'numpy neuronales Netz.py' TensorFlow (und GradientTape) anstelle von NumPy verwenden, um Iris-Daten zu klassifizieren