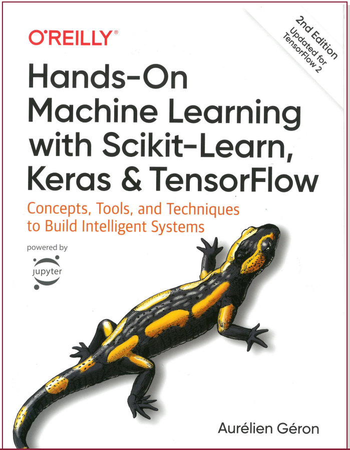
Deep Learning mit Python

Neuronale Netzwerke I

Fabian Gieseke & Moritz Seiler

Department of Information Systems
University of Münster

Heutiger Plan



Literatur

Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd edition, 2019. Kapitel 10: Seiten 279–308 ([Gér19]).

Übersicht

Übersicht

Biologisches Neuronales Netzwerk

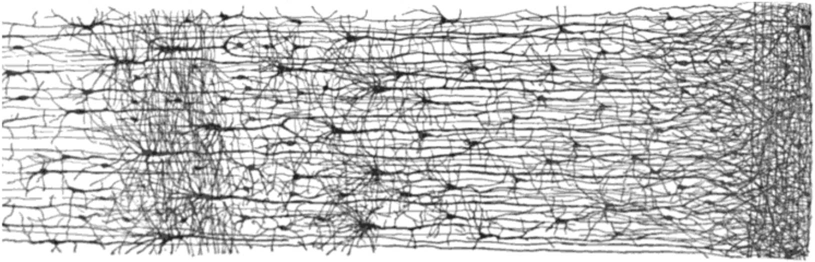
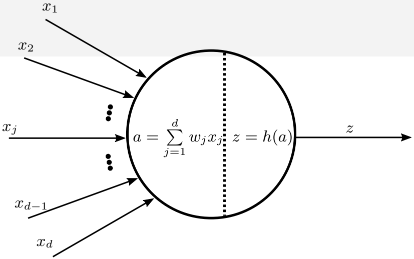


Bild: Santiago Ramon y Cajal (public domain); https://en.wikipedia.org/wiki/Cerebral_cortex

Perzeptron



Künstliches Neuron

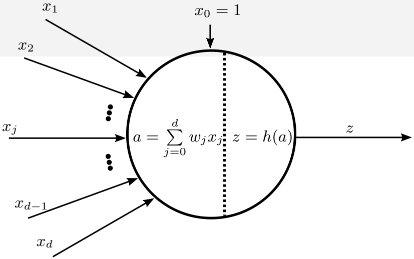
- Eingabe: Merkmalsvektoren der Form $\mathbf{x} = [x_1, \ldots, x_d]^\mathsf{T} \in \mathbb{R}^d$
- Gewichte: Jeder Eingabewert x_j wird mit einem Gewicht w_j multipliziert, d.h.:

$$a = \sum_{j=1}^{d} w_j x_j$$

- Stufenfunktion: Anschließend wird eine Stufenfunktion h angewendet. Z.B.:

$$h(a) = \begin{cases} 0, & \text{falls } a \leq 0 \\ 1, & \text{falls } a > 0 \end{cases}$$

Perzeptron



Künstliches Neuron (mit impliziten Bias-Gewicht)

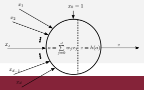
- Eingabe: Merkmalsvektoren der Form $\mathbf{x} = [x_1, \ldots, x_d]^\mathsf{T} \in \mathbb{R}^d$
- Gewichte: Jeder Eingabewert x_j wird mit einem Gewicht w_j multipliziert, d.h.:

$$a = \sum_{j=0}^{d} w_j x_j$$

- Stufenfunktion: Anschließend wird eine Stufenfunktion h angewendet. Z.B.:

$$h(a) = \begin{cases} 0, & \text{falls } a \leq 0 \\ 1, & \text{falls } a > 0 \end{cases}$$

Perzeptron: Training



Perzeptron-Algorithmus

Require: Let $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \{0, 1\}$ be the training set and $\eta > 0$.

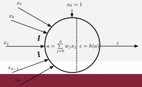
```
Initialize the weights w_0, w_1, ..., w_d with small random values.
repeat
    for each training example (x_i, y_i) ∈ T do
        Compute the predicted output ŷ_i = h(a_i) = h(∑_{j=0}^{d} w_j x_{i,j})
        for j = 0, ..., d do
            Update weight w_j ← w_j + η(y_i − ŷ_i) x_{i,j}
        end for
    end for
until no mistakes are made anymore
```

Initialize the weights $w_0, w_1, \ldots, w_d$ with small random values.

$\hat{y}_i = h(a_i) = h(\sum_{j=0}^{d} w_j x_{i,j})$

$w_j \leftarrow w_j + \eta(y_i - \hat{y}_i)x_{i,j}$

Perzeptron: Training



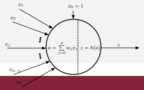
Perzeptron-Algorithmus

Require: Let $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \{0, 1\}$ be the training set and $\eta > 0$.

Initialize the weights $w_0, w_1, \ldots, w_d$ with small random values.
repeat
    for each training example $(\mathbf{x}_i, y_i) \in T$ **do**
        Compute the predicted output $\hat{y}_i = h(a_i) = h(\sum_{j=0}^{d} w_j x_{i,j})$
        for $j = 0, \ldots, d$ **do**
            Update weight $w_j \leftarrow w_j + \eta(y_i - \hat{y}_i)x_{i,j}$
        end for
    end for
until no mistakes are made anymore

- Die Gewichte $w_0, w_1, \ldots, w_d$ werden angepasst, falls Vorhersage falsch war:
  1. $y_i = 1$ und $\hat{y}_i = 0$: Vergrößere $a_i = \sum_{j=0}^{d} w_j x_{i,j}$, indem alle w_j mit $x_{i,j} > 0$ vergrößert und alle w_j mit $x_{i,j} < 0$ verkleinert werden.
  2. $y_i = 0$ und $\hat{y}_i = 1$: Verkleinere $a_i = \sum_{j=0}^{d} w_j x_{i,j}$, indem alle w_j mit $x_{i,j} > 0$ verkleinert und alle w_j mit $x_{i,j} < 0$ vergrößert werden.
- Der Parameter $\eta > 0$ wird Lernrate genannt (z.B. $\eta = 0.1$)

Perzeptron: Training



Perzeptron-Algorithmus

Require: Let $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \{0, 1\}$ be the training set and $\eta > 0$.

```
Initialize the weights w_0, w_1, ..., w_d with small random values.
repeat
    for each training example (x_i, y_i) ∈ T do
        Compute the predicted output ŷ_i = h(a_i) = h(∑_{j=0}^{d} w_j x_{i,j})
        for j = 0, ..., d do
            Update weight w_j ← w_j + η(y_i − ŷ_i) x_{i,j}
        end for
    end for
until no mistakes are made anymore
```

- Die Gewichte $w_0, w_1, \ldots, w_d$ werden angepasst, falls Vorhersage falsch war:
  1. $y_i = 1$ und $\hat{y}_i = 0$: Vergrößere $a_i = \sum_{j=0}^{d} w_j x_{i,j}$, indem alle w_j mit $x_{i,j} > 0$ vergrößert und alle w_j mit $x_{i,j} < 0$ verkleinert werden.
  2. $y_i = 0$ und $\hat{y}_i = 1$: Verkleinere $a_i = \sum_{j=0}^{d} w_j x_{i,j}$, indem alle w_j mit $x_{i,j} > 0$ verkleinert und alle w_j mit $x_{i,j} < 0$ vergrößert werden.
- Der Parameter $\eta > 0$ wird Lernrate genannt (z.B. $\eta = 0.1$)

Falls die Datenpunkte linear trennbar sind, konvergiert dieser Algorithmus.

Perzeptron: Demo

jupyter Perceptron (Iris Dataset) Last Checkpoint: a few seconds ago (autosaved)

Logout

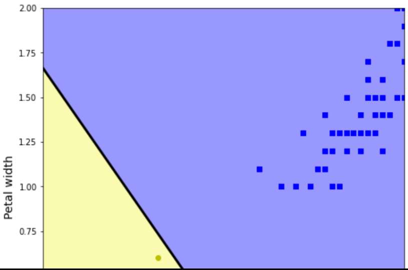
File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

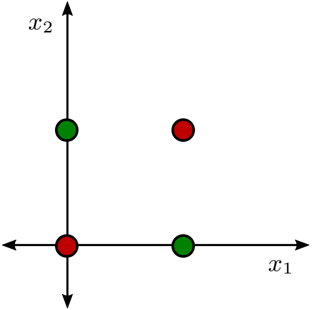
Run Code

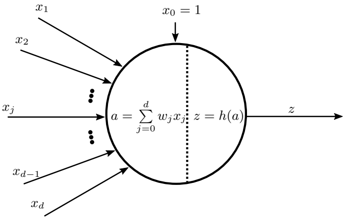
```
zz = y_predict.reshape(x0.shape)

plt.figure(figsize=(8, 8))
plt.plot(X[y==0, 0], X[y==0, 1], "bs", label="Not Iris-Setosa")
plt.plot(X[y==1, 0], X[y==1, 1], "yo", label="Iris-Setosa")

plt.plot([axes[0], axes[1]], [a * axes[0] + b, a * axes[1] + b], "k-", linewidth=3)
from matplotlib.colors import ListedColormap
custom_cmap = ListedColormap(['#9898ff', '#fafab0'])

plt.contourf(x0, x1, zz, cmap=custom_cmap)
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.legend(loc="lower right", fontsize=14)
plt.axis(axes)
```

Out[4]: (0.0, 5.0, 0.0, 2.0)

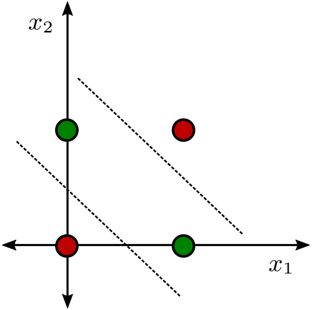


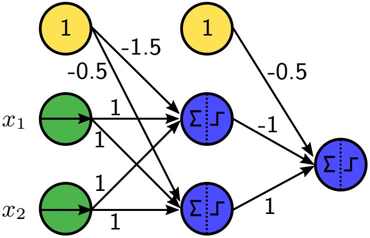
Grenzen des Perzeptrons?





Idee: „Stapeln“ von Perzeptrons!

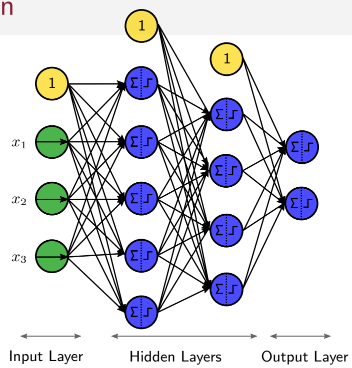




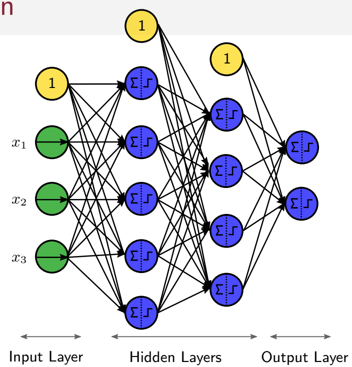
$T = \{([0,0]^\top, 0), ([1,1]^\top, 0), ([0,1]^\top, 1), ([1,0]^\top, 1)\}$

Übersicht

Multi-Layer Perzeptron



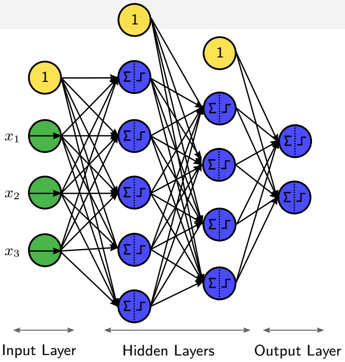
Multi-Layer Perzeptron



Mehrschichtige Modelle

- Architektur: Ein Multi-Layer Perzeptron (MLP) besteht aus einer Eingabeschicht *(input layer)*, mehreren versteckten Schichten *(hidden layers)* und einer Ausgabeschicht *(output layer)*.
- Implizite Bias-Gewichte: Bis auf die Ausgabeschicht verfügt jede Schicht über ein Bias-Gewicht (gelb; werden oft nicht explizit gezeigt).
- Feed-Forward: Eine Schicht ist immer nur mit höheren Schichten verbunden.

Ausgabe pro Schicht

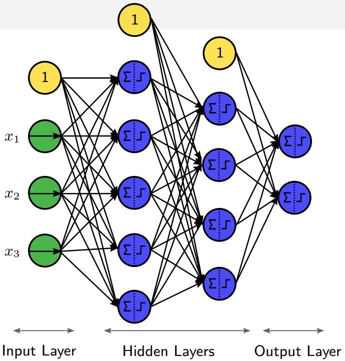


Notation

- Sei $l = 0, \ldots, L$ eine Numerierung der Schichten ($l = 0$ entspricht Eingabeschicht). Für $l \geq 1$ bezeichne $\mathbf{x}^{(l)}$ die Eingabe und $\mathbf{h}^{(l)}$ die Ausgabe der Schicht l.
- Bezeichne $\mathbf{w}_i^{(l)}$ die Gewichte des i-ten Neurons in Schicht l und $b_i^{(l)}$ das Gewicht zum Bias-Neuron von Schicht l. Dann gilt

$$\mathbf{h}_i^{(l)} = h\left((\mathbf{w}_i^{(l)})^\mathsf{T}\mathbf{x}^{(l)} + b_i^{(l)}\right)$$

Ausgabe pro Schicht



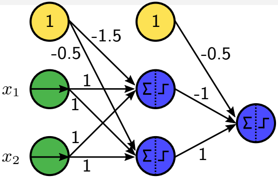
Notation

- Indem man nun alle $\mathbf{w}_i^{(l)}$ des Layers l zu einer Matrix $\mathbf{W}^{(l)}$ und alle Bias-Gewichte $b_i^{(l)}$ zu einem Vektor $\mathbf{b}^{(l)}$ zusammenfasst, erhält man:

$$\mathbf{h}^{(l)} = h\left(\mathbf{W}^{(l)}\mathbf{x}^{(l)} + \mathbf{b}^{(l)}\right)$$

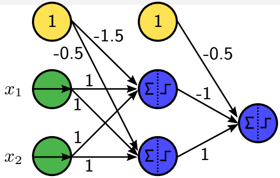
Hierbei ist die Anwendung von h elementweise zu verstehen.

Ausgabe pro Schicht: Beispiel



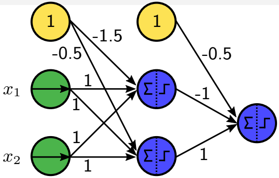
Fragen: *Wie viele Parameter besitzt das MLP?*
Wie sehen $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$, $\mathbf{h}^{(l)}$ *für* $l = 1, 2$ *aus?*

Ausgabe pro Schicht: Beispiel



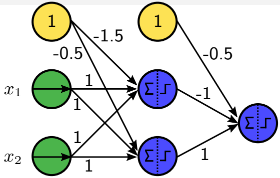
- Für $l = 1$ erhalten wir: $\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ und $\mathbf{b}^{(1)} = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}$ und somit $\mathbf{h}^{(1)} = h\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} + \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}\right)$

Ausgabe pro Schicht: Beispiel



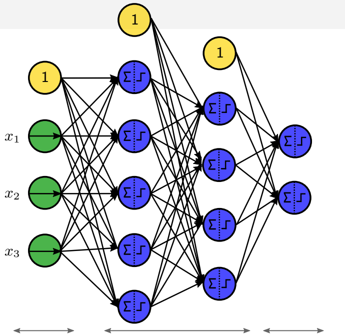
- Für $l = 1$ erhalten wir: $\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ und $\mathbf{b}^{(1)} = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}$ und somit
$$\mathbf{h}^{(1)} = h\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} + \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix} \right)$$
- Für $l = 2$ erhalten wir: $\mathbf{W}^{(2)} = \begin{bmatrix} -1 & 1 \end{bmatrix}$ und $\mathbf{b}^{(2)} = \begin{bmatrix} -0.5 \end{bmatrix}$ und somit
$$\mathbf{h}^{(2)} = h\left(\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} + \begin{bmatrix} -0.5 \end{bmatrix} \right)$$

Ausgabe pro Schicht: Beispiel



- Für $l = 1$ erhalten wir: $\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ und $\mathbf{b}^{(1)} = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}$ und somit
$$\mathbf{h}^{(1)} = h\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} + \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}\right)$$
- Für $l = 2$ erhalten wir: $\mathbf{W}^{(2)} = \begin{bmatrix} -1 & 1 \end{bmatrix}$ und $\mathbf{b}^{(2)} = \begin{bmatrix} -0.5 \end{bmatrix}$ und somit
$$\mathbf{h}^{(2)} = h\left(\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} + \begin{bmatrix} -0.5 \end{bmatrix}\right)$$
- Insgesamt 9 (trainierbare) Parameter.

Ausgabe des MLP



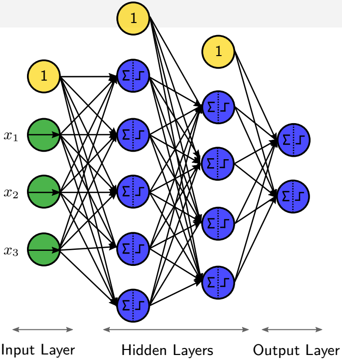
Forward Pass

Dies Ausgabe des Modells kann somit geschrieben werden als

$$\begin{aligned} f(\mathbf{x}; \mathbf{W}, \mathbf{b}) &= h\left(\mathbf{W}^{(L)}\mathbf{x}^{(L)} + \mathbf{b}^{(L)}\right) \\ &= h\left(\mathbf{W}^{(L)}\left(h\left(\mathbf{W}^{(L-1)}\mathbf{x}^{(L-1)} + \mathbf{b}^{(L-1)}\right)\right) + \mathbf{b}^{(L)}\right) = \dots \end{aligned}$$

wobei $\mathbf{W}$ und $\mathbf{b}$ alle (normalen) Gewichte und Bias-Gewichte des MLP zusammenfassen.

Ausgabe des MLP



Forward Pass

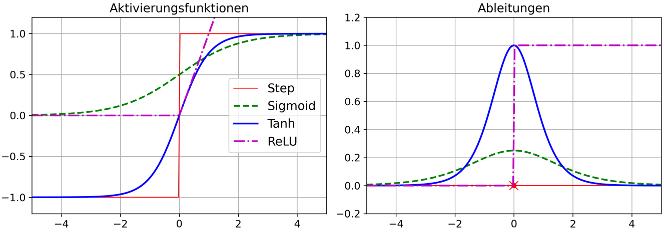
Dies Ausgabe des Modells kann somit geschrieben werden als

$$\begin{aligned} f(\mathbf{x}; \mathbf{W}, \mathbf{b}) &= h\left(\mathbf{W}^{(L)}\mathbf{x}^{(L)} + \mathbf{b}^{(L)}\right) \\ &= h\left(\mathbf{W}^{(L)}\left(h\left(\mathbf{W}^{(L-1)}\mathbf{x}^{(L-1)} + \mathbf{b}^{(L-1)}\right)\right) + \mathbf{b}^{(L)}\right) = \dots \end{aligned}$$

wobei $\mathbf{W}$ und $\mathbf{b}$ alle (normalen) Gewichte und Bias-Gewichte des MLP zusammenfassen.

Frage: Warum ist die Funktion h wichtig?

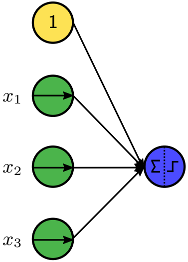
Multi-Layer Perzeptron: Aktivierungsfunktionen

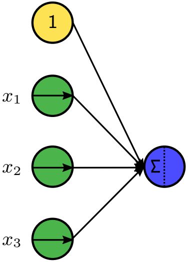


Aktivierungsfunktion

- Anstatt der Stufenfunktion h können auch andere Aktivierungsfunktionen betrachtet werden.
- Die Aktivierungsfunktionen müssen (fast überall) differenzierbar sein. Zudem ist eine Ableitung ungleich Null wichtig für das Trainieren der MLPs.

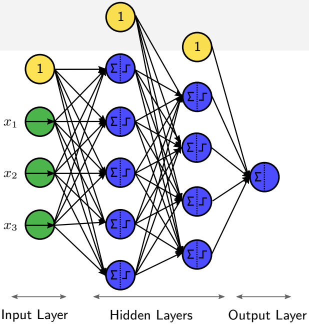
Was stellen diese MLPs dar?





Links: Sigmoid-Funktion σ als Aktivierungsfunktion; Rechts: Lineare Aktivierung $h(z) = z$.

Beispiel: Regression

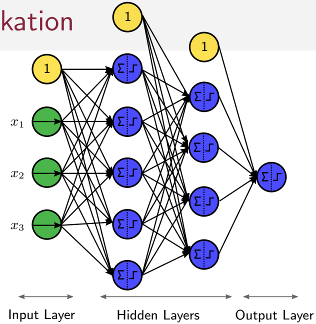


MLPs für Regression

- Trainingsdaten: $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$
- Ausgabeschicht: Im Allgemeinen lineare Aktivierungsfunktion $h_{out}(z) = z$ für das Neuron in der Ausgabeschicht.
- Verlustfunktion: Analog zur linearen Regression, z.B.

$$C(\mathbf{W}, \mathbf{b}) = \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i; \mathbf{W}, \mathbf{b}) - y_i)^2$$

Beispiel: Binäre Klassifikation

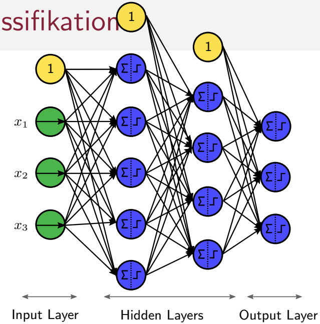


MLPs für binäre Klassifikation

- Trainingsdaten: $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \{0, 1\}$
- Ausgabeschicht: Im Allgemeinen Sigmoid-Aktivierungsfunktion $h_{out}(z) = \sigma(z)$ für das Neuron in der Ausgabeschicht.
- Verlustfunktion: Analog zur logistischen Regression, z.B.

$$C(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log f(\mathbf{x}_i; \mathbf{W}, \mathbf{b}) + (1 - y_i) \log(1 - f(\mathbf{x}_i; \mathbf{W}, \mathbf{b}))$$

Beispiel: Multiclass-Klassifikation



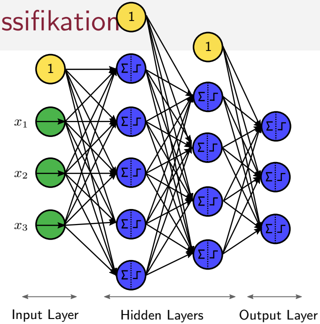
MLPs für Multiclass-Klassifikation

- Trainingsdaten: $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \{0,1\}^K$
  - One-Hot Encoding: $y_i^{(j)} = 1$ falls $\mathbf{x}_i$ zur j-ten Klasse gehört; sonst $y_i^{(j)} = 0$.
  - Beispiel: Gehört $\mathbf{x}_i$ zur zweiten von $K = 3$ Klassen, so ist $\mathbf{y}_i = [0,1,0]^\mathsf{T}$.
- Ausgabeschicht: Im Allgemeinen Softmax-Aktivierungsfunktion:

$$\hat{p}_k(\mathbf{a}) = \frac{\exp(a_k)}{\sum_{j=1}^K \exp(a_j)} \in [0,1]$$

Hierbei enthält $\mathbf{a} \in \mathbb{R}^K$ die K Ausgabewerte vor der Aktivierung.

Beispiel: Multiclass-Klassifikation



MLPs für Multiclass-Klassifikation

- Verlustfunktion: Analog zur multinomialen logistischen Regression (Kreuzentropie):

$$C(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_i^{(k)} \log(\hat{p}_k(\mathbf{a}_i))$$

Hierbei enthält $\mathbf{a}_i \in \mathbb{R}^K$ die K Ausgabewerte für $\mathbf{x}_i$ vor der Aktivierung.

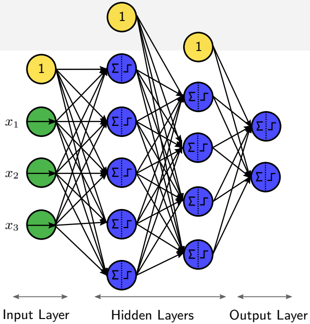
Multi-Layer Perzeptron: Training

MLP (batch gradient descent)

Require: Training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathcal{Y}$, cost function C, and learning rate $\eta > 0$.
Ensure: Weights $\mathbf{W}, \mathbf{b}$ for the model $f(\mathbf{x}; \mathbf{W}, \mathbf{b})$

```
// small random values (e.g., normally distributed)
Initialize W and b
repeat
    // gradient of the cost function based on all training instances
    Compute ∇C(W, b)
    // model parameter update
    (W, b) ← (W, b) − η∇C(W, b)
until stopping criterion is met
```

Backpropagation

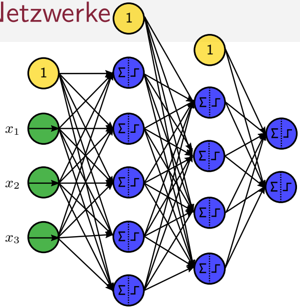


Bestimmung des Gradienten

- Backpropagation: Effiziente Methode zur Bestimmung von $\nabla C(\mathbf{W}, \mathbf{b})$.
  (d.h. Bestimmung aller partiellen Ableitungen der Modellparameter)
  1. Forward pass: Berechnung von $f(\mathbf{x}_i; \mathbf{W}, \mathbf{b})$ für eine/mehrere Trainingsinstanz(en) $\mathbf{x}_i$.
  2. Backward pass: Berechnung von $\nabla C(\mathbf{W}, \mathbf{b})$ (mittels Kettenregel, d.h., die Ableitungen werden sukzessive von „hinten nach vorne“ bestimmt).
- Automatic Differentiation (Autodiff): Moderne Implementationen ermöglichen die automatische Bestimmung des Gradienten (z.B. auf Basis von Backpropagation).

Mehr dazu in den nächsten Vorlesungen ...

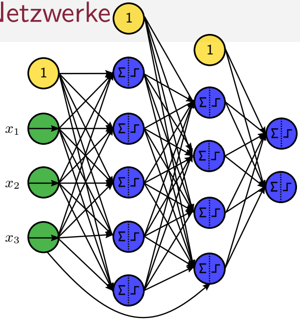
(Künstliche) Neuronale Netzwerke



Architektur

- Erweiterung von Multi-Layer Perceptrons
  - Feed-Forward-Struktur: Jedes Neuron in einer Schicht besitzt gerichtete Verbindungen zu den Neuronen in der darauffolgenden Schicht.

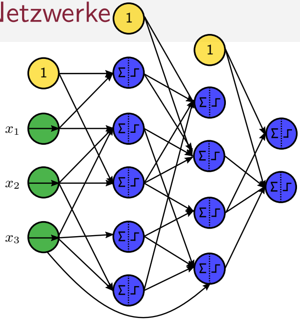
(Künstliche) Neuronale Netzwerke



Architektur

- Erweiterung von Multi-Layer Perceptrons
  - Feed-Forward-Struktur: Jedes Neuron in einer Schicht besitzt gerichtete Verbindungen zu den Neuronen in der darauffolgenden Schicht.
  - Überspringen: Weitere Verbindungen, welche Schichten „überspringen“.

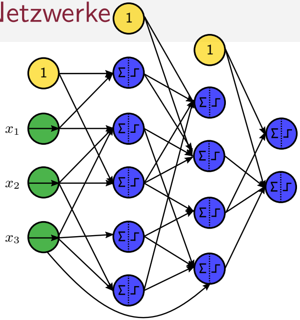
(Künstliche) Neuronale Netzwerke



Architektur

- Erweiterung von Multi-Layer Perceptrons
  - Feed-Forward-Struktur: Jedes Neuron in einer Schicht besitzt gerichtete Verbindungen zu den Neuronen in der darauffolgenden Schicht.
  - Überspringen: Weitere Verbindungen, welche Schichten „überspringen“.
  - Ausdünnung: Weglassen von Verbindungen

(Künstliche) Neuronale Netzwerke

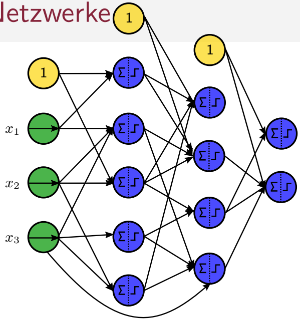


Architektur

- Erweiterung von Multi-Layer Perceptrons
  - Feed-Forward-Struktur: Jedes Neuron in einer Schicht besitzt gerichtete Verbindungen zu den Neuronen in der darauffolgenden Schicht.
  - Überspringen: Weitere Verbindungen, welche Schichten „überspringen“.
  - Ausdünnung: Weglassen von Verbindungen

$\rightarrow$ Kantenstruktur entspricht einem gerichteten azyklischen Graphen

(Künstliche) Neuronale Netzwerke

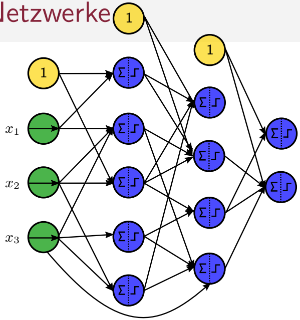


Architektur

- Erweiterung von Multi-Layer Perceptrons
  - Feed-Forward-Struktur: Jedes Neuron in einer Schicht besitzt gerichtete Verbindungen zu den Neuronen in der darauffolgenden Schicht.
  - Überspringen: Weitere Verbindungen, welche Schichten „überspringen“.
  - Ausdünnung: Weglassen von Verbindungen

  → Kantenstruktur entspricht einem gerichteten azyklischen Graphen
- Varianten: Z.B. sogenannte rekurrente Netzwerke

(Künstliche) Neuronale Netzwerke

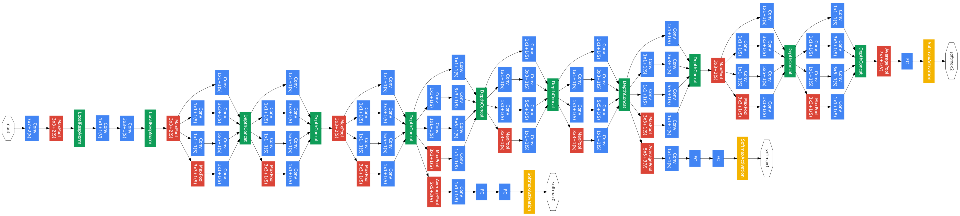


Architektur

- Erweiterung von Multi-Layer Perceptrons
  - Feed-Forward-Struktur: Jedes Neuron in einer Schicht besitzt gerichtete Verbindungen zu den Neuronen in der darauffolgenden Schicht.
  - Überspringen: Weitere Verbindungen, welche Schichten „überspringen“.
  - Ausdünnung: Weglassen von Verbindungen

  $\rightarrow$ Kantenstruktur entspricht einem gerichteten azyklischen Graphen
- Varianten: Z.B. sogenannte rekurrente Netzwerke
- Allgemein: Kombinationen aus „differenzierbaren“ Blöcken/Gates

Kombinationen aus „differenzierbaren“ Blöcken/Gates

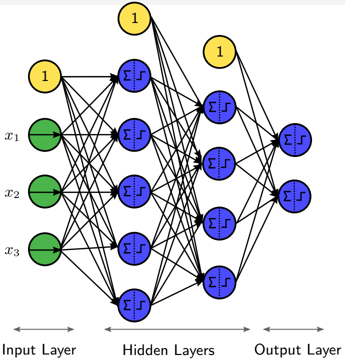


Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke und Andrew Rabinovich. *Going Deeper with Convolutions*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015.

Übersicht

Übersicht

Zusammenfassung & Ausblick



- Heute: Perzeptrons, Multi-Layer Perzeptrons, Gradientverfahren für MLPs, Implementation in Keras, ...
- Mittwoch: Neuronale Netzwerke II (Kapitel 10 + Backpropagation + ...)

Literatur I

[Gér19] Aurélien Géron.
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.
O'Reilly Media, 2 edition, 2019.