Multivariate Datenanalyse

MSc Psychologie WiSe 2021/22

Prof. Dr. Dirk Ostwald

(9) Neuronale Netze

Vorbemerkungen

Funktionale Architektur

Lernen

Backpropagation

Anwendungsbeispiel

Selbstkontrollfragen

Vorbemerkungen

Funktionale Architektur

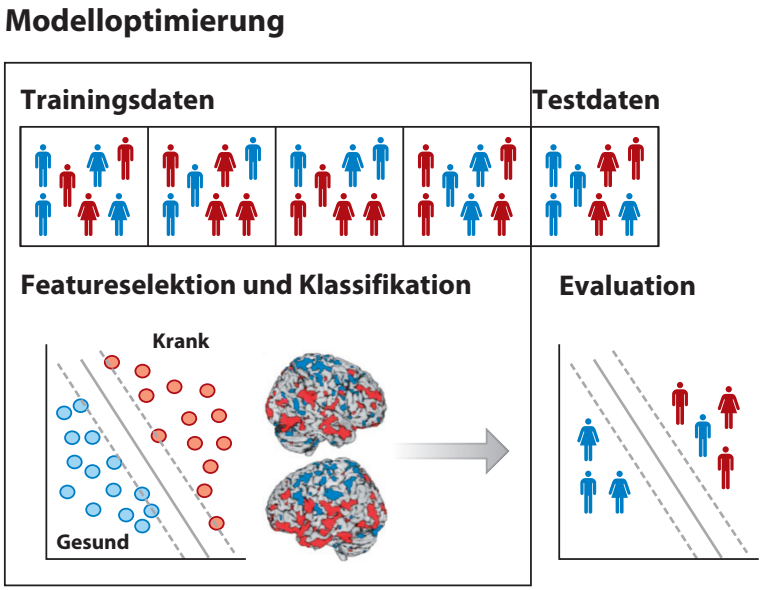
Lernen

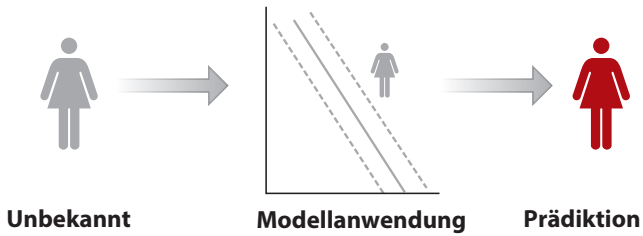
Backpropagation

Anwendungsbeispiel

Selbstkontrollfragen

Struktur der Prädiktiven Modellierung





nach Dwyer, Falkai, and Koutsouleris (2018)

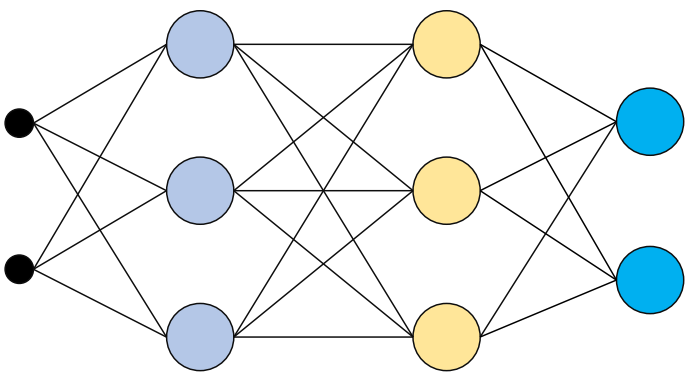
Rhetorik der Prädiktiven Modellierung

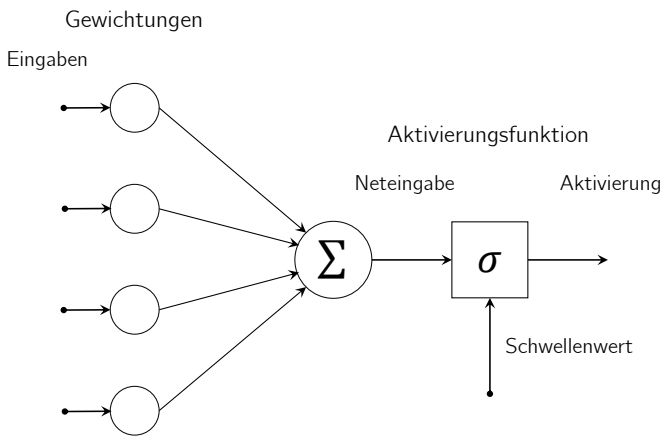
Daten	Trainingsdaten und Testdaten
Statistisches Modell	Modell, Machine Learning Algorithmus
Schätzen von Parametern	Trainieren des Modells, Lernen von Parametern, Supervised Learning

Neuronale Netze (Neural Networks)

- AKA *Künstliche Neuronale Netze (Artificial Neural Networks)*.
- Keine Modelle für biologische neuronale Netze.
- Mathematische Modelle zur Approximation multivariater vektorwertiger Funktionen.

Typische Visualisierungen





Zur Geschichte neuronaler Netze

Anfänge

- McCulloch and Pitts (1943) | Analyse der mit biologischen Neuronen möglichen logischen Operationen.
- Rosenblatt (1958) | Implementation eines Mustererkennungsalgorithmus in einem frühen Computer.
- Minsky and Papert (1969) | Mathematische Analyse der logischen Stärken und Schwächen eines Perzeptrons.

$\Rightarrow$ Erster Winter Neuronaler Netze

Erste Renaissance

- Hopfield (1982) | Mehrschichtige neuronale Netze beleben das Interesse an neuronalen Netzen erneut.
- Rumelhart, Hinton, and Williams (1986) | Popularisierung des Backpropagation Algorithmus.
- Hauptinteresse in den 1990er und 2000er Jahren im Machine Learning gilt aber SVMs und Bayesian Inference.

$\Rightarrow$ Zweiter Winter Neuronaler Netze

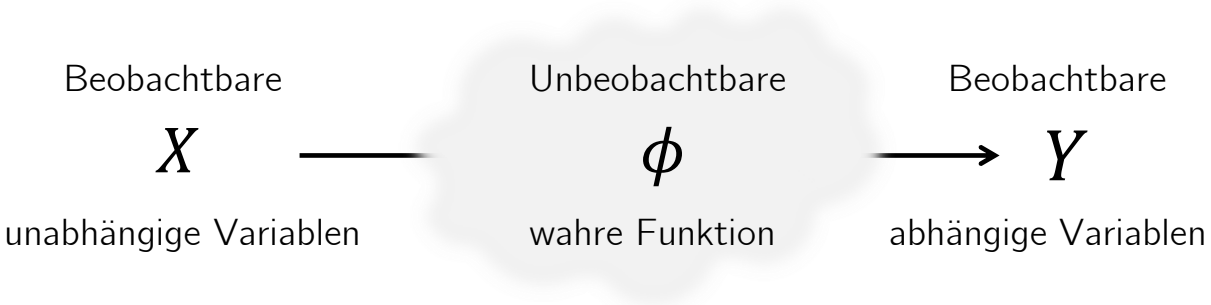
Zweite Renaissance

- 2009 - 2012 | Schmidhuber (2015) gewinnen Klassifikationswettbewerbe mit neuronalen Netzen.
- LeCun, Bengio, and Hinton (2015) | Neuronale Netze unter dem Label “Deep Learning” wieder sehr in Mode.
- 2015 - 2022 | Viele Menschen verwechseln die Begriffe “Künstliche Intelligenz” und “Neuronales Netz”.
- Ostwald and Usée (2021) | Beweis der Validität des Backpropagation Algorithmus in Matrixform.

Neuronale Netze und Prädiktive Modellierung

Explanatorische Modellierung ⇔ Wissenschaft

Bestimmung von $\hat{\phi} \coloneqq \operatorname{argmin} \|\hat{\phi} - \phi\|$



Bestimmung von $f \coloneqq \operatorname{argmin}_{\tilde{f} \in F} \|Y - \tilde{f}(X)\|$

Prädiktive Modellierung ⇔ Anwendung

⇒ Neuronale Netze zur Approximation multivariater vektorwertiger Funktionen im prädiktiven Sinn.

Universelle Approximationstheoreme

Topologische Aussagen über die Dichten von Funktionenräumen (cf. Friedman (1970)).

Neuronale Netze können eine Vielzahl von Funktionen sehr genau approximieren, wenn

- die Anzahl der Neurone gegen Unendlich geht (*arbitrary width case*) bzw.
- die Anzahl der Neuronenschichten gegen Unendlich geht (*arbitrary depth case*).

Arbitrary width case $\Rightarrow$ Cybenko (1989), Hornik (1991), Leshno et al. (1993), Pinkus (1999)

Arbitrary depth case $\Rightarrow$ Lu et al. (2017), Hanin and Sellke (2018), Kidger and Lyons (2020)

Universelle Approximationstheoreme sind Existenzaussagen, keine Konstruktionsaussagen.

$\Rightarrow$ Parameter neuronaler Netze müssen durch Gradientenverfahren gelernt werden.

Universelle Approximationstheoreme

Beispiel

Theorem (Universelles Approximationstheorem nach Kidger (2020))

$\mathcal{X}$ sei eine kompakte Teilmenge von $\mathbb{R}^m$, $\sigma : \mathbb{R} \to \mathbb{R}$ sei eine nicht-affine stetige und zumindest in einem Punkt stetig differenzierbare Funktion mit einer von Null verschiedenen Ableitung in diesem Punkt. F sei die Menge der neuronalen Netze f mit Inputdimension m, Outputdimension n_k und einer beliebigen Anzahl verdeckter Schichten mit jeweils $m + n_k + 2$ Neuronen und Aktivierungsfunktion σ, sowie der Identitätsabbildung als Aktivierungsfunktion der Outputschicht. Dann existiert zu jeder stetigen multivariaten vektorwertigen Funktion

$$g : \mathcal{X} \to \mathbb{R}^{n_k}, x \mapsto g(x) \tag{1}$$

ein neuronales Netz $f \in F$, so dass für ein beliebig kleines $\epsilon > 0$ gilt, dass

$$\sup_{x \in \mathcal{X}} \|f(x) - g(x)\| < \epsilon. \tag{2}$$

Bemerkungen

- Das Supremum $\sup$ kann intuitiv als Maximum verstanden werden.
- $\|\cdot\|$ bezeichnet eine *Metrik* (Abstandsfunktion) auf $\mathbb{R}^{n_k}$.
- Für jedes $x \in \mathcal{X}$ wird der Abstand zwischen dem Wert von g und dem Wert von f also beliebig klein.
- Man sagt dazu auch, dass F im Raum der stetigen multivariaten vektorwertigen Funktionen *dicht* ist.
- Man kann das Theorem sicherlich noch präziser formulieren und sollte es beweisen.
- Wir verzichten hier darauf und führen das Theorem nur als "intuitives Beispiel" auf.

Vorbemerkungen

Funktionale Architektur

Lernen

Backpropagation

Anwendungsbeispiel

Selbstkontrollfragen

Definition (Potentialfunktionen)

$W \in \mathbb{R}^{m \times (n+1)}$ sei eine Matrix, die wir *Wichtungsmatrix* nennen und $a \in \mathbb{R}^n$ sei ein Vektor, den wir *Aktivierungsvektor* nennen. Dann nennen wir eine Funktion der Form

$$\Phi : \mathbb{R}^{m \times (n+1)} \times \mathbb{R}^n \to \mathbb{R}^m, (W, a) \mapsto \Phi(W, a) := W \cdot \begin{pmatrix} a \\ 1 \end{pmatrix} \tag{3}$$

eine *bivariate Potentialfunktion*. Für ein festes $a \in \mathbb{R}^n$ nennen wir eine Funktion der Form

$$\Phi_a : \mathbb{R}^{m \times (n+1)} \to \mathbb{R}^m, \ W \mapsto \Phi_a(W) := \Phi(W, a) \tag{4}$$

eine *Wichtungsmatrix-variate Potentialfunktion*. Weiterhin nennen wir für eine feste Matrix $W \in \mathbb{R}^{m \times (n+1)}$ eine Funktion der Form

$$\Phi_W : \mathbb{R}^n \to \mathbb{R}^m, a \mapsto \Phi_W(a) := \Phi(W, a) \tag{5}$$

eine *Potentialfunktion*. Schließlich nennen wir $z := \Phi_W(a)$ einen *Potentialvektor*.

Definition (Aktivierungsfunktion)

Wir nennen eine multivariate vektorwertige Funktion der Form

$$\Sigma : \mathbb{R}^n \to \mathbb{R}^n, z \mapsto \Sigma(z) := (\sigma(z_1), \ldots, \sigma(z_n))^T, \tag{6}$$

mit

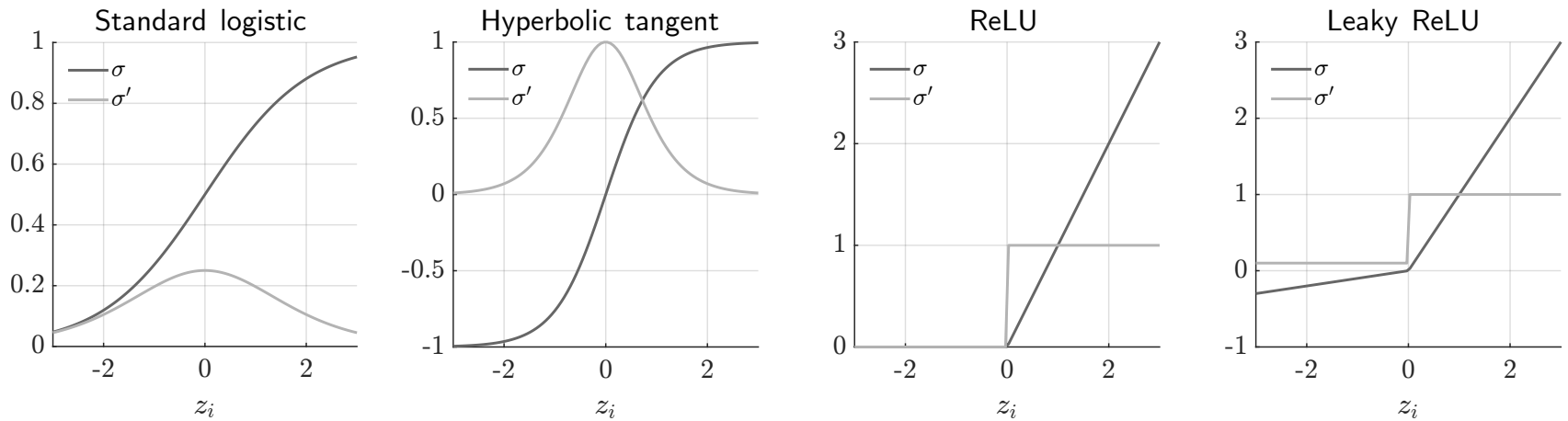
$$\sigma : \mathbb{R} \to \mathbb{R}, z_i \mapsto \sigma(z_i) =: a_i \text{ für alle } i = 1, \ldots, n, \tag{7}$$

eine *komponentenweise Aktivierungsfunktion* und die univariate reellwertige Funktion σ eine *Aktivierungsfunktion*.

Typische Aktivierungsfunktionen und ihre Ableitungen

Name	Definition	Ableitung
Standard logistic	$\sigma(z_i) := \frac{1}{1+\exp(-z_i)}$	$\sigma'(z_i) = \frac{\exp(z_i)}{(1+\exp(z_i))^2}$
Hyperbolic tangent	$\sigma(z_i) := \tanh(z_i)$	$\sigma'(z_i) = 1 - \tanh^2(z_i)$
ReLU	$\sigma(z_i) := \max(0, z_i)$	$\sigma'(z_i) = \begin{cases} 0, & z_i < 0 \\ \emptyset, & z_i = 0 \\ 1, & z_i > 0 \end{cases}$
Leaky ReLU	$\sigma(z_i) := \begin{cases} 0.1z_i, & z_i \leq 0 \\ z_i, & z_i > 0 \end{cases}$	$\sigma'(z_i) = \begin{cases} 0.01, & z_i \leq 0 \\ 1, & z_i > 0 \end{cases}$

Typische Aktivierungsfunktionen und ihre Ableitungen



Definition (k-schichtiges neuronales Netz)

Eine multivariate vektorwertige Funktion

$$f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_k}, x \mapsto f(x) =: y \tag{8}$$

heißt *k-schichtiges neuronales Netz*, wenn f von der Form

$$f : \mathbb{R}^{n_0} \xrightarrow{\Phi^1_{W^1}} \mathbb{R}^{n_1} \xrightarrow{\Sigma^1} \mathbb{R}^{n_1} \xrightarrow{\Phi^2_{W^2}} \mathbb{R}^{n_2} \xrightarrow{\Sigma^2} \mathbb{R}^{n_2} \xrightarrow{\Phi^3_{W^3}} \dots \xrightarrow{\Phi^{k-1}_{W^{k-1}}} \mathbb{R}^{n_{k-1}} \xrightarrow{\Sigma^{k-1}} \mathbb{R}^{n_{k-1}} \xrightarrow{\Phi^k_{W^k}} \mathbb{R}^{n_k} \xrightarrow{\Sigma^k} \mathbb{R}^{n_k}, \tag{9}$$

ist, wobei für $l = 1, \dots, k$

$$\Phi^l_{W^l} : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}, a^{l-1} \mapsto \Phi^l_{W^l}(a^{l-1}) := W^l \cdot \begin{pmatrix} a^{l-1} \\ 1 \end{pmatrix} =: z^l \tag{10}$$

Potentialfunktionen und

$$\Sigma^l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}, z^l \to \Sigma^l(z^l) =: a^l \tag{11}$$

komponentenweise Aktvierungsfunktionen sind. Für ein $x \in \mathbb{R}^{n_0}$ nimmt ein k-schichtiges neuronales Netz den Wert

$$f(x) := \Sigma^k(\Phi^k_{W^k}(\Sigma^{k-1}(\Phi^{k-1}_{W^{k-1}}(\Sigma^{k-2}(\dots(\Sigma^1(\Phi^1_{W^1}(x))\dots)))))) \in \mathbb{R}^{n_k}. \tag{12}$$

an.

Funktionale Architektur

Bemerkungen

- Die Vektoren $a^l = (a^l_1, \ldots, a^l_{n_l})^T \in \mathbb{R}^{n_l}, l = 0, 1, \ldots, k$ heißen *Aktivierungsvektoren der $l$ten Schicht*.
- Die Komponenten $a^l_i \in \mathbb{R}, i = 1, \ldots, n_l, l = 0, 1, \ldots, k$ heißen *Aktivierungen der $l$ten Schicht*.
- Die Schicht mit Index $l = 0$ und Dimension n_0 heißt *Inputschicht*.
- Der Aktivierungsvektor mit Index $l = 0$ heißt *Input* und wird mit $x := a^0$ bezeichnet.
- Die Schicht mit Index $l = k$ und Dimension n_k heißt *Outputschicht*
- Der Aktivierungsvektor mit Index $l = k$ heißt Output und wird mit $y := a^k$ bezeichnet.
- Die Schichten mit den Indizes $l = 1, \ldots, k-1$ heißen *verdeckte Schichten (hidden layers)*.
- $w^l_{ij} \in \mathbb{R}$ sei der ijte Eintrag der $l$ten Wichtungsmatrix, d.h.

$$W^l = (w^l_{ij})_{1 \leq i \leq n_l, 1 \leq j \leq n_{l-1}+1} \in \mathbb{R}^{n_l \times (n_{l-1}+1)} \text{ für } l = 1, \ldots, k. \tag{13}$$

- w^l_{ij} heißt *(synaptisches) Gewicht* der Verbindung von Neuron j in Schicht $l-1$ and Neuron i in Schicht l.
- Für $i = 1, \ldots, n_l$ heißt $w_{i,n_{l-1}+1}$ *Bias* von Neuron i in Schicht l.
- Die letzte Spalte von W^l enkodiert also die Biases für die Neuronen in Schicht l.

Bemerkungen

Auf der Ebene einzelner Neurone ergibt sich damit folgende Nomenklatur:

- Das *Potential* von Neuron i in Schicht l für $i = 1, \ldots, n_l$ und $l = 1, \ldots, k$ ist gegeben durch

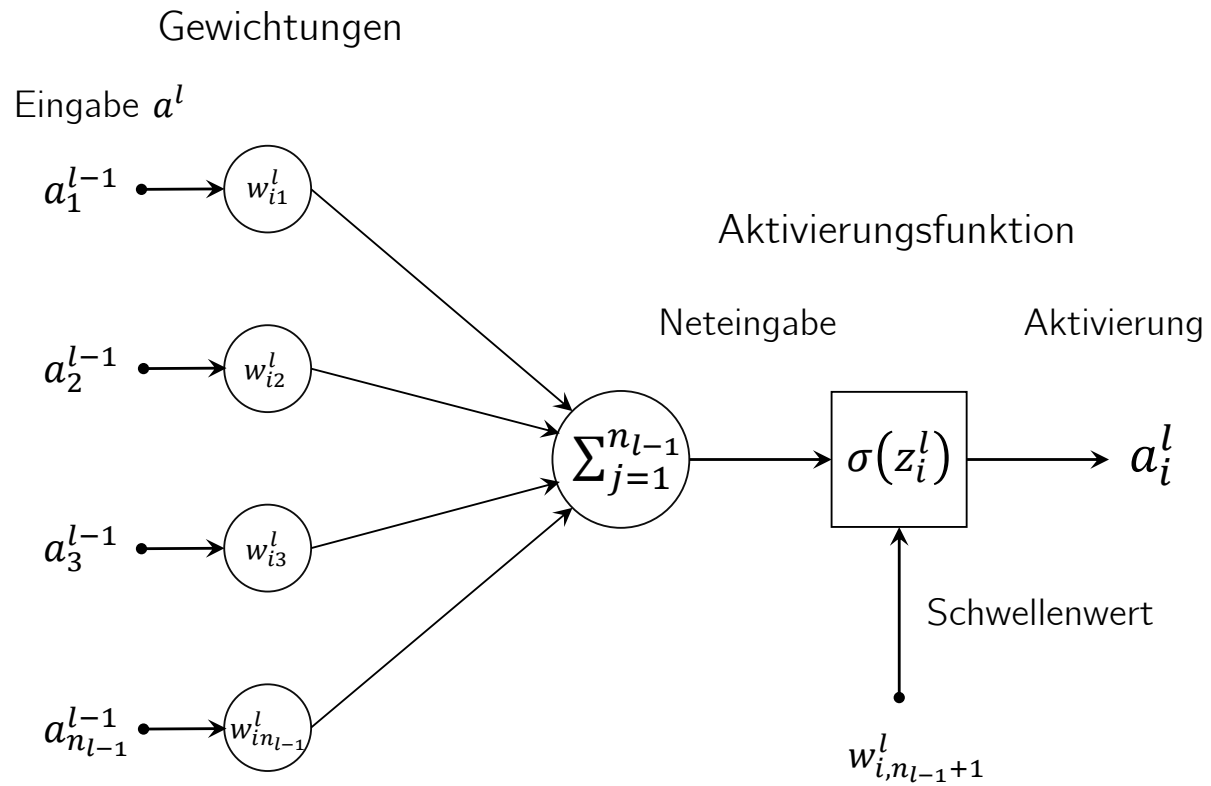
$$z_i^l = \sum_{j=1}^{n_{l-1}} w_{ij}^l a_j^{l-1} + w_{i,n_{l-1}+1} \in \mathbb{R}. \tag{14}$$

- Die *Aktivierung* von Neuron i in Schicht l für $i = 1, \ldots, n_l$ und $l = 1, \ldots, k$ ist gegeben durch

$$a_i^l = \sigma \left(\sum_{j=1}^{n_{l-1}} w_{ij}^l a_j^{l-1} + w_{i,n_{l-1}+1} \right) \in \mathbb{R}, \tag{15}$$

- Die Aktivierung a_i^l kann als die mittlere Feuerungsrate des $i$ten Neuron in der $l$ten Schicht verstanden werden.

Funktionale Architektur



Beispiel

$k = 3, n_0 = 2, n_1 = 3,\ n_2 = 3, n_3 = 2$

$$\begin{pmatrix} a^0 \\ 1 \end{pmatrix} = \begin{pmatrix} a_1^0 \\ a_2^0 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} a^1 \\ 1 \end{pmatrix} = \begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} a^2 \\ 1 \end{pmatrix} = \begin{pmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \\ 1 \end{pmatrix} \qquad a^3 = \begin{pmatrix} a_1^3 \\ a_2^3 \end{pmatrix}$$

$$W^1 = \begin{pmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \end{pmatrix} \quad W^2 = \begin{pmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 & w_{14}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 & w_{24}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 & w_{34}^2 \end{pmatrix} \quad W^3 = \begin{pmatrix} w_{11}^3 & w_{12}^3 & w_{13}^3 & w_{14}^3 \\ w_{21}^3 & w_{22}^3 & w_{23}^3 & w_{24}^3 \end{pmatrix}$$

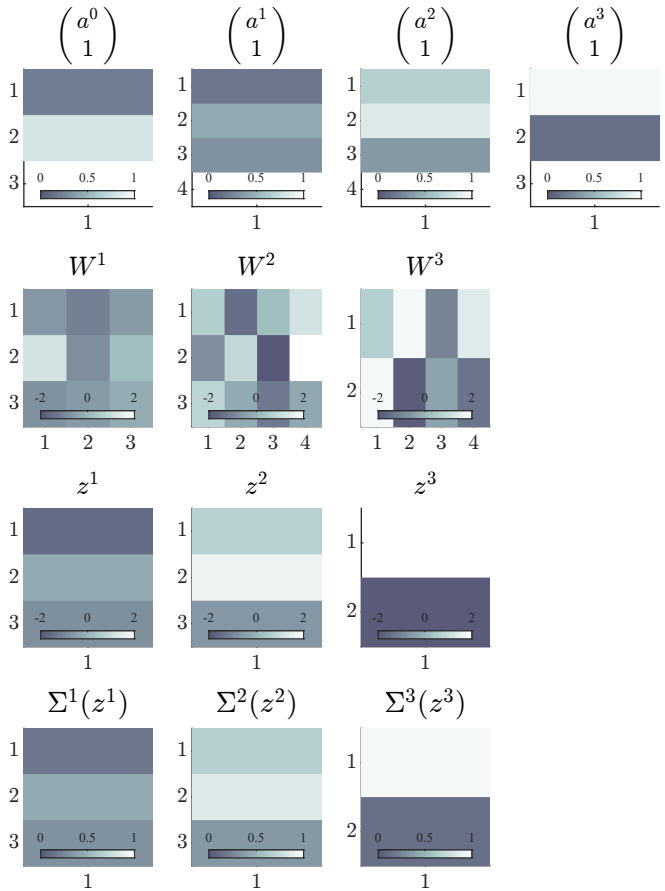
$$z^1 = \begin{pmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \end{pmatrix} \qquad z^2 = \begin{pmatrix} z_1^2 \\ z_2^2 \\ z_3^2 \end{pmatrix} \qquad z^3 = \begin{pmatrix} z_1^3 \\ z_2^3 \end{pmatrix}$$

$$\Sigma^1(z^1) = \begin{pmatrix} \sigma(z_1^1) \\ \sigma(z_2^1) \\ \sigma(z_3^1) \end{pmatrix} \qquad \Sigma^2(z^2) = \begin{pmatrix} \sigma(z_1^2) \\ \sigma(z_2^2) \\ \sigma(z_3^2) \end{pmatrix} \qquad \Sigma^3(z^3) = \begin{pmatrix} \sigma(z_1^3) \\ \sigma(z_2^3) \end{pmatrix}$$

Es gilt $x = a^0$ und $a^3 = y$.

Beispiel

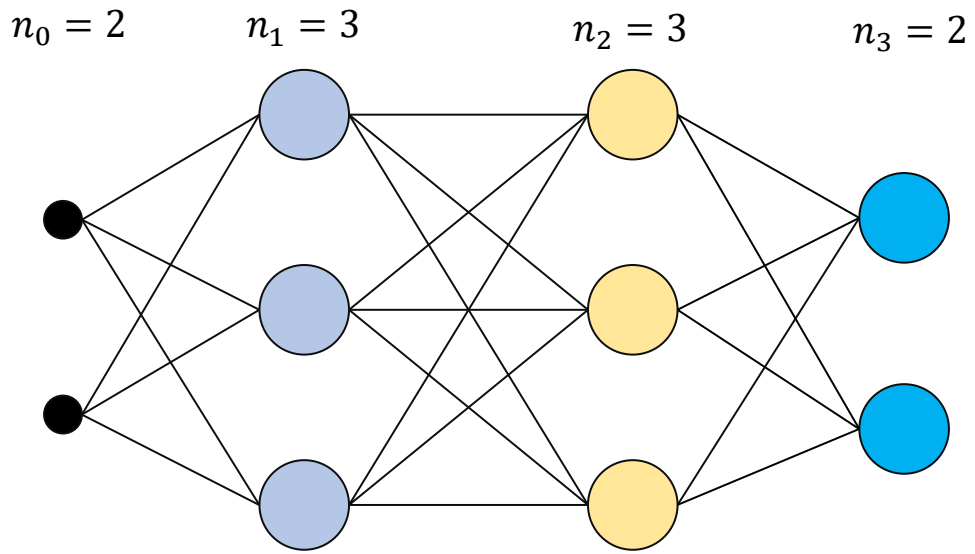
$k = 3, n_0 = 2, n_1 = 3,\ n_2 = 3, n_3 = 2$



Es gilt $x = a^0$ und $a^3 = y$.

Beispiel

$k = 3, n_0 = 2, n_1 = 3, n_2 = 3, n_3 = 2$



- Die Biases sind hier nicht visualiert.

Vorbemerkungen

Funktionale Architektur

Lernen

Backpropagation

Anwendungsbeispiel

Selbstkontrollfragen

Überblick

Anhand eines Trainingsdatensatzes werden die Parameter eines neuronalen Netzes wie folgt gelernt:

- Zunächst wird eine Funktion definiert, die misst, inwiefern sich bei einem gegebenen Inputvektor und Zielvektor der Output des neuronalen Netzes basierend auf einem Wert der Parameter unterscheidet. Diese Funktion nennt man eine *Kostenfunktion* oder *Zielfunktion*.
- Der summierte Wert der Kostenfunktion über alle Trainingsdatenpunkte wird dann durch Veränderung der Parameter minimiert, so dass Parameterwerte gefunden werden, für die die Abweichung zwischen Zielvektor und Output des neuronalen Netzes bei gegebenemen Inputvektor möglichst gering ist.
- Zur Minimierung der Kostenfunktion wird üblicherweise ein Gradientenverfahren benutzt.
- Zur Berechnung der in diesem Verfahren auftretenden Zielfunktionsgradienten wird ein komputational effizienter Algorithmus eingesetzt, der die spezielle Struktur neuronaler Netze ausnutzt und unter dem Namen *Backpropagation Algorithmus* bekannt ist.

In der Folge wollen wir die Aspekte dieses Lernprozesses genauer betrachten.

Definition (Trainingsdatensatz)

Ein *Trainingsdatensatz* für ein neuronales Netz ist eine Menge

$$\mathcal{D} := \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}, \tag{16}$$

wobei $x^{(i)} \in \mathbb{R}^{n_0}$ *Featurvektor* und $y^{(i)} \in \mathbb{R}^{n_k}$ *Zielvektor* genannt werden.

Bemerkungen

- Im Kontext der zuvor betrachteten multivariaten Verfahren gilt hier $n_0 = m$.
- Typische Zielvektorformate beim Training neuronaler Netze sind
  - $y^{(i)} \in \{0, 1\}$ für binäre Klassifikationsprobleme,
  - $y^{(i)} \in \{0, 1\}^{n_k}$ mit $\sum_{i=1}^{n_k} y_i = 1, n_k > 1$ für n_k-fache Klassifikationsprobleme,
  - $y^{(i)} \in \mathbb{R}^{n_k}, n_k > 1$ für Regressionsprobleme.

Definition (Trainieren eine neuronalen Netzes)

f sei ein k-schichtiges neuronales Netz und $\mathcal{D}$ sei ein Trainingsdatensatz. Dann bezeichnet der Begriff des *Trainierens* den Prozess der Adaptation der Wichtungsmatrizen $W^1, ..., W^k$ des neuronalen Netzes mit dem Ziel, ein Abweichungskriterium zwischen der Ouptutaktivierung $f(x^{(i)})$ und dem assoziierten Wert des Zielvektors $y^{(i)}$ über alle Trainingsdatenpunkte $(x^{(i)}, y^{(i)}), i = 1, ..., n$ des Trainingsdatensatzes $\mathcal{D}$ hinweg zu minimieren.

Bemerkungen

- Wir erinnern an das Ziel $f := \text{argmin}_{\tilde{f} \in F} \|Y - \tilde{f}(X)\|$ der prädiktiven Modellierung.
- Das erwähnte Abweichungskriterium wird in Form von *Kostenfunktionen* definiert.
- Wir benötigen noch den Begriff der *Wichtungsmatrix-variaten neuronalen Netzfunktion*.

Definition (Wichtungsmatrix-variate neuronale Netzfunktion)

f sei ein k-schichtiges neuronales Netz und x sei ein Input von f. Dann ist *Wichtungsmatrix-variate neuronale Netzfunktion f_x von f* definiert als die Funktion

$$\begin{aligned} f_x : \mathbb{R}^{n_1 \times (n_0+1)} \times \cdots \times \mathbb{R}^{n_k \times (n_{k-1}+1)} &\to \mathbb{R}^{n_k}, (W^1, \ldots, W^k) \mapsto f_x(W^1, \ldots, W^k) \\ &:= \Sigma^k(\Phi^k(W^k, \Sigma^{k-1}(\Phi^{k-1}(W^{k-1}, \cdots (W^2, \Sigma^1(\Phi^1(W^1, x))) \cdots)))), \end{aligned} \tag{17}$$

wobei für $l = 1, \ldots, k$, Φ^l die bivariate Potentialfunktion bezeichnet, die der Potentialfunktion $\Phi^l_{W^l}$ in der Definition des neuronalen Netzes entspricht. Weiterhin definieren wir für $l = 1, \ldots, k$, die *Wichtungsmatrix-variate neuronale Netzfunktion der $l$ten Schicht* f_x^l für festes fixed $W^\ell \in \mathbb{R}^{n_\ell \times (n_{\ell-1}+1)}$m mit $\ell = 1, \ldots, k$ und $\ell \neq l$ als

$$f_x^l : \mathbb{R}^{n_1 \times (n_{l-1}+1)} \to \mathbb{R}^{n_k}, W^l \mapsto f_x^l(W^l) := f_x^l(W^1, \ldots, W^k). \tag{18}$$

Bemerkungen

- Die Definition von f in der Definition eines k-schichtiges neuronales Netzes ist eine Funktion des Inputs x bei festen Wichtungsmatrizen $W^1, \ldots, W^l$. Zum Trainieren eines neuronalen Netzes ist es aber entscheidend, bei festem Input den Output des neuronalen Netzes bei Variation der Parameter $W^1, \ldots, W^l$ zu monitoren. Dies motiviert den Begriff der Wichtungsmatrix-variaten neuronalen Netzfunktion: Die Definition von f_x in (17) ist eine Funktion der Wichtungsmatrizen $W^1, \ldots, W^l$ bei festem Input x.

Definition (Output-spezifische Kostenfunktionen)

f sei ein k-schichtiges neuronales Netz und y sei ein Zielvektor von f. Dann wird eine multivariate reelwertige Funktion der Form

$$c_y : \mathbb{R}^{n_k} \to \mathbb{R}, a^k \mapsto c_y(a^k) \tag{19}$$

Output-spezifische Kostenfunktion genannt.

Bemerkung

- Eine Output-spezifische Kostenfunktion c_y misst die Abweichung des Outputs a^k eines neuronalen Netzes von einem Zielvektor y. Untenstehende Tabelle führt zwei typische Beispiele für Output-spezifische Kostenfunktionen und ihre Gradienten, die in der Folge wichtig werden, auf.

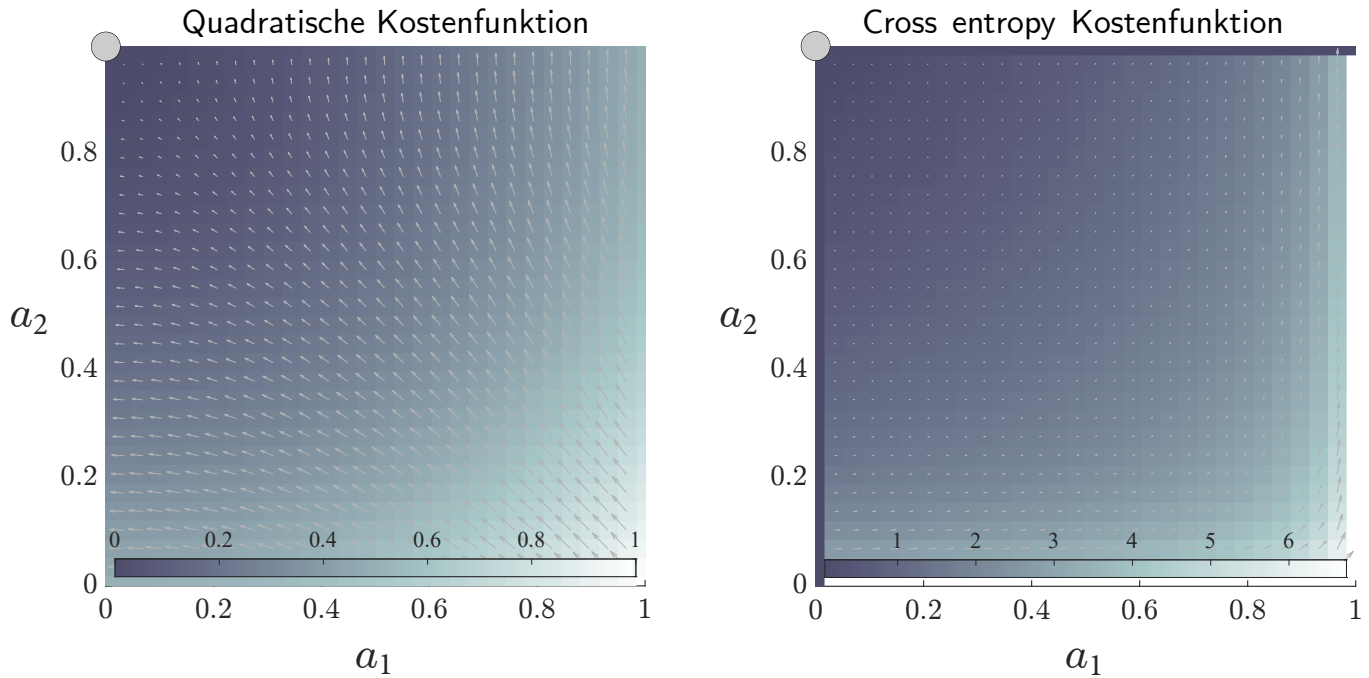
Quadratische Kostenfunktion

Definition	Gradient
$c_y(a^k) := \frac{1}{2}\sum_{j=1}^{n_k}(a_j^k - y_j)^2$	$\nabla c_y(a^k) := \left(a_j^k - y_j\right)_{j=1,\ldots,n_k}$

Cross-entropy Kostenfunktion

Definition	Gradient
$c_y(a^k) := -\sum_{i=1}^{n_k} y_j \ln a_j^k + (1-y_j)\ln(1-a_j^k)$	$\nabla c_y(a^k) := \left(-\frac{y_j}{a_j^k} + \frac{1-y_j}{1-a_j^k}\right)_{j=1,\ldots,n_k}$

Output-spezifische Kostenfunktionswerte für $y = (0, 1)^T$ bei logistischer Aktivierungsfunktion



- Beide Funktionen haben ihr Minimum bei $a = y$.
- Die Pfeile bilden die skalierten Gradientenwerte der jeweiligen Funktion ab.

Definition (Trainingsdatenpunkt-spezifische Kostenfunktionen)

f sei ein k-schichtiges neuronales Netz, f_x sei die zugehörige wichtungsmatrix-variate neuronale Netzfunktion, x und y seien Inputs und Outputs des neuronalen Netzes, $\mathcal{D}$ sei ein Trainingsdatensatz und c_y sei eine Output-spezifische Kostenfunktion. Dann heißt eine multimatrixvariate reellwertige Funktion der Form

$$c_{xy} : \mathbb{R}^{n_1 \times (n_0+1)} \times \cdots \times \mathbb{R}^{n_k \times (n_{k-1}+1)} \to \mathbb{R},$$
$$(W^1, \ldots, W^k) \mapsto c_{xy}(W^1, \ldots, W^k) := c_y(f_x(W^1, \ldots, W^k)) \quad (20)$$

Trainingsdatenpunkt-spezifische Kostenfunktion.

Bemerkung

- Eine Trainingsdatenpunkt-spezifische Kostenfunktion c_{xy} misst die Abweichung des Outputs a^k eines neuronalen Netzes von einem Zielvektor y mithilfe einer Output-spezifischen Kostenfunktion c_y für einen festen Input x als Funktion der (also bei variablen) Wichtungsmatrizen.

Definition (Gewichtsvektor)

f sei ein k-schichtige neuronales Netz mit $n_l \times n_{l-1} + 1$-dimensionalen Gewichtsmatrizen $W^l, l = 1, .., k$ und es sei

$$p := \sum_{l=1}^{n_k} n_l(n_{l-1} + 1). \tag{21}$$

die Anzahl der Gewichtsparameter des neuronalen Netzes. Dann heißt

$$\mathcal{W} := \left(\text{vec}\left(W^l\right)\right)_{1 \leq l \leq k} \in \mathbb{R}^p \tag{22}$$

der *Gewichtsvektor* des neuronalen Netzes.

Bemerkung

- Die Vektorisierung und Konkatenation der Wichtungsmatrizen im Sinne des Gewichtsvektors erlaubt es, dass Trainieren eines neuronalen Netzes als ein Standardoptimierungsproblem einer multivariaten (nicht multimatrixvariaten) reellwertigen zu formulieren.

Definition (Additive Kostenfunktion)

$\mathcal{D}$ sei ein Trainingsdatensatz und c_{xy} sei eine Trainingsdatenpunkt-spezifische Kostenfunktion. Dann nennt man eine multivariate reellwertige Funktion der Form

$$c_{\mathcal{D}} : \mathbb{R}^p \to \mathbb{R}, \mathcal{W} \mapsto c_{\mathcal{D}}(\mathcal{W}) := \frac{1}{n} \sum_{i=1}^{n} c_{x^{(i)}\, y^{(i)}}(W^1, \ldots, W^k) \tag{23}$$

eine *additive Kostenfunktion*.

Bemerkung

- Die additive Kostenfunktion ist die zentrale Zielfunktion beim Trainierung eines neuronalen Netzes.
- $c_{\mathcal{D}}$ ist eine multivariate reellwertige Funktion, es liegt also ein Standardoptimierungsproblem vor.
- Wir nehmen dabei stillschweigend an, dass die sinnvolle Aufteilung des Gewichtsvektors $\mathcal{W}$ auf die Wichtungsmatrizen $W^1, \ldots, W^k$ in der Auswertung der Funktion $c_{\mathcal{D}}$ geschieht.

Definition (Batch Gradientenverfahren für neuronale Netze)

f sei ein k-schichtiges neuronales Netz mit Gewichtsvektor $\mathcal{W}$, $\mathcal{D}$ sei ein Trainingsdatensatz bestehend aus n Trainingsdatenpunkten, und $c_{\mathcal{D}}$ sei eine additive Kostenfunktion mit assoziierter Trainingsexemplar-spezifischer Kostenfunktion c_{xy}. Dann ist ein Gradientenverfahren zur Minimierung der additiven Kostenfunktion $c_{\mathcal{D}}$ (und damit zum Lernen der Parameter von f) definiert durch

Initialisierung

Wahl eines Startpunktes $\mathcal{W}^{(0)}$ und einer Lernrate $\alpha > 0$.

Iterationen

Für $j = 1, 2, \ldots$ setze

$$\mathcal{W}^{(j)} := \mathcal{W}^{(j-1)} - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla c_{x^{(i)}\, y^{(i)}}(W^1, \ldots, W^k), \tag{24}$$

wobei

$$\nabla c_{x^{(i)}\, y^{(i)}}(W^1, \ldots, W^k) = \left(\nabla_{W^l} c_{x^{(i)}\, y^{(i)}}(W^1, \ldots, W^k) \right)_{1 \leq l \leq k} \tag{25}$$

für $i = 1, \ldots, n$ den Gradienten der $i$ten Trainingsexemplar-spezifischen Kostenfunktion bezeichnet

Bemerkungen

- $\mathcal{W}^{(j)}$ wird in (22) in die negative Richtung des Gradientenmittelwerts über Trainingsdatenpunkte adaptiert. Wird der Gradientenmittelwert dagegen nur über eine zufällig gewählte Teilmenge der Trainingsdatenpunkte berechnet, so spricht man von einem *stochastischen Gradientenverfahren*.

Vorbemerkungen

Funktionale Architektur

Training

Backpropagation

Anwendungsbeispiel

Selbstkontrollfragen

Wesen und Motivation des Backpropagation Algorithmus

Der Backpropagation (BP) Algorithmus dient der numerischen Bestimmung der Komponenten

$$\frac{\partial}{\partial w_{ij}^l} c_{xy}(W^1, ..., W^k) \text{ für alle } i = 1, ..., n_l, j = 1, ..., n_{l-1} + 1, \text{ und } l = 1, ..., k. \quad (26)$$

des Gradienten $\nabla c_{x^{(i)}y^{(i)}}(W^1, ..., W^k)$ der $i$ten Trainingsexemplar-spezifischen Kostenfunktion.

Prinzipiell können diese partiellen Ableitungen numerisch durch

$$\frac{\partial}{\partial w_{ij}^l} c_{xy}(W^1, ..., W^k) \approx \frac{c_{xy}(W^1, ..., \tilde{W}^l, ..., W^k) - c_{xy}(W^1, ..., W^l, ..., W^k)}{\epsilon}, \quad (27)$$

mit

- $\tilde{W}^l := W^l + 1_{ij}^l \epsilon$,
- einer Matrix $1_{ij}^l \in \mathbb{R}^{n_l \times (n_{l-1}+1)}$ aus 0en mit einer 1 an der w_{ij}^l Stelle in W^l und
- einem Schrittweitenparameter $\epsilon > 0$

approximiert werden (vgl. Definition der partiellen Ableitung).

Wesen und Motivation des Backpropagation Algorithmus

Dieses Vorgehen würde für jede Iteration des Gradientenverfahren und für jeden Trainingsdatenpunkt

$$K := 1 + \sum_{l=1}^{k} n_l (n_{l-1} + 1) \tag{28}$$

Auswertungen der Trainingsdatenpunkt-spezifischen Kostenfunktion c_{xy} und somit von f erfordern. Man nennt die Auswertung von f für einen Trainingsdatenpunkt x einen *Forward Pass*.

Die zentrale Eigenschaft des Backpropagation Algorithmus ist es, für die Auswertung von ∇c_{xy} die Anzahl der notwendigen *Forward Passes* pro Gradientenverfahrensiteration von K auf 1 zu reduzieren.

Um dies zu erreichen, nutzt der Backpropagation Algorithmus einen sogenannten *Backward Pass*, der die gleiche komputationale Komplexität wie der *Forward Pass* hat und auf einer multivariate Version der Kettenregel der Differentialrechnung sowie der repetitiven funktionalen Architektur neuronaler Netze beruht.

Der Backpropagation Algorithmus reduziert die Anzahl nötiger *Passes* zur Auswertung von ∇c_{xy} also von K *Forward Passes* auf 1 *Forward Pass* und 1 *Backward Pass*.

Theorem (Backpropagation Algorithmus)

f sei ein k-schichtiges neuronales Netz, $W^l_\bullet \in \mathbb{R}^{n_l \times n_{l-1}}$ seien für $l = 1, \ldots, k$ Matrizen, die durch das Entfernen der letzten Spalte der Wichtungsmatrizen $W^l \in \mathbb{R}^{n_l \times n_{l-1}+1}$ entstehen, c_{xy} sei eine Trainingsdatenpunkt-spezifische Kostenfunktion, $\nabla c_y(a^k)$ sei der Gradient der Output-spezifischen Kostenfunktion, $\tilde{\Sigma}^l(z^l) := (\sigma'(z^l_1), \ldots, \sigma'(z^l_{n_l}))^T$ sei der Vektor der Aktivierungsfunktionenableitungen ausgewertet an der Stelle z^l und $\Sigma^l(z^l)$ die komponentenweise Aktivierungsfunktion evaluiert an der Stelle z^l. Dann können die partiellen Gradienten von c_{xy} hinsichtlich der Wichtungsmatrizen W^l für $l = k, k-1, \ldots, 1$ mit folgendem Algorithmus berechnet werden:

Initialisierung

Setze $W^{k+1} := (1 \quad 0)$ und $\delta^{k+1} := \nabla c_y(a^k)$.

Iterationen

Für $l = k, k-1, k-2, \ldots, 1$, setze

$$\delta^l := \left(\left(W^{l+1}_\bullet \right)^T \cdot \delta^{l+1} \right) \circ \tilde{\Sigma}^l(z^l) \tag{29}$$

und

$$\nabla_{W^l} c_{xy}(W^1, \ldots, W^k) := \text{vec}\left(\delta^l \cdot \left(\Sigma^{l-1}(z^{l-1})^T \quad 1 \right) \right), \tag{30}$$

mit Rekursionstermination durch $\Sigma^0(z^0) := x^T$ und dem Hadamard-Produkt $\circ$.

Für weitere Details und einen Beweis verweisen wir auf Ostwald and Usée (2021).

Vorbemerkungen

Funktionale Architektur

Training

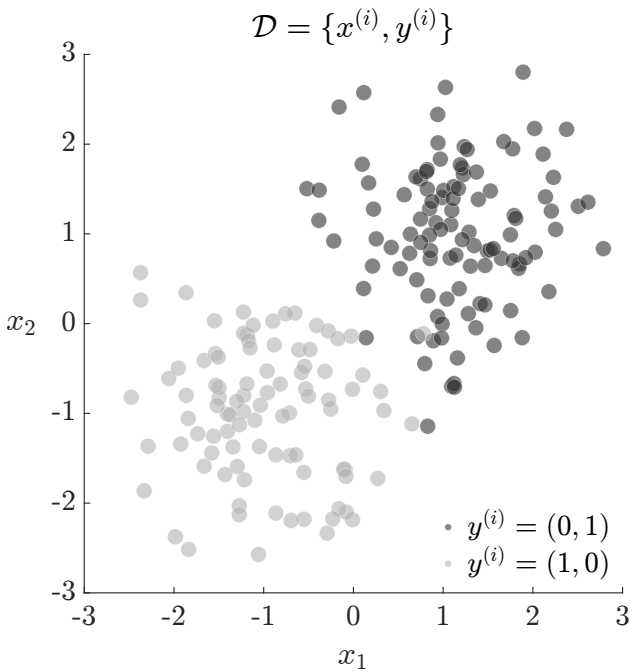
Backpropagation

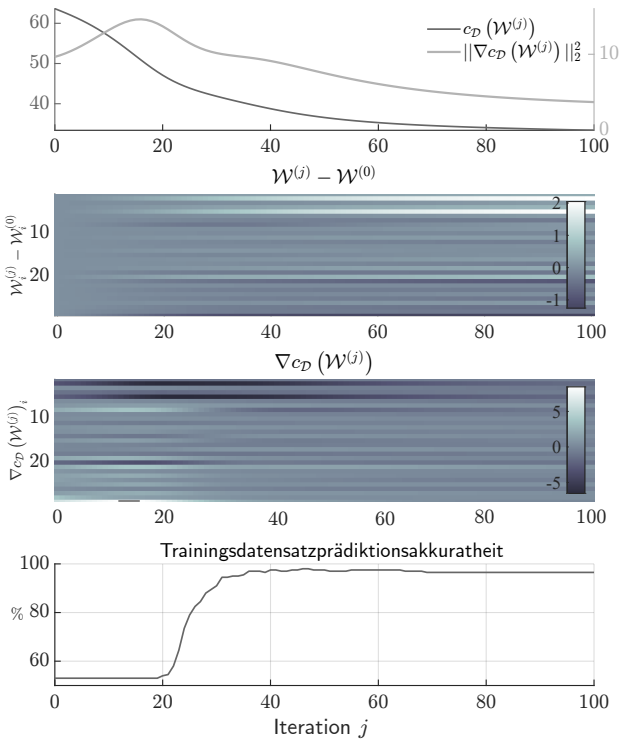
Anwendungsbeispiele

Selbstkontrollfragen

Simulation und Analyse mit Matlab Implementation (Ostwald and Usée (2021))

- Neuronales Netz mit $k = 3, n_0 = 2, n_1 = 3, n_2 = 3, n_3 = 2$.
- Trainingsdatensatz anhand eines LDA Modells simuliert.





Prädiktive Modellierung von *studienerfolg.sav* mit neuralnet() (Günther and Fritsch (2010))

$\Rightarrow$ Eine verdeckte Schicht mit zwei Neuronen

```r
# R Pakete
library(foreign)
library(neuralnet)

# Einlesen der Studienerfolgdaten und Fokus auf binäre Klassifikation
D              = read.spss(file.path(getwd(), "9_Daten", "studienerfolg.sav"),
                           to.data.frame = T)
D              = D[D[,1] != "befriedigend",]                  # Binärisierung des Datensatzes
D$Intelligenz  = D$X1                                         # Variablenbennenung
D$Mathematik   = D$X2                                         # Variablenbenennung
D$Erfolg       = c(rep(0,15), rep(1,15))                      # Recoding der Binärisierung


# Trainieren eines neuronalen Netzes mit 2 Hidden Neurons
set.seed(7)                                                   # Der Algorithmus ist nicht deterministisch!
nn             = neuralnet(Erfolg ~ Intelligenz + Mathematik, # y^{(i)}, x^{(i)} Definitionen
                           data          = D,                 # Datensatz
                           hidden        = 2,                 # 2 Neurone in 1 verdeckte Schicht
                           err.fct       = "ce",              # Cross Entropie Kostenfunktion
                           linear.output = FALSE)             # Sigmoide Aktivierungsfunktion


# Resultatsaufbereitung
R              = data.frame(nn$covariate, nn$response, nn$net.result[[1]], as.numeric(nn$net.result[[1]] > 0.5))
colnames(R)    = c("Intelligenztest", "Mathematiktest", "Erfolg", "Output", "Prädiktion")
print(sprintf("Prädiktionsakkuratheit = %0.2f", mean(R$Prädiktion == R$Erfolg)))
```

```
> [1] "Prädiktionsakkuratheit = 0.80"
```

Prädiktive Modellierung von *studienerfolg.sav* mit neuralnet() (Günther and Fritsch (2010))

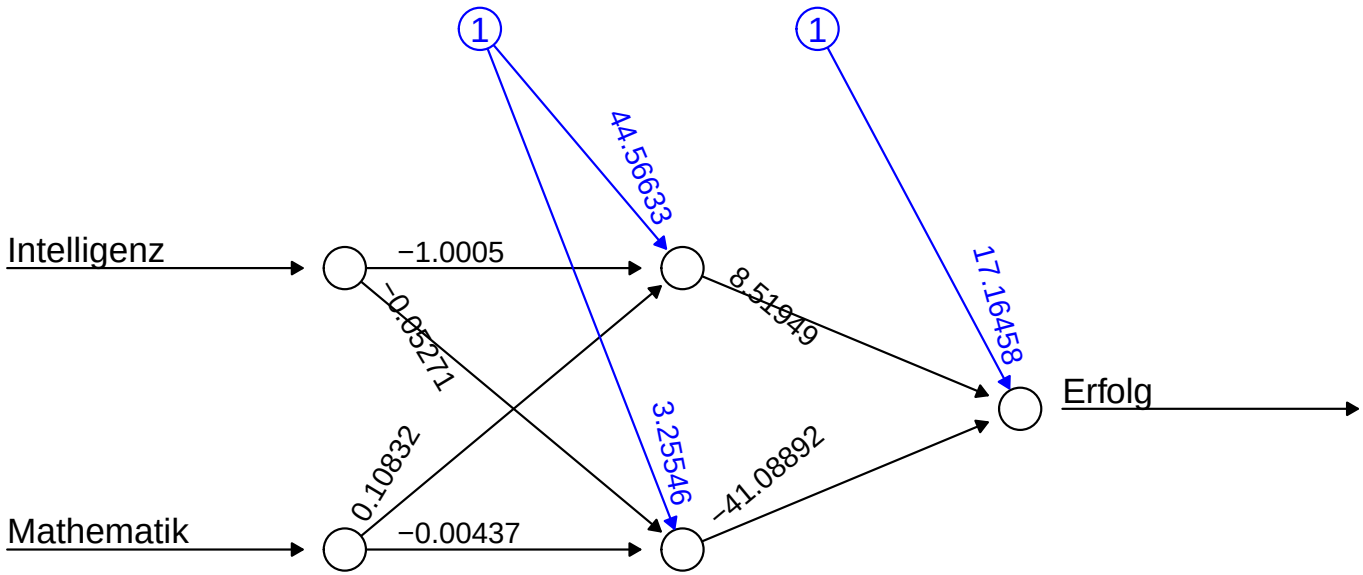
$\Rightarrow$ Eine verdeckte Schicht mit zwei Neuronen

	Intelligenztest	Mathematiktest	Erfolg	Output	Prädiktion
1	54	44	0	0.0	0
2	60	20	0	0.0	0
3	67	36	0	0.7	1
4	41	39	0	0.0	0
5	66	57	0	0.8	1
6	51	28	0	0.0	0
7	51	46	0	0.0	0
8	37	46	0	0.0	0
9	57	54	0	0.0	0
10	47	12	0	0.0	0
11	50	67	0	0.6	1
12	42	63	0	0.1	0
13	60	64	0	0.2	0
14	36	64	0	0.0	0
15	60	71	0	0.2	0
31	71	41	1	1.0	1
32	65	28	1	0.4	0
33	67	76	1	0.9	1
34	68	54	1	0.9	1
35	75	33	1	1.0	1
36	71	82	1	1.0	1
37	68	64	1	0.9	1
38	63	72	1	0.6	1
39	48	54	1	0.4	0
40	53	86	1	0.8	1
41	62	71	1	0.5	0
42	69	25	1	0.8	1
43	67	72	1	0.9	1
44	74	92	1	1.0	1
45	76	75	1	1.0	1

Prädiktive Modellierung von *studienerfolg.sav* mit neuralnet() (Günther and Fritsch (2010))

⇒ Eine verdeckte Schicht mit zwei Neuronen

```
plot(nn)
```



Error: 8.46792 Steps: 64768

Prädiktive Modellierung von *studienerfolg.sav* mit neuralnet() (Günther and Fritsch (2010))

⇒ Zwei verdeckte Schichten mit drei Neuronen

```
# R Pakete
library(foreign)
library(neuralnet)

# Einlesen der Studienerfolgdaten und Fokus auf binäre Klassifikation
D              = read.spss(file.path(getwd(), "9_Daten", "studienerfolg.sav"),
                           to.data.frame = T)
D              = D[D[,1] != "befriedigend",]                  # Binärisierung des Datensatzes
D$Intelligenz  = D$X1                                         # Variablenbennenung
D$Mathematik   = D$X2                                         # Variablenbenennung
D$Erfolg       = c(rep(0,15), rep(1,15))                      # Recoding der Binärisierung


# Trainieren eines neuronalen Netzes mit 2 Hidden Neurons
set.seed(7)                                                   # Der Algorithmus ist nicht deterministisch!
nn             = neuralnet(Erfolg ~ Intelligenz + Mathematik, # y^{(i)}, x^{(i)} Definitionen
                           data          = D,                 # Datensatz
                           hidden        = c(3,3),            # 3 Neurone in 2 verdeckten Schichten
                           err.fct       = "ce",              # Cross Entropie Kostenfunktion
                           linear.output = FALSE)             # Sigmoide Aktivierungsfunktion


# Resultstatsaufbereitung
R              = data.frame(nn$covariate, nn$response, nn$net.result[[1]], as.numeric(nn$net.result[[1]] > 0.5))
colnames(R)    = c("Intelligenztest", "Mathematiktest", "Erfolg", "Output", "Prädiktion")
print(sprintf("Prädiktionsakkuratheit = %0.2f", mean(R$Prädiktion == R$Erfolg)))
```

```
> [1] "Prädiktionsakkuratheit = 0.97"
```

Prädiktive Modellierung von *studienerfolg.sav* mit neuralnet() (Günther and Fritsch (2010))

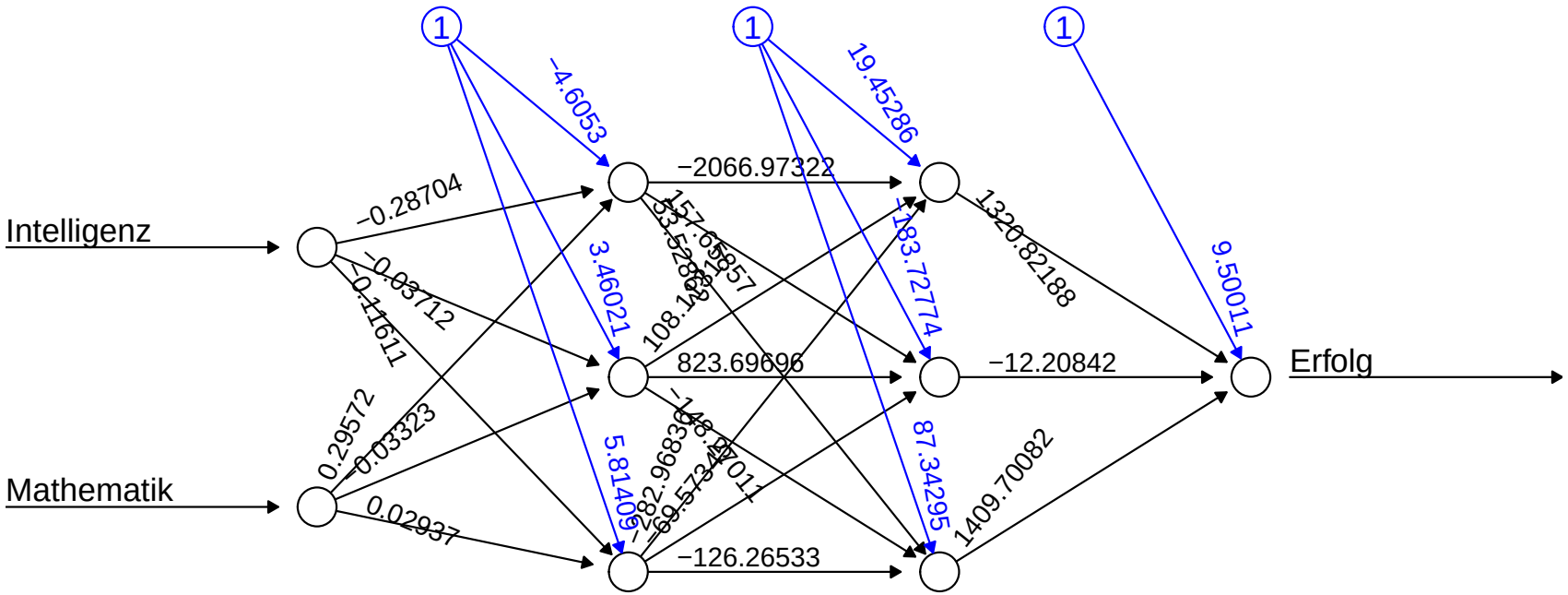
$\Rightarrow$ Zwei verdeckte Schichten mit drei Neuronen

	Intelligenztest	Mathematiktest	Erfolg	Output	Prädiktion
1	54	44	0	0.1	0
2	60	20	0	0.1	0
3	67	36	0	0.1	0
4	41	39	0	0.1	0
5	66	57	0	0.1	0
6	51	28	0	0.1	0
7	51	46	0	0.1	0
8	37	46	0	0.1	0
9	57	54	0	0.1	0
10	47	12	0	0.1	0
11	50	67	0	0.1	0
12	42	63	0	0.1	0
13	60	64	0	0.1	0
14	36	64	0	0.1	0
15	60	71	0	0.1	0
31	71	41	1	1.0	1
32	65	28	1	1.0	1
33	67	76	1	1.0	1
34	68	54	1	1.0	1
35	75	33	1	1.0	1
36	71	82	1	1.0	1
37	68	64	1	1.0	1
38	63	72	1	1.0	1
39	48	54	1	0.1	0
40	53	86	1	1.0	1
41	62	71	1	1.0	1
42	69	25	1	1.0	1
43	67	72	1	1.0	1
44	74	92	1	1.0	1
45	76	75	1	1.0	1

Prädiktive Modellierung von *studienerfolg.sav* mit neuralnet() (Günther and Fritsch (2010))

⇒ Zwei verdeckte Schichten mit drei Neuronen

```
plot(nn)
```



Vorbemerkungen

Funktionale Architektur

Lernen

Backpropagation

Anwendungsbeispiel

Selbstkontrollfragen

Selbstkontrollfragen

1. Erläutern Sie die zentralen Ideen Universeller Approximationstheoreme im Kontext neuronaler Netze.
2. Nennen Sie die Formel für das Potential z_i^l eines Neurons i in einer Schicht l eines neuronalen Netzes und erläutern Sie die verschiedenen Komponenten dieser Formel und ihre intuitive Bedeutung.
3. Skizzieren Sie die Standard Logistic und ReLU Aktivierungsfunktionen und ihre Ableitungen.
4. Nennen Sie die Formel für die Aktivierung a_i^l eines Neurons i in einer Schicht l eines neuronalen Netzes und erläutern Sie ihre Bestandteile und deren intuitive Bedeutung.
5. Erläutern Sie das prinzipielle Vorgehen zum Trainieren eines neuronalen Netzes.
6. Definieren Sie die (Output-spezifische) Quadratische Kostenfunktion und erläutern Sie ihre Bestandteile.
7. Geben Sie das Batch Gradientenverfahren zum Trainieren neuronaler Netze wieder.
8. Differenzieren Sie die Begriffe Batch und Stochastischen Gradientenverfahren zum Trainieren neuronaler Netze.
9. Erläutern Sie Wesen und Motivation des Backpropagation Algorithmus.
10. Lesen Sie den Datensatz studienerfolg.sav mit R ein und bestimmen Sie den Trainingsdatenprädiktionsfehler nach Trainieren eines neuronalen Netzes mithilfe des R Pakets neuralnet zur prädiktiven Modellierung des Studienerfolgs (gut, ungenügend) basierend auf (1) den Intelligenztestdaten, (2) den Intelligenz- und Mathematiktestdaten und (3) den Intelligenztest-, Mathematiktest-, und Gewissenhaftigkeitsdaten. Wiederholen Sie Ihre Analyse zur Bestimmung des Generalisierungsfehlers im Rahmen einer Leave-One-Out Kreuzvalidierung.

References I

Cybenko, G. 1989. "Approximation by Superpositions of a Sigmoidal Function." *Mathematics of Control, Signals, and Systems*, 12.

Dwyer, Dominic B., Peter Falkai, and Nikolaos Koutsouleris. 2018. "Machine Learning Approaches for Clinical Psychology and Psychiatry." *Annual Review of Clinical Psychology* 14 (1): 91–118. https://doi.org/10.1146/annurev-clinpsy-032816-045037.

Friedman, Avner. 1970. *Foundations of Modern Analysis*. Dover Publications.

Günther, Frauke, and Stefan Fritsch. 2010. "Neuralnet: Training of Neural Networks." *The R Journal* 2 (1): 30. https://doi.org/10.32614/RJ-2010-006.

Hanin, Boris, and Mark Sellke. 2018. "Approximating Continuous Functions by ReLU Nets of Minimal Width." *arXiv:1710.11278 [Cs, Math, Stat]*, March. https://arxiv.org/abs/1710.11278.

Hopfield, J. J. 1982. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities." *Proceedings of the National Academy of Sciences* 79 (8): 2554–58. https://doi.org/10.1073/pnas.79.8.2554.

Hornik, Kurt. 1991. "Approximation Capabilities of Multilayer Feedforward Networks." *Neural Networks* 4 (2): 251–57. https://doi.org/10.1016/0893-6080(91)90009-T.

Kidger, Patrick, and Terry Lyons. 2020. "Universal Approximation with Deep Narrow Networks." *arXiv:1905.08539 [Cs, Math, Stat]*, June. https://arxiv.org/abs/1905.08539.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. "Deep Learning." *Nature* 521 (7553): 436–44. https://doi.org/10.1038/nature14539.

Leshno, Moshe, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. 1993. "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function." *Neural Networks* 6 (6): 861–67. https://doi.org/10.1016/S0893-6080(05)80131-5.

Lu, Zhou, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. "The Expressive Power of Neural Networks: A View from the Width." *arXiv:1709.02540 [Cs]*, November. https://arxiv.org/abs/1709.02540.

McCulloch, Warren S, and Walter Pitts. 1943. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics* 5: 115–33.

Minsky, Marvin, and Seymour A. Papert. 1969. *Perceptrons: An Introduction to Computational Geometry*. 2. print. with corr. Cambridge/Mass.: The MIT Press.

Ostwald, Dirk, and Franziska Usée. 2021. "An Induction Proof of the Backpropagation Algorithm in Matrix Notation." *arXiv:2107.09384 [Cs, Math, q-Bio, Stat]*, July. https://arxiv.org/abs/2107.09384.

Pinkus, Allan. 1999. "Approximation Theory of the MLP Model in Neural Networks." *Acta Numerica* 8 (January): 143–95. https://doi.org/10.1017/S0962492900002919.

References II

Rosenblatt, F. 1958. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological Review* 65 (6): 386–408. https://doi.org/10.1037/h0042519.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1986. "Learning Representations by Back-Propagating Errors." *Nature*, no. 323: 533–36.

Schmidhuber, Jürgen. 2015. "Deep Learning in Neural Networks: An Overview." *Neural Networks* 61 (January): 85–117. https://doi.org/10.1016/j.neunet.2014.09.003.