



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Mathematik

Masterarbeit

**Analyse der Konvergenzgeschwindigkeit eines einfach
berechenbaren Neuronale-Netze-Regressionsschätzers**

Adrian Gabel

12. März 2020

Betreuer: Prof. Dr. Michael Kohler

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Adrian Gabel, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Darmstadt, 12. März 2020

Adrian Gabel

Inhaltsverzeichnis

Einleitung	6
1 Grundlagen und Hilfsresultate	8
1.1 Definitionen	8
1.2 Hilfsresultate	12
2 Konstruktion des Neuronale-Netze-Schätzers	21
2.1 Definition der Netzwerkarchitektur	21
2.2 Definition der Gewichte der Ausgabeschicht	31
3 Resultat zur Konvergenzgeschwindigkeit	35
4 Anwendungsbeispiel auf simulierte Daten	48
Literaturverzeichnis	52
Appendix	53

Einleitung

Erfinder träumen schon lange davon, Maschinen zu schaffen, die denken. Dieser Wunsch geht zumindest auf die Zeit des antiken Griechenlands zurück. Die mythischen Figuren Pygmalion, Daedalus und Hephaestus können alle als legendäre Erfinder interpretiert werden, und Galatea, Talos und Pandora können alle als künstliches Leben betrachtet werden (Ovid und Martin, 2004; Sparkes, 1996; Tandy, 1997).

Als programmierbare Computer zum ersten Mal konzipiert wurden, fragten sich die Menschen, ob solche Maschinen intelligent werden könnten, mehr als hundert Jahre bevor man sie baute (Lovelace, 1842). Heute ist die künstliche Intelligenz (KI) ein blühendes Feld mit vielen praktischen Anwendungen und aktiven Forschungsthemen. Künstliche Intelligenz ist längst in unserem Alltag präsent und dringt in immer mehr Bereiche vor. Sprachassistenten etwa sind bereits als Helfer auf dem Smartphone, im Auto oder zu Hause Normalität geworden. Fortschritte im Bereich der KI beruhen vor allem auf der Verwendung neuronaler Netze. Vergleichbar mit der Funktionsweise des menschlichen Gehirns verknüpfen sie mathematisch definierte Einheiten miteinander.

Es besteht eine große Lücke zwischen den Schätzungen, für die schönen Konvergenzergebnisse die in der Theorie nachgewiesen wurden, und den Schätzungen, die in der Praxis verwendet werden können.

Maschinelle Lernverfahren können *als Lernen einer Funktion* (f) zusammengefasst werden, die Eingangsvariablen (X) auf Ausgangsvariablen (Y) abbildet.

$$Y = f(x)$$

Ein Algorithmus lernt diese Zielabbildungsfunktion aus Trainingsdaten. Die Form der Funktion ist unbekannt, so dass es unsere Aufgabe als Praktiker des maschinellen Lernens ist, verschiedene Algorithmen des maschinellen Lernens zu evaluieren und zu sehen, welcher die zugrunde liegende Funktion besser annähert. Unterschiedliche Algorithmen machen unterschiedliche Annahmen oder Vorurteile über die Form der Funktion und die Art und Weise, wie sie gelernt werden kann.

Algorithmen, die keine starken Annahmen über die Form der Abbildungsfunktion treffen, werden als nichtparametrische Algorithmen des maschinellen Lernens bezeichnet. Indem

sie keine Annahmen treffen, können sie jede beliebige Funktionsform aus den Trainingsdaten lernen. Nichtparametrische Methoden sind gut, wenn Sie über viele Daten und keine Vorkenntnisse verfügen und wenn Sie sich nicht allzu sehr um die Auswahl der richtigen Funktionen kümmern wollen. (Referenz - Artificial Intelligence: A Modern Approach, Seite 757)

Ziel dieser Arbeit ist es, die folgende Frage genauer zu betrachten: Wenn wir eine Regressionsschätzung des neuronalen Netzes theoretisch genau so definieren, wie sie in der Praxis umgesetzt wird, welches Konvergenzresultat können wir dann für diese Schätzung vorweisen?

Als erstes werden wir in Kapitel 1 grundlegende Definition und Hilfsresultate für den weiteren Verlauf der Arbeit sammeln. Anschließend definieren wir in Kapitel 2 eine neue Regressionsschätzung für neuronale Netze, bei der die meisten Gewichte unabhängig von den Daten gewählt werden, die durch einige neuere Approximationsergebnisse für neuronale Netze motiviert sind, und die daher leicht zu implementieren ist. In Kapitel 3 zeigen wir dann unser Hauptresultat, dass wir für diese Schätzung Konvergenzraten ableiten können, falls die Regressionsfunktion glatt ist.

Mit diesem Hauptergebnis ist es nun möglich einfach zu implementierende Regressionsverfahren für neuronale Netze zu definieren, die die gleiche Konvergenzrate wie lineare Regressionsschätzungen (z.B. Kernel- oder Spline-Schätzungen) erreichen, d.h. sie erreichen (bis zu einem logarithmischen Faktor) die optimale Minimax-Konvergenzrate $n - 2p/(2p + d)$ im Falle einer (p, C) -glatten Regressionsfunktion, für jedes $p > 0$. Abschließend wird die Leistung unseres neu vorgeschlagenen Schätzers für simulierte Daten in Kapitel 4 veranschaulicht. Diese Arbeit orientiert sich an [AB19].

Kapitel 1

Grundlagen und Hilfsresultate

Der Zweck dieses Kapitels ist es, grundlegende Definitionen zu sammeln, die in den folgenden Kapiteln verwendet werden. Weiterhin werden wir Hilfsresultate darstellen und beweisen, welche wir für das Resultat der Konvergenzgeschwindigkeit des einfach berechenbaren Neuronale-Netze-Regressionsschätzer benötigen werden.

1.1 Definitionen

Es ist bekannt, dass man Glattheitsvoraussetzungen an die Regressionsfunktion haben muss, um nichttriviale Konvergenzresultate für nichtparametrische Regressionsschätzer herzuleiten. Dafür verwenden wir die folgende Definition.

Definition 1.1 ((p, C) -Glattheit). Sei $p = q + s$ mit $q \in \mathbb{N}_0$ und $s \in (0, 1]$ (also $p \in (0, \infty)$) und sei $C > 0$. Eine Funktion $f: \mathbb{R}^d \rightarrow \mathbb{R}$ heißt (p, C) -glatt, falls für alle $\alpha = (\alpha_1, \dots, \alpha_d) \in \mathbb{N}_0^d$ mit $\sum_{j=1}^d \alpha_j = q$ die partielle Ableitung

$$\frac{\partial^q f}{\partial x_1^{\alpha_1} \dots \partial x_d^{\alpha_d}}$$

existiert und falls für alle $x, z \in \mathbb{R}^d$ die Abschätzung

$$\left| \frac{\partial^q f}{\partial x_1^{\alpha_1} \dots \partial x_d^{\alpha_d}}(x) - \frac{\partial^q f}{\partial x_1^{\alpha_1} \dots \partial x_d^{\alpha_d}}(z) \right| \leq C \cdot \|x - z\|^r,$$

gilt, wobei $\|\cdot\|$ die euklidische Norm ist.

Bemerkung 1.2. Im Falle von $p \leq 1$ ist eine Funktion genau dann (p, C) -glatt, wenn sie hölderstetig ist mit *Exponent* p und *Hölderkonstante* C .

Da wir uns in dieser Arbeit mit neuronalen Netzen beschäftigen, ist es hilfreich zu wissen was man darunter versteht. Ein neuronales Netz ist nichts anderes als eine Ansammlung

von Neuronen, welche als Informationsverarbeitungseinheiten dienen, die schichtweise in einer Netzwerkarchitektur angeordnet sind. Beginnend mit der Eingabeschicht (*Input Layer*) fließen Informationen über eine oder mehrere verborgene Schichten (*Hidden Layer*) bis hin zur Ausgabeschicht (*Output Layer*). Die Informationsweitergabe der Neuronen verläuft so, dass sie die Eingaben x_1, \dots, x_n , die einerseits aus dem beobachteten Prozess resultieren können, dessen Werte dem Neuron übergeben werden, oder wiederum aus den Ausgaben anderer Neuronen stammen, verarbeiten und entsprechend über seine Aktivierung reagieren. Dazu werden für ein künstliches Neuron j die Eingaben mit w_{1j}, \dots, w_{nj} gewichtet an eine Aktivierungsfunktion σ übergeben, welche die Neuronenaktivierung berechnet. Der Endpunkt des Informationsflusses in einem neuronalen Netz ist die Ausgabeschicht, die hinter den verborgenen Schichten liegt. Sie bildet damit die letzte Schicht in einem neuronalen Netz. Die Ausgabeschicht enthält somit das Ergebnis der Informationsverarbeitung durch das Netz. Zwei wichtige Charakteristika, die neuronale Netze aufweisen können, sind:

- Wenn in einem neuronalen Netz die Information von der Eingabeschicht über die verborgenen Schichten bis hin zur Ausgabeschicht in eine Richtung („vorwärts“) weitergereicht wird, spricht man von einem *feedforward* neuronalen Netz.
- Ein neuronales Netz wird als ein *fully connected* („vollständig verbundenes“) neuronales Netz bezeichnet, wenn sämtliche Neuronen einer Schicht mit allen der darauffolgenden verbunden sind.

Für ein neuronales Netz sind die Aktivierungsfunktionen von großer Bedeutung, da sie dabei helfen, das wirklich komplizierte und nichtlineare komplexe funktionale Mapping zwischen den Eingangsdaten und den abhängigen Ergebnissen zu verstehen.

Definition 1.3. Eine *Aktivierungsfunktion* ist eine Transformation, die wir über die Eingabe durchführen, bevor wir sie an die nächste Schicht von Neuronen senden oder sie als Ausgabe finalisieren.

Definition 1.4. Ein *neuronales Netz* ist eine Funktion $f: X \rightarrow Y$, wobei $f(x)$ für $x \in X$, definiert ist als Komposition von weiteren Funktionen $g_i: X \rightarrow Y$, die weiter in andere Funktionen zerlegt werden können. Die Funktion f ist eine nichtlinear gewichtete Summe, mit $f(x) = \sigma(\sum_i w_i g_i(x))$, wobei w_i die Gewichte sind und σ eine Aktivierungsfunktion ist.

Als nächstes Definieren wir, was wir unter einer Netzwerkarchitektur verstehen um damit anschließend eine Definition eines mehrschichtigen feedforward neuronalen Netzes zu konstruieren.

Definition 1.5. Eine *Netzwerkarchitektur* (L, \mathbf{k}) ist eine Klasse aller neuronaler Netze f für die gilt:

- Das neuronale Netz hat $L \in \mathbb{N}_0$ verborgene Schichten.
- Die Anzahl an Neuronen in jeder verborgenen Schicht wird durch den Vektor $\mathbf{k} = (k_1, \dots, k_L) \in \mathbb{N}^L$ angegeben.

Mit dieser Definition können wir nun sagen, dass eine Netzwerkarchitektur wie ein Grundgerüst für die Konstruktion weiterer neuronale Netze ist.

Definition 1.6. Ein *mehrschichtiges feedforward neuronales Netz mit Architektur* (L, \mathbf{k}) und *Aktivierungsfunktion* σ , ist eine reellwertige Funktion $f: \mathbb{R}^d \rightarrow \mathbb{R}$ definiert durch:

$$f(x) = \sum_{i=1}^{k_L} c_i^{(L)} \cdot f_i^{(L)}(x) + c_0^{(L)}$$

mit Gewichten $c_0^{(L)}, \dots, c_{k_L}^{(L)} \in \mathbb{R}$ und für $f_i^{(L)}$ mit $i = 1, \dots, k_L$ rekursiv definiert durch:

$$f_i^{(r)}(x) = \sigma \left(\sum_{j=1}^{k_{r-1}} c_{i,j}^{(r-1)} \cdot f_j^{(r-1)}(x) + c_{i,0}^{(r-1)} \right) \quad (1.1)$$

mit Gewichten $c_{i,0}^{(r-1)}, \dots, c_{i,k_{r-1}}^{(r-1)} \in \mathbb{R}$ mit $r = 2, \dots, L$ und:

$$f_i^{(1)}(x) = \sigma \left(\sum_{j=1}^d c_{i,j}^{(0)} \cdot x^{(j)} + c_{i,0}^{(0)} \right)$$

für die Gewichte $c_{i,0}^{(0)}, \dots, c_{i,d}^{(0)} \in \mathbb{R}$.

Da wir bei Neuronale-Netze-Regressionsschätzern Funktionswerte schätzen möchten, haben wir in der Ausgabeschicht nur ein Neuron. Damit erreichen wir einen eindimensionalen Output und nehmen als Aktivierungsfunktion die Identität. Es wäre auch möglich für jede Schicht eine andere Aktivierungsfunktion zu wählen. In dieser Arbeit beschränken wir uns nur auf den Fall in welchem wir in allen verborgenen Schichten die selbe Aktivierungsfunktion verwenden.

Abbildung 1.1 zeigt schematisch ein mehrschichtiges fully connected feedforward neuronales Netz, welches aus einer Eingabeschicht (*Input feature 1 - Input feature n_{in}*), n_{lyr} verborgenen Schichten und einer Ausgabeschicht (*Output 1 - Output n_{out}*) besteht.

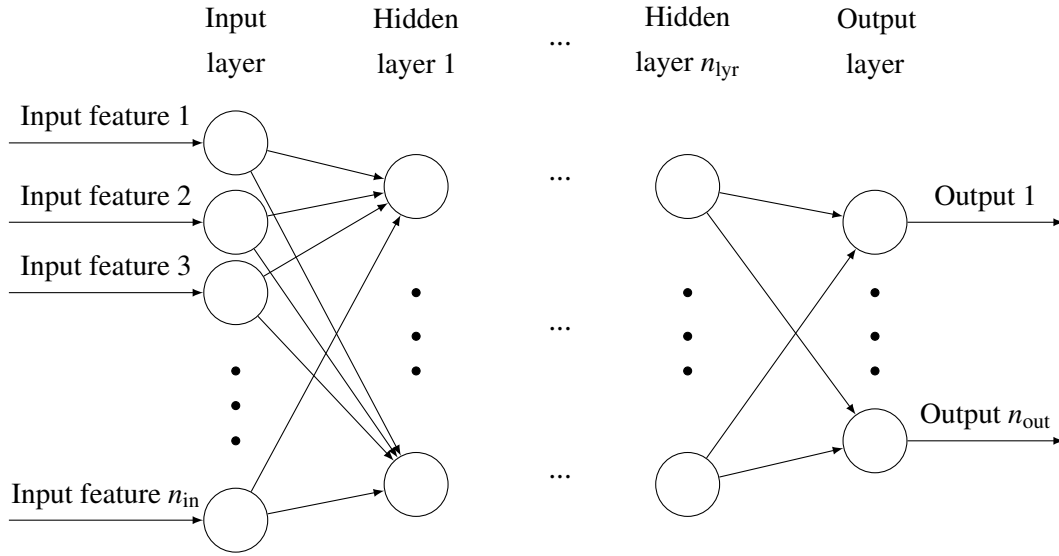


Abbildung 1.1: Fully connected feedforward neuronales Netz mit einer Eingabeschicht mit n_{in} Neuronen, n_{lyr} vielen verborgenen Schichten dessen Anzahl an Neuronen variieren kann und einer Ausgabeschicht bestehend aus n_{out} Neuronen.

Wie zuvor erwähnt ist eines der Ausgangspunkte für die Definition eines neuronalen Netzes die Wahl einer Aktivierungsfunktion $\sigma: \mathbb{R} \rightarrow \mathbb{R}$. Wir haben uns in dieser Arbeit für die sogenannten *squashing functions* entschieden, welche eine monoton wachsend Funktion ist, für die $\lim_{x \rightarrow -\infty} \sigma(x) = 0$ und $\lim_{x \rightarrow \infty} \sigma(x) = 1$ gilt. Ein Beispiel für eine squashing function ist der sogenannte *sigmoidal* bzw. *logistische squasher*

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (x \in \mathbb{R}). \quad (1.2)$$

Definition 1.7. Sei $N \in \mathbb{N}_0$. Eine Funktion $\sigma: \mathbb{R} \rightarrow [0, 1]$ wird *N-zulässig* genannt, wenn sie monoton wachsend und lipschitzsteig [For16] ist und wenn zusätzlich die folgenden drei Bedingungen erfüllt sind:

- (i) Die Funktion σ ist $N + 1$ mal stetig differenzierbar mit beschränkten Ableitungen.
- (ii) Es existiert ein Punkt $t_\sigma \in \mathbb{R}$, in welchem alle Ableitungen bis hin zur N -ten Ableitung von σ ungleich Null sind.
- (iii) Wenn $y > 0$ ist, gilt $|\sigma(y) - 1| \leq \frac{1}{y}$. Wenn $y < 0$ ist, gilt $|\sigma(y)| \leq \frac{1}{|y|}$.

In Lemma 1.10 werden wir zudem zeigen, dass der logistische squasher (1.2) *N-zulässig* ist für beliebiges $N \in \mathbb{N}_0$.

Als geben wir eine Definition von wir Überdeckungszahlen an, da wir im Beweis für unser Hauptresultat eine Abschätzung einer L_p - ε -Überdeckungszahl anwenden.

Definition 1.8. (X, d) sei ein halbmetrischer Raum [For16] . Für $x \in X$ und $\varepsilon > 0$ sei:

$$U_\varepsilon(x) = \{z \in X : d(x, z) < \varepsilon\}$$

die Kugel um x mit Radius ε .

a) $\{z_1, \dots, z_N\} \subseteq X$ heißt ε -Überdeckung einer Menge $A \subseteq X$, falls gilt:

$$A \subseteq \bigcup_{k=1}^N U_\varepsilon(z_k).$$

b) Ist $A \subseteq X$ und $\varepsilon > 0$, so ist die sogenannte ε -Überdeckungszahl von A in (X, d) definiert als:

$$\mathcal{N}_{(X, d)}(\varepsilon, A) = \inf \{ |U| : U \subseteq X \text{ ist } \varepsilon\text{-Überdeckung von } A \}.$$

Definition 1.9. Sei \mathcal{F} eine Menge von Funktionen $f: \mathbb{R}^d \rightarrow \mathbb{R}$, sei $\varepsilon > 0$, $1 \leq p < \infty$ und seien $x_1, \dots, x_n \in \mathbb{R}^d$ und $x_1^n = (x_1, \dots, x_n)$. Dann ist die L_p - ε -Überdeckungszahl von \mathcal{F} auf x_1^n definiert durch:

$$\mathcal{N}_p(\varepsilon, \mathcal{F}, x_1^n) := \mathcal{N}_{(X, d)}(\varepsilon, \mathcal{F}),$$

wobei der halbmetrische Raum (X, d) gegeben ist durch

- $X =$ Menge aller Funktionen $f: \mathbb{R}^d \rightarrow \mathbb{R}$,
- $d(f, g) = d_p(f, g) = (\frac{1}{n} \sum_{i=1}^n |f(x_i) - g(x_i)|^p)^{1/p}$.

In anderen Worten: $\mathcal{N}_p(\varepsilon, \mathcal{F}, x_1^n)$ ist das minimal $N \in \mathbb{N}$, so dass Funktionen $f_1, \dots, f_N: \mathbb{R}^d \rightarrow \mathbb{R}$ existieren mit der Eigenschaft, dass für jedes $f \in \mathcal{F}$ gilt:

$$\min_{j=1, \dots, N} \left(\frac{1}{n} \sum_{i=1}^n |f(x_i) - f_j(x_i)|^p \right)^{1/p} < \varepsilon.$$

In Lemma 3.4 liefern wir ein Resultat zur Abschätzung einer L_p - ε -Überdeckungszahl.

1.2 Hilfsresultate

Lemma 1.10. Sei $N \in \mathbb{N}_0$ beliebig, dann erfüllt der logistische squasher $\sigma: \mathbb{R} \rightarrow [0, 1]$, $\sigma(x) = \frac{1}{1+\exp(-x)}$ die Bedingungen aus Definition 1.7.

Beweis. Sei $N \in \mathbb{N}_0$ beliebig. Wir wissen, dass σ monoton wachsend ist, da für beliebige $s, t \in \mathbb{R}$ mit $s \leq t$ gilt:

$$\sigma(s) = \frac{1}{1 + \exp(-s)} \leq \frac{1}{1 + \exp(-t)} = \sigma(t),$$

wobei wir bei der Ungleichung die Monotonie der Exponentialfunktion verwendet haben und die obige Ungleichung aus

$$\begin{aligned} \exp(s) &\leq \exp(t) \\ \Leftrightarrow \exp(-s) &\geq \exp(-t) \\ \Leftrightarrow 1 + \exp(-s) &\geq 1 + \exp(-t) \\ \Leftrightarrow \frac{1}{1 + \exp(-s)} &\leq \frac{1}{1 + \exp(-t)} \end{aligned}$$

folgt. Zudem ist σ als Komposition glatter Funktionen, $N + 1$ mal stetig differenzierbar ist. Die Ableitungen von σ haben die Form:

$$\begin{aligned} \frac{\partial \sigma}{\partial x}(x) &= -\frac{1}{(1 + \exp(-x))^2} \cdot (-\exp(-x)) \\ &= \frac{\exp(-x)}{1 + \exp(-x)} \cdot \frac{1}{1 + \exp(-x)} \\ &= \left(1 - \frac{1}{1 + \exp(-x)}\right) \cdot \frac{1}{1 + \exp(-x)} \\ &= (1 - \sigma(x)) \cdot \sigma(x). \end{aligned}$$

Da wir bei weiterem Ableiten die Produktregel wiederholt anwenden sind alle Ableitungen von σ , Polynome in σ . Dadurch folgt Bedingung (i) aus Definition 1.7, da σ nach Voraussetzung durch 0 und 1 beschränkt ist, und die Ableitungen von σ als Produkt von beschränkten Faktoren daher auch. Da hiermit auch die erste Ableitung von σ beschränkt ist wissen wir, dass σ lipschitzstetig ist.

Nun kommen wir zum Beweis von Bedingung (ii). Polynome, die nicht das 0-Polynom sind, haben nach Satz ... (REFERENZ) auf $(0, 1)$ endlich viele Nullstellen und σ bildet nach Voraussetzung in das Intervall $[0, 1] \supseteq (0, 1)$ ab. Da die Ableitungen von σ , als Zusammensetzung von Polynomen in σ , wieder Polynome sind für die die obere Eigenschaft ebenfalls gilt, existiert ein $t_\sigma \in \mathbb{R}$ mit $\sigma(t_\sigma) \neq 0$, sodass alle Ableitungen bis zum Grad N von σ , aufgrund ihrer Struktur ungleich 0 sind. Daher ist Bedingung (ii) ebenfalls erfüllt. Wir wissen, dass für $x \in R$ und damit insbesondere für ein beliebiges $x > 0$:

$$x \leq \exp(x) + 1$$

gilt. Daraus erhalten wir mit Umformungen da $x > 0$ und $1 + \exp(-x) > 0$ ist:

$$\begin{aligned}
 x &\leq \exp(x) + 1 \\
 \Leftrightarrow x \cdot \exp(-x) &\leq 1 + \exp(-x) \\
 \Leftrightarrow \frac{\exp(-x)}{1 + \exp(-x)} &\leq \frac{1}{x} \\
 \Leftrightarrow 1 - \frac{1}{1 + \exp(-x)} &\leq \frac{1}{x} \\
 \Leftrightarrow |\sigma(x) - 1| &\leq \frac{1}{x}.
 \end{aligned}$$

Wobei die letzte Ungleichung aus der Eigenschaft des Betrags kommt, da $\frac{1}{1 + \exp(-x)} - 1 < 0$ ist, weil $1 + \exp(-x) > 1$, da $\exp(-x) > 0$. Dies zeigt die erste Relation aus Bedingung (iii). Die zweite Relation folgt durch die gleiche Art und Weise, da wir durch

$$\frac{1}{1 + \exp(x)} - \frac{1}{2} = \sigma(0 - x) - \frac{1}{2} = -\sigma(0 + x) + \frac{1}{2} = -\frac{1}{1 + \exp(-x)} + \frac{1}{2}$$

wissen, dass σ punktsymmetrisch in $(0, \frac{1}{2})$ ist. Die obige Gleichheit folgt aus

$$\begin{aligned}
 \frac{1}{1 + \exp(x)} - \frac{1}{2} &= -\frac{1}{1 + \exp(-x)} + \frac{1}{2} \\
 \Leftrightarrow \frac{1}{1 + \exp(x)} - \frac{1}{2} + \frac{1}{1 + \exp(-x)} &= \frac{1}{2} \\
 \Leftrightarrow \frac{1 + \exp(-x) + 1 + \exp(x)}{(1 + \exp(x)) \cdot (1 + \exp(-x))} - \frac{1}{2} &= \frac{1}{2} \\
 \Leftrightarrow \frac{2 + \exp(-x) + \exp(x)}{2 + \exp(-x) + \exp(x)} - \frac{1}{2} &= \frac{1}{2} \\
 \Leftrightarrow 1 - \frac{1}{2} &= \frac{1}{2}.
 \end{aligned}$$

Aus dieser Eigenschaft folgt mit

$$\sigma(-x) - 1 = \frac{1}{1 + \exp(x)} - 1 = -\frac{1}{1 + \exp(-x)} = -\sigma(x)$$

für $x < 0$ aus der ersten Relation, da $-x > 0$ ist:

$$|\sigma(x)| = |-\sigma(x)| = |\sigma(-x) - 1| \leq \frac{1}{-x} = \frac{1}{|x|}.$$

Damit haben wir alle drei Bedingungen aus Definition 1.7 gezeigt und unsere Aussage bewiesen. \square

Lemma 1.11 (Langrangesche Form des Restglieds [For16]). *Sei $f: I \rightarrow \mathbb{R}$ eine $(N+1)$ -mal stetig differenzierbare Funktion und $u, x \in I$. Dann existiert ein $\xi \in [u, x]$, so dass*

$$f(x) = \sum_{k=0}^N \frac{f^{(k)}(u)}{k!} (x-u)^k + \frac{f^{(N+1)}(\xi)}{(N+1)!} (x-u)^{N+1}.$$

Lemma 1.12. Sei $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ eine Funktion und $R, a > 0$.

a) Angenommen σ ist zwei mal stetig differenzierbar und $t_{\sigma, \text{id}} \in \mathbb{R}$ so, dass $\sigma'(t_{\sigma, \text{id}}) \neq 0$ ist. Dann gilt mit

$$f_{\text{id}}(x) = \frac{R}{\sigma'(t_{\sigma, \text{id}})} \cdot \left(\sigma\left(\frac{x}{R} + t_{\sigma, \text{id}}\right) - \sigma(t_{\sigma, \text{id}}) \right)$$

für beliebige $x \in [-a, a]$:

$$|f_{\text{id}}(x) - x| \leq \frac{\|\sigma''\|_{\infty} \cdot a^2}{2 \cdot |\sigma'(t_{\sigma, \text{id}})|} \cdot \frac{1}{R}.$$

b) Angenommen σ ist drei mal stetig differenzierbar und $t_{\sigma, \text{sq}} \in \mathbb{R}$ so, dass $\sigma''(t_{\sigma, \text{sq}}) \neq 0$ ist. Dann gilt mit

$$f_{\text{sq}}(x) = \frac{R^2}{\sigma''(t_{\sigma, \text{sq}})} \cdot \left(\sigma\left(\frac{2 \cdot x}{R} + t_{\sigma, \text{sq}}\right) - 2 \cdot \sigma\left(\frac{x}{R} + t_{\sigma, \text{sq}}\right) + \sigma(t_{\sigma, \text{sq}}) \right)$$

für beliebige $x \in [-a, a]$:

$$|f_{\text{sq}}(x) - x^2| \leq \frac{5 \cdot \|\sigma'''\|_{\infty} \cdot a^3}{3 \cdot |\sigma''(t_{\sigma, \text{sq}})|} \cdot \frac{1}{R}.$$

Beweis. a) Wir wissen, dass f_{id} 2-mal differenzierbar ist, da nach Voraussetzung σ 2-mal stetig differenzierbar ist. Damit folgt mit Lemma 1.11 für $f = f_{\text{id}}, N = 1$ und $I = [-a, a]$, dass ein $\xi \in [0, x]$ existiert, so dass:

$$f_{\text{id}}(x) = \sum_{k=0}^N \frac{f_{\text{id}}^{(k)}(0)}{k!} x^k + \frac{f_{\text{id}}^{(N+1)}(\xi)}{(N+1)!} x^{N+1}.$$

Mit $f_{\text{id}}(0) = 0$ und $f'_{\text{id}}(0) = 1$ erhalten wir:

$$\begin{aligned} & |f_{\text{id}}(x) - x| \\ &= \left| 0 + 1 \cdot x + \frac{1}{2} \cdot \frac{1}{R \cdot \sigma'(t_{\sigma, \text{id}})} \sigma''\left(\frac{\xi}{R} + t_{\sigma, \text{id}}\right) \cdot x^2 - x \right| \\ &= \left| \frac{\sigma''\left(\frac{\xi}{R} + t_{\sigma, \text{id}}\right) \cdot x^2}{2R \cdot \sigma'(t_{\sigma, \text{id}})} + x - x \right| \\ &\leq \frac{\|\sigma''\|_{\infty} \cdot a^2}{2 \cdot |\sigma'(t_{\sigma, \text{id}})|} \cdot \frac{1}{R}, \end{aligned}$$

Wobei sich die letzte Ungleichung aus den Eigenschaften der Supremumsnorm ergibt und zudem aus $x \in [-a, a] \Leftrightarrow -a \leq x \leq a$ durch Quadrieren der Ungleichung folgt, dass $x^2 \leq a^2$ ist.

- b) Die Funktion f_{sq} die hier nun 3-mal differenzierbar ist, da σ nach Voraussetzung 3-mal stetig differenzierbar ist. Es wie in a) durch Lemma 1.11 mit $f = f_{\text{sq}}, N = 2$ und $I = [-a, a]$, dass ein $\xi \in [0, x]$ existiert, so dass:

$$f_{\text{sq}}(x) = \sum_{k=0}^N \frac{f_{\text{sq}}^{(k)}(0)}{k!} x^k + \frac{f_{\text{sq}}^{(N+1)}(\xi)}{(N+1)!} x^{N+1}.$$

Mit $f_{\text{sq}}(0) = 0$, $f'_{\text{sq}}(0) = 0$ und $f''_{\text{sq}}(0) = 2$ erhalten wir:

$$\begin{aligned} |f_{\text{sq}}(x) - x^2| &= \left| x^2 + \frac{1}{6} \cdot \frac{R^2}{\sigma''(t_{\sigma, \text{sq}})} \left(\frac{8}{R^3} \sigma''' \left(\frac{2\xi}{R} + t_{\sigma, \text{sq}} \right) - \frac{2}{R^3} \sigma''' \left(\frac{\xi}{R} + t_{\sigma, \text{sq}} \right) \right) \cdot x^3 - x^2 \right| \\ &\leq \frac{a^3}{6 \cdot |\sigma''(t_{\sigma, \text{sq}})|} \cdot \frac{1}{R} \cdot \left(8 \cdot \left| \sigma''' \left(\frac{2\xi}{R} + t_{\sigma, \text{sq}} \right) \right| + 2 \left| \sigma''' \left(\frac{\xi}{R} + t_{\sigma, \text{sq}} \right) \right| \right) \\ &\leq \frac{10 \cdot a^3}{6 \cdot |\sigma''(t_{\sigma, \text{sq}})|} \cdot \frac{1}{R} \cdot \|\sigma'''\|_{\infty} \\ &= \frac{5 \cdot \|\sigma'''\|_{\infty} \cdot a^3}{3 \cdot |\sigma''(t_{\sigma, \text{sq}})|} \cdot \frac{1}{R}. \end{aligned}$$

□

Lemma 1.13. Sei $\sigma : \mathbb{R} \rightarrow [0, 1]$ 2-zulässig. Zudem sei $R > 0$ und $a > 0$ beliebig. Dann gilt für das neuronale Netz

$$\begin{aligned} f_{\text{mult}}(x, y) = \frac{R^2}{4 \cdot \sigma''(t_{\sigma})} \cdot &\left(\sigma \left(\frac{2 \cdot (x+y)}{R} + t_{\sigma} \right) - 2 \cdot \sigma \left(\frac{x+y}{R} + t_{\sigma} \right) \right. \\ &\left. - \sigma \left(\frac{2 \cdot (x-y)}{R} + t_{\sigma} \right) + 2 \cdot \sigma \left(\frac{x-y}{R} + t_{\sigma} \right) \right) \end{aligned}$$

für beliebige $x, y \in [-a, a]$ die folgende Ungleichung:

$$|f_{\text{mult}}(x, y) - x \cdot y| \leq \frac{20 \cdot \|\sigma'''\|_{\infty} \cdot a^3}{3 \cdot |\sigma''(t_{\sigma})|} \cdot \frac{1}{R}.$$

Beweis. Durch Einsetzen erhalten wir

$$f_{\text{mult}}(x, y) = \frac{1}{4} (f_{\text{sq}}(x+y) - f_{\text{sq}}(x-y))$$

und

$$x \cdot y = \frac{1}{4} ((x+y)^2 - (x-y)^2).$$

Aus diesen beiden Gleichungen folgt durch Ausklammern von $\frac{1}{4}$, der Homogenität des

Betrags und der Anwendung der Dreiecksungleichung:

$$\begin{aligned}
|f_{\text{mult}}(x, y) - x \cdot y| &= \frac{1}{4} \cdot |f_{\text{sq}}(x+y) - f_{\text{sq}}(x-y) - (x+y)^2 + (x-y)^2| \\
&\leq \frac{1}{4} \cdot |f_{\text{sq}}(x+y) - (x+y)^2| + \frac{1}{4} \cdot |(x-y)^2 - f_{\text{sq}}(x-y)| \\
&\leq 2 \cdot \frac{1}{4} \cdot \frac{40 \cdot \|\sigma'''\|_{\infty} \cdot a^3}{3 \cdot |\sigma''(t_{\sigma})|} \cdot \frac{1}{R} \\
&= \frac{20 \cdot \|\sigma'''\|_{\infty} \cdot a^3}{3 \cdot |\sigma''(t_{\sigma})|} \cdot \frac{1}{R},
\end{aligned}$$

wobei wir bei der letzten Ungleichung verwendet haben, dass $a > 0$ nach Lemma 1.12b) beliebig gewählt wurde und daher insbesondere für beliebiges $x \in [-2a, 2a]$

$$|f_{\text{sq}}(x) - x^2| \leq \frac{40 \cdot \|\sigma'''\|_{\infty} \cdot a^3}{3 \cdot |\sigma''(t_{\sigma})|} \cdot \frac{1}{R}$$

gilt. □

Lemma 1.14. Sei $\sigma: \mathbb{R} \rightarrow [0, 1]$ 2-zulässig. Sei f_{mult} das neuronale Netz aus Lemma 1.13 und f_{id} das neuronale Netz aus Lemma 1.12. Angenommen es gelten die Ungleichungen

$$a \geq 1 \quad \text{und} \quad R \geq \frac{\|\sigma''\|_{\infty} \cdot a}{2 \cdot |\sigma'(t_{\sigma})|}. \quad (1.3)$$

Dann erfüllt das neuronale Netz

$$f_{\text{ReLU}}(x) = f_{\text{mult}}(f_{\text{id}}(x), \sigma(R \cdot x)) \quad (1.4)$$

für alle $x \in [-a, a]$ folgende Ungleichung:

$$|f_{\text{ReLU}}(x) - \max\{x, 0\}| \leq 56 \cdot \frac{\max\{|\sigma''|_{\infty}, \|\sigma'''\|_{\infty}, 1\}}{\min\{2 \cdot |\sigma'(t_{\sigma})|, |\sigma''(t_{\sigma})|, 1\}} \cdot a^3 \cdot \frac{1}{R}.$$

Beweis. Da σ nach Voraussetzung 2-zulässig nach Definition 1 ist, gilt für $R \geq 0$, und $x \in \mathbb{R} \setminus \{0\}$:

$$|\sigma(R \cdot x) - 1| \leq \frac{1}{R \cdot x} \quad \text{für } x > 0$$

und

$$|\sigma(R \cdot x)| \leq \frac{1}{|R \cdot x|} \quad \text{für } x < 0.$$

Damit folgt aus der Homogenität des Betrags für alle $x \neq 0$:

$$|\sigma(R \cdot x) - \mathbb{1}_{[0, \infty)}(x)| \leq \frac{1}{|R \cdot x|} = \frac{1}{R \cdot |x|}. \quad (1.5)$$

Nach Lemma 1.12 und Lemma 1.13 gilt:

$$|f_{\text{id}}(x) - x| \leq \frac{\|\sigma''\|_{\infty} \cdot a^2}{2 \cdot |\sigma'(t_{\sigma})|} \cdot \frac{1}{R} \quad \text{für } x \in [-a, a] \quad (1.6)$$

und

$$|f_{\text{mult}}(x, y) - x \cdot y| \leq \frac{160 \cdot \|\sigma'''\|_{\infty} \cdot a^3}{3 \cdot |\sigma''(t_{\sigma})|} \cdot \frac{1}{R} \quad \text{für } x, y \in [-2a, 2a]. \quad (1.7)$$

Da nach Voraussetzung $a \geq 1$ ist, gilt insbesondere $[0, 1] \subseteq [-2a, 2a]$ und daher gilt insbesondere $\sigma(Rx) \in [0, 1] \subseteq [-2a, 2a]$. Zudem erhalten wir durch eine Nulladdition, das Anwenden der Dreiecksungleichung, die Verwendung von Lemma 1.12 und (1.3) für R :

$$\begin{aligned} |f_{\text{id}}(x)| &= |f_{\text{id}}(x) - x + x| \\ &\leq |f_{\text{id}}(x) - x| + |x| \\ &\leq \frac{\|\sigma''\|_{\infty} \cdot a^2}{2 \cdot |\sigma'(t_{\sigma})|} \cdot \frac{1}{R} + |x| \\ &\leq \frac{\|\sigma''\|_{\infty} \cdot a^2}{2 \cdot |\sigma'(t_{\sigma})|} \cdot \frac{2 \cdot |\sigma'(t_{\sigma})|}{\|\sigma''\|_{\infty} \cdot a} + |x| \\ &= a + |x| \\ &\leq 2 \cdot a \end{aligned}$$

wobei $x \in [-a, a]$. Daraus folgt insbesondere $f_{\text{id}}(x) \in [-2a, 2a]$. Mithilfe von $\max\{x, 0\} = x \cdot \mathbb{1}_{[0, \infty)}(x)$, Gleichung (1.4), zweier Nulladditionen und dem zweifachen Anwenden der Dreiecksungleichung erhalten wir:

$$\begin{aligned} |f_{\text{ReLU}}(x) - \max\{x, 0\}| &= |f_{\text{mult}}(f_{\text{id}}(x), \sigma(R \cdot x)) - x \cdot \mathbb{1}_{[0, \infty)}(x)| \\ &\leq |f_{\text{mult}}(f_{\text{id}}(x), \sigma(R \cdot x)) - f_{\text{id}}(x) \cdot \sigma(R \cdot x)| \\ &\quad + |f_{\text{id}}(x) \cdot \sigma(R \cdot x) - x \cdot \sigma(R \cdot x)| + |x \cdot \sigma(R \cdot x) - x \cdot \mathbb{1}_{[0, \infty)}(x)|. \end{aligned}$$

Daraus ergibt sich mithilfe von (1.5) - (1.7), $\sigma(Rx) \in [0, 1]$ und $a^3 \geq 1$

$$\begin{aligned} &\leq \frac{160 \cdot \|\sigma'''\|_{\infty} \cdot a^3}{3 \cdot |\sigma''(t_{\sigma})|} \cdot \frac{1}{R} + \frac{\|\sigma''\|_{\infty} \cdot a^3}{2 \cdot |\sigma'(t_{\sigma})|} \cdot \frac{1}{R} \cdot 1 + \frac{1}{R} \\ &\leq \left(\frac{160}{3} \cdot \frac{\|\sigma'''\|_{\infty} \cdot a^3}{|\sigma''(t_{\sigma})|} + \frac{\|\sigma''\|_{\infty} \cdot a^3}{2 \cdot |\sigma'(t_{\sigma})|} + \frac{a^3}{a^3} \right) \cdot \frac{1}{R} \\ &\leq \left(\frac{160 \cdot \|\sigma'''\|_{\infty} \cdot a^3 + 3 \cdot \|\sigma''\|_{\infty} \cdot a^3 + 3 \cdot a^3}{3 \cdot \min\{2 \cdot |\sigma'(t_{\sigma})|, |\sigma''(t_{\sigma})|, 1\}} \right) \cdot \frac{1}{R} \\ &\leq \frac{166}{3} \cdot \left(\frac{\max\{\|\sigma'''\|_{\infty}, \|\sigma''\|_{\infty}, 1\}}{\min\{2 \cdot |\sigma'(t_{\sigma})|, |\sigma''(t_{\sigma})|, 1\}} \right) \cdot a^3 \cdot \frac{1}{R} \\ &\leq 56 \cdot \frac{\max\{\|\sigma'''\|_{\infty}, \|\sigma''\|_{\infty}, 1\}}{\min\{2 \cdot |\sigma'(t_{\sigma})|, |\sigma''(t_{\sigma})|, 1\}} \cdot a^3 \cdot \frac{1}{R}. \end{aligned}$$

□

Lemma 1.15. Sei $M \in \mathbb{N}$ und sei $\sigma: \mathbb{R} \rightarrow [0, 1]$ 2-zulässig. Sei $a > 0$ und

$$R \geq \frac{\|\sigma''\|_{\infty} \cdot (M + 1)}{2 \cdot |\sigma'(t_{\sigma})|},$$

sei $y \in [-a, a]$ und f_{ReLU} das neuronale Netz aus Lemma 1.14. Dann erfüllt das neuronale Netz

$$f_{\text{hat},y}(x) = f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x-y) + 1\right) - 2 \cdot f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x-y)\right) + f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x-y) - 1\right)$$

für alle $x \in [-a, a]$ mit $z_+ = \max\{0, z\}$ ($z \in \mathbb{R}$):

$$\left| f_{\text{hat},y}(x) - \left(1 - \frac{M}{2a} \cdot |x-y|\right)_+ \right| \leq 1792 \cdot \frac{\max\{\|\sigma''\|_\infty, \|\sigma'''\|_\infty, 1\}}{\min\{2 \cdot |\sigma'(t_{\sigma, \text{id}})|, |\sigma''(t_\sigma)|, 1\}} \cdot M^3 \cdot \frac{1}{R}.$$

Beweis. Für $x \in \mathbb{R}$ gilt die Gleichung:

$$\left(1 - \frac{M}{2a} \cdot |x|\right)_+ = \max\left\{\frac{M}{2a} \cdot x + 1, 0\right\} - 2 \cdot \max\left\{\frac{M}{2a} \cdot x, 0\right\} + \max\left\{\frac{M}{2a} \cdot x - 1, 0\right\}, \quad (1.8)$$

die wir im zweiten Teil dieses Beweises zeigen werden. Damit beweisen wir das Resultat mit Hilfe von Lemma 1.14, denn mit der Definition von $f_{\text{hat},y}(x)$ und zwei mal der Dreiecksungleichung folgt:

$$\begin{aligned} \left| f_{\text{hat},y}(x) - \left(1 - \frac{M}{2a} \cdot |x-y|\right)_+ \right| &\leq \left| f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x-y) + 1\right) - \max\left\{\frac{M}{2a} \cdot (x-y) + 1, 0\right\} \right| \\ &\quad + 2 \cdot \left| f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x-y)\right) - \max\left\{\frac{M}{2a} \cdot (x-y), 0\right\} \right| \\ &\quad + \left| f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x-y) - 1\right) - \max\left\{\frac{M}{2a} \cdot (x-y) - 1, 0\right\} \right| \\ &\leq 1792 \cdot \frac{\max\{\|\sigma''\|_\infty, \|\sigma'''\|_\infty, 1\}}{\min\{2 \cdot |\sigma'(t_\sigma)|, |\sigma''(t_\sigma)|, 1\}} \cdot M^3 \cdot \frac{1}{R}, \end{aligned}$$

wobei die letzte Ungleichung daraus folgt, dass wir auf jeden Summanden mit $1 \leq a = M+1$ Lemma 1.14 angewendet haben und die Abschätzung

$$(M+1)^3 \leq (2M)^3 = 8M^3$$

verwendet haben, da $M \geq 1$ ist. (□)

Um Gleichung (1.8) zu zeigen unterscheiden wir vier Fälle.

Fall 1 ($x < 0$) In diesem Fall hat die linke Seite nach der Definition des Betrags die Gestalt

$$\max\left\{1 + \frac{M}{2a} \cdot x, 0\right\}$$

und die rechte Seite die Form

$$\max\left\{\frac{M}{2a} \cdot x + 1, 0\right\} - 2 \cdot 0 + 0,$$

da $x < 0$ und damit die letzten zwei Summanden 0 sind. Es erfordert hier eine weitere Fallunterscheidung.

Fall 1.1 ($0 > x \geq -\frac{2a}{M}$) In diesem Fall gilt für die linke und rechte Seite:

$$\max\{1 + \frac{M}{2a} \cdot x, 0\} = 1 + \frac{M}{2a} \cdot x.$$

Fall 1.2 ($x < -\frac{2a}{M}$) In diesem Fall sind beide Seiten gleich 0, da $1 + \frac{M}{2a} \cdot x \leq 0$ ist. (\square)

Fall 2 ($x \geq 0$) In diesem Fall hat die linke Seite nach der Definition des Betrags die Gestalt

$$\max\{1 - \frac{M}{2a} \cdot x, 0\}$$

und die rechte Seite die Form

$$\max\{\frac{M}{2a} \cdot x + 1, 0\} - 2 \cdot \max\{\frac{M}{2a} \cdot x, 0\} + \max\{\frac{M}{2a} \cdot x - 1, 0\}$$

und erfordert daher eine weitere Fallunterscheidung.

Fall 2.1 ($0 \leq x < \frac{2a}{M}$) In diesem Fall hat die linke Seite die Gestalt

$$1 - \frac{M}{2a} \cdot x$$

und die rechte Seite die Form

$$\frac{M}{2a} \cdot x + 1 - 2 \cdot \frac{M}{2a} \cdot x + 0 = 1 - \frac{M}{2a} \cdot x.$$

und stimmt daher mit der linken Seite überein.

Fall 2.2 ($x \geq \frac{2a}{M}$) In diesem ist die linke Seite gleich 0, da $1 - \frac{M}{2a} \cdot x < 0$ ist und die rechte Seite besitzt die Form

$$\frac{M}{2a} \cdot x + 1 - 2 \cdot \frac{M}{2a} \cdot x + \frac{M}{2a} \cdot x - 1 = 0.$$

(\square)

Durch diese Fallunterscheidung wurde die Gleichung (1.8) bewiesen und damit ist der Beweis vollständig. \square

Das nächste Lemma ist ein Kombinatorikargument welches wir in Kapitel 2 benötigen werden.

Lemma 1.16. Sei $d, N \in \mathbb{N}$ und $k \in \mathbb{N}_0$, dann gilt:

$$\left| \left\{ \mathbf{j} \in [N]^d : |\mathbf{j}|_1 = k \right\} \right| = \binom{d+k-1}{k}.$$

Beweis. Diese Aussage folgt aus einer Analogie zu einem Urnenexperiment. Wir betrachten eine Urne mit d -vielen Kugeln die wir mit j_1, \dots, j_d beschriften. Wir ziehen k -Mal aus dieser Urne mit Zurücklegen und ohne Beachtung der Reihenfolge und konstruieren einen Vektor (j_1, \dots, j_d) welcher mit jeder Komponente angibt, wie oft welche Kugel gezogen wurde. \square

Kapitel 2

Konstruktion des Neuronale-Netze-Schätzers

In diesem Kapitel werden wir mithilfe der uns gegebenen Datenmenge

$$\mathcal{D}_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}, \quad (2.1)$$

unseren Regressionsschätzer \tilde{m}_n konstruieren. In Kapitel 1 haben wir bereits in Definition 1.6 vorgestellt, was wir unter einem mehrschichtigen feedforward neuronalen Netz verstehen.

Für die Konstruktion unseres Neuronale-Netze-Schätzers wählen wir den logistischen Squasher (1.2) als Aktivierungsfunktion, verwenden die gegebene Datenmenge \mathcal{D}_n und wählen die Gewichte des neuronalen Netzes so, dass die resultierende Funktion aus Gleichung (1.1) eine gute Schätzung für die Regressionsfunktion m ist. Dafür wählen wir die Gewichte bis auf die in der Ausgabeschicht fest und schätzen die Gewichte in der Ausgabeschicht, indem wir mit unserer Datenmenge (2.1) ein regularisiertes Kleinst-Quadrat-Problem (REFERENZ) lösen.

2.1 Definition der Netzwerkarchitektur

Sei $a > 0$ fest und $M \in \mathbb{N}$. Zunächst fixieren wir die Multiindexnotation, die wir aufgrund der Übersichtlichkeit im weiteren Verlauf dieser Arbeit verwenden werden.

Sei $[M]^d := \{0, 1, \dots, M\}^d$. Für $(\mathbf{i}^{(1)}, \dots, \mathbf{i}^{(d)}) = \mathbf{i} \in [M]^d$ und $x \in \mathbb{R}^d$ definieren wir

$$|\mathbf{i}|_1 := \sum_{k=1}^d \mathbf{i}^{(k)} \quad \mathbf{i}! := \mathbf{i}^{(1)}! \dots \mathbf{i}^{(d)}! \quad x^{\mathbf{i}} := x_1^{\mathbf{i}^{(1)}} \dots x_d^{\mathbf{i}^{(d)}}.$$

Für $f: \mathbb{R}^d \rightarrow \mathbb{R}$ ausreichend oft differenzierbar definieren wir

$$\partial^{\mathbf{i}} f(x) := \frac{\partial^{|\mathbf{i}|} f}{\partial^{\mathbf{i}^{(1)}} x_1 \cdots \partial^{\mathbf{i}^{(d)}} x_d}(x).$$

Wir betrachten im Folgenden ein d -dimensionales äquidistantes Gitter im Würfel $[-a, a]^d$ mit Schrittweite $\frac{2a}{M}$. Dann ordnen wir jedem Multiindex $\mathbf{i} \in [M]^d$ einen Gitterpunkt $x_{\mathbf{i}}$ mit:

$$x_{\mathbf{i}} = \left(-a + \mathbf{i}^{(1)} \cdot \frac{2a}{M}, \dots, -a + \mathbf{i}^{(d)} \cdot \frac{2a}{M} \right) = -\mathbf{a} + \frac{2a}{M} \cdot \mathbf{i}, \quad (2.2)$$

zu, mit $\mathbf{a} = (a, a, \dots, a) \in \mathbb{R}^d$.

Hiermit lässt sich das zu m gehörige Taylorpolynom der Ordnung q mit Entwicklungspunkt $x_{\mathbf{i}}$ schreiben als

$$p_{\mathbf{i}}^m(x) = \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \partial^{\mathbf{j}} m(x_{\mathbf{i}}) \cdot \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!}.$$

Zudem betrachten wir eine Funktion

$$P_m(x) = \sum_{\mathbf{i} \in [M]^d} p_{\mathbf{i}}^m(x) \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+, \quad (2.3)$$

für die wir im weiteren Verlauf dieser Arbeit zeigen werden, dass diese die Regressionsfunktion m approximiert.

Wir zeigen mit folgendem Lemma dass $P_m(x)$ eine lokale Spline Interpolation von Taylorpolynomen von m ist.

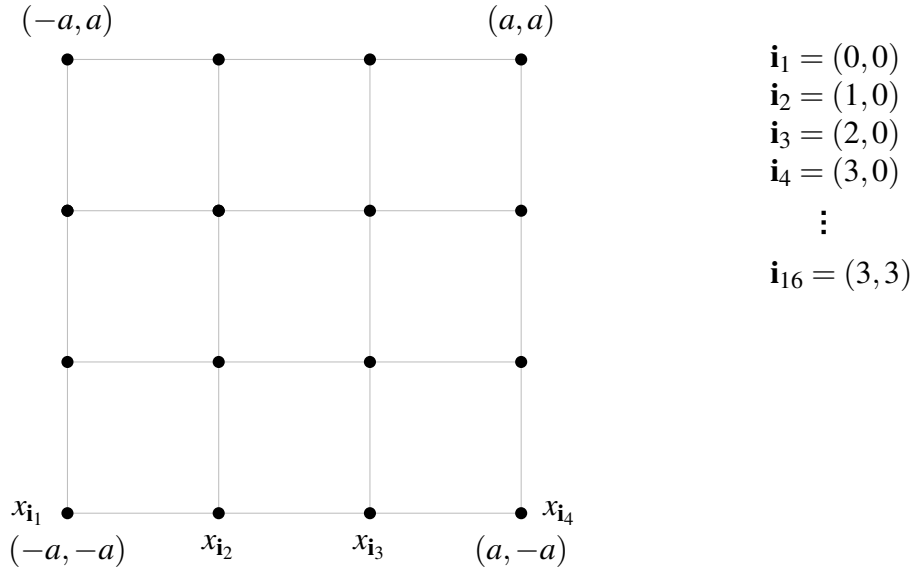
Lemma 2.1. *Sei $a > 0$ und $M \in \mathbb{N}$. Dann sind für $\mathbf{i} \in [M]^d$ und $x \in \mathbb{R}^d$ die Funktionen*

$$B_{\mathbf{i}}(x) = \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+,$$

B-Splines im \mathbb{R}^d , wobei wir unter $x_{\mathbf{i}}$ die Gitterpunkte aus Gleichung (2.2) verstehen. Dafür sind folgende drei Bedingungen für $x \in \mathbb{R}^d$ zu überprüfen:

- i) Zerlegung der Eins: $\sum_{\mathbf{i} \in [M]^d} B_{\mathbf{i}}(x) = 1$.*
- ii) Nicht-Negativität: $B_{\mathbf{i}}(x) \geq 0$ für alle $\mathbf{i} \in [M]^d$.*
- iii) Lokaler Träger: Für $\mathbf{i} \in [M]^d$ ist $B_{\mathbf{i}}(x) > 0$ falls $|x^{(j)} - x_{\mathbf{i}}^{(j)}| < \frac{2a}{M}$ für alle $j \in \{1, \dots, d\}$ gilt und andernfalls $B_{\mathbf{i}}(x) = 0$.*

Beweis. Als erstes möchten wir für $d = 2$ und $M = 3$ eine Skizze angeben um die Idee des Beweises zu veranschaulichen. Es ist ein Gitter mit $(M + 1)^d$ Gitterpunkten die den $x_{\mathbf{i}_k}$

Abbildung 2.1: Beispielhafte Darstellung der x_{i_k} für $d = 2$ und $M = 3$.

entsprechen. Der Abstand zwischen zwei Gitterpunkten beträgt $\frac{2a}{M}$. Man betrachtet immer den Abstand zu den nächsten 2^d Gitterpunkten, da $(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|)_+ = 0$ immer dann gilt, wenn der Abstand zwischen $x^{(j)}$ und $x_{\mathbf{i}}^{(j)}$ größer als $\frac{2a}{M}$ ist.

i) Im Folgenden wollen wir

$$\sum_{\mathbf{i} \in [M]^d} B_{\mathbf{i}}(x) = 1 \quad (x \in \mathbb{R}^d) \quad (2.4)$$

per Induktion über d zeigen.

Induktionsanfang (IA): Für $d = 1$ kann x nur zwischen zwei Gitterpunkten $x_{i_1} \neq x_{i_2}$ liegen. Sei ohne Beschränkung der Allgemeinheit $x_{i_1} \leq x \leq x_{i_2}$, dann gilt mit der gleichen Begründung wie im einleitenden Beispiel:

$$\begin{aligned} \sum_{\mathbf{i} \in [M]^d} (1 - \frac{M}{2a} \cdot |x - x_{\mathbf{i}}|)_+ &= (1 - \frac{M}{2a} \cdot |x - x_{i_1}|)_+ + (1 - \frac{M}{2a} \cdot |x - x_{i_2}|)_+ \\ &= 1 + 1 - \frac{M}{2a} \cdot (x - x_{i_1} + x_{i_2} - x) \\ &= 1 + 1 - \frac{M}{2a} \cdot \frac{2a}{M} \\ &= 1, \end{aligned}$$

wobei wir unter anderem verwendet haben, dass beide Summanden unabhängig von dem Positivteil nichtnegativ sind, da der Abstand von x zu den beiden Gitterpunkten x_{i_1} und x_{i_2} kleiner gleich $\frac{2a}{M}$ ist. Zudem haben wir verwendet, dass $x_{i_2} - x_{i_1} = \frac{2a}{M}$ gilt, da beides Gitterpunkte sind.

Induktionshypothese (IH): Aussage (2.4) gelte für ein beliebiges aber festes $d \in \mathbb{N}$.

Induktionsschritt (IS): Wir nehmen ohne Beschränkung der Allgemeinheit an, dass $x_{(0,\dots,0)} \leq x \leq x_{(1,\dots,1)}$ gilt bzgl. $(d+1)$ -stelliger Relation. Das heißt also, dass $x \in [-a, -a + \frac{2a}{M}]^{d+1}$ gilt. Im Folgenden zeigen wir

$$\sum_{\mathbf{i} \in [M]^{(d+1)}} B_{\mathbf{i}}(x) = \sum_{\mathbf{i} \in [M]^{(d+1)}} \prod_{j=1}^{d+1} \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+ = 1.$$

Ein Summand ist Null, wenn ein $j \in \{1, \dots, d+1\}$ existiert mit $|x^{(j)} - x_{\mathbf{i}}^{(j)}| \geq \frac{2a}{M}$. Zudem haben wir ohne Beschränkung der Allgemeinheit angenommen dass $x \in [-a, -a + \frac{2a}{M}]^{d+1}$ gilt, damit haben wir also nur noch 2^{d+1} Summanden, was der Anzahl der Gitterpunkte die am nächsten bei x liegen entspricht. Zudem wissen wir, dass alle Gitterpunkte, die in der $(d+1)$ -ten Komponente den selben Wert haben, in dieser Dimension gleich weit von $x_{(0,\dots,0)}^{(d+1)}$ entfernt sind. Das heißt, in jedem Summanden kommt der Faktor $(1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(0,\dots,0)}^{(d+1)}|)$ bzw. $(1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(1,\dots,1)}^{(d+1)}|)$ vor, da

$$\left(1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{\mathbf{i}}^{(d+1)}| \right) = \begin{cases} (1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(0,\dots,0)}^{(d+1)}|) & \mathbf{i} \in \{0, 1\}^d \times \{0\} \\ (1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(1,\dots,1)}^{(d+1)}|) & \mathbf{i} \in \{0, 1\}^d \times \{1\} \end{cases}$$

gilt. Daraus ergibt sich:

$$\begin{aligned} & \sum_{\mathbf{i} \in [M]^{(d+1)}} \prod_{j=1}^{d+1} \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+ \\ &= \sum_{\mathbf{i} \in \{0,1\}^{d+1}} \prod_{j=1}^{d+1} \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right) \\ &= \left(\sum_{\mathbf{i} \in \{0,1\}^d \times \{0\}} \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right) \right) \cdot \left(1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(0,\dots,0)}^{(d+1)}| \right) \\ &\quad + \left(\sum_{\mathbf{i} \in \{0,1\}^d \times \{1\}} \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right) \right) \cdot \left(1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(1,\dots,1)}^{(d+1)}| \right) \\ &\stackrel{\text{(IV)}}{=} 1 \cdot \left(1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(0,\dots,0)}^{(d+1)}| \right) + 1 \cdot \left(1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(1,\dots,1)}^{(d+1)}| \right) \\ &= 1 + 1 - \frac{M}{2a} \cdot |x^{(d+1)} - x_{(0,\dots,0)}^{(d+1)}| + x_{(1,\dots,1)}^{(d+1)} - x_{(0,\dots,0)}^{(d+1)}| \\ &= 1 + 1 - 1 \\ &= 1, \end{aligned}$$

wobei wir bei der vorletzten Gleichung angewendet haben, dass $x_{(1,\dots,1)}^{(d+1)} - x_{(0,\dots,0)}^{(d+1)} = \frac{2a}{M}$ ist, da beides Gitterpunkte sind. \square

ii) Es folgt $\prod_{j=1}^d (1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|)_+ \geq 0$ für alle $\mathbf{i} \in [M]^d$, da

$$z_+ = \max\{z, 0\} \geq 0 \quad (z \in \mathbb{R})$$

gilt. Damit wäre die Nichtnegativität der Koeffizienten der Linearkombination gezeigt.

Damit ist jeder Summand in

$$\sum_{\mathbf{i} \in [M]^d} \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|\right)_+$$

größer gleich Null und wegen (2.4) muss dann auch

$$\prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|\right)_+ \leq 1 \quad (2.5)$$

gelten.

iii) Es handelt sich hierbei um einen lokale Träger, da nach der Konstruktion von $B_{\mathbf{i}}(x)$ der Funktionswert genau dann Null ist, wenn ein $j \in \{1, \dots, d\}$ existiert, sodass $|x^{(j)} - x_{\mathbf{i}}^{(j)}| \geq \frac{2a}{M}$ gilt. Andernfalls erhalten wir mit Bedingung ii), dass $B_{\mathbf{i}}(x) > 0$ ist. \square

Hiermit erhalten wir, dass:

$$\begin{aligned} P_m(x) &= \sum_{\mathbf{i} \in [M]^d} p_{\mathbf{i}}^m(x) \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|\right)_+ \\ &= \sum_{\mathbf{i} \in [M]^d} p_{\mathbf{i}}(x) \cdot B_{\mathbf{i}}(x) \end{aligned}$$

gilt und damit $P_m(x)$ eine Spline Interpolation von Taylorpolynomen von m ist. Die Wahl der Netzwerkarchitektur und der Werte aller Gewichte bis auf die aus der Ausgabeschicht ist durch folgendes Approximationsresultat durch eine lokale Spline Interpolation von Taylorpolynomen für (p, C) -glatte Funktionen auf dem Intervall $[-a, a]^d$ motiviert.

Lemma 2.2. Sei $M \in \mathbb{N}$, $c > 0$, $a > 0$ und f eine (p, C) -glatte Funktion, wobei $p = q + s$ mit $q \in \mathbb{N}_0, s \in (0, 1]$ und $C > 0$ sind. Sei zudem $P_f(x)$ analog wie in (2.3) eine lokale Spline Interpolation von Taylorpolynomen von f . Dann gilt:

$$\sup_{x \in [-a, a]^d} |f(x) - P_f(x)| \leq c \cdot \left(\frac{a}{M}\right)^p.$$

Beweis. Nach Lemma 1.11 über die Lagrange Form des Restglieds existiert ein $\xi \in [0, 1]$, so, dass

$$\begin{aligned} f(x) &= T_{x_{\mathbf{i}}, q-1}[f(x)] \\ &= \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q-1}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \cdot \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} + \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 = q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})) \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!}. \end{aligned} \quad (2.6)$$

Nach der Basispline Eigenschaft aus Gleichung (2.4) erhalten wir

$$f(x) = \sum_{\mathbf{i} \in [M]^d} f(x_{\mathbf{i}}) \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+.$$

Zudem wissen wir, dass man immer den Abstand zu den nächsten 2^d Gitterpunkten betrachtet, da $(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|)_+ = 0$ immer dann gilt, wenn der Abstand zwischen $x^{(j)}$ und $x_{\mathbf{i}}^{(j)}$ größer als $\frac{2a}{M}$ ist. Daher ergibt sich:

$$\sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 = q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})) \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \leq \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 = q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})) \frac{1}{\mathbf{j}!} \cdot \left(\frac{2a}{M} \right)^q. \quad (2.7)$$

Mithilfe der Dreiecksungleichung und der Definition von $P_f(x)$ erhalten wir:

$$\begin{aligned} & |f(x) - P_f(x)| \\ & \leq \sum_{\mathbf{i} \in [M]^d} \left| f(x) - \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \cdot \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \right| \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+. \end{aligned} \quad (2.8)$$

Nach Gleichung (2.6) erhalten wir:

$$\begin{aligned} & \left| f(x) - \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \cdot \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \right| \\ & = \left| \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q-1}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \cdot \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} + \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 = q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})) \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \right. \\ & \quad \left. - \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \cdot \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \right| \\ & = \left| \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 = q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})) \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} - \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 = q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \right|. \end{aligned} \quad (2.9)$$

Aus der (p, C) -Glattheit von f nach Definition 1.1 mit $x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}}), x_{\mathbf{i}} \in \mathbb{R}^d$ folgt durch (2.7) und (2.9):

$$\begin{aligned} & \left| \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 = q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})) \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} - \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 = q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \right| \\ & \leq \left(\frac{2a}{M} \right)^q \|x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}}) - x_{\mathbf{i}}\|^s \cdot C. \end{aligned} \quad (2.10)$$

Fassen wir die Gleichungen (2.8), (2.9) und (2.10) zusammen, erhalten wir:

$$\begin{aligned}
|f(x) - P_f(x)| &\leq \sum_{\mathbf{i} \in [M]^d} \left| f(x) - \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \cdot \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \right| \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+ \\
&\leq \left(\frac{2a}{M} \right)^q \|x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}}) - x_{\mathbf{i}}\|^s \cdot C \cdot \sum_{\mathbf{i} \in [M]^d} \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+ \\
&\leq \left(\frac{2a}{M} \right)^q \cdot \left(\frac{2a}{M} \right)^s \cdot d^{s/2} \cdot C \\
&= c \cdot \left(\frac{a}{M} \right)^p,
\end{aligned}$$

wobei wir bei der letzten Gleichheit Bedingung (2.4) und $q + s = p$ verwendet und

$$c = 2^p \cdot d^{s/2} \cdot C,$$

gewählt haben. □

$P_m(x)$ lässt sich in die Form

$$\sum_{\mathbf{i} \in [M]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} a_{\mathbf{i}, \mathbf{j}} \cdot (x - x_{\mathbf{i}})^{\mathbf{j}} \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+$$

durch geeignet gewählte $a_{\mathbf{i}, \mathbf{j}} \in \mathbb{R}$ bringen, da sich jedes $p_{\mathbf{i}}^m(x)$ als Polynom umordnen lässt und wir daher auch $P_m(x)$ umschreiben können.

Als nächstes wollen wir geeignete neuronale Netze $f_{\text{net}, \mathbf{j}, \mathbf{i}}$ definieren, die die Funktionen

$$x \mapsto (x - x_{\mathbf{i}})^{\mathbf{j}} \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+$$

approximieren. Zudem möchten wir die Netzwerkarchitektur so wählen, dass neuronale Netze der Form

$$\sum_{\mathbf{i} \in [M]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} a_{\mathbf{i}, \mathbf{j}} \cdot f_{\text{net}, \mathbf{j}, \mathbf{i}}(x) \quad (a_{\mathbf{i}, \mathbf{j}} \in \mathbb{R})$$

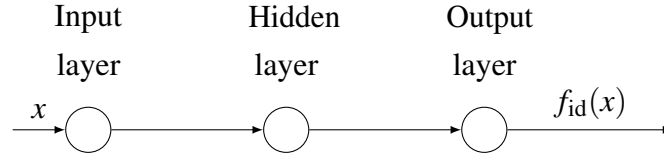
in ihr enthalten sind. Um dies zu erreichen, sei

$$\sigma(x) = \frac{1}{(1 + \exp(-x))} \quad (x \in \mathbb{R})$$

der logistische Squasher (1.2), wählen $R \geq 1$ und definieren die folgenden neuronale Netze:

Das neuronale Netz

$$f_{\text{id}}(x) = 4R \cdot \sigma\left(\frac{x}{R}\right) - 2R, \quad (2.11)$$

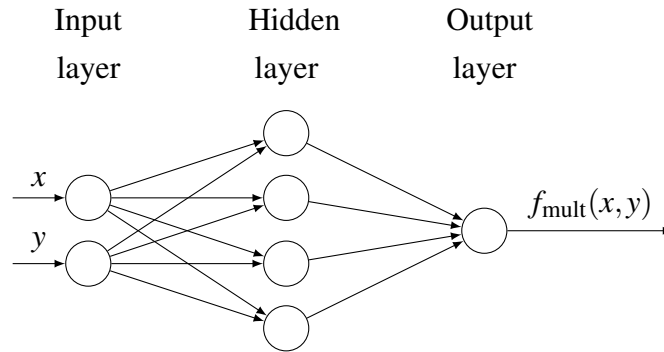
Abbildung 2.2: Neuronales Netz $f_{\text{id}}(x) = 4R \cdot \sigma(\frac{x}{R}) - 2R$

welches, wie in Lemma 1.12 gezeigt, die Funktion $f(x) = x$ approximiert und in Abbildung 2.2 veranschaulicht wird.

Das neuronale Netz

$$f_{\text{mult}}(x, y) = \frac{R^2}{4} \cdot \frac{(1 + \exp(-1))^3}{\exp(-2) - \exp(-1)} \cdot \left(\sigma\left(\frac{2(x+y)}{R} + 1\right) - 2 \cdot \sigma\left(\frac{x+y}{R} + 1\right) - \sigma\left(\frac{2(x-y)}{R} + 1\right) + 2 \cdot \sigma\left(\frac{x-y}{R} + 1\right) \right), \quad (2.12)$$

welches, wie in Lemma 1.13 gezeigt, die Funktion $f(x, y) = x \cdot y$ approximiert und in Abbildung 2.3 veranschaulicht wird.

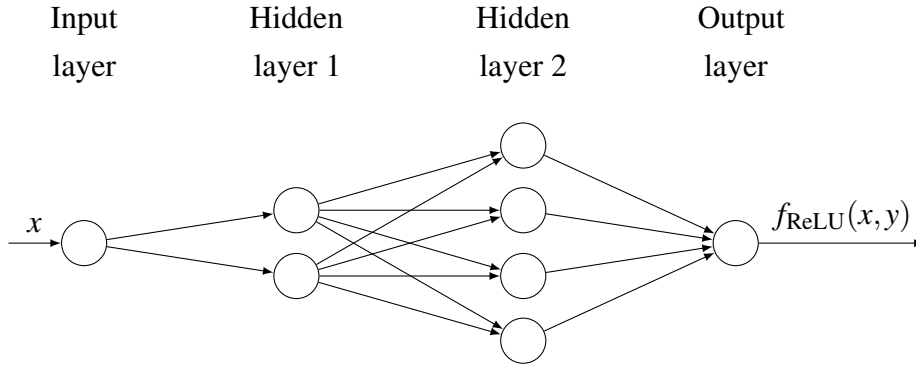
Abbildung 2.3: Neuronales Netz $f_{\text{mult}}(x) = \dots$

Das neuronale Netz

$$f_{\text{ReLU}}(x) = f_{\text{mult}}(f_{\text{id}}(x), \sigma(R \cdot x)), \quad (2.13)$$

welches, wie in Lemma 1.14 gezeigt, die Funktion $f(x) = x_+$ approximiert und in Abbildung 2.4 veranschaulicht wird und schließlich das neuronale Netz

$$f_{\text{hat}, y}(x) = f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x - y) + 1\right) - 2 \cdot f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x - y)\right) + f_{\text{ReLU}}\left(\frac{M}{2a} \cdot (x - y) - 1\right), \quad (2.14)$$

Abbildung 2.4: Neuronales Netz $f_{\text{ReLU}}(x) = \dots$

welches, wie in Lemma 1.15 gezeigt, für fixes $y \in \mathbb{R}$ die Funktion

$$f(x) = \left(1 - \left(\frac{M}{2a} \right) \cdot |x - y| \right)_+$$

approximiert und in Abbildung 2.5 veranschaulicht wird.

Mit diesen neuronalen Netzen können wir nun $f_{\text{net}, \mathbf{j}, \mathbf{i}}$ rekursiv definieren. Dafür wählen wir $N > q$, setzen $s = \lceil \log_2(N + d) \rceil$ und definieren für $\mathbf{j} \in [N]^d$, $\mathbf{i} \in [M]^d$ und $k \in \{1, \dots, (M+1)^d\}$:

$$f_{\text{net}, \mathbf{j}, \mathbf{i}}(x) = f_1^{(0)}(x).$$

wobei

$$f_k^{(l)}(x) = f_{\text{mult}}\left(f_{2k-1}^{(l+1)}(x), f_{2k}^{(l+1)}(x)\right)$$

für $k \in \{1, 2, \dots, 2^l\}$ und $l \in \{0, \dots, s-1\}$, und

$$f_k^{(s)}(x) = f_{\text{id}}(f_{\text{id}}(x^{(l)} - x_{\mathbf{i}_k}^{(l)}))$$

für $j_1 + j_2 + \dots + j_{l-1} + 1 \leq k \leq j_1 + j_2 + \dots + j_l$ und $1 \leq l \leq d$ und

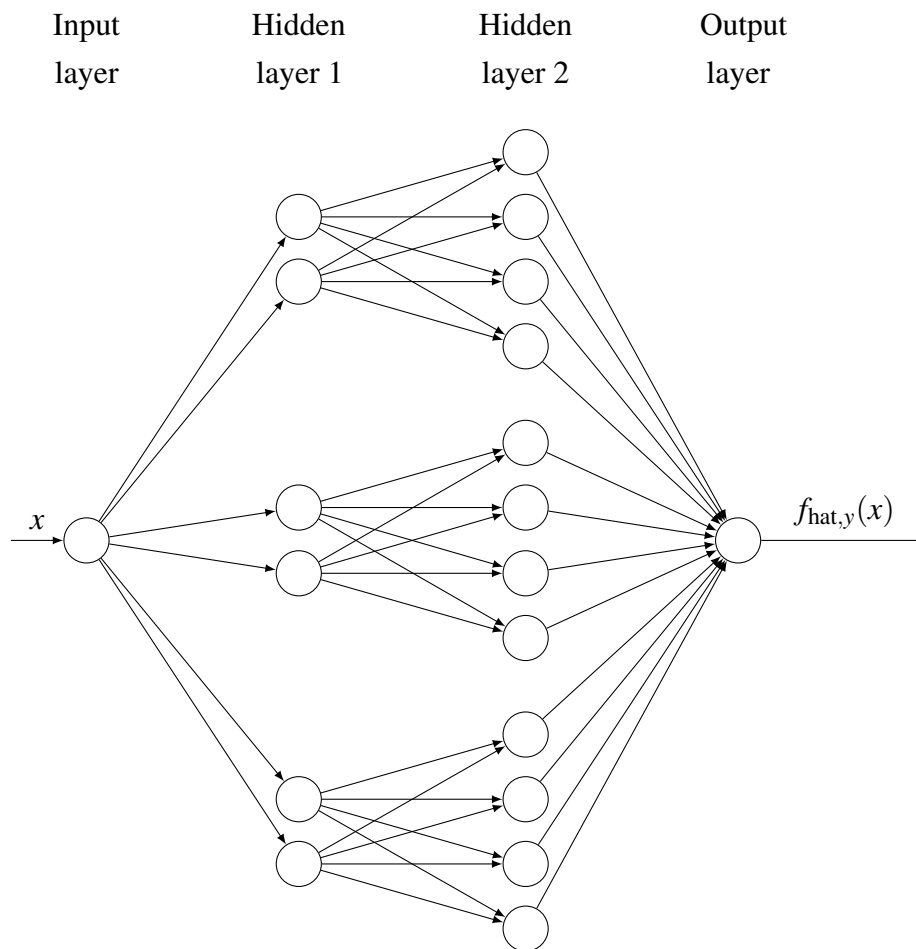
$$f_{|\mathbf{j}|_1 + k}^{(s)}(x) = f_{\text{hat}, x_{\mathbf{i}_k}^{(k)}}(x^{(k)}) \quad (2.15)$$

für $1 \leq k \leq d$ und

$$f_k^{(s)}(x) = 1$$

für $|\mathbf{j}|_1 + d + 1 \leq k \leq 2^s$.

Da das neuronale Netz $f_{\text{net}, \mathbf{j}, \mathbf{i}}$ aus mehrere neuronalen Netzen zusammengebaut wurde, lässt sich dadurch auch die Anzahl an Schichten und Neuronen pro Schicht durch diese

Abbildung 2.5: Neuronales Netz $f_{\text{hat},y}(x) = \dots$

Struktur erklären. Aus (2.15) entnimmt man, dass f_{net} $s + 2$ verborgene Schichten, durch s -maliges Anwenden von f_{mult} und einer Anwendung von f_{hat} bzw. $f_{\text{id}}(f_{\text{id}})$, hat. Da f_{hat} zwei verborgene Schichten besitzt, ergibt sich daraus die Anzahl an verborgenen Schichten von f_{net} . Die Anzahl der Neuronen pro verborgener Schicht von f_{net} ergeben sich wie folgt:

- Die erste verborgene Schicht enthält maximal $3 \cdot 2 \cdot 2^s = 6 \cdot 2^s$ Neuronen, da dies die erste verborgene Schicht von f_{hat} ist und maximal 2^s mal aufgerufen wird.
- Die zweite verborgene Schicht maximal $3 \cdot 4 \cdot 2^s = 12 \cdot 2^s$ Neuronen, da dies die zweite verborgene Schicht von f_{hat} ist und maximal 2^s -mal aufgerufen wird.
- Die verborgenen Schichten $3, \dots, s + 2$ enthalten maximal $2^{s+1}, 2^s, \dots, 2^3, 2^2$ Neuronen, da wir s -mal f_{mult} ineinander geschachtelt aufrufen.

Da man bei fully connected neuronalen Netzen die Gewichte der Verbindungen zwischen zwei Neuronen auf Null setzen kann, können auch nicht fully connected neuronale Netze in dieser Klasse enthalten sein. Daher liegt auch $f_{\text{net}, \mathbf{j}, \mathbf{i}}$ in der der Klasse aller fully connected neuronaler Netze, mit $s + 2$ verborgenen Schichten mit jeweils $24 \cdot (N + d)$ Neuronen pro Schicht, da für die größte Anzahl an Neuronen in einer Schicht

$$12 \cdot 2^s = 12 \cdot 2^{\lceil \log_2(N+d) \rceil} \leq 12 \cdot 2^{\log_2(N+d)+1} = 24 \cdot (N + d)$$

gilt. Weiterhin erkennt man durch die Zusammensetzung der neuronalen Netze, dass alle Gewichte im Betrag durch $c \cdot \max\{\frac{M}{2a}, R^2\}$ beschränkt sind, wobei $c > 0$ ist.

2.2 Definition der Gewichte der Ausgabeschicht

Wir definieren unseren Neuronale-Netze-Regressionsschätzer $\tilde{m}_n(x)$ durch:

$$\tilde{m}_n(x) = \sum_{\mathbf{i} \in [M]^d} \sum_{\substack{\mathbf{j} \in [N]^d \\ \|\mathbf{j}\|_1 \leq N}} a_{\mathbf{i}, \mathbf{j}} \cdot f_{\text{net}, \mathbf{j}, \mathbf{i}}(x), \quad (2.16)$$

wobei n die Größe unserer gegebenen Datenmenge (2.1) ist und wir die Koeffizienten $a_{\mathbf{i}, \mathbf{j}}$ durch Minimierung des Funktionals

$$\begin{aligned} \varphi(a_{\mathbf{i}, \mathbf{j}}) &:= \frac{1}{n} \sum_{i=1}^n |Y_i - \tilde{m}_n(X_i)|^2 + \lambda \|(a_{\mathbf{i}, \mathbf{j}})\|_2^2 \\ &= \frac{1}{n} \sum_{i=1}^n |Y_i - \tilde{m}_n(X_i)|^2 + \frac{c}{n} \cdot \sum_{\mathbf{i} \in [M]^d} \sum_{\substack{\mathbf{j} \in [N]^d \\ \|\mathbf{j}\|_1 \leq N}} a_{\mathbf{i}, \mathbf{j}}^2 \end{aligned} \quad (2.17)$$

für eine Konstante $c > 0$ und Regularitätsterm $\lambda \|(a_{\mathbf{i}, \mathbf{j}})\|_2^2$ erhalten. Dieses regularisierte lineare Kleinste-Quadrate Schätzung erhalten wir durch die Lösung des, in (2.18) folgenden, linearen Gleichungssystems. Dafür definieren wir uns die Menge

$$\{U_s \mid s = 1, \dots, S\} = \left\{ f_{\text{net}, \mathbf{j}, \mathbf{i}}(x) \mid \mathbf{i} \in [M]^d \text{ und } |\mathbf{j}|_1 \leq N \text{ mit } \mathbf{j} \in [N]^d \right\}$$

wobei

$$S = |[M]^d| \cdot \binom{N+d}{d} = (M+1)^d \cdot \binom{N+d}{d}$$

die Kardinalität der Menge ist. Diese Kardinalität erhalten wir mit einem Kombinatorik Argument. Wir wissen, dass es insgesamt $(M+1)^d$ Möglichkeiten gibt d viele Zahlen aus einer Menge mit der Größe $(M+1)$ zu ziehen mit Zurücklegen und da wir Vektoren betrachten und die Komponenten nicht vertauschbar sind ist auch die Reihenfolge der Ziehung zu beachten. Für jede dieser $(M+1)^d$ Möglichkeiten ist noch zu beachten, dass wir zusätzlich d mal aus einer Menge mit $(N+1)$ vielen Zahlen ziehen müssen und gleichzeitig die Bedingung dass die Summe der gezogen d Elemente zwischen Null und N liegt. Gesucht ist also

$$\left| \left\{ \mathbf{j} \in [N]^d : |\mathbf{j}|_1 \leq N \right\} \right| =: H.$$

Wir stellen fest, dass:

$$\begin{aligned} & \left\{ \mathbf{j} \in [N]^d : |\mathbf{j}|_1 \leq N \right\} \\ &= \left\{ \mathbf{j} \in [N]^d : |\mathbf{j}|_1 = 0 \right\} \cup \left\{ \mathbf{j} \in [N]^d : |\mathbf{j}|_1 = 1 \right\} \cup \dots \cup \left\{ \mathbf{j} \in [N]^d : |\mathbf{j}|_1 = N \right\} \end{aligned}$$

gilt. Mit Lemma 1.16 wissen wir, dass für $d, N \in \mathbb{N}$ und $k \in \mathbb{N}_0$:

$$\left| \left\{ \mathbf{j} \in [N]^d : |\mathbf{j}|_1 = k \right\} \right| = \binom{d+k-1}{k}$$

gilt. Damit erhalten wir:

$$|H| = \sum_{k=0}^N \binom{N-1+k}{k} = \binom{N+d}{d},$$

mit dem *Hockey Stick Lemma*

$$\binom{n}{k} = \sum_{i=0}^k \binom{n-k-1+i}{i} \quad (k, n \in \mathbb{N} \text{ mit } k < n).$$

Wir setzen nun

$$\mathbf{U} = (U_s(X_i))_{1 \leq i \leq n, 1 \leq s \leq S} \quad \text{und} \quad \mathbf{Y} = (Y_i)_{i=1, \dots, n}.$$

Im folgenden Lemma bestimmen wir den Koeffizientenvektor unseres Schätzers (2.16).

Lemma 2.3. *Der Koeffizientenvektor unseres Schätzers (2.16) ist die eindeutige Lösung des linearen Gleichungssystems*

$$\left(\frac{1}{n} \mathbf{U}^T \mathbf{U} + \frac{c}{n} \cdot \mathbf{1} \right) \mathbf{a} = \frac{1}{n} \mathbf{U}^T \mathbf{Y} \quad (2.18)$$

Hierbei ist $\mathbf{1}$ eine $S \times S$ -Einheitsmatrix und $\mathbf{a} \in \mathbb{R}^S$, wobei \mathbf{a} wie in Gleichung (2.17) den Ausdruck:

$$\begin{aligned} & \frac{1}{n} \sum_{i=1}^n |Y_i - \tilde{m}_n(X_i)|^2 + \frac{c}{n} \cdot \sum_{\mathbf{i} \in [M]^d} \sum_{\substack{\mathbf{j} \in [N]^d \\ |\mathbf{j}|_1 \leq N}} a_{\mathbf{i}, \mathbf{j}}^2 \\ &= \frac{1}{n} (\mathbf{Y} - \mathbf{U}\mathbf{a})^T (\mathbf{Y} - \mathbf{U}\mathbf{a}) + \frac{c}{n} \cdot \mathbf{a}^T \mathbf{a} \end{aligned} \quad (2.19)$$

minimiert.

Beweis. Der Schätzer aus Gleichung (2.16) lässt sich umschreiben zu

$$\tilde{m}_n(x) = \sum_{s=1}^S a_s \cdot U_s(x) \quad (2.20)$$

mit $(a_s)_{s=1, \dots, S} = \mathbf{a} \in \mathbb{R}^S$. Da $\mathbf{Y}^T \mathbf{U}\mathbf{a} = \mathbf{a}^T \mathbf{U}^T \mathbf{Y}$ gilt, da dieser Ausdruck eine reelle Zahl und damit insbesondere symmetrisch ist erhalten wir mit Gleichung (2.19):

$$\begin{aligned} & \frac{1}{n} (\mathbf{Y} - \mathbf{U}\mathbf{a})^T (\mathbf{Y} - \mathbf{U}\mathbf{a}) + \frac{c}{n} \cdot \mathbf{a}^T \mathbf{a} \\ &= \frac{1}{n} (\mathbf{Y}^T \mathbf{Y} - \mathbf{Y}^T \mathbf{U}\mathbf{a} - \mathbf{a}^T \mathbf{U}^T \mathbf{Y} + \mathbf{a}^T \mathbf{U}^T \mathbf{U}\mathbf{a}) + \frac{c}{n} \cdot \mathbf{a}^T \mathbf{a} \\ &= \frac{1}{n} (\mathbf{Y}^T \mathbf{Y} - 2\mathbf{Y}^T \mathbf{U}\mathbf{a}) + \mathbf{a}^T \left(\frac{1}{n} \mathbf{U}^T \mathbf{U} + \frac{c}{n} \cdot \mathbf{1} \right) \mathbf{a}. \end{aligned} \quad (2.21)$$

Die Matrix $\mathbf{U}^T \mathbf{U} \in \mathbb{R}^{S \times S}$ ist positiv semidefinit, denn aufgrund der Verschiebungseigenschaft des Standardskalarprodukts gilt für alle $x \in \mathbb{R}$:

$$\langle x, \mathbf{U}^T \mathbf{U} x \rangle = \langle \mathbf{U} x, \mathbf{U} x \rangle \geq 0.$$

Zudem wissen wir dass $\frac{c}{n} \mathbf{1}$ durch die Wahl von c nur positive Eigenwerte besitzt und damit positiv definit ist. Daher wissen wir, dass die Matrix

$$\mathbf{A} := \frac{1}{n} \mathbf{U}^T \mathbf{U} + \frac{c}{n} \cdot \mathbf{1}$$

also Summe einer positiv semidefiniten und einer positiv definiten Matrix nur positive Eigenwerte besitzt (REFERENZ), damit also positiv definit und damit insbesondere invertierbar ist. Zudem ist die Matrix \mathbf{A} symmetrisch. Mit

$$\mathbf{b} = \frac{1}{n} \cdot \mathbf{A}^{-1} \mathbf{U}^T \mathbf{Y} \in \mathbb{R}^S$$

und

$$\mathbf{b}^T \mathbf{A} \mathbf{a} = \mathbf{a}^T \mathbf{A} \mathbf{b} = \mathbf{Y}^T \mathbf{U} \mathbf{a},$$

was aus der Symmetrie von \mathbf{A} folgt, erhalten wir in Gleichung (2.21):

$$\begin{aligned} & \frac{1}{n}(\mathbf{Y}^T \mathbf{Y} - 2\mathbf{Y}^T \mathbf{U} \mathbf{a}) + \mathbf{a}^T \left(\frac{1}{n} \mathbf{U}^T \mathbf{U} + \frac{c}{n} \cdot \mathbf{1} \right) \mathbf{a} \\ &= \frac{1}{n}(\mathbf{Y}^T \mathbf{Y} - 2\mathbf{Y}^T \mathbf{U} \mathbf{a}) + \mathbf{a}^T \mathbf{A} \mathbf{a} \\ &= \frac{1}{n} \mathbf{Y}^T \mathbf{Y} - \frac{1}{n} \mathbf{b}^T \mathbf{A} \mathbf{a} - \frac{1}{n} \mathbf{a}^T \mathbf{A} \mathbf{b} + \mathbf{a}^T \mathbf{A} \mathbf{a} \\ &= \frac{1}{n} \mathbf{Y}^T \mathbf{Y} - \frac{1}{n} \mathbf{b}^T \mathbf{A} \mathbf{a} - \frac{1}{n} \mathbf{a}^T \mathbf{A} \mathbf{b} + \frac{1}{n} \mathbf{b}^T \mathbf{U}^T \mathbf{Y} - \frac{1}{n} \mathbf{Y}^T \mathbf{U} \mathbf{b} + \mathbf{a}^T \mathbf{A} \mathbf{a} \\ &= \frac{1}{n} \mathbf{Y}^T \mathbf{Y} - \frac{1}{n} \mathbf{b}^T \mathbf{A} \mathbf{a} - \frac{1}{n} \mathbf{a}^T \mathbf{A} \mathbf{b} + \frac{1}{n} \mathbf{b}^T \mathbf{A} \mathbf{A}^{-1} \mathbf{U}^T \mathbf{Y} - \frac{1}{n} \mathbf{Y}^T \mathbf{U} \mathbf{b} + \mathbf{a}^T \mathbf{A} \mathbf{a} \\ &= \frac{1}{n} \mathbf{Y}^T \mathbf{Y} - \frac{1}{n} \mathbf{b}^T \mathbf{A} \mathbf{a} - \frac{1}{n} \mathbf{a}^T \mathbf{A} \mathbf{b} + \frac{1}{n} \mathbf{b}^T \mathbf{A} \mathbf{b} - \frac{1}{n^2} \mathbf{Y}^T \mathbf{U} \mathbf{A}^{-1} \mathbf{U}^T \mathbf{Y} + \mathbf{a}^T \mathbf{A} \mathbf{a} \\ &= \mathbf{a}^T \mathbf{A} \mathbf{a} - \frac{1}{n} \mathbf{b}^T \mathbf{A} \mathbf{a} - \frac{1}{n} \mathbf{a}^T \mathbf{A} \mathbf{b} + \mathbf{b}^T \mathbf{A} \mathbf{b} + \frac{1}{n} \mathbf{Y}^T \mathbf{Y} - \frac{1}{n^2} \mathbf{Y}^T \mathbf{U} \mathbf{A}^{-1} \mathbf{U}^T \mathbf{Y} \\ &= \left(\mathbf{a} - \frac{1}{n} \cdot \mathbf{A}^{-1} \mathbf{U}^T \mathbf{Y} \right)^T \mathbf{A} \left(\mathbf{a} - \frac{1}{n} \cdot \mathbf{A}^{-1} \mathbf{U}^T \mathbf{Y} \right) - \frac{1}{n} \mathbf{Y}^T \mathbf{Y} - \frac{1}{n^2} \mathbf{Y}^T \mathbf{U} \mathbf{A}^{-1} \mathbf{U}^T \mathbf{Y}. \end{aligned}$$

Die letzte Gleichung wird für $\mathbf{a} = \frac{1}{n} \cdot \mathbf{A}^{-1} \mathbf{U}^T \mathbf{Y}$ minimal, da wir wissen dass \mathbf{A} positiv definit ist und damit $x^T \mathbf{A} x > 0$ für alle $x \in \mathbb{R}^S$ mit $x \neq 0$ gilt und $(\mathbf{a} - \mathbf{b})^T \mathbf{A} (\mathbf{a} - \mathbf{b}) = 0$ genau dann, wenn $\mathbf{a} = \mathbf{b}$ gilt. Dies zeigt also, dass der Koeffizientenvektor unseres Schätzers (2.16) die eindeutige Lösung des linearen Gleichungssystems (2.18) ist. \square

Bemerkung 2.4. Da der Koeffizientenvektor \mathbf{a} die Gleichung (2.19) minimiert, erhalten wir, wenn wir den Koeffizientenvektor gleich Null setzen:

$$\frac{1}{n}(\mathbf{Y} - \mathbf{U} \mathbf{a})^T (\mathbf{Y} - \mathbf{U} \mathbf{a}) + \frac{c}{n} \cdot \mathbf{a}^T \mathbf{a} \leq \frac{1}{n} \sum_{i=1}^n Y_i^2,$$

was uns erlaubt eine obere Schranke für den absoluten Wert unserer Koeffizienten abzuleiten. Daraus können wir folgern, dass unser Neuronale-Netze-Regressionsschätzer \tilde{m}_n beschränkt ist, da $f_{\text{net},j,i}$ ebenfalls beschränkt ist.

Kapitel 3

Resultat zur

Konvergenzgeschwindigkeit

In dieser Arbeit behandeln wir Neuronale-Netze-Regressionsschätzer im Kontext der nichtparametrischen Regression mit zufälligem Design.

Im Gegensatz zur parametrischen Regression ist bei der nichtparametrischen die Bauart der zu schätzenden Funktion komplett unbekannt, was den Vorteil besitzt, dass weniger Annahmen getroffen werden müssen, man aber dadurch noch mehr Daten benötigt, um eine Funktion zu schätzen.

Bei der nichtparametrischen Regressionsschätzung seien $(X, Y), (X_1, Y_1), (X_2, Y_2), \dots$ unabhängig identisch verteilte (u.i.v.) $\mathbb{R}^d \times \mathbb{R}$ -wertige Zufallsvariablen mit $\mathbb{E}[Y^2] < \infty$ und $d \in \mathbb{N}$. Zudem sei $m: \mathbb{R}^d \rightarrow \mathbb{R}$ definiert durch $m(x) = \mathbb{E}[Y \mid X = x]$ die zugehörige Regressionsfunktion. Ausgehend von der Stichprobe

$$(X_1, Y_1), \dots, (X_n, Y_n),$$

mit Stichprobengröße $n \in \mathbb{N}$, soll die Regressionsfunktion m geschätzt werden.

Das Problem der Regressionsschätzung bei zufälligem Design lässt sich wie folgt erläutern: In Anwendungsfällen ist üblicherweise die Verteilung von (X, Y) unbekannt, daher kann $m(x) = \mathbb{E}[Y \mid X = x]$ nicht berechnet werden. Oft ist es aber möglich, Werte von (X, Y) zu beobachten. Ziel ist es dann, daraus die Regressionsfunktion m zu schätzen. Im Hinblick auf die Minimierung des L_2 -Risikos sollte dabei der L_2 -Fehler der Schätzfunktion möglichst klein sein.

Für das L_2 -Risiko einer beliebigen messbaren Funktion $f: \mathbb{R}^d \rightarrow \mathbb{R}$ gilt:

$$\mathbb{E}[|f(X) - Y|^2] = \mathbb{E}[|m(X) - Y|^2] + \int_{\mathbb{R}^d} |f(x) - m(x)|^2 \mathbb{P}_X(dx),$$

d.h. der mittlere quadratische Vorhersagefehler einer Funktion ist darstellbar als Summe des L_2 -Risikos der Regressionsfunktion (unvermeidbarer Fehler) und des L_2 -Fehlers

der entsteht aufgrund der Verwendung von f an Stelle von m bei der Vorhersage bzw. Approximation des Wertes von Y .

Formal führt das daher auf folgende Problemstellung: $(X, Y), (X_1, Y_1), (X_2, Y_2), \dots$ seien u.i.v. $\mathbb{R}^d \times \mathbb{R}$ wertige Zufallsvariablen mit $\mathbb{E}[Y^2] < \infty$ und $m: \mathbb{R}^d \rightarrow \mathbb{R}$ definiert durch $m(x) = \mathbb{E}[Y | X = x]$ sei die zugehörige Regressionsfunktion. Gegeben sei die Datenmenge

$$\mathcal{D}_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}.$$

Gesucht ist eine Schätzung

$$m_n(\cdot) = m_n(\cdot, \mathcal{D}_n): \mathbb{R}^d \rightarrow \mathbb{R}$$

von m , für die der L_2 -Fehler

$$\int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx)$$

möglichst „klein“ ist. (Referenz Györfi (2002))

In diesem Kapitel stellen wir das Hauptresultat dieser Arbeit vor. Ziel im Folgenden ist es, eine Abschätzung des erwarteten L_2 -Fehlers

$$\mathbb{E} \int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx)$$

im Falle unseres Neuronale-Netze-Regresssionsschätzers aus Kapitel 2 mit einer (p, C) -glatten Regressionsfunktion m herzuleiten.

Satz 3.1. *Angenommen die Verteilung von Y erfüllt*

$$\mathbb{E} \left[e^{c_1 \cdot |Y|^2} \right] < \infty$$

für eine Konstante $c_1 > 0$ und die Verteilung von X hat einen beschränkten Träger $\text{supp}(\mathbb{P}_X)$. Sei $m(x) = \mathbb{E}[Y | X = x]$ die zu dem Tupel (X, Y) gehörige Regressionsfunktion. Angenommen m ist (p, C) -glatt, mit $q \in \mathbb{N}_0$ und $s \in (0, 1]$ und sei $C > 0$.

Wir betrachten unseren Neuronale-Netze-Regresssionsschätzer \tilde{m}_n aus Kapitel 2.2 welcher aus mehreren neuronalen Netzen zusammengebaut wurde. Für diesen wählen wir unsere Aktivierungsfunktion σ als den logistische squasher aus Gleichung logsquasher. Aufgrund der Konstruktion unseren Schätzers wählen wir zudem $N \geq q, M = M_n = \lceil c_2 \cdot n^{1/(2p+d)} \rceil$ mit $c_2 > 0$ und unabhängig von n , $R = R_n = n^{d+4}$ und $a = a_n = (\log n)^{1/(6(N+d))}$. Sei $\beta_n = c_3 \cdot \log(n)$ für eine hinreichend große und von n unabhängige Konstante $c_3 > 0$ und sei m_n gegeben durch

$$m_n(x) = T_{\beta_n} \tilde{m}_n(x)$$

mit $T_\beta z = \max\{\min\{z, \beta\}, -\beta\}$ für $z \in \mathbb{R}$ und $\beta > 0$.

Dann erhalten wir für hinreichend großes n :

$$\mathbb{E} \int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \leq c_{fn} \cdot (\log n)^3 \cdot n^{-\frac{2p}{2p+d}},$$

wobei $c_{fn} > 0$ ist und nicht von n abhängt.

Die nächsten Lemmata benötigen wir für den Beweis unseres Hauptresultats, einer Aussage über die Konvergenzgeschwindigkeit unseres Neuronale-Netze-Schätzers. Diese Lemmata werden hier nur der Vollständigkeit halber und ohne Beweis aufgeführt.

Lemma 3.2 ([AB19], Lemma 5). *Sei $M \in \mathbb{N}$ und $\sigma: \mathbb{R} \rightarrow [0, 1]$ 2-zulässig nach Definition 1.7. Sei $a \geq 1$ und $R \in \mathbb{R}$ mit*

$$R \geq \max \left\{ \frac{\|\sigma''\|_\infty \cdot (M+1)}{2 \cdot |\sigma'(t_{\sigma, \text{id}})|}, \frac{9 \cdot \|\sigma''\|_\infty \cdot a}{|\sigma'(t_{\sigma, \text{id}})|}, \frac{20 \cdot \|\sigma'''\|_\infty}{3 \cdot |\sigma''(t_\sigma)|} \cdot 3^{3 \cdot 3^s} \cdot a^{3 \cdot 2^s}, 1792 \cdot \frac{\max\{\|\sigma''\|_\infty, \|\sigma'''\|_\infty, 1\}}{\min\{2 \cdot |\sigma'(t_{\sigma, \text{id}})|, |\sigma''(t_\sigma)|, 1\}} \cdot M^3 \right\}$$

und sei $y \in [-a, a]^d$. Sei $N \in \mathbb{N}$ und $\mathbf{j} \in [\mathbb{N}_0]^d$ so, dass $|\mathbf{j}|_1 \leq N$ gilt und wir setzen $s = \lceil \log_2(N+d) \rceil$. Sei $f_{\text{id}}, f_{\text{mult}}$ und $f_{\text{hat}, z}$ (für $z \in \mathbb{R}$) die neuronalen Netze aus Lemma 1.12, Lemma 1.13 und Lemma 1.15. Wir definieren das Netz $f_{\text{net}, \mathbf{j}, y}$ durch:

$$f_{\text{net}, \mathbf{j}, y}(x) = f_1^{(0)}(x).$$

wobei

$$f_k^{(l)}(x) = f_{\text{mult}}\left(f_{2k-1}^{(l+1)}(x), f_{2k}^{(l+1)}(x)\right)$$

für $1 \leq k \leq 2^l$ und $0 \leq l \leq s-1$ und

$$f_k^{(s)}(x) = f_{\text{id}}(f_{\text{id}}(x^{(l)} - y^{(l)}))$$

für $j_1 + j_2 + \dots + j_{l-1} + 1 \leq k \leq j_1 + j_2 + \dots + j_l$ und $1 \leq l \leq d$ und

$$f_{|\mathbf{j}|_1+k}^{(s)}(x) = f_{\text{hat}, y^{(k)}}(x^{(k)})$$

für $1 \leq k \leq d$ und

$$f_k^{(s)}(x) = 1$$

für $|\mathbf{j}|_1 + d + 1 \leq k \leq 2^s$.

Dann erhalten wir für $x \in [-a, a]^d$:

$$\begin{aligned} & \left| f_{\text{net}, \mathbf{j}, y}(x) - (x^{(1)} - y^{(1)})^{j_1} \dots (x^{(d)} - y^{(d)})^{j_d} \prod_{j=1}^d \left(1 - \frac{M}{2a} \cdot |x^{(j)} - y^{(j)}|\right)_+ \right| \\ & \leq c \cdot 3^{3 \cdot 3^s} \cdot a^{3 \cdot 2^s} \cdot M^3 \cdot \frac{1}{R}, \end{aligned}$$

für eine Konstante $c > 0$.

Lemma 3.3 ([AB19], Lemma 8). *Sei $\beta_n = c_1 \cdot \log(n)$ für eine hinreichend große Konstante $c_1 > 0$. Angenommen die Verteilung von Y erfüllt*

$$\mathbb{E} \left[e^{c_2 \cdot |Y|^2} \right] < \infty$$

für eine Konstante $c_2 > 0$ und dass der Betrag der Regressionsfunktion m beschränkt ist. Sei \mathcal{F}_n eine Menge von Funktionen $f: \mathbb{R}^d \rightarrow \mathbb{R}$ und wir nehmen an, dass der Schätzer m_n

$$m_n = T_{\beta_n} \tilde{m}_n$$

erfüllt, mit

$$\tilde{m}_n(\cdot) = \tilde{m}_n(\cdot, (X_1, Y_1), \dots, (X_n, Y_n)) \in \mathcal{F}_n$$

und

$$\frac{1}{n} \sum_{i=1}^n |Y_i - \tilde{m}_n(X_i)|^2 \leq \min_{l \in \Theta_n} \left(\frac{1}{n} \sum_{i=1}^n |Y_i - g_{n,l}(X_i)|^2 + \text{pen}_n(g_{n,l}) \right)$$

mit einer nichtleeren Parametermenge Θ_n , zufällige Funktionen $g_{n,l}: \mathbb{R}^d \rightarrow \mathbb{R}$ und deterministischen penalty Terme $\text{pen}_n(g_{n,l}) \geq 0$, wobei die zufälligen Funktionen $g_{n,l}: \mathbb{R}^d \rightarrow \mathbb{R}$ nur von den Zufallsvariablen

$$\mathbf{b}_1^{(1)}, \dots, \mathbf{b}_r^{(1)}, \dots, \mathbf{b}_1^{(I_n)}, \dots, \mathbf{b}_r^{(I_n)},$$

abhängen, die unabhängig von $(X_1, Y_1), (X_2, Y_2), \dots$ sind. Dann erfüllt m_n :

$$\begin{aligned} & \mathbb{E} \int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \\ & \leq \frac{c \cdot \log(n)^2 \cdot \left(\log \left(\sup_{x_1^n \in (\text{supp}(X))^n} \mathcal{N}_1 \left(\frac{1}{n \cdot \beta_n}, \mathcal{F}_n, x_1^n \right) \right) + 1 \right)}{n} \\ & \quad + 2 \cdot \mathbb{E} \left(\min_{l \in \Theta_n} \int |g_{n,l}(x) - m(x)|^2 \mathbb{P}_X(dx) + \text{pen}_n(g_{n,l}) \right), \end{aligned}$$

für $n > 1$ und einer Konstante $c > 0$ welche nicht von n abhängt.

Das nächste Lemma benötigen wir um eine Schranke für die Überdeckungszahl $\mathcal{N}_1 \left(\frac{1}{n \cdot \beta_n}, \mathcal{F}_n, x_1^n \right)$ zu finden.

Lemma 3.4 ([AB19], Lemma 9). Sei $a > 0$ und $d, N, J_n \in \mathbb{N}$ so, dass $J_n \leq n^{c_1}$ und setze $\beta_n = c_2 \cdot \log(n)$. Sei σ 2-zulässig nach Definition 1.7. Sei \mathcal{F} die Menge aller Funktionen die durch Definition 1.6 definiert sind mit $k_1 = k_2 = \dots = k_L = 24 \cdot (N + d)$ und einer Beschränkung der Betrag der Gewichte durch $c_3 \cdot n^{c_4}$.

$$\mathcal{F}^{(J_n)} = \left\{ \sum_{j=1}^{J_n} a_j \cdot f_j : f_j \in \mathcal{F} \quad \text{und} \quad \sum_{j=1}^{J_n} a_j^2 \leq c_5 \cdot n^{c_6} \right\}.$$

Dann gilt für $n > 1$:

$$\log \left(\sup_{x_1^n \in [-a, a]^{d \cdot n}} \mathcal{N}_1 \left(\frac{1}{n \cdot \beta_n}, \mathcal{F}^{(J_n)}, x_1^n \right) \right) \leq c \cdot \log(n) \cdot J_n,$$

für eine Konstante c die nur von L, N, a und d abhängt.

Mit diesen Hilfsresultaten können wir nun unser Hauptresultat beweisen.

Beweis von Satz 3.1. Da nach Voraussetzung $\text{supp}(\mathbb{P}_X)$ beschränkt ist und $\text{supp}(\mathbb{P}_X)$ immer abgeschlossen ist, wissen wir dass $\text{supp}(\mathbb{P}_X)$ beschränkt ist. Wir wissen, dass m als (p, C) -glatte Funktion insbesondere stetig ist und damit auf einer beschränkten Menge ihr Maximum und Minimum annimmt. Wir können unser n also wieder so groß wählen, dass ohne Beschränkung der Allgemeinheit $\|m\|_\infty \leq \beta_n$ gilt.

Sei \mathcal{F} die Menge aller Funktion aus Definition 1.6 mit Aktivierungsfunktion σ und $L = s + 2 = \lceil \log_2(N + d) \rceil + 2$, mit $k_1 = k_2 = \dots = k_L = 24 \cdot (N + d)$ und der Eigenschaft, dass der Betrag der Gewichte durch n^{c_4} , mit einer Konstante $c_4 > 0$ beschränkt ist. Sei für eine Konstante $c_5 > 0$

$$\mathcal{F}^{(J_n)} = \left\{ \sum_{j=1}^{J_n} a_j \cdot f_j \mid f_j \in \mathcal{F} \quad \text{und} \quad \sum_{j=1}^{J_n} a_j^2 \leq c_5 \cdot n \right\}$$

wobei

$$c_5 := \max \left\{ \frac{1 + \mathbb{E}[Y^2]}{c_6}, c_2 \cdot (N + 1)^d \cdot \max \left\{ \left| \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(x_i) \right|^2 \mid \mathbf{j} \in [q]^d, |\mathbf{j}|_1 \leq q \right\} \right\}, \quad (3.1)$$

mit c_6 als Konstante aus Gleichung (2.17). Da wir uns in der nichtparametrischen Regressionsschätzung befinden gilt unter anderem die Bedingung $\mathbb{E}[Y^2] < \infty$, und daher ist c_5 auch wohldefiniert. Weiterhin ist

$$J_n = (M_n + 1)^d \cdot |\{\mathbf{j} \mid \mathbf{j} \in [N]^d, |\mathbf{j}|_1 \leq N\}|,$$

die Kardinalität der Menge $\mathcal{F}^{(J_n)}$. Ohne die Restriktion $|\mathbf{j}|_1 \leq N$ lässt sich

$$|\{\mathbf{j} \mid \mathbf{j} \in [N]^d\}|$$

durch eine Analogie zu einem Urnenexperiment bestimmten. Wir betrachten die Anzahl an Möglichkeiten, wie man d -Mal mit Zurücklegen (da auch mehrere Komponenten den gleichen Wert haben können) und mit Beachtung der Reihenfolge (da wir einen Vektor betrachten und die Komponenten nicht vertauschen können) aus $(N + 1)$ Kugeln ziehen kann. Durch das Weglassen der Restriktion $|\mathbf{j}|_1 \leq N$ erhalten wir:

$$J_n \leq (M_n + 1)^d \cdot (N + 1)^d. \quad (3.2)$$

Da m nach Voraussetzung (p, C) -glatt ist, folgt:

$$z := \max_{\mathbf{i} \in [M]^d, \mathbf{j} \in [q]^d, |\mathbf{j}|_1 \leq q} |\partial^{\mathbf{j}} m(x_{\mathbf{i}})| < \infty, \quad (3.3)$$

denn das größte Element muss auch beschränkt sein, wenn der Abstand zweier beliebiger Elemente beschränkt ist. Sei

$$g_n(x) = \sum_{\mathbf{i} \in [M_n]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(x_{\mathbf{i}}) \cdot f_{\text{net}, \mathbf{j}, \mathbf{i}}(x).$$

Da nach Konstruktion $f_{\text{net}, \mathbf{j}, \mathbf{i}} \in \mathcal{F}$ ist, folgt mit Gleichung (3.1), dass für n hinreichend groß

$$\sum_{\mathbf{i} \in [M_n]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \left| \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(x_{\mathbf{i}}) \right|^2 \leq (M_n + 1)^d (N + 1)^d \cdot \left| \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(x_{\mathbf{i}}) \right|^2 \leq c_5 \cdot n,$$

gilt und damit g_n in $\mathcal{F}^{(J_n)}$ liegt. Wir wählen das Ereignis

$$A_n := \frac{1}{n} \sum_{i=1}^n Y_i^2 \leq 1 + \mathbb{E}[Y^2]. \quad (3.4)$$

Wir wissen, dass aufgrund der Unabhängigkeit und identischen Verteiltheit der $\mathbb{R}^d \times \mathbb{R}$ -wertigen Zufallsvariablen $(X, Y), (X_1, Y_1), (X_2, Y_2), \dots$ auch die Zufallsvariablen Y_1, \dots, Y_n unabhängig und identisch verteilt sind. Daraus folgern wir $\mathbb{E}[\frac{1}{n} \sum_{i=1}^n Y_i^2] = \mathbb{E}[Y^2]$ mit der Linearität des Erwartungswerts. Mit Hilfe der Monotonie der Wahrscheinlichkeitsfunktion \mathbb{P} und der Tschebycheff-Ungleichung für $\varepsilon = 1$ ([Kle13], Satz 5.11) erhalten wir:

$$\begin{aligned} \mathbb{P}(A_n^c) &= \mathbb{P}\left(\frac{1}{n} \sum_{i=1}^n Y_i^2 - \mathbb{E}[Y^2] \geq 1\right) \\ &\leq \mathbb{P}\left(\left|\frac{1}{n} \sum_{i=1}^n Y_i^2 - \mathbb{E}[Y^2]\right| \geq 1\right) \\ &\leq \mathbb{V}\left[\frac{1}{n} \sum_{i=1}^n Y_i^2\right]. \end{aligned}$$

Da die Zufallsvariablen Y_1, \dots, Y_n u.i.v. sind, folgt mit den Rechenregeln der Varianz:

$$\begin{aligned}\mathbb{P}(A_n^c) &\leq \frac{n \cdot \mathbb{V}[Y^2]}{n^2} \\ &= \frac{\mathbb{V}[Y^2]}{n} \\ &= \frac{c_7}{n},\end{aligned}\tag{3.5}$$

wobei $c_7 := \mathbb{V}[Y^2]$ ist.

Sei

$$\hat{m}_n := \mathbb{1}_{A_n} m_n + \mathbb{1}_{A_n^c} T_{\beta_n} g_n$$

mit $m_n = T_{\beta_n} \tilde{m}_n$.

Für zwei beliebige reelle Zahlen $u, v \in \mathbb{R}$ gilt durch $0 \leq (u - v)^2 = u^2 + v^2 - 2uv$:

$$u^2 + v^2 \geq 2uv$$

und damit schließlich:

$$\begin{aligned}|(u - v)^2| &= |u^2 - 2uv + v^2| \\ &\leq u^2 + 2uv + v^2 \\ &\leq 2u^2 + 2v^2.\end{aligned}\tag{3.6}$$

Mit Ungleichung (3.6) und durch die Unabhängigkeit von A_n von den Zufallsvariablen X, X_1, \dots, X_n erhalten wir:

$$\begin{aligned}\mathbb{E} \left[\int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \mathbb{1}_{A_n^c} \right] &= \mathbb{E} \left[\int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \right] \cdot \mathbb{P}(A_n^c) \\ &\leq \mathbb{E} \left[\int 2m_n(x)^2 + 2m(x)^2 \mathbb{P}_X(dx) \right] \cdot \mathbb{P}(A_n^c) \\ &\leq \mathbb{E} \left[\int 2\beta_n^2 + 2\beta_n^2 \mathbb{P}_X(dx) \right] \cdot \mathbb{P}(A_n^c) \\ &= 4\beta_n^2 \cdot \mathbb{P}(A_n^c) \\ &\stackrel{(3.5)}{\leq} \frac{4 \cdot c_7 \cdot \beta_n^2}{n}.\end{aligned}\tag{3.7}$$

Da wir nach Bemerkung (2.4) wissen, dass \tilde{m}_n beschränkt ist, ist m_n nach Konstruktion ebenfalls beschränkt. Wir haben daher bei Ungleichung (3.7) zudem verwendet, dass wir n und c_3 so groß wählen, dass $\max\{\|m\|_\infty, \|m_n\|_\infty\} < \beta_n$ gilt. Bei der letzten Gleichung habe wir schließlich verwendet dass β_n deterministisch und $\mathbb{P}(X \in \text{supp}(\mathbb{P}_X)) = 1$ ist. Durch unsere Definition von \hat{m}_n erhalten wir durch die Monotonie des Erwartungswerts

und einer Abschätzung über den ganzen Raum:

$$\begin{aligned}\mathbb{E} \left[\int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \mathbb{1}_{A_n} \right] &= \mathbb{E} \left[\int |\hat{m}_n(x) - m(x)|^2 \mathbb{P}_X(dx) \mathbb{1}_{A_n} \right] \\ &\leq \mathbb{E} \left[\int |\hat{m}_n(x) - m(x)|^2 \mathbb{P}_X(dx) \right].\end{aligned}\quad (3.8)$$

Zusammen mit (3.5), (3.7), (3.8) und der Linearität des Erwartungswerts erhalten wir dann:

$$\begin{aligned}\mathbb{E} \int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) &= \mathbb{E} \left[\int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \cdot (\mathbb{1}_{A_n^c} + \mathbb{1}_{A_n}) \right] \\ &= \mathbb{E} \left[\int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \mathbb{1}_{A_n^c} \right] \\ &\quad + \mathbb{E} \left[\int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \mathbb{1}_{A_n} \right] \\ &\leq \frac{4 \cdot c_7 \cdot \beta_n^2}{n} + \mathbb{E} \left[\int |\hat{m}_n(x) - m(x)|^2 \mathbb{P}_X(dx) \right].\end{aligned}\quad (3.9)$$

Wir zeigen nun $\tilde{m}_n \in \mathcal{F}^{(J_n)}$.

Nach Gleichung (2.20) können wir unseren Schätzer \tilde{m}_n darstellen durch:

$$\tilde{m}_n(x) = \sum_{j=1}^{J_n} \hat{a}_j \cdot f_j$$

für geeignete $f_j \in \mathcal{F}$ und \hat{a}_j welche

$$\begin{aligned}\frac{c_6}{n} \sum_{j=1}^{J_n} \hat{a}_j^2 &= \frac{c_6}{n} \sum_{\mathbf{i} \in [M]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} a_{\mathbf{i}, \mathbf{j}}^2 \\ &\leq \frac{1}{n} \sum_{i=1}^n |Y_i - \tilde{m}_n(X_i)|^2 + \frac{c_6}{n} \sum_{\mathbf{i} \in [M]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} a_{\mathbf{i}, \mathbf{j}}^2 \\ &\leq \sum_{i=1}^n Y_i^2,\end{aligned}$$

erfüllen, wobei wir bei der letzten Ungleichung wie in (2.17) die minimierende Eigenschaft von $a_{\mathbf{i}, \mathbf{j}}$ verwendet haben und zum Schluss die Koeffizienten Null gesetzt haben. Da $c_6 > 0$ ist, erhalten wir dass die Koeffizienten \hat{a}_j die Eigenschaft

$$\sum_{j=1}^{J_n} \hat{a}_j^2 \leq \frac{1}{n} \sum_{i=1}^n Y_i^2 \cdot \frac{n}{c_6}$$

erfüllen müssen. Aus (3.4) und (3.1) erhalten wir dann die Abschätzung

$$\sum_{j=1}^{J_n} \hat{a}_j^2 \leq \frac{1 + \mathbb{E}[Y^2]}{c_6} \cdot n \leq c_5 \cdot n,$$

woraus durch $f_j \in \mathcal{F}$ dann $\tilde{m}_n \in \mathcal{F}^{(J_n)}$ folgt.

Daher setzen wir $\hat{m}_n := T_{\beta_n} \tilde{m}_n$ für $\tilde{m}_n \in \{\tilde{m}_n, g_n\} \subseteq \mathcal{F}^{(J_n)}$. Die Funktionen \tilde{m}_n und g_n unterscheiden sich in den Vorfaktoren von f_j . Da wir die Koeffizienten $a_{\mathbf{i}, \mathbf{j}}$ von \tilde{m}_n durch Minimierung von (2.17) erhalten haben und nach Voraussetzung $N \geq q$ ist, damit dann $\{0, \dots, q\} \subseteq \{0, \dots, N\}$ und wir bei der Minimierung daher auch insbesondere die Koeffizienten von g_n betrachtet haben, erhalten wir:

$$\begin{aligned}
& \frac{1}{n} \sum_{i=1}^n |Y_i - \tilde{m}_n(X_i)|^2 \\
& \leq \frac{1}{n} \sum_{i=1}^n |Y_i - \tilde{m}_n(X_i)|^2 + \frac{c_6}{n} \cdot \sum_{\mathbf{i} \in [M_n]^d} \sum_{\substack{\mathbf{j} \in [N]^d \\ |\mathbf{j}|_1 \leq N}} a_{\mathbf{i}, \mathbf{j}}^2 \\
& \leq \frac{1}{n} \sum_{i=1}^n |Y_i - g_n(X_i)|^2 + \frac{c_6}{n} \cdot \sum_{\mathbf{i} \in [M_n]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \left| \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(\mathbf{x}_i) \right|^2 \\
& = \frac{1}{n} \sum_{i=1}^n |Y_i - g_n(X_i)|^2 + c_8 \cdot \frac{(M_n + 1)^d}{n},
\end{aligned} \tag{3.10}$$

mit $c_8 = c_6 \cdot (q + 1)^d \cdot \left| \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(\mathbf{x}_i) \right|^2$ als Konstante die unabhängig von n ist. Wir erhalten damit:

$$\frac{1}{n} \sum_{i=1}^n |Y_i - \tilde{m}_n(X_i)|^2 \leq \frac{1}{n} \sum_{i=1}^n |Y_i - g_n(X_i)|^2 + c_8 \cdot \frac{(M_n + 1)^d}{n}, \tag{3.11}$$

da für $\tilde{m}_n = g_n$ die Ungleichung unmittelbar folgt.

Da g_n nach Definition deterministisch, damit also unabhängig von $(X_1, Y_1), (X_2, Y_2), \dots$ ist, sind mit der einelementigen Parametermenge Θ_n , der Funktion $g_{n,1} = g_n$, der Abschätzung (3.11) für $\hat{m}_n = T_{\beta_n} \tilde{m}_n$ mit $\hat{m}_n \in \mathcal{F}^{(J_n)}$ und dem penalty Term $\text{pen}_n(g_{n,1}) = c_8 \cdot \frac{(M_n + 1)^d}{n} > 0$ die Voraussetzungen für Lemma 3.3 erfüllt und wir erhalten durch dessen Anwendung:

$$\begin{aligned}
& \mathbb{E} \int |\hat{m}_n(x) - m(x)|^2 \mathbb{P}_X(dx) \\
& \leq \frac{c_8 \cdot \log(n)^2 \cdot \left(\log \left(\sup_{x_1^n \in (\text{supp}(X))^n} \mathcal{N}_1 \left(\frac{1}{n \cdot \beta_n}, \mathcal{F}^{(J_n)}, x_1^n \right) \right) + 1 \right)}{n} \\
& \quad + 2 \cdot \mathbb{E} \left(\min_{l \in \Theta_n} \int |g_{n,l}(x) - m(x)|^2 \mathbb{P}_X(dx) + c_8 \cdot \frac{(M_n + 1)^d}{n} \right) \\
& = \frac{c_8 \cdot \log(n)^2 \cdot \left(\log \left(\sup_{x_1^n \in (\text{supp}(X))^n} \mathcal{N}_1 \left(\frac{1}{n \cdot \beta_n}, \mathcal{F}^{(J_n)}, x_1^n \right) \right) + 1 \right)}{n} \\
& \quad + 2 \int |g_n(x) - m(x)|^2 \mathbb{P}_X(dx) + 2 \cdot c_8 \cdot \frac{(M_n + 1)^d}{n},
\end{aligned}$$

wobei wir bei der letzten Gleichheit verwendet haben, dass der letzte Summand deterministisch ist. Zudem wissen wir, dass c_8 unabhängig von n ist und $n > 1$, da wir n hinreichend

groß wählen. Als nächstes überprüfen wir die Voraussetzungen von Lemma 3.4, um damit dann die letzte Gleichung mit der Überdeckungszahl \mathcal{N}_1 weiter abzuschätzen. Nach Voraussetzung ist $\beta_n = c_3 \cdot \log(n)$ und $a_n = (\log n)^{1/(6(N+d))} > 0$ für hinreichend großes n . Nach Voraussetzung sind zudem $d, N, J_n \in \mathbb{N}$ und es gilt nach Gleichung (3.2):

$$J_n \leq (M_n + 1)^d \cdot (N + 1)^d \leq n^c,$$

für hinreichend großes n und Konstante $c > 0$. Wir betrachten hier den logistischen squash σ welcher nach Lemma 1.10 insbesondere 2-zulässig ist. Da die hier betrachtete Menge von Funktionen $\mathcal{F}^{(J_n)}$ identisch mit der aus Lemma 3.4 ist, sind nun alle Voraussetzungen für Lemma 3.4 erfüllt. Nach Voraussetzung wissen wir, dass $\text{supp}(\mathbb{P}_X)$ beschränkt ist, und wir können unsern n so groß wählen, dass wir ohne Beschränkung der Allgemeinheit annehmen können, dass $\text{supp}(\mathbb{P}_X) = \{x \in \mathbb{R}^d \mid \forall \varepsilon > 0 : \mathbb{P}_X(S_\varepsilon(x)) > 0\} \subseteq [-a_n, a_n]^d$ ist, mit S_ε als ε -Umgebung um $x \in \mathbb{R}^d$. In der nächsten Ungleichung bezeichnen wir die Konstanten die sich von c_8 unterscheiden und die unabhängig von n sind, zu einer Konstante c zusammen. Wir erhalten damit durch Lemma 3.4 für hinreichend großes n :

$$\begin{aligned} & \frac{c_8 \cdot \log(n)^2 \cdot \left(\log \left(\sup_{x_1^n \in (\text{supp}(X))^n} \mathcal{N}_1 \left(\frac{1}{n \cdot \beta_n}, \mathcal{F}^{(J_n)}, x_1^n \right) \right) + 1 \right)}{n} \\ & \leq \frac{c_8 \cdot \log(n)^2 \cdot \left((c \cdot \log(n) \cdot (M_n + 1)^d \cdot (N + 1)^d) + 1 \right)}{n} \\ & \leq \frac{c_8 \cdot \log(n)^2 \cdot (2 \cdot c \cdot \log(n) \cdot (M_n + 1)^d \cdot (N + 1)^d)}{n} \\ & \leq c_9 \cdot \frac{\log(n)^3 \cdot (M_n + 1)^d \cdot (N + 1)^d}{n}, \end{aligned} \tag{3.12}$$

wobei $c_9 := c_8 \cdot c$ eine von n unabhängige Konstante ist. Sei

$$P_n(x) = \sum_{\mathbf{i} \in [M_n]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(x_{\mathbf{i}}) \cdot (x - x_{\mathbf{i}})^{\mathbf{j}} \prod_{j=1}^d \left(1 - \frac{M_n}{2a_n} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+.$$

Mit Ungleichung (3.6), zusammen mit einer Nulladdition und der Linearität des Integrals erhalten wir:

$$\begin{aligned} & \int |g_n(x) - m(x)|^2 \mathbb{P}_X(dx) \\ & = \int |g_n(x) - P_n(x) + P_n(x) - m(x)|^2 \mathbb{P}_X(dx) \\ & \leq \int 2|g_n(x) - P_n(x)|^2 + 2|P_n(x) - m(x)|^2 \mathbb{P}_X(dx). \end{aligned}$$

Aus der Supremumseigenschaft folgt

$$\begin{aligned}
& \int |g_n(x) - m(x)|^2 \mathbb{P}_X(dx) \\
& \leq 2 \int \sup_{x \in [-a_n, a_n]^d} |g_n(x) - P_n(x)|^2 \mathbb{P}_X(dx) + 2 \int \sup_{x \in [-a_n, a_n]^d} |P_n(x) - m(x)|^2 \mathbb{P}_X(dx) \quad (3.13) \\
& = 2 \sup_{x \in [-a_n, a_n]^d} |g_n(x) - P_n(x)|^2 + 2 \sup_{x \in [-a_n, a_n]^d} |P_n(x) - m(x)|^2,
\end{aligned}$$

wobei wir im letzten Schritt $\text{supp}(\mathbb{P}_X) \subseteq [-a_n, a_n]^d$ und $\mathbb{P}(X \in \text{supp}(\mathbb{P}_X)) = 1$ verwendet haben. Um die letzten beiden Summanden der Ungleichung (3.13) weiterhin abzuschätzen möchten wir Lemma 3.2 anwenden. Dafür überprüfen wir, ob dafür alle Voraussetzungen erfüllt sind. Wir betrachten wieder den logistischen squasher σ aus Gleichung 1.2, welcher nach Lemma 1.10 insbesondere 2-zulässig ist. Zudem ist für hinreichend großes n die Bedingung

$$\begin{aligned}
R_n \geq \max \left\{ \frac{\|\sigma''\|_\infty \cdot (M_n + 1)}{2 \cdot |\sigma'(t_\sigma)|}, \frac{9 \cdot \|\sigma''\|_\infty \cdot a_n}{|\sigma'(t_\sigma)|}, \right. \\
\left. \frac{20 \cdot \|\sigma'''\|_\infty}{3 \cdot |\sigma''(t_\sigma)|} \cdot 3^{3 \cdot 3^s} \cdot a^{3 \cdot 2^s}, 1792 \cdot \frac{\max\{\|\sigma''\|_\infty, \|\sigma'''\|_\infty, 1\}}{\min\{2 \cdot |\sigma'(t_\sigma)|, |\sigma''(t_\sigma)|, 1\}} \cdot M_n^3 \right\}
\end{aligned}$$

erfüllt und da unser neuronales Netz (2.15) mit $x_i \in [-a_n, a_n]^d$ identisch mit der Definition aus Lemma 3.2 ist, sind alle Voraussetzungen für Lemma 1.10 erfüllt. Wir erhalten damit für $x \in [-a_n, a_n]^d$ und n hinreichend groß:

$$\begin{aligned}
& |g_n(x) - P_n(x)| \\
& = \left| \sum_{\mathbf{i} \in [M_n]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(x_{\mathbf{i}}) \right| \cdot \left| f_{\text{net}, \mathbf{j}, \mathbf{i}}(x) - (x - x_{\mathbf{i}})^{\mathbf{j}} \cdot \prod_{j=1}^d \left(1 - \frac{M_n}{2a_n} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|\right)_+ \right| \\
& \stackrel{(3.1)}{\leq} (M_n + 1)^d \cdot (q + 1)^d \cdot z \cdot \left| f_{\text{net}, \mathbf{j}, \mathbf{i}}(x) - (x - x_{\mathbf{i}})^{\mathbf{j}} \cdot \prod_{j=1}^d \left(1 - \frac{M_n}{2a_n} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|\right)_+ \right| \quad (3.14) \\
& \leq (M_n + 1)^d \cdot (q + 1)^d \cdot z \cdot c \cdot 3^{3 \cdot 3^s} \cdot a_n^{3 \cdot 2^s} \cdot M_n^3 \cdot \frac{1}{R_n} \\
& \leq (M_n + 1)^d \cdot (q + 1)^d \cdot c_{10} \cdot a_n^{3 \cdot (N+d) \cdot 2} \cdot \frac{M_n^3}{R_n} \\
& = (M_n + 1)^d \cdot (q + 1)^d \cdot c_{10} \cdot \log(n) \cdot \frac{M_n^3}{R_n},
\end{aligned}$$

wobei c eine von n unabhängige Konstante ist, welche wir aus Lemma 1.10 erhalten und $c_{10} := z \cdot c$ damit ebenfalls von n unabhängig ist. Wie haben unter anderem verwendet, dass für hinreichend großes n :

$$a_n^{2^{\lceil \log_2(N+d) \rceil}} \leq a_n^{2^{\log_2(N+d)+1}} = a_n^{(N+d) \cdot 2},$$

gilt. Im letzten Schritt haben wir in Ungleichung (3.14) die Definition von a_n eingesetzt. Da nach Konstruktion $a > 0$, m (p, C) -glatt und $P_n(x)$ nach Lemma 2.1 eine Spline Interpolation von Taylorpolynomen von m ist, erhalten wir mit Lemma 2.2:

$$|P_n(x) - m(x)| \leq c_{11} \cdot \frac{a_n^p}{M_n^p} \leq c_{11} \cdot \log(n) \cdot \frac{1}{M_n^p}. \quad (3.15)$$

In dieser Ungleichung ist c_{11} eine von n unabhängige Konstante und wir haben zudem verwendet, dass:

$$a_n^p = a_n^{q+s} \leq a_n^{N+d} \leq a_n^{6 \cdot (N+d)} = \log(n),$$

für $p = q + s$ für hinreichend großes n gilt, da nach Voraussetzung $N \geq q$ und $d \geq s$ mit $s \in (0, 1]$ ist. Durch Quadrieren bleiben die Ungleichungen Ungleichung (3.14) und (3.15) auch erhalten und da in beiden Ungleichungen die rechte Seite unabhängig von x ist, gelten die Ungleichungen ebenfalls für das Supremum. Durch einsetzen der Definitionen von M_n und R_n erhalten wir für n hinreichend groß:

$$\begin{aligned} \sup_{x \in [-a_n, a_n]^d} |g_n(x) - P_n(x)|^2 &\leq \left((M_n + 1)^d \cdot (q + 1)^d \cdot c_{10} \cdot \log(n) \cdot \frac{M_n^3}{R_n} \right)^2 \\ &\leq c_{10}^2 \cdot (M_n + 1)^{2d} \cdot \log(n)^2 \cdot \frac{M_n^6}{R_n^2} \\ &\leq c_{10}^2 \cdot (M_n + 1)^{2d} \cdot \log(n)^2 \cdot \frac{(M_n + 1)^{6d}}{R_n^2} \\ &\leq c_{10}^2 \cdot \log(n)^2 \cdot \frac{(M_n + 1)^{8d}}{R_n^2} \\ &\leq c_{10}^2 \cdot \frac{n^{\frac{8d}{2p+d}}}{n^{2d+8}} \cdot \log(n)^2 \\ &= c_{10}^2 \cdot n^{\frac{8d}{2p+d} - 2d - 8} \cdot \log(n)^2 \\ &\leq c_{10}^2 \cdot n^{\frac{8d}{2p+d} - 8 \frac{2p+d}{2p+d}} \cdot \log(n)^2 \\ &= c_{10}^2 \cdot n^{-\frac{16p}{2p+d}} \cdot \log(n)^2 \\ &\leq c_{10}^2 \cdot n^{-\frac{2p}{2p+d}} \cdot \log(n)^3, \end{aligned} \quad (3.16)$$

wobei wir bei der letzten Ungleichung verwendet haben, dass $\frac{16p}{2p+d} > \frac{2p}{2p+d}$, da $p > 0$ ist und $\log(n)^2 < \log(n)^3$ für n hinreichend groß. Ebenfalls erhalten wir:

$$\begin{aligned} \sup_{x \in [-a_n, a_n]^d} |P_n(x) - m(x)|^2 &\leq \left(c_{11} \cdot \log(n) \cdot \frac{1}{M_n^p} \right)^2 \\ &\leq c_{11}^2 \cdot \log(n)^2 \cdot c_2^{-2p} \cdot n^{-\frac{2p}{2p+d}} \\ &\leq (c_{11}^2 \cdot c_2^{-2p}) \cdot \log(n)^3 \cdot n^{-\frac{2p}{2p+d}}. \end{aligned} \quad (3.17)$$

Mit analogem Vorgehen erhalten wir für (3.12):

$$\begin{aligned}
& \frac{c_8 \cdot \log(n)^2 \cdot \left(\log \left(\sup_{x_1^n \in (\text{supp}(X))^n} \mathcal{N}_1 \left(\frac{1}{n \cdot \beta_n}, \mathcal{F}^{(J_n)}, x_1^n \right) \right) + 1 \right)}{n} \\
& \leq c_9 \cdot \frac{\log(n)^3 \cdot (M_n + 1)^d \cdot (N + 1)^d}{n} \\
& \leq c_9 \cdot (N + 1)^d \cdot \log(n)^3 \cdot \frac{(c_2 \cdot n^{\frac{1}{2p+d}} + 2)^d}{n} \\
& \leq c_9 \cdot (N + 1)^d \cdot 2^d \cdot \log(n)^3 \cdot \frac{c_2^d \cdot n^{\frac{d}{2p+d}}}{n} \\
& = c_9 \cdot (N + 1)^d \cdot 2^d \cdot c_2^d \cdot \log(n)^3 \cdot n^{\frac{d}{2p+d} - 1} \\
& = c_{12} \cdot \log(n)^3 \cdot n^{-\frac{2p}{2p+d}},
\end{aligned} \tag{3.18}$$

mit einer von n unabhängigen Konstante $c_{12} := c_9 \cdot (N + 1)^d \cdot 2^d \cdot c_2^d > 0$. Zudem erhalten wir, da für n hinreichend groß $\log(n)^3 > 1$ gilt, mit analogem Vorgehen:

$$\begin{aligned}
c_8 \cdot \frac{(M_n + 1)^d}{n} & \leq c_{13} \cdot \frac{n^{\frac{d}{2p+d}}}{n} \\
& \leq c_{13} \cdot \log(n)^3 \cdot n^{-\frac{2p}{2p+d}}
\end{aligned} \tag{3.19}$$

und

$$\begin{aligned}
\frac{4 \cdot c_4 \cdot \beta_n^2}{n} & \leq c_{14} \cdot \log(n)^3 \cdot n^{-1} \\
& \leq c_{14} \cdot \log(n)^3 \cdot n^{-\frac{2p}{2p+d}},
\end{aligned} \tag{3.20}$$

mit von n unabhängigen Konstanten $c_{13} := c_8 \cdot 2^d > 0$ und $c_{14} := 4 \cdot c_4 \cdot c_3^2 > 0$. Nun haben wir alle Summanden von Ungleichung (3.9) abgeschätzt und erhalten schließlich mit Ungleichungen (3.14) - (3.20):

$$\mathbb{E} \int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \leq c_{\text{fin}} \cdot \log(n)^3 \cdot n^{-\frac{2p}{2p+d}},$$

mit

$$c_{\text{fin}} = c_{14} + 2 \cdot c_{13} + c_{12} + 2 \cdot (2 \cdot c_{11}^2 \cdot c_2^{-2p} + 2 \cdot c_{10}^2),$$

wobei c_{fin} als Summe nichtnegativer oder positiver Konstanten, die unabhängig von n sind, nichtnegativ und unabhängig von n ist. Damit haben wir unser Hauptresultat bewiesen. \square

Kapitel 4

Anwendungsbeispiel auf simulierte Daten

In diesem Kapitel betrachten wir die Leistung des hier vorgestellten Neuronale-Netze-Regressionsschätzers bei endlicher Stichprobengröße auf simulierte Daten in *Python*.

Die simulierten Daten welchen wir verwenden werden, sehen wie gefolgt aus: Wir wählen X gleichverteilt auf $[-2, 2]^d$, wobei d die Dimension des Inputs ist, zudem wählen wir ε als standardnormalverteilt und unabhängig von X und wir definieren Y durch:

$$Y = m_j(X) + \sigma \cdot \lambda_j \cdot \varepsilon,$$

mit $m_j: [-2, 2]^d \rightarrow \mathbb{R}$ ($j \in \{1, 2\}$) wie unten definiert, $\lambda_j > 0$ als Skalierungsfaktor welcher wie unten definiert wird und einen Rauschfaktor $\sigma = 0.05$. Als Regressionsfunktionen verwenden wir die Funktionen:

$$m_1(x) = \sin(0.2 \cdot x^2) + \exp(0.5 \cdot x) + x^3$$

und

$$m_2(x_0, x_1) = \sin(\sqrt[2]{x_0^2 + x_1^2}).$$

Wir wählen λ_j als Interquartilsabstand einer Stichprobe von $m(X)$ der Größe $N = 8000$. Mit diesen Daten lässt sich nun auch Y darstellen.

Um die Leistung unseres neuronalen Netze Regressionsschätzers zu überprüfen, haben wir erstmals m_1 und m_2 und die jeweilige Schätzung durch unseren Neuronale-Netze-Regressionsschätzer für $\sigma = 0.05$ zeichnen lassen. Man erkennt dass der Schätzer eine sehr gute Approximation der Funktion liefert, aber um genauer beurteilen zu können wie gut die Schätzung wirklich ist und wie gut unser Schätzer im Vergleich zu anderen Schätzern abschneidet betrachten wir in Tabelle ... den Interquartilsabstand und den Median der skalierten L_2 -Fehler der einzelnen Schätzer.

Unser vorgehen zum Vergleich der drei hier betrachteten Regressionsschätzern gestaltet sich wie gefolgt: Wir bestimmen erst den L_2 -Fehler der einzelnen Schätzer approximativ durch den empirisch arithmetischen L_2 -Fehler $\varepsilon_{L_2,N}$ auf einer unabhängigen Stichprobe von X der Größe $N = 10000$.

Da wir unsere Regressionsfunktionen kennen und der Fehler stark vom Verhalten der korrekten Funktion von m_j abhängt, betrachten wir den empirischen L_2 -Fehler im Verhältnis zum einfachsten Schätzer von m_j , einer konstanten Funktion. Der Wert dieses *konstanten* Schätzers bestimmen wir in dem wir das empirische Mittel der beobachtete Daten Y nehmen. Wir erhalten damit einen skaliertes Fehlermaß $\varepsilon_{L_2,N}(m_{n,i})/\bar{\varepsilon}_{L_2,N}(avg)$ mit $\bar{\varepsilon}_{L_2,N}(avg)$ als Median von 50 unabhängigen Realisierungen von $\varepsilon_{L_2,N}$.

Wir bestimmten $\varepsilon_{L_2,N}(\cdot)$ als empirisches Mittel von 25 quadratischen Fehlern der Schätzung des konstanten Schätzers. Dieses skalierte Fehlermaß ist so zu deuten, dass ein großer Fehler durch einen der drei Regressionsschätzer im Falle dass der Fehler des konstanten Schätzers klein ist, auf eine noch schlechtere Performance hindeutet. Der Fehler $\varepsilon_{L_2,N}(m_{n,i})$ wird also durch $\bar{\varepsilon}_{L_2,N}(avg)$ gewichtet.

Die resultierenden skalierten Fehler hängen noch von der Stichprobe von (X, Y) ab und um diese Werte besser vergleichen zu können, führen wir die Fehlerberechnung jeweils 50 mal durch und geben dann den Median und Interquartilsabstand für die Schätzung der betrachteten Regressionsschätzer aus. Wir teilen für jeden Schätzer die Stichprobe auf in ein *learning sample* der Größe $n_l = 0.8 \cdot N$ und in ein *testing sample* der Größe $n_t = 0.2 \cdot N$. Wir bestimmen den Schätzer für alle Parameterwerte mit dem learning sample und bestimmen das korrespondierende L_2 -Risiko auf dem testing sample und wählen dann die Parameter die zu einem minimalen empirischen L_2 -Risiko auf dem testing sample führen. Unser erster Schätzer *fc_neural_1_estimate* ist ein fully connected neuronales Netz mit einer verborgenen Schicht. Dieser Schätzer hat eine feste Anzahl an Neuronen die wir aus der Menge $\{5, 10, 25, 50, 75\}$ auswählen die bei der Simulation zu einem minimalen empirischen L_2 -Risiko führt. Unser zweiter Schätzer *nearest_neighbor_estimate* ist ein Nächste-Nachbar Schätzer bei der die Anzahl an nächsten Nachbarn aus der Menge $\{1, 2, 3\} \cup \{4, 8, 12, 16, \dots, 4 \cdot \lfloor \frac{n_l}{4} \rfloor\}$ ausgewählt wird. Unser letzter Schätzer ist der hier vorgestellte Neuronale-Netze-Regressionsschätzer den wir mit *new_neural_network_estimate* bezeichnen. Hier haben wir die Parameter je nachdem welche Regressionsfunktion wir betrachtet haben entsprechend angepasst. Da wir zum Beispiel den Grad der zu schätzenden Funktion kennen und dies bei unserem Schätzer berücksichtigen können.

Wie wir in den Tabellen anhand des skalierten L_2 -Fehlers sehen können, übertrifft unserer Neuronale-Netze-Schätzer in allen Fällen die Leistung der anderen Schätzer.

	m_1		m_2	
<i>noise</i>	5%	10%	5%	10%
$\bar{\epsilon}_{L_2,N}(avg)$				
<i>approach</i>	Median (IQR)	Median (IQR)	Median (IQR)	Median (IQR)
new_neural_network_estimate	Afghanistan	AF	AFG	004
fc_neural_1_estimate	Aland	AX	ALA	248
nearest_neighbor_estimate	Albania	AL	ALB	008

Tabelle 4.1: Truth Tables and Accuracy Measures for each modeling library.

Literaturverzeichnis

- [AB19] A. BRAUN, M. KOHLER UND A. KRZYZAK: *Analysis of the rate of convergence of neural network regression estimates which are easy to implement*. 2019.
- [AHM09] ALBRECHER H., BINDER, A. und P. MAYER: *Einführung in die Finanzmathematik*. Birkhäuser, Basel, 2009.
- [Car96] CARRIERE, J.: *Valuation of the early-exercise price for options using simulations and nonparametric regression*. Insurance: mathematics and Economics, 19(1):19–30, 1996.
- [EKT⁺07] EGLOFF, D., M. KOHLER, N. TODOROVIC et al.: *A dynamic look-ahead Monte Carlo algorithm for pricing Bermudan options*. The Annals of Applied Probability, 17(4):1138–1171, 2007.
- [For16] FORSTER, O.: *Analysis I - Differential- und Integralrechnung einer Veränderlichen*. Springer, Berlin, 2016.
- [GKW02] GYÖRFI, L. KOHLER, M., A. KRZYZAK und H. WALK: *A Distribution-Free Theory of Nonparametric Regression*. Springer series in statistics. Springer, Berlin, 2002.
- [Gla13] GLASSERMAN, P.: *Monte Carlo methods in financial engineering*, Band 53 der Reihe *Stochastic Modelling and Applied Probability*. Springer, New York, 2013.
- [Hil13] HILDEBRANDT, S.: *Analysis 2*. Springer, Berlin, 2013.
- [Kle13] KLENKE, A.: *Wahrscheinlichkeitstheorie*. Springer, Berlin, 2013.
- [Koh10] KOHLER, M.: *A review on regression-based Monte Carlo methods for pricing American options*. In: DEVROYE L., KARASÖZEN, B. KOHLER M. und

- R. KORN (Herausgeber): *Recent Developments in Applied Probability and Statistics*, Seiten 37–58. Physica, Heidelberg, 2010.
- [KS98] KARATZAS, I. und S. SHREVE: *Methods of mathematical finance*, Band 39 der Reihe *Stochastic Modelling and Applied Probability*. Springer, New York, 1998.
- [Kö13] KÖNIGSBERGER, K.: *Analysis 2*. Springer, Berlin, 2013.
- [LS01] LONGSTAFF, F. und E. SCHWARTZ: *Valuing American options by simulation: a simple least-squares approach*. *The review of financial studies*, 14(1):113–147, 2001.
- [Mal00] MALKIEL, B.: *Börsenerfolg ist kein Zufall - die besten Investmentstrategien für das neue Jahrtausend*. FinanzBuch, München, 2000.
- [R D17] R DEVELOPMENT CORE TEAM: *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, 2017.
- [SB07] STOER, J. und R. BULIRSCH: *Numerische Mathematik 1*. Springer, Berlin, 2007.
- [Sto18] STORCH, U. UND WIEBE, H: *Grundkonzepte der Mathematik - Mengentheoretische, algebraische, topologische Grundlagen sowie reelle und komplexe Zahlen*. Springer, Berlin, 2018.
- [TVR99] TSITSIKLIS, J. N. und B. VAN ROY: *Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives*. *IEEE Transactions on Automatic Control*, 44(10):1840–1851, 1999.

Appendix

Der Programmcode ist wie folgt aufgebaut:

- `main.py` ist das Hauptprogramm welches alle Schätzer aufruft und die Outputs generiert.
- `data_gen.py` generiert die Daten die wir für unsere Simulation benötigen.
- `help_neural_networks.py` fasst alle Hilfsfunktion zusammen.
- `new_neural_network.py` enthält unseren Neuronale-Netze-Regressionsschätzer.
- `fc_neural_network.py` enthält das fully connected neuronale Netz mit einer verborgenen Schicht.
- `nearest_neighbor.py` enthält einen Nächste-Nachbar Schätzer.
- `constant.py` enthält den konstanten Schätzer.

Listing 4.1: `main.py`

```

1
2 """
3 @author: adrian
4
5 Main Datei die die Simulation und damit den Vergleich der implementierten Schätzer
6 durchführt.
7 """
8 import numpy as np
9 #from mpl_toolkits import mplot3d
10 #import matplotlib . pyplot as plt
11 import pandas as pd
12 #import tikzplotlib
13 from scipy.stats import iqr
14 from data_gen import gen_data_Y
15 from constant import constant_estimate
16 from new_neural_network import new_neural_network_estimate
17 from nearest_neighbor import nearest_neighbor_estimate
18 from fc_neural_network import fc_neural_1_estimate
19

```

```

20 n = 10000
21 n_train = int(n * 0.8)
22 n_test = int(n * 0.2)
23
24 '''
25 #EINDIMENSIONALER FALL (d = 1) wird geplottet
26 '''
27 #
28 #N = 3
29 #q = 2
30 #R = 10 ** 6
31 #a = 2
32 #M = 2
33 #d = 1
34 #
35 #sigma = 0.05
36 #
37 ## Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
38 #X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
39 #m_X_train, Y_train = gen_data_Y(X_train, sigma)
40 #
41 #X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))
42 #
43 #Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a
    ,)
44 ##Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train, Y_train, X_test)
45 ##Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train, Y_train, X_test)
46 #m_X_test, dummy = gen_data_Y(X_test, sigma)
47 #
48 #
49 ##plt.plot(X_test, Y_pred_nearest_neighbor, '-r', label='nearest_neighbor')
50 ##plt.plot(X_test, Y_pred_fc_nn_1, '-g', label='fc_nn_1')
51 ##colors = (0,0,0)
52 #area = 4
53 #plt.scatter(X_test, m_X_test, s=area, color = 'blue', label='m_1', alpha=0.5)
54 #plt.scatter(X_test, Y_pred_new_nn, s=area, color = 'red', label='new_neural_network_
    estimate', alpha=0.5)
55 #plt.title('...')
56 #plt.legend(loc='upper left')
57 #plt.xlabel('x')
58 #plt.ylabel('y')
59 ##plt.savefig('graph_d_1.png')
60 #tikzplotlib.save("mytikz_dl.tex")
61 ##plt.show()
62 ##plt.plot(X_test, Y_pred_new_nn, 'ro-', label='new_nn')
63 ##plt.plot(X_test, m_X_test, '-b', label='m_d')
64 #
65 ##plt.xlim(-2, 2)
66 ##plt.xlim(-2,2)
67 ##plt.show()
68 #
69 #
70 '''
71 #ZEIDIMENSIONALER FALL (d = 2) wird geplottet
72 '''

```

```

73 #N = 2
74 #q = 2
75 #R = 10 ** 6
76 #a = 2
77 #M = 2
78 #d = 2
79 #
80 #sigma = 0.05
81 #
82 #
83 ## Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
84 #X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
85 #m_X_train , Y_train = gen_data_Y(X_train , sigma)
86 #
87 #X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))
88 #
89 #Y_pred_new_nn = new_neural_network_estimate(X_train , Y_train , X_test , N, q, R, d, M, a
    ,)
90 ##Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train , Y_train , X_test)
91 ##Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train , Y_train , X_test)
92 #m_X_test , dummy = gen_data_Y(X_test , sigma)
93 #
94 #
95 #
96 #x = np.ravel(X_test[:,0])
97 #y = np.ravel(X_test[:,1])
98 #
99 ## so wie es sein soll
100 ##z = m_X_test[:,0]
101 ## was der Schätzer auswirft
102 #z_new = Y_pred_new_nn[:,0]
103 #
104 #ax = plt.axes(projection='3d')
105 #ax.scatter(x, y, z_new, c=z_new, cmap='viridis', linewidth=0.5);
106 #ax.view_init(40, 20)
107 #plt.savefig('graph_d_2_new_estimate.png')
108 #
109 ## so wie es sein soll
110 #z = m_X_test[:,0]
111 ## was der Schätzer auswirft
112 #
113 #ax = plt.axes(projection='3d')
114 #ax.scatter(x, y, z, c=z, cmap='viridis', linewidth=0.5);
115 #ax.view_init(40, 20)
116 #plt.savefig('test.png')
117 ##tikzplotlib.save("mytikz_d2.tex")
118 #
119 #postpro = np.asarray([ X_test[:,0], X_test[:,1], Y_pred_new_nn[:,0] ])
120 #np.savetxt("plotpostpro.csv", np.transpose(postpro), delimiter=",")
121 #
122 ##plt.savefig('graph_d_2_m_2.png')
123
124 '''
125 ein Vergleich des emp. L2 Fehler gemacht für d = 1
126 '''

```

```

127 #Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
128
129 N = 3
130 q = 2
131 R = 10 ** 6
132 a = 2
133 M = 2
134 d = 1
135
136 spreads = [0.05, 0.1]
137
138 scaled_error = np.empty((50, 3,))
139 scaled_error[:] = np.nan
140
141 e_L2_avg = np.zeros(25)
142 e_L2_avg[:] = np.nan
143
144 for sigma in spreads:
145     for i in range(0, np.size(scaled_error, 0), 1):
146
147         X_train = np.random.uniform(low=-2, high=2, size=(int(n_train), d))
148         m_X_train, Y_train = gen_data_Y(X_train, sigma)
149
150         X_test = np.random.uniform(low=-2, high=2, size=(int(n_test), d))
151
152         #Y_pred_constant = constant_estimate(Y_train)
153         Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d
154             , M, a,)
155         Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train, Y_train, X_test)
156         Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train, Y_train, X_test)
157
158         m_X_test, not_needed = gen_data_Y(X_test, sigma)
159
160         e_L2_new_nn = np.mean(abs(Y_pred_new_nn - m_X_test) ** 2)
161         e_L2_fc_nn_1 = np.mean(abs(Y_pred_fc_nn_1 - m_X_test) ** 2)
162         e_L2_nearest_neighbor = np.mean(abs(Y_pred_nearest_neighbor - m_X_test) ** 2)
163
164         for j in range(0, np.size(e_L2_avg), 1):
165
166             X = np.random.uniform(low=-2, high=2, size=(n_test, d))
167             m_X, Y = gen_data_Y(X, sigma)
168             Y_pred_constant = constant_estimate(Y)
169
170             e_L2_avg[j] = np.mean(abs(Y_pred_constant - m_X) ** 2)
171
172             scaled_error[i, 0] = e_L2_new_nn / np.median(e_L2_avg)
173             scaled_error[i, 1] = e_L2_fc_nn_1 / np.median(e_L2_avg)
174             scaled_error[i, 2] = e_L2_nearest_neighbor / np.median(e_L2_avg)
175
176         iqr_new_nn = iqr(scaled_error[:, 0])
177         iqr_fc_nn_1 = iqr(scaled_error[:, 1])
178         iqr_nearest_neighbor = iqr(scaled_error[:, 2])
179
180         median_new_nn = np.median(scaled_error[:, 0])
181         median_fc_nn_1 = np.median(scaled_error[:, 1])

```



```

181     median_nearest_neighbor = np.median(scaled_error[:,2])
182
183     rows = ["noise","e_L2_avg","approach","new_nn", "fc_nn_1", "nearest_neighbor"]
184
185     if sigma == 0.05:
186         series_noise_1 = pd.Series([repr(sigma)+'%',np.median(e_L2_avg),"(Median, IQR)"
187                                     ,(median_new_nn, iqr_new_nn), (median_fc_nn_1, iqr_fc_nn_1), (median_nearest
188                                     _neighbor, iqr_nearest_neighbor)], index=rows)
187         series_noise_1.name = ""
188         print("Der empirische L2 Fehler für d = 1 und sigma = 0.05 ist berechnet worden!"
189               ")")
189
190     else:
191         series_noise_2 = pd.Series([repr(sigma)+'%',np.median(e_L2_avg),"(Median, IQR)"
192                                     ,(median_new_nn, iqr_new_nn), (median_fc_nn_1, iqr_fc_nn_1), (median_nearest
193                                     _neighbor, iqr_nearest_neighbor)], index=rows)
192         series_noise_2.name = ""
193         print("Der empirische L2 Fehler für d = 1 und sigma = 0.1 ist berechnet worden!"
194               ")")
194
195
196     error_df = pd.concat([series_noise_1, series_noise_2], axis=1)
197     #print(error_df)
198     error_df.to_csv('out_d_1.csv',index = True)
199
200     '''
201     ein Vergleich des emp. L2 Fehler gemacht für d = 2
202     '''
203     # Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
204
205     N = 2
206     q = 2
207     R = 10 ** 6
208     a = 2
209     M = 2
210     d = 2
211
212     spreads = [0.05,0.1]
213
214     scaled_error = np.empty((50, 3,))
215     scaled_error[:,] = np.nan
216
217     e_L2_avg = np.zeros(25)
218     e_L2_avg[:,] = np.nan
219
220     for sigma in spreads:
221         for i in range(0,np.size(scaled_error,0),1):
222
223             X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
224             m_X_train, Y_train = gen_data_Y(X_train,sigma)
225
226             X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))
227
228             #Y_pred_constant = constant_estimate(Y_train)
229             Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d

```

```

    , M, a,)
230 Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train , Y_train , X_test)
231 Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train , Y_train , X_test)
232
233 m_X_test , not_needed = gen_data_Y(X_test , sigma)
234
235 e_L2_new_nn = np.mean(abs(Y_pred_new_nn - m_X_test) ** 2)
236 e_L2_fc_nn_1 = np.mean(abs(Y_pred_fc_nn_1 - m_X_test) ** 2)
237 e_L2_nearest_neighbor = np.mean(abs(Y_pred_nearest_neighbor - m_X_test) ** 2)
238
239 for j in range(0,np.size(e_L2_avg),1):
240
241     X = np.random.uniform(low=-2,high=2,size=(n_test,d))
242     m_X, Y = gen_data_Y(X, sigma)
243     Y_pred_constant = constant_estimate(Y)
244
245     e_L2_avg[j] = np.mean(abs(Y_pred_constant - m_X) ** 2)
246
247     scaled_error[i,0] = e_L2_new_nn / np.median(e_L2_avg)
248     scaled_error[i,1] = e_L2_fc_nn_1 / np.median(e_L2_avg)
249     scaled_error[i,2] = e_L2_nearest_neighbor / np.median(e_L2_avg)
250
251 iqr_new_nn = iqr(scaled_error[:,0])
252 iqr_fc_nn_1 = iqr(scaled_error[:,1])
253 iqr_nearest_neighbor = iqr(scaled_error[:,2])
254
255 median_new_nn = np.median(scaled_error[:,0])
256 median_fc_nn_1 = np.median(scaled_error[:,1])
257 median_nearest_neighbor = np.median(scaled_error[:,2])
258
259 rows = [ "noise", "e_L2_avg", "approach", "new_nn", "fc_nn_1", "nearest_neighbor" ]
260
261 if sigma == 0.05:
262     series_noise_1 = pd.Series([repr(sigma)+'%', np.median(e_L2_avg), "(Median, IQR)"
263                               ,(median_new_nn, iqr_new_nn), (median_fc_nn_1, iqr_fc_nn_1), (median_nearest
264                               _neighbor, iqr_nearest_neighbor)], index=rows)
265     series_noise_1.name = ""
266     print("Der empirische L2 Fehler für d = 2 und sigma = 0.05 ist berechnet worden!"
267           ")
268
269 else:
270     series_noise_2 = pd.Series([repr(sigma)+'%', np.median(e_L2_avg), "(Median, IQR)"
271                               ,(median_new_nn, iqr_new_nn), (median_fc_nn_1, iqr_fc_nn_1), (median_nearest
272                               _neighbor, iqr_nearest_neighbor)], index=rows)
273     series_noise_2.name = ""
274     print("Der empirische L2 Fehler für d = 2 und sigma = 0.1 ist berechnet worden!"
275           ")
276
277 error_df = pd.concat([series_noise_1, series_noise_2], axis=1)
278 #print(error_df)
279 error_df.to_csv('out_d_2.csv', index = True)

```

Listing 4.2: data_gen.py

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-

```

```

3 """
4 Created on Fri Oct 11 12:01:42 2019
5
6 @author: adrian
7
8 Generieren der Daten die wir für einen Vergleich von Regressionsschätzern benötigen
9 """
10 # Wir wählen x gleichverteilt auf  $[-2,2]^d$ , wobei d die dimension des Inputs ist
11 # n is die Größe der Stichprobe
12
13 import numpy as np
14 from scipy.stats import iqr
15
16 # Regressionsfunktionen
17 #
18 # x: Ein Vektor x der Dimension d
19 # d: Dimension des Vektors x
20
21 def m_d (x, d):
22
23     pi = np.pi
24     cos = np.cos
25     sin = np.sin
26     exp = np.exp
27
28     if d == 1:
29         return sin(0.2 * x[0] ** 2) + exp(0.5 * x[0]) + x[0] ** 3
30
31     elif d == 2:
32         return np.sin(np.sqrt(x[0] ** 2 + x[1] ** 2))
33
34     else:
35         print("Your data has the wrong dimension!")
36
37 def error_limit (x, p, c, d):
38
39     return c * (np.log(x) ** 3) * (x ** (-(2 * p)/(2 * p + d)))
40
41 # Generiert den Vektor Y_1,...,Y_n für den Datensatz (X_1,Y_1),...,(X_n,Y_n)
42 #
43 # X: Inputdaten der Form (X_1,...,X_n), wobei  $X_i \in \mathbb{R}^d$  für  $i = 1, \dots, n$ 
44 # sigma: Schwankung in den Werten (Noise)  $\in \{0.05, 0.1\}$ 
45
46 def gen_data_Y (X, sigma):
47
48     n = np.size(X, 0)
49     d = np.size(X, 1)
50
51     m_X = np.zeros((n,1))
52     m_X[:] = np.nan
53
54     S = np.random.standard_normal(size=(n,1))
55     for t in range(0,n):
56         m_X[t] = m_d(X[t], d)
57

```

```

58     Y = m_X + sigma * iqr(m_X) * S
59     return (m_X, Y)

```

Listing 4.3: help_neural_networks.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Tue Oct 15 11:22:02 2019
5
6  @author: adrian
7
8  Implementation von Neuronale-Netzen welche wir für die Konstruktion unseres
9  Neuronale-Netze-Regressionschätzers benötigen
10 """
11 import numpy as np
12
13 # Sigmoidfunktion
14 #
15 # x: x \in \mathbb{R}
16
17 def sigmoid (x):
18
19     return 1 / (1 + np.exp(-x))
20
21 # Neuronales Netz welches die Funktion f(x) = x approximiert
22 #
23 # x: reelle Zahl
24 # R: reelle Zahl >= 1
25
26 def f_id (x, R):
27
28     return 4 * R * sigmoid(x / R) - 2 * R
29
30 # Neuronales Netz welches die Funktion f(x, y) = x * y approximiert
31 #
32 # x: reelle Zahl
33 # y: reelle Zahl
34 # R: reelle Zahl >= 1
35
36 def f_mult (x, y, R):
37
38     return (((R ** 2) / 4) * (((1 + np.exp(-1)) ** 3) / (np.exp(-2) - np.exp(-1)))) \
39         * (sigmoid(((2 * (x + y)) / R) + 1) - 2 * sigmoid(((x + y) / R) + 1) \
40         - sigmoid(((2 * (x - y)) / R) + 1) + 2 * sigmoid(((x - y) / R) + 1))
41
42 # Neuronales Netz welches die Funktion f(x) = max(x,0) approximiert
43 #
44 # x: reelle Zahl
45 # R: reelle Zahl >= 1
46
47 def f_relu (x, R):
48
49     return f_mult(f_id(x, R), sigmoid(R * x), R)
50

```

```

51 # Neuronales Netz welches die Funktion  $f(x) = \max(1 - (M/(2 * a)) * \text{abs}(x - y), 0)$ 
    approximiert
52 #
53 # x: reelle Zahl
54 # y: fixe reelle Zahl
55 # R: reelle Zahl  $\geq 1$ 
56 # M: fixe natürliche Zahl
57 # a: fixe Zahl  $> 0$ 
58
59 def f_hat (x, y, R, M, a):
60
61     return f_relu((M / (2 * a)) * (x - y) + 1, R) - 2 * f_relu((M / (2 * a)) * (x - y),
        R) + 2 * f_relu((M / (2 * a)) * (x - y) - 1, R)

```

Listing 4.4: new_neural_network.py

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Wed Oct 16 15:40:14 2019
5
6 @author: adrian
7
8 Um die Gewichte der Ausgabeschicht zu bestimmen lösen wir ein regularisiertes
9 Kleinste-Quadrate Problem.
10
11 """
12 import scipy.special
13 import numpy as np
14 import itertools
15 from help_neural_networks import f_id, f_mult, f_hat
16 import math
17
18 # Neuronales Netz welches die Funktion  $f(x) = (x^{(1)} - x_{ik}^{(1)})^{j_1} * \dots * (x^{(d)} - x_{ik}^{(d)})^{j_d} * \prod_{j=1}^d \max((1 - (M/2a) * \text{abs}(x^{(j)} - x_{ik}^{(j)})), 0)$ 
19 #
20 #
21 # x: Eingabevektor für das Neuronale Netz  $x \in [-a, a]^d$ 
22 # d: Ist die Dimension des Eingabevektors  $d > 0$ 
23 # j_1_d: Ist ein d-dimensionaler Vektor  $j_1, \dots, j_d \in \{0, 1, \dots, N\}$ 
24 # X_i: Ist eine  $d \times (M+1)^d$  Matrix.
25 # N: Natürliche Zahl  $\geq q$ 
26 # q:
27 # s:  $\lceil \log_2(N + d) \rceil$ 
28 # R: Zahl  $\geq 1$ 
29 # M:  $M \in \mathbb{N}$ 
30 # a:  $> 0$ 
31
32 def f_net (x, d, j_1_d, X_i, N, q, s, R, M, a):
33     #initialize f_l_k
34     #  $(2^{**s}) + 1$ 
35     #  $((1 + M) ** d) + 1$ 
36     f_l_k = np.empty((s + 1, (2 ** s) + 1,))
37     f_l_k[:] = np.nan
38
39
40     for k in range(np.sum(j_1_d) + d + 1, (2 ** s) + 1, 1):

```

```

41         f_l_k[s, k] = 1
42
43     for k in range(1, d + 1, 1):
44         f_l_k[s, np.sum(j_l_d) + k] = f_hat(x[k - 1], X_i[k - 1], R, M, a)
45
46     for l in range(1, d + 1, 1):
47         k = j_l_d[range(0, l - 1, 1)].sum() + 1
48         while k in range(j_l_d[range(0, l - 1, 1)].sum() + 1, j_l_d[range(0, l, 1)].sum()
49             + 1, 1):
50             f_l_k[s, k] = f_id(f_id(x[l - 1] - X_i[l - 1], R), R)
51             k += 1
52
53     for l in range(s - 1, -1, -1):
54         for k in range((2 ** l), 0, -1):
55             f_l_k[l, k] = f_mult(f_l_k[l + 1, (2 * k) - 1], f_l_k[l + 1, 2 * k], R)
56
57     return f_l_k[0, 1]
58
59 # Bestimmung der Gewichte der Ausgabeschicht durch lösen eines regularisierten
60 # Kleinste-Quadrate Problems
61 #
62 # X: Eingabevektoren der Form (X_1,...,X_n) für das Neuronale Netz aus dem Datensatz (X
63 # _1,Y_1) ,... ,(X_n,Y_n)
64 # Y: Eingabevektoren der Form (Y_1,...,Y_n) für das Neuronale Netz aus dem Datensatz (X
65 # _1,Y_1) ,... ,(X_n,Y_n)
66 # N: Natürliche Zahl >= q
67 # q:
68 # R: Zahl >= 1
69 # d: Ist die Dimension des Eingabevektors d > 0
70 # M: M \in \mathbb{N}
71 # a: >0
72
73 def output_weights(X, Y, N, q, R, d, M, a):
74
75     s = math.ceil(math.log2(N + d))
76
77     # Anzahl der Eingabevektoren X_1,...,X_n
78
79     n = np.size(X, 0)
80
81     # Eine beliebige constante > 0
82
83     #c_3 = np.random.randint(1,10)
84     c_3 = 0.01
85
86     # Anzahl der Spalten der Matrix für das Kleinste-Quadrate Problem
87     # In den Spalten sind die Funktionswerte von f_net gespeichert
88
89     J = int(((1 + M) ** d) * scipy.special.binom(N + d, d))
90
91     # Für die Konstruktion der Matrix brauchen wir erstmal alle Inputparameter
92     # für f_net, da wir dort nur den Funktionswert für einen Vektor j_1,...,j_d
93     # einsetzen
94     # müssen wir erstmals alle möglichen Vektoren dieser Art konstruieren die die
95     # Bedingung 0 <= j_1 + ... + j_d <= N erfüllen

```

```

91     # X_ik hat in den Zeilen die Vektoren X_i aus dem Paper
92
93     X_ik = np.transpose(np.empty((d, (1 + M) ** d)))
94     X_ik[:,] = np.nan
95
96     I_k = np.array(list(itertools.product(range(0, M + 1), repeat = d)))
97     X_ik[:,] = (I_k[:,] * ((2 * a) / M)) - a
98
99     all_j1_jd = np.array(list(itertools.product(range(0, N + 1), repeat = d)))
100    all_j1_jd_by_cond = all_j1_jd[all_j1_jd.sum(axis=1) <= N]
101
102    B = np.empty((n, J,))
103    B[:,] = np.nan
104
105    for i in range(0, n):
106        j = 0
107        for k in range(0, ((M + 1) ** d)):
108            for z in range(0, int(scipy.special.binom(N + d, d))):
109                B[i, j] = f_net(X[i], d, all_j1_jd_by_cond[z], X_ik[k], N, q, s, R, M, a)
110                j += 1
111
112    #weights = np.linalg.solve((1 / n) * np.dot(np.transpose(B), B) + (c_3 / n) * np.
113        identity(J), (1 / n) * np.dot(np.transpose(B), Y))
114    weights = np.linalg.solve(np.dot(np.transpose(B), B) + (c_3) * np.identity(J), np.dot
115        (np.transpose(B), Y))
116
117    return (weights, J, all_j1_jd_by_cond, X_ik)
118
119 # Bestimmung des Funktionswert des Neuronale-Netzte-Regressionsschätzers
120 #
121 # x: Eingabe für einen Vektor der Form [-a,a]^d für den eine Schätzung bestimmt werden
122     soll
123 # X: Eingabevektoren der Form (X_1,...,X_n) für das Neuronale Netz aus dem Datensatz (X
124     _1,Y_1) ,...,(X_n,Y_n)
125 # Y: Eingabevektoren der Form (Y_1,...,Y_n) für das Neuronale Netz aus dem Datensatz (X
126     _1,Y_1) ,...,(X_n,Y_n)
127 # N: Natürliche Zahl >= q
128 # q:
129 # s: [log_2(N + d)]
130 # R: Zahl >= 1
131 # d: Ist die Dimension des Eingabevektors d > 0
132 # M: M \in \N
133 # a: >0
134
135 def new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a):
136
137     Y_pred = np.empty((len(X_test), 1,))
138     Y_pred[:,] = np.nan
139
140     s = math.ceil(math.log2(N + d))
141
142     weights, J, all_j1_jd_by_cond, X_ik = output_weights(X_train, Y_train, N, q, R, d, M
143         , a)
144
145     F_net = np.empty((1, J,))

```

```

140     F_net[:] = np.nan
141
142     for u in range(0, len(X_test), 1):
143         j = 0
144         while j < J:
145             for k in range(0, ((M + 1) ** d)):
146                 for z in range(0, int(scipy.special.binom(N + d, d))):
147                     F_net[0, j] = f_net(X_test[u], d, all_jl_jd_by_cond[z], X_ik[k], N, q
148                                     , s, R, M, a)
149                     j += 1
150
151     Y_pred[u] = np.sum(np.transpose(weights) * F_net)
152
153     return Y_pred

```

Listing 4.5: fc_neural_network.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Oct 11 14:23:15 2019
5
6  @author: adrian
7  """
8  import numpy as np
9  from keras.models import Sequential
10 from keras.layers import Dense
11
12 # Fully-Connected Neuronales Netz mit einer Verborgenen schicht welches die
13 # Anzahl der Neuronen adaptiv, durch minimierung des L2 fehlers, aus der Menge \{5, 10,
14 # 25, 50, 75\} auswählt.
15 #
16 # X: Eingabevektoren der Form (X_1,...,X_n) für das Neuronale Netz aus dem Datensatz (X
17 # _1,Y_1) ,...,(X_n,Y_n)
18 # Y: Eingabevektoren der Form (Y_1,...,Y_n) für das Neuronale Netz aus dem Datensatz (X
19 # _1,Y_1) ,...,(X_n,Y_n)
20
21 def fc_neural_1_estimate (X_train ,Y_train ,X_test):
22
23     Ynew = np.empty((len(X_train) , len([5,10,25,50,75]) ,))
24     Ynew[:] = np.nan
25
26     count = 0
27     n_neurons = [5,10,25,50,75]
28
29     d = np.size(X_train , 1)
30
31     for j in n_neurons:
32         model = Sequential()
33         model.add(Dense(j, input_dim=d, activation='relu'))
34         model.add(Dense(1, activation='linear'))
35         model.compile(loss='mse' , optimizer='adam')
36         model.fit(X_train , Y_train , epochs=1000, verbose=0)
37
38         Ynew[:, count] = model.predict(X_train)[: ,0]
39         count += 1

```



```

37
38     Diff = Ynew[:] - Y_train[:]
39     best_n_neurons = n_neurons[(1/len(X_train))*(Diff.sum(axis=0)**2).argmin()]
40
41     model = Sequential()
42     model.add(Dense(best_n_neurons, input_dim=d, activation='relu'))
43     model.add(Dense(1, activation='linear'))
44     model.compile(loss='mse', optimizer='adam')
45     model.fit(X_train, Y_train, epochs=1000, verbose=1)
46
47     return model.predict(X_test)

```

Listing 4.6: nearest_neighbor.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Oct 11 11:14:47 2019
5
6  @author: adrian
7  K-Nearest-Neighbors Algorithm
8  """
9
10 # Generate sample data
11 import numpy as np
12 from sklearn import neighbors
13 from sklearn.model_selection import GridSearchCV
14 import warnings
15
16 warnings.simplefilter(action='ignore', category=FutureWarning)
17
18 # Implementierung des k-Nächste-Nachbarn-Algorithmus. Dieser bestimmt auch selber bei
19 # einer Liste von Anzahlen an Nachbarn die betrachtet werden
20 # sollen welches die beste Wahl ist.
21 #
22 # X: Inputvektor für das Kalibrieren des Modells
23 # Y: Inputvektor für das Kalibrieren des Modells (Zielvektor an den die Gewichte
24 # angepasst werden)
25 # T: Inputvektor für den eine Vorhersage bestimmte werden soll
26
27 def nearest_neighbor_estimate (X_train, Y_train, X_test):
28
29     params = {'n_neighbors':[2,3,4,5,6,7,8,9], 'weights': ['uniform', 'distance']}
30
31     knn = neighbors.KNeighborsRegressor()
32
33     knn_gridsearch_model = GridSearchCV(knn, params, cv=5)
34     knn_gridsearch_model.fit(X_train, Y_train)
35
36     return knn_gridsearch_model.predict(X_test)

```

Listing 4.7: constant.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-

```

```
3 """
4 Created on Wed Oct 23 15:26:19 2019
5
6 @author: adrian
7 Constant Estimator
8 """
9 import numpy as np
10 from scipy import mean
11
12 # Gibt den Mittelwert der Funktionswerte einer Funktion als Schätzer zurück
13 #
14 # Y: Datensatz der Form (Y_1,...) wobei  $Y_i \in \mathbb{R}$  für  $i = 1, \dots$ 
15
16 def constant_estimate(Y):
17     m = np.zeros((len(Y),1,))
18     m[:] = mean(Y)
19     return m
```