TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Mathematik

Masterarbeit

Analyse der Konvergenzgeschwindigkeit eines einfach berechenbaren Neuronale-Netze-Regressionsschätzers

Adrian Gabel

XX.03.2020

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, 22.02.2020

Adrian Gabel

Inhaltsverzeichnis

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.

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 Konvergenzergebnis 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.
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 $\mathbb{E}[Y^2] < \infty$. Zudem sei $m\colon \mathbb{R}^d \to \mathbb{R}$ definiert durch $m(x) = \mathbb{E}[Y \mid X = x]$ die zugehörige Regressionsfunktion. Ausgehend von der Stichprobe

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

soll 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 $f\colon \mathbb{R}^d \to \mathbb{R}$ 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: $(X,Y),(X_1,Y_1),(X_2,Y_2),\ldots$ seien u.i.v. $\mathbb{R}^d \times \mathbb{R}$ wertige Zufallsvariablen mit $\mathbb{E}[Y^2] < \infty$ und $m\colon \mathbb{R}^d \to \mathbb{R}$ definiert durch $m(x) = \mathbb{E}[Y \mid X = x]$ sei die zugehörige Regressionsfunktion. Gegeben sei die Datenmenge

$$\mathscr{D}_n = \{(X_1,Y_1),\ldots,(X_n,Y_n)\}.$$

Gesucht ist eine Schätzung

$$m_n(\cdot) = m_n(\cdot,\mathscr{D}_n)\colon \mathbb{R}^d \to \mathbb{R}$$

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))

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 $f\colon \mathbb{R}^d \to \mathbb{R}$ heißt (p,C)-glatt, falls für alle $\alpha = (\alpha_1,\ldots,\alpha_d) \in \mathbb{N}_0^d$ mit $\sum_{j=1}^d \alpha_j = q$ die partielle Ableitung

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

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

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

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 Netzarchitektur 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 $x_1, \ldots, x_n$, 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 $w_{1_j}, \ldots, w_{n_j}$ 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 Eingabesicht 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 „vollständig verbundenes“ (*fully connected*) neuronales Netz bezeichnet, wenn sämtliche Neuronen einer Schicht mit allen der darauffolgenden verbunden sind. Jedes Neuron der verborgenen Schicht ist also mit allen Neuronen der Eingabeschicht verbunden und ebenso ist jedes Neuron der Ausgabeschicht mit allen der (letzten) verborgenen Schicht verbunden.

Da wir uns in dieser Arbeit mit neuronalen Netzen zur Regressionsschätzung beschäftigen und Funktionswerte schätzen wollen, betrachten wir nur neuronale Netze die in der Ausgabeschicht ein einzelnes Neuron besitzen.
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.

Der Ausgangspunkt für die Definition eines neuronalen Netzes ist die Wahl einer Aktivierungsfunktion $\sigma\colon \mathbb{R} \to \mathbb{R}$. Wir haben uns in dieser Arbeit für die sogenannten *squashing functions* entschieden, welche eine monoton wachsend Funktion ist, für die

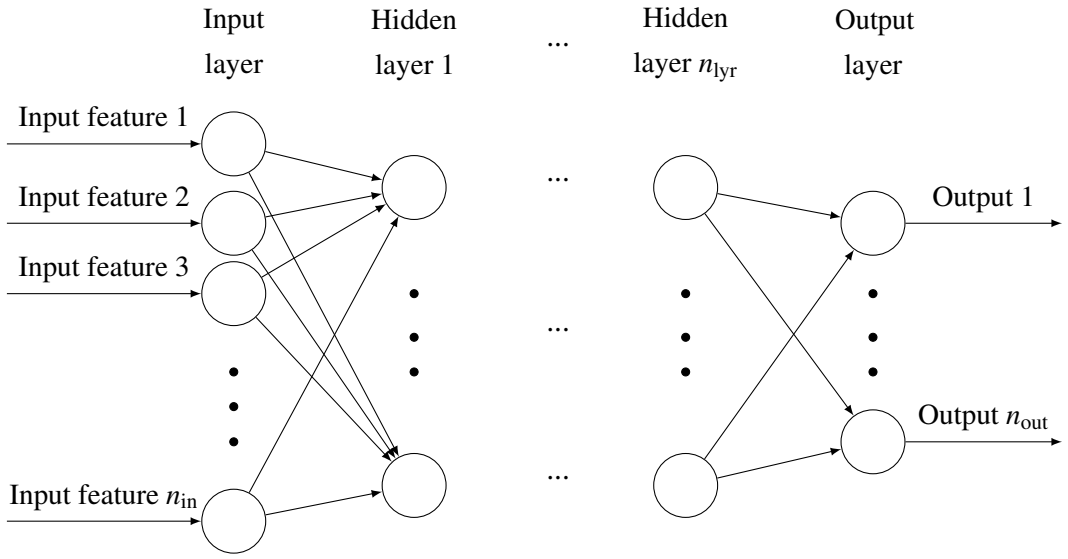

Abbildung 1.1: Fully connected feedforward neuronales Netz mit einer Eingabeschicht mit n_{in} Neuronen, n_{lyr} vielen verborgenen Schichten dessen Anzahl an Neuronen variieren kann und einer Ausgabeschicht bestehend aus n_{out} Neuronen.

$\lim_{x\to-\infty}\sigma(x)=0$ und $\lim_{x\to\infty}\sigma(x)=1$ gilt. Ein Beispiel für eine squashing function ist der sogenannte sigmoidal bzw. logistische squasher

$$\sigma(x)=\frac{1}{1+\exp(-x)} \quad (x\in\mathbb{R}). \tag{1.1}$$

Definition 1.3. Sei $N\in\mathbb{N}_0$. Eine Funktion $\sigma\colon\mathbb{R}\to[0,1]$ 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 1.6 werden wir zudem zeigen, dass der logistische squasher (1.1) N-zulässig ist für beliebiges $N\in\mathbb{N}$.
Als nächstes beschreiben wir Überdeckungszahlen, da wir im Beweis für unser Hauptresultat eine Abschätzung einer L_p-ε-Überdeckungszahl anwenden.

Definition 1.4. (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:

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

Definition 1.5. Sei $\mathscr{F}$ eine Menge von Funktionen $f\colon \mathbb{R}^d \to \mathbb{R}$, sei $\varepsilon > 0$, $1 \le p < \infty$ 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 $\mathscr{F}$ auf x_1^n definiert durch:

$$\mathscr{N}_p(\varepsilon,\mathscr{F},x_1^n) \coloneqq \mathscr{N}_{(X,d)}(\varepsilon,\mathscr{F}),$$

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

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

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

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

In Lemma 1.14 liefern wir ein Resultat zur Abschätzung einer L_p-ε-Überdeckungszahl.

1.2 Hilfsresultate

Lemma 1.6. *Sei $N \in \mathbb{N}_0$ beliebig, dann erfüllt der logistische squasher $\sigma\colon \mathbb{R} \to [0,1], \sigma(x) = \frac{1}{1+\exp(-x)}$ die Bedingungen aus Definition 1.3.*

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

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

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

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

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 1.3, 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 nach Satz ... aus (REFERENZ), 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:

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

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

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

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 1.3 gezeigt und unsere Aussage bewiesen. □

Lemma 1.7 (Langrangesche Form des Restglieds [For16]). *Sei $f : I \to \mathbb{R}$ 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 1.8. *Sei $\sigma\colon \mathbb{R} \to \mathbb{R}$ eine Funktion und R, $a > 0$.*

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

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

für beliebige $x \in [-a,a]$:

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

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

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

für beliebige $x \in [-a,a]$:

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

Beweis. a) Wir wissen, dass f_{id} 2-mal differenzierbar ist, da nach Vorraussetzung σ 2-mal stetig differenzierbar ist. Damit folgt mit Lemma 1.7 für $f = f_{\mathrm{id}}, N = 1$ und $I = [-a,a]$, dass ein $\xi \in [0,x]$ existiert, so dass:

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

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

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

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 1.7 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:

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

□

Lemma 1.9. *Sei $\sigma\colon \mathbb{R} \to [0,1]$ 2-zulässig. Zudem sei $R > 0$ und $a > 0$ beliebig. Dann gilt für das neuronale Netz*

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

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

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

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

Betrags und der Anwendung der Dreiecksungleichung:

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

wobei wir bei der letzten Ungleichung verwendet haben, dass $a>0$ nach Lemma 1.8b) 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 1.10. *Sei $\sigma\colon\mathbb{R}\to[0,1]$ 2-zulässig. Sei f_{mult} das neuronale Netz aus Lemma 1.9 und f_{id} das neuronale Netz aus Lemma 1.8. Angenommen es gelten die Ungleichungen*

$$a\geq 1\quad\textit{und}\quad R\geq\frac{\|\sigma''\|_\infty\cdot a}{2\cdot|\sigma'(t_\sigma)|}. \tag{1.2}$$

Dann erfüllt das neuronale Netz

$$f_{\text{ReLU}}(x)=f_{\text{mult}}(f_{\text{id}}(x),\sigma(R\cdot x)) \tag{1.3}$$

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 $x\in\mathbb{R}\setminus\{0\}$:

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

und

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

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

$$|\sigma(R\cdot x)-\mathbb{1}_{[0,\infty)}(x)|\leq\frac{1}{|R\cdot x|}=\frac{1}{R\cdot|x|}. \tag{1.4}$$

Nach Lemma 1.8 und Lemma 1.9 gilt:

$$|f_{\text{id}}(x)-x|\leq\frac{\|\sigma''\|_\infty\cdot a^2}{2\cdot|\sigma'(t_{\sigma,\text{id}})|}\cdot\frac{1}{R}\ \text{ für }\ x\in[-a,a] \tag{1.5}$$

und

$$|f_{\text{mult}}(x,y) - x \cdot y| \leq \frac{160 \cdot \|\sigma'''\|_\infty \cdot a^3}{3 \cdot |\sigma''(t_\sigma)|} \cdot \frac{1}{R} \text{ für } x,y \in [-2a,2a]. \tag{1.6}$$

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 1.8 und (1.2) für R:

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

wobei $x \in [-a,a]$. Daraus folgt insbesondere $f_{\text{id}}(x) \in [-2a,2a]$. Mithilfe von $\max\{x,0\} = x \cdot \mathbb{1}_{[0,\infty)}(x)$, (1.3, zweier Nulladditionen und dem zweifachen Anwenden der Dreiecksungleichung erhalten wir:

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

Daraus ergibt sich mithilfe von (1.4) - 1.6), $\sigma(Rx) \in [0,1]$ 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 1.11. *Sei $M \in \mathbb{N}$ und sei $\sigma\colon \mathbb{R} \to [0,1]$ 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 1.10. 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:

$$(1 - \frac{M}{2a}\cdot|x|)_+ = \max\{\frac{M}{2a}\cdot x+1,0\} - 2\cdot\max\{\frac{M}{2a}\cdot x,0\} + \max\{\frac{M}{2a}\cdot x-1,0\}, \quad (1.7)$$

die wir im zweiten Teil dieses Beweises zeigen werden. Damit beweisen wir das Resultat mit Hilfe von Lemma 1.10, denn mit der Definition von $f_{\text{hat},y}(x)$ und zwei mal der Dreiecksungleichung folgt:

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

wobei die letzte Ungleichung daraus folgt, dass wir auf jeden Summanden mit $1 \leq a = M+1$ Lemma 1.10 angewendet haben und die Abschätzung

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

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

Um Gleichung (1.7) zu zeigen unterscheiden wir vier Fälle.

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

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

und die rechte Seite die Form

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

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 bestitz 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 (1.7) bewiesen und damit ist der Beweis vollständig. $\square$

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 1.12 ([AB19], Lemma 5)**.** *Sei $M \in \mathbb{N}$ und $\sigma\colon \mathbb{R} \to [0,1]$ 2-zulässig nach Definition 1.3. (AUF SCHREIBWEISE VON FNET AUFPASSEN) Sei $a \geq 1$ und*

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

und sei $y \in [-a,a]^d$. Sei $N \in \mathbb{N}$ und $j_1,\dots,j_d \in \mathbb{N}_0$ so, dass $j_1+\cdots+j_d \leq N$ gilt und wir setzen $s = \lceil \log_2(N+d) \rceil$. Sei $f_{\mathrm{id}}, f_{\mathrm{mult}}$ und $f_{\mathrm{hat},z}$ (für $z \in \mathbb{R}$) die neuronalen Netze wir in Lemma 1.8, Lemma 1.9 und Lemma 1.11. Wir definieren das Netz $f_{net,j_1,\dots,j_d,y}$ durch:

$$f_{net,j_1,\dots,j_d,y}(x) = f_1^{(0)}(x).$$

wobei

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

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

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

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

$$f_{j_1+j_2+\cdots+j_d+k}^{(s)}(x) = f_{\mathrm{hat},y^{(k)}}(x^{(k)})$$

für $1 \leq k \leq d$ und

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

für $j_1+j_2+\cdots+j_d+d+1 \leq k \leq 2^s$.

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

$$\left| f_{net,y}(x) - (x^{(1)}-y^{(1)})^{j_1} \cdots (x^{(d)}-y^{(d)})^{j_d} \prod_{j=1}^{d} (1 - \frac{M}{2a} \cdot |x^{(j)}-y^{(j)}|)_+ \right|$$
$$\leq c_{12} \cdot 3^{3 \cdot 3^s} \cdot a^{3 \cdot 2^s} \cdot M^3 \cdot \frac{1}{R}.$$

Lemma 1.13 ([AB19], Lemma 8)**.** *Sei $\beta_n = c_6 \cdot \log(n)$ für eine hinreichend große Konstante $c_6 > 0$. Angenommen die Verteilung von (X,Y) erfüllt*

$$\mathbb{E}\left(\mathrm{e}^{c_4 \cdot |Y|^2}\right) < \infty$$

für eine Konstante $c_4 > 0$ und dass der Betrag der Regressionsfunktion m beschränkt ist. Sei $\mathscr{F}_n$ eine Menge von Funktionen $f\colon \mathbb{R}^d \to \mathbb{R}$ und wir nehmen an, dass der Schätzer m_n

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

erfüllt, mit

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

und

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

mit einer nichtleeren Parametermenge Θ_n, zufällige Funktionen $g_{n,l}\colon \mathbb{R}^d\to\mathbb{R}$ und deterministischen penalty Terme $pen_n(g_{n,l})\ge 0$, wobei die zufälligen Funktionen $g_{n,l}\colon \mathbb{R}^d\to\mathbb{R}$ 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:

$$\begin{aligned}&\mathbb{E}\int|m_n(x)-m(x)|^2\mathbb{P}_X(dx)\\ &\le \frac{c_{13}\cdot\log(n)^2\cdot\left(\log\left(\sup_{x_1^n\in(supp(X))^n}\mathscr{N}_1\left(\frac{1}{n\cdot\beta_n},\mathscr{F}_n,x_1^n\right)\right)+1\right)}{n}\\ &\quad+2\cdot\mathbb{E}\left(\min_{l\in\Theta_n}\int|g_{n,l}(x)-m(x)|^2\mathbb{P}_X(dx)+pen_n(g_{n,l})\right),\end{aligned}$$

für $n>1$ und einer Konstante $c_{13}>0$ welche nicht von n abhängt.

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

Lemma 1.14 ([AB19], Lemma 9)**.** *Sei $a>0$ und $d,N,J_n\in\mathbb{N}$ so, dass $J_n\le n^{c_{14}}$ und setze $\beta_n=c_6\cdot\log(n)$. Sei σ 2-zulässig nach Definition 1.3. Sei $\mathscr{F}$ die Menge aller Funktionen die durch (2.1) definiert sind mit $k_1=k_2=\cdots=k_L=24\cdot(N+d)$ und dass der Betrag der Gewichte durch $c_{15}\cdot n^{c_{16}}$ beschränkt ist. Sei*

$$\mathscr{F}^{(J_n)}=\left\{\sum_{j=1}^{J_n}a_j\cdot f_j : f_j\in\mathscr{F}\quad und\quad \sum_{j=1}^{J_n}a_j^2\le c_{17}\cdot n^{c_{18}}\right\}.$$

Dann gilt für $n>1$:

$$\log\left(\sup_{x_1^n\in[-a,a]^{d\cdot n}}\mathscr{N}_1\left(\frac{1}{n\cdot\beta_n},\mathscr{F}^{(J_n)},x_1^n\right)\right)\le c_{19}\cdot\log(n)\cdot J_n,$$

für eine Konstante c_{19} die nur von L,N,a und d abhängt.

Kapitel 2

Konstruktion des Neuronale Netze Schätzers

In diesem Kapitel werden wir mithilfe von unseren gegebenen Datenmenge

$$\mathscr{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\},$$

unseren Regressionsschätzer konstruieren.
Die Netzwerkarchitektur (L,k) hängt von der Anzahl der verborgenen Schichten $L \in \mathbb{N}_0$ und und einem Vektor $k = (k_1, \ldots, k_L) \in \mathbb{N}^L$ ab, der mit jeder Komponente die Anzahl der Neuronen in der jeweiligen verborgen Schicht angibt.
Ein mehrschichtiges feedforward neuronales Netz mit Architektur (L,k) und σ aus (1.1) als Aktivierungsfunktion, ist eine reelwertige Funktion $f\colon \mathbb{R}^d \to \mathbb{R}$ definiert durch:

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

für $c_0^{(L)}, \ldots, c_{k_L}^{(L)} \in \mathbb{R}$ und für $f_i^{(L)}$ rekursiv definiert durch:

$$f_i^{(r)}(x) = \sigma\left(\sum_{j=1}^{k_r - 1} c_{i,j}^{(r-1)} \cdot f_j^{(r-1)}(x) + c_{i,0}^{(r-1)} \right) \tag{2.1}$$

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

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

für $c_{i,0}^{(0)}, \ldots, c_{i,d}^{(0)} \in \mathbb{R}$.
Für die Konstruktion unseres Schätzers verwenden wir die gegebene Datenmenge $\mathscr{D}_n$ und wählen die Gewichte des neuronalen Netzes so, dass die resultierende Funktion aus (2.1)

eine gute Schätzungen für die Regressionsfunktion 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 ein regularisiertes Kleinste-Quadrate-Problem (REFERENZ) lösen.

2.1 Definition der Netzwerkarchitektur

Sei $a > 0$ fest und $M \in \mathbb{N}$. Die Wahl der Netzwerkarchitektur und der Werte aller Gewichte bis auf die aus der Ausgabeschicht ist durch folgendes Approximationsresultat durch eine lokale lokale Spline Interpolation von Taylorpolynomen für (p,C)-glatte Funktionen auf dem Intervall $[-a,a]^d$ motiviert. Sei $[M]^d := \{0,1,\dots,M\}^d = \{\mathbf{i}_1,\dots,\mathbf{i}_{(M+1)^d}\}$.
Für $(\mathbf{i}^{(1)},\dots,\mathbf{i}^{(d)}) = \mathbf{i} \in [M]^d$ und $x \in \mathbb{R}^d$ definieren wir

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

Für $f\colon \mathbb{R}^d \to \mathbb{R}$ ausreichend oft differenzierbar definieren wir

$$\partial^{\mathbf{i}} f(x) := \frac{\partial^{|\mathbf{i}|_1} f}{\partial^{\mathbf{i}^{(1)}} x_1 \cdots \partial^{\mathbf{i}^{(d)}} x_d}(x).$$

Wir betrachten im Folgenden ein d-dimensionales äquidistantes Gitter im Würfel $[-a,a]^d$ mit Schrittweite $\frac{2a}{M}$. Sei $\mathbf{i}_1,\dots,\mathbf{i}_{(M+1)^d}$ eine Aufzählung der Elemente von $[M]^d$. Dann ordnen wir jedem Index $\mathbf{i} \in [M]^d$ einen Gitterpunkt $x_{\mathbf{i}}$ mit:

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

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

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

und sei

$$P(x) = \sum_{\mathbf{i} \in [M]^d} p_{\mathbf{i}}(x) \prod_{j=1}^{d} \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}| \right)_+. \tag{2.2}$$

Wir zeigen im folgenden Lemma dass $P(x)$ eine lokale Spline Interpolation von Taylorpolynomen von m ist.

Lemma 2.1 (Multilineare Spline Interpolation)**.** *Sei $a > 0, M \in \mathbb{N}$ und*

$$P(x) = \sum_{\mathbf{i} \in [M]^d} p_{\mathbf{i}}(x) \cdot B_{\mathbf{i}}(x).$$

mit $p_{\mathbf{i}}(x)$ wie oben und

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

dann ist $P(x)$ eine lokale Spline Interpolation von Taylorpolynomen von m.

Beweis. Es sind drei Bedingungen zu überprüfen. Als erstes geben wir aber für $d = 2$ und $M = 3$ eine Skizze an um die Idee des Beweises zu veranschaulichen. Es ist ein Gitter mit

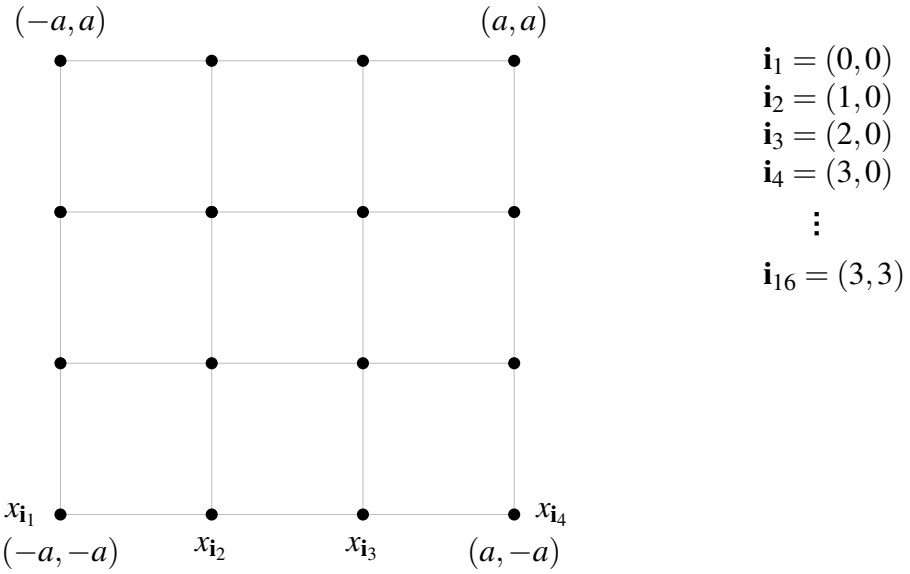

Abbildung 2.1: Beispielhafte Darstellung der $x_{\mathbf{i}_k}$ für $d = 2$ und $M = 3$.

$(M+1)^d$ Gitterpunkten die den $x_{\mathbf{i}_k}$ entsprechen. Der Abstand zwischen zwei Gitterpunkten beträgt $\frac{2a}{M}$. Man betrachtet immer den Abstand zu den nahesten 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.

• Im Folgenden wollen wir

$$\sum_{\mathbf{i} \in [M]^d} B_{\mathbf{i}}(x) = 1 \quad (x \in \mathbb{R}^d) \tag{2.3}$$

per Induktion über d zeigen.

Induktionsanfang (IA): Für $d = 1$ kann nur zwischen zwei Gitterpunkten liegen und mit der obigen Begründung ist der Rest gleich Null, daher nehmen wir oBdA an, dass $x_{\mathbf{i}_1} \leq x \leq x_{\mathbf{i}_2}$.

Damit folgt

$$\begin{aligned}\sum_{\mathbf{i}\in[M]^d}(1-\frac{M}{2a}\cdot|x-x_{\mathbf{i}}|)_+ &= (1-\frac{M}{2a}\cdot|x-x_{\mathbf{i}_1}|)_+ + (1-\frac{M}{2a}\cdot|x-x_{\mathbf{i}_2}|)_+ \\ &= 1+1-\frac{M}{2a}\cdot(x-x_{\mathbf{i}_1}+x_{\mathbf{i}_2}-x) \\ &= 1+1-\frac{M}{2a}\cdot\frac{2a}{M} \\ &= 1,\end{aligned}$$

wobei wir unter anderem verwendet haben, dass beide Summanden unabhängig von dem Positivteil nichtnegativ sind, da der Abstand von x zu den beiden Gitterpunkten $x_{\mathbf{i}_1}$ und$x_{\mathbf{i}_2}$ kleiner gleich $\frac{2a}{M}$ ist. Zudem haben wir verwendet, dass $x_{\mathbf{i}_2}-x_{\mathbf{i}_1}=\frac{2a}{M}$ gilt, da beides Gitterpunkte sind.

Induktionshypothese (IH): Aussage (2.3) gelte für ein beliebiges aber festes $d\in\mathbb{N}$.

Induktionsschritt (IS): Wir nehmen oBdA an, dass $x_{(0,\dots,0)}\le x\le x_{(1,\dots,1)}$ gilt, mit $\mathbf{i}_1=(0,\dots,0)$ und $\mathbf{i}_{(M+1)^{(d+1)}}=(1,\dots,1)$. Das heißt also, dass $x\in[-a,-a+\frac{2a}{M}]^{d+1}$ gilt. Im Folgenden zeigen wir

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

Ein Summand ist Null, wenn $\left|x^{(j)}-x_{\mathbf{i}}^{(j)}\right|\ge\frac{2a}{M}$ ist. Zudem haben wir oBdA 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 nahesten bei x liegen entspricht. Zudem wissen wir, dass alle Gitterpunkte, die in der $(d+1)$ Komponente den selben Wert haben, in dieser Dimension gleich weit von $x^{(d+1)}$ entfernt sind. Das heißt, in jedem Summanden kommt der Faktor $(1-\frac{M}{2a}\cdot\left|x^{(d+1)}-x_{(0,\dots,0)}^{(d+1)}\right|)$ bzw. $(1-\frac{M}{2a}\cdot\left|x^{(d+1)}-x_{(1,\dots,1)}^{(d+1)}\right|)$ vor, da

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

daraus ergibt sich:

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

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. (□)

• 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 (2.3) muss dann auch

$$\prod_{j=1}^{d}\left(1-\frac{M}{2a}\cdot|x^{(j)}-x_{\mathbf{i}}^{(j)}|\right)_+\leq 1 \tag{2.4}$$

gelten.

• Es handelt sich hierbei um eine lokale Konvexität, da die Bedingungen (2.3) und (2.4) für alle $x\in[-a,a]^d$ gelten. □

Als nächstes zeigen wir ein Resultat für (p,C)-glatte Funktionen, welches wir im weiteren Verlauf dieser Arbeit wieder benötigen werden.

Lemma 2.2. *Sei $M \in \mathbb{N}$, c_1 eine Konstante, $a > 0$ und f eine (p,C)-glatte Funktion, wobei $p = q + s$ mit $q \in \mathbb{N}_0$ und $s \in (0,1]$. Sei zudem $P(x)$ analog wie in (2.2) eine lokale Spline Interpolation von Taylorpolynomen von f. Dann gilt*:

$$\sup_{x \in [-a,a]^d} |f(x) - P(x)| \leq c_1 \cdot \frac{1}{M^p}.$$

Beweis. Nach dem Satz über die Lagrange Form des Restglieds (REFERENZ) existiert ein $\xi \in [0,1]$, so, dass

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

Nach (2.3) erhalten wir

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

Zudem wissen wir dass man immer den Abstand zu den nahesten 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:

$$\sum_{q-1 < |\mathbf{j}|_1 \leq q} \partial^{\mathbf{j}} f(x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})) \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \leq \sum_{q-1 < |\mathbf{j}|_1 \leq q} \partial^{\mathbf{j}} f(x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})) \frac{1}{\mathbf{j}!} \cdot \left(\frac{2a}{M}\right)^q$$

und folgern mithilfe der Dreiecksungleichung und der (p,C)-Glattheit von f:

$$\begin{aligned} |f(x) - P(x)| &\leq \sum_{\mathbf{i} \in [M]^d} \left| f(x) - \sum_{\substack{\mathbf{j} \in \{0,\ldots,q\}^d \\ |\mathbf{j}|_1 \leq q}} \partial^{\mathbf{j}} f(x_{\mathbf{i}}) \cdot \frac{(x - x_{\mathbf{i}})^{\mathbf{j}}}{\mathbf{j}!} \right| \prod_{j=1}^{d} \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|\right)_+ \\ &\leq \left(\frac{2a}{M}\right)^q \|x_{\mathbf{i}} - x_{\mathbf{i}} + \xi(x - x_{\mathbf{i}})\|^s \cdot C \cdot \sum_{\mathbf{i} \in [M]^d} \prod_{j=1}^{d} \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|\right)_+ \\ &\leq \left(\frac{2a}{M}\right)^q \cdot d^{s/2} \cdot \left(\frac{2a}{M}\right)^s \cdot C \cdot \sum_{\mathbf{i} \in [M]^d} \prod_{j=1}^{d} \left(1 - \frac{M}{2a} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|\right)_+ \\ &= c_1 \cdot \frac{1}{M^p}, \end{aligned}$$

wobei wir bei der letzten Gleichheit Bedingung (2.3) und $q + s = p$ verwendet haben. □

$P(x)$ 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. 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

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

in ihr enthalten sind. Um dies zu erreichen, sei

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

der logistische Squasher (1.1), wählen $R \geq 1$ und definieren die folgenden neuronale Netze:

Das neuronale Netz

$$f_{\text{id}}(x) = 4R \cdot \sigma\left(\frac{x}{R}\right) - 2R, \tag{2.5}$$

welches, wie in Lemma 1.8 gezeigt, die Funktion $f(x) = x$ approximiert und in Abbildung 2.2 veranschaulicht wird.

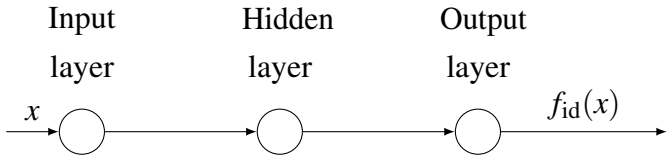

Abbildung 2.2: Neuronales Netz $f_{\text{id}}(x) = 4R \cdot \sigma(\frac{x}{R}) - 2R$

Das neuronale Netz

$$\begin{aligned} f_{\text{mult}}(x,y) = \frac{R^2}{4} \cdot \frac{(1+\exp(-1))^3}{\exp(-2)-\exp(-1)} \cdot \Bigg(& \sigma\left(\frac{2(x+y)}{R} + 1\right) - 2 \cdot \sigma\left(\frac{x+y}{R} + 1\right) \\ & - \sigma\left(\frac{2(x-y)}{R} + 1\right) + 2 \cdot \sigma\left(\frac{x-y}{R} + 1\right)\Bigg), \end{aligned} \tag{2.6}$$

welches, wie in Lemma 1.9 gezeigt, die Funktion $f(x,y) = x \cdot y$ approximiert und in Abbildung 2.3 veranschaulicht wird.

Das neuronale Netz

$$f_{ReLu}(x) = f_{\text{mult}}(f_{\text{id}}(x), \sigma(R \cdot x)), \tag{2.7}$$

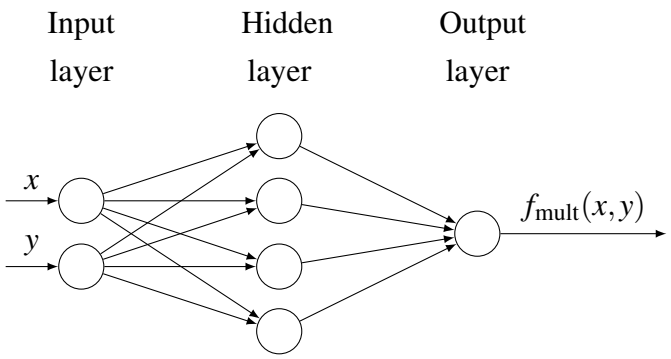

Abbildung 2.3: Neuronales Netz $f_{\text{mult}}(x) = ...$

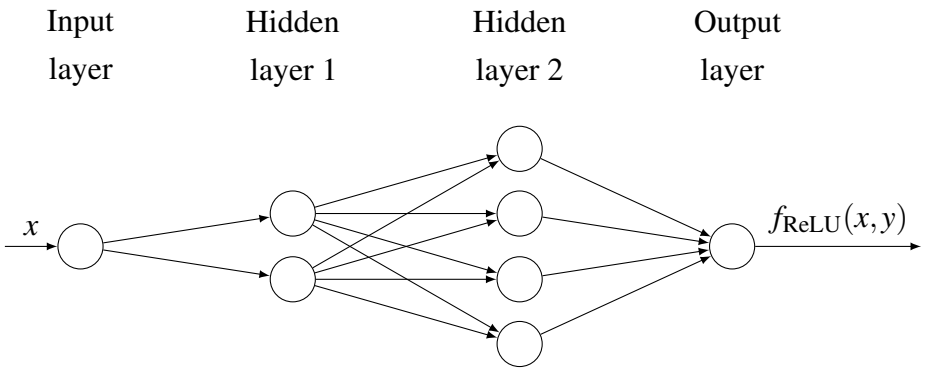

Abbildung 2.4: Neuronales Netz $f_{\text{ReLU}}(x) = ...$

welches, wie in Lemma 1.10 gezeigt, die Funktion $f(x) = x_+$ approximiert und in Abbildung 2.4 veranschaulicht wird und schließlich noch das neuronale Netz

$$f_{\text{hat},y}(x) = f_{ReLu}\left(\frac{M}{2a}\cdot(x-y)+1\right) - 2\cdot f_{ReLu}\left(\frac{M}{2a}\cdot(x-y)\right) + f_{ReLu}\left(\frac{M}{2a}\cdot(x-y)-1\right), \tag{2.8}$$

welches, wie in Lemma 1.11 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

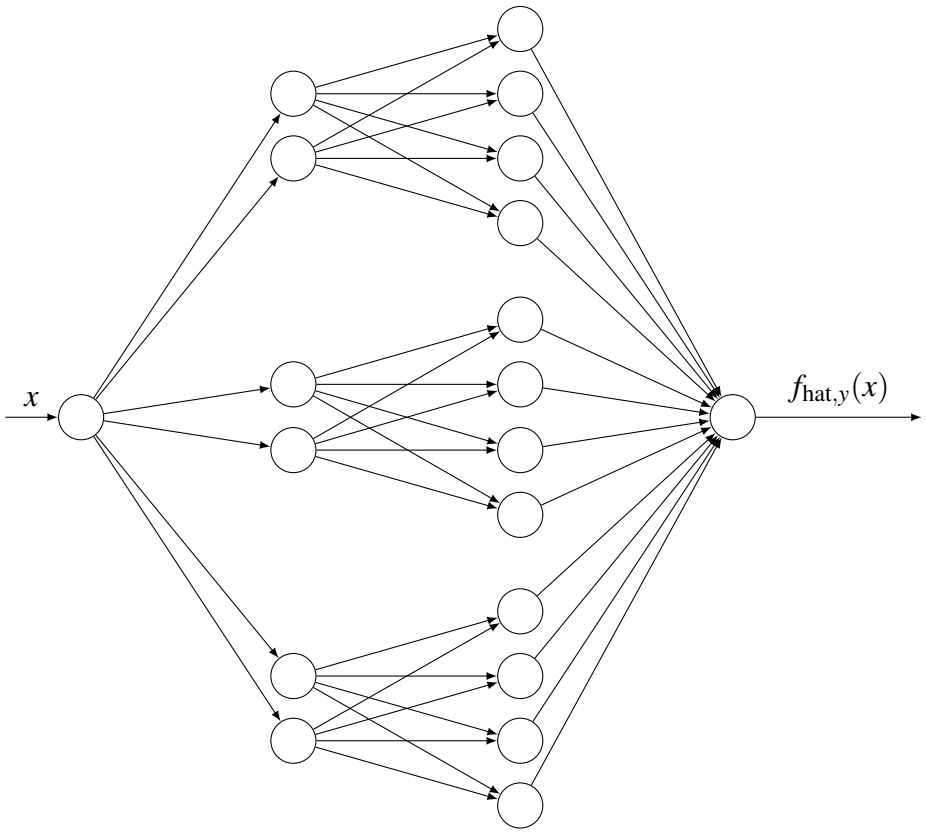

Abbildung 2.5: Neuronales Netz $f_{\text{hat},y}(x) = ...$

können wir nun $f_{\text{net},\mathbf{j},\mathbf{i}}$ rekursiv definieren. Dafür wählen wir $N \geq q$, setzen $s = \lceil \log_2(N+d) \rceil$ und definieren für $\mathbf{j} = j_1,\ldots,j_d \in \{0,1,\ldots,N\}$, $\mathbf{i} \in [M]^d$ und $k \in \{1,\ldots,(M+1)^d\}$:

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

wobei

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

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

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

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

$$f_{j_1+j_2+\cdots+j_d+k}^{(s)}(x) = f_{\text{hat},x_{\mathbf{i}_k}^{(k)}}(x^{(k)}) \tag{2.9}$$

für $1 \leq k \leq d$ und

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

für $j_1 + j_2 + \cdots + j_d + d + 1 \leq k \leq 2^s$.
Da das neuronale Netz $f_{\text{net},\mathbf{j},\mathbf{i}}$ aus mehrere neuronalen Netzen zusammengebaut wurden lässt sich dadurch auch die Anzahl an Schichten und Neuronen pro Schicht durch diese Struktur erklären. Durch (2.9) erkennt man, dass f_{net} $s+2$ verborgene Schichten, durch s-maliges Anwenden von f_{mult} und einer Anwendung von f_{hat}, hat. Da f_{hat} zwei verbogene Schichten besitzt ergibt sich daraus die Anzahl an verborgenen Schichten von f_{net}. Für die Anzahl an Neuronen für die jeweiligen Schichten können wir nur eine oberen Schranke angeben, da ... 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 $3,\ldots,s+2$ enthalten maximal $2 \cdot 2^s, 2^s, \ldots, 8, 4$, 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 $s+2$ verbogenen Schichten mit jeweils $24 \cdot (N+d)$ Neuronen pro Schicht, da für die größte Anzahl an Neuronen in einer Schicht

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

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:

$$\tilde{m}_n(x) = \sum_{\mathbf{i}\in[M]^d} \sum_{\substack{\mathbf{j}\in[N]^d \\ |\mathbf{j}|_1 \leq N}} a_{\mathbf{i},\mathbf{j}} \cdot f_{\text{net},\mathbf{j},\mathbf{i}}(x), \tag{2.10}$$

wobei wir die Koeffizienten $a_{\mathbf{i},\mathbf{j}}$ durch Minimierung von

$$\frac{1}{n}\sum_{i=1}^{n} |Y_i - \tilde{m}_n(X_i)|^2 + \frac{c_3}{n} \cdot \sum_{\mathbf{i}\in[M]^d} \sum_{\substack{\mathbf{j}\in[N]^d \\ |\mathbf{j}|_1 \leq N}} a_{\mathbf{i},\mathbf{j}}^2 \tag{2.11}$$

für eine Konstante $c_3 > 0$. Dieses regularisierte lineare Kleinste-Quadrate Schätzung erhalten wir durch die Lösung eine linearen Gleichungssystems. Dafür definieren wir uns

$$\{B_j \mid j = 1,\ldots,J\} = \left\{ f_{\text{net},\mathbf{j},\mathbf{i}}(x) \mid \mathbf{i} \in [M]^d \quad \text{und} \quad 0 \leq |\mathbf{j}|_1 \leq N \text{ mit } \mathbf{j} \in [N]^d \right\}$$

wobei

$$J = (M+1)^d \cdot \binom{N+d}{d}$$

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 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. Wir betrachten also... (TBD) Das lässt sich zurückführen auf ... (TBD) und damit ergibt sich für $d \geq 1$ aus:

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

mit Identität des Binominialkoeffizienten:

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

TBD ERKLÄRUNG WIESO DIESE ANALOGIE (FABI FRAGEN) Wir setzen nun

$$\mathbf{B} = (B_j(X_i))_{1\leq i\leq n, 1\leq j\leq J} \quad \text{und} \quad \mathbf{Y} = (Y_i)_{i=1,\ldots,n}.$$

Wir zeigen im Folgenden, dass der Koeffizientenvektor unseres Schätzers 2.10 die eindeutige Lösung des linearen Gleichungssystems

$$\left(\frac{1}{n}\mathbf{B}^T\mathbf{B}+\frac{c_3}{n}\cdot\mathbf{1}\right)\mathbf{a}=\frac{1}{n}\mathbf{B}^T\mathbf{Y}, \tag{2.12}$$

wobei $\mathbf{1}$ eine JxJ-Einheitsmatrix ist. Den Schätzer aus 2.10 kann man umschreiben zu

$$\tilde{m}_n(x)=\sum_{j=1}^{J} a_j\cdot B_j(x)$$

wobei $\mathbf{a}=(a_j)_{j=1,\dots,J}\in\mathbb{R}^J$ wie in 2.11 den Ausdruck

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}|Y_i-\tilde{m}_n(X_i)|^2+\frac{c_3}{n}\cdot\sum_{\mathbf{i}\in[M]^d}\sum_{\substack{\mathbf{j}\in[N]^d\\ |\mathbf{j}|_1\leq N}}a_{\mathbf{i},\mathbf{j}}^2\\
&=\frac{1}{n}(\mathbf{Y}-\mathbf{Ba})^T(\mathbf{Y}-\mathbf{Ba})+\frac{c_3}{n}\cdot\mathbf{a}^T\mathbf{a}\\
&=\frac{1}{n}(\mathbf{Y}^T\mathbf{Y}-\mathbf{Y}^T\mathbf{Ba}-\mathbf{a}^T\mathbf{B}^T\mathbf{Y}+\mathbf{a}^T\mathbf{B}^T\mathbf{Ba})+\frac{c_3}{n}\cdot\mathbf{a}^T\mathbf{a}\\
&=\frac{1}{n}(\mathbf{Y}^T\mathbf{Y}-2\mathbf{Y}^T\mathbf{Ba})+\mathbf{a}^T\left(\frac{1}{n}\mathbf{B}^T\mathbf{B}+\frac{c_3}{n}\cdot\mathbf{1}\right)\mathbf{a},
\end{aligned} \tag{2.13}$$

minimiert. In der vorletzten Gleichung haben wir verwendet das $\mathbf{Y}^T\mathbf{Ba}=\mathbf{a}^T\mathbf{B}^T\mathbf{Y}$ gilt, da dieser Ausdruck eine reelle Zahl und damit insbesondere symmetrisch ist. Die Matrix $\mathbf{B}^T\mathbf{B}\in\mathbb{R}^{JxJ}$ ist positiv semidefinit, denn aufgrund der Verschiebungseigenschaft des Standardskalarprodukts gilt für alle $x\in\mathbb{R}$:

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

Zudem wissen wir dass $\frac{c_3}{n}\mathbf{1}$ durch die Wahl von c_3 nur positive Eigenwerte besitzt und damit positiv definit ist. Daher wissen wir, dass die Matrix

$$\mathbf{A}=\frac{1}{n}\mathbf{B}^T\mathbf{B}+\frac{c_3}{n}\cdot\mathbf{1}$$

also Summe einer positiv semidefiniten und einer positiv definiten Matrix nur positive Eigenwerte besitzt (REFERENZ), damit also positiv definit ist und eine inverse Matrix $\mathbf{A}^{-1}$ existiert. Zudem ist die Matrix $\mathbf{A}$ symmetrisch. Mit $\mathbf{b}=\frac{1}{n}\cdot\mathbf{A}^{-1}\mathbf{B}^T\mathbf{Y}\in\mathbb{R}^J$ und $\mathbf{b}^T\mathbf{Aa}=\mathbf{aAb}=\mathbf{Y}^T\mathbf{Ba}$, was aus der Symmetrie von $\mathbf{A}$ folgt, erhalten wir:

$$\begin{aligned}
&\frac{1}{n}(\mathbf{Y}^T\mathbf{Y}-2\mathbf{Y}^T\mathbf{Ba})+\mathbf{a}^T\left(\frac{1}{n}\mathbf{B}^T\mathbf{B}+\frac{c_3}{n}\cdot\mathbf{1}\right)\mathbf{a}\\
&=\mathbf{a}^T\mathbf{Aa}+\mathbf{b}^T\mathbf{Aa}+\mathbf{a}^T\mathbf{Ab}+\mathbf{b}^T\mathbf{Ab}+\frac{1}{n}\mathbf{Y}^T\mathbf{Y}-\frac{1}{n^2}\mathbf{Y}^T\mathbf{BA}^{-1}\mathbf{B}^T\mathbf{Y}\\
&=(\mathbf{a}+\frac{1}{n}\cdot\mathbf{A}^{-1}\mathbf{B}^T\mathbf{Y})^T\mathbf{A}(\mathbf{a}-\frac{1}{n}\cdot\mathbf{A}^{-1}\mathbf{B}^T\mathbf{Y})-\frac{1}{n}\mathbf{Y}^T\mathbf{Y}-\frac{1}{n^2}\mathbf{Y}^T\mathbf{BA}^{-1}\mathbf{B}^T\mathbf{Y}.
\end{aligned}$$

Die letzte Gleichung wird für $\mathbf{a} = \frac{1}{n} \cdot \mathbf{A}^{-1}\mathbf{B}^T\mathbf{Y}$ minimal, da wir wissen dass $\mathbf{A}$ positiv definit ist und damit $x^T\mathbf{A}x > 0$ für alle $x \in \mathbb{R}^J$ mit $x \neq 0$ gilt und $(\mathbf{a}-\mathbf{b})^T\mathbf{A}(\mathbf{a}-\mathbf{b}) = 0$ ist für $\mathbf{a} = \mathbf{b}$. Dies zeigt also, dass der Koeffizientenvektor unseres Schätzers 2.10 die eindeutige Lösung des linearen Gleichungssystems 2.12 ist. Da der Koeffizientenvektor die Gleichung 2.11 minimiert, erhalten wir wenn wir den Koeffizientenvektor mit dem Nullvektor gleichsetzen:

$$\frac{1}{n}(\mathbf{Y}-\mathbf{B}\mathbf{a})^T(\mathbf{Y}-\mathbf{B}\mathbf{a}) + \frac{c_3}{n} \cdot \mathbf{a}^T\mathbf{a} \leq \frac{1}{n}\sum_{i=1}^{n} Y_i^2,$$

was uns erlaubt eine obere Schranke für den absoluten Wert unsere Koeffizienten abzuleiten.

Kapitel 3

Resultat zur Konvergenzgeschwindigkeit

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 2.10 mit einer (p,C)-glatten Regressionsfunktion herzuleiten.

Satz 3.1. *Angenommen die Verteilung von (X,Y) erfüllt*

$$\mathbb{E}\left(e^{c_4 \cdot |Y|^2}\right) < \infty$$

für eine Konstante $c_4 > 0$ und die Verteilung von X hat einen beschränkten Träger $supp(\mathbb{P}_X)$. Sei $m(x) = \mathbb{E}[Y \mid X = x]$ die entsprechende Regressionsfunktion. Angenommen m ist (p,C)-glatt, mit $p = q + s$ für $q \in \mathbb{N}_0$ und $s \in (0,1]$. Wir betrachten unseren neuronale Netze Regressionsschätzer $\tilde{m}_n$ aus 2.10, wobei σ der logistische squasher ist und $N \geq q, M = M_n = \lceil c_5 \cdot n^{1/(2p+d)} \rceil, R = R_n = n^{d+4}$ und $a = a_n = (\log n)^{1/(6(N+d))}$. Sei $\beta_n = c_6 \cdot \log(n)$ für eine hinreichend große Konstante $c_6 > 0$ 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:

$$\mathbb{E} \int |m_n(x) - m(x)|^2 \mathbb{P}_X(dx) \leq c_7 \cdot (\log n)^3 \cdot n^{-\frac{2p}{2p+d}},$$

wobei $c_7 > 0$ ist und nicht von n abhängt.

Beweis. Nach Voraussetzung wissen wir, dass $supp(\mathbb{P}_X)$ beschränkt ist, daher nehmen wir ohne Beschränkung der Allgemeinheit an, dass $supp(X) = \{x \mid \mathbb{P}_X(x) > 0\} \subseteq [-a_n, a_n]^d$ ist, da $\mathbb{P}(X \in supp\mathbb{P}_X)) = 1$ gilt und wir ansonsten die Zufallsvariable X auf Nullmengen abändern können. Zudem haben wir angenommen, dass m (p,C)-glatt und damit insbesondere hölderstetig mit $q = 0$ ist. Daraus können wir auf die gleichmäßige Stetigkeit von m schließen [Sto18]. Da wir nur über den beschränkten $supp(\mathbb{P}_X)$ integrieren, wissen wir, dass m als gleichmäßig stetige Funktion auf einer beschränkten Menge auch beschränkt ist [Sto18]. Wir können daher ohne Beschränkung der Allgemeinheit folgern, dass $\|m\|_\infty \leq \beta_n$ ist, da aufgrund der Beschränktheit von m, ebenfalls $|m|$ beschränkt ist und wir daher ansonsten eine Skalierung von m nehmen können.
Sei $\mathscr{F}$ die durch 2.1 definierte Menge von Funktionen mit $L = s+2 = \lceil \log_2(N+d) \rceil + 2$, mit $k_1 = k_2 = \cdots = k_L = 24 \cdot (N+d)$ und der Eigenschaft, dass der Betrag der Gewichte durch $n^{c_{20}}$ beschränkt ist. Sei

$$\mathscr{F}^{(J_n)} = \left\{ \sum_{j=1}^{J_n} a_j \cdot f_j \mid f_j \in \mathscr{F} \quad \text{und} \quad \sum_{j=1}^{J_n} a_j^2 \leq c_{21} \cdot n \right\}$$

wobei c_{21} in (3.3) gewählt wird und

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

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

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

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:

$$J_n \leq (M_n+1)^d \cdot (N+1)^d. \tag{3.1}$$

Da nach Voraussetzung m (p,C)-glatt ist, und $x_\mathbf{i} \in \mathbb{R}^d$ für alle $\mathbf{i} \in [M]^d$, folgt:

$$\max_{\mathbf{i} \in [M]^d, \mathbf{j} \in [q]^d, |\mathbf{j}|_1 \leq q} \left| \partial^\mathbf{j} m(x_\mathbf{i}) \right| < \infty, \tag{3.2}$$

denn das größte Elemente muss auch beschränkt sein, wenn der Abstand zweier beliebiger Elemente beschränkt ist. Da wir uns in der nichtparametrischen Regressionsschätzung befinden und dafür als Bedingung $\mathbb{E}[Y^2] < \infty$ gelten muss, wählen wir mit dem bisher gezeigten:

$$c_{21} = \max\left\{ \frac{1+\mathbb{E}[Y^2]}{c_3}, (N+1)^d \cdot \max\left\{ \left| \frac{1}{\mathbf{j}!} \cdot \partial^\mathbf{j} m(x_\mathbf{i}) \right|^2 \mid \mathbf{j} \in [q]^d, |\mathbf{j}|_1 \leq q \right\} \right\}. \tag{3.3}$$

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 $f_{\text{net},\mathbf{j},\mathbf{i}} \in \mathscr{F}$ ist, folgt mit (3.3), dass g_n in $\mathscr{F}^{(J_n)}$ liegt. Sei A_n das Event, dass

$$\frac{1}{n}\sum_{i=1}^{n} Y_i^2 \leq 1 + \mathbb{E}[Y^2] \tag{3.4}$$

gilt. Wir wissen, dass aufgrund der Unabhängigkeit der $\mathbb{R}^d \times \mathbb{R}$-wertigen Zufallsvariablen $(X,Y),(X_1,Y_1),(X_2,Y_2),\dots$ mit

$$\begin{aligned}
\mathbb{P}(Y_1 \in \mathbb{R},\dots,Y_n \in R) &= \mathbb{P}((X_1,Y_1) \in \mathbb{R}^d \times \mathbb{R},\dots,(X_n,Y_n) \in \mathbb{R}^d \times \mathbb{R}) \\
&= \mathbb{P}((X_1,Y_1) \in \mathbb{R}^d \times \mathbb{R}) \cdots \mathbb{P}((X_n,Y_n) \in \mathbb{R}^d \times \mathbb{R}) \\
&= \mathbb{P}(Y_1 \in \mathbb{R}) \cdots \mathbb{P}(Y_n \in R),
\end{aligned}$$

und durch

$$\begin{aligned}
\mathbb{P}(Y_1 \in \mathbb{R},\dots,Y_n \in R) &= \mathbb{P}((X_1,Y_1) \in \mathbb{R}^d \times \mathbb{R},\dots,(X_n,Y_n) \in \mathbb{R}^d \times \mathbb{R}) \\
&= \mathbb{P}((X_1,Y_1) \in \mathbb{R}^d \times \mathbb{R}) \cdots \mathbb{P}((X_n,Y_n) \in \mathbb{R}^d \times \mathbb{R}) \\
&= \mathbb{P}((X,Y) \in \mathbb{R}^d \times \mathbb{R})^n \\
&= \mathbb{P}(Y \in \mathbb{R})^n
\end{aligned}$$

dass die Zufallsvariablen $Y_1,\dots,Y_n$ auch unabhängig und identisch verteilt sind. Daraus folgern wir $\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n Y_i^2\right] = \mathbb{E}[Y^2]$ mit der Linearität des Erwartungswerts. Mit Hilfe der Monotonie und Homogenität der Wahrscheinlichkeitsfunktion $\mathbb{P}$ und der Chebyshev'sche Ungleichung für $\varepsilon = 1$ ([Kle13], Satz 5.11) erhalten wir:

$$\begin{aligned}
\mathbb{P}(A_n^{\mathsf{c}}) &= \mathbb{P}(\frac{1}{n}\sum_{i=1}^{n} Y_i^2 - \mathbb{E}[Y^2] \geq 1) \\
&\leq \mathbb{P}(\left|\frac{1}{n}\sum_{i=1}^{n} Y_i^2 - \mathbb{E}[Y^2]\right| \geq 1) \\
&\leq \mathbb{V}[\frac{1}{n}\sum_{i=1}^{n} Y_i^2] \\
&= \frac{n \cdot \mathbb{V}[Y^2]}{n^2} \\
&= \frac{\mathbb{V}[Y^2]}{n} \\
&\leq \frac{c_{22}}{n},
\end{aligned} \tag{3.5}$$

wobei wir bei der letzten Gleichheit die identische Verteiltheit der $Y_1,\dots,Y_n$ und Rechenregeln für die Varianz verwendet haben welche wir unter anderem aufgrund der

Unabhängigkeit der $Y_1,\dots,Y_n$ verwenden durften. Sei $\hat{m}_n = T_{\beta_n}\tilde{m}_n = m_n$ im Falle dass Ereignis A_n gilt und andernfalls $\hat{m}_n = T_{\beta_n} g_n$. Durch die Unabhängigkeit von A_n zu den Zufallsvariablen $X, X_1, \dots, X_n$ und der Jensenschen Ungleichung ([Kle13], Satz 7.9) erhalten wir:

$$\begin{aligned}
\mathbb{E}\left[\int |m_n(x)-m(x)|^2 \mathbb{P}_X(dx)\mathbb{1}_{A_n^c}\right] &= \mathbb{E}\left[\int |m_n(x)-m(x)|^2 \mathbb{P}_X(dx)\right]\cdot \mathbb{P}(A_n^c) \\
&\leq \mathbb{E}\left[2m_n(x)^2 + 2m(x)^2 \mathbb{P}_X(dx)\right]\cdot \mathbb{P}(A_n^c) \\
&\leq \mathbb{E}\left[2\beta_n^2 + 2\beta_n^2 \mathbb{P}_X(dx)\right]\cdot \mathbb{P}(A_n^c) \\
&= 4\beta_n^2 \cdot \mathbb{P}(A_n^c),
\end{aligned} \tag{3.6}$$

wobei wir bei der letzten Ungleichung verwendet haben dass wir anfangs angenommen haben, dass $\|m\|_\infty < \beta_n$ und damit für c_6 und n hinreichend groß auch $\tilde{m}_n \leq \beta_n$ und nach der Definition von m_n zudem $m_n \leq \beta_n$ gilt. Bei der letzten Gleichung habe wir dann schließlich noch verwendet dass β_n deterministisch und $\mathbb{P}(X \in supp(\mathbb{P}_X)) = 1$ ist. Durch unsere Definition von $\hat{m}_n$ erhalten wir durch die Monotonie des Erwartungswert und der Abschätzung durch den ganzen Raum:

$$\begin{aligned}
\mathbb{E}\left[\int |m_n(x)-m(x)|^2 \mathbb{P}_X(dx)\mathbb{1}_{A_n}\right] &= \mathbb{E}\left[\int |\hat{m}_n(x)-m(x)|^2 \mathbb{P}_X(dx)\mathbb{1}_{A_n}\right] \\
&\leq \mathbb{E}\left[\int |\hat{m}_n(x)-m(x)|^2 \mathbb{P}_X(dx)\right].
\end{aligned} \tag{3.7}$$

Zusammen mit (3.5), (3.6), (3.7) und der Linearität des Erwartungswerts erhalten wir dann:

$$\begin{aligned}
\mathbb{E}\int |m_n(x)-m(x)|^2 \mathbb{P}_X(dx) &= \mathbb{E}\left[\int |m_n(x)-m(x)|^2 \mathbb{P}_X(dx)\cdot(\mathbb{1}_{A_n^c}+\mathbb{1}_{A_n})\right] \\
&= \mathbb{E}\left[\int |m_n(x)-m(x)|^2 \mathbb{P}_X(dx)\mathbb{1}_{A_n^c}\right] \\
&\quad + \mathbb{E}\left[\int |m_n(x)-m(x)|^2 \mathbb{P}_X(dx)\mathbb{1}_{A_n}\right] \\
&\leq 4\beta_n^2 \cdot \mathbb{P}(A_n^c) + \mathbb{E}\left[\int |\hat{m}_n(x)-m(x)|^2 \mathbb{P}_X(dx)\right] \\
&\leq \frac{4\cdot c_{22}\cdot \beta_n^2}{n} + \mathbb{E}\left[\int |\hat{m}_n(x)-m(x)|^2 \mathbb{P}_X(dx)\right].
\end{aligned} \tag{3.8}$$

Nach (2.10) 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 $f_j \in \mathscr{F}$ und $\hat{a}_j$ welche

$$\begin{aligned}\frac{c_3}{n}\sum_{j=1}^{J_n}\hat{a}_j^2 &= \frac{c_3}{n}\sum_{\mathbf{i}\in[M]^d}\sum_{\substack{\mathbf{j}\in[q]^d\\ |\mathbf{j}|_1\le q}} a_{\mathbf{i},\mathbf{j}}^2 \\ &\le \frac{1}{n}\sum_{i=1}^{n}|Y_i-\tilde{m}_n(X_i)|^2 + \frac{c_3}{n}\sum_{\mathbf{i}\in[M]^d}\sum_{\substack{\mathbf{j}\in[q]^d\\ |\mathbf{j}|_1\le q}} a_{\mathbf{i},\mathbf{j}}^2 \\ &\le \sum_{i=1}^{n} Y_i^2,\end{aligned}$$

erfüllen, wobei wir bei der letzten Ungleichung wie in (2.11) die minimierende Eigenschaft von $a_{\mathbf{i},\mathbf{j}}$ verwendet haben und zum Schluss die Koeffizienten Null gesetzt haben. Da $c_3 > 0$ ist, erhalten wir dass die Koeffizienten $\hat{a}_j$ die Eigenschaft

$$\sum_{j=1}^{J_n}\hat{a}_j^2 \le \frac{1}{n}\sum_{i=1}^{n} Y_i^2 \cdot \frac{n}{c_3}$$

erfüllen müssen. Auf A_n erhalten wir dann:

$$\sum_{j=1}^{J_n}\hat{a}_j^2 \overset{(3.4)}{\le} \frac{1+\mathbb{E}[Y^2]}{c_3}\cdot n \overset{(3.3)}{\le} c_{21}\cdot n,$$

woraus durch $f_j \in \mathscr{F}$ dann $\tilde{m}_n \in \mathscr{F}^{(J_n)}$ folgt. Deswegen nehmen wir nun ohne Beschränkung der Allgemeinheit $\hat{m}_n = T_{\beta_n}\bar{m}_n$ für $\bar{m}_n \in \{\tilde{m}_n, g_n\} \subseteq \mathscr{F}^{(J_n)}$ an. Da $c_3 > 0$ ist, erhalten wir:

$$\begin{aligned}&\frac{1}{n}\sum_{i=1}^{n}|Y_i-g_n(X_i)|^2 \\ &\le \frac{1}{n}\sum_{i=1}^{n}|Y_i-g_n(X_i)|^2 + \frac{c_3}{n}\cdot\sum_{\mathbf{i}\in[M]^d}\sum_{\substack{\mathbf{j}\in[q]^d\\ |\mathbf{j}|_1\le q}}\left|\frac{1}{\mathbf{j}!}\cdot\partial^{\mathbf{j}}m(x_{\mathbf{i}})\right|^2 \\ &\overset{(3.3)}{\le} \frac{1}{n}\sum_{i=1}^{n}|Y_i-g_n(X_i)|^2 + \frac{c_3}{n}\cdot\sum_{\mathbf{i}\in[M]^d} c_{21} \qquad (3.9)\\ &= \frac{1}{n}\sum_{i=1}^{n}|Y_i-g_n(X_i)|^2 + \frac{c_3\cdot c_{21}\cdot(M_n+1)^d}{n} \\ &= \frac{1}{n}\sum_{i=1}^{n}|Y_i-g_n(X_i)|^2 + c_{23}\cdot\frac{(M_n+1)^d}{n}.\end{aligned}$$

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 (2.11) 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:

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n} |Y_i - \tilde{m}_n(X_i)|^2 \\
&\leq \frac{1}{n}\sum_{i=1}^{n} |Y_i - \tilde{m}_n(X_i)|^2 + \frac{c_3}{n} \cdot \sum_{\mathbf{i}\in[M]^d} \sum_{\substack{\mathbf{j}\in[N]^d \\ |\mathbf{j}|_1 \leq N}} a_{\mathbf{i},\mathbf{j}}^2 \\
&\leq \frac{1}{n}\sum_{i=1}^{n} |Y_i - g_n(X_i)|^2 + \frac{c_3}{n} \cdot \sum_{\mathbf{i}\in[M]^d} \sum_{\substack{\mathbf{j}\in[q]^d \\ |\mathbf{j}|_1 \leq q}} \left| \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(x_{\mathbf{i}}) \right|^2 \\
&= \frac{1}{n}\sum_{i=1}^{n} |Y_i - g_n(X_i)|^2 + c_{23} \cdot \frac{(M_n+1)^d}{n}
\end{aligned} \tag{3.10}$$

Mit (3.9) und (3.10) erhalten wir zusammen:

$$\frac{1}{n}\sum_{i=1}^{n} |Y_i - \bar{m}_n(X_i)|^2 \leq \frac{1}{n}\sum_{i=1}^{n} |Y_i - g_n(X_i)|^2 + c_{23} \cdot \frac{(M_n+1)^d}{n}. \tag{3.11}$$

Da g_n nach Definition deterministisch, damit also unabhängig von $(X_1,Y_1),(X_2,Y_2),\dots$ ist, sind mit $|\Theta_n| = 1$, $g_{n,1} = g_n$, der Abschätzung (3.11) für $\hat{m}_n = \mathscr{T}_{\beta_n}\bar{m}_n$ mit $\hat{m}_n \in \mathscr{F}^{(J_n)}$ und dem penalty Term $pen_n(g_{n,1}) = c_{23} \cdot \frac{(M_n+1)^d}{n} > 0$ die Voraussetzungen für Lemma 1.13 erfüllt und wir erhalten durch dessen Anwendung:

$$\begin{aligned}
&\mathbb{E}\int |\hat{m}_n(x) - m(x)|^2 \mathbb{P}_X(dx) \\
&\leq \frac{c_{23}\cdot\log(n)^2 \cdot \left(\log\left(\sup_{x_1^n\in(supp(X))^n} \mathscr{N}_1\left(\frac{1}{n\cdot\beta_n}, \mathscr{F}^{(J_n)}, x_1^n\right)\right)+1\right)}{n} \\
&\quad + 2\cdot\mathbb{E}\left(\min_{l\in\Theta_n}\int |g_{n,l}(x) - m(x)|^2\mathbb{P}_X(dx) + c_{23}\cdot\frac{(M_n+1)^d}{n}\right) \\
&= \frac{c_{23}\cdot\log(n)^2 \cdot \left(\log\left(\sup_{x_1^n\in(supp(X))^n} \mathscr{N}_1\left(\frac{1}{n\cdot\beta_n}, \mathscr{F}^{(J_n)}, x_1^n\right)\right)+1\right)}{n} \\
&\quad + 2\int |g_n(x) - m(x)|^2\mathbb{P}_X(dx) + 2\cdot c_{23}\cdot\frac{(M_n+1)^d}{n},
\end{aligned}$$

wobei wir bei der letzten Gleichheit verwendet haben, dass der letzte Summand deterministisch ist. Zudem wissen wir, dass c_{23} unabhängig von n ist und $n > 1$, da wir n hinreichend groß wählen. Als nächstes überprüfen wir die Voraussetzungen von Lemma 1.14 um damit dann die letzte Gleichung weiter abzuschätzen. Nach Voraussetzung ist $\beta_n = c_6 \cdot \log(n)$ und $a_n = (\log n)^{1/(6(N+d))}$. Damit ist $a_n > 0$ für hinreichend großes n. Nach Voraussetzung ist zudem $d,N,J_n \in \mathbb{N}$ und es gilt nach (3.1)

$$J_n \leq (M_n+1)^d \cdot (N+1)^d \leq n^{c_{14}},$$

für hinreichend großes n. Wir betrachten hier den logistischen squasher welcher nach Lemma 1.6 insbesondere 2-zulässig nach Definition 1.3 ist. Da die hier betrachtete Menge von Funktionen $\mathscr{F}^{(J_n)}$ identisch mit der aus Lemma 1.14 ist, sind nun alle Voraussetzungen erfüllt. Da wir ohne Beschränkung der Allgemeinheit angenommen haben, dass $supp(X) \subseteq [-a_n, a_n]^d$ ist, erhalten wir mit Lemma 1.14 für hinreichend großes n:

$$\begin{aligned}
&\frac{c_{23} \cdot \log(n)^2 \cdot \left(\log\left(\sup_{x_1^n \in (supp(X))^n} \mathscr{N}_1\left(\frac{1}{n \cdot \beta_n}, \mathscr{F}^{(J_n)}, x_1^n\right)\right) + 1\right)}{n} \\
&\leq \frac{c_{23} \cdot \log(n)^2 \cdot \left(\left(c_{19} \cdot \log(n) \cdot (M_n+1)^d \cdot (N+1)^d\right) + 1\right)}{n} \\
&\leq \frac{c_{23} \cdot \log(n)^2 \cdot \left(2 \cdot c_{19} \cdot \log(n) \cdot (M_n+1)^d \cdot (N+1)^d\right)}{n} \\
&\leq c_{24} \cdot \frac{\log(n)^3 \cdot (M_n+1)^d \cdot (N+1)^d}{n}.
\end{aligned} \tag{3.12}$$

Sei

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

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

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

und zusammen mit einer Nulladdition, der Linearität des Integrals und der Supremumseigenschaft erhalten wir:

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

wobei wir im letzten Schritt $supp(X) \subseteq [-a_n, a_n]^d$ und $\mathbb{P}(X \in supp(\mathbb{P}_X)) = 1$ verwendet haben. Um die letzten beiden Summanden weiterhin abzuschätzen möchten wir Lemma 1.12 anwenden. Dafür überprüfen wir ob dafür alle Voraussetzungen erfüllt sind. Wir betrachten den logistischen squasher, welcher nach Lemma 1.9 insbesondere 2-zulässig ist.

Zudem ist für hinreichend großes n die Bedingung für R erfüllt und da unser neuronales Netz (2.9) mit $x_{\mathbf{i}} \in [-a_n, a_n]^d$ identisch mit der Definition aus Lemma 1.12 ist, sind alle Voraussetzungen erfüllt. Wir erhalten damit für $x \in [-a_n, a_n]^d$ und n hinreichend groß:

$$\begin{aligned}
|g_n(x) - P_n(x)| &= \left| \sum_{\mathbf{i} \in [M_n]^d} \sum_{\substack{\mathbf{j} \in [q]^d \\ |\mathbf{j}|_1 \leq q}} \frac{1}{\mathbf{j}!} \cdot \partial^{\mathbf{j}} m(x_{\mathbf{i}}) \right| \cdot \left| f_{\text{net},\mathbf{j},\mathbf{i}}(x) - (x - x_{\mathbf{i}})^{\mathbf{j}} \cdot \prod_{j=1}^{d} (1 - \frac{M_n}{2a_n} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|)_+ \right| \\
&\leq (M_n + 1)^d \cdot (q+1)^d \cdot e \cdot \left| f_{\text{net},\mathbf{j},\mathbf{i}}(x) - (x - x_{\mathbf{i}})^{\mathbf{j}} \cdot \prod_{j=1}^{d} (1 - \frac{M_n}{2a_n} \cdot |x^{(j)} - x_{\mathbf{i}}^{(j)}|)_+ \right| \\
&\leq (M_n + 1)^d \cdot (q+1)^d \cdot e \cdot \hat{c}_{12} \cdot 3^{3 \cdot 3^s} \cdot a_n^{3 \cdot 2^s} \cdot M_n^3 \cdot \frac{1}{R_n} \\
&\leq (M_n + 1)^d \cdot (q+1)^d \cdot c_{25} \cdot a_n^{3 \cdot (N+d) \cdot 2} \cdot \frac{M_n^3}{R_n} \\
&= (M_n + 1)^d \cdot (q+1)^d \cdot c_{25} \cdot \log(n) \cdot \frac{M_n^3}{R_n},
\end{aligned} \tag{3.14}$$

wobei wir unter anderem verwendet haben, 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 dann noch die Definition von a_n eingesetzt. Da nach Konstruktion $a > 0$, m (p, C)-glatt und $P_n(x)$ nach Lemma 2.1 eine lokale Konvexkombination von Taylorpolynomen von m ist, erhalten wir mit Lemma 2.2:

$$|P_n(x) - m(x)| \leq c_{26} \cdot \frac{a_n^p}{M_n^p} \leq c_{26} \cdot \log(n) \cdot \frac{1}{M_n^p}, \tag{3.15}$$

wobei wir verwendet haben, dass für $p = q + s$ für hinreichend großes n gilt:

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

da nach Voraussetzung $N \geq q$ und $d \geq s$ mit $s \in (0, 1]$ ist. Durch Quadrieren bleiben die Ungleichungen auch erhalten und da die rechten Seiten von Ungleichung (3.14) und (3.15) nicht von x abhängen, gelten die Ungleichung ebenfalls für das Supremum. Durch

einsetzen der Definitionen von M_n und R_n erhalten wir für n hinreichend groß:

$$\begin{aligned}
\sup_{x\in[-a_n,a_n]^d} |g_n(x)-P_n(x)|^2 &\leq \left((M_n+1)^d\cdot(q+1)^d\cdot c_{25}\cdot\log(n)\cdot\frac{M_n^3}{R_n}\right)^2 \\
&\leq c_{25}^2\cdot(M_n+1)^{2d}\cdot\log(n)^2\cdot\frac{M_n^6}{R_n^2} \\
&\leq c_{25}^2\cdot(M_n+1)^{2d}\cdot\log(n)^2\cdot\frac{(M_n+1)^{6d}}{R_n^2} \\
&\leq c_{25}^2\cdot\log(n)^2\cdot\frac{(M_n+1)^{8d}}{R_n^2} \\
&\leq c_{25}^2\cdot\frac{n^{\frac{8d}{2p+d}}}{n^{2d+8}}\cdot\log(n)^2 \\
&= c_{25}^2\cdot n^{\frac{8d}{2p+d}-2d-8}\cdot\log(n)^2 \\
&\leq c_{25}^2\cdot n^{\frac{8d}{2p+d}-8\frac{2p+d}{2p+d}}\cdot\log(n)^2 \\
&= c_{25}^2\cdot n^{-\frac{16p}{2p+d}}\cdot\log(n)^2 \\
&\leq c_{25}^2\cdot n^{-\frac{2p}{2p+d}}\cdot\log(n)^3,
\end{aligned} \tag{3.16}$$

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:

$$\begin{aligned}
\sup_{x\in[-a_n,a_n]^d} |P_n(x)-m(x)|^2 &\leq \left(c_{26}\cdot\log(n)\cdot\frac{1}{M_n^p}\right)^2 \\
&\leq c_{26}^2\cdot\log(n)^2\cdot c_5^{-2p}\cdot n^{-\frac{2p}{2p+d}} \\
&\leq (c_{16}^2\cdot c_5^{-2p})\cdot\log(n)^3\cdot n^{-\frac{2p}{2p+d}}.
\end{aligned} \tag{3.17}$$

Mit analogem Vorgehen erhalten wir für (3.12):

$$\begin{aligned}
&\frac{c_{23}\cdot\log(n)^2\cdot\left(\log\left(\sup_{x_1^n\in(supp(X))^n}\mathscr{N}_1\left(\frac{1}{n\cdot\beta_n},\mathscr{F}^{(J_n)},x_1^n\right)\right)+1\right)}{n} \\
&\leq c_{24}\cdot\frac{\log(n)^3\cdot(M_n+1)^d\cdot(N+1)^d}{n} \\
&\leq \hat{c}_{24}\cdot\log(n)^3\cdot\frac{(c_5\cdot n^{\frac{1}{2p+d}}+2)^d}{n} \\
&\leq \hat{c}_{24}\cdot\log(n)^3\cdot\frac{c_5^d\cdot n^{\frac{d}{2p+d}}}{n} \\
&= \tilde{c}_{24}\cdot\log(n)^3\cdot n^{\frac{d}{2p+d}-1} \\
&= \tilde{c}_{24}\cdot\log(n)^3\cdot n^{-\frac{2p}{2p+d}}.
\end{aligned} \tag{3.18}$$

Zudem erhalten wir, da für n hinreichend groß $\log(n)^3 > 1$ gilt, mit analogem Vorgehen:

$$\begin{aligned} c_{23} \cdot \frac{(M_n+1)^d}{n} &\leq \tilde{c}_{23} \cdot \frac{n^{\frac{d}{2p+d}}}{n} \\ &\leq \tilde{c}_{23} \cdot \log(n)^3 \cdot n^{-\frac{2p}{2p+d}} \end{aligned} \tag{3.19}$$

und

$$\begin{aligned} \frac{4 \cdot c_{22} \cdot \beta_n^2}{n} &\leq \tilde{c}_{22} \cdot \log(n)^3 \cdot n^{-1} \\ &\leq \tilde{c}_{22} \cdot \log(n)^3 \cdot n^{-\frac{2p}{2p+d}}. \end{aligned} \tag{3.20}$$

Nun haben wir alle Summanden von (3.8) abgeschätzt und erhalten schließlich mit mit (3.14) - (3.20):

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

mit

$$c_{fin} = \tilde{c}_{22} + 2 \cdot \tilde{c}_{23} + \tilde{c}_{24} + 2 \cdot (2 \cdot c_{16}^2 \cdot c_5^{-2p} + 2 \cdot c_{25}^2),$$

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. □

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 $m_j \colon [-2,2]^d \to \mathbb{R} \quad (j \in \{1,2\})$ 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:

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

und

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

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 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 komplett 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 korrespondiere 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 $\{1,2,3\} \cup \{4,8,12,16,\ldots,4 \cdot \lfloor \frac{n_l}{4} \rfloor\}$ ausgewählt wird. Unser letzter Schätzer *new_neural_network_estimate* ist unser hier vorgestelltes neuronale Netze Regressionsschätzer. 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	
noise	5%	10%	5%	10%
$\bar{\varepsilon}_{L_2,N}(avg)$				
approach	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 1 - Differential- und Integralrechnung einer Veränderlichen*. Springer, Berlin, 2016.

[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.

[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 Ouputs 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`

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 23 15:08:26 2019

@author: adrian

Main Datei die die Simulation und damit den Vergleich der implementierten Schätzer
durchführt.
"""
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
import pandas as pd
import tikzplotlib
from scipy.stats import iqr
from data_gen import gen_data_Y
from constant import constant_estimate
from new_neural_network import new_neural_network_estimate
from nearest_neighbor import nearest_neighbor_estimate
from fc_neural_network import fc_neural_1_estimate

n = 10000
n_train = n * 0.8
n_test = n * 0.2

```

```
sigma = 0.05

'''
EINDIMENSIONALER FALL (d = 1) wird geplottet
'''
N = 3
q = 2
R = 10 ** 6
a = 2
M = 2
d = 1

#sigma = 0.05

#n_train = 100
#n_test = 2000
# Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
m_X_train, Y_train = gen_data_Y(X_train,sigma)

X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))

Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a,)
#Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train, Y_train, X_test)
#Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train, Y_train, X_test)
m_X_test, dummy = gen_data_Y(X_test,sigma)


#plt.plot(X_test, Y_pred_nearest_neighbor, '-r', label='nearest_neigbhor')
#plt.plot(X_test, Y_pred_fc_nn_1, '-g', label='fc_nn_1')
#colors = (0,0,0)
area = 4
plt.scatter(X_test, m_X_test, s=area, color = 'blue', label='m_1', alpha=0.5)
plt.scatter(X_test, Y_pred_new_nn, s=area, color = 'red', label='new_neural_network_estimate', alpha=0.5)
plt.title('...')
plt.legend(loc='upper left')
plt.xlabel('x')
plt.ylabel('y')
#plt.savefig('graph_d_1.png')
tikzplotlib.save("mytikz_d1.tex")
#plt.show()
#plt.plot(X_test, Y_pred_new_nn, 'ro-', label='new_nn')
#plt.plot(X_test, m_X_test, '-b', label='m_d')

#plt.xlim(-2, 2)
#plt.xlim(-2,2)
#plt.show()


'''
ZEIDIMENSIONALER FALL (d = 2) wird geplottet
'''
N = 2
q = 2
R = 10 ** 6
a = 2
M = 2
d = 2

#sigma = 0.05

#n_train = 100
#n_test = 2000
# Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
m_X_train, Y_train = gen_data_Y(X_train,sigma)

X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))

Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a,)
#Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train, Y_train, X_test)
#Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train, Y_train, X_test)
m_X_test, dummy = gen_data_Y(X_test,sigma)


```

```
102
103 x = np.ravel(X_test[:,0])
104 y = np.ravel(X_test[:,1])
105
106 # so wie es sein soll
107 #z = m_X_test[:,0]
108 # was der SChätzer auswirft
109 z_new = Y_pred_new_nn[:,0]
110
111 ax = plt.axes(projection='3d')
112 ax.scatter(x, y, z_new, c=z_new, cmap='viridis', linewidth=0.5);
113 ax.view_init(40, 20)
114 plt.savefig('graph_d_2_new_estimate.png')
115
116 # so wie es sein soll
117 z = m_X_test[:,0]
118 # was der Schätzer auswirft
119
120 ax = plt.axes(projection='3d')
121 ax.scatter(x, y, z, c=z, cmap='viridis', linewidth=0.5);
122 ax.view_init(40, 20)
123 plt.savefig('test.png')
124 #tikzplotlib.save("mytikz_d2.tex")
125
126 postpro = np.asarray([ X_test[:,0], X_test[:,1], Y_pred_new_nn[:,0] ])
127 np.savetxt("plotpostpro.csv", np.transpose(postpro), delimiter=",")
128
129 #plt.savefig('graph_d_2_m_2.png')
130
131 '''
132 ein Vergleich des emp. L2 Fehler gemacht für d = 1
133 '''
134 #n_train = 100
135 #n_test = 2000
136 # Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
137
138 N = 3
139 q = 2
140 R = 10 ** 6
141 a = 2
142 M = 2
143 d = 1
144
145 #sigma = 0.05
146
147 scaled_error = np.empty((5, 3,))
148 scaled_error[:] = np.nan
149
150 e_L2_avg = np.zeros(5)
151 e_L2_avg[:] = np.nan
152
153 for i in range(0,50,1):
154
155     X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
156     m_X_train, Y_train = gen_data_Y(X_train,sigma)
157
158     X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))
159
160     #Y_pred_constant = constant_estimate(Y_train)
161     Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a,)
162     Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train, Y_train, X_test)
163     Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train, Y_train, X_test)
164
165     m_X_test, not_needed = gen_data_Y(X_test,sigma)
166
167     e_L2_new_nn = np.mean(abs(Y_pred_new_nn - m_X_test) ** 2)
168     e_L2_fc_nn_1 = np.mean(abs(Y_pred_fc_nn_1 - m_X_test) ** 2)
169     e_L2_nearest_neighbor = np.mean(abs(Y_pred_nearest_neighbor - m_X_test) ** 2)
170
171     for j in range(0,25,1):
172
173         X = np.random.uniform(low=-2,high=2,size=(n_test,d))
174         m_X, Y = gen_data_Y(X,sigma)
175         Y_pred_constant = constant_estimate(Y)
176
```

```
        e_L2_avg[j] = np.mean(abs(Y_pred_constant - m_X) ** 2)

    scaled_error[i,0] = e_L2_new_nn / np.median(e_L2_avg)
    scaled_error[i,1] = e_L2_fc_nn_1 / np.median(e_L2_avg)
    scaled_error[i,2] = e_L2_nearest_neighbor / np.median(e_L2_avg)

iqr_new_nn = iqr(scaled_error[:,0])
iqr_fc_nn_1 = iqr(scaled_error[:,1])
iqr_nearest_neighbor = iqr(scaled_error[:,2])

median_new_nn = np.median(scaled_error[:,0])
median_fc_nn_1 = np.median(scaled_error[:,1])
median_nearest_neighbor = np.median(scaled_error[:,2])

rows = ["noise","e_L2_avg","approach","new_nn", "fc_nn_1", "nearest_neighbor"]

series_noise_1 = pd.Series([repr(sigma)+'%',np.median(e_L2_avg),"(Median, IQR)",(median_new_nn, iqr_new_nn), (median_
    fc_nn_1, iqr_fc_nn_1), (median_nearest_neighbor, iqr_nearest_neighbor)], index=rows)
series_noise_1.name = ""
#series_noise_2 = pd.Series([repr(sigma)+'%',np.median(e_L2_avg),"(Median, IQR)",(median_new_nn, iqr_new_nn), (median
    _fc_nn_1, iqr_fc_nn_1), (median_nearest_neighbor, iqr_nearest_neighbor)], index=rows)
#series_noise_2.name = ""

error_df = pd.concat([series_noise_1], axis=1)
#print(error_df)
error_df.to_csv('out_d_1.csv',index = True)

'''
ein Vergleich des emp. L2 Fehler gemacht für d = 2
'''
n_train = 100
n_test = 200
# Parameter für unseren neuen Neuronale-Netze-Regressionschätzer

N = 2
q = 2
R = 10 ** 6
a = 2
M = 2
d = 2

sigma = 0.1

scaled_error = np.empty((5, 3,))
scaled_error[:] = np.nan

e_L2_avg = np.zeros(5)
e_L2_avg[:] = np.nan

for i in range(0,5,1):

    X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
    m_X_train, Y_train = gen_data_Y(X_train,sigma)

    X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))

    #Y_pred_constant = constant_estimate(Y_train)
    Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a,)
    Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train, Y_train, X_test)
    Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train, Y_train, X_test)

    m_X_test, not_needed = gen_data_Y(X_test,sigma)

    e_L2_new_nn = np.mean(abs(Y_pred_new_nn - m_X_test) ** 2)
    e_L2_fc_nn_1 = np.mean(abs(Y_pred_fc_nn_1 - m_X_test) ** 2)
    e_L2_nearest_neighbor = np.mean(abs(Y_pred_nearest_neighbor - m_X_test) ** 2)

    for j in range(0,5,1):

        X = np.random.uniform(low=-2,high=2,size=(n_test,d))
        m_X, Y = gen_data_Y(X,sigma)
        Y_pred_constant = constant_estimate(Y)

        e_L2_avg[j] = np.mean(abs(Y_pred_constant - m_X) ** 2)

```

```
250     scaled_error[i,0] = e_L2_new_nn / np.median(e_L2_avg)
251     scaled_error[i,1] = e_L2_fc_nn_1 / np.median(e_L2_avg)
252     scaled_error[i,2] = e_L2_nearest_neighbor / np.median(e_L2_avg)
253
254 iqr_new_nn = iqr(scaled_error[:,0])
255 iqr_fc_nn_1 = iqr(scaled_error[:,1])
256 iqr_nearest_neighbor = iqr(scaled_error[:,2])
257
258 median_new_nn = np.median(scaled_error[:,0])
259 median_fc_nn_1 = np.median(scaled_error[:,1])
260 median_nearest_neighbor = np.median(scaled_error[:,2])
261
262 rows = ["noise","e_L2_avg","approach","new_nn", "fc_nn_1", "nearest_neighbor"]
263
264 series_noise_1 = pd.Series([repr(sigma)+'%',np.median(e_L2_avg),"(Median, IQR)",(median_new_nn, iqr_new_nn), (median_
        fc_nn_1, iqr_fc_nn_1), (median_nearest_neighbor, iqr_nearest_neighbor)], index=rows)
265 series_noise_1.name = ""
266 #series_noise_2 = pd.Series([repr(sigma)+'%',np.median(e_L2_avg),"(Median, IQR)",(median_new_nn, iqr_new_nn), (median
        _fc_nn_1, iqr_fc_nn_1), (median_nearest_neighbor, iqr_nearest_neighbor)], index=rows)
267 #series_noise_2.name = ""
268
269 error_df = pd.concat([series_noise_1], axis=1)
270 #print(error_df)
271 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 \R^d für i = 1,...,n
44 # sigma: Schwankung in den Werten (Noise) \in \{0.05,0.1\}
45
46 def gen_data_Y (X, sigma):
47
```

```
    n = np.size(X, 0)
    d = np.size(X, 1)

    m_X = np.zeros((n,1,))
    m_X[:] = np.nan

    S = np.random.standard_normal(size=(n,1))
    for t in range(0,n):
        m_X[t] = m_d(X[t], d)

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

Listing 4.3: `help_neural_networks.py`

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 15 11:22:02 2019

@author: adrian

Implementation von Neuronalen-Netzen welche wir für die Konstruktion unseres
Neuronale-Netze-Regressionschätzers benötigen
"""
import numpy as np

# Sigmoidfunktion
#
# x: x \in \R

def sigmoid (x):

    return 1 / (1 + np.exp(-x))

# Neuronales Netz welches die Funktion f(x) = x approximiert
#
# x: reelle Zahl
# R: reelle Zahl >= 1

def f_id (x, R):

    return 4 * R * sigmoid(x / R) - 2 * R

# Neuronales Netz welches die Funktion f(x, y) = x * y approximiert
#
# x: reelle Zahl
# y: reelle Zahl
# R: reelle Zahl >= 1

def f_mult (x, y, R):

    return (((R ** 2) / 4) * (((1 + np.exp(-1)) ** 3) / (np.exp(-2) - np.exp(-1)))) \
    * (sigmoid(((2 * (x + y)) / R) + 1) - 2 * sigmoid(((x + y) / R) + 1) \
    - sigmoid(((2 * (x - y)) / R) + 1) + 2 * sigmoid(((x - y) / R) + 1))

# Neuronales Netz welches die Funktion f(x) = max(x,0) approximiert
#
# x: reelle Zahl
# R: reelle Zahl >= 1

def f_relu (x, R):

    return f_mult(f_id(x, R), sigmoid(R * x),R)

# Neuronales Netz welches die Funktion f(x) = max(1 - (M/(2 * a)) * abs(x - y),0) approximiert
#
# x: reelle Zahl
# y: fixe reelle Zahl
# R: reelle Zahl >= 1
# M: fixe natürliche Zahl
# a: fixe Zahl > 0

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))^j1 * ... *
19 # (x^(d) - x_ik^(d))^jd * \prod_{j = 1}^d max((1 - (M/2a) * abs(x^(j) - x_ik^(j))),0)
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,...,j_d \in {0,1,...,N}
24 # X_i: Ist eine  d x (M+1)^d  Matrix.
25 # N: Natürliche Zahl >= q
26 # q:
27 # s: [log_2(N + d)]
28 # R: Zahl >= 1
29 # M: M \in 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_1_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_1_d[range(0, l - 1, 1)].sum() + 1
48         while k in range(j_1_d[range(0, l - 1, 1)].sum() + 1, j_1_d[range(0, l, 1)].sum() + 1, 1):
49             f_l_k[s, k] = f_id(f_id(x[l - 1] - X_i[l - 1], R), R)
50             k += 1
51
52     for l in range(s - 1, -1, -1):
53         for k in range((2 ** l), 0, -1):
54             f_l_k[l, k] = f_mult(f_l_k[l + 1, (2 * k) - 1], f_l_k[l + 1, 2 * k], R)
55
56     return f_l_k[0,1]
57
58 # Bestimmung der Gewichte der Ausgabeschicht durch lösen eines regularisierten
59 # Kleineste-Quadrate Problems
60 #
61 # X: Eingabevektoren der Form (X_1,...,X_n) für das Neuronale Netz aus dem Datensatz (X_1,Y_1),...,(X_n,Y_n)
62 # Y: Eingabevektoren der Form (Y_1,...,Y_n) für das Neuronale Netz aus dem Datensatz (X_1,Y_1),...,(X_n,Y_n)
63 # N: Natürliche Zahl >= q
64 # q:
65 # R: Zahl >= 1
66 # d: Ist die Dimension des Eingabevektors d > 0
67 # M: M \in \N
68 # a: >0
```

```
69
70 def output_weights(X, Y, N, q, R, d, M, a):
71
72     s = math.ceil(math.log2(N + d))
73
74     # Anzahl der Eingabevektoren X_1,...,X_n
75
76     n = np.size(X, 0)
77
78     # Eine beliebige constante > 0
79
80     #c_3 = np.random.randint(1,10)
81     c_3 = 0.01
82
83     # Anzahl der Spalten der Matrix für das Kleinste-Quadrate Problem
84     # In den Spalten sind die Funktionswerte von f_net eingespeichert
85
86     J = int(((1 + M) ** d) * scipy.special.binom(N + d, d))
87
88     # Für die Konstruktion der Matrix brauchen wir erstmal alle Inputparameter
89     # für f_net, da wir dort nur den Funktionswert für einen Vektor j_1,...,j_d einsetzen
90     # müssen wir erstmals alle möglichen Vektoren dieser Art konstruieren die die 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.identity(J), (1 / n) * np.dot(np.
           transpose(B),Y))
113    weights = np.linalg.solve(np.dot(np.transpose(B),B) + (c_3) * np.identity(J), np.dot(np.transpose(B),Y))
114
115    return (weights, J, all_j1_jd_by_cond, X_ik)
116
117 # Bestimmung des Funktionswert des Neuronale-Netzte-Regressionsschätzers
118 #
119 # x: Eingabe für einen Vektor der Form [-a,a]^d für den eine Schätzung bestimmt werden soll
120 # X: Eingabevektoren der Form (X_1,...,X_n) für das Neuronale Netz aus dem Datensatz (X_1,Y_1),...,(X_n,Y_n)
121 # Y: Eingabevektoren der Form (Y_1,...,Y_n) für das Neuronale Netz aus dem Datensatz (X_1,Y_1),...,(X_n,Y_n)
122 # N: Natürliche Zahl >= q
123 # q:
124 # s: [log_2(N + d)]
125 # R: Zahl >= 1
126 # d: Ist die Dimension des Eingabevektors d > 0
127 # M: M \in \N
128 # a: >0
129
130 def new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a):
131
132    Y_pred = np.empty((len(X_test), 1,))
133    Y_pred[:] = np.nan
134
135    s = math.ceil(math.log2(N + d))
136
137    weights, J, all_j1_jd_by_cond, X_ik = output_weights(X_train, Y_train, N, q, R, d, M, a)
138
139    F_net = np.empty((1, J,))
140    F_net[:] = np.nan
141
```

```
    for u in range (0,len(X_test),1):
        j = 0
        while j < J:
            for k in range(0, ((M + 1) ** d)):
                for z in range(0, int(scipy.special.binom(N + d, d))):
                    F_net[0,j] = f_net(X_test[u], d, all_j1_jd_by_cond[z], X_ik[k], N, q, s, R, M, a)
                    j += 1

        Y_pred[u] = np.sum(np.transpose(weights) * F_net)

    return Y_pred
```

Listing 4.5: `fc_neural_network.py`

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 11 14:23:15 2019

@author: adrian
"""
import numpy as np
from keras.models import Sequential
from keras.layers import Dense

# Fully-Connected Neuronales Netzt mit einer Verborgenen schicht welches die
# Anzahl der Neuronen adaptiv, durch minimierung des L2 fehlers, aus der Menge \{5, 10, 25, 50, 75\} auswählt.
#
# X: Eingabevektoren der Form (X_1,...,X_n) für das Neuronale Netz aus dem Datensatz (X_1,Y_1),...,(X_n,Y_n)
# Y: Eingabevektoren der Form (Y_1,...,Y_n) für das Neuronale Netz aus dem Datensatz (X_1,Y_1),...,(X_n,Y_n)

def fc_neural_1_estimate (X_train,Y_train,X_test):

    Ynew = np.empty((len(X_train), len([5,10,25,50,75]),))
    Ynew[:] = np.nan

    count = 0
    n_neurons = [5,10,25,50,75]

    d = np.size(X_train, 1)

    for j in n_neurons:
        model = Sequential()
        model.add(Dense(j, input_dim=d, activation='relu'))
        model.add(Dense(1, activation='linear'))
        model.compile(loss='mse', optimizer='adam')
        model.fit(X_train, Y_train, epochs=1000, verbose=0)

        Ynew[:,count] = model.predict(X_train)[:,0]
        count += 1

    Diff = Ynew[:] - Y_train[:]
    best_n_neurons = n_neurons[(1/len(X_train) *(Diff.sum(axis=0) ** 2)).argmin()]

    model = Sequential()
    model.add(Dense(best_n_neurons, input_dim=d, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mse', optimizer='adam')
    model.fit(X_train, Y_train, epochs=1000, verbose=1)

    return model.predict(X_test)
```

Listing 4.6: `nearest_neighbor.py`

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 11 11:14:47 2019

@author: adrian
K-Nearest-Neighbors Algorithm
"""
```

```
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 einer Liste von Anzahlen an
       Nachbarn die betrachtet werden
19 # sollen welches die beste Wahl ist.
20 #
21 # X: Inputvektor für das Kalibirieren des Modells
22 # Y: Inputvektor für das Kalibirieren des Modells (Zielvektor an den die Gewichte angepasst werden)
23 # T: Inputvektor für den eine Vorhersage bestimmte werden soll
24
25 def nearest_neighbor_estimate (X_train,Y_train,X_test):
26
27     params = {'n_neighbors':[2,3,4,5,6,7,8,9], 'weights': ['uniform', 'distance']}
28
29     knn = neighbors.KNeighborsRegressor()
30
31     knn_gridsearch_model = GridSearchCV(knn, params, cv=5)
32     knn_gridsearch_model.fit(X_train,Y_train)
33
34     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 \R für i = 1,...
15
16 def constant_estimate(Y):
17     m = np.zeros((len(Y),1,))
18     m[:] = mean(Y)
19     return m
```