



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	6
1 Grundlagen und Hilfsresultate	7
1.1 Definitionen	7
1.2 Hilfsresultate	7
2 Konstruktion des Neuronale Netze Regresssionsschätzers	12
3 Resultat zur Konvergenzgeschwindigkeit	13
4 Anwendungsbeispiel auf Simulierte Daten	14
Literaturverzeichnis	16
Appendix	17

Einleitung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue dui dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue dui dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer. Diese Arbeit orientiert sich an [EKT⁺07] und [Koh10].

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 vor allem für das Resultat der Konvergenzgeschwindigkeit des einfach berechenbaren Neuronale Netze Regressionschätzer benötigt werden.

1.1 Definitionen

Definition 1.1.1 (Stoppzeit, vgl. [Kle13] 9.15 und 9.16).

- (i) Eine *Stoppzeit* τ mit Werten in $[0, T]$ und $T \geq 0$ ist eine bezüglich $(X_t)_{0 \leq t \leq T}$ messbare Funktion. Zusätzlich muss für jedes $r \in [0, T]$ das Ereignis $\{\tau \leq r\}$ in der durch $(X_s)_{0 \leq s \leq r}$ erzeugten σ -Algebra $\mathcal{F}_r = \sigma((X_s)_{0 \leq s \leq r})$ enthalten sein. Die Klasse aller Stoppzeiten mit Werten im Intervall $[0, T]$ bezeichnen wir mit $\mathcal{T}([0, T])$.
- (ii) Ist $T \in \mathbb{N}_0$, so nennen wir eine bezüglich X_0, \dots, X_T messbare Funktion τ mit Werten in $\{0, 1, \dots, T\}$, *Stoppzeit*, wenn zusätzlich für jedes $r \in \{0, \dots, T\}$ das Ereignis $\{\tau = r\}$ in der durch X_0, \dots, X_r erzeugten σ -Algebra $\sigma(X_0, \dots, X_r)$ enthalten ist. Die Klasse aller Stoppzeiten mit Werten in $\{0, \dots, T\}$ bezeichnen wir mit $\mathcal{T}(0, \dots, T)$.

1.2 Hilfsresultate

Lemma 1.2.1. Sei $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ eine Funktion und $R, a > 0$.

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

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

für beliebige $x \in [-a, a]$:

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

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

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

für beliebige $x \in [-a, a]$:

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

Beweis. a) Sei $u = \frac{c}{R} + t_{\sigma,id}$, $\xi = 0$ und $x \in [-a, a]$ beliebig. Wir wissen, dass f_{id} 1-mal differenzierbar ist, da nach Voraussetzung σ 2-mal stetig differenzierbar ist, existiert nach der Restgliedformel von Lagrange (REFERENZ) ein $c \in [\xi, x]$, sodass mit Ausklammern von $\frac{R}{\sigma'(t_{\sigma,id})}$ folgt:

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

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) Folgt analog wie in a) durch 2-maliges Anwenden der Restgliedformel von Lagrange (REFERENZ) auf die Funktion f die hier nun 2-mal differenzierbar ist, da σ nach Voraussetzung 3-mal stetig differenzierbar ist.

□

Lemma 1.2.2. Sei $\sigma: \mathbb{R} \rightarrow [0, 1]$ nach Definition 1, 2-zulässig. Zudem sei $R > 0$ und $a > 0$ beliebig. Dann gilt für das neuronale Netz

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

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

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

Beweis. Durch Ausmultiplizieren erhalten wir

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

und

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

Aus diesen beiden Gleichungen folgt durch Ausklammern von $\frac{1}{4}$, der Homogenität des Betrags und der Anwendung der Dreiecksungleichung:

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

wobei bei der letzten Ungleichung verwendet haben, dass $a > 0$ nach Lemma 1.2.1b) beliebig gewählt wurde und daher insbesondere für beliebiges $x \in [-2a, 2a]$

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

gilt. □

Lemma 1.2.3. Sei $\sigma: \mathbb{R} \rightarrow [0, 1]$ nach Definition 1, 2-zulässig. Sei f_{mult} das neuronale Netz aus Lemma 1.2.2 und f_{id} das neuronale Netz aus Lemma 1.2.1. Angenommen es gilt

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

Dann erfüllt das neuronale Netz

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

für alle $x \in [-a, a]$:

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

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

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

und

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

Damit folgt aus der Homogenität des Betrags

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

Nach Lemma 1.2.1 und Lemma 1.2.2 gilt:

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

und

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

Da nach Voraussetzung $a \geq 1$ ist gilt insbesondere $[0, 1] \in [-2a, 2a]$ und daher gilt insbesondere $\sigma(x) \in [-2a, 2a]$. Zudem erhalten wir durch eine Nulladdition, das Anwenden der Dreiecksungleichung, die Verwendung von Lemma 1.2.1 und der Voraussetzung für R :

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

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

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

□

Kapitel 2

Konstruktion des Neuronale Netze Regresssionsschätzers

Kapitel 3

Resultat zur Konvergenzgeschwindigkeit

In diesem Kapitel stellen wir das Hauptresultat dieser Arbeit vor, ein Resultat zur Konvergenzgeschwindigkeit des in dieser Arbeit vorgestellten Neuronale Netze Regressions-schätzers.

Satz 3.0.1. *Unter obigen Voraussetzungen gilt für alle $t \in \{0, \dots, T\}$ und \mathbb{P}_{X_t} -f.s. für alle $x \in \mathbb{R}^d$*

$$V_t(x) = \mathbb{E}[f_{\tau_{t-1}^*}(X_{\tau_{t-1}^*}) \mid X_t = x] = \max\{f_t(x), q_t(x)\}. \quad (3.1)$$

Weiterhin gilt

$$\mathbf{V}_0 = \mathbb{E}[f_{\tau^*}(X_{\tau^*})]. \quad (3.2)$$

Beweis.

□

Kapitel 4

Anwendungsbeispiel auf Simulierte Daten

Literaturverzeichnis

- [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.
- [Gla13] GLASSERMAN, P.: *Monte Carlo methods in financial engineering*, Band 53 der Reihe *Stochastic Modelling and Applied Probability*. Springer, New York, 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 I*. Springer, Berlin, 2007.
- [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 TBD.
- data_gen.py TBD.
- help_neural_networks.py fasst alle Hilfsfunktion zusammen.
- new_neural_network.py TBD.
- fc_neural_network.py TBD.
- nearest_neighbor.py TBD.
- constant.py TBD.

Listing 4.1: main.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Wed Oct 23 15:08:26 2019
5
6  @author: adrian
7
8  Main Datei die die Simulation und damit den Vergleich der implementierten Schätzer
9  durchführt.
10 """
11 import numpy as np
12 from mpl_toolkits import mplot3d
13 import matplotlib.pyplot as plt
14 import pandas as pd
15 from scipy.stats import iqr
16 from data_gen import gen_data_Y, error_limit
17 from constant import constant_estimate
18 from new_neural_network import new_neural_network_estimate
19 from nearest_neighbor import nearest_neighbor_estimate
20 from fc_neural_network import fc_neural_l_estimate
21
22 '''
23 EINDIMENSIONALER FALL (d = 1) wird geplottet
24 '''
25 N = 3
26 q = 2
27 R = 10 ** 6
28 a = 2
29 M = 2
30 d = 1

```

```

31
32 sigma = 0.05
33
34 n_train = 100
35 n_test = 2000
36 # Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
37 X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
38 m_X_train , Y_train = gen_data_Y(X_train , sigma)
39
40 X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))
41
42 Y_pred_new_nn = new_neural_network_estimate(X_train , Y_train , X_test , N, q, R, d, M, a,)
43 #Y_pred_fc_nn_l = fc_neural_l_estimate(X_train , Y_train , X_test)
44 #Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train , Y_train , X_test)
45 m_X_test , dummy = gen_data_Y(X_test , sigma)
46
47
48 plt.plot(X_test , Y_pred_nearest_neighbor , '-r' , label='nearest_neighbor')
49 plt.plot(X_test , Y_pred_fc_nn_l , '-g' , label='fc_nn_l')
50 #colors = (0,0,0)
51 area = 4
52 plt.scatter(X_test , Y_pred_new_nn , s=area , color = 'red' , alpha=0.5)
53 plt.title('Scatter plot pythonspot.com')
54 plt.xlabel('x')
55 plt.ylabel('y')
56 plt.show()
57 #plt.plot(X_test , Y_pred_new_nn , 'ro-' , label='new_nn')
58 #plt.plot(X_test , m_X_test , '-b' , label='m_d')
59 #plt.legend(loc='upper left')
60 #plt.xlim(-2, 2)
61 #plt.xlim(-2,2)
62 #plt.show()
63 #plt.savefig('foo.png')
64
65 '''
66 ZEIDIMENSIONALER FALL (d = 2) wird geplottet
67 '''
68 N = 2
69 q = 2
70 R = 10 ** 6
71 a = 2
72 M = 2
73 d = 2
74
75 sigma = 0.05
76
77 n_train = 100
78 n_test = 2000
79 # Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
80 X_train = np.random.uniform(low=-2,high=2,size=(int(n_train),d))
81 m_X_train , Y_train = gen_data_Y(X_train , sigma)
82
83 X_test = np.random.uniform(low=-2,high=2,size=(int(n_test),d))
84
85 Y_pred_new_nn = new_neural_network_estimate(X_train , Y_train , X_test , N, q, R, d, M, a,)
86 #Y_pred_fc_nn_l = fc_neural_l_estimate(X_train , Y_train , X_test)
87 #Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train , Y_train , X_test)
88 m_X_test , dummy = gen_data_Y(X_test , sigma)
89
90
91
92 x = np.ravel(X_test[:,0])
93 y = np.ravel(X_test[:,1])
94
95 # so wie es sein soll
96 #z = m_X_test[:,0]
97 # was der Schätzer auswirft
98 z = Y_pred_new_nn[:,0]
99
100 ax = plt.axes(projection='3d')
101 ax.scatter(x, y, z, c=z, cmap='viridis' , linewidth=0.5);
102 ax.view_init(40, 20)
103 fig
104
105 '''

```

```

106 ein Vergleich des emp. L2 Fehler gemacht für d = 1
107 '''
108 n_train = 100
109 n_test = 2000
110 # Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
111
112 N = 3
113 q = 2
114 R = 10 ** 6
115 a = 2
116 M = 2
117 d = 1
118
119 sigma = 0.05
120
121 scaled_error = np.empty((5, 3,))
122 scaled_error[:] = np.nan
123
124 e_L2_avg = np.zeros(5)
125 e_L2_avg[:] = np.nan
126
127 for i in range(0,5,1):
128
129     X_train = np.sort(np.random.uniform(low=-2,high=2,size=(int(n_train),d)), axis = 0)
130     m_X_train, Y_train = gen_data_Y(X_train, sigma)
131
132     X_test = np.sort(np.random.uniform(low=-2,high=2,size=(int(n_test),d)), axis = 0)
133
134     #Y_pred_constant = constant_estimate(Y_train)
135     Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a,)
136     Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train, Y_train, X_test)
137     Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train, Y_train, X_test)
138
139     m_X_test, not_needed = gen_data_Y(X_test, sigma)
140
141     e_L2_new_nn = np.mean(sum(abs(Y_pred_new_nn - m_X_test) ** 2))
142     e_L2_fc_nn_1 = np.mean(sum(abs(Y_pred_fc_nn_1 - m_X_test) ** 2))
143     e_L2_nearest_neighbor = np.mean(sum(abs(Y_pred_nearest_neighbor - m_X_test) ** 2))
144
145     for j in range(0,5,1):
146
147         X = np.sort(np.random.uniform(low=-2,high=2,size=(n_test,d)), axis = 0)
148         m_X, Y = gen_data_Y(X, sigma)
149         Y_pred_constant = constant_estimate(Y)
150
151         e_L2_avg[j] = np.mean(sum(abs(Y_pred_constant - m_X) ** 2))
152
153     scaled_error[i,0] = e_L2_new_nn / np.median(e_L2_avg)
154     scaled_error[i,1] = e_L2_fc_nn_1 / np.median(e_L2_avg)
155     scaled_error[i,2] = e_L2_nearest_neighbor / np.median(e_L2_avg)
156
157     iqr_new_nn = iqr(scaled_error[:,0])
158     iqr_fc_nn_1 = iqr(scaled_error[:,1])
159     iqr_nearest_neighbor = iqr(scaled_error[:,2])
160
161     median_new_nn = np.median(scaled_error[:,0])
162     median_fc_nn_1 = np.median(scaled_error[:,1])
163     median_nearest_neighbor = np.median(scaled_error[:,2])
164
165     rows = ["noise", "e_L2_avg", "approach", "new_nn", "fc_nn_1", "nearest_neighbor"]
166
167     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)
168     series_noise_1.name = ""
169     #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)
170     #series_noise_2.name = ""
171
172     error_df = pd.concat([series_noise_1], axis=1)
173     print(error_df)
174     #error_df.to_csv('out.csv', index = True)
175
176 '''
177 ein Vergleich des emp. L2 Fehler gemacht für d = 2
178 '''

```

```

179 n_train = 100
180 n_test = 2000
181 # Parameter für unseren neuen Neuronale-Netze-Regressionschätzer
182
183 N = 2
184 q = 2
185 R = 10 ** 6
186 a = 2
187 M = 2
188 d = 2
189
190 sigma = 0.05
191
192 scaled_error = np.empty((5, 3,))
193 scaled_error[:] = np.nan
194
195 e_L2_avg = np.zeros(5)
196 e_L2_avg[:] = np.nan
197
198 for i in range(0,5,1):
199
200     X_train = np.sort(np.random.uniform(low=-2,high=2,size=(int(n_train),d)), axis = 0)
201     m_X_train, Y_train = gen_data_Y(X_train, sigma)
202
203     X_test = np.sort(np.random.uniform(low=-2,high=2,size=(int(n_test),d)), axis = 0)
204
205     #Y_pred_constant = constant_estimate(Y_train)
206     Y_pred_new_nn = new_neural_network_estimate(X_train, Y_train, X_test, N, q, R, d, M, a,)
207     Y_pred_fc_nn_1 = fc_neural_1_estimate(X_train, Y_train, X_test)
208     Y_pred_nearest_neighbor = nearest_neighbor_estimate(X_train, Y_train, X_test)
209
210     m_X_test, not_needed = gen_data_Y(X_test, sigma)
211
212     e_L2_new_nn = np.mean(sum(abs(Y_pred_new_nn - m_X_test) ** 2))
213     e_L2_fc_nn_1 = np.mean(sum(abs(Y_pred_fc_nn_1 - m_X_test) ** 2))
214     e_L2_nearest_neighbor = np.mean(sum(abs(Y_pred_nearest_neighbor - m_X_test) ** 2))
215
216     for j in range(0,5,1):
217
218         X = np.sort(np.random.uniform(low=-2,high=2,size=(n_test,d)), axis = 0)
219         m_X, Y = gen_data_Y(X, sigma)
220         Y_pred_constant = constant_estimate(Y)
221
222         e_L2_avg[j] = np.mean(sum(abs(Y_pred_constant - m_X) ** 2))
223
224         scaled_error[i,0] = e_L2_new_nn / np.median(e_L2_avg)
225         scaled_error[i,1] = e_L2_fc_nn_1 / np.median(e_L2_avg)
226         scaled_error[i,2] = e_L2_nearest_neighbor / np.median(e_L2_avg)
227
228     iqr_new_nn = iqr(scaled_error[:,0])
229     iqr_fc_nn_1 = iqr(scaled_error[:,1])
230     iqr_nearest_neighbor = iqr(scaled_error[:,2])
231
232     median_new_nn = np.median(scaled_error[:,0])
233     median_fc_nn_1 = np.median(scaled_error[:,1])
234     median_nearest_neighbor = np.median(scaled_error[:,2])
235
236     rows = ["noise", "e_L2_avg", "approach", "new_nn", "fc_nn_1", "nearest_neighbor"]
237
238     series_noise_1 = pd.Series([repr(sigma)+'%', np.median(e_L2_avg), "(Median, IQR)", (median_new_nn, iqr_new_nn), (median_
239         fc_nn_1, iqr_fc_nn_1), (median_nearest_neighbor, iqr_nearest_neighbor)], index=rows)
240     series_noise_1.name = ""
241     #series_noise_2 = pd.Series([repr(sigma)+'%', np.median(e_L2_avg), "(Median, IQR)", (median_new_nn, iqr_new_nn), (median
242         _fc_nn_1, iqr_fc_nn_1), (median_nearest_neighbor, iqr_nearest_neighbor)], index=rows)
243     #series_noise_2.name = ""
244
245     error_df = pd.concat([series_noise_1], axis=1)
246     print(error_df)
247     #error_df.to_csv('out.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 Regressionschätzern benötigen
9 """
10 # Wir wählen x gleichverteilt auf  $[-1,1]^d$ , 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 \in  $[-1,1]^d$ 
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  $[-1,1]^d$  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
48     n = np.size(X, 0)
49     d = np.size(X, 1)
50
51     m_X = np.zeros((n,1,))
52     m_X[:] = np.nan
53
54     S = np.random.standard_normal(size=(n,1))
55     for t in range(0,n):
56         m_X[t] = m_d(X[t], d)
57
58     Y = m_X + sigma * iqr(m_X) * S
59     return (m_X, Y)

```

Listing 4.3: help_neural_networks.py

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Tue Oct 15 11:22:02 2019
5
6 @author: adrian
7
8 Implementation von Neuronale-Netzen welche wir für die Konstruktion unseres
9 Neuronale-Netze-Regressionschätzers benötigen
10 """
11 import numpy as np
12
13 # Sigmoidfunktion
14 #

```

```

15 # x: x \in \mathbb{R}
16
17 def sigmoid (x):
18
19     return 1 / (1 + np.exp(-x))
20
21 # Neuronales Netz welches die Funktion f(x) = x approximiert
22 #
23 # x: reelle Zahl
24 # R: reelle Zahl >= 1
25
26 def f_id (x, R):
27
28     return 4 * R * sigmoid(x / R) - 2 * R
29
30 # Neuronales Netz welches die Funktion f(x, y) = x * y approximiert
31 #
32 # x: reelle Zahl
33 # y: reelle Zahl
34 # R: reelle Zahl >= 1
35
36 def f_mult (x, y, R):
37
38     return (((R ** 2) / 4) * (((1 + np.exp(-1)) ** 3) / (np.exp(-2) - np.exp(-1)))) \
39         * (sigmoid(((2 * (x + y)) / R) + 1) - 2 * sigmoid(((x + y) / R) + 1) \
40         - sigmoid(((2 * (x - y)) / R) + 1) + 2 * sigmoid(((x - y) / R) + 1))
41
42 # Neuronales Netz welches die Funktion f(x) = max(x,0) approximiert
43 #
44 # x: reelle Zahl
45 # R: reelle Zahl >= 1
46
47 def f_relu (x, R):
48
49     return f_mult(f_id(x, R), sigmoid(R * x), R)
50
51 # Neuronales Netz welches die Funktion f(x) = max(1 - (M/(2 * a)) * abs(x - y), 0) approximiert
52 #
53 # x: reelle Zahl
54 # y: fixe reelle Zahl
55 # R: reelle Zahl >= 1
56 # M: fixe natürliche Zahl
57 # a: fixe Zahl > 0
58
59 def f_hat (x, y, R, M, a):
60
61     return f_relu((M / (2 * a)) * (x - y) + 1, R) - 2 * f_relu((M / (2 * a)) * (x - y), R) + 2 * f_relu((M / (2 * a))
        * (x - y) - 1, R)

```

Listing 4.4: new_neural_network.py

```

1 #!/usr/bin/env python3
2 #-*- coding: utf-8 -*-
3 """
4 Created on Wed Oct 16 15:40:14 2019
5
6 @author: adrian
7
8 Um die Gewichte der Ausgabeschicht zu bestimmen lösen wir ein regularisiertes
9 Kleinste-Quadrate Problem.
10
11 """
12 import scipy.special
13 import numpy as np
14 import itertools
15 from help_neural_networks import f_id, f_mult, f_hat
16 import math
17
18 # Neuronales Netz welches die Funktion f(x) = (x^(1)- x_ik^(1))^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 # Kleinste-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
91     # erfüllen
92     # X_ik hat in den Zeilen die Vektoren X_i aus dem Paper
93
94     X_ik = np.transpose(np.empty((d, (1 + M) ** d,)))
95     X_ik[:] = np.nan
96
97     I_k = np.array(list(itertools.product(range(0, M + 1), repeat = d)))
98     X_ik[:] = (I_k[:] * ((2 * a) / M)) - a

```

```

98
99     all_jl_jd = np.array(list(itertools.product(range(0, N + 1), repeat = d)))
100     all_jl_jd_by_cond = all_jl_jd[all_jl_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_jl_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.
113         transpose(B), Y))
114     weights = np.linalg.solve(np.dot(np.transpose(B), B) + (c_3) * np.identity(J), np.dot(np.transpose(B), Y))
115     return (weights, J, all_jl_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_jl_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
142     for u in range(0, len(X_test), 1):
143         j = 0
144         while j < J:
145             for k in range(0, ((M + 1) ** d)):
146                 for z in range(0, int(scipy.special.binom(N + d, d))):
147                     F_net[0, j] = f_net(X_test[u], d, all_jl_jd_by_cond[z], X_ik[k], N, q, s, R, M, a)
148                     j += 1
149
150             Y_pred[u] = np.sum(np.transpose(weights) * F_net)
151
152     return Y_pred

```

Listing 4.5: fc_neural_network.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Oct 11 14:23:15 2019
5
6  @author: adrian
7  """
8  import numpy as np
9  from keras.models import Sequential
10 from keras.layers import Dense
11
12 # Fully-Connected Neuronales Netzt mit einer Verborgenen schicht welches die
13 # Anzahl der Neuronen adaptiv, durch minimierung des L2 fehlers, aus der Menge \{5, 10, 25, 50, 75\} auswählt.
14 #
15 # X: Eingabevektoren der Form (X_1, ..., X_n) für das Neuronale Netz aus dem Datensatz (X_1, Y_1), ..., (X_n, Y_n)

```



```

16 # Y: Eingabevektoren der Form (Y_1,...,Y_n) für das Neuronale Netz aus dem Datensatz (X_1,Y_1) ,..., (X_n,Y_n)
17
18 def fc_neural_l_estimate (X_train,Y_train,X_test):
19
20     Ynew = np.empty((len(X_train), len([5,10,25,50,75]),))
21     Ynew[:] = np.nan
22
23     count = 0
24     n_neurons = [5,10,25,50,75]
25
26     d = np.size(X_train , 1)
27
28     for j in n_neurons:
29         model = Sequential()
30         model.add(Dense(j, input_dim=d, activation='relu'))
31         model.add(Dense(1, activation='linear'))
32         model.compile(loss='mse', optimizer='adam')
33         model.fit(X_train , Y_train , epochs=1000, verbose=0)
34
35         Ynew[:,count] = model.predict(X_train)[: ,0]
36         count += 1
37
38     Diff = Ynew[:] - Y_train[:]
39     best_n_neurons = n_neurons [(1/len(X_train) *(Diff.sum(axis=0) ** 2)).argmax()]
40
41     model = Sequential()
42     model.add(Dense(best_n_neurons , input_dim=d, activation='relu'))
43     model.add(Dense(1, activation='linear'))
44     model.compile(loss='mse', optimizer='adam')
45     model.fit(X_train , Y_train , epochs=1000, verbose=1)
46
47     return model.predict(X_test)

```

Listing 4.6: nearest_neighbor.py

```

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

```

Listing 4.7: constant.py

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Wed Oct 23 15:26:19 2019
5
6 @author: adrian
7 Constant Estimator
8 """
9 import numpy as np
10 from scipy import mean
11
12 # Gibt den Mittelwert der Funktionswerte einer Funktion als Schätzer zurück
13 #
14 # Y: Datensatz der Form (Y_1,...) wobei  $Y_i \in \mathbb{R}$  für  $i = 1, \dots$ 
15
16 def constant_estimate(Y):
17     m = np.zeros((len(Y),1,))
18     m[:] = mean(Y)
19     return m
```