Universität Bielefeld
Fakultät Wirtschaftswissenschaften
Lehrstuhl für Decision and Operation Technologies

Bachelorarbeit
im Studiengang Wirtschaftswissenschaften

Neuronale Netze zur Evaluation von Algorithmen Konfiguratoren

von
Gregor Felix Foitzik
3982803
Westring 8, 33378 Rheda-Wiedenbrueck
gregor.foitzik@uni-bielefeld.de

vorgelegt bei
Kevin Tierney
Elias Arnold Schede

10. Februar 2023

Abstract - englisch

Empirical performance models are concerned with predicting the goodness of a previously unseen algorithm configuration on various instances, some of which are unknown. In this paper, artificial neural networks are presented that predict the running time of different configurations of algorithms on a wide number of instances.

Performance data from K. Eggesperger was used to train and validate the neural network. This allows a direct comparison to the Quantile Regression Forest (QRF) provided by her.

Findings from the work include evaluation of different network architectures and impact of various regularization methods. Using the presented network, a variety of algorithm configurations can be tested in a few seconds.

Keywords: Empirical performance model, Algorithm configuration, Neural networks, PyTorch, Dropout-Layer, Lasso-Regulization, Ridge-Regulization, Early Stopping, AClib

Abstract - deutsch

Die empirischen Performance-Modelle (EPM) beschäftigen sich damit die Laufzeit einer zuvor ungesehenen Algorithmuskonfiguration auf verschiedene, zum Teil unbekannte, Instanzen vorherzusagen. In dieser Arbeit werden neuronale Netzwerke als EPM vorgestellt, die die Laufzeit verschiedener Konfigurationen von Algorithmen auf eine breite Anzahl an Instanzen vorhersagt.

Dabei wurden Performance-Daten von K. Eggensperger verwendet, um das neuronale Netzwerk zu trainieren und validieren. Dadurch kann ein direkter Vergleich zu dem von K. Eggensperger bereitgestellten *Quantile Regression Forest* (QRF) hergestellt werden.

Erkenntnisse aus der Arbeit umfassen dabei Evaluation verschiedener Netzwerkarchitekturen und Auswirkung diverser Regularisierungsmethoden. Mit Hilfe des vorgelegten Netzwerks können in wenigen Sekunden eine Vielzahl von Algorithmenkonfigurationen getestet werden.

Stichworte: Empirische Performance-Modelle, Algorithmuskonfiguration, Neuronale Netzwerke, PyTorch, Dropout-Schichten, Lasso-Regularisierung, Ridge-Regularisierung, Early Stopping, AClib

Inhaltsverzeichnis

1. Einleitung **1**

2. Literaturüberblick **2**
2.1. Generierung der Instanzen 2
2.1.1. ParamILS 3
2.1.2. ROAR 3
2.1.3. SMAC 4
2.2. Aufbau der Instanzen 4

3. Neuronale Netzwerke **7**
3.1. Aufbau der Netzwerke 7
3.1.1. Aktivierungsfunktionen 7
3.1.2. Verlustfunktionen 9
3.2. ADAM 10
3.3. Backpropagation 11
3.4. Initialisierung der Gewichte 11
3.5. Regularisierung 12
3.5.1. Dropout 12
3.5.2. Lasso-Regularisierung 12
3.5.3. Ridge-Regularisierung 12
3.5.4. Early Stopping 13

4. Implementierung und Auswertung **14**
4.1. Implementierung und Hypertuning Prozess 14
4.1.1. Anwendungen 14
4.1.2. Modellauswahl 14
4.1.3. Dropout 15
4.1.4. Lasso- und Ridge-Regularisierung 15
4.1.5. Early Stopping 15
4.2. Auswertung Hypertuning 16
4.3. Auswertung Qualität 17

5. Fazit & Ausblick **19**
5.1. Fazit 19
5.2. Ausblick 19

Literaturverzeichnis **20**

A. Verteilung der Laufzeiten **23**

B. Auswertung Dropout-Schichten **26**

C. Auswertung der Güte **28**

D. Verlustfunktionen der ausgewählten NN **35**

Abkürzungsverzeichnis

EPM Empirische Performace-Modelle

QRF Quantile Regression Forest

ML Maschineller Lernen

HPO Hyperparameter Optimierung

AC Algorithmuskonfigurationsproblem

NN Neurolale Netzwerke

GS GridSearch

RS RandomSearch

BOA Bayesian Optimization Algorithm

ParamILS Iterated Local Search in Parameter Configuration Space

ROAR Random Online Aggressive Racing

SMAC Sequential Model-Based Algorithm Configuration

IL Input Layer (Eingabeschicht)

HL Hidden Layer (versteckte Schicht)

OL Output Layer (Ausgabeschicht)

DO Dropout Layer

logPAR10 Logarithmierte bestrafte Laufzeiten gemittelt über 10 Durchläufe

CC Spearman's Rank Correlation

MSE Mean Squard Error

RMSE Root Mean Squard Error

MAE Mean Absolute Error

LOCO Leave One Configuration Out

ECDF Exponential Cumulatice Distribution Function

CPU Central Processing Unit

GPU Graphics Processor Unit

CUDA Compute Unified Device Architecture

Nomenklatur

Die folgende Liste beschreibt Symbole, die im späteren genutzt werden.

Algorithmuskonfiguration

θ	Parameterkonfiguration
Θ	Menge aller Parameterkonfigurationen
θ'	Nachbarkonfiguration von θ
Θ'	Menge aller Nachbarkonfigurationen von θ'
θ_{inc}	Parameterkonfiguration mit bester Güte
r	Anzahl der Parameter eines Algorithmus
s	Vordefinierte Anzahl zufälliger Parameterkonfigurationen

Neuronale Netzwerke

L	Anzahl der HL in einem NN
n_0^{IL}	Anzahl der Neuronen im IL
n_l^{HL}	Anzahl der Neuronen im l-ten HL
N^{HL}	Menge aller HL in einem NN
n_{L+1}^{OL}	Anzahl der Neuronen im OL
$\mathbf{w}, \mathbf{b}$	Gewichte und Bias
$w^{(l)}$	Gewichte der Neuronen für die l-te Schicht
$b^{(l)}$	Bias der Neuronen für die l-te Schicht
y^l	Ausgabe der l-ten Schicht
$\sigma(x)$	Logistische Sigmoid-Funktion
$\mathbf{t}(x)$	Tangens hyperperbolicus
$ReLU(x)$	Rectified Linear Unit

p	Dropout-Rate
λ_{L1}	Lagrange-Multiplikator für Lasso-Regularisierung
λ_{L2}	Lagrange-Multiplikator für Ridge-Regularisierung
A_E	stetige Architektur
A_D	rückläufige Architektur

1. Einleitung

Maschinelles Lernen (ML) ist in der Forschung und Wirtschaft ein essenzieller Bestandteil geworden. In diversen Bereichen kann ML angewendet werden.

Die Prognose von Absatzentwicklungen ist eine Anwendung in der Wirtschaft. Durch die Analyse von Verkaufsdaten können Unternehmen ihre Absatzprognosen verbessern und ihre Geschäftstätigkeiten entsprechend planen (Tsoumakas, 2019). Ebenso kann das Marketing durch Empfehlungssysteme effektiver gestaltet werden. Kunden bekommen personalisierte Vorschläge von Produkten oder Dienstleistungen, die ihren Wünschen entsprechen (Resnick and Varian, 1997). Für Banken und Versicherungsgesellschaften ist Machine Learning bei der Risikomodellierung mittlerweile ein essentieller Bestandteil (Leo et al., 2019). Personalisierten Bildung zur Unterstützung von Studierenden ist ebenfalls ein Einsatzgebiet von Machine Learning. (Ciolacu et al., 2018) untersuchte den Effekt von Systemen, die den Erfolg von Studierenden steigern sollten. Neuronale Netzwerke werden unter anderem in der medizinischen Diagnostik eingesetzt. Erkrankungen sollen schneller und genauer diagnostiziert werden, um Behandlungen früher und zielgerichteter anzuwenden (Kononenko, 2001).

Die Qualität von künstlichen Intelligenzen ist stark abhängig von den Daten und gewählten Hyperparametern. Um Bestwerte hinsichtlich der Qualität und Laufzeit zu erzielen, ist die Wahl der Konfiguration eines Modells ausschlaggebend.

Algorithmuskonfiguratoren beschäftigen sich damit optimale Hyperparameter für einen Algorithmus zu finden. Dieses Problem wird als *Hyperparameter Optimierung* (HPO) bezeichnet. Das manuelle Justieren von Parametern ist besonders zeitaufwendig und mühsam. Um dem traditionellen Weg des manuellen Justierens zu umgehen, werden Systeme entwickelt die Konfigurationen automatisch ermitteln (Olof, 2018). Automatisierte Systeme verfolgen den Ansatz mit Hilfe von Ersatzszenarien optimale Konfigurationen für Algorithmen (AC) zu bestimmen. Ersatzszenarien haben den Vorteil, dass sie nah an den Ausgangsproblemen sind, wobei sie leichter berechnet werden können (Eggensperger et al., 2018). Daher werden sie häufig für Approximationen der Leistung von Zielalgorithmen verwendet.

EPM sind ein Teilgebiet der HPO und konzentrieren sich darauf eine Prognose der Laufzeit für unbekannte Algorithmenkonfigurationen und Instanzen zu generieren. Das Ziel dieser Arbeit ist das Trainieren von neuronalen Netzwerken (NN) und das Untersuchen ihrer Anwendbarkeit als empirische Performance-Modelle.

2. Literaturüberblick

Ein häufig angewandter Konfigurator ist *GridSearch* (GS). Dabei werden alle Konfigurationen eines vorgegebenen Gitters ausgewertet (Injadat et al., 2020). Das Anwenden von GS erfordert kein Vorwissen und ist einfach. Jedoch ist es ineffizient, weil die Anzahl der Berechnungen exponentiell ansteigt je mehr Hyperparameter verwendet werden (Claesen et al., 2014).

RandomSearch (RS) wählt Parameterwerte aus einem vordefinierten Bereich zufällig aus. Wenn der Konfigurationsbereich ausreichend groß gewählt wird, können Approximationen der globalen Optima gefunden werden (Bergstra and Bengio, 2012).

Im Gegensatz zum GS und RS ist der *Bayesian Optimization Algorithm* (BOA) ein iterativer Algorithmus. Der BOA berechnet die Qualität zukünftiger Konfigurationen auf Basis der zuvor erzielten Ergebnisse und schätzt eine gemeinsame Verteilungsfunktion vielversprechender Konfigurationen (Pelikan et al., 1999).

Das *Empirical Benchmarking* beschäftigt sich damit die Laufzeit von unbekannten Algorithmenkonfigurationen und Instanzen zu prognostizieren. Die Modelle dieses Teilgebiets werden als EPM bezeichnet. Eggensperger et al. (2018) entwickelten einen *Quantile Regression Forests* der sehr gute Ergebnisse erzielt.

Um die Qualität des QRF zu bewerten, verwendeten Eggensperger et al. (2018) den *Root Mean Squared Error* (RMSE) der gemittelten logarithmierten bestraften Laufzeiten über 10 Durchläufe (logPAR10) und die *Spearman's rank correlation* (CC) der wahren und prognostizierten logPAR10. Es wurden zehn Modelle mit einem *random state* trainiert und anschließend über die prognostizierten Laufzeiten gemittelt. Mit Hilfe des QRF konnte die Berechnungsdauer einer Konfiguration um mehr als den Faktor 100[1] verringert werden. Des Weiteren beträgt der durchschnittliche RMSE 0.49 ± 0.06 und die durchschnittliche CC 0.77 ± 0.05, der auch in dieser Arbeit verwendeten Daten.

2.1. Generierung der Instanzen

Die Algorithmuskonfiguratoren *ParamILS*, *ROAR* und *SMAC* generierten die Daten. Es wurden zehn Durchläufe mit jedem Konfigurator durchgeführt. Die Informationen stammen von Hutter et al. (2009) und Hutter et al. (2011)

[1] 27.29 ± 100.26 zu $0.23 \pm 0.13s$ ($\mu \pm \sigma$)

2.1.1. ParamILS

Das Vorgehen, an dem sich der ParamILS bedient ist unkompliziert. Der Algorithmus stoppt die Evaluation, sobald ein lokales Optimum gefunden wird. Ein lokales Optimum definiert sich dadurch, dass keine Nachbarkonfiguration[2] $\theta' \in \Theta'$ eine Verbesserung darstellt.

Der ParamILS startet mit einer initialen Konfiguration $\theta_{inc} \in \Theta$, die entweder den Voreinstellungen oder falls keine vorhanden sind, einer zufälligen Konfiguration entspricht. Zuerst werden r[3] zufällige Konfigurationen $\theta_{new} \in \Theta$ mit θ_{inc} verglichen. Wenn θ_{new} bessere Werte als θ_{inc} hinsichtlich der gewählten Metrik erzielt, wird θ_{inc} aktualisiert. Danach wird θ_{inc} mit allen Nachbarkonfigurationen verglichen. Sobald eine Nachbarkonfiguration gefunden wird, die besser ist als die aktuell beste Konfiguration, wird θ_{inc} ebenfalls aktualisiert. Sonst folgt $\theta_{inc} \leftarrow \theta_{inc}$. Dieser Schritt sucht nach einem lokalen Optimum und wird von Hutter et al. (2009) *ItereativeFirstImprovement* (θ) genannt. Im Folgeschritt wird s-Mal[4] ein zufälliges $\theta' \in \Theta$ generiert. Vom letzten zufällig generierten θ' wird ein neues lokales Optimum nach dem *ItereativeFirstImprovement* (θ') Prinzip gesucht. Die resultierende Konfiguration θ_{inc} ist die beste gefundene Konfiguration.

2.1.2. ROAR

Beim ROAR Ansatz werden verschiedene Parameterkonfigurationen uniform und zufällig ausgewählt. Hutter et al. (2011) unterteilten die Vorgehensweise in die vier Komponenten *Initialize*, *FitModel*, *SelectConfigurations* und *Intensify*.

Das Ausgangsmodell besteht aus den voreingestellten Parametern, ansonsten wird es ebenfalls uniform und zufällig ausgewählt. ROAR ist modellfrei und gibt daher im *FitModel* ein konstantes Modell zurück, das jedoch nie verwendet wird. Im dritten Schritt werden Konfigurationen Θ_{new} generiert, die aus der Menge aller Konfigurationen Θ stammen. Im letzten Schritt werden die Parametereinstellungen θ_{new} mit den bisher besten θ_{inc} verglichen. Wenn ein θ_{new} empirisch besser ist als θ_{inc}, wird es als beste Parametereinstellung gespeichert. Dieser Prozess wird erneut für die Konfigurationen θ_{new} ausgeführt. Sobald ein Durchlauf eine festgelegte maximale Laufzeit überschreitet stoppt der Algorithmus und gibt die bisher beste Parametereinstellung θ_{inc} zurück. Trotz des einfachen Prinzips erzielt ROAR gute Ergebnisse.

[2] Eine Nachbarkonfiguration beschreibt eine Konfiguration bei der nur ein Parameter verändert wurde.

[3] r beschreibt die Anzahl an Parametern eines Algorithmus

[4] s ist eine vordefinierte ganzzahlige Zahl

2.1.3. SMAC

Aufgrund der gleichen Funktionsweise von *Initialize* und *Intesify* kann SMAC als eine Erweiterung von ROAR verstanden werden. Anstatt due Konfiguration nach einem uniformen Zufallsprinzip auszuwählen zeiht SMAC ein Modell heran. Die von SMAC verwendeten Modelle basieren auf *Random Forests*, die bereits von Bartz-Beielstein and Markon (2004) auf AC untersucht wurden. Aufgrunddessen können von SMAC ebenfalls kategorische Parameter berücksichtigt werden. Hutter et al. (2011) konnten zeigen, dass SMAC in mehreren Szenarien und mit indiskreten Konfigurationsräumen bessere Ergebnisse erzielt als ROAR.

2.2. Aufbau der Instanzen

Um die neuronalen Netzwerke zu trainieren und ihre Leistungen evaluieren zu können, werden die von Eggensperger et al. (2018) erstellten und bereits für den QRF verwendeten Daten, genutzt. Bei der Erstellung wurde eine Reihe von verschiedenen Instanzen angewandt. Die Daten beinhalten sowohl die Konfigurationen der Algorithmen als auch Informationen über die Beschaffenheit der Instanzen an denen sich Eggensperger et al. (2018) bedienten.

Datensatz	#Parameter cat./num./co. (cond.)	Dimensionen	
		Train	Val
CPLEX on Regions200	63/7/4 (4)	645.984 x 373	338.000 x 373
CPLEX on RCW2	63/7/4 (4)	157.181 x 373	50.985 x 373
Probsat on 7SAT90	5/1/3 (5)	189.048 x 148	45.500 x 148
Minisat on randomK3	10/0/0 (0)	166.716 x 152	41.750 x 152
Clasp on Rooks	38/30/7 (55)	234.964 x 229	64.935 x 229
Lingeling on Circuitfuzz	137/185/0 (0)	138.735 x 521	27.482 x 521

Tabelle 2.1.: Aufbau der Datensätze

In der Tabelle 2.1 sind Informationen zu den verwendeten Datensätzen. In der zweiten Spalte sind Datentypen und die Anzahl von Parameter des Algorithmus vermerkt[5]. Die Anzahl der Ersatzszenarien und die Variablen sind der Spalte *Dimensionen*, aufgeteilt in Trainings- und Validierungsdatensatz, zu entnehmen.

[5]cat: Kategorische Parameter | int: Ganzzahliger Parameter | co: Kontinuierlicher Parameter | cond: Bedingung für Parameter

Eggensperger et al. (2018) entschlüsselten für den QRF die kategorischen Parameter der Algorithmen nicht. Daher kann es zu Unterschieden bei der Anzahl von Spalten kommen. Für das NN wurden die kategorischen Parameter mit dem Verfahren One-Hot-Encoding entschlüsselt. One-Hot-Encoding ist ein Verfahren, um kategoriale Variablen in numerische Variablen umzuwandeln. Dabei wird für jeden Wert eine neue binäre Spalte erzeugt. Zur Reduzierung der Dimensionen wurde die erste binäre Spalte einer Kategorie entfernt. Dadurch gehen keine Informationen verloren.

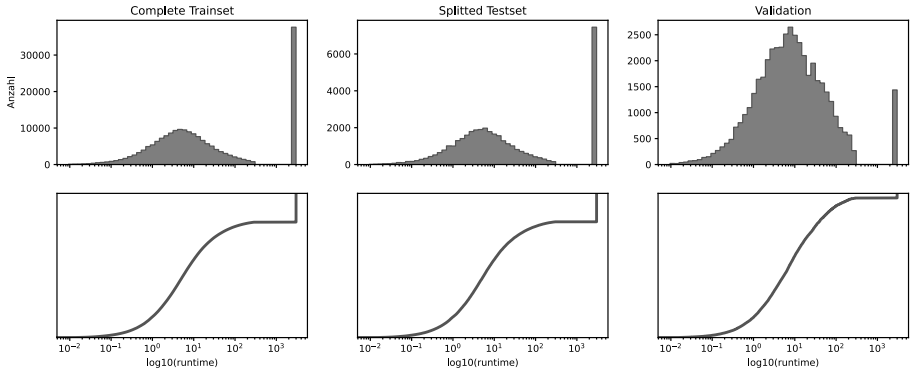

Abbildung 2.1.: Histogramme und ECDF der Laufzeit (*Probsat 7SAT90*)

In der Abbildung 2.1 zeigt, dass die Laufzeiten stark unausgeglichen verteilt[6] sind. Durch das Bestrafen der abgebrochenen Durchläufe entsteht dieses Ungleichgewicht.

Die Histogramme dienen zur Darstellung der logarithmierten Laufzeiten. Die *Exponential Cumulative Distribution Function* (ECDF) veranschaulicht das Ungleichgewicht, indem hervorgehoben wird, dass zwischen den nicht bestraften und bestraften Laufzeiten keine Daten vorhanden sind. Im Anhang (A) befinden sich Grafiken zu den Verteilungen weiterer Datensätze.

Um mit den Daten von Hutter et al. (2014) arbeiten zu können, müssen die Datensätze so angepasst werden, dass die Instanz des Durchlaufs in der ersten Spalte ist. Die Laufzeit der Konfiguration muss sich in der zweiten Spalte befinden. Die Parameter der jeweiligen Konfiguration liegen in den weiteren Spalten. Nachdem die Informationen zu den Konfigurationen eingelesen wurden, werden die Eigenschaften der Instanzen hinzugefügt.

Falls kategorische Werte nicht in einem Datensatz vorlagen, fehlten die binären Spalten dieser Werte nach dem One-Hot-Encoding. Die Matrizen konnten nicht miteinander verrechnet werden. Die fehlenden binären Spalten wurden ergänzt. Für das Einlesen der Daten wurden in Microsoft Excel manuell Dateien erstellt, die Parametereigenschaften der Algorithmen enthalten.

[6]Mit einer unausgeglichenen Verteilung ist gemeint, dass keine statistische Verteilung die Verteilung der Daten erklären kann.

Im Folgenden wird unter den Parametern die bereits entschlüsselte Form der kategorischen Hyperparameter zuzüglich der Eigenschaften der Instanzen verstanden.

3. Neuronale Netzwerke

In den vergangenen Jahren haben NN stark an Beliebtheit gewonnen und finden in vielen Bereichen Anwendung. Künstliche neuronale Netzwerke adaptieren die Funktionsweise von biologischen neuronalen Netzwerken, die aus einer Vielzahl an Neuronen und deren Vernetzung bestehen. Die Neuronen erhalten, verarbeiten und geben, unter bestimmten Voraussetzungen, Informationen an folgende Neuronen weiter.

3.1. Aufbau der Netzwerke

Ein NN besteht aus drei Arten von Schichten. Dabei kann jede Schicht aus mehreren Neuronen bestehen. In die erste Schicht werden alle Daten eingegeben. Die Anzahl der Neuronen n_0^{IL} entspricht der Anzahl der Merkmale des Datensatzes. Im Falle des Empirical Benchmarking Problems ist dies ein Vektor $\vec{e} \in R^{n_0^{IL}}$. Die Neuronenanzahl der versteckten Schichten (HL) n_l^{HL} mit $l \in \{1, ..., L\}$ können von dieser Zahl abweichen. In der letzten Schicht werden zuvor generierten Erkenntnisse aus den versteckten Schichten aggregiert. Bei den angewandten NN besteht die letzte Schicht aus einem Neuron n_{L+1}^{OL}, das die logarithmierte Laufzeit der Konfiguration als Gleitkommazahl ausgibt.

Srivastava et al. (2014) beschrieben einen Vorwärtsschritt wie folgt: Sei y^l die Ausgabe der Schicht l und $w_i^{(l+1)} \in W^{(l+1)}$ das Gewicht des Neuron i und $b_i^{(l+1)} \in B^{(l+1)}$ der Bias in der darauf folgenden Schicht. Die Aktivierungsfunktion ist als $f(x)$ gegeben.

$$\begin{aligned} z_i^{(l+1)} &= w_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}) \end{aligned} \tag{3.1}$$

3.1.1. Aktivierungsfunktionen

Aktivierungsfunktionen simulieren das „Feuern“ eines Neurons. Unter diesem Begriff wird das Weiterleiten von Informationen verstanden. Die Aktivierungsfunktion bestimmt darüber, ob Informationen weitergegeben werden oder zurückgehalten werden. Im Folgenden werden drei bekannte Aktivierungsfunktion vorgestellt. Die Information wurde Calin (2020) entnommen, wenn nicht anders gekennzeichnet.

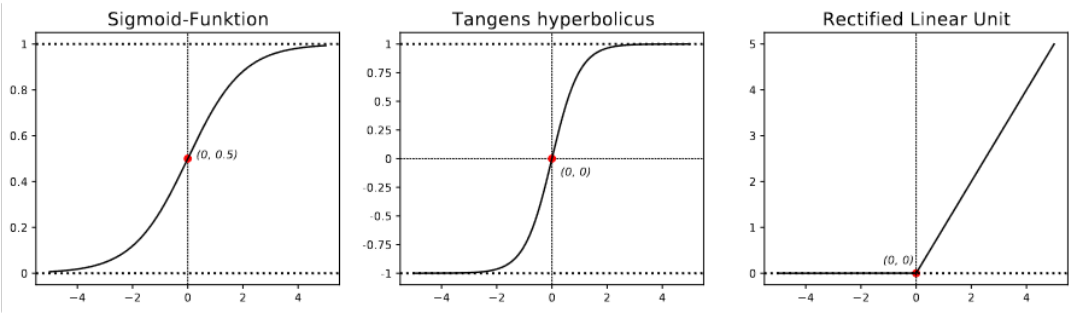

Abbildung 3.1.: Aktivierungsfunktionen

Logistische Sigmoid-Funktion

Die Sigmoid-Funktion (Abbildung 3.1, links) hat den Vorteil, dass sie glatt ist und eine Stufenfunktion mit beliebiger Genauigkeit approximieren kann. Da $\sigma(x) \in [0, 1]$ kann die Ausgabe der Funktion als Wahrscheinlichkeit interpretiert werden. Dadurch ist sie für Klassifikationen geeignet. Ein Nachteil der Sigmoid-Funktion ist, dass die Ableitung der bei $x \to -\infty$ und $x \to \infty$ gegen null konvergiert[1]. Zudem gilt $\sigma(x)' \in [0, 0.25]$, weshalb die Gewichte nur begrenzt angepasst werden können. Dadurch wird der Lernprozess verlangsamt.

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{3.2}$$

Tangens hyperbolicus

Beim Tangens hyperbolicus (Abbildung 3.1, mitte) liegen die Werte der Funktion zwischen -1 und 1. Die Ableitung $\mathbf{t}(x)' \in (0, 1]$ ermöglicht größere Gradienten. Dadurch kann das NN schneller lernen. Jedoch ist es arbeitsintensiver diese Funktion zu berechnen. Die Dauer des Trainings erhöht sich. Außerdem tritt, wie bei der logistischen Sigmoid-Funktion, das Vanishing-Problem auf.

$$\mathbf{t}(x) = \tanh \mathrm{x} = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{3.3}$$

[1]Dann ist das Netzwerk nicht mehr in der Lage zu lernen. Dieses Problem ist als „Vanishing“ (Verschwinden) bekannt.

Rectified Linear Unit *(ReLU)*

Die Vorteile der ReLU-Funktion (Abbildung 3.1, rechts) sind, dass die Funktion und ihre Ableitung einfach zu berechnen sind. Dadurch lernen neuronale Netze um ein Vielfaches schneller als identische Netzwerke mit sättigenden Aktivierungsfunktionen (Krizhevsky et al., 2017). Nwankpa et al. (2018) verglichen zahlreiche Aktivierungsfunktionen. Eine wichtige Erkenntnis aus der Arbeit ist, dass unter anderem ReLU keine verschwindenden Gradienten aufweist. Im Gegensatz zu $\sigma(x)$ und $t(x)$ wird $ReLU(x)$ nicht gesättigt (Calin, 2020).

$$ReLU(x) = xH(x) = \max\{0, x\} = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{3.4}$$

3.1.2. Verlustfunktionen

Für neuronale Netzwerke gibt es eine große Auswahl an Verlustfunktionen. Für Klassifikationsprobleme ist die Auswahl um ein Vielfaches größer. Dabei hat jede Verlustfunktion Vor- und Nachteile. Im Folgenden werden drei häufig für Regression verwendete Verlustfunktionen vorgestellt. Die Informationen stammen von Paszke et al. (2019).

Mean Squared Error *(MSE)*

Der MSE ist eine vielfach angewandte Metrik für neuronale Netzwerke, die für Regressionen verwendet werden. Durch das Quadrieren des Fehlers fallen größere Fehler überproportional ins Gewicht. Zudem ist der MSE eine konvexe Funktion mit nur einem globalen Minimum.

$$MSE(y, \hat{y}) = \sum_{i=1}^{I} (y_i - \hat{y}_i)^2 \tag{3.5}$$

Mean Absolute Error *(MAE)*

Im Gegensatz zum MSE werden beim MAE die Beträge der Fehler aufsummiert. Größere Fehler fallen daher nicht überproportional ins Gewicht. Dadurch ist der MAE resistenter gegen Ausreißer.

$$MAE(y, \hat{y}) = \sum_{i=1}^{I} |y_i - \hat{y}_i| \tag{3.6}$$

HuberLoss

Die HuberLoss-Funktion kombiniert den MSE und MAE, sodass kleinere Fehler überproportional bestraft werden als größere Fehler. Die HuberLoss kann verwendet werden, wenn wenige sehr große Ausreißer in den Daten vorhanden sind.

$$l_n = \begin{cases} 0.5(y - \hat{y})^2, & if|y - \hat{y}| < delta \\ delta * (|y - \hat{y}| - 0.5 * delta), & otherwise \end{cases} \tag{3.7}$$

3.2. ADAM

Algorithm 1 Adapive Moment Estimation

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ: Inital parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta t - 1 - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Der Adam Optimierer ist eine Kombination der Methoden AdaGrad und RMSProp. AdaGrad erzielt gute Ergebnisse bei geringen Gradienten und RMSProp bei linearen und nicht stationären Eigenschaften. Der Adam Optimierer nutzt zwei Arten von Momenten, die initial auf null gesetzt werden. Im ersten Schritt werden die Gradienten der Verlustfunktion berechnet. Danach werden die Momente aktualisiert. Im Gegensatz zum ersten Moment wird beim zweiten Moment der quadrierte Gradient verwendet. Bevor die Parameter aktualisiert, werden korrigiert Adam die Momente, sodass diese nicht verzerrt sind. (Kingma and Ba, 2014)

3.3. Backpropagation

Nachdem die grundlegenden Aspekte eines NN besprochen wurden, wird die Funktionsweise der *Backpropagation* („Fehlerrückführung“) erläutert. Mittels des iterativen Algorithmus ist es dem NN möglich Parameter zu lernen und eine nicht-lineare Funktion zu approximieren. Wenn nicht anders gekennzeichnet, stammen die Informationen aus Calin (2020) und Wythoff (1993).

Im ersten Schritt müssen die Gewichte initialisiert werden. Dies kann auf unterschiedliche Weisen erreicht werden. Narkhede et al. (2022) unterscheideten in zwei Hauptkategorien: *ohne vorheriges Training* und *mit vorherigem Training*. Dabei können die Gewichte ohne vorherigem Training zufällig aus einer Gleich- oder Normalverteilung generiert oder durch datengetriebene Ansätze ermittelt werden. Kombination aus zufälligen und datengetriebenen Ansätzen existieren ebenfalls.

Die Anpassung des Netzwerkes an die Daten wird mit Hilfe der zuvor festgelegten Verlustfunktion ermittelt. Um die Qualität zu steigern, wird die Gradientenmethode angewandt. Diese soll das Minimum der Verlustfunktion finden. Der Gradient wird dabei abhängig von den Parametern berechnet und im Anschluss werden die Parameter mit der Lernrate aktualisiert. Dieses Verfahren ist auch als *gradient descent* (Gradientenabstieg) bekannt.

3.4. Initialisierung der Gewichte

In Abschnitt 3.3 wurden verschiedene Ansätze zur Initialisierung der Gewichte kategorisiert. Folgend werden zwei Methoden ohne vorheriges Training thematisiert. Diese basieren auf Varianzskalierung und bewährten sich durch Verbesserungen im Trainingsprozess.

Basierend auf der Annahme, dass die Aktivierungsfunktion linear ist, entwickelten Glorot and Bengio (2010) eine Methode, bei der die Gewichte aus einer Gleichverteilung stammen. Später wurde diese mit einem Faktor skaliert. Es konnte sichergestellt werden, dass durch diese Verteilung die Varianz der Ausgabeschicht nicht zu klein oder zu groß wurde. Jedoch gilt diese Annahme nicht für alle Aktivierungsfunktionen, weshalb He et al. (2015) die *Xavier-Verteilung*[2] für gleichgerichtete Aktivierungsfunktionen optimierten.

[2] Als *Xavier-Verteilung* ist die von Glorot and Bengio (2010) entwickelte Methode bekannt.

3.5. Regularisierung

Mit Hilfe von neuronalen Netzwerken können komplexe Zusammenhänge gelernt werden. NN sind anfällig für Über- und Unteranpassung. Überanpassung bedeutet, dass das Modell zu stark an die Trainingsdaten angepasst ist und auf Testdaten wesentlich schlechter abschneidet. Analog ist die Rede von Unteranpassung, wenn das Modell nicht ausreichend komplex ist, um die Beziehungen der Daten abzubilden.

3.5.1. Dropout

Hinton et al. (2012) und Srivastava et al. (2014) konnten zeigen, dass das zufällige Aussetzen eines Neurons einer HL einen signifikanten Einfluss auf die Leistung haben kann.

Für die Implementierung von Dropout Layer (DO) im Vorwärtsschritt wird ein Vektor r^l benötigt, der Werte einer Bernoulli-Verteilung enthällt, sodass $r_j^l \sim Bernoulli(p)$. Die Dropout-Rate p ist die Wahrscheinlichkeit mit der ein Neuron ausgesetzt wird. Die Gleichung 3.1 wird um einen Zwischenschritt ergänzt (Srivastava et al., 2014):

$$\begin{aligned} \tilde{\mathbf{y}}^l &= \mathbf{r}^l \mathbf{y}^l, \\ z_i^{(l+1)} &= w_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}) \end{aligned} \tag{3.8}$$

3.5.2. Lasso-Regularisierung

Bei der Lasso-Regularisierung (auch L1-Norm) werden die Beträge der Gewichte aufsummiert und zu der Verlustfunktion addiert. Dadurch werden die Gewichte klein gehalten oder null. Die Gewichte haben keinen Einfluss mehr auf die Vorhersage. Der Lagrange Multiplikator $\lambda_{L1} > 0$ kontrolliert wie stark hohe Gewichte bestraft werden. (Calin, 2020)

$$L_1(w) = C(w) + \lambda_{L1}|w| \tag{3.9}$$

3.5.3. Ridge-Regularisierung

Ähnlich zu der Lasso-Regularisierung hält die Ridge-Regularisierung (auch L2-Norm) die Gewichte zwischen den Schichten klein. In diesem Fall werden die quadrierten Gewichte zu der Verlustfunktion addiert. Durch das Quadrieren werden größere Gewichte bei der Ridge-Regularisierung stärker bestraft als kleine Gewichte. Zudem werden die Gewichte nicht null, sondern nur sehr klein. Ebenfalls kontrolliert der Lagrange Multiplikator $\lambda_{L2} > 0$ wie stark die Gewichte bestraft werden.

$$L_2(w) = C(w) + \lambda_{L2} w^2 \tag{3.10}$$

3.5.4. Early Stopping

Eine simple Anwendung des *Early Stopping* bricht den Trainingsprozess ab, wenn der Fehler auf den Testdaten schlechter ist, als der vorherige Fehler.

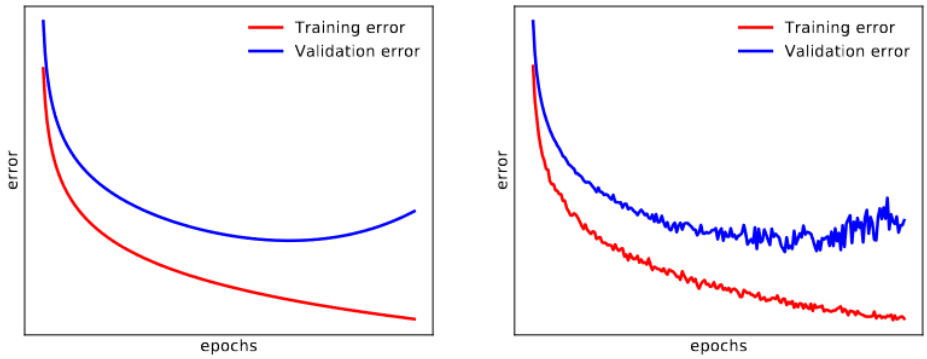

Abbildung 3.2.: Verlustfunktionen (links Theorie | rechts Realität)

Die linke Grafik aus 3.2 zeigt eine idealisierte Verlustfunktion. In diesem Fall wird abgebrochen, sobald die Verlustfunktion der Testdaten größer ist als der vorherige Wert der Verlustfunktion. Das Minimum wird erreicht. In der rechten Grafik aus 3.2 wurden die Verlustfunktionen um einen Fehlerterm erweitert. Dadurch wird die reale Welt simuliert. Es wird deutlich, dass die simple Implementierung in diesem Fall zu einer starken Unteranpassung führt. Daher ist es sinnvoll, erst nach mehrfachen Überschreiten des Optimums der Verlustfunktion abzubrechen.

Des Weiteren bietet *Early Stopping* nicht nur Vorteile hinsichtlich des Vermeidens einer Überanpassung, sondern reduziert ebenfalls die Dauer des Trainings und beschleunigt die Evaluation einer geeigneten Architektur des NN.

4. Implementierung und Auswertung

Im Folgenden werden die Ergebnisse der neuronalen Netzwerke vorgestellt. Alle Durchläufe wurden auf einem Intel Xeon E5-2678 v3 CPU mit 30MB Cache, 16GB RAM und einer GeForce GTX 1080 vollzogen.

4.1. Implementierung und Hypertuning Prozess

Das Ziel des *Hypertuning* ist eine gute Einstellung für das neuronale Netzwerk zu finden. Für jedes NN wurden mindestens 288 verschiedene Konfigurationen getestet. Für die Datensätze *Probsat on 7SAT90* und *Minisat on randomK3* wurden 576 Konfigurationen getestet, weil der Effekt von DO auf diesen untersucht wurde. Aus den getesteten Konfigurationen wurden die nach der Verlustfunktion betrachteten besten Konfigurationen ausgewählt. Da für den CPLEX zwei Datensätze vorliegen, wurde die Konfiguration auf dem kleineren Datensatz ermittelt. Für den Vergleich mit Eggensperger et al. (2018) wurden für beide Datensätze NN trainiert.

4.1.1. Anwendungen

Die Erstellung der Grafiken sowie der Evaluation der Modelle erfolgte innerhalb der Programmiersprache Python. Für die Implementierung der neuronalen Netzwerke wurde die *Open-Source-Programmbibliothek* PyTorch verwendet. Die Programmierschnittstelle *Compute Unified Device Architecture (CUDA)* von NVIDIA ermöglichte Berechnungen auf der *Graphics Processing Unit* (GPU) durchzuführen. CUDA parallelisiert Prozesse, sodass diese signifikant schneller abgearbeitet werden.

4.1.2. Modellauswahl

Es gibt vielseitige Methoden neuronale Netzwerke zu strukturieren. Im Zuge dieser Arbeit wurden zwei Strukturen getestet. Sei N^{HL} die Menge der HL eines NN, sodass $n_l^{HL} \in N^{HL}$ die Anzahl der Neuronen des l-ten HL ist. Die Lernrate für die Fehlerrückführung entspricht $\frac{1}{1000}$. Der ADAM passt die Lernrate an.

Bei der ersten Struktur enthalten alle HL die gleiche Anzahl an Neuronen, daraus folgt $n_l^{HL} = n_{l+1}^{HL}$. Diese Struktur wird als A_E gekennzeichnet. Die zweite Struktur verfügt über HL, bei denen die Anzahl der Neuronen nach dem ersten HL gleichmäßig um den Faktor $\frac{1}{\mathrm{L}}$ abnimmt. Daraus folgt $n_l^{HL} = n_{l-1}^{HL} - n_1^{HL} * \frac{1}{L}, \forall l \geq 2$. Diese Struktur wird mit A_D gekennzeichnet. Ebenso wurden für n_l^{HL} verschiedene Werte

überprüft. Dabei wurde n_0^{IL} mit ρ multipliziert, sodass $n_0^{IL} * \rho = n_1^{HL}$. Die Werte für L und ρ wurden auf $\{1, 2, 4\}$ beschränkt.

Als Aktivierungsfunktion wurde $ReLU(x)$ gewählt. Das „Vanishing " konnte ausgeschlossen werden. Durch die einfachen Berechnungen beschleunigte sich das Training.

In Abschnitt 3.1.2 wurden bereits die Vorteile des MSE (Gleichung 3.5) gegenüber dem MAE (Gleichung 3.6) und HuberLoss (Gleichung 3.7) thematisiert. Für die NN wurde ausschließlich der MSE verwendet.

Für die neuronalen Netzwerke wurde, aufgrund der in Abschnitt 3.4 vorgestellten Vorteile, ausschließlich die von He et al. (2015) vorgestellte Normalverteilung genutzt.

4.1.3. Dropout

Um die Anwendbarkeit von Dropouts zu testen, wurden zwei Modi implementiert. Im ersten Modus befindet sich eine DO mit $p = \frac{1}{5}$ nach der Eingabeschicht. Nach jeder HL befindet sich eine weitere DO mit $p = \frac{1}{2}$. Die Dropout-Raten wurden von Hinton et al. (2012) übernommen. Die Sinnhaftigkeit von Dropout-Schichten wurde auf den Datensätzen *Probsat on 7SAT90* und *Minisat on randomK3* getestet (siehe B). Es konnte nicht festgestellt werden, dass Dropout-Schichten einen Effekt auf die Verlustfunktion der Testdaten haben. Eine Überanpassung wird demnach nicht vermieden, weshalb sie bei den anderen Algorithmen nicht angewendet wurden. DO werden nicht in der Auswertung berücksichtigt.

4.1.4. Lasso- und Ridge-Regularisierung

Für die Regularisierung der Gewichte wurden mehrere Werte untersucht. Für die Analyse des Effekts der Lasso- und Ridge-Regularisierung wurden die Werte $\{0, 0.5, 0.01, 0.001\}$ für λ_{L1} und λ_{L2}, sowie die Kombinationen dieser Werte, angewendet.

4.1.5. Early Stopping

Im Evaluationsprozess wurden die neuronalen Netze mit einem Early Stopping ausgestattet. Um eine Unteranpassung zu vermeiden, wurde nach 10 Epochen ohne Verbesserung der Verlustfunktion auf den Testdaten das Training beendet. Die maximale Anzahl an Epochen betrug 200 Epochen.

4.2. Auswertung Hypertuning

	Epochen	L	$n_0^{IL} * \rho$	Struktur	Dropout	λ_{L1}	λ_{L2}
CPLEX	46	2	$373 * 1$	A_E	False	0	0
Probsat	40	2	$148 * 1$	A_E	False	$\frac{1}{1000}$	$\frac{1}{1000}$
Minisat	73	2	$152 * 2$	A_D	False	0	0
Clasp	76	2	$229 * 2$	A_E	False	0	0
Lingeling	50	4	$521 * 2$	A_E	False	$\frac{1}{1000}$	0

Tabelle 4.1.: Finale Parametereinstellungen

In der Tabelle 4.1 sind die besten Konfigurationen der berechneten NN aufgelistet. Bei jedem Datensatz wurde frühzeitig abgebrochen. Es wurden nicht einmal 100 Epochen überschritten. Eine Ursache kann eine zu restriktive Toleranz sein. Ebenso ist es denkbar, dass die Toleranz nicht zu restriktiv ist. Um die komplexen Beziehungen abzubilden benötigt ein NN nicht zwingend viele Epochen. In den meisten Fällen werden 2 HL verwendet. Für das NN des Lingeling ergab das Hypertuning hingehen 4 HL. Der ausgewählte Wert für ρ ist stärker abhängig vom Datensatz. Jedoch lässt sich $\rho = 4$ ausschließen, da es für kein NN zu Bestwerten hinsichtlich der Verlustfunktion führte. Ein stetige Architektur erzielte, außer beim *Minisat on randomK3*, die besten Ergebnisse. Der Einfluss der Lasso- und Ridge-Regularisierung ist marginal. In drei von fünf Fällen gilt $\lambda_{L1} \wedge \lambda_{L2} = 0$. In einem Fall gilt $\lambda_{L1} \vee \lambda_{L2} = 0$. In keinem NN wurde ein $\lambda_{L1} \vee \lambda_{L2} \geq \frac{1}{1000}$ verwendet.

Im Anhang D wurden Abbildungen die Verlustfunktionen des Evaluationsprozess der ausgewählten Parametereinstellungen hinterlegt.

4.3. Auswertung Qualität

	$RMSE_{NN}$	$RMSE_{QRF}^{LOCO}$	CC_{NN}	CC_{QRF}^{LOCO}
CPLEX on Regions200	6.14	0.33	0.0272	0.90
CPLEX on RCW2	7.24	0.08	0.0003	0.99
Probsat on 7SAT90	2.31	0.82	0.0045	0.36
Minisat on randomK3	3.45	0.48	-0.0064	0.89
Clasp on Rooks	3.69	0.49	-0.0092	0.98
Lingeling on Circuitfuzz	495.59	0.72	-0.0019	0.70

Tabelle 4.2.: Modell Qualität

In der Tabelle 4.2 wird für jedes Netzwerk der RMSE und die CC der logPAR10 dargestellt. Die Metriken wurden nicht mit einem *Leave-One-Configuration-Out* (LOCO) Ansatz aufgrund zu langer Berechnungsdauer ermittelt. Hinsichtlich beider Metriken erreichen die NN um ein Vielfaches schlechtere Werte als der QRF von Eggensperger et al. (2018). Die Werte für den RMSE der logPAR10 ist um den Faktor 2.8 (*Probsat on 7SAT90*) bis 90.5 (*CPLEX on RCW2*) größer. Beim *Lingeling on Circuitfuzz* traten mehrfach Unter- und Überanpassungen auf, die nicht elimiert werden konnten. Aus diesem Grund wurde dieser Datensatz nicht berücksichtigt.

Ausschließlich für den CPLEX on Regions200 ist die CC signifikant[1]. Für kein NN konnte eine signifikante positive Korrelation festgestellt werden.

[1]Der p-Wert ist ausschließlich beim CPLEX Regions200 kleiner als 0.05.

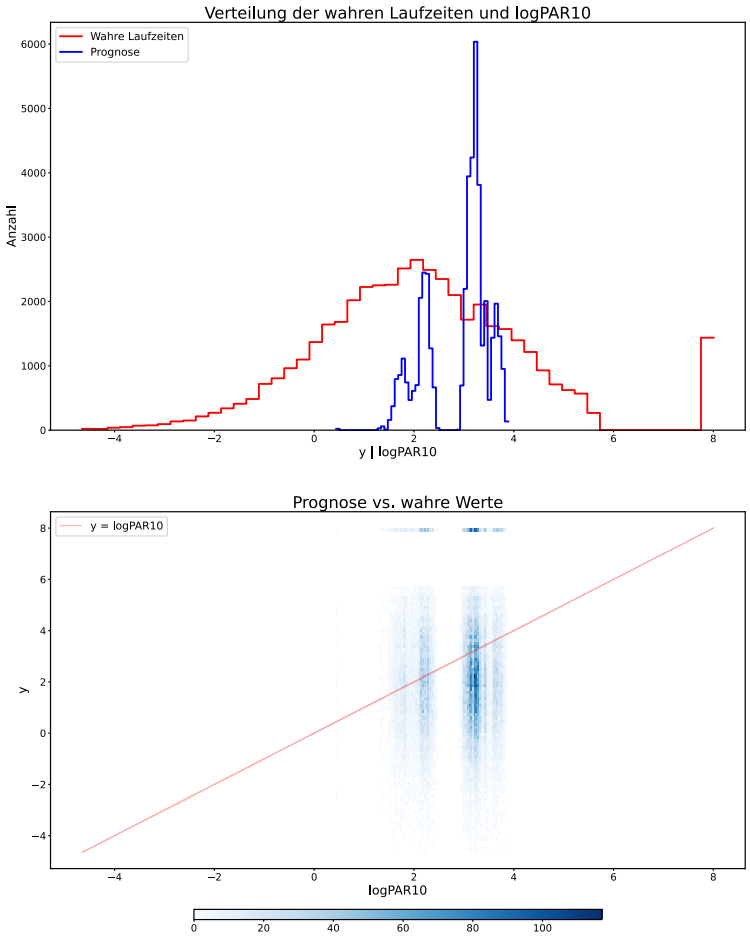

Abbildung 4.1.: Auswertung Güte (*Probsat*)

In der Abbildung 4.1 wird in der oberen Grafik die Verteilung der wahren logarithmierten Laufzeiten mit der Verteilung der prognostizierten logPAR10 verglichen. Dem NN für den Datensatz *Probsat on 7SAT90* gelingt es nicht die Laufzeiten adäquat vorherzusagen. Aus dieser Grafik kann nicht auf die Qualität des Modells geschlossen werden. Die Verteilung der wahren Laufzeiten und logPAR10 können identisch sein, obwohl die Prognosen des Modells weit von den wahren Laufzeiten entfernt sind. Dieses Problem tritt in der unteren Grafik der Abbildung 4.1 nicht auf. Es ist ein zweidimensionales Histogramm dargestellt, dass die logPAR10 (x-Achse) den wahren Laufzeiten (y-Achse) gegenübergestellt. Für die Interpretation wurde die identische Funktion[2] hinzugefügt. Die Intensität der Farbe gibt die Anzahl der Punkte in den Spektra an. Die Abbildungen für die anderen Datensätze sind im Anhang C hinterlegt.

[2]Die identische Funktion ist mit einem optimalen Modell gleichzusetzen.

5. Fazit & Ausblick

5.1. Fazit

In dieser Bachelorarbeit wurden NN als EPM untersucht. Es kamen verschiedene Architekturen und Eigenschaften zum Einsatz. Zudem wurden bewährte Regularisierungsmethoden, die eine Überanpassung an die Trainingsdaten vermeiden sollten, angewandt. Die Auswertung der Daten zeigte deutlich, dass die entwickelten NN nicht zuverlässig die Laufzeiten von unbekannten Algorithmuskonfigurationen und Instanzen prognostizieren können. Der RMSE und die CC für die logPAR10 zeigen, dass die NN in dieser Form nicht geeignet sind.

Im Ausblick werden weiterführende Methoden empfohlen, die untersucht werden können, um eine Verbesserung der NN zu bewirken.

5.2. Ausblick

Eine Reduzierung der Lernrate, mit der die Gewichte in der Fehlerrückführung aktualisiert werden, kann vermeiden, dass schnell ein schlechtes lokales Optimum gefunden wird. Dadurch wird sich jedoch der Trainingsprozess hinauszögern.

Die Auswertung des Hypertuningprozesses ergab das eine zu strenge Regulierung hinsichtlich des *Early Stopping* gewählt wurde. Reaktionen kann eine Erhöhung der Toleranz von 10 Epochen sein. Von den Epochen unabhängige Implementierungen eines *Early Stopping* hat Prechelt (2012) entwickelt und in verschiedenen Szenarien untersucht. Eine Methode ermittelt die prozentuale Differenz zwischen dem aktuellen und besten Wert der Verlustfunktion. Bei dem Überschreiten eines vordefinierten Schwellenwertes wird der Trainingsprozess abgebrochen.

Die schlechten Ergebnisse der NN können ebenfalls auf die Wahl der initialen Gewichte zurückführen. Erweiterte Methoden für das Initialisieren der Gewichte anzuwenden, kann daher sinnvoll sein und die Qualität der Netzwerke steigern. Narkhede et al. (2022) kategorisierten und bewerteten verschiedene Ansätze.

Literaturverzeichnis

Thomas Bartz-Beielstein and Sandor Markon. Tuning search algorithms for real-world applications: A regression tree based approach. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, volume 1, pages 1111–1118. IEEE, 2004.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Ovidiu Calin. *Deep Learning Architectures*. Springer International Publishing, 2020. doi: 10.1007/978-3-030-36721-3. URL http://dx.doi.org/10.1007/978-3-030-36721-3.

Monica Ciolacu, Ali Fallah Tehrani, Leon Binder, and Paul Mugur Svasta. Education 4.0-artificial intelligence assisted higher education: early recognition system with machine learning to support students' success. In *2018 IEEE 24th International Symposium for Design and Technology in Electronic Packaging(SIITME)*, pages 23–30. IEEE, 2018.

Marc Claesen, Jaak Simm, Dusan Popovic, Yves Moreau, and Bart De Moor. Easy hyperparameter search using optunity. *arXiv preprint arXiv:1412.1114*, 2014.

Katharina Eggensperger, Marius Lindauer, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107(1):15–41, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Marius Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Aclib: A benchmark library for algorithm configuration. *Lecture Notes in Computer Science*, page 36–40, 2014. doi: 10.1007/978-3-319-09584-4_4. URL http://dx.doi.org/10.1007/978-3-319-09584-4_4.

MohammadNoor Injadat, Abdallah Moubayed, Ali Bou Nassif, and Abdallah Shami. Multi-split optimized bagging ensemble model selection for multi-class educational data mining. *Applied Intelligence*, 50:4506–4528, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1):89–109, 2001.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL https://doi.org/10.1145/3065386.

Martin Leo, Suneel Sharma, and Koilakuntla Maddulety. Machine learning in banking risk management: A literature review. *Risks*, 7(1):29, 2019.

Meenal V Narkhede, Prashant P Bartakke, and Mukul S Sutaone. A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 55(1):291–322, 2022.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

Skogby Steinholtz Olof. A comparative study of black-box optimization algorithms for tuning of hyper-parameters in deep neural networks, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban

Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. Boa: The bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO-99*, volume 1, pages 525–532. Citeseer, 1999.

Lutz Prechelt. Early stopping—but when? *Neural networks: tricks of the trade: second edition*, pages 53–67, 2012.

Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Grigorios Tsoumakas. A survey of machine learning techniques for food sales prediction. *Artificial Intelligence Review*, 52(1):441–447, 2019.

Barry J Wythoff. Backpropagation neural networks: a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18(2):115–155, 1993.

Anhang A.

Verteilung der Laufzeiten

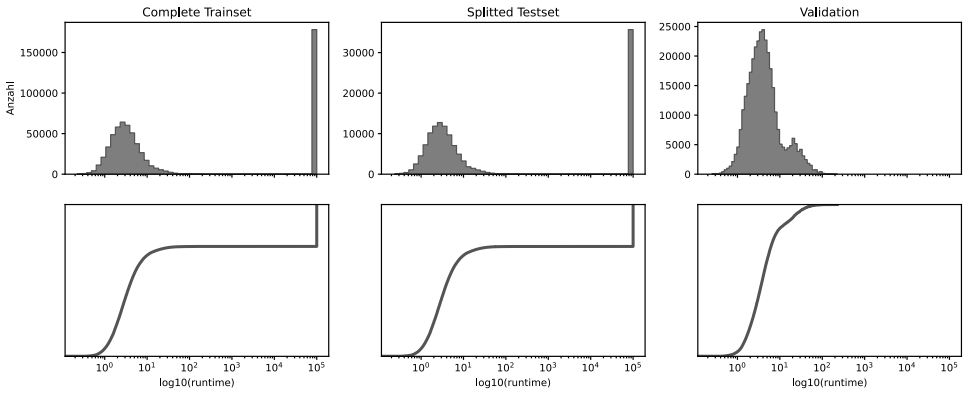

Abbildung A.1.: Verteilung der Laufzeit (*CPLEX Regions200*)

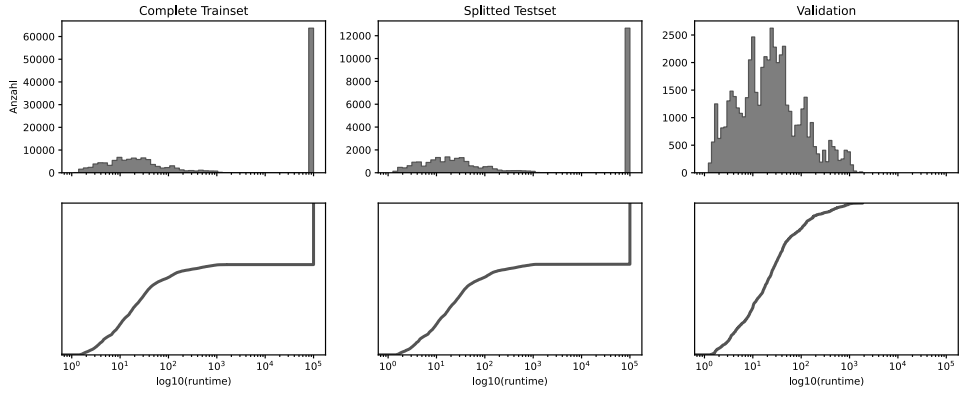

Abbildung A.2.: Verteilung der Laufzeit (*CPLEX RCW2*)

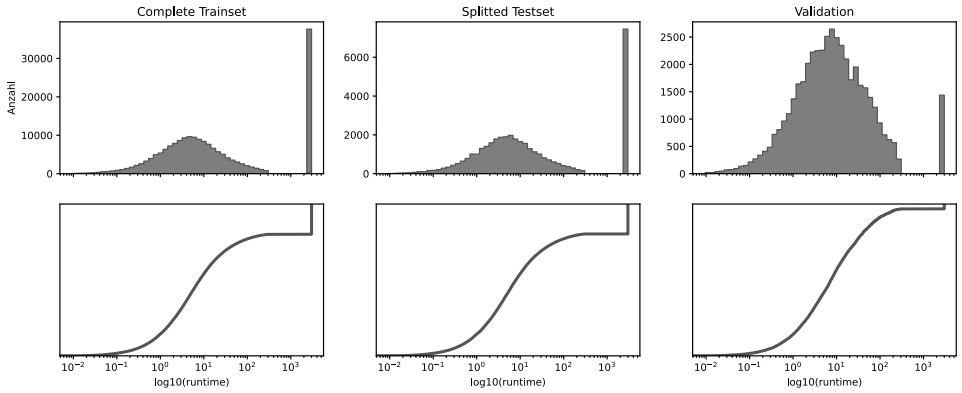

Abbildung A.3.: Verteilung der Laufzeit (*Probsat 7SAT90*)

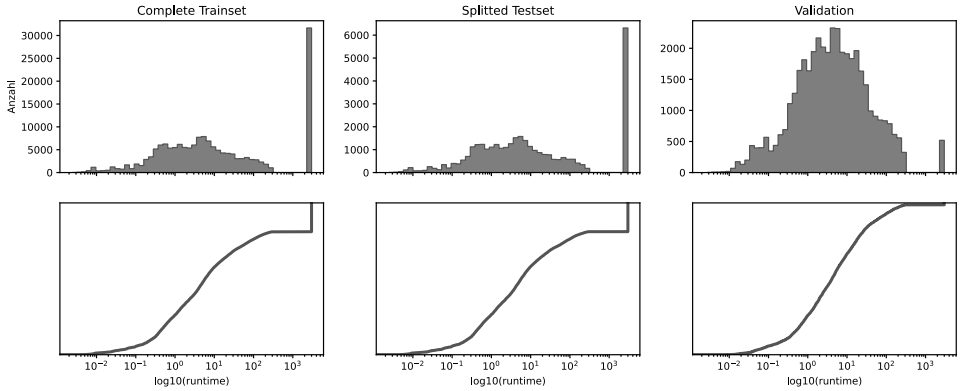

Abbildung A.4.: Verteilung der Laufzeit (*Minisat randomK3*)

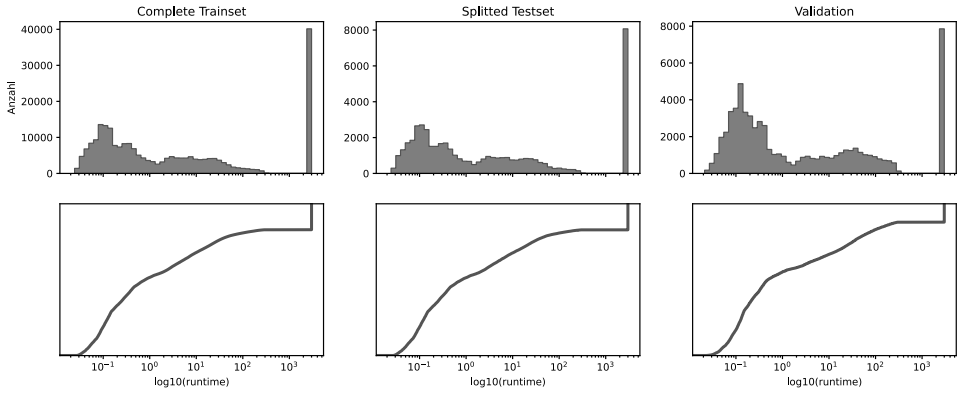

Abbildung A.5.: Verteilung der Laufzeit (*Clasp Rooks*)

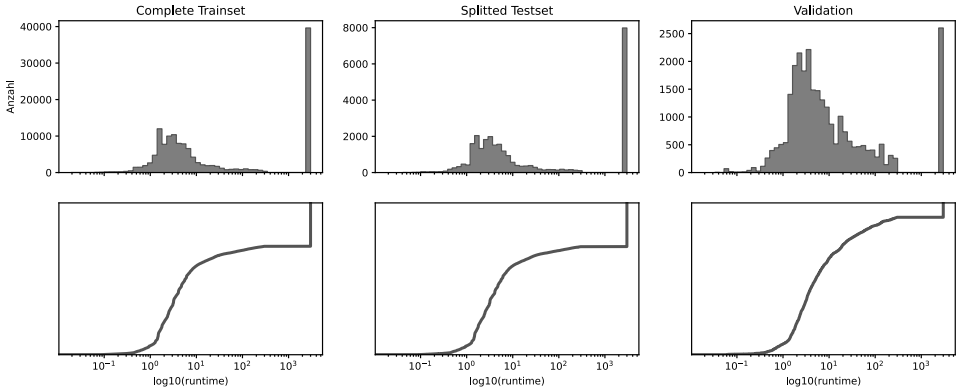

Abbildung A.6.: Verteilung der Laufzeit (*Lingeling Circuitfuzz*)

Anhang B.

Auswertung Dropout-Schichten

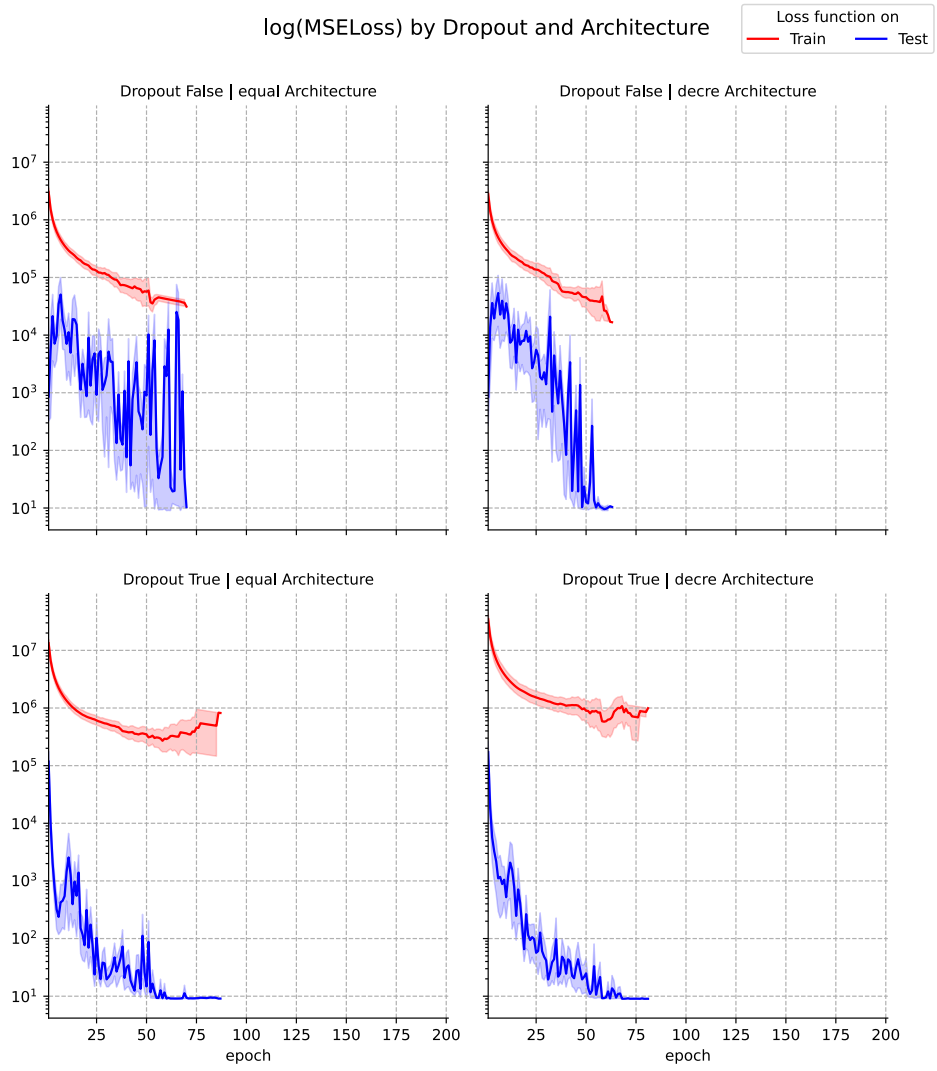

Abbildung B.1.: Auswertung Dropout-Schichten (*Probsat*)

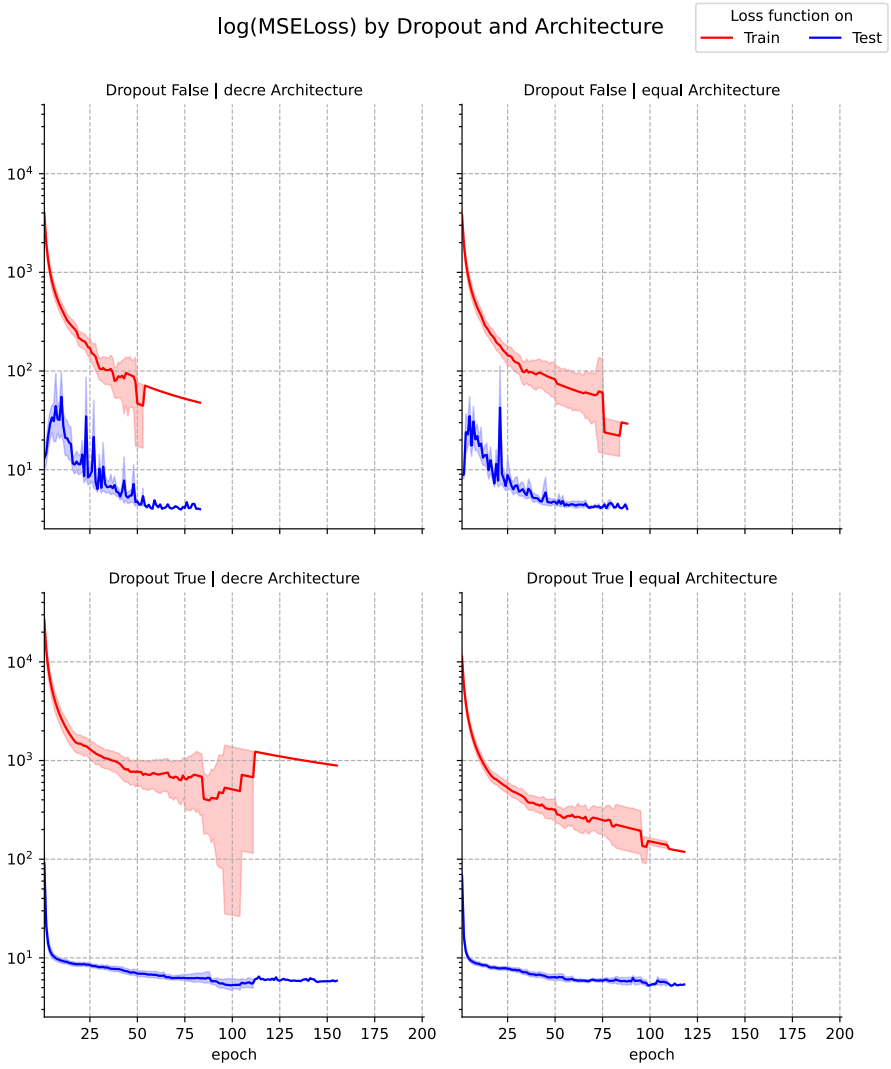

Abbildung B.2.: Auswertung Dropout-Schichten (*Minisat*)

Anhang C.

Auswertung der Güte

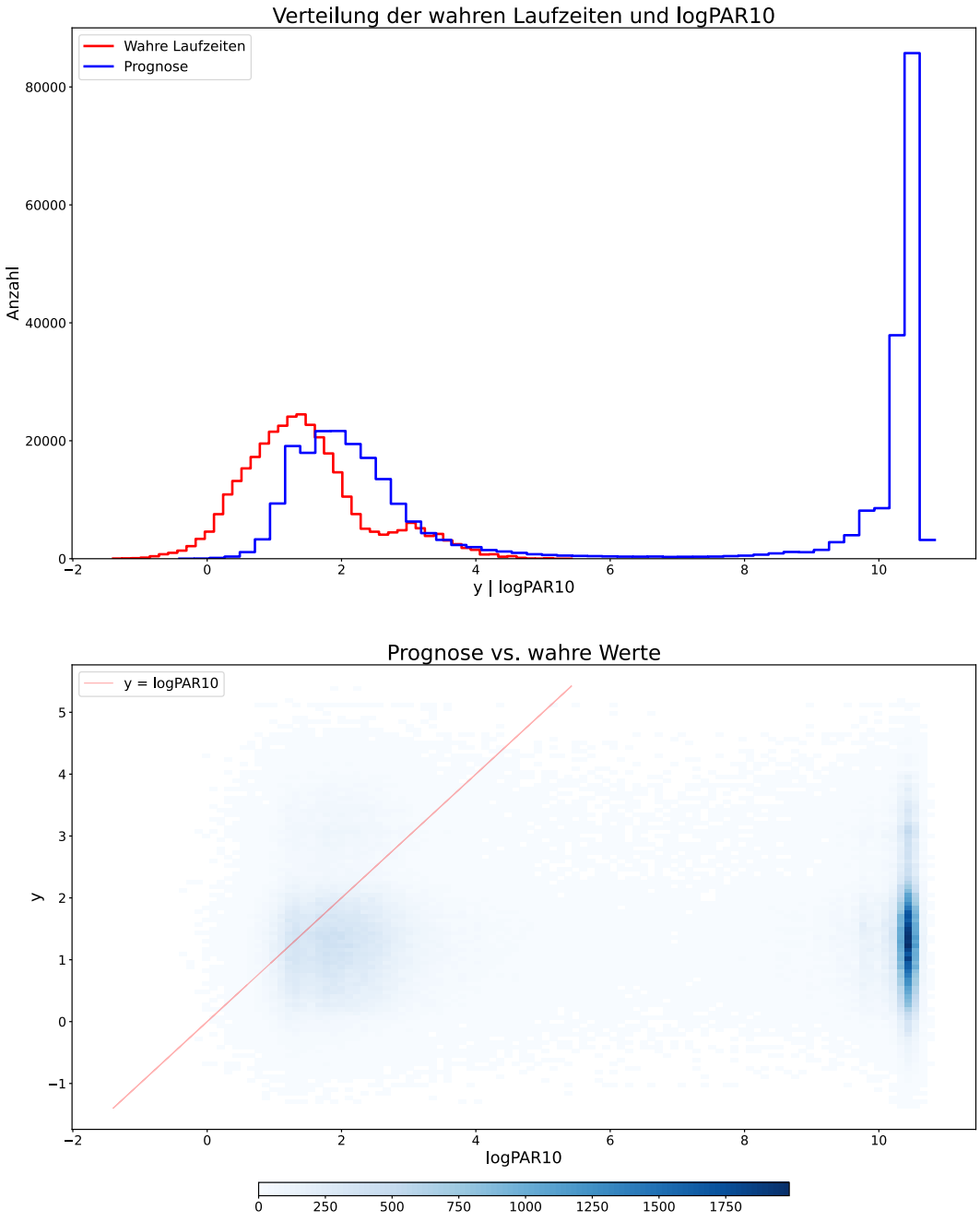

Abbildung C.1.: Auswertung Güte (*CPLEX Regions*)

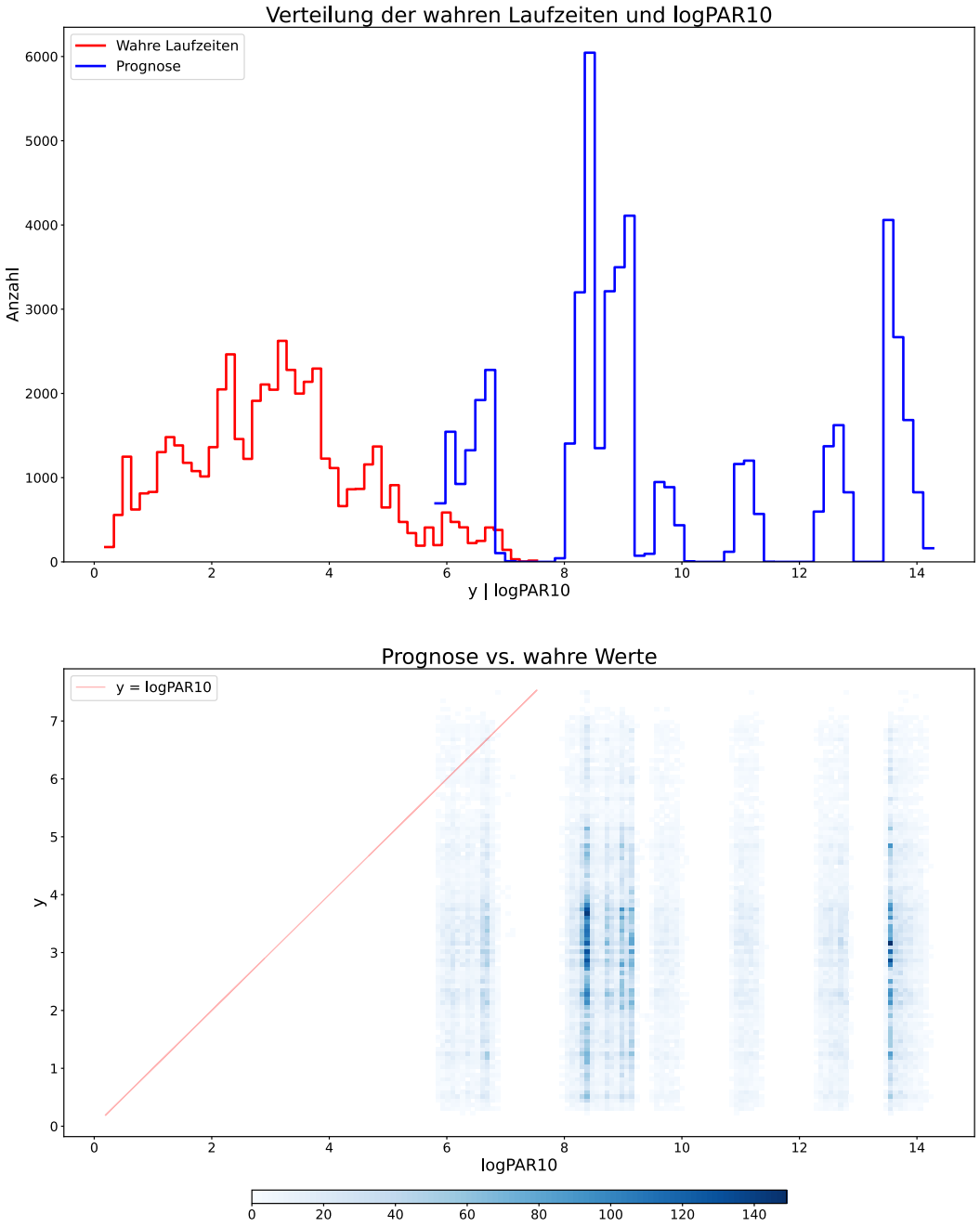

Abbildung C.2.: Auswertung Güte (*CPLEX RCW2*)

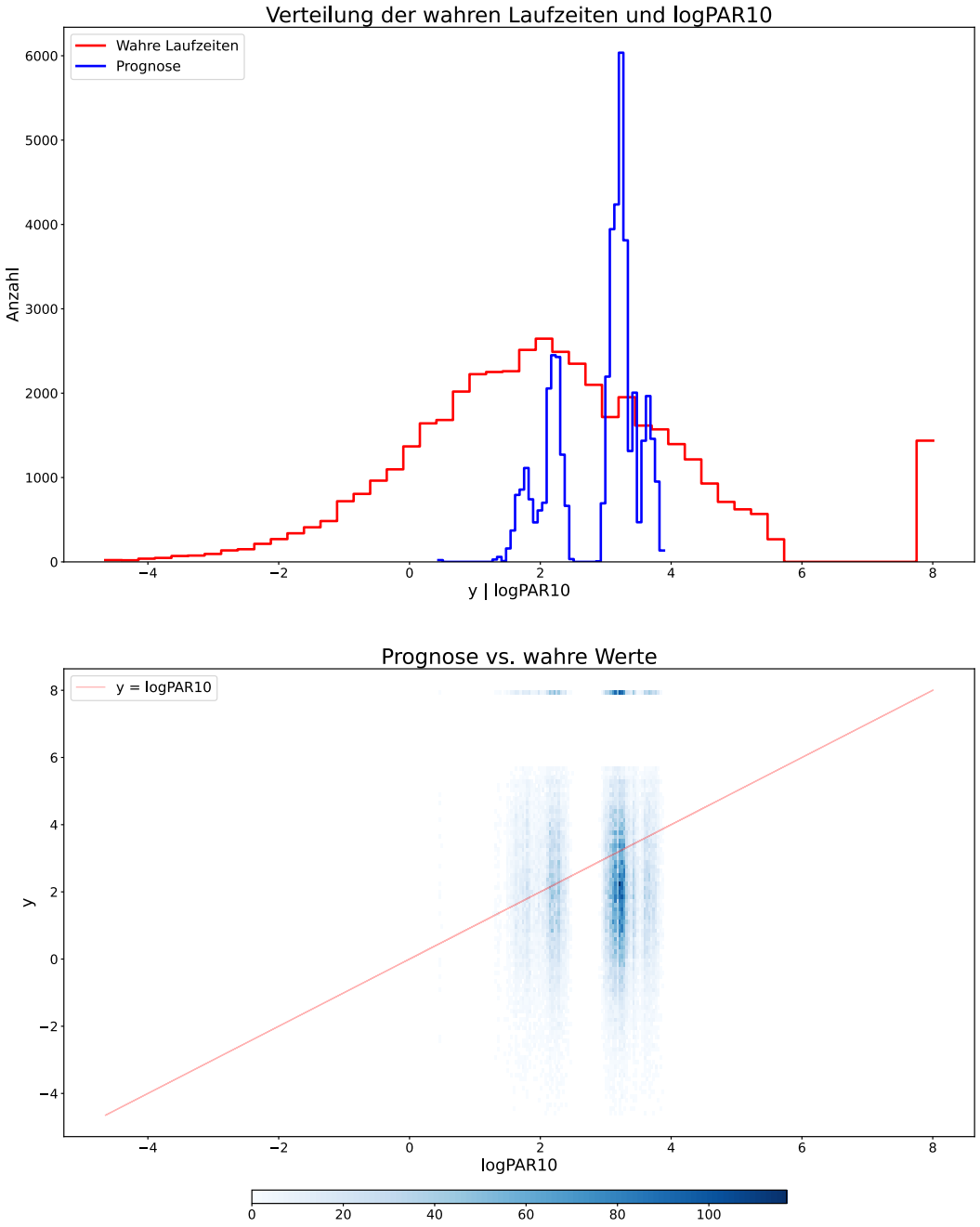

Abbildung C.3.: Auswertung Güte (*Probsat*)

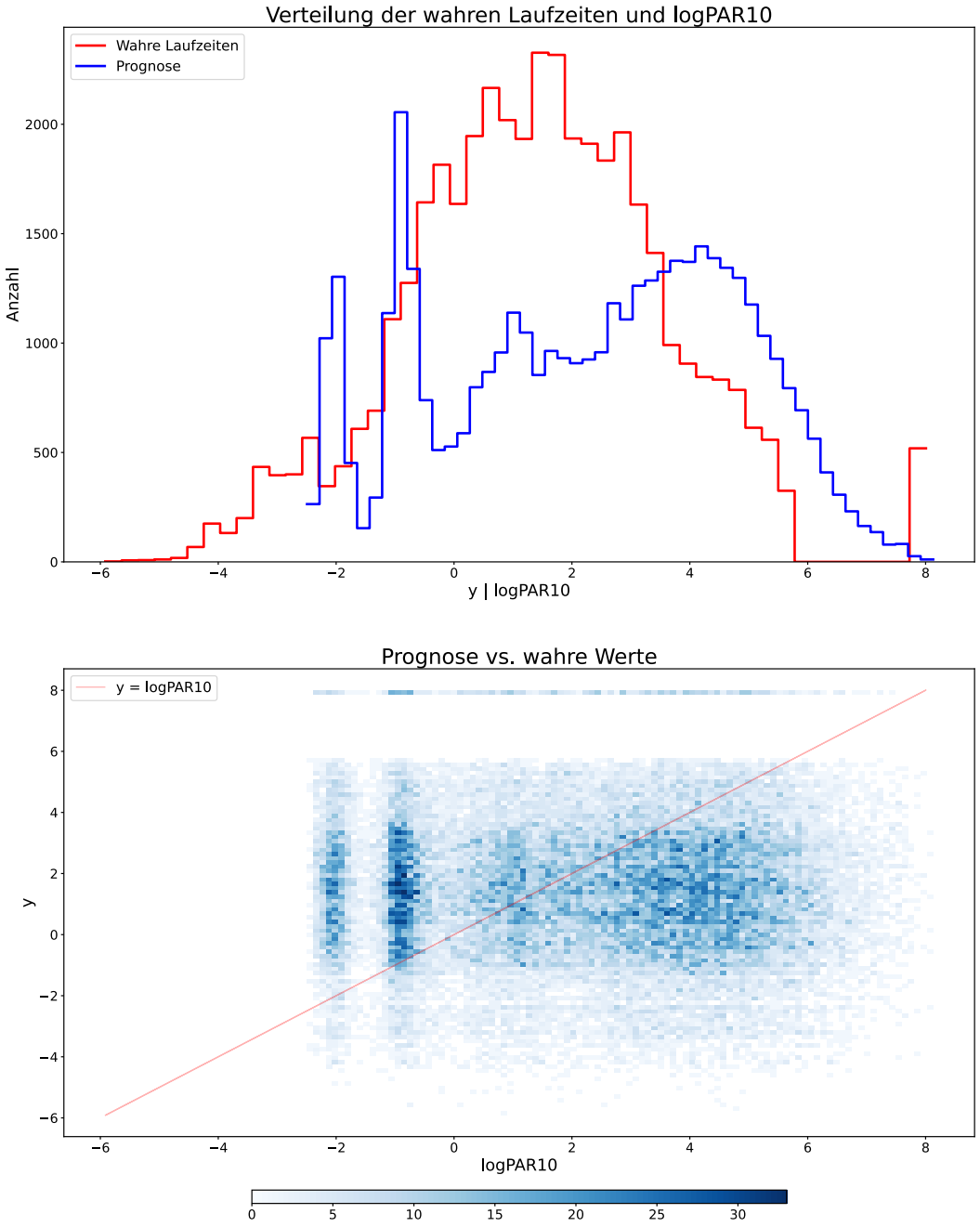

Abbildung C.4.: Auswertung Güte (*Minisat*)

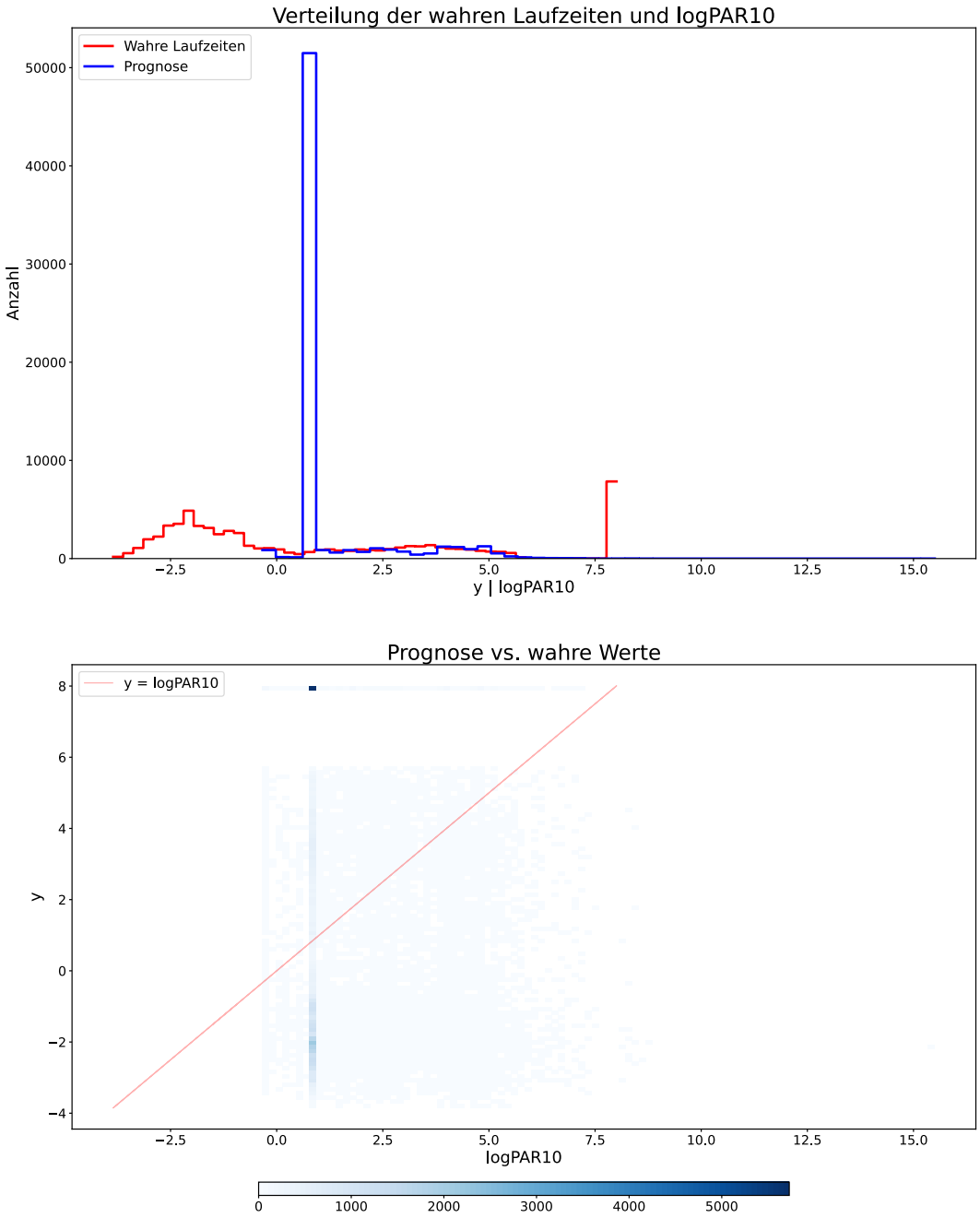

Abbildung C.5.: Auswertung Güte (*Clasp Rooks*)

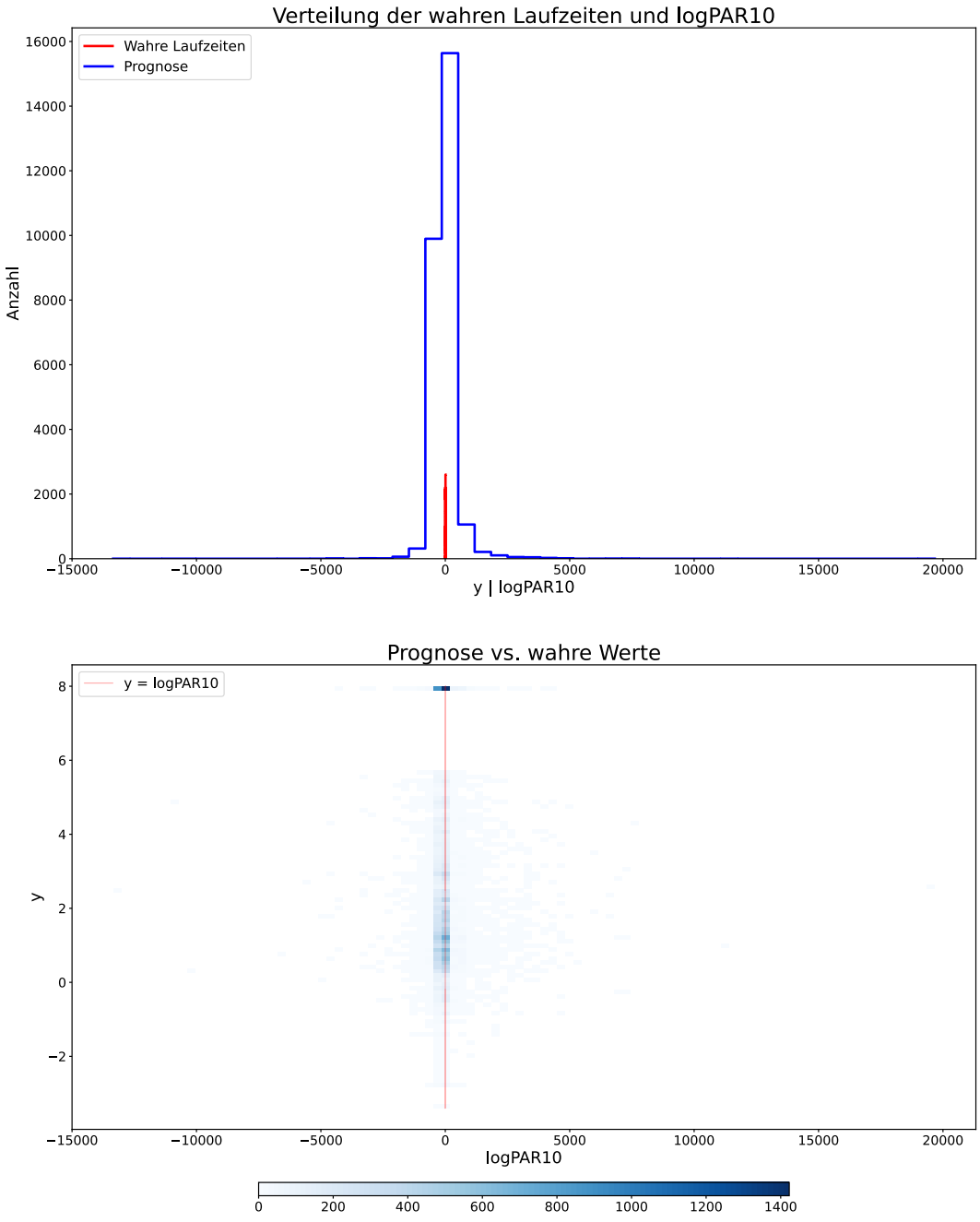

Abbildung C.6.: Auswertung Güte (*Lingeling CF*)

Anhang D.

Verlustfunktionen der ausgewählten NN

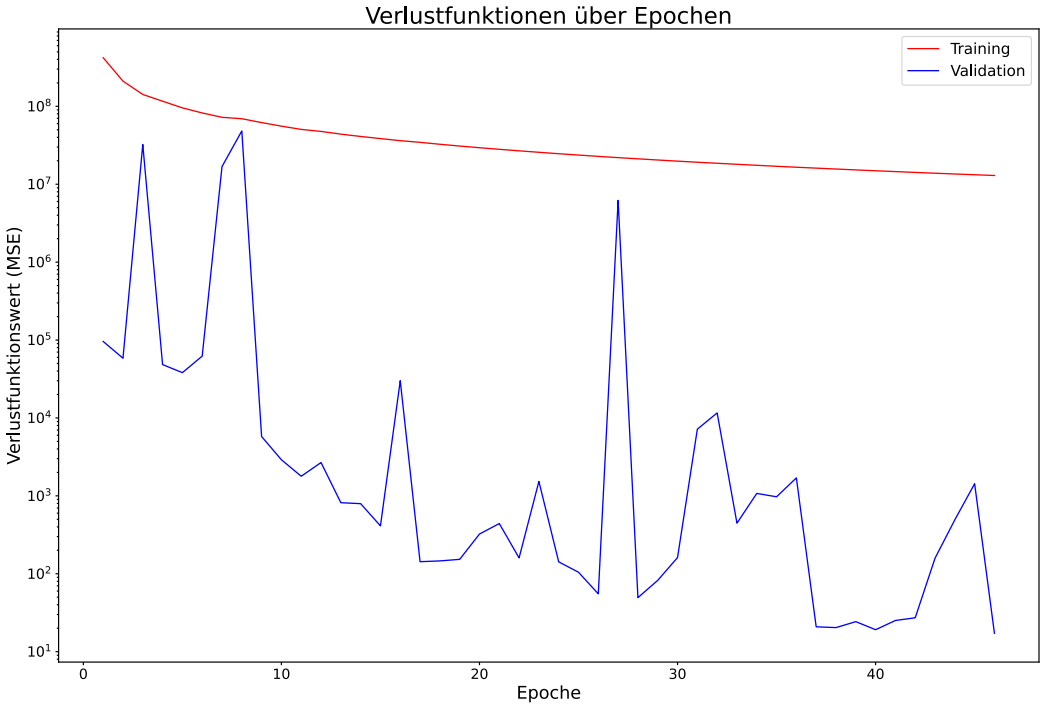

Abbildung D.1.: Verlustfunktion des gewählten NN (*CPLEX RCW2*)

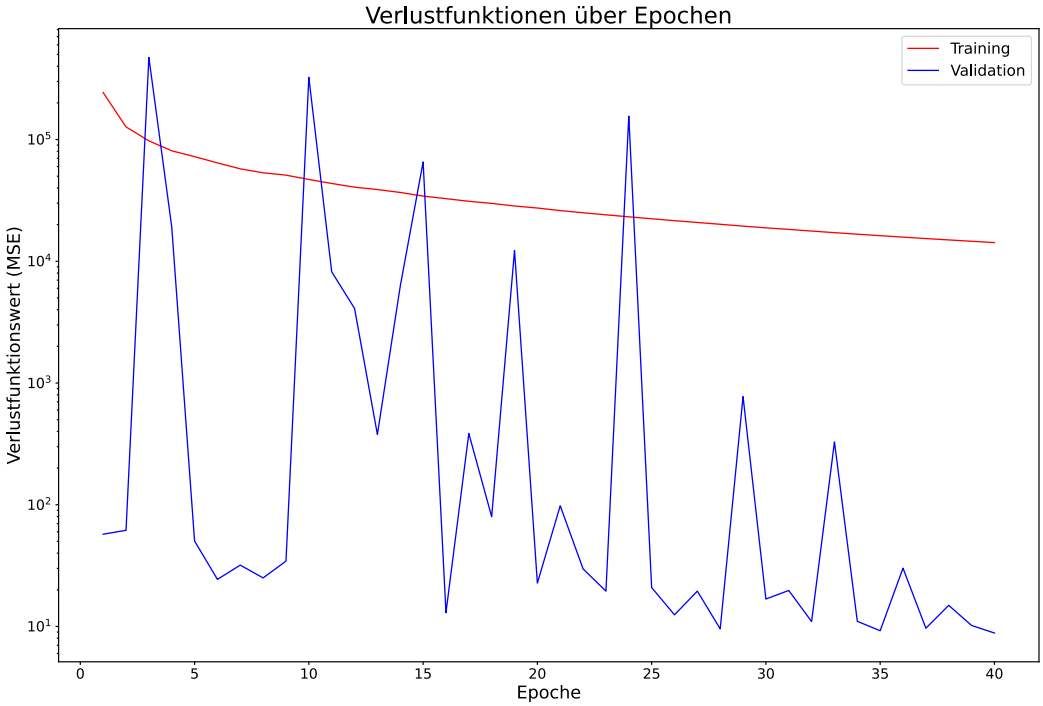

Abbildung D.2.: Verlustfunktion des gewählten NN (*Probsat*)

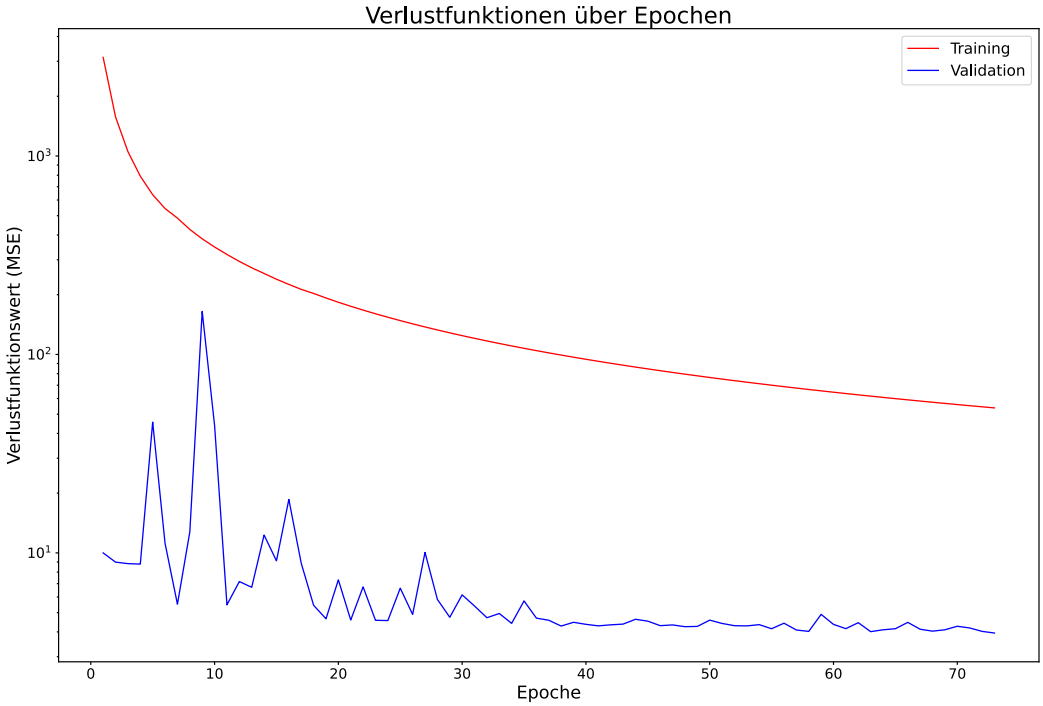

Abbildung D.3.: Verlustfunktion des gewählten NN (*Minisat*)

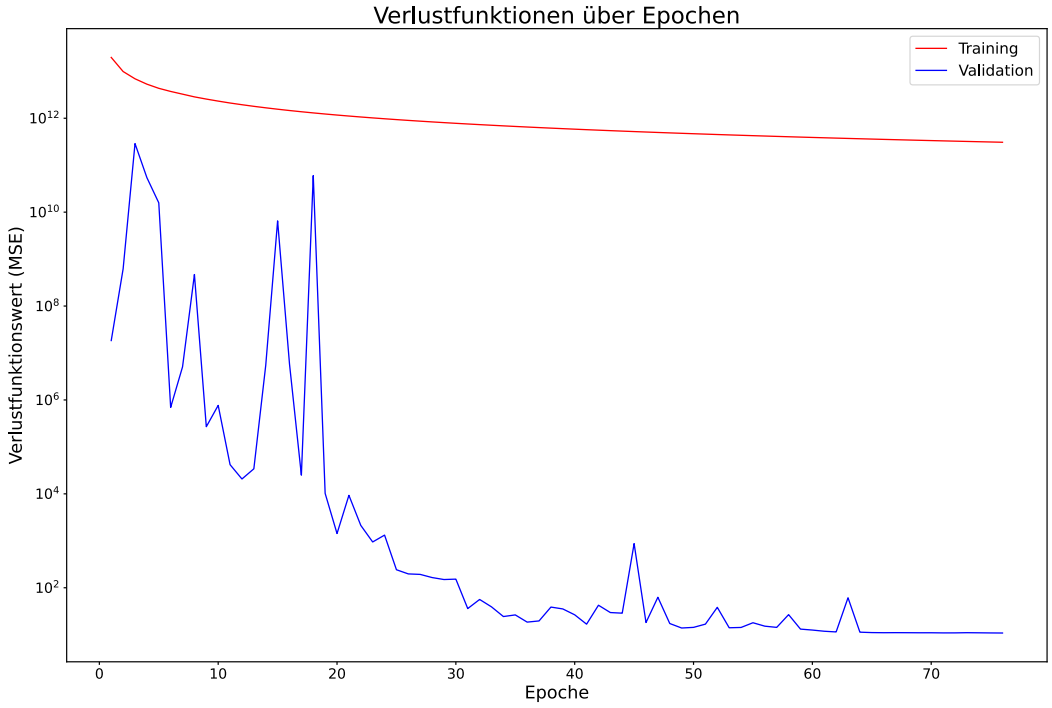

Abbildung D.4.: Verlustfunktion des gewählten NN (*Clasp Rooks*)

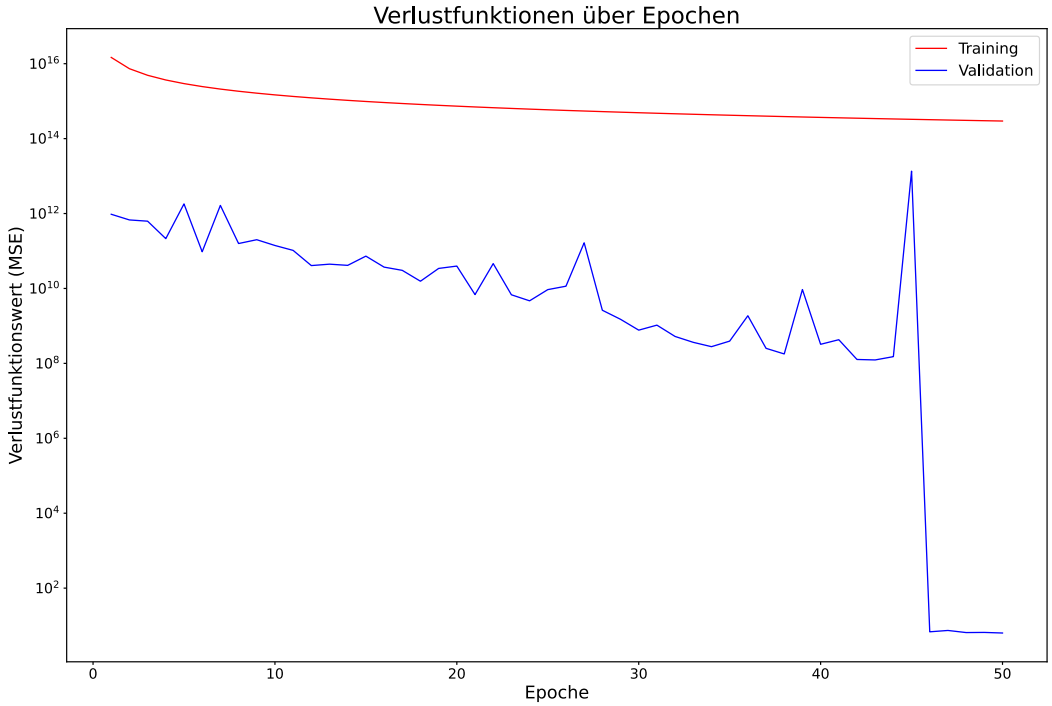

Abbildung D.5.: Verlustfunktion des gewählten NN (*Lingeling CF*)

Versicherung

Name: Foitzik

Vorname: Gregor Felix

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und keine anderen, als die angegebenen, Quellen benutzt habe. Die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen habe ich als solche kenntlich gemacht. Diese Versicherung gilt auch für alle gelieferten Datensätze, Zeichnungen, Skizzen oder grafischen Darstellungen. Des Weiteren versichere ich, dass ich das Merkblatt zum „Umgang mit Plagiaten“[1] gelesen habe.

Bielefeld, den 10. Februar 2023

Foitzik

Unterschrift

[1] www.wiwi.uni-bielefeld.de/organisation/pamt/~/organisation/pamt/uploads/PlagiatInfo-BlattStudenten.pdf