HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG

Fakultät Informatik

Vorhersage von Energieverbrauch

Ausarbeitung im Fach Neuronale Netze

Prof. Dr. Oliver Bittel

Sommersemester 2008

von

Max Nagl

nagl@fh-konstanz.de

Andreas Hofmann

ahofmann@htwg-konstanz.de

Inhaltsverzeichnis

1. Aufgabenstellung

Ziel dieser Ausarbeitung ist es, ein neuronales Netz zu entwickeln, welches aufgrund der Attributwerte Datum, Uhrzeit, Außentemperatur, Luftfeuchtigkeit, Sonneneinstrahlung und Windgeschwindigkeit den stündlichen Verbrauch von elektrischer Energie und von Kalt- und Warmwasser vorhersagen kann.

Als Daten, sowohl zum Lernen als auch zum Testen der entwickelten Netze, stehen in einer Datei die Attributwerte von einem Haus zur Verfügung, bei dem sechs Monate lang alle Werte gemessen wurden.

Dabei sollen zwei Problemstellungen gesondert betrachtet werden:

Interpolationsproblem In diesem Szenario trainieren wir mit einer Auswahl von Daten aus dem gesamten Messungszeitraum und testen das trainierte Netz dann ebenfalls mit Daten aus dem Messungszeitraum. Das Netz muss also Werte für Datenpunkte finden, die innerhalb des Trainingsdatenraums liegen, also die trainierten Daten interpolieren.

Extrapolationsproblem In diesem Szenario trainieren wir mit einer Auswahl von Daten aus den ersten vier Monaten des Messungszeitraums und testen das trainierte Netz dann mit Daten aus den letzten beiden Monaten. Der Testdatenraum liegt also außerhalb des Trainingsdatenraums, das Netz muss extrapolieren.

Als Maßstab für die Qualität der resultierenden Netze ziehen wir den MSE (Mean squared error) der Testergebnisse und deren Verhältnis zum MSE der Trainingsergebnisse heran. Anstrebenswertes Ziel wäre für das Interpolationsproblem ein Test-MSE von unter zwei Prozent, für das Extrapolationsproblem aufgrund dessen generell schlechterer Lösbarkeit [Lohninger, 2007] ein Test-MSE von unter 20 Prozent.

2. Lösungsansatz

In diesem Kapitel beschreiben wir, welchen Lösungsansatz wir gewählt haben, um die optimalen neuronalen Netze zu finden. Dabei gehen wir auf die Aufarbeitung der Daten zur besseren Verarbeitung, die Startkonfiguration der Netze sowie das Testverfahren, mit dem die optimalen Werte für die verschiedenen Parameter gefunden werden sollen, ein.

2.1. Aufbereitung der Daten

Die Daten werden aus der Datei „building.dat“ eingelesen. Diese liegt im Klartextformat vor und enthält die Daten in Tabellenform, wobei eine Zeile eine Messung darstellt und die einzelnen Werte durch ein oder mehrere Leerzeichen getrennt sind. Die ersten acht Spalten stellen die Eingangsdaten dar, die nachfolgenden drei Spalten die Ergebnisse. Um die gegebenen Daten sinnvoll mittels eines neuronalen Netzes verarbeiten zu können, müssen noch einige Anpassungen vorgenommen werden.

Zunächst sollten die Datumsangaben, die hier als Monatstag, Monat und Jahr vorliegen, in eine angesichts des relativ kleinen Messzeitraums von sechs Monaten angemessene Form gebracht werden. Die Jahreszahl zum Beispiel liefert für den Messzeitraum keine sinnvoll verwertbare Information, wir geben diese Spalte also nicht an das Neuronale Netz weiter. Ebenso fragwürdig ist der Informationsgehalt des Monatstages, da sich der Energie- und Wasserverbrauch eher am Wochenzyklus (Werktags/Wochenende) orientiert als am Monatszyklus. Wir berechnen also für jede Messung aus den vorhandenen Datumsangaben den Wochentag und füttern diesen in das Neuronale Netz.

Die Uhrzeitspalte ist im 24-Stunden-Format mit zwei angehängten Nullen dargestellt. Messungen erfolgen also immer zur vollen Stunde, der Wertebereich läuft von 0 für Mitternacht bis 2300 für 23 Uhr. 2400 kommt als Wert nicht vor. Wir normieren diese

Spalte, indem ihre Werte durch 2400 teilen, damit alle Werte im Intervall $[0, 1)$ liegen. Die Ausgabewerte werden auf den Bereich zwischen 0 und 1 normiert.

2.2. Einteilung in Trainings- und Testdaten

Wie in Kapitel 1 erwähnt betrachten wir zwei getrennte Problemstellungen, Dateninterpolation und Datenextrapolation. Für jedes dieser beiden Szenarien müssen wir die gegebenen Daten unterschiedlich in Trainings- und Testdaten aufteilen.

Für die Interpolation wählen wir Trainings- und Testdaten aus dem gesamtem Messbereich. Dabei sollten sich beide Datensätze nicht überschneiden, da ein Test auf bereits trainierte Datensätze das Ergebnis verzerren würden. Außerdem sollten die Trainingsdaten so gewählt werden, dass jede Stunde gleichmäßig trainiert wird. Dies erreichen wir, in dem wir zum trainieren jede n-te Zeile wählen, so dass n kein Teiler von 24 ist. Wir wählen deshalb jeden fünften Datensatz zum Trainieren aus. Zum Testen verwenden wir dann jeden fünften Datensatz mit einem Offset von zwei Datensätzen. Dies stellt sicher, dass Trainings- und Testdaten disjunkt sind.

Für die Extrapolation teilen wir den Messzeitraum auf. Wir wählen aus den ersten vier Monaten Datensätze zum Trainieren aus. Um die Anzahl der Datensätze und damit den Zeitaufwand des Trainings gering zu halten, wählen wir auch hier nur jeden fünften Datensatz aus. Parallel dazu wählen wir die Testdaten aus den letzten beiden Monaten, ebenfalls jeden fünften Datensatz. Da hier mit den Daten der Monate September bis Dezember trainiert wird und anschließend mit den Daten der Monate Januar und Februar getestet wird kann das Netz vermutlich nicht auf die jahreszeitlichen Veränderungen zwischen Herbst und Winter trainiert werden. Es ist ein schlechteres Ergebnis zu erwarten.

2.3. Netzarchitektur

Um den Rahmen dieser Ausarbeitung nicht zu sprengen beschränken wir uns auf die Betrachtung von Feed-Forward-Netzen. Die Eingabeschicht nimmt die acht Eingabewerte auf, die Ausgabeschicht hat drei Neuronen, jeweils eines den Ausgabewert für elektrische Energie, Warm- und Kaltwasser Verbrauch (normiert auf den Bereich zwischen 0 und 1, siehe Abschnitt 2.1).

Es wird ein oder zwei versteckte Schichten geben. Die ideale Anzahl der Schichten sowie der Neuronen pro Schicht wird durch das Testen von verschiedenen Variationen ermittelt.

2.4. Testablauf

Wir fahren mehrere Testserien, wobei wir stets die optimale Einstellung für einen bestimmten Parameter suchen. Die in einer Testserie gefundene optimale Einstellung wird dann für nachfolgende Testserien übernommen.

Wir suchen in folgender Reihenfolge nach den optimalen Parameterwerten:

1. Anzahl der Neuronen pro Schicht
2. verwendete Trainingsfunktion
3. verwendete Performanzfunktion
4. verwendete Transferfunktionen

Dieser Ablauf wird insgesamt viermal durchgeführt, jeweils zweimal für jedes Szenario, einmal für Netze mit einer versteckten Schicht und einmal für Netze mit zwei versteckten Schichten.

Der allgemeine Ablauf einer Testserie ist folgender: Zuerst werden die Daten aus der Datei geladen und wie in Abschnitt 2.1 beschrieben aufbereitet. Außerdem werden die Testfälle geladen. Diese sind je nach zu testendem Parameter unterschiedlich. Dann wird pro Testfall jeweils drei neuronale Netze mit dem im Testfall definierten Parametern erstellt, trainiert und getestet. Jede Netzkonfiguration wird also drei Testdurchläufen unterworfen. Bei jedem Testdurchlauf wird pro Ausgabewert jeweils der MSE und MAE jeweils für Trainings- und Testdaten errechnet. Um eine Netzwerkkonfiguration mit den anderen vergleichen zu können, berechnen wir für jeden Testdurchlauf den Durchschnitt der drei MSE-Werte der Testdaten. Aus diesen wählen wir den kleinsten als für diese Konfiguration repräsentativ aus. Indem wir nur den Testdurchlauf mit dem kleinsten Wert nehmen, statt den Durchschnittswert aller Testdurchläufe zu bilden, verhindern wir, dass eine Netzkonfiguration durch einzelne Ausreißer überproportional abgewertet wird.

Nachdem für alle Netzkonfigurationen der Gesamt-MSE berechnet wurde, wählen wir die Konfiguration mit dem niedrigsten Wert.

2.4.1. Neuronenanzahl

Um den Zeitaufwand in Grenzen zu halten, bestimmen wir die Neuronenanzahl pro Schicht in zwei Stufen. Zunächst variieren wir die Anzahl grob von 5 bis 100 mir Schrittweite 5. So werden wir den ungefähren Bereich, in dem das Optimum liegen muss, finden. Dort führen wir noch einmal eine feinkörnigere Testserie durch.

2.4.2. Trainingsfunktionen

Wir testen das gesamte Spektrum an Trainingsfunktionen, die Matlab bietet, mit Ausnahme von trainbfgc, da diese die Benutzung des Neural Network Model Reference Adaptive Controllers voraussetzt. Alle zu testenden Trainingsfunktionen sind in Tabelle 2.3 aufgelistet.

Funktion	Beschreibung
trainb	Batch Training with weight and bias learning rules
trainbfg	BFGS quasi-Newton backpropagation
trainbr	Bayesian regularization backpropagation
trainc	Cyclical order incremental training with learning functions
traincgb	Conjugate gradient backpropagation with Powell-Beale restarts
traincgf	Conjugate gradient backpropagation with Fletcher-Reeves updates
traincgp	Conjugate gradient backpropagation with Polak-Ribiére updates
traingd	Gradient descent backpropagation
traingda	Gradient descent with adaptive learning rate backpropagation
traingdm	Gradient descent with momentum backpropagation
traingdx	Gradient descent with momentum and adaptive learning rate backpropagation
trainlm	Levenberg-Marquardt backpropagation
trainoss	One step secant backpropagation
trainr	Random order incremental training with learning functions
trainrp	Resilient backpropagation
trains	Sequential oder incremental training with learning functions
trainscg	Scaled conjugate gradient backpropagation

Tabelle 2.1.: Die für uns relevanten Trainingsfunktionen von Matlab [Demuth u. a., 2007]

2.4.3. Performanzfunktionen

Wir testen verschiedene Performanzfunktionen. Wir lassen MSE aus, da dessen Ergebnisse bei höherem Rechenaufwand äquivalent zu denen der SSE-Funktion wären. Alle zu testenden Performanzfunktionen sind in Tabelle 2.3 aufgelistet.

Funktion	Beschreibung
MAE	Mean absolute error
SSE	Sum squared error
MSEREG	Mean squared error with regularization
MSEREGEC	Mean squared error with regularization and economization

Tabelle 2.2.: Die für uns relevanten Performanzfunktionen von Matlab [Demuth u. a., 2007]

2.4.4. Transferfunktionen

Als Aktivierungs- bzw. Transferfunktion werden verschiedene Funktionieren getestet. Da wir viele Trainingsfunktionen mit Backpropagation testen wollen, müssen alle Transferfunktionen differenzierbar sein. Wir gehen davon aus, das wir mit der logarithmisch-sigmoiden Funktion die besten Ergebnisse erzielen werden. Alle zu testenden Transferfunktionen sind in Tabelle 2.3 aufgelistet.

Funktion	Beschreibung
purelin	Linear Transfer Function
logsig	Log-Sigmoid Transfer Function
tansig	Tan-Sigmoid Transfer Function

Tabelle 2.3.: Die für uns relevanten Transferfunktionen von Matlab [Demuth u. a., 2007]

2.4.5. Startkonfiguration

Bevor wir durch Testen die optimalen Parameter ermitteln, legen wir eine Startkonfiguration für die neuronalen Netze fest. Diese ist Tabelle 2.4 zu entnehmen.

Parameter	Wert
Netztyp	Feed-Forward
Epochen	1000
Trainingsfunktion	trainrp
Performanzfunktion	sse
Transferfunktion	logsig

Tabelle 2.4.: Startkonfiguration unserer neuronalen Netze

3. Ergebnisse

3.1. Neuronenanzahl

3.1.1. Interpolation

Wir führen zuerst die Testserien für neuronale Netze mit einer versteckten Schicht durch. Die erste Testserie (Tabelle 3.1) zur groben Bestimmung der optimalen Neuronenanzahl zeigt beim Testen keinen eindeutig überlegenen MSE-Wert. Aus Abbildung 3.1 wird ersichtlich, dass der MSE beim Training mit steigender Neuronenanzahl zwar sinkt, aber beim Testen in etwa gleich bleibt. Das Generalisierungsverhalten wird also schlechter, je mehr Neuronen die Schicht besitzt. Wir wählen für die Feinabstimmung deshalb den Bereich von 15 bis 25 Neuronen.

Layersize	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
5	14,592165	0,005777	0,056148	0,005779	0,008184
10	12,250263	0,004850	0,050112	0,005092	0,007124
15	11,152128	0,004415	0,048089	0,004834	0,006785
20	9,376879	0,003715	0,044247	0,004143	0,005079
25	10,714647	0,004242	0,047163	0,004713	0,006397
30	9,368219	0,003709	0,042999	0,004087	0,005565
35	9,019662	0,003571	0,042602	0,004053	0,005345
40	8,763218	0,003469	0,041895	0,004003	0,005080
45	8,526387	0,003375	0,041744	0,004216	0,005422
50	8,668573	0,003432	0,043165	0,004364	0,005412
55	7,898110	0,003127	0,039893	0,004306	0,005864
60	7,509657	0,002973	0,039499	0,004171	0,005268
65	7,523095	0,002978	0,040004	0,004567	0,005515
70	6,920648	0,002740	0,038300	0,003934	0,004961
75	6,890441	0,002728	0,037839	0,004869	0,006518
80	6,831944	0,002705	0,037960	0,004309	0,005675
85	6,684178	0,002646	0,037723	0,004491	0,005662
90	5,839840	0,002312	0,035693	0,003660	0,003607
95	6,134294	0,002428	0,035991	0,004263	0,005314
100	6,634683	0,002627	0,037144	0,004490	0,005339

Tabelle 3.1.: Grobbestimmung der Neuronenanzahl (Interpolation, drei Schichten)

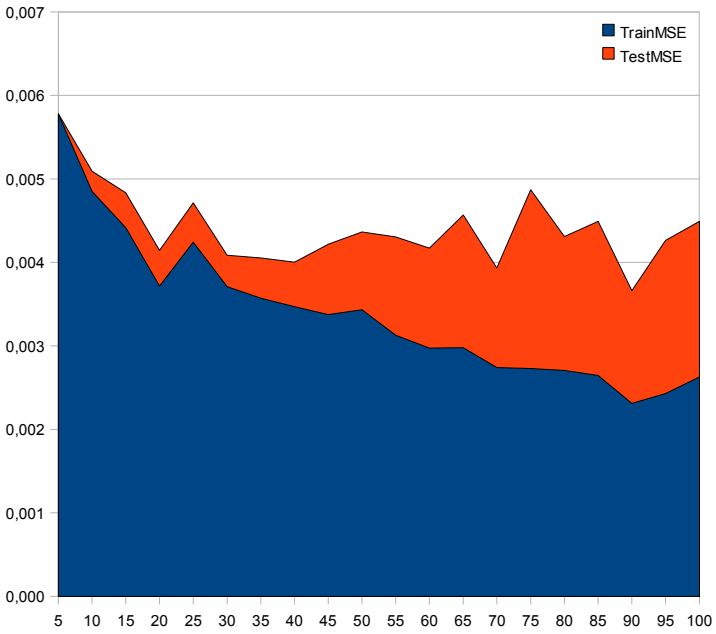

Abbildung 3.1.: Visualisierung der Ergebnisse aus Tabelle 3.1

Die zweite Testserie mit einer versteckten Schicht (Tabelle 3.2) zeigt ein über den gesamten Wertebereich ein gutes Generalisierungsverhalten. Abbildung 3.2 zeigt ein relativ konstantes Verhältnis zwischen Trainings-MSE und Test-MSE besteht. Wir wählen 18 als optimale Neuronenanzahl für die Interpolation mit einer versteckten Schicht. An dieser Stelle ist das Verhältnis zwischen Trainings-MSE und Test-MSE sehr gut und die Umgebung ist relativ flach, was einen Ausreißer recht unwahrscheinlich macht.

Layersize	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
15	10,325266	0,004088	0,045623	0,004304	0,046718
16	11,579651	0,004584	0,048068	0,004975	0,049203
17	10,084156	0,003992	0,045056	0,004470	0,047205
18	10,011823	0,003964	0,045544	0,004194	0,046683
19	9,719597	0,003848	0,045075	0,004219	0,046785
20	10,667371	0,004223	0,047204	0,005005	0,050477
21	10,112103	0,004003	0,045124	0,004326	0,046837
22	9,386564	0,003716	0,043655	0,004290	0,045906
23	9,841065	0,003896	0,044850	0,004198	0,045725
24	10,142176	0,004015	0,045741	0,004804	0,049461
25	9,969994	0,003947	0,044983	0,004384	0,046936

Tabelle 3.2.: Feinbestimmung der Neuronenanzahl (Interpolation, drei Schichten)

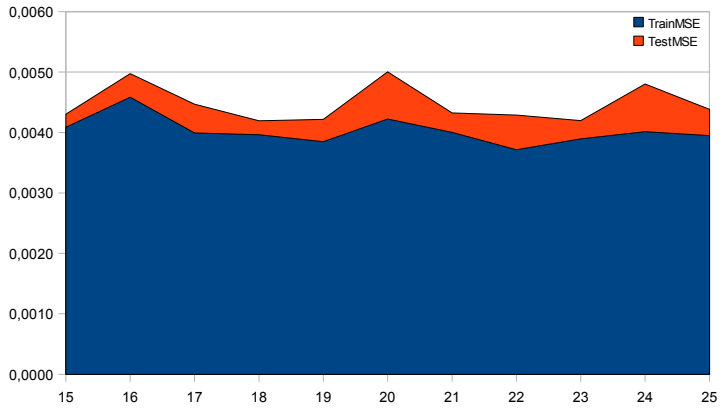

Abbildung 3.2.: Visualisierung der Ergebnisse aus Tabelle 3.2

Wir gehen nun zum Testen der Netze mit zwei versteckten Schichten über. Zunächst erfolgt wieder die Grobbestimmung, deren Ergebnisse in Tabelle 3.3 dargestellt sind. Wir nutzen hier unser Vorwissen aus den vorherigen Testserien und beschränken uns gleich auf den Bereich zwischen 5 und 20. Dies ist auch gut so, denn da wir diesmal zwei Parameter variieren, quadriert sich natürlich die Anzahl der Testfälle. Die Schrittweite ist wieder fünf. Wie sich an Abbildung 3.3 zeigt sich auch hier von einigen Ausreißern abgesehen ein Trend zur schlechteren Generalisierung bei höherer Neuronenanzahl. Interessanterweise scheint die Anzahl der Neuronen in der ersten versteckten Schicht stärkere Auswirkungen zu haben als die in der Zweiten.

Layersize1	Layersize2	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
5	5	15,794287	0,006253	0,056646	0,006405	0,057399
5	10	10,665044	0,004222	0,047625	0,004447	0,048644
5	15	15,070906	0,005966	0,055627	0,006231	0,055879
5	20	10,295375	0,004076	0,046475	0,004314	0,047667
10	5	9,741996	0,003857	0,045862	0,004161	0,046763
10	10	9,474178	0,003751	0,044971	0,004249	0,047046
10	15	9,310274	0,003686	0,044422	0,004013	0,045522
10	20	8,564569	0,003391	0,042480	0,004128	0,045408
15	5	9,143876	0,003620	0,043490	0,003947	0,044664
15	10	8,496477	0,003364	0,041232	0,003826	0,043545
15	15	8,313077	0,003291	0,041826	0,004025	0,045030
15	20	7,708744	0,003052	0,040303	0,003869	0,044898
20	5	10,435377	0,004131	0,048077	0,005041	0,051782
20	10	7,953199	0,003149	0,040768	0,003824	0,043895
20	15	7,892118	0,003124	0,040045	0,003905	0,044276
20	20	7,301486	0,002891	0,038841	0,003791	0,043524

Tabelle 3.3.: Grobbestimmung der Neuronenanzahl (Interpolation, vier Schichten)

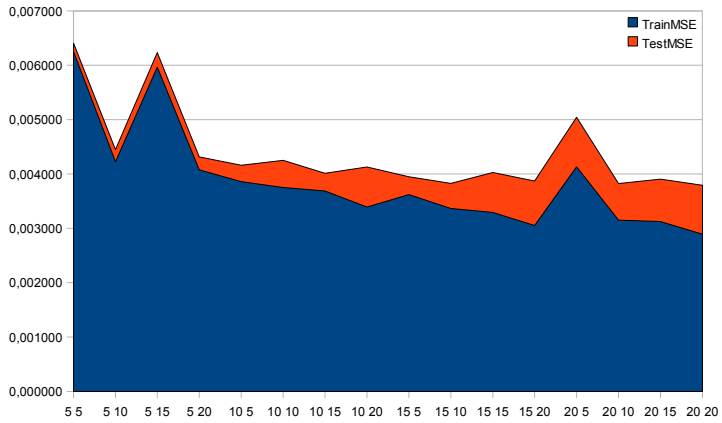

Abbildung 3.3.: Visualisierung der Ergebnisse aus Tabelle 3.3

Für die Feinabstimmung betrachten wir deshalb den Bereich zwischen 5 und 15, allerdings nur mit Schrittweite 2, um die Anzahl der Testfälle zu verringern. Die Auswahl der „optimalen“ Neuronenanzahl ist hier schwieriger als zuvor. Wir entscheiden uns für 13 und 13, da diese Konfiguration niedrige MSE-Werte mit noch gutem Generalisierungsverhalten verbindet. Eine Alternative wäre 5 und 7, wo die MSE-Werte zwar höher sind, aber eine sehr gute Generalisierung gegeben ist (siehe Abbildung 3.4).

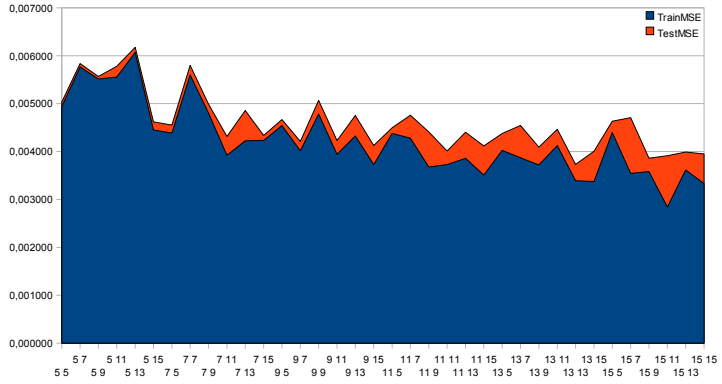

Abbildung 3.4.: Visualisierung der Ergebnisse aus Tabelle 3.4

Wir haben somit also für das Interpolationsproblem ein dreischichtiges Netz mit 18 Neuronen in der versteckten Schicht und ein vierschichtiges Netz mit jeweils 13 Neuronen in den versteckten Schichten, mit denen wir weitere Testserien durchführen werden.

Layersize1	Layersize2	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
5	5	12,480591	0,004941	0,050914	0,005035	0,050976
5	7	14,566104	0,005766	0,055158	0,005837	0,054726
5	9	13,924490	0,005512	0,053871	0,005567	0,054459
5	11	14,037150	0,005557	0,053510	0,005780	0,054539
5	13	15,331651	0,006070	0,055674	0,006175	0,056325
5	15	11,245175	0,004452	0,049134	0,004619	0,049614
7	5	11,077337	0,004385	0,048195	0,004552	0,048373
7	7	14,143492	0,005599	0,053186	0,005798	0,053447
7	9	12,125778	0,004800	0,049539	0,004987	0,050556
7	11	9,910774	0,003924	0,045434	0,004316	0,047675
7	13	10,669942	0,004224	0,047552	0,004856	0,050139
7	15	10,688377	0,004231	0,047873	0,004337	0,047660
9	5	11,471913	0,004542	0,048186	0,004669	0,048388
9	7	10,147763	0,004017	0,046450	0,004212	0,047254
9	9	12,083840	0,004784	0,050738	0,005068	0,051706
9	11	9,956294	0,003942	0,046110	0,004231	0,047110
9	13	10,928089	0,004326	0,046588	0,004754	0,048014
9	15	9,419654	0,003729	0,043909	0,004127	0,046538
11	5	11,059612	0,004378	0,046924	0,004497	0,047473
11	7	10,806245	0,004278	0,047228	0,004757	0,049163
11	9	9,286210	0,003676	0,044652	0,004409	0,047246
11	11	9,420511	0,003729	0,044259	0,004009	0,045768
11	13	9,749899	0,003860	0,043843	0,004403	0,046801
11	15	8,866199	0,003510	0,044078	0,004118	0,046319
13	5	10,165814	0,004024	0,047393	0,004378	0,048359
13	7	9,774503	0,003870	0,044445	0,004543	0,046994
13	9	9,394778	0,003719	0,043924	0,004092	0,045916
13	11	10,423802	0,004127	0,046831	0,004464	0,047995
13	13	8,570439	0,003393	0,041565	0,003732	0,043191
13	15	8,524041	0,003375	0,041574	0,004007	0,044241
15	5	11,108076	0,004397	0,048267	0,004633	0,049223
15	7	8,951191	0,003544	0,043020	0,004708	0,047670
15	9	9,047929	0,003582	0,043406	0,003861	0,044022
15	11	7,179871	0,002842	0,038153	0,003911	0,042225
15	13	9,133658	0,003616	0,043143	0,003988	0,044830
15	15	8,410732	0,003330	0,041242	0,003950	0,044749

Tabelle 3.4.: Feinbestimmung der Neuronenanzahl (Interpolation, vier Schichten)

3.1.2. Extrapolation

Kommen wir nun zur Bestimmung der optimalen Neuronenanzahl für das Extrapolationsproblem. Auch hier betrachten wir zunächst das dreischichtige Netz und führen mit diesem zwei Teststufen durch. Die grobgranulierte Bestimmung (Tabelle 3.5) zeigt, dass die Generalisierungsfähigkeit bei Extrapolation allgemein wesentlich schlechter ist als bei Interpolation. Lagen dort die MSE-Werte von Training und Testen relativ nahe beieinander, so sind hier die MSE-Werte des Testens um Größenordnungen schlechter als die des Trainings. Wie bei der Interpolation lässt sich in Abbildung 3.5 eine stetige Verbesserung des Trainings-MSE bei steigender Neuronenanzahl ausmachen.Der Test-MSE währenddessen bleibt hier nicht nur gleich, sondern scheint sogar leicht zu steigen, wobei wir davon ausgehen, dass die Werte bei 10, 40 und 65 Ausreißer darstellen.

Layersize	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
5	9,514506	0,005412	0,054218	0,091185	0,253210
10	7,121360	0,004051	0,046801	0,065537	0,207026
15	6,653733	0,003785	0,045039	0,093715	0,240029
20	6,031674	0,003431	0,043040	0,127887	0,300678
25	5,502661	0,003130	0,041720	0,122938	0,313229
30	4,848048	0,002758	0,039652	0,127655	0,314699
35	5,383692	0,003062	0,040930	0,109226	0,284763
40	4,581852	0,002606	0,037716	0,070227	0,224971
45	4,761850	0,002709	0,038783	0,096777	0,261035
50	4,501166	0,002560	0,037078	0,108050	0,279597
55	3,901191	0,002219	0,035475	0,149584	0,339299
60	3,885651	0,002210	0,035139	0,134999	0,328065
65	4,142190	0,002356	0,035893	0,340768	0,563228
70	3,895156	0,002216	0,034606	0,120327	0,304685
75	3,840269	0,002184	0,034660	0,162251	0,365278
80	3,265304	0,001857	0,031741	0,173803	0,366848
85	3,362653	0,001913	0,032677	0,154367	0,347431
90	3,101382	0,001764	0,030885	0,180949	0,390501
95	3,189240	0,001814	0,031461	0,158472	0,353183
100	3,202914	0,001822	0,031609	0,155152	0,353447

Tabelle 3.5.: Grobbestimmung der Neuronenanzahl (Extrapolation, drei Schichten)

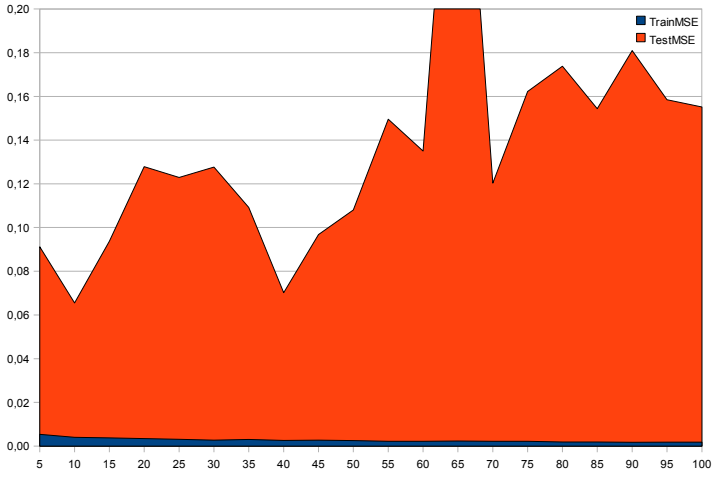

Abbildung 3.5.: Visualisierung der Ergebnisse aus Tabelle 3.5

Es scheint also bei der Extrapolation ebenfalls angebracht zu sein, keine zu hohe Neuronenanzahl auszuwählen. Zur Feinabstimmung (Tabelle 3.6) untersuchen wird wieder den Bereich zwischen 15 und 25. Wie aus Abbildung 3.6 ersichtlich wird, ist der Trainings-MSE in diesem Bereich relativ zum Test-MSE-Wertebereich nahezu konstant. Wir suchen also eine Neuronenanzahl mit möglichst geringem Test-MSE. Bei 22 liegt der beste Wert, die benachbarten Werte weichen aber extrem ab. Deshalb wählen wir 18 aus, den zweitbesten Test-MSE mit deutlich flacherer Umgebung.

Layersize	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
15	6,951998	0,003954	0,045520	0,124952	0,319375
16	6,446648	0,003667	0,044178	0,127828	0,304408
17	6,357929	0,003617	0,044425	0,122361	0,318631
18	6,121005	0,003482	0,043613	0,095756	0,269565
19	6,482989	0,003688	0,044593	0,160671	0,358835
20	6,366558	0,003621	0,043997	0,114866	0,307361
21	6,423592	0,003654	0,044622	0,214533	0,386057
22	5,895147	0,003353	0,043356	0,081851	0,238233
23	6,231913	0,003545	0,044685	0,152602	0,351575
24	5,548180	0,003156	0,041771	0,112429	0,295281
25	5,988831	0,003407	0,043997	0,122746	0,309717

Tabelle 3.6.: Feinbestimmung der Neuronenanzahl (Extrapolation, drei Schichten)

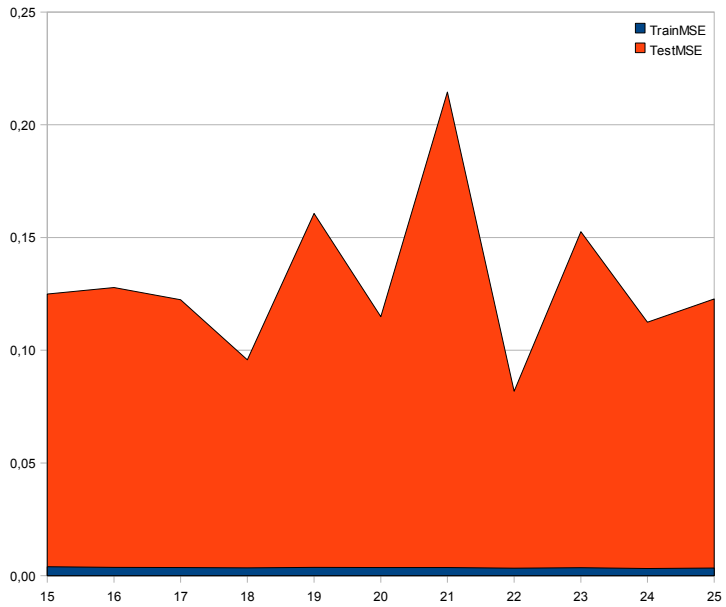

Abbildung 3.6.: Visualisierung der Ergebnisse aus Tabelle 3.6

Nun betrachten wir noch die Testserien für Extrapolation mit zwei versteckten Schichten. Die Grobbestimmung (Tabelle 3.7) liefert abgesehen von der für Extrapolation üblichen

deutlich schlechteren Generalisierungsfähigkeit ähnliche ambivalente Ergebnisse wie bei der Interpolation.

Layersize1	Layersize2	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
5	5	8,901668	0,005064	0,051652	0,117418	0,292853
5	10	8,547194	0,004862	0,050785	0,122402	0,302582
5	15	8,263902	0,004701	0,049464	0,067984	0,221530
5	20	6,737559	0,003833	0,045426	0,120140	0,286547
10	5	7,398386	0,004208	0,048357	0,089481	0,256339
10	10	5,024844	0,002858	0,040354	0,080075	0,224536
10	15	5,383667	0,003062	0,040983	0,134864	0,319483
10	20	5,474213	0,003114	0,041143	0,105470	0,271977
15	5	6,570412	0,003737	0,045345	0,084317	0,236076
15	10	5,267511	0,002996	0,040924	0,068122	0,224636
15	15	5,202779	0,002959	0,038654	0,095671	0,268368
15	20	4,104990	0,002335	0,036569	0,082504	0,225695
20	5	6,097318	0,003468	0,043067	0,118982	0,299097
20	10	4,701745	0,002674	0,038712	0,126753	0,312667
20	15	4,010167	0,002281	0,036585	0,131460	0,311490
20	20	3,923531	0,002232	0,035730	0,090317	0,260921

Tabelle 3.7.: Grobbestimmung der Neuronenanzahl (Extrapolation, vier Schichten)

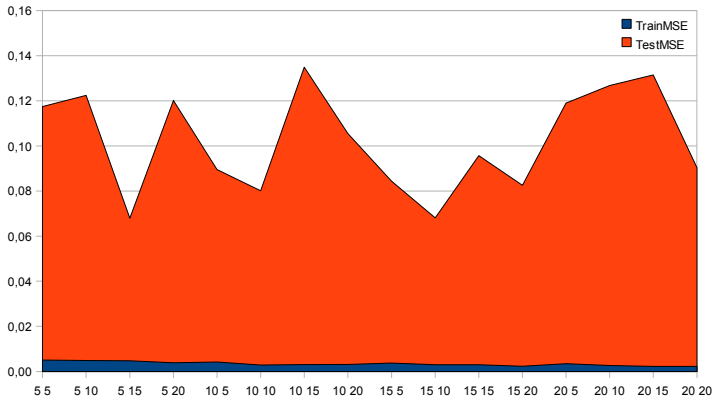

Abbildung 3.7.: Visualisierung der Ergebnisse aus Tabelle 3.7

Für die Feinabstimmung (Tabelle 3.8) wählen wir den selben Wertebereich wie bei der Interpolation, 5 bis 15. Wie aus Abbildung 3.8 ersichtlich gibt es an den Stellen 5/7, 11/9 und 15/5 besonders niedrige Test-MSE-Werte. Um Ausreißer auszuschließen haben wir die Testserie wiederholt und noch separate Vergleichstests mit diesen Konfigurationen gefahren. Das Ergebnis ist, dass das Netz mit 5 Neuronen in der ersten versteckten Schicht und 7 in der zweiten versteckten Schicht im Schnitt die beste Performance liefert.

Die gewählten Netzkonfigurationen für das Extrapolationsproblem sind also eine versteckte Schicht mit 18 Neuronen und zwei versteckte Schichten mit 5 Neuronen in der ersten und 7 Neuronen in der zweiten Schicht.

Layersize1	Layersize2	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
5	5	8,525988	0,004850	0,050641	0,105440	0,260856
5	7	6,619046	0,003765	0,045933	0,035489	0,154079
5	9	7,942929	0,004518	0,048670	0,096290	0,260316
5	11	7,375650	0,004195	0,047343	0,078040	0,227208
5	13	7,751558	0,004409	0,048404	0,113263	0,291606
5	15	7,305654	0,004156	0,047001	0,126927	0,294890
7	5	9,673023	0,005502	0,053473	0,115568	0,279447
7	7	6,373146	0,003625	0,045128	0,216090	0,424693
7	9	6,385845	0,003632	0,044259	0,094720	0,256252
7	11	7,533085	0,004285	0,048026	0,091378	0,256926
7	13	6,418571	0,003651	0,044159	0,067195	0,202725
7	15	6,341700	0,003607	0,043342	0,065726	0,198914
9	5	6,258981	0,003560	0,043648	0,099958	0,254376
9	7	6,434303	0,003660	0,044520	0,139131	0,311893
9	9	5,412941	0,003079	0,041924	0,082441	0,242645
9	11	5,812523	0,003306	0,043122	0,058728	0,211122
9	13	5,532164	0,003147	0,041335	0,094463	0,259546
9	15	6,374307	0,003626	0,043054	0,068452	0,211515
11	5	6,694321	0,003808	0,046321	0,114258	0,286796
11	7	6,498269	0,003696	0,044003	0,096334	0,249177
11	9	5,534404	0,003148	0,041584	0,041960	0,162965
11	11	5,458282	0,003105	0,041998	0,112509	0,287344
11	13	5,644151	0,003211	0,041932	0,087251	0,238848
11	15	5,910056	0,003362	0,042179	0,110596	0,284642
13	5	5,620534	0,003197	0,042625	0,086907	0,267929
13	7	5,811292	0,003306	0,042500	0,101308	0,251770
13	9	5,752503	0,003272	0,042973	0,072807	0,202441
13	11	5,457724	0,003105	0,041026	0,083350	0,247391
13	13	5,470209	0,003112	0,041624	0,069267	0,209173
13	15	4,867876	0,002769	0,039556	0,071997	0,218624
15	5	6,096508	0,003468	0,044853	0,039519	0,156519
15	7	6,535855	0,003718	0,045038	0,162738	0,352866
15	9	4,943708	0,002812	0,039928	0,119632	0,290528
15	11	4,942405	0,002811	0,039332	0,089867	0,258806
15	13	4,577772	0,002604	0,037870	0,111040	0,266851
15	15	5,137725	0,002922	0,040345	0,143086	0,335100

Tabelle 3.8.: Feinbestimmung der Neuronenanzahl (Extrapolation, vier Schichten)

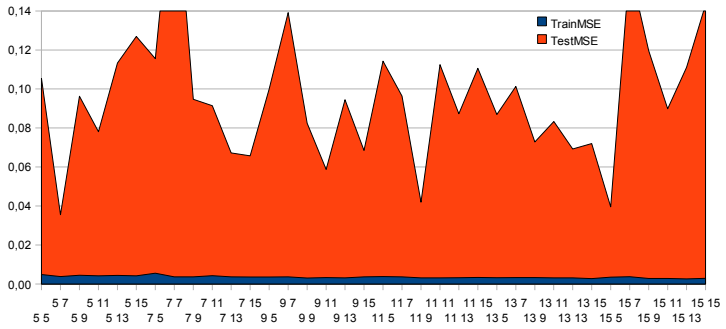

Abbildung 3.8.: Visualisierung der Ergebnisse aus Tabelle 3.8

3.2. Trainingsfunktionen

Nachdem wir jetzt vier Neuronale Netze ausgewählt haben, überprüfen wir nun, mit welcher der in Abschnitt 2.4.2 vorgestellten Trainingsfunktionen diese die besten Ergebnisse liefern.

3.2.1. Interpolation

TrainFcn	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
trainb	179,103708	0,070904	0,216623	0,071167	0,216730
trainbfg	40,923823	0,016201	0,101046	0,016197	0,100929
trainbr	6,278283	0,002485	0,035449	0,003855	0,039676
trainc	35,307291	0,032395	0,136425	0,032865	0,138002
traincgb	17,658615	0,006991	0,062316	0,007263	0,062578
traincgf	23,584730	0,009337	0,073743	0,009251	0,072916
traincgp	20,684008	0,008188	0,068841	0,008472	0,069670
traingd	218,187416	0,123952	0,287566	0,124140	0,287856
traingda	56,438740	0,022531	0,120770	0,022535	0,121053
traingdm	326,473280	0,129245	0,292481	0,129206	0,292591
traingdx	180,501568	0,071581	0,213680	0,071522	0,213541
trainlm	5,464181	0,002163	0,033635	0,004819	0,040372
trainoss	26,578916	0,010522	0,080037	0,010876	0,081035
trainr	41,421225	0,017056	0,096244	0,018272	0,099451
trainrp	9,567071	0,003787	0,044821	0,004173	0,046679
trains	276,188084	0,134605	0,316244	0,134809	0,316344
trainscg	20,050998	0,007938	0,067559	0,008031	0,067642

Tabelle 3.9.: Bestimmung der Trainingsfunktion (Interpolation, drei Schichten)

Aus Tabelle 3.9 ergibt sich, dass die Trainingsfunktion trainbr (Bayesian regularization backpropagation) für Interpolation mit unserem dreischichtigen Netz die besten Ergebnisse liefert.

Beim vierschichtigen Netz ist trainlm (Levenberg-Marquardt backpropagation) mit sehr großem Abstand die beste Trainingsfunktion (siehe Tabelle 3.10).

TrainFcn	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
trainb	163,437162	0,064702	0,199382	0,065520	0,200811
trainbfg	41,593467	0,016466	0,099519	0,017945	0,102296
trainbr	3,091910	0,001224	0,026190	0,002031	0,030750
trainc	39,896119	0,030090	0,136294	0,030327	0,137651
traincgb	18,071508	0,007154	0,062835	0,007260	0,062982
traincgf	36,265126	0,014357	0,095415	0,014333	0,095126
traincgp	21,085446	0,008347	0,068434	0,008494	0,068498
traingd	79,438366	0,031448	0,145169	0,031449	0,144716
traingda	53,507673	0,021815	0,118105	0,022686	0,120336
traingdm	180,078366	0,071290	0,217868	0,071377	0,217522
traingdx	42,394528	0,017122	0,105507	0,017287	0,105822
trainlm	2,604922	0,001031	0,024071	0,001788	0,029034
trainoss	56,124593	0,022219	0,115129	0,022303	0,114810
trainr	35,266310	0,014139	0,086332	0,015169	0,089511
trainrp	9,624668	0,003810	0,045051	0,004549	0,048264
trains	602,716410	0,147746	0,341348	0,147535	0,341089
trainscg	21,399098	0,008472	0,070294	0,008647	0,070829

Tabelle 3.10.: Bestimmung der Trainingsfunktion (Interpolation, vier Schichten)

3.2.2. Extrapolation

TrainFcn	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
trainb	132,681244	0,075473	0,220880	0,080277	0,237923
trainbfg	18,995816	0,010805	0,079476	0,089054	0,228300
trainbr	2,997745	0,001705	0,030553	0,074984	0,227421
trainc	18,381020	0,021710	0,121469	0,067270	0,196025
traincgb	11,699233	0,006655	0,062735	0,116619	0,287422
traincgf	11,767953	0,006694	0,062959	0,096462	0,248956
traincgp	11,796673	0,006710	0,061756	0,090059	0,264317
traingd	163,716105	0,145438	0,336335	0,207303	0,427652
traingda	32,762703	0,019062	0,105083	0,070919	0,224425
traingdm	125,126010	0,071175	0,207705	0,078331	0,235304
traingdx	24,595644	0,013991	0,092122	0,066160	0,215140
trainlm	3,242172	0,001844	0,031203	0,132039	0,325898
trainoss	15,470068	0,008800	0,074110	0,118251	0,270268
trainr	18,032747	0,011672	0,081319	0,104017	0,256798
trainrp	5,860549	0,003334	0,043014	0,087103	0,269138
trains	208,570278	0,113681	0,299862	0,106574	0,291309
trainscg	9,936992	0,005652	0,055287	0,066585	0,205924

Tabelle 3.11.: Bestimmung der Trainingsfunktion (Extrapolation, drei Schichten)

Tabelle 3.11 zeigt mehrere gute Trainingsfunktionen für Extrapolation mit dem dreischichtigen Netz. Die Funktion trainscg (Scaled conjugate gradient backpropagation) vereint einen sehr guten Trainings-MSE, Test-MSE und Test-MAE.

Unsere Testserie für das Extrapolationsproblem mit dem vierschichtigen Netz (Tabelle 3.12) weist die besten Ergebnisse mit traingda (Gradient descent with adaptive learning rate backpropagation) als Trainingsfunktion auf.

TrainFcn	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
trainb	187,165521	0,106465	0,262989	0,165007	0,336496
trainbfg	15,131266	0,008607	0,070061	0,080632	0,222884
trainbr	1,402046	0,000798	0,021394	0,057275	0,199360
trainc	21,252064	0,018918	0,113692	0,082575	0,215643
traincgb	10,614972	0,006038	0,058202	0,071307	0,214088
traincgf	17,668127	0,010050	0,075537	0,074470	0,215441
traincgp	12,811322	0,007287	0,062860	0,112421	0,288972
traingd	39,452447	0,024112	0,126946	0,033941	0,138306
traingda	52,141578	0,029660	0,138353	0,028472	0,130765
traingdm	48,151193	0,027390	0,132568	0,033783	0,139428
traingdx	33,290896	0,018937	0,103980	0,057108	0,196018
trainlm	1,575178	0,000896	0,022996	0,184890	0,360871
trainoss	261,009838	0,148470	0,265087	0,197620	0,377031
trainr	25,986992	0,016421	0,093922	0,077024	0,211485
trainrp	4,600399	0,002617	0,038377	0,052432	0,177501
trains	212,284901	0,105008	0,262700	0,161831	0,343351
trainscg	11,843951	0,006737	0,061247	0,121116	0,297443

Tabelle 3.12.: Bestimmung der Trainingsfunktion (Extrapolation, vier Schichten)

Für jede unserer vier Netzkonfigurationen liefert also eine andere Trainingsfunktion die besten Ergebnisse. Dies ist etwas ungewöhnlich, wir hätten zumindest innerhalb der selben Problemstellung erwartet, dass eine Trainingsfunktion konsistent bessere Ergebnisse liefert.

3.3. Performanzfunktionen

In diesem Abschnitt bestimmen wir die optimale Performanzfunktion für jede unserer Netzkonfigurationen.

3.3.1. Interpolation

Bei der Interpolation scheint die Wahl der Performanzfunktion allgemein keine allzu schwerwiegenden Auswirkungen zu haben. Am besten eignen sich MAE und SSE. Beim Netz mit einer versteckten Schicht (Tabelle 3.13) erzielt MAE als Performanzfunktionn leicht bessere Ergebnisse als SSE. Beim Netz mit zwei versteckten Schichten (Tabelle 3.14) liefert dagegen SSE geringfügig bessere Test-MSE-Werte.

Performance Function	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
mae	0,035855	0,002553	0,035855	0,003076	0,038470
sse	5,380572	0,002130	0,034323	0,003151	0,039042
msereg	90,444347	0,003597	0,043401	0,003863	0,044543
mseregec	55,681264	0,004255	0,047420	0,004575	0,048796

Tabelle 3.13.: Bestimmung der Performanzfunktion (Interpolation, drei Schichten)

Performance Function	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
mae	0,027338	0,001350	0,027338	0,002395	0,031992
sse	2,532397	0,001003	0,023601	0,002141	0,029904
msereg	36,836302	0,003406	0,042508	0,003547	0,042734
mseregec	30,498080	0,002990	0,039428	0,003372	0,041021

Tabelle 3.14.: Bestimmung der Performanzfunktion (Interpolation, vier Schichten)

3.3.2. Extrapolation

Bei der Extrapolation ergibt sich hinsichtlich der optimalen Performanzfunktion allgemein ein ähnliches Bild wie bei der Interpolation. Auch hier sind MAE und SSE die Favoriten, auch hier ist beim Netz mit einer versteckten Schicht (Tabelle 3.15) MAE besser, beim Netz mit zwei versteckten Schichten (Tabelle 3.16) hat SSE den Vorteil.

Bemerkenswerter Unterschied zwischen Inter- und Extrapolation ist, dass die Ergebnisse bei Extrapolation viel eindeutiger sind. War der Unterschied zwischen MAE und SSE bei Interpolation sehr gering, so liefert besteht bei Extrapolation ein deutlicher Abstand zwischen der für die jeweilige Netzkonfiguration besten Performanzfunktion und den anderen.

Performance Function	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
mae	0,032641	0,001949	0,032641	0,060822	0,204418
sse	3,287005	0,001870	0,031876	0,134118	0,336316
msereg	112,980559	0,004273	0,047453	0,105378	0,263489
mseregec	46,707655	0,003245	0,043581	0,121873	0,297094

Tabelle 3.15.: Bestimmung der Performanzfunktion (Extrapolation, drei Schichten)

Wir wählen also als Performanzfunktionen sowohl bei Inter- als auch Extrapolation für die Netze mit einer versteckten Schicht MAE, für die Netze mit zwei versteckten Schichten SSE.

Performance Function	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
mae	0,023552	0,000982	0,023552	0,083436	0,240595
sse	1,386801	0,000789	0,021388	0,038024	0,157788
msereg	36,939771	0,003327	0,042865	0,070088	0,222811
mseregec	32,751688	0,002010	0,033477	0,086824	0,266202

Tabelle 3.16.: Bestimmung der Performanzfunktion (Extrapolation, vier Schichten)

3.4. Transferfunktionen

In diesem Abschnitt bestimmen wir die optimalen Transferfunktionen für jede unserer Netzkonfigurationen.

3.4.1. Interpolation

Beim das Interpolationsproblem liefert für das dreischichtige Netz (Tabelle 3.17) eine Kombination aus logsig als Transferfunktion für die versteckte Schicht und purelin als Transferfunktion für die Ausgabeschicht die besten Ergebnisse.

Transferfunction1	Transferfunction2	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
logsig	logsig	5,453046	0,002159	0,033469	0,002992	0,038063
logsig	tansig	7,566077	0,002995	0,038758	0,003739	0,042370
logsig	purelin	5,756178	0,002279	0,033924	0,002732	0,036934
tansig	logsig	6,429174	0,002545	0,035888	0,003095	0,038665
tansig	tansig	5,448250	0,002157	0,031892	0,003166	0,036976
tansig	purelin	6,016201	0,002382	0,034519	0,002892	0,038003
purelin	logsig	19,500358	0,007720	0,063838	0,007781	0,064221
purelin	tansig	21,003944	0,008315	0,067739	0,008305	0,067810
purelin	purelin	19,412147	0,007685	0,063632	0,007717	0,063869

Tabelle 3.17.: Bestimmung der Transferfunktionen (Interpolation, drei Schichten)

Für das vierschichtige Netz (Tabelle 3.18) funktionieren die Transferfunktionen tansig für die erste versteckte Schicht, logsig für die zweite versteckte Schicht und purelin für die Ausgangsschicht mit Abstand am besten.

Es fällt auf, dass hier Mischungen aus verschiedenen Transferfunktionen für jede Schicht besser abschneiden als die Verwendung einer einzigen Transferfunktion für das gesamte Netz.

Transferfunction1	Transferfunction2	Transferfunction3	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
logsig	logsig	logsig	2,782609	0,001102	0,024834	0,003165	0,033704
logsig	logsig	tansig	3,495673	0,001384	0,028107	0,013658	0,045004
logsig	logsig	purelin	2,681353	0,001062	0,024769	0,585237	0,059701
logsig	tansig	logsig	3,232146	0,001280	0,026932	0,002082	0,031484
logsig	tansig	tansig	3,397234	0,001345	0,027580	0,002426	0,034680
logsig	tansig	purelin	3,126738	0,001238	0,026403	0,002848	0,033508
logsig	purelin	logsig	7,560357	0,002993	0,038990	0,003399	0,040745
logsig	purelin	tansig	6,917359	0,002738	0,037145	0,003130	0,039255
logsig	purelin	purelin	6,894602	0,002729	0,036869	0,003111	0,039243
tansig	logsig	logsig	3,810064	0,001508	0,028419	0,002626	0,034839
tansig	logsig	tansig	3,141809	0,001244	0,026476	0,004427	0,035243
tansig	logsig	purelin	2,654372	0,001051	0,024331	0,001642	0,029101
tansig	tansig	logsig	3,189336	0,001263	0,026742	0,002239	0,032782
tansig	tansig	tansig	3,116041	0,001234	0,026251	0,007500	0,036755
tansig	tansig	purelin	3,926497	0,001554	0,028649	0,003707	0,037268
tansig	purelin	logsig	7,043442	0,002788	0,038111	0,005233	0,044459
tansig	purelin	tansig	7,607969	0,003012	0,039171	0,003507	0,042020
tansig	purelin	purelin	5,546514	0,002196	0,035694	0,003669	0,039612
purelin	logsig	logsig	13,400490	0,005305	0,052054	0,005665	0,053402
purelin	logsig	tansig	17,149196	0,006789	0,059265	0,007513	0,061211
purelin	logsig	purelin	17,517395	0,006935	0,061660	0,007738	0,063740
purelin	tansig	logsig	21,973768	0,008699	0,069533	0,009026	0,070726
purelin	tansig	tansig	21,787648	0,008625	0,070312	0,009532	0,073716
purelin	tansig	purelin	18,493749	0,007321	0,064438	0,007560	0,065303
purelin	purelin	logsig	129,021001	0,051077	0,150292	0,051245	0,150491
purelin	purelin	tansig	645,342998	0,255480	0,448330	0,261097	0,450947
purelin	purelin	purelin	19,412147	0,007685	0,063632	0,007717	0,063869

Tabelle 3.18.: Bestimmung der Transferfunktionen (Interpolation, vier Schichten)

3.4.2. Extrapolation

Beim Extrapolationsproblem gilt für das dreischichtige Netz (Tabelle 3.19) wie bei der Interpolation, dass die Kombination aus den Transferfunktionen logsig für die versteckte Schicht und purelin für die Ausgangsschicht die besten Ergebnisse liefert.

Transferfunction1	Transferfunction2	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
logsig	logsig	3,124770	0,001777	0,031637	0,140573	0,325004
logsig	tansig	3,899707	0,002218	0,034740	0,164245	0,331722
logsig	purelin	3,127868	0,001779	0,031149	0,042746	0,167728
tansig	logsig	3,413546	0,001942	0,033008	0,200619	0,405183
tansig	tansig	3,451014	0,001963	0,033222	0,069752	0,232909
tansig	purelin	2,676237	0,001522	0,029524	0,068704	0,235621
purelin	logsig	11,706476	0,006659	0,059372	0,124338	0,293720
purelin	tansig	13,605645	0,007739	0,066304	0,145227	0,315197
purelin	purelin	12,090730	0,006878	0,060745	0,173686	0,347103

Tabelle 3.19.: Bestimmung der Transferfunktionen (Extrapolation, drei Schichten)

Auch beim vierschichtigen Netz (Tabelle 3.20) ist die optimale Transferfunktionenkombination für die Extrapolation identisch mit der für die Interpolation – tansig für die erste versteckte Schicht, logsig für die zweite versteckte Schicht und purelin für die Ausgabeschicht.

Allgemein lässt sich also beobachten, dass die optimalen Transferfunktionskombinationen für Interpolation und Extrapolation identisch sind, wobei alle optimalen Kombina-

Transferfunction1	Transferfunction2	Transferfunction3	Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
logsig	logsig	logsig	1,747099	0,000994	0,024132	0,196532	0,378502
logsig	logsig	tansig	1,966845	0,001119	0,024924	0,144431	0,332656
logsig	logsig	purelin	1,508642	0,000858	0,022036	0,332278	0,495410
logsig	tansig	logsig	1,266488	0,000720	0,020231	0,106862	0,255413
logsig	tansig	tansig	1,741748	0,000991	0,023816	0,677354	0,634858
logsig	tansig	purelin	1,879583	0,001069	0,024683	0,220003	0,339860
logsig	purelin	logsig	4,077059	0,002319	0,036404	0,104705	0,282836
logsig	purelin	tansig	7,912068	0,004501	0,050002	0,066118	0,193316
logsig	purelin	purelin	3,534469	0,002011	0,033383	0,116713	0,302017
tansig	logsig	logsig	1,645267	0,000936	0,023387	0,147539	0,336919
tansig	logsig	tansig	1,877269	0,001068	0,024629	0,114514	0,268923
tansig	logsig	purelin	1,400169	0,000796	0,021368	0,022018	0,113434
tansig	tansig	logsig	2,038587	0,001160	0,025290	0,099882	0,277334
tansig	tansig	tansig	1,543198	0,000878	0,022531	0,053761	0,179292
tansig	tansig	purelin	1,568946	0,000892	0,022511	0,112285	0,272750
tansig	purelin	logsig	3,978884	0,002263	0,036256	0,098519	0,272071
tansig	purelin	tansig	4,042778	0,002300	0,036677	0,078936	0,241904
tansig	purelin	purelin	3,367291	0,001915	0,032792	0,106427	0,266347
purelin	logsig	logsig	32,072691	0,018244	0,086912	0,126807	0,300697
purelin	logsig	tansig	10,365111	0,005896	0,056568	0,125458	0,286201
purelin	logsig	purelin	14,276092	0,008121	0,067032	0,128507	0,289475
purelin	tansig	logsig	8,491831	0,004830	0,050417	0,123223	0,302316
purelin	tansig	tansig	14,086474	0,008013	0,066853	0,132480	0,302142
purelin	tansig	purelin	8,003080	0,004552	0,049902	0,153397	0,327181
purelin	purelin	logsig	103,283011	0,058750	0,155437	0,139277	0,306696
purelin	purelin	tansig	235,709909	0,134078	0,270389	0,244575	0,457435
purelin	purelin	purelin	12,090730	0,006878	0,060745	0,173686	0,347103

Tabelle 3.20.: Bestimmung der Transferfunktionen (Extrapolation, vier Schichten)

tionen aus verschiedenen Transferfunktionen bestehen. Für die Ausgabeschicht liefert immer die lineare Transferfunktion die besten Ergebnisse.

3.5. Die optimalen Netzkonfigurationen

Fassen wir also die erzielten Ergebnisse zusammen und wählen die optimalen Netzkonfigurationen für jeweils Interpolation und Extrapolation aus. In beiden Fällen liefern die vierschichtigen Netze bessere Ergebnisse, wir betrachten also nur noch diese.

Das optimale Netz für das Interpolationsproblem hat jeweils 13 Neuronen in beiden versteckten Schichten und benutzt trainlm als Trainingsfunktion.

Das optimale Netz für das Extrapolationsproblem hat fünf Neuronen in der ersten und sieben Neuronen in der zweiten versteckten Schicht und benutzt traingda als Trainingsfunktion.

Beiden Netzen gemeinsam ist die Nutzung von SSE als Performanzfunktion und den Transferfunktionen tansig für die erste versteckte Schicht, logsig für die zweite versteckte Schicht und purelin für die Ausgabeschicht.

Wir haben beide Netze in ihrer ermittelten optimalen Konfiguration noch einer finalen Testreihe mit mehreren Durchläufen unterworfen. Die Ergebnisse sind Tabelle 3.21 und 3.22 zu entnehmen.

Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
3,59	0,001421	0,027616	0,001885	0,030604
4,08	0,001614	0,027704	0,001940	0,030097
3,21	0,001271	0,026751	0,001873	0,030860
3,26	0,001292	0,026208	0,001716	0,029351
2,73	0,001082	0,024645	0,001678	0,028363

Tabelle 3.21.: Finale Testserie der Interpolation

Performance	TrainMSE	TrainMAE	TestMSE	TestMAE
18,41	0,002097	0,032118	0,096968	0,249792
26,80	0,003054	0,040873	0,077298	0,211825
22,36	0,002547	0,036656	0,080152	0,239557
22,37	0,002548	0,036083	0,035950	0,152395
23,81	0,002713	0,037689	0,015410	0,100027

Tabelle 3.22.: Finale Testserie der Extrapolation

4. Fazit

Im Rahmen dieser Ausarbeitung konnten wir einige allgemeine Erkenntnisse gewinnen beziehungsweise bestätigen. So zeigten die Testserien in Abschnitt 3.1 deutlich, dass Generalisierungsfähigkeit eines neuronalen Netzes mit steigender Neuronenanzahl stetig abnimmt, während die Spezialisierung auf die Trainingsdaten zunimmt. Wir haben auch festgestellt, dass zwei versteckte Schichten zu besseren Ergebnissen führen als nur eine versteckte Schicht. Wir vermuten, dass mehr versteckte Schichten keine signifikante Verbesserung bewirken, ein Nachweis hätte jedoch die Anzahl der zu testenden Netzkonfigurationen exponentiell ansteigen lassen und den Rahmen dieser Arbeit gesprengt. Wie erwartet sind die Abweichungen bei Extrapolation größer als bei Interpolation. Aufgefallen ist, dass für gute Extrapolationsergebnisse eine niedrigere Neuronenanzahl nötig war (5 und 7 gegenüber 13 und 13 bei der Interpolation), also ein gutes Generalisierungsverhalten für Extrapolation extrem wichtig ist.

Alles in allem sind unsere Ergebnisse sehr zufriedenstellend. Aus den Ergebnissen der finalen Testserien (siehe Abschnitt 3.5) geht hervor, dass unsere durchschnittlichen Test-MSE-Werte bei Interpolation unter 0,2% liegen, bei Extrapolation zwischen 1% und 10%. Damit haben wir die in Abschnitt 1 gesteckten Ziele erreicht.

A. Matlab-Programm

A.1. runall.m

```
RunTest('Inter',2,'layersizes');
RunTest('Inter',2,'layersizes2');
RunTest('Extra',2,'layersizes');
RunTest('Extra',2,'layersizes2');
RunTest('Inter',3,'layersizes3');
RunTest('Inter',3,'layersizes4');
RunTest('Extra',3,'layersizes3');
RunTest('Extra',3,'layersizes4');

RunTest('Inter',2,'performFcns');
RunTest('Inter',3,'performFcns2');

RunTest('Inter',2,'trainFcns');
RunTest('Inter',3,'trainFcns2');
RunTest('Extra',2,'trainFcns');
RunTest('Extra',3,'trainFcns2');

RunTest('Extra',2,'performFcns');
RunTest('Extra',3,'performFcns2');

RunTest('Inter',2,'transferfunctions');
RunTest('Inter',3,'transferfunctions2');
RunTest('Extra',2,'transferfunctions');
RunTest('Extra',3,'transferfunctions2');
```

A.2. RunTest.m

```
function [ ] = RunTest( Lines, Layers, TestCase )

Data = LoadData(Lines);
TestCases = GetTestCases(TestCase, Layers);

fid=fopen(['stat ',Lines,',',num2str(Layers),',',TestCase,'.csv'],'w');
if Layers == 2
    fprintf(fid, '   TestCase Of Layersize Epochen Transferfunction1 Transferfunction2
    TrainFcn Performance TrainMSE TrainMSE1 TrainMSE2 TrainMSE3 TrainMAE TrainMAE1 TrainMAE2
    TrainMAE3 TestMSE TestMSE1 TestMSE2 TestMSE3 TestMAE TestMAE1 TestMAE2 TestMAE3 TestMBE
```

```
    TestMBE1 TestMBE2 TestMBE3\n');
elseif Layers == 3
    fprintf(fid, '   TestCase Of Layersize1 Layersize2 Epochen Transferfunction1
    Transferfunction2 Transferfunction3 TrainFcn Performance TrainMSE TrainMSE1 TrainMSE2
    TrainMSE3 TrainMAE TrainMAE1 TrainMAE2 TrainMAE3 TestMSE TestMSE1 TestMSE2 TestMSE3
    TestMAE TestMAE1 TestMAE2 TestMAE3 TestMBE TestMBE1 TestMBE2 TestMBE3\n');
end

i = 0;
for TestCase = TestCases
    minTestMSE = 10000000;
    minTest = 5;
    for t = 1:1
        fprintf('Test %4d\n', t);
        TestCaseT(t) = TestNN(Data, Layers, TestCase);
        if (minTestMSE > sum(TestCaseT(t).TestMSE)/3)
            minTestMSE = sum(TestCaseT(t).TestMSE)/3;
            minTest = t;
        end
    end
    TestCase = TestCaseT(minTest);
    i = i + 1;

    if Layers == 2
        fprintf(fid,'%4d %4d %4d %4d %8s %8s %8s %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f
        %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f
        %4.20f %4.20f %4.20f\n', ...
            i, size(TestCases,2), TestCase.layersize1, TestCase.epochs,
            TestCase.transferfunction{:}, TestCase.trainFcn, TestCase.performance,
            sum(TestCase.TrainMSE)/3, TestCase.TrainMSE, sum(TestCase.TrainMAE)/3,
            TestCase.TrainMAE, sum(TestCase.TestMSE)/3, TestCase.TestMSE,
            sum(TestCase.TestMAE)/3, TestCase.TestMAE, sum(abs(TestCase.TestMBE))/3,
            TestCase.TestMBE );
        fprintf('TestCase %4d of %4d: Layersize: %4d Epochen: %4d Transferfunction: { %8s %8s
        } TrainFcn: %8s Performance: %4.20f TrainMSE: %4.20f (%4.20f %4.20f %4.20f) TrainMAE:
        %4.20f (%4.20f %4.20f %4.20f) TestMSE: %4.20f (%4.20f %4.20f %4.20f) TestMAE: %4.20f
        (%4.20f %4.20f %4.20f) TestMBE: %4.20f (%4.20f %4.20f %4.20f)\n', ...
            i, size(TestCases,2), TestCase.layersize1, TestCase.epochs,
            TestCase.transferfunction{:}, TestCase.trainFcn, TestCase.performance,
            sum(TestCase.TrainMSE)/3, TestCase.TrainMSE, sum(TestCase.TrainMAE)/3,
            TestCase.TrainMAE, sum(TestCase.TestMSE)/3, TestCase.TestMSE,
            sum(TestCase.TestMAE)/3, TestCase.TestMAE, sum(abs(TestCase.TestMBE))/3,
            TestCase.TestMBE);
    elseif Layers == 3
        fprintf(fid,'%4d %4d %4d %4d %4d %8s %8s %8s %8s %4.20f %4.20f %4.20f %4.20f %4.20f
        %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f %4.20f
        %4.20f %4.20f %4.20f %4.20f\n', ...
            i, size(TestCases,2), TestCase.layersize1, TestCase.layersize2, TestCase.epochs,
            TestCase.transferfunction{:}, TestCase.trainFcn, TestCase.performance,
            sum(TestCase.TrainMSE)/3, TestCase.TrainMSE, sum(TestCase.TrainMAE)/3,
            TestCase.TrainMAE, sum(TestCase.TestMSE)/3, TestCase.TestMSE,
            sum(TestCase.TestMAE)/3, TestCase.TestMAE, sum(abs(TestCase.TestMBE))/3,
            TestCase.TestMBE );
```

```
        fprintf('TestCase %4d of %4d: Layersize: %4d %4d Epochen: %4d Transferfunction: { %8s
        %8s %8s } TrainFcn: %8s Performance: %4.20f TrainMSE: %4.20f (%4.20f %4.20f %4.20f)
        TrainMAE: %4.20f (%4.20f %4.20f %4.20f) TestMSE: %4.20f (%4.20f %4.20f %4.20f)
        TestMAE: %4.20f (%4.20f %4.20f %4.20f) TestMBE: %4.20f (%4.20f %4.20f %4.20f)\n', ...
            i, size(TestCases,2), TestCase.layersize1, TestCase.layersize2, TestCase.epochs,
            TestCase.transferfunction{:}, TestCase.trainFcn, TestCase.performance,
            sum(TestCase.TrainMSE)/3, TestCase.TrainMSE, sum(TestCase.TrainMAE)/3,
            TestCase.TrainMAE, sum(TestCase.TestMSE)/3, TestCase.TestMSE,
            sum(TestCase.TestMAE)/3, TestCase.TestMAE, sum(abs(TestCase.TestMBE))/3,
            TestCase.TestMBE );
    end

end

fclose(fid);
```

A.3. LoadData.m

```
function [ Data ] = LoadData( Lines )

load building.dat % building.dat einlesen
buildingsize = size(building);
buildinglines = buildingsize(1);
buildingminmax = minmax(building');

convertcase = 1;

if convertcase == 1 % MONTH DAY WEEKDAY HOUR TEMP  HUMID  SOLAR  WIND
    P = building(:,1:8)'; %  die ersten 8 Spalten aus der Datei builing.dat werden in die
    Matrix P kopiert.
    P(4,:) = P(4,:)./2400; % die Stundenspalte wird durch 2400 geteilt.
    P(3,:) = weekday(datenum([P(3,:);P(1,:);P(2,:)]'))'; % die Jahresspalte wird durch
    Wochentage ersetzt
elseif convertcase == 2
    P = building(:,4:8)'; %  die ersten 8 Spalten aus der Datei builing.dat werden in die
    Matrix P kopiert.
    P(1,:) =
datenum([P(3,:)',P(1,:)',P(2,:)',P(4,:)'./100,zeros(size(P,2),1),zeros(size(P,2),1)])';
end

T = building(:,9:11)'; %  die hinteren 3 Spalten aus der Datei builing.dat werden in die
Matrix T kopiert.
% normierung der Ausgabewerte auf den Bereich zwischen 0 und 1
T(1,:) = T(1,:)/ buildingminmax(09,2);
T(2,:) = T(2,:)/ buildingminmax(10,2);
T(3,:) = T(3,:)/ buildingminmax(11,2);

stat = 0;
a = 0;

if strcmp (Lines, 'Test') == 1
    TrainLines = 1:50:buildinglines;
```

```
    TestLines = 3:50:buildinglines;
elseif strcmp (Lines, 'Inter') == 1
    TrainLines = 1:5:buildinglines;
    TestLines = 3:5:buildinglines;
elseif strcmp (Lines, 'Extra') == 1
    TrainLines = 1:5:2926;
    TestLines = 2927:5:buildinglines;
elseif strcmp (Lines, 'ExtraAll') == 1
    TrainLines = 1:2926;
    TestLines = 2927:buildinglines;
end

Data.TrainP = P(:,TrainLines);
Data.TrainT = T(:,TrainLines);
Data.TestP = P(:,TestLines);
Data.TestT = T(:,TestLines);
Data.T = T;
Data.P = P;
```

A.4. GetTestCases.m

```
function [ TestCases ] = GetTestCases( test, layers )

stdepochs = 1000;

if strcmp (test, 'InterFinal') == 1
    for i = 1:20
        TestCases(i).trainFcn = 'trainbr';
        TestCases(i).performFcn = 'sse';
        TestCases(i).epochs = 100;
        TestCases(i).layersize1 = 13;
        TestCases(i).layersize2 = 13;
        TestCases(i).transferfunction = {'tansig','logsig','purelin'};
        TestCases(i).performance = 1000;
        TestCases(i).TrainMSE = 1000;
        TestCases(i).TrainMAE = 1000;
        TestCases(i).TestMSE = 1000;
        TestCases(i).TestMAE = 1000;
    end
elseif strcmp (test, 'ExtraFinal') == 1
    for i = 1:20
        TestCases(i).trainFcn = 'traingda';
        TestCases(i).performFcn = 'sse';
        TestCases(i).epochs = 100;
        TestCases(i).layersize1 = 5;
        TestCases(i).layersize2 = 7;
        TestCases(i).transferfunction = {'tansig','logsig','purelin'};
        TestCases(i).performance = 1000;
        TestCases(i).TrainMSE = 1000;
        TestCases(i).TrainMAE = 1000;
        TestCases(i).TestMSE = 1000;
        TestCases(i).TestMAE = 1000;
```

```
    end
else
    if strcmp (test, 'trainFcns') == 1
        Test.trainFcns =
        {'trainb','trainbfg','trainbr','trainc','traincgb','traincgf','traincgp','traingd',
        'traingda','traingdm','traingdx','trainlm','trainoss','trainr','trainrp','trains',
        'trainscg'};
        Test.performFcns = {'sse'};
        Test.epochs = [stdepochs];
        Test.layersizes = [18];
        Test.transferfunctions = {'logsig'};
    elseif strcmp (test, 'trainFcns2') == 1
        Test.trainFcns = {'trainb','trainbfg','trainbr','trainc','traincgb','traincgf',
        'traincgp','traingd','traingda','traingdm','traingdx','trainlm','trainoss','trainr',
        'trainrp','trains','trainscg'};
        Test.performFcns = {'sse'};
        Test.epochs = [stdepochs];
        Test.layersizes = [13];
        Test.transferfunctions = {'logsig'};
    elseif strcmp (test, 'performFcns') == 1
        Test.trainFcns = {'trainbr'};
        Test.performFcns = {'mae','sse','msereg','mseregec'};
        Test.epochs = [stdepochs];
        Test.layersizes = [18];
        Test.transferfunctions = {'logsig'};
    elseif strcmp (test, 'performFcns2') == 1
        Test.trainFcns = {'trainbr'};
        Test.performFcns = {'mae','sse','msereg','mseregec'};
        Test.epochs = [stdepochs];
        Test.layersizes = [13];
        Test.transferfunctions = {'logsig'};
    elseif strcmp (test, 'transferfunctions') == 1
        Test.trainFcns = {'trainbr'};
        Test.performFcns = {'sse'};
        Test.epochs = [stdepochs];
        Test.layersizes = [18];
        Test.transferfunctions = {'logsig','tansig','purelin'};
    elseif strcmp (test, 'transferfunctions2') == 1
        Test.trainFcns = {'trainbr'};
        Test.performFcns = {'sse'};
        Test.epochs = [stdepochs];
        Test.layersizes = [13];
        Test.transferfunctions = {'logsig','tansig','purelin'};
    elseif strcmp (test, 'layersizes') == 1
        Test.trainFcns = {'trainrp'};
        Test.performFcns = {'sse'};
        Test.epochs = [stdepochs];
        Test.layersizes = 5:5:100;
        Test.transferfunctions = {'logsig'};
    elseif strcmp (test, 'layersizes2') == 1
        Test.trainFcns = {'trainrp'};
        Test.performFcns = {'sse'};
        Test.epochs = [stdepochs];
```

```
    Test.layersizes = 15:1:25;
    Test.transferfunctions = {'logsig'};
elseif strcmp (test, 'layersizes3') == 1
    Test.trainFcns = {'trainrp'};
    Test.performFcns = {'sse'};
    Test.epochs = [stdepochs];
    Test.layersizes = 5:5:20;
    Test.transferfunctions = {'logsig'};
elseif strcmp (test, 'layersizes4') == 1
    Test.trainFcns = {'trainrp'};
    Test.performFcns = {'sse'};
    Test.epochs = [stdepochs];
    Test.layersizes = 5:2:15;
    Test.transferfunctions = {'logsig'};
else
    Test.trainFcns = {'trainrp'};
    Test.performFcns = {'sse'};
    Test.epochs = [1000];
    Test.layersizes = [18];
    Test.transferfunctions = {'logsig'};
end

n = 0;
if layers == 2
    for trainFcn = Test.trainFcns
        for performFcn = Test.performFcns
            for epochs = Test.epochs
                for layersize = Test.layersizes
                    for transferfunction1 = Test.transferfunctions
                        for transferfunction2 = Test.transferfunctions
                            n = n + 1;
                            TestCases(n).trainFcn = trainFcn{1};
                            TestCases(n).performFcn = performFcn{1};
                            TestCases(n).epochs = epochs;
                            TestCases(n).layersize1 = layersize;
                            TestCases(n).transferfunction =
                            {transferfunction1{1},transferfunction2{1}};
                            TestCases(n).performance = 1000;
                            TestCases(n).TrainMSE = 1000;
                            TestCases(n).TrainMAE = 1000;
                            TestCases(n).TestMSE = 1000;
                            TestCases(n).TestMAE = 1000;
                        end
                    end
                end
            end
        end
    end
elseif layers == 3
    for trainFcn = Test.trainFcns
        for performFcn = Test.performFcns
            for epochs = Test.epochs
                for layersize1 = Test.layersizes
```

```
                    for layersize2 = Test.layersizes
                        for transferfunction1 = Test.transferfunctions
                            for transferfunction2 = Test.transferfunctions
                                for transferfunction3 = Test.transferfunctions
                                    n = n + 1;
                                    TestCases(n).trainFcn = trainFcn{1};
                                    TestCases(n).performFcn = performFcn{1};
                                    TestCases(n).epochs = epochs;
                                    TestCases(n).layersize1 = layersize1;
                                    TestCases(n).layersize2 = layersize2;
                                    TestCases(n).transferfunction =
                                    {transferfunction1{1},
                                    transferfunction2{1},
                                    transferfunction3{1}};
                                    TestCases(n).performance = 1000;
                                    TestCases(n).TrainMSE = 1000;
                                    TestCases(n).TrainMAE = 1000;
                                    TestCases(n).TestMSE = 1000;
                                    TestCases(n).TestMAE = 1000;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```

A.5. TestNN.m

```
function [ TestCase ] = TestNN(Data, Layers, TestCase)

if Layers == 2
    net = newff(minmax(Data.P),[TestCase.layersize1,3],TestCase.transferfunction,'traingd');
elseif Layers == 3
    net = newff(minmax(Data.P),[TestCase.layersize1,TestCase.layersize2,3],
    TestCase.transferfunction,'traingd');
end
net = init(net);

% Training
net.trainFcn = TestCase.trainFcn;
net.performFcn = TestCase.performFcn;   % Performance-Fkt
net.trainParam.show = NaN;  % keine Display-Rate
net.trainParam.epochs = TestCase.epochs; % Max Anz. Epochen
net.trainParam.goal = 0; % Fehlerziel: SSE = 0

[net,tr,ac] = train(net,Data.TrainP,Data.TrainT);
TrainO = sim(net,Data.TrainP);
TestO = sim(net,Data.TestP);
```

```
TestCase.TrainMAE = sum(abs((Data.TrainT - TrainO)'))/size(TrainO,2));
TestCase.TrainMSE = sum(abs((Data.TrainT - TrainO)'
.*(Data.TrainT - TrainO)'))/size(TrainO,2));
TestCase.TestMAE = sum(abs((Data.TestT - TestO)'))/size(TestO,2));
TestCase.TestMSE = sum(abs((Data.TestT - TestO)'.*(Data.TestT - TestO)'))/size(TestO,2));
TestCase.TestMBE = sum((Data.TestT - TestO)')./(sum(Data.TestT'));

TestCase.performance = min(tr.perf');
```

Literaturverzeichnis

[Bittel 2007] Bittel, Prof. Dr. O.: *Neuronale Netze und Fuzzy-Logik*. Version: April 2007. `http://www-home.fh-konstanz.de/~bittel/nnfl/nnfl.htm`, Abruf: 19. Juni 2008

[Demuth u. a. 2007] Demuth, Howard ; Beale, Mark ; Hagan, Martin: *Neural Network Toolbox™ User's Guide*. Natick, MA: The MathWorks, Inc., September 2007

[Kreider u. Haberl 1994] Kreider, J.F. ; Haberl, J.S.: Predicting hourly building energy use: the great energy predictor shootout – overview and discussion of results. In: *ASHRAE Transactions* 100 (1994), Nr. 2, S. 1104 – 1118

[Lohninger 2007] Lohninger, Hans: *Grundlagen der Statistik*. Version: Oktober 2007. `http://www.statistics4u.info/fundstat_germ/`, Abruf: 18. Juni 2008