Fachhochschule Aachen
Campus Jülich
Fachbereich 09 – Medizintechnik und Technomathematik
Studiengang Scientific Programming

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Untersuchung räumlicher Muster in meteorologischen Modellen für die Ensemble-Kalibrierung mittels tiefer neuronaler Netze

Bachelorarbeit von Daniel Todt
Matrikelnummer 3110512
Jülich, 3. September 2019

Erklärung

Diese Arbeit ist von mir selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Ort, Datum Daniel Todt

Diese Arbeit wurde betreut von:
1. Prüfer: Prof. Dr. rer. nat. Matthias Grajewski
2. Prüfer: Dr. phil. nat. Kai Krajsek

Inhaltsverzeichnis

Zusammenfassung

In der Energiewirtschaft werden genaue und verlässliche Kurzfristvorhersagen meteorologischer Größen für Sonnen- und Windenergieanlagen benötigt. Am Institut für Energie und Klimaforschung – Troposphäre (IEK-8) am Forschungszentrum Jülich werden auf Basis des Weather Research and Forecasting Model (WRF) Methoden zur quantitativen Vorhersage im Bereich der Energiemeteorologie entwickelt. Um Unsicherheiten in den Vorhersagen quantitativ ausdrücken zu können, wird ein Ensemble eingesetzt, das mehrere Modelle (sogenannte Member) mit unterschiedlichen Anfangsbedingungen und Modellparametern enthält. Das Ensemble muss kalibriert werden, um verlässliche Prognosen zu erhalten. Dazu werden die Vorhersagen statistisch nachbearbeitet, sodass systematische Fehler korrigiert werden.

In dieser Bachelorarbeit wird ein Ensemble mit 30 Membern mithilfe verschiedener Verfahren bezüglich der Vorhersagen von Temperatur in 2 m Höhe kalibriert. Zur Durchführung einer Ensemble-Kalibrierung werden in der Praxis Messungen herangezogen, die in der Regel nur mit den Modellwerten an den Messstationen verglichen werden. Um den Einfluss räumlicher Muster in den Modelldaten zu untersuchen, werden in dieser Arbeit Umgebungen um die einzelnen Messstationen herum aus dem Modell entnommen und für die Kalibrierung eingesetzt. Da sich künstliche neuronale Netze bereits in vielen Anwendungsgebieten zur Erkennung von Merkmalen in mehrdimensionalen Daten bewährt haben, liegt der Fokus in dieser Arbeit auf dem Einsatz von tiefen neuronalen Netzen. Mithilfe verschiedener Netzstrukturen wird untersucht, ob die Einbeziehung räumlicher Umgebungen aus dem Modell zu einer besseren Kalibrierung des Ensembles führt.

Zuerst werden die Grundlagen zur Ensemble-Kalibrierung sowie die vorliegenden Daten und Vorbereitungsschritte erklärt. Anschließend werden verschiedene Verfahren zur Ensemble-Kalibrierung, insbesondere künstliche neuronale Netze, vorgestellt. Außerdem wird die Implementierung der genutzten Verfahren beschrieben und das durchgeführte Experiment erläutert. Zum Schluss werden die Ergebnisse diskutiert und ein Ausblick auf weitere Experimente gegeben.

1. Einleitung

Das Institut für Energie- und Klimaforschung – Troposphäre (IEK-8, [Jül18]) am Forschungszentrum Jülich beschäftigt sich mit den physikalischen und chemischen Vorgängen in der Troposphäre, die die Zusammensetzung der Atmosphäre beeinflussen. Die Troposphäre ist die unterste Schicht der Erdatmosphäre und steht in direkter Verbindung zum menschlichen Lebensraum. Zur Untersuchung und Modellierung der Troposphäre und der in ihr ablaufenden chemischen Transportprozesse werden meteorologische Vorhersagen benötigt. Dazu wird das Weather Research and Forecasting Model (WRF, [UCA18]) eingesetzt.

Genaue und verlässliche Vorhersagen meteorologischer Größen gewinnen außerdem zunehmend an sozialer und wirtschaftlicher Bedeutung. Für kurzzeitige Prognosen von Solarstrom, Niederschlag und auch Verunreinigungen in der Luft werden genaue Vorhersagen zur Wolkenbedeckung und zur Sonnenstrahlung benötigt. Unsicherheiten in numerischen Wettermodellen, die unter anderem durch ungenaue Kenntnis des Anfangszustands der Atmosphäre entstehen, verhindern eine verlässliche Beschreibung des zukünftigen Wetterverlaufs.

1.1. Ensembles meteorologischer Modelle

Um die Unsicherheiten in den Vorhersagen beschreiben zu können, werden Ensembles eingesetzt. Darin werden mehrere Modelle (Member) zusammengefasst, die sich in ihren Startbedingungen und Modellparametern unterscheiden. Dadurch werden probabilistische Vorhersagen, die den zukünftigen Verlauf des Wetters über Wahrscheinlichkeitsaussagen beschreiben, ermöglicht. In der Praxis reicht für die meisten meteorologischen Variablen wie z. B. Temperatur eine Ensemblegröße von 5 bis 50 Membern aus. Am Europäischen Zentrum für mittelfristige Wettervorhersage [Med] wird für die Erstellung der (globalen) Wettervorhersagen ein Ensemble mit bis zu 51 Membern eingesetzt. Um systematische Fehler zu korrigieren und somit verlässliche Vorhersagen zu erstellen, benötigen Ensembles eine statistische Nachbearbeitung. Dies wird Kalibrierung genannt. Die zu prognostizierende Variable wird dazu häufig über eine Wahrscheinlichkeitsverteilung ausgedrückt, deren Parameter mithilfe einer Regression anhand eines Trainingsdatensatzes geschätzt werden.

In dieser Bachelorarbeit soll zur Vorhersage der Temperatur in 2 m Höhe ein Ensemble des WRF mit 30 Membern untersucht und kalibriert werden. Für den Einsatz bei Temperaturvorhersagen sind Verfahren wie Ensemble Model Output Statistics [Gne+04], Quantile Regression Forests [Tai+16] und insbesondere künstliche neuronale Netze [RL18] aktuell sehr erfolgreich.

1.2. Einfluss räumlicher Muster

Zur Kalibrierung des Ensembles werden in dieser Arbeit Messungen der Temperatur in 2 m Höhe herangezogen, die an einigen Messstationen in Europa aufgenommen wurden. Im Gegensatz zur gängigen Vorgehensweise, bei der nur die Modellwerte an den Messstationen für die Kalibrierung genutzt werden, soll in dieser Arbeit der Einfluss räumlicher Muster in den Modelldaten untersucht werden. Dazu werden quadratische Umgebungen in den Modelldaten um die einzelnen Messstationen herum in Betracht gezogen. Es wird geprüft, ob durch Hinzunahme der räumlichen Umgebungen der Messstationen aus dem Modell eine bessere Kalibrierung und somit eine höhere Vorhersagequalität des Ensembles erzielt werden kann.

Künstliche neuronale Netze konnten bereits in vielen Anwendungsgebieten Merkmale und Gesetzmäßigkeiten in mehrdimensionalen Signalen erkennen und zur Verbesserung ihrer Performanz nutzen. Sie können zudem nichtlineare Zusammenhänge in den Daten erkennen und sind in ihrer Anwendung sehr flexibel. Daher konzentriert sich diese Arbeit bei der Untersuchung räumlicher Muster auf die Anwendung tiefer neuronaler Netze. Neben den bereits in [RL18] zur Kalibrierung eingesetzten Feedforward-Netzen werden in dieser Arbeit auch neuronale Faltungsnetze eingesetzt, da sie sich besonders gut für die Verarbeitung mehrdimensionaler Daten eignen.

2. Problemstellung

In dieser Bachelorarbeit soll die Kalibrierung eines meteorologischen Ensembles anhand von Temperaturvorhersagen mittels tiefer neuronaler Netze durchgeführt werden. Dabei soll insbesondere der Einfluss räumlicher Merkmale im Modell untersucht werden. In diesem Kapitel wird die für diese Arbeit genutzte Datengrundlage beschrieben. Außerdem werden die Eigenschaften der Kalibrierung und Schärfe probabilistischer Vorhersagen erklärt. Darüber hinaus werden erste Vorbereitungsschritte erläutert und das eingesetzte Rechnersystem kurz vorgestellt.

2.1. Meteorologisches Modell

In dieser Arbeit wird das Weather Research and Forecasting Model (WRF, [UCA18]) eingesetzt. Es handelt sich dabei um ein numerisches System zur Wettervorhersage, das sowohl in der Atmosphärenforschung als auch bei Anwendungen zur operationellen Vorhersage Einsatz findet. Das Modell kann basierend auf aktuellen (oder idealen) atmosphärischen Bedingungen Simulationen erzeugen. Das mathematische Modell des WRF besteht aus einem System von partiellen Differentialgleichungen, mit denen die physikalischen Vorgänge in der Atmosphäre beschrieben werden. Da das System analytisch nicht lösbar ist, wird die Lösung numerisch auf ein Gitter approximiert.

Die für diese Arbeit verwendeten Modelldaten des WRF liegen im NetCDF-Format vor. Dort wird ein Teil Europas als Gitter mit einer Auflösung von 20 km in Süd-Nord- und West-Ost-Richtung abgedeckt. Es werden außerdem 49 Höhenschichten bereitgestellt. Die Höhen werden mithilfe von Sigma-Koordinaten modelliert, d. h. es handelt sich um Höhenschichten konstanten Luftdrucks. Die Modelldateien sind für den Zeitraum vom 31. März 2015 bis zum 30. September 2015 verfügbar. Dies sind insgesamt 184 Tage. Für jeden Tag liegt je eine Modelldatei vor, die stündliche Modellwerte für je einen Vorhersagezeitraum von 49 Stunden enthält. In dieser Arbeit werden lediglich die Vorhersagen des nächsten Tages (also den ersten 24 Stunden des Vorhersagezeitraums) untersucht, da die Ensemble-Kalibrierung für kurzfristige Vorhersagen eingesetzt werden soll.

Im Modell werden Vorhersagen zu einigen meteorologischen Variablen wie z. B. Wind, Temperatur und Luftdruck erstellt. Je nach Variable liegen die Modellwerte nur für die Erdoberfläche oder alle Höhenschichten vor. Zusätzlich ist Temperatur auf 2 m Höhe und die Windgeschwindigkeit auf 10 m Höhe interpoliert, da dies in der Meteorologie übliche Größen sind. Das vom Modell abgedeckte Gebiet ist gleichmäßig in Gitterboxen eingeteilt. Dabei repräsentiert ein Modellwert in der Regel die Mitte der zugehörigen Gitterbox. Manche Variablen (wie z. B. Wind) können entlang einer der Raumdimensio-

nen gestaggert sein, d. h. sie enthalten nicht die Werte in der Mitte, sondern genau an den Rändern der einzelnen Gitterboxen. Um solche Variablen an den Gitterboxmitten zu interpolieren, werden je zwei nebeneinanderliegende Werte entlang der betroffenen Dimension gemittelt. Dieser Prozess wird Destaggern genannt und musste bei der Vorbereitung der Daten für diese Arbeit durchgeführt werden. Da bei gestaggerten Variablen genau ein Wert in der gestaggerten Dimension zusätzlich enthalten ist, können somit die Arrays auf die gleiche Größe abgebildet werden wie ungestaggerte Arrays. Dadurch wird eine programmtechnisch und inhaltlich gleiche Behandlung aller Variablen ermöglicht.

2.1.1. Ensemble-Vorhersagen

Numerische Wettervorhersagen wie im WRF werden mithilfe eines Anfangswertproblems modelliert und sind mit einer Unsicherheit behaftet, die durch ungenaue Anfangswerte sowie Fehler und Unsicherheiten im Modell zustande kommen [Ber18]. Der aktuelle Zustand der Atmosphäre, der zur exakten Lösung des Anfangswertproblems bekannt sein muss, kann nur näherungsweise ermittelt werden, da Messungen nicht für das gesamte Volumen vorliegen und ebenfalls mit Messfehlern behaftet sein können. Zudem fließen Annahmen in die Formulierungen des Modells ein. Darüber hinaus werden die abzubildenden physischen Prozesse approximiert und die numerische Lösung von Anfangswertproblemen auf einem Gitter führt zu Ungenauigkeiten.

Die Unsicherheiten in den Anfangswerten, Randwerten und anderen Parametern im Modell können durch Verwendung eines Ensembles quantifiziert werden. Dabei werden mehrere Modelle mit unterschiedlichen Anfangszuständen und Einstellungen gerechnet. Die einzelnen Modelle, aus denen ein Ensemble besteht, werden Member genannt. Bei einem Ensemble sollen mit einer endlichen Anzahl von Membern, die mit verschiedenen Modellformulierungen und mit gestörten Anfangswerten gerechnet wurden, Vorhersagen erzeugt werden, die von einer Stichprobe der echten Wahrscheinlichkeitsverteilung des Wetters nicht unterscheidbar ist. Für kurze Vorhersagezeiten bleiben die Vorhersagen der einzelnen Member in der Regel nah beieinander. Mit fortschreitender Vorhersagezeit laufen die Vorhersagen oftmals auseinander, was die Unsicherheiten in der Vorhersage des Wetters widerspiegelt.

Aus den einzelnen Vorhersagen der Member kann die Wahrscheinlichkeitsverteilung der vorherzusagenden Variable geschätzt werden. Dies wird probabilistische Vorhersage genannt. Wichtige Kenngrößen bilden dabei das Ensemblemittel und die Ensemblevarianz. Sie beschreiben das arithmetische Mittel bzw. die Varianz der Membervorhersagen zu einem Zeitpunkt und sind Schätzer für den Erwartungswert und die Varianz der zugrundeliegenden Wahrscheinlichkeitsverteilung. Im Idealfall entspricht die geschätzte Verteilung der Wahrscheinlichkeitsverteilung des Vorhersagemodells. Da das Wetter allerdings nur eine Realisierung der Verteilung liefert, können Ensemblevorhersagen nur über einen langen Beobachtungszeitraum verifiziert werden [TG09].

Wegen limitierter Rechenleistung ist die Anzahl der Member eines Ensembles begrenzt. Dadurch wird maßgeblich seine Performanz eingeschränkt. In der Meteorologie sind 5 bis 50 Member für ein Ensemble eine übliche Größe. Diese Bachelorarbeit stützt sich auf ein Ensemble des WRF, von dem 30 Member eingesetzt werden. Zu jedem dieser

Member existiert für jeden Tag eine eigene Modelldatei.

2.1.2. Kalibrierung und Schärfe

Die einzelnen Membervorhersagen repräsentieren die Wahrscheinlichkeitsverteilung des Wetters meist nicht hinreichend. Der Übergang von einzelnen Membervorhersagen zu einer Wahrscheinlichkeitsverteilung, die die Unsicherheit der Vorhersage gut beschreibt, kann mithilfe verschiedener Methoden geschehen. Um die Güte einer geschätzten Verteilung zu beschreiben, wird die Kalibrierung und die Schärfe der probabilistischen Vorhersage nach [GBR07] betrachtet. Eine Vorhersage gilt als kalibriert, wenn die reale Beobachtung nicht von einer zufälligen Stichprobe der vorhergesagten Wahrscheinlichkeitsverteilung unterschieden werden kann. Die Kalibrierung stellt damit die statistische Konsistenz zwischen Vorhersageverteilung und Beobachtung dar. Die Schärfe hingegen beschreibt die Konzentration der Wahrscheinlichkeitsfunktion bzw. der Dichtefunktion und ist somit im Gegensatz zur Kalibrierung lediglich eine Eigenschaft der Vorhersage unabhängig von der Beobachtung. Das Ziel ist es, die Schärfe der Ensemblevorhersagen zu maximieren und gleichzeitig kalibrierte Vorhersagen zu erzielen. Dazu existieren einige Verfahren, von denen ausgewählte in Kapitel 3 beschrieben werden.

Die Kalibrierung und Schärfe von Ensemblevorhersagen kann mittels Verification-Rank-Histogrammen (auch Talagrand-Diagramme genannt, [TVS97]) veranschaulicht werden. Um die Grundidee der Diagramme zu erklären, werden im Folgenden die Membervorhersagen $(X_1, \ldots, X_n)$ zu einem festen Zeitpunkt mit $X_1 \leq X_2 \leq \ldots \leq X_n$ betrachtet. Die Beobachtung ist nicht bekannt und wird daher zunächst mit der Zufallsvariablen Y beschrieben. Die Verteilung von Y gilt als kalibriert, wenn ihre Wahrscheinlichkeiten denen des real eintretenden Wetters entsprechen. Unter der Voraussetzung, dass Y kalibriert ist, können die Membervorhersagen und die tatsächliche Beobachtung y als Stichprobe derselben Wahrscheinlichkeitsverteilung betrachtet werden. Würde man die Beobachtung und die Membervorhersagen zusammennehmen und aufsteigend sortieren, wäre die Beobachtung demnach an jeder Stelle der sortierten Werte mit derselben Wahrscheinlichkeit zu erwarten. Es gilt nach [Ham01]

$$E[P(Y < X_i)] = \frac{i}{n+1} \tag{2.1}$$

mit E als Erwartungswert und P als Wahrscheinlichkeitsfunktion. Erweitert man die Membervorhersagen um X_0 und X_{n+1}, sodass $P(Y < X_0) = 0$ und $P(Y < X_{n+1}) = 1$ gilt, also alle möglichen Realisierungen von Y im Intervall $[X_0; X_{n+1})$ liegen, dann ergibt sich aus Gleichung (2.1)

$$E[P(X_{i-1} \leq Y < X_i)] = \frac{1}{n+1}. \tag{2.2}$$

Damit ist die erwartete Wahrscheinlichkeit, dass die tatsächliche Beobachtung zwischen X_i und X_{i+1} liegt, für alle i gleich. Die Beobachtung ist also an jeder Position in den aufsteigenden Vorhersagen mit gleicher Wahrscheinlichkeit einzusortieren.

Um mithilfe dieser Kenntnisse ein Verification-Rank-Histogramm zu erstellen, werden zunächst für jeden Datenpunkt die einzelnen Membervorhersagen der Größe nach sortiert. Sie bilden die Grenzen der Klassen des Histogramms. Zusätzlich wird eine Klasse für Werte $y < X_1$ und eine Klasse für Werte $y \geq X_n$ eingeführt. Anschließend wird die zugehörige Beobachtung in die entsprechende Klasse einsortiert. Die Verteilung aller Beobachtungen auf die zugehörigen Klassen wird dann im Histogramm dargestellt. Dazu wird die relative Häufigkeitsdichte der einsortierten Beobachtungen gegen die Ordnungszahlen der Klassen (Ränge) aufgetragen. Wenn die Verteilung kalibriert ist, würde gemäß Gleichung (2.2) eine Gleichverteilung vorliegen. Eine Gleichverteilung im Histogramm ist ein notwendiges, aber nicht hinreichendes Kriterium für ein kalibriertes Ensemble [Ham01]. Das Verification-Rank-Histogramm des in dieser Arbeit verwendeten Ensembles ist in Abb. 2.1 zu sehen. Die rot gestrichelte Linie zeigt die Höhe der Flächen, die bei einer Gleichverteilung vorliegen müsste. Man erkennt in der Abbildung eine U-Form, wie es meist bei meteorologischen Ensembles der Fall ist. Diese Form weist auf eine zu geringe Ensemblevarianz hin, da viele Beobachtungen außerhalb des durch die Membervorhersagen gegebenen Rahmens liegen.

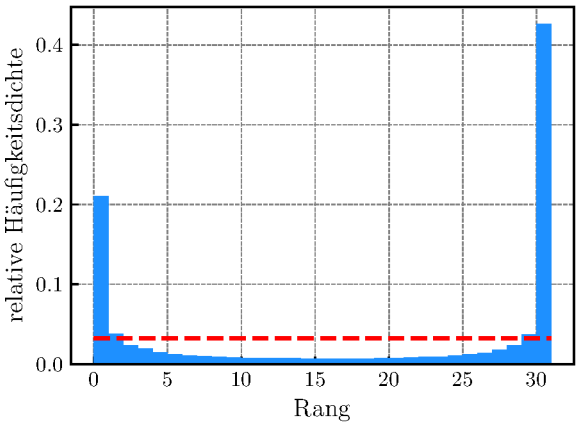

Abbildung 2.1.: Verification-Rank-Histogramm des verwendeten Ensembles

Falls anstelle einzelner Membervorhersagen bereits durch eine statistische Nachbearbeitung eine stetige Wahrscheinlichkeitsverteilung als probabilistische Vorhersage vorliegt, ist das Erstellen eines Verification-Rank-Histogramms nach dem oben beschriebenen Prinzip nicht möglich. Statt die Beobachtung in die sortierten Membervorhersagen einzuordnen, wird nach [GBR07] der Probability-Integral-Transform-Wert (PIT) betrachtet. Dieser ergibt sich als

$$p_i = F_i(y_i) \tag{2.3}$$

mit F_i als Verteilungsfunktion der Vorhersage und der zugehörigen Beobachtung y_i. Im Fall von idealen Vorhersagen F_i ist p_i gleichverteilt auf dem Intervall $[0; 1]$. Zu jedem Sample wird der PIT-Wert berechnet. Anschließend werden die Häufigkeiten der PIT-Werte in einem Histogramm dargestellt. Die Klassen sind in diesem Fall nicht durch

die Vorhersagen gegeben, sondern können frei gewählt werden. Die Interpretation eines PIT-Histogramms geschieht analog zu Verification-Rank-Histogrammen.

2.2. Messungen

Neben dem Modell liegen auch Messungen der Lufttemperatur (in 2 m Höhe) und der horizontalen Windgeschwindigkeit (in 10 m Höhe) im NetCDF-Format vor. Diese wurden an 168 Messstationen in Deutschland und in einigen Nachbarländern aufgenommen und sind Teil des Datensatzes NCEP ADP Global Upper Air and Surface Weather Observations [RL08]. Die Messungen aller Stationen sind für je einen Zeitraum von 6 Stunden in einer NetCDF-Datei gespeichert. Die Messungen eines gesamten Tages verteilen sich demnach auf vier Dateien. In jeder dieser Dateien liegen die Werte zu den gemessenen Größen in je einem Array bereit. In diesen Arrays sind für jeden Messwert der Längen- und Breitengrad sowie die Höhe der jeweiligen Messstation enthalten. Zusätzlich liegt zu jeder Variable noch ein Array vor, das die Zeitstempel der zugehörigen Messungen enthält. Somit sind die genauen Aufnahmezeiten der Messwerte bekannt. Zuletzt ist zu jedem Messwert die Stationskennung der zugehörigen Messstation hinterlegt.

Die Messungen können mit den auf die entsprechende Höhe interpolierten Modellwerten der Temperatur und des Windes verglichen werden, die im Modell bereits vorliegen. Die Messdaten wurden jedoch nicht genau zu jeder vollen Stunde, sondern je nach Messstation zu unterschiedlichen Zeitpunkten und in unterschiedlichen Abständen gemessen. Da die Modellwerte stündlich vorliegen, müssen die Messdaten im Vorhinein auf ganze Stunden interpoliert werden. Dazu wird jeweils derjenige Messwert einer Messstation bestimmt, der zeitlich am nächsten zum Modellzeitpunkt liegt, aber höchstens eine Viertelstunde vom Modellzeitpunkt entfernt ist. Diese interpolierten Messwerte aller Stationen werden für den gesamten Zeitraum wieder in einer NetCDF-Datei gespeichert. In dieser Datei sind auch die stationsspezifischen Informationen wie Längen-, Breitengrad und Höhe zu jeder Station enthalten. Da nicht zu jedem Modellzeitpunkt bei jeder Messstation eine Messung vorliegt, werden Lücken in den Messungen mit speziellen Füllwerten belegt, die das Fehlen eines Messwerts signalisieren.

Für diese Bachelorarbeit wurden die Daten von 100 der 168 Messstationen in Betracht gezogen. Dabei wurden Stationen aus dem gesamten verfügbaren Gebiet gewählt. Die Standorte der genutzten Stationen sind in Abb. 2.2 gelb markiert. Die restlichen Stationen sind rot eingezeichnet.

2.3. Vorbereitung

Um eine Kalibrierung des vorliegenden Ensembles durchführen zu können, müssen die Modell- und Messdaten zunächst vorbereitet werden. Ziel dabei ist es, die Modelldaten mit den Messungen so zusammenzubringen, dass die Daten ohne großen Aufwand von den einzusetzenden Verfahren zur Kalibrierung genutzt werden können. Dazu müssen jeder einzelnen Messung die entsprechenden Modellvorhersagen zugeordnet werden. Da-

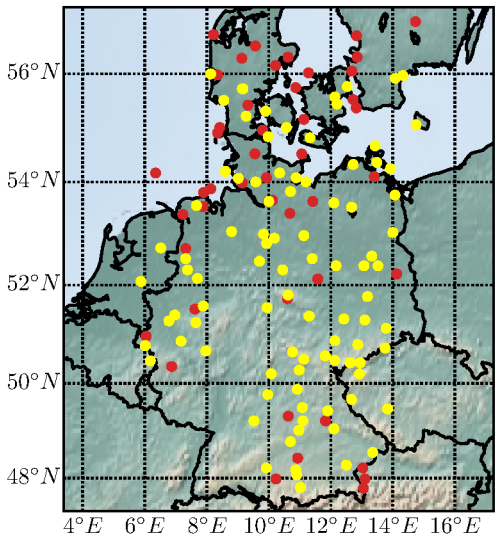

Abbildung 2.2.: Standorte der Messstationen

bei ist zu beachten, dass durch die Nutzung des Ensembles die Vorhersagen von den gesamten 30 Membern verarbeitet werden müssen. In bisherigen Verfahren [Gne+04; Tai+16; RL18] ist es üblich, nur die Modellwerte an den Messstationen als Prädiktoren für die Ensemble-Kalibrierung einzusetzen. Um räumliche Muster und Merkmale in den Modelldaten untersuchen zu können, werden in dieser Arbeit quadratische Umgebungen aus der untersten Höhenschicht des Modells entnommen, bei denen sich jeweils der Standort einer Messstation in der mittleren Gitterbox befindet. Dies ist in Abb. 2.3 veranschaulicht. Dazu wird der Standort der Messstation mithilfe ihres Längen- und

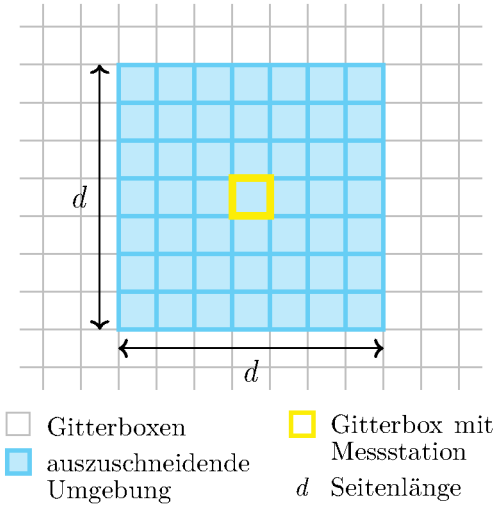

Abbildung 2.3.: Vorgehensweise zur Entnahme von Umgebungen aus dem Modell

Breitengrads im Modellgitter bestimmt. Die dazugehörige Gitterbox bildet den Mittelpunkt der zu entnehmenden Umgebung. Im Fall dieser Arbeit wurden Umgebungen mit einer Seitenlänge von 11 Modellwerten entnommen. Aufgrund der Gitterauflösung von 20 km entspricht dies einer tatsächlichen Seitenlänge von 220 km.

Diese Umgebungen müssen von jedem der 30 Member entnommen werden. Zu einem Messwert gehören folglich 30 Umgebungen der Größe 11×11. Die Messungen einer Station decken nicht zwingend den gesamten Modellzeitraum ab. Zeitpunkte, an denen bei einer Station anstelle eines Messwerts ein Füllwert vorliegt, müssen bei der Vorbereitung der Daten ignoriert werden. Neben den Mess- und Modellwerten werden zusätzlich die Informationen zu Längen-, Breitengrad und Höhe der Messstationen für jeden Zeitpunkt mitgespeichert. Diese Informationen können bei den Verfahren zur Kalibrierung zusätzlich als Prädiktoren einfließen. Im Gegensatz zu [RL18] werden in diesem Fall keine Stationskennungen gespeichert und mitverarbeitet, damit die Kalibrierung nicht auf die Messstationen beschränkt ist, sondern prinzipiell am ganzen Modellgitter angewendet werden kann.

Für diese Arbeit werden nur bestimmte Variablen aus dem Modell betrachtet. Diese sind in Tabelle 2.1 dargestellt. Außerdem sind dort auch die Informationen zu den Messstationen aufgeführt, die zusätzlich in die Kalibrierung einfließen. Da die Modellwerte als zweidimensionale Umgebung genutzt werden, können diese nicht zusammen mit den skalaren stationsspezifischen Informationen in einem Array gespeichert werden. Daher sind die Koordinaten der Messstationen separat in einem Array der Form `(n, 3)` mit `n` als Anzahl der Datenpunkte gespeichert. Die Umgebungen werden für jedes Member einzeln in einem Array der Form `(n, 1, 11, 11, 16)` hinterlegt. Die letzte Dimension dient der Indizierung der verschiedenen Variablen. Entlang der zweiten Dimension können die Arrays aller Member aneinandergehängt werden. Dann ergibt sich ein Array der Form `(n, 30, 11, 11, 16)`, in dem alle für diese Arbeit relevanten Membervorhersagen gespeichert sind. Die Messungen werden in einem eindimensionalen Array der Größe `n` hinterlegt. In dem für diese Arbeit eingesetzten Datensatz, der den gesamten verfügbaren Modellzeitraum abdeckt, befinden sich `n = 393073` Datenpunkte. Durch die verteilte Speicherung der Daten auf verschiedene Arrays ergeben sich individuelle Möglichkeiten zur Nutzung der Daten, ohne jeweils den kompletten Datensatz laden zu müssen.

Für die Berechnungen zur Ensemble-Kalibrierung im Zuge dieser Arbeit steht das Pilotsystem JURON [Jül16] zur Verfügung. Es befindet sich im Jülich Supercomputing Centre (JSC) am Forschungszentrum Jülich. Es setzt sich aus 18 IBM-Servern zusammen, die je mit 2 IBM POWER8-Prozessoren sowie 4 NVIDIA Tesla P100 GPUs ausgestattet sind und je 256 GB Speicher bereitstellen, um riesige Datenmengen insbesondere zur Simulation des menschlichen Gehirns im Rahmen des Human Brain Project [Pro] zu verarbeiten. Mittlerweile steht JURON auch für Projekte anderer Fachrichtungen zur Verfügung. Als Programmiersprache kommt in dieser Arbeit hauptsächlich Python 3.6.1 [Tea15] zum Einsatz. Für die oben dargelegten Vorbereitungsschritte wurden die Module `netCDF4` [Whi] sowie `numpy` [Oli06] verwendet.

Tabelle 2.1.: Übersicht über die verwendeten Variablen

Variable	**Einheit**	**Beschreibung**
Modellvorhersagen		
T2	K	Lufttemperatur in 2 m Höhe
PSFC	Pa	Luftdruck am Boden
CFRACT		gesamter Wolkenbedeckungsgrad
HFX	W/m^2	aufwärts gerichteter Wärmefluss
LH	W/m^2	latenter Wärmefluss
U_10	m/s	ostwärts gerichtete Windgeschwindigkeit in 10 m Höhe
V_10	m/s	nordwärts gerichtete Windgeschwindigkeit in 10 m Höhe
Q2	kg/kg	spezifische Luftfeuchtigkeit in 2 m Höhe
SWDOWN	W/m^2	abwärts gerichteter kurzwelliger Strahlungsfluss
GLW	W/m^2	abwärts gerichteter langwelliger Strahlungsfluss
QVAPOR	kg/kg	Wasserdampf-Mischungsverhältnis
P	Pa	Störung des Luftdrucks
PB	Pa	Grundzustand des Luftdrucks
XLAT	°N	Breitengrad der Gitterboxmitte
XLONG	°E	Längengrad der Gitterboxmitte
HGT	m	Höhe der Erdoberfläche
stationsspezifische Informationen		
LAT	°N	Breitengrad der Messstation
LON	°E	Längengrad der Messstation
ALT	m	Höhe der Messstation

3. Verfahren zur Kalibrierung

Meist neigen Ensembles von meteorologischen Modellen zu unterdispersiven Vorhersagen, d. h. die Varianz des Ensembles ist zu gering [Gne+04]. Um dem entgegenzuwirken, können Ensembles mit verschiedenen Verfahren statistisch nachbearbeitet werden. Aufgabe eines solchen Verfahrens ist es, aus den einzelnen Membervorhersagen zu ggf. verschiedenen Variablen eine Wahrscheinlichkeitsverteilung zu schätzen, sodass diese Verteilung in Bezug auf die Messungen möglichst scharf und kalibriert ist.

In Abb. 3.1 ist der gesamte Ablauf der Ensemble-Kalibrierung für diese Arbeit grob dargestellt. Nach der Vorbereitung der Daten, die bereits in Abschnitt 2.3 beschrieben wurde, wird ein statistisches Verfahren angewendet, das aus den gegebenen Daten kalibrierte Vorhersagen gewinnt. Die Kalibrierung wird später mithilfe des Continuous Ranked Probability Score (CRPS, Abschnitt 3.2) und PIT-Histogrammen (Abschnitt 2.1.2) evaluiert. In diesem Kapitel werden aktuelle Verfahren zur Ensemble-Kalibrierung vorgestellt, die dem mittleren Teil (Kalibrierung) des in Abb. 3.1 dargestellten Ablaufs zuzuordnen sind. Außerdem werden alle nötigen Grundlagen zum Verständnis dieser Verfahren erläutert. Die vorgestellten Verfahren nutzen die bereits vorliegenden Vorhersagen des meteorologischen Modells. Es kann sich bei den Verfahren allerdings ebenfalls um Ensemble-Techniken handeln (Abschnitt 3.1.3), die vom Ensemble des meteorologischen Modells stets unabhängig und getrennt zu betrachten sind.

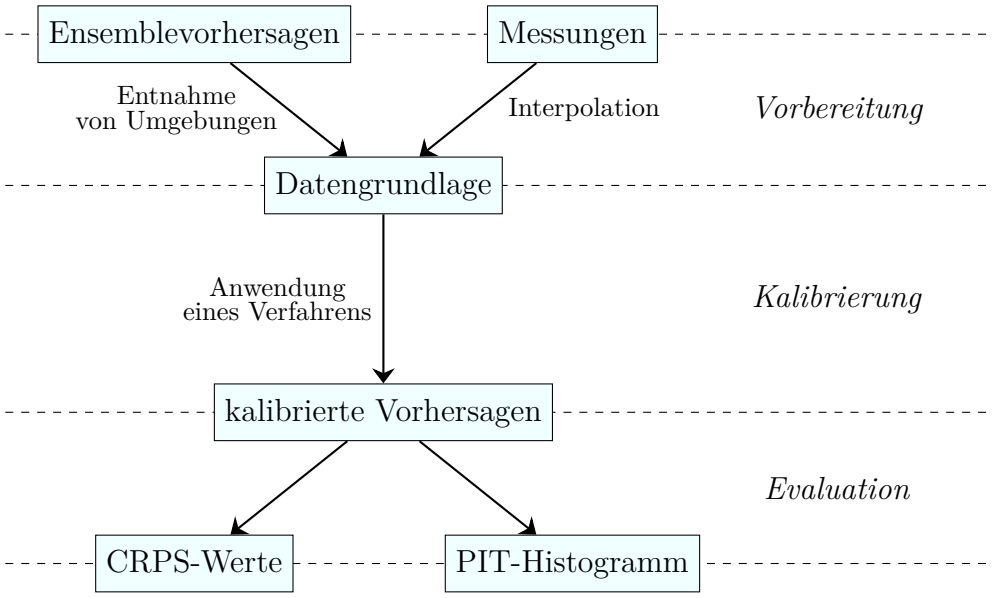

Abbildung 3.1.: Schaubild zum groben Ablauf der Ensemble-Kalibrierung

3.1. Grundlagen des Machine Learning

Im Gebiet des Machine Learning soll Wissen aus Erfahrung mittels Modellen und Lernalgorithmen modelliert werden. Das Ziel ist, funktionale Zusammenhänge sowie Muster und Gesetzmäßigkeiten in den vorliegenden Daten zu erkennen, um Lösungen für die zugrundeliegenden Probleme zu finden. Dazu lässt sich das Gebiet grob in die Teilgebiete des überwachten, unüberwachten und bestärkenden Lernens gliedern.

In dieser Arbeit wird überwachtes Lernen durchgeführt. Dabei soll ein Lernalgorithmus die Beziehung zwischen Eingabedaten $\mathbb{X}$ und Zielwerten $\mathbb{Y}$ erlernen, wobei schon während des Lernprozesses die Richtigkeit dieser Zuordnung überprüft werden kann. Dazu ist ein Datensatz $\mathbb{D}$ gegeben. In diesem Datensatz befinden sich Datenpunkte (auch Samples genannt) in Form von Tupeln $(x_i, y_i) \in \mathbb{D}$, die jeder Eingabe $x_i \in \mathbb{X}$ einen gewünschten zugehörigen Zielwert $y_i \in \mathbb{Y}$ (auch Label genannt) zuordnen. Der Datensatz wird als Stichprobe einer multivariaten Wahrscheinlichkeitsverteilung $p_{\text{data}}(x, y)$ betrachtet. Dabei sind die Samples unabhängig und identisch verteilt.

Da die Zusammenhänge zwischen den Eingaben x_i und den Labels y_i oft komplex sind und von verschiedenen Merkmalen und Eigenschaften des betrachteten Problems abhängen, werden häufig mehrere Variablen als Prädiktoren bei Lernalgorithmen eingesetzt. Diese werden auch Features genannt. Eine Eingabe mit m Features lässt sich als Vektor $x_i \in \mathbb{R}^m$ ausdrücken. Jede Komponente des Vektors repräsentiert den Wert eines bestimmten Features.

Es wird oft angenommen, dass eine Funktion $f : \mathbb{X} \to \mathbb{Y}$ existiert, die die vorliegenden Daten erzeugt hat. Ziel eines Lernalgorithmus ist es, diese wahre Zielfunktion f mit einer Funktion $g : \mathbb{X} \to \mathbb{Y}$ möglichst gut zu approximieren. Dazu wird einem Algorithmus wie in [Alp10] beschrieben in der Regel eine Menge von Kandidatenfunktionen $\mathbb{H}$ bereitgestellt. Diese ergibt sich häufig aus der Struktur des Lernmodells und wird Hypothesen-Set genannt. So kann z. B. eine einfache lineare Regression ohne vorige Transformationen der Eingabedaten lediglich finale Hypothesen g in Form einer linearen Funktion finden, nicht etwa eine Parabel oder Exponentialfunktion. Der Lernalgorithmus selektiert die finale Hypothese $g \in \mathbb{H}$ so, dass möglichst die Zielfunktion f getroffen wird, also $g(x_i) \approx f(x_i)\ \forall x_i \in \mathbb{X}$.

Im Teilgebiet des überwachten Lernens unterscheidet man zwischen zwei Arten von Problemen: der Klassifikation und der Regression. In dieser Arbeit wird eine Regression durchgeführt. Dort besteht das Bild von f in der Regel aus reellen Zahlen, die approximiert werden sollen. Um die Performanz einer Hypothese g auf einem Datensatz zu messen, wird eine Verlustfunktion verwendet, die den Fehler zwischen den Vorhersagen des Lernalgorithmus $g(x_i)$ und den zugehörigen Labels y_i bestimmt. Für eine einfache Regression wird dafür häufig der quadratische Fehler verwendet. In dieser Arbeit ist dies jedoch nicht möglich. Die Vorhersage des Lernalgorithmus $g(x_i)$ ist hier eine Wahrscheinlichkeitsverteilung, die die kalibrierte probabilistische Vorhersage des Ensembles darstellt. Diese muss mit der zugehörigen skalaren Beobachtung y_i in Bezug gebracht werden. Daher muss der Fehler auf eine andere Art bestimmt werden. Die dazu in dieser Arbeit verwendete Verlustfunktion wird in Abschnitt 3.2 vorgestellt.

Die Vorhersagen eines Lernmodells repräsentieren eine bedingte Wahrscheinlichkeits-

verteilung $p_\vartheta(y \mid x)$, wie es in [GBC16] beschrieben wird. Durch die Struktur und Einstellungen am Modell (wie dem Hypothesen-Set) liegt dabei eine Familie von möglichen Wahrscheinlichkeitsverteilungen parametrisiert durch ϑ bereit. Jede konkret mögliche Ausprägung der Parameter ϑ resultiert in einer anderen finalen Hypothese g. Damit möglichst gute Vorhersagen auf unbekannten Daten erzielt werden, soll die Modellverteilung p_ϑ möglichst nah an die bedingte datenerzeugende Wahrscheinlichkeitsverteilung $p_{\text{data}}(y \mid x)$ angepasst werden. Die möglichen Modellverteilungen p_ϑ sowie das Hypothesen-Set $\mathbb{H}$ hängen eng mit der Modellkomplexität zusammen. Diese beschreibt die Fähigkeit des Modells, die Daten mittels geeigneter Hypothesen zu approximieren. Die Komplexität kann insbesondere durch Anpassung des Hypothesen-Sets beeinflusst werden. Ein Lernmodell erzielt nach [Alp10] die besten Ergebnisse, wenn die Modellkomplexität zur Komplexität des zu lösenden Problems und der Größe des vorhandenen Datensatzes passt.

Der Prozess der Wahl einer finalen Hypothese g, bei der der Fehler auf den gegebenen Daten minimiert wird, wird Training genannt. Der Fehler auf den im Training verwendeten Daten wird als Stichprobenfehler, Trainingsfehler oder In-Sample-Fehler E_{in} bezeichnet. Er wird mittels einer speziell gewählten Verlustfunktion berechnet, die die Vorhersagen eines Modells mit den Labels vergleicht. Der Fehler, den der Algorithmus nach dem Training auf allen erdenklichen Daten der jeweiligen Anwendung hat, heißt Generalisierungsfehler oder Out-of-Sample-Fehler E_{out}. Da dieser Fehler nur in seltenen Fällen exakt berechenbar ist, wird er meist durch Nutzung eines speziellen Datensatzes geschätzt. Dieser Datensatz enthält Datenpunkte, die dem Algorithmus im Training zuvor nicht zur Verfügung gestellt wurden, also nicht zur Findung der finalen Hypothese g in Betracht gezogen wurden. Die Schätzung des Out-of-Sample-Fehlers E_{out} wird auf bis dahin für das Lernmodell unbekannten Daten berechnet. Diese Daten werden meist direkt zu Beginn eines Machine-Learning-Projekts aus dem vorliegenden Datensatz entnommen. Der Datensatz $\mathbb{D}$ wird dazu in zwei Datensätze aufgeteilt: den Trainingsdatensatz, der dem Training des Lernalgorithmus dient, und den Testdatensatz. Die Testdaten werden erst am Ende eines Projekts dazu genutzt, den Out-of-Sample-Fehler E_{out} zu schätzen.

Um mit einem Lernalgorithmus möglichst gute Ergebnisse zu erzielen, müssen einige Dinge beachtet werden, wie sie in [Alp10] beschrieben werden. Zum einen ist es wichtig, während des Lernprozesses keine Entscheidung aufgrund der Testdaten zu treffen. Diese dienen allein der Schätzung des Out-of-Sample-Fehlers E_{out}. Jede Entscheidung, die auf Grundlage der Testdaten getätigt wurde, verzerrt die Schätzung von E_{out} auf positive Weise. Daher ist beispielsweise bei Vorskalierungen der Daten wichtig, nur den Trainingsdatensatz zu betrachten. Fließt dennoch der Testdatensatz in Entscheidungen ein, handelt es sich um sogenanntes Data Snooping. Zum anderen ist die Wahl der Daten wichtig. Falls zuvor bestimmte Daten aussortiert werden, kann es zu einer Stichprobenverzerrung kommen. Die Wahrscheinlichkeitsverteilung, aus denen die Trainingsdaten kommen, entspricht dann nicht mehr der Verteilung, nach der die Daten in der Realität auftreten. Daher ist bei einer Stichprobenverzerrung eine schlechtere Performanz des Algorithmus auf neuen Daten zu erwarten.

3.1.1. Bias-Varianz-Abwägung

Gemäß der statistischen Lerntheorie ist es möglich, im wahrscheinlichkeitstheoretischen Sinn Aussagen über die beobachteten Daten zu treffen. Dazu werden nach [Alp10] im ersten Schritt die Trainingsdaten möglichst gut mit der finalen Hypothese g approximiert. Dabei ist das Ziel, den In-Sample-Fehler E_{in} möglichst gering zu halten. Im zweiten Schritt soll die finale Hypothese g genauso gute Vorhersagen auf unbekannten Daten wie auf den zuvor genutzten Trainingsdaten machen. Dies wird Generalisierung genannt und spiegelt sich darin wider, dass der Out-of-Sample-Fehler etwa so groß wie der In-Sample-Fehler ist, also $E_{\text{out}} \approx E_{\text{in}}$. Da alle Datenpunkte aus derselben datenerzeugenden Wahrscheinlichkeitsverteilung p_{data} gezogen wurden, ist bei perfekter Anpassung der Modellverteilung p_ϑ zu erwarten, dass die Fehler E_{in} und E_{out} etwa gleich sind. Die beiden Eigenschaften der Approximation und der Generalisierung stehen im Konflikt zueinander. So ist in der Praxis häufig zu beobachten, dass perfekte Approximation und schlechte Generalisierung erreicht wird. Daher müssen beide Eigenschaften gezielt gegeneinander abgewogen werden. Wichtige Einflussfaktoren bilden dabei die Größe des Datensatzes und die Komplexität des Lernmodells.

In direkter Verbindung mit der Approximations-Generalisierungsabwägung stehen der Bias (Verzerrung) und die Varianz eines Lernmodells. Der Bias beschreibt die Abweichung zwischen der Zielfunktion f und der über allen möglichen Datensätzen gemittelten finalen Hypothese g. Er kann als prinzipielle Abweichung zwischen Zielfunktion und finaler Hypothese interpretiert werden. Eine geringe Modellkomplexität und das damit verbundene kleine Hypothesen-Set $\mathbb{H}$ resultiert oft in einem großen Bias. Die Varianz hingegen misst die Streuung der finalen Hypothese eines konkreten Datensatzes um die mittlere finale Hypothese. Sie lässt sich als Instabilität des Modells charakterisieren. Eine hohe Instabilität macht sich durch große Variation des Lernmodells bei kleinen Änderungen in den Daten bemerkbar. Ist die Modellkomplexität hoch und damit das Hypothesen-Set $\mathbb{H}$ groß, ist zwar oft ein kleiner Bias, aber eine hohe Varianz zu beobachten. Zudem wird die Varianz durch Vergrößerung des Datensatzes meist geringer. Die Beziehung zwischen Bias, Varianz und Out-of-Sample-Fehler E_{out} in Abhängigkeit der Modellkomplexität ist in Abb. 3.2 veranschaulicht.

Bei der Durchführung eines Lernalgorithmus kann es zu Overfitting (Überanpassung) oder Underfitting (Unteranpassung) kommen. Overfitting wird in [Alp10] als Prozess beschrieben, finale Hypothesen zu wählen, die einen zunehmend geringeren In-Sample-Fehler E_{in} erreichen, aber zu ansteigenden Out-of-Sample-Fehlern E_{out} führen. In so einem Fall werden die Trainingsdaten zwar gut approximiert, unbekannte Daten wie die Testdaten jedoch nicht. Dies macht sich durch eine große Abweichung zwischen E_{in} und E_{out} bemerkbar. Das Modell generalisiert demnach schlecht. Nach [Alp10] ist ein wichtiger Einflussfaktor bei Overfitting, wie die Qualität und die Quantität der gegebenen Daten zur gewählten Modellkomplexität passt. Bei einer zu hoch gewählten Komplexität kann das Modell viele unwichtige Merkmale und Eigenschaften in den Daten lernen, was die Generalisierung negativ beeinflusst. Overfitting kann z. B. durch eine größere Anzahl von Datenpunkten, der Anpassung der Modellkomplexität oder Anwendung von Regularisierung verringert werden. Bei Underfitting werden die Trainingsdaten nicht

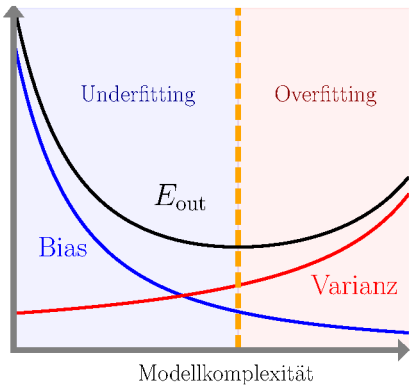

Abbildung 3.2.: Bias, Varianz und Out-of-Sample-Fehler in Abhängigkeit der Modellkomplexität

hinreichend gut beschrieben, da die Modellkomplexität nicht ausreicht, die Beziehung zwischen Ein- und Ausgabe abzubilden. Damit bildet Underfitting das Gegenteil zu Overfitting. Sowohl der Fehler auf den Trainingsdaten als auch der Fehler auf den Testdaten bleibt groß. Die Daten werden dabei schlecht approximiert und das Lernmodell hat einen hohen Bias. Die Auswirkung von Under- und Overfitting bei Näherung einer reellwertigen Funktion ist in Abb. 3.3 dargestellt.

3.1.2. Hyperparameter und Validierung

Um die Modellkomplexität und das Hypothesen-Set zu beeinflussen, können verschiedene Anpassungen am Lernmodell vorgenommen werden. Die einzustellenden Parameter an einem Modell werden Hyperparameter genannt. Es ist nicht empfehlenswert, Hyperparameter, die das Hypothesen-Set beeinflussen, als freie Parameter im Modell mitzulernen, da sich dadurch die Modellkomplexität erhöhen würde, bis der In-Sample-Fehler E_{in} sehr klein ist. Dies würde sich durch Overfitting bemerkbar machen. Bei manchen Hyperparametern ist es hingegen zu schwierig, sie im Algorithmus geeignet zu optimieren. Daher müssen sie vorab manuell festgelegt werden. Dabei ist auf die Wahl geeigneter Einstel-

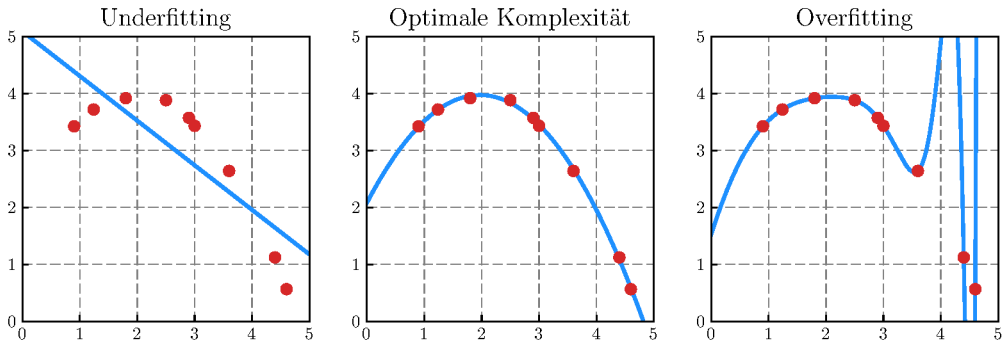

Abbildung 3.3.: Under-, Overfitting und optimales Fitting bei Näherung einer reellen Funktion

lungen zu achten, da diese die Ergebnisse maßgeblich beeinflussen können. Häufig werden verschiedene Konfigurationen der Hyperparameter systematisch ausprobiert, z. B. in Form einer Gittersuche. Inzwischen gibt es jedoch auch andere Verfahren und Strategien zur Findung optimaler Einstellungen, wie in [Sha+16] und [DMC] beschrieben wird. Der Prozess zur Anpassung der Hyperparameter wird als Hyperparameter-Optimierung oder auch Hyperparameter Tuning bezeichnet.

Vor Durchführung eines Lernalgorithmus wird ein Validierungsdatensatz als weiterer Datensatz aus den Trainingsdaten $\mathbb{D}_{\text{train}}$ entnommen. Der übrige Trainingsdatensatz ist die Datengrundlage, die im Trainingsprozess des Modells zum Finden der finalen Hypothese g eingesetzt wird. Der Validierungsdatensatz hingegen dient nach [GBC16] der Schätzung des Out-of-Sample-Fehlers E_{out} schon während des Trainings und wird nicht zur Wahl der finalen Hypothese in Betracht gezogen. So können anhand der Schätzungen von E_{out} während des Trainings Entscheidungen bezüglich des Modells, insbesondere der Hyperparameter, getroffen werden, ohne dabei den Testdatensatz zu nutzen. Es wird am Schluss üblicherweise dasjenige Modell gewählt, das den geringsten Fehler auf den Validierungsdaten vorweist. Dieser Prozess wird Model Selection genannt.

3.1.3. Ensemble-Techniken

Es ist möglich, die Stabilität eines Lernmodells zu erhöhen und damit den Out-of-Sample-Fehler E_{out} zu verringern, indem mehrere Modelle miteinander verknüpft werden. Dabei werden beispielsweise zwei oder mehr Lernalgorithmen durchgeführt und deren Vorhersagen am Schluss gemittelt. Solche und ähnliche Strategien werden auch als Ensemble-Techniken bezeichnet. Die hier im Kontext des Machine Learning vorgestellten Ensemble-Techniken und das Ensemble des meteorologischen Modells basieren zwar auf demselben Prinzip, sind für diese Arbeit jedoch abgegrenzt voneinander zu betrachten.

Dieses Prinzip funktioniert, da verschiedene Lernmodelle in der Regel unterschiedliche Fehler auf dem Testdatensatz machen. Im Folgenden wird von einem Ensemble von k Regressionsmodellen ausgegangen, bei dem jedes der Modelle einen Fehler ε_i auf einem Sample macht. Der Fehler sei normalverteilt mit Erwartungswert 0, Varianz $E[\varepsilon_i^2] = v$ und Kovarianz $E[\varepsilon_i \varepsilon_j] = c$. Der über die k Lernmodelle gemittelte Fehler beträgt dann $\frac{1}{k}\sum_{i=1}^{k} \varepsilon_i$. Der erwartete quadratische Fehler der Vorhersage des Ensembles ist damit nach [GBC16]

$$E\left[\left(\frac{1}{k}\sum_{i=1}^{k}\varepsilon_i\right)^2\right] = \frac{1}{k^2}E\left[\sum_{i=1}^{k}\left(\varepsilon_i^2 + \sum_{j\neq i}\varepsilon_i\varepsilon_j\right)\right] \tag{3.1}$$

$$= \frac{1}{k}v + \frac{k-1}{k}c. \tag{3.2}$$

Sind die Modelle nun alle korreliert mit $c = v$, so ergibt sich der mittlere quadratische Fehler v und das Mitteln der einzelnen Vorhersagen hat keinen Vorteil erbracht. Falls die Modelle jedoch alle genau unkorreliert sind mit $c = 0$, dann beträgt der mittlere

quadratische Fehler nur noch $\frac{1}{k}v$. Damit ist der erwartete Fehler antiproportional zur Anzahl der verwendeten Lernmodelle. Im Mittel ist das Ensemble demnach mindestens so gut wie jedes der zugrundeliegenden Modelle und erreicht bei unabhängigen Fehlern einen signifikant kleineren Gesamtfehler.

Um durch Ensemble-Techniken eine Verbesserung zu erzielen, reicht es, wenn die einzelnen Modelle lediglich minimal besser als bloßes Raten sind. Um bei solchen sogenannten schwachen Lernern trotzdem eine gute Performanz zu erzielen, muss die Größe des Ensembles ausreichen und die Modelle müssen unabhängig voneinander sein. Dieses Prinzip macht sich das Gesetz der großen Zahlen zunutze. Je mehr schwache Lerner ins Ensemble einfließen, desto besser macht sich die minimale Verbesserung gegenüber zufälligen Vorhersagen der einzelnen Modelle bemerkbar.

In den verschiedenen Ensemble-Techniken werden die einzelnen Modelle unterschiedlich erzeugt. Dabei ist es wichtig, dass die Fehler der Modelle untereinander möglichst unabhängig sind. So ist es denkbar, dass jedes Modell einen anderen Lernalgorithmus verwendet oder eine andere Verlustfunktion minimiert. Es ist ebenfalls möglich, denselben Algorithmus wiederzuverwenden, wie es bei Bootstrap Aggregation (kurz Bagging, [Bre96]) der Fall ist. Um dabei möglichst unabhängige Modelle zu erhalten, werden k unterschiedliche Trainingsdatensätze erzeugt, indem sogenannte Bootstrap-Stichproben aus dem Trainingsdatensatz entnommen werden. Dabei werden die einzelnen Datenpunkte des neuen Datensatzes mit Zurücklegen aus dem Trainingsdatensatz gezogen. Da die einzelnen Datensätze meist genauso viele Samples enthalten sollen wie der ursprüngliche Datensatz, führt das Zurücklegen dazu, dass zu einer großen Wahrscheinlichkeit einige Samples mehrfach in einem Datensatz auftreten und andere wiederum nicht enthalten sind. Jedes zu trainierende Modell erhält auf diese Weise einen individuellen Datensatz mittels Resampling. Werden die Datensätze durch Stichproben ohne Zurücklegen erzeugt, spricht man von Pasting.

Beim Bagging und Pasting hat nach [Gér17] jedes einzelne Modell einen höheren Bias, als wenn es mit dem ursprünglichen Datensatz trainiert worden wäre. Bei der Zusammenführung der Ergebnisse im Ensemble werden dann allerdings sowohl der Bias als auch die Varianz gegenüber der einzelnen Modelle verringert. Insgesamt hat das Ensemble schließlich einen ähnlich hohen Bias wie ein einzelnes Modell, das auf dem ursprünglichen Datensatz trainiert wurde, aber eine geringere Varianz. Deshalb eignen sich Bagging und Pasting zur Verringerung der Varianz eines Lernmodells.

Neben der Möglichkeit, die Ergebnisse der Modelle erst am Schluss zusammenzuführen, ist es auch möglich, ein Ensemble sequentiell aufzubauen. Dies wird Boosting genannt [FS96]. Dabei werden nacheinander verschiedene Modelle trainiert, die jeweils den vom Vorgängermodell übrig gebliebenen Fehler korrigieren sollen. Ebenfalls verbreitet ist Stacked Generalization (kurz Stacking, [Wol92]). Dort wird das Zusammenführen der Ergebnisse nicht durch eine vorgegebene Funktion (z. B. arithmetisches Mittel) vorgeschrieben, sondern von einem weiteren Modell gelernt.

Ensemble-Techniken sind sehr mächtig und zuverlässig. So ist Random Forest ein aktuell sehr erfolgreiches Verfahren, bei dem mehrere Lerner (in Form von Entscheidungsbäumen) mittels einer Ensemble-Technik zu einem besonders performanten Modell zusammengefasst werden. Dies wird in Abschnitt 3.4 näher beschrieben. Ein Nachteil

bei Ensemble-Techniken ist allerdings, dass das Trainieren mehrerer Modelle nur durch höheren Zeit- und/oder Speicheraufwand möglich ist.

3.2. Continuous Ranked Probability Score

Der Continuous Ranked Probability Score (CRPS) ist nach [RL18] eine sogenannte Proper Scoring Rule. Eine Scoring Rule beschreibt ein Maß für die Performanz eines Ensembles, indem sie die Kalibrierung und die Schärfe einer probabilistischen Vorhersage mit der zugehörigen Beobachtung vergleicht. Sie bildet eine Vorhersage in Form einer Verteilungsfunktion F und die zugehörige Beobachtung y auf eine reelle Zahl ab. Ist die Scoring Rule genau dann optimiert, wenn die Vorhersageverteilung F der empirischen Verteilung der Beobachtung y entspricht, so handelt es sich um eine Proper Scoring Rule.

Der CRPS kann als Verlustfunktion für die Vorhersage von Wahrscheinlichkeitsverteilungen eingesetzt werden. Er findet häufig bei meteorologischen Modellen Anwendung. Allgemein ist der CRPS definiert als

$$\mathrm{CRPS}(F, y) = \int_{-\infty}^{\infty} \left(F(z) - \mathbb{1}(y \leq z)\right)^2 \mathrm{d}z. \tag{3.3}$$

Dabei ist $F : \mathbb{R} \to [0; 1]$ die Verteilungsfunktion der Vorhersage, $y \in \mathbb{R}$ die Beobachtung und $\mathbb{1}$ die Indikatorfunktion. Der Ausdruck $\mathbb{1}(y \leq z)$ nimmt für alle $z \geq y$ den Wert 1 an, für alle anderen Fälle (also $z < y$) den Wert 0. Dies entspricht der empirischen Verteilung der Beobachtung. Um die Bedeutung des CRPS zu verdeutlichen, kann das Integral in zwei Integrale aufgeteilt werden. In der Form

$$\mathrm{CRPS}(F, y) = \int_{-\infty}^{y} \left(F(z)\right)^2 \mathrm{d}z \; + \; \int_{y}^{\infty} \left(1 - F(z)\right)^2 \mathrm{d}z \tag{3.4}$$

$$= \int_{-\infty}^{y} \left(P(Y \leq z)\right)^2 \mathrm{d}z \; + \; \int_{y}^{\infty} \left(P(Y > z)\right)^2 \mathrm{d}z \tag{3.5}$$

mit Y als Zufallsvariable der Vorhersage wird deutlich, dass für Werte der Integrationsvariablen z ab dem Wert der Beobachtung y die Gegenwahrscheinlichkeiten der Verteilungsfunktion betrachtet werden. Bis zum Wert der Beobachtung wird also die Unterschätzung der Beobachtung $P(Y \leq z)$ mit $z \leq y$ und ab dem Wert der Beobachtung die Überschätzung der Beobachtung $P(Y > z)$ mit $z > y$ betrachtet.

Die grafische Bedeutung des CRPS ist in Abb. 3.4 skizziert. Neben der Verteilungsfunktion der Vorhersage ist die Beobachtung y als empirische Verteilung eingezeichnet. Minimieren des CRPS entspricht dem Minimieren der in Abb. 3.4 rot gekennzeichneten Fläche. Dadurch wird die vorhergesagte Verteilung an die empirische Verteilung der Beobachtung angepasst. Ein kleinerer Wert des CRPS bedeutet also eine bessere Anpassung der vorhergesagten Verteilung an die Beobachtung. Beträgt der CRPS 0, so ist

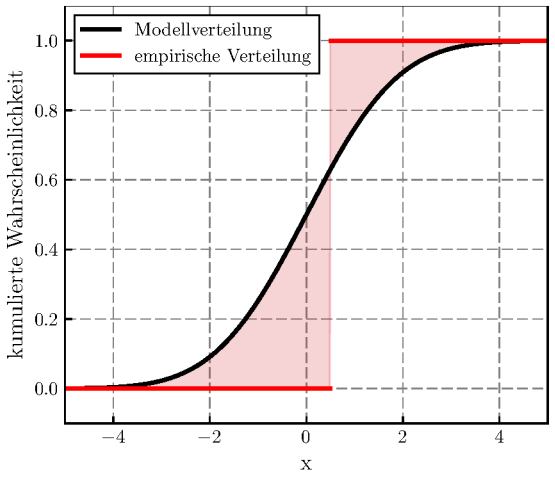

Abbildung 3.4.: Veranschaulichung des CRPS

keine Fläche zwischen den Verteilungen vorhanden und die Vorhersage entspricht der Beobachtung. Negative Werte sind nicht möglich. Die Einheit des CRPS entspricht der Einheit der betrachteten Größe [Eum].

In dieser Arbeit wird die vorhergesagte Verteilung F oft als Normalverteilung modelliert. Für eine beliebige Normalverteilung lässt sich das Integral nach [Gne+04] analytisch lösen und beträgt

$$\mathrm{CRPS}(F_{\mu,\sigma}, y) = \sigma \left(\frac{y-\mu}{\sigma} \left(2\Phi \left(\frac{y-\mu}{\sigma} \right) - 1 \right) + 2\varphi \left(\frac{y-\mu}{\sigma} \right) - \frac{1}{\sqrt{\pi}} \right) \tag{3.6}$$

mit $F_{\mu,\sigma}$ als Normalverteilung $\mathcal{N}(\mu, \sigma^2)$ mit Erwartungswert μ und Standardabweichung σ, φ als Dichtefunktion der Standardnormalverteilung und Φ als Verteilungsfunktion der Standardnormalverteilung. Mithilfe dieser Darstellung lässt sich der Gradient des CRPS bilden, der zur numerischen Optimierung nützlich ist.

Um in praktischen Anwendungen Aussagen über die Prognosen eines Modells für einen kompletten Datensatz treffen zu können, wird der CRPS über alle Datenpunkte gemittelt. Für n Datenpunkte ergibt sich dann

$$\frac{1}{n} \sum_{i=1}^{n} \mathrm{CRPS}\left(F_i, y_i\right) \tag{3.7}$$

mit den probabilistischen Vorhersagen F_i des Modells und den zugehörigen Beobachtungen y_i.

Handelt es sich bei den Vorhersagen F_i um deterministische Vorhersagen, dann entspricht der CRPS dem mittleren absoluten Fehler. Dadurch kann der CRPS als Verallgemeinerung des mittleren absoluten Fehlers auf probabilistische Vorhersagen interpretiert werden.

Um die relative Verbesserung einer Vorhersage F gegenüber einer Referenzvorhersage F_{ref} ermitteln zu können, wird der Continuous Ranked Probability Skill Score (CRPSS)

betrachtet. Er berechnet sich zu

$$\mathrm{CRPSS}\,(F, y) = 1 - \frac{\mathrm{CRPS}\,(F, y)}{\mathrm{CRPS}\,(F_{\mathrm{ref}}, y)}. \tag{3.8}$$

Der CRPSS ist positiv orientiert, d. h. je größer der CRPSS-Wert, desto besser ist die Vorhersage F gegenüber der Referenzvorhersage F_{ref}. Ein negativer CRPSS weist auf eine Verschlechterung gegenüber der Referenz hin. Wenn es sich bei der zu untersuchenden Vorhersage F um die perfekte Vorhersage handelt, beträgt der CRPS 0 und damit der CRPSS 1. Dies ist der größte zu erreichende CRPSS-Wert. Der Fall, dass die Vorhersage F der Referenzvorhersage entspricht, resultiert in einem CRPSS von 0. Nach unten ist der CRPSS nicht beschränkt. In praktischen Anwendungen wird der CRPSS in der Regel aus gemittelten CRPS-Werten berechnet.

3.3. Ensemble Model Output Statistics

Ein Verfahren zur Nachbearbeitung probabilistischer Vorhersagen für Temperatur aus dem Bereich des Machine Learning ist Ensemble Model Output Statistics (EMOS, [Gne+04]). In den letzten Jahren wurde das Verfahren auch für die Vorhersage von anderen meteorologischen Variablen wie Windgeschwindigkeit [BL15] oder Niederschlag [SH15] weiterentwickelt. Im Folgenden wird zunächst das Verfahren für den Einsatz bei Temperaturvorhersagen erklärt und anschließend die für diese Arbeit verwendete Implementierung beschrieben.

3.3.1. Grundlagen

Das EMOS-Verfahren basiert auf der multiplen linearen Regression und benötigt in seiner Grundform lediglich die Vorhersagen der einzelnen Member eines Ensembles und die dazugehörigen Beobachtungen. Ziel des Verfahrens ist es, die Parameter der optimierenden Wahrscheinlichkeitsverteilung geeignet zu schätzen.

Bei Temperaturvorhersagen wird eine Normalverteilung zugrunde gelegt. Für eine Normalverteilung müssen zum einen der Erwartungswert μ und zum anderen die Varianz σ^2 als Parameter geschätzt werden. Der Erwartungswert wird als gewichteter Mittelwert der einzelnen Membervorhersagen X_i mit einem Korrekturfaktor a modelliert. Die Varianz ist als lineare Funktion der Ensemble-Varianz S^2 umgesetzt. Zusammengefasst ergibt sich die Wahrscheinlichkeitsverteilung der Vorhersage als

$$Y \sim \mathcal{N}\Big(a + \sum_{i=1}^{m} b_i X_i,\ \ c + dS^2\Big) \tag{3.9}$$

$$\sim \mathcal{N}\left(a + b_1 X_1 + \ldots + b_m X_m,\ \ c + dS^2\right). \tag{3.10}$$

Die (EMOS-)Koeffizienten $a, b_1, \ldots, b_m, c$ und d sind im Verfahren geeignet zu bestimmen.

Um möglichst gute Vorhersagen zu erreichen, sollen die EMOS-Koeffizienten so geschätzt werden, dass der CRPS minimiert wird. Dazu wird wie in [Gne+04] beschrieben die Parametrisierung der Normalverteilung nach Gleichung (3.10) in den CRPS aus Gleichung (3.6) eingesetzt, wie in Gleichung (3.7) über die Trainingsdaten gemittelt und als Funktion der EMOS-Koeffizienten aufgefasst. Man erhält damit als Verlustfunktion

$$J(a, b_1, \ldots, b_m, c, d) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{CRPS}(F_{\mu_i,\sigma_i}, y_i) \tag{3.11}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \sqrt{c + dS_i^2} \left(Z_i \Big(2\Phi(Z_i) - 1 \Big) + 2\varphi(Z_i) - \frac{1}{\sqrt{\pi}} \right) \tag{3.12}$$

$$\text{mit } Z_i = \frac{y_i - (a + b_1 X_{i1} + \ldots + b_m X_{im})}{\sqrt{c + dS_i^2}} \tag{3.13}$$

als standardisierten Vorhersagefehler. Zur Minimierung dieser Verlustfunktion kann auf übliche numerische Verfahren zur Optimierung zurückgegriffen werden. So eignet sich beispielsweise das Broyden-Fletcher-Goldfarb-Shanno-Verfahren (kurz BFGS-Verfahren, [Bro70]) als Quasi-Newton-Verfahren häufig zur Berechnung der EMOS-Koeffizienten.

Bei diesem Verfahren wird nach [RL18] in der Regel ein Trainingsdatensatz benutzt, der lediglich wenige Tage aus der direkten Vergangenheit des vorherzusagenden Tages beinhaltet. Es ist jedoch ebenfalls möglich, einen Datensatz zu verwenden, der einen größeren Zeitraum enthält. Außerdem wird häufig für jede Messstation eine eigene, lokale Instanz des EMOS-Verfahrens durchgeführt. Damit das Verfahren später an beliebigen Stellen des Modellgitters ausgewertet werden kann, wird in dieser Arbeit allerdings die globale Variante des Verfahrens bevorzugt. Im Trainingsdatensatz befinden sich dann nicht nur Daten zu einer einzigen Messstation, sondern zu allen vorhandenen Messstationen.

Neben der bisher erklärten Grundform des Verfahrens gibt es noch eine vereinfachte Variante, wie sie in [RL18] eingesetzt wurde. Statt auf die einzelnen Vorhersagen der Ensemblemember wird dabei nur auf das arithmetische Mittel $\overline{X}$ der Membervorhersagen (Ensemblemittel) zurückgegriffen. Der Erwartungswert der Verteilung wird dann als lineare Funktion des Ensemblemittels modelliert. Man erhält damit

$$Y \sim \mathcal{N}\Big(a + b\overline{X},\ c + dS^2 \Big) \tag{3.14}$$

mit den zu schätzenden Koeffizienten a, b, c und d. Die Verlustfunktion kann analog zu Gleichung (3.12) gebildet werden. In diesem Fall ergibt sich jedoch entsprechend

$$Z_i = \frac{y_i - (a + b\overline{X}_i)}{\sqrt{c + dS_i^2}} \tag{3.15}$$

für den standardisierten Vorhersagefehler.

In der hier beschriebenen Form nutzt das EMOS-Verfahren nur die Temperatur als Prädiktorvariable. Die Hinzunahme von weiteren Prädiktoren ist nicht trivial, da das Verfahren durch die dabei erhöhte Parameteranzahl schnell zu Overfitting neigt. Daher wurden spezielle Boosting-Varianten des Verfahrens entwickelt, die die Verwendung von mehreren Features ermöglichen [MMZ17].

3.3.2. Implementierung

In bisherigen Arbeiten zur Ensemble-Kalibrierung wie [Gne+04] und [RL18] wurde zur Implementierung des EMOS-Verfahrens die Programmiersprache R eingesetzt. Im Comprehensive R Archive Network (CRAN, [Hor12]) wird das Paket `ensembleMOS` [Yue+18] bereitgestellt, das Implementierungen des EMOS-Verfahrens für verschiedene Wahrscheinlichkeitsverteilungen enthält. Da in dieser Arbeit jedoch hauptsächlich Python als Programmiersprache zum Einsatz kommt, wäre ein entsprechendes Python-Modul zu bevorzugen. Das EMOS-Verfahren liegt jedoch nicht in einer Python-Implementierung vor.

Die Datenvorbereitungsroutinen wurden bereits in Python umgesetzt. Damit die vorbereiteten Daten möglichst effizient in R zur Verfügung gestellt werden können, wurde zunächst das Python-Modul `rpy2` [Gau+16] eingesetzt, um R von Python aus anzusteuern. Dieses Modul liegt jedoch noch nicht auf dem für diese Arbeit verwendeten System zur Verfügung. Um unnötige Installationen zu vermeiden, wurden vorher verschiedene Implementierungen auf einem Desktop-Rechner verglichen. Neben der verfügbaren Implementierung im R-Paket `ensembleMOS` wurden zwei eigene Implementierungen des EMOS-Verfahrens in R und Python umgesetzt. Diese drei Implementierungen wurden schließlich mit künstlich erzeugten Datensätzen verschiedener Größen hinsichtlich ihrer Laufzeit untersucht. Es wurde die Anzahl der Samples und die Anzahl der Member variiert. In Abb. 3.5 ist zu erkennen, dass die Python-Implementierung performanter als die anderen beiden Implementierungen ist. Daher wurde in dieser Arbeit die eigene

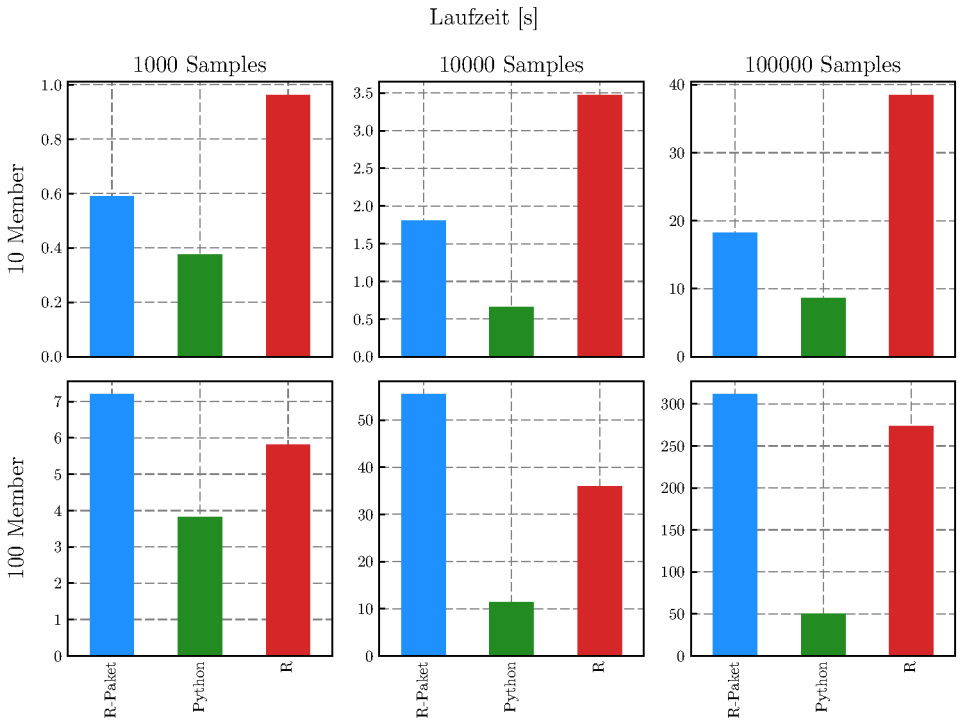

Abbildung 3.5.: Laufzeitvergleich der EMOS-Implementierungen

Python-Implementierung genutzt.

Zur Optimierung der Verlustfunktion aus Gleichung (3.12) wurde das im Python-Modul `scipy` [JOP+01] implementierte BFGS-Verfahren verwendet. Bei Durchführung des EMOS-Verfahrens ist es wichtig, dass die EMOS-Parameter der Varianz positiv geschätzt werden, da sonst negative Varianzen auftreten können. Für den Parameter c ist dies bei der Wahl geeigneter Startwerte kein Problem. Um in der Implementierung einen positiven Parameter für d zu erhalten, wurde wie in [Gne+04] der Parameter d durch δ^2 substituiert. In der Implementierung wird δ optimiert, jedoch wird als Ergebnis für den Parameter d schließlich δ^2 zurückgegeben. Durch das Quadrieren bleibt der Wert $d = \delta^2$ stets positiv.

In der vereinfachten Variante des EMOS-Verfahrens, in welcher der Erwartungswert gemäß Gleichung (3.14) als lineare Funktion des Ensemblemittels geschätzt wird und das im Folgenden mit `EMOS-mean` bezeichnet wird, ist die Wahl günstiger Startwerte noch kein Problem. Hier wurde a mit dem Ensemblemittel und c mit der Ensemblevarianz der Trainingsdaten initialisiert. Die restlichen Parameter sind gleichverteilte Pseudozufallszahlen zwischen 0 und 1. Bei dem Verfahren, das gemäß Gleichung (3.10) alle Membervorhersagen verarbeitet, im Folgenden `EMOS` genannt, ist stärker von der Initialisierung der Parameter abhängig, da nun die einzelnen Membervorhersagen geeignet zu gewichten sind. Um zu gewährleisten, dass `EMOS` auf den Trainingsdaten mindestens so gut wie `EMOS-mean` ist, werden die bereits optimierten Parameter von `EMOS-mean` zur Initialisierung der Startparameter von `EMOS` genutzt. Die Parameter können größtenteils direkt übernommen werden. Aus Gleichung (3.9) und Gleichung (3.14) ergibt sich

$$\mu = a + b\overline{X} = a + \frac{b}{m}\sum_{i=1}^{m} X_i \stackrel{!}{=} a + \sum_{i=1}^{m} b_i X_i \tag{3.16}$$

$$\Leftrightarrow b_i = \frac{b}{m} \tag{3.17}$$

mit den Parametern b_i für `EMOS`, dem optimierten Parameter b von `EMOS-mean` und m als Anzahl der eingesetzten Member. Die Gewichtungen b_i der Membervorhersagen müssen demnach noch zusätzlich durch die Anzahl der Member geteilt werden, um so die Mittelwertbildung des `EMOS-mean` zu imitieren.

3.4. Quantile Regression Forests

Bei der Regression von Parametern einer Wahrscheinlichkeitsverteilung wie beim EMOS-Verfahren ist die Wahl einer geeigneten Verteilung entscheidend. Die Temperaturverteilung lässt sich gut durch eine Normalverteilung approximieren. Für andere Variablen wie Windgeschwindigkeit oder Niederschlag kann die Wahl einer Verteilung deutlich schwieriger ausfallen [BL18]. Durch eine alternative Vorgehensweise kann diese Problematik umgangen werden. Bei der Quantilregression wird die bedingte Verteilung approximiert, indem ihre Quantile geschätzt werden. Dabei müssen keine Annahmen über die zugrundeliegende Verteilung getroffen werden. Ein aktuell sehr erfolgreiches Verfahren zur

Ensemble-Kalibrierung in diesem Kontext ist Quantile Regression Forest (QRF) nach [Mei06]. Es basiert auf Random Forest, bei dem Entscheidungsbäume in einem Ensemble zusammengefasst werden. Das Verfahren wurde im Rahmen dieser Bachelorarbeit nicht durchgeführt, da die Performanz bestehender Python-Implementierungen bei der genutzten Problemgröße nicht ausreicht, um mit den anderen Verfahren dieser Arbeit mitzuhalten.

In Entscheidungsbäumen können Entscheidungsregeln dargestellt und modelliert werden. Sie sind verständlich und einfach zu interpretieren. Im Kontext einer Regression wird von einem Regressionsbaum gesprochen. Die Daten werden im Fall von CART (Classification and Regression Trees) nach [Bre+84] an jedem Knoten anhand eines Schwellenwerts bezüglich eines Features in zwei Gruppen aufgeteilt. Der Aufbau eines Baumes geschieht durch rekursives binäres Aufteilen des Trainingsdatensatzes. Dazu wird an jedem Knoten jeweils die bestmögliche Aufteilung gewählt, zu der die Summe der Varianzen der Labels in den Kindknoten minimiert wird. Diese Aufteilung wird an den Kindknoten fortgeführt, bis eine Abbruchbedingung erfüllt ist. Typische Abbruchbedingungen sind das Erreichen einer maximalen Baumtiefe oder das Unterschreiten einer Mindestanzahl an Samples in einem Knoten. An den Blättern eines Baumes befinden sich am Ende Teilmengen der Labels, die im Fall einer Regression auf ihr arithmetisches Mittel reduziert werden. Zum Erstellen einer Vorhersage wird der Baum mit den Eingabedaten gemäß der Entscheidungsregeln durchlaufen, bis ein Blatt erreicht wird. Der im Blatt gespeicherte Mittelwert bildet die Vorhersage. Ein Entscheidungs- bzw. Regressionsbaum hat oft eine hohe Varianz. Kleine Änderungen in den Daten führen häufig zu einer grundlegend anderen Struktur des zugehörigen Baumes.

Bei Random Forest nach [Bre01] wird eine höhere Stabilität der Vorhersagen von Bäumen erreicht, indem über mehrere Bäume gemittelt wird. Es handelt sich um eine Ensemble-Technik (Abschnitt 3.1.3). Die Unabhängigkeit der einzelnen Bäume wird erreicht, indem beim Aufbau jedes Baumes an jedem Knoten zufällig eine Menge von Features gewählt wird, die bei dem Aufteilungsschritt in Betracht gezogen werden dürfen. Wie beim Bagging werden hier zusätzlich für jeden Baum Bootstrap-Stichproben des Trainingsdatensatzes gezogen. Mit Random Forests können nichtlineare Beziehungen dargestellt und verschiedene Features eingesetzt werden. Gleichzeitig bietet das Verfahren einen schnellen Trainingsprozess, kann gut parallelisiert werden und ist robust gegenüber Overfitting.

Random Forest kann als Quantile Regression Forest auf eine Quantilregression ausgeweitet werden. Dazu werden die Bäume nach demselben Prinzip aufgebaut wie bei Random Forest. Die empirische Verteilung der in den Blättern eines Baumes enthaltenen Teilmenge der Labels bietet eine Schätzung der wahren anzunähernden Verteilung. Die Quantile der Verteilung werden über die Quantile der über alle Bäume kombinierten empirischen Verteilung der Labels geschätzt. Dazu müssen im Gegensatz zu Random Forest nicht nur die Mittelwerte in den Blättern gespeichert werden, sondern alle in den Blättern befindlichen Labels.

Quantile Regression Forests wurden in [Tai+16] und [RL18] für die Ensemble-Kalibrierung eingesetzt. Nachteilig dabei ist, dass die vorhergesagten Quantile nach Konzeption des Algorithmus nur im Bereich der im Trainingsdatensatz vorhandenen Labels liegen

können. Daher können z. B. bei Temperaturvorhersagen durch Unterschiede zwischen den Jahreszeiten schlechte Vorhersagen bei kurzen Trainingszeiträumen entstehen. Eine Weiterführung des Verfahrens in [Tai+17] verbindet die Quantilregression mit der Regression von Parametern einer Wahrscheinlichkeitsverteilung, indem an jedem Blatt eine parametrische Verteilungsfunktion an die Daten angepasst wird statt auf die empirische Verteilung zurückzugreifen.

3.5. Künstliche neuronale Netze

Künstliche neuronale Netze (kurz neuronale Netze, KNN) sind zurzeit ein beliebter Forschungsgegenstand des Machine Learning. Sie werden häufig zur Mustererkennung (z. B. Sprach- oder Schrifterkennung), zur Fehlererkennung oder bei Zeitreihenanalysen eingesetzt. In den letzten Jahren wurden viele Netzstrukturen erforscht, die für den Einsatz bei künstlicher Intelligenz entstanden sind.

In dieser Arbeit werden neuronale Netze zur Kalibrierung des vorliegenden Ensembles eingesetzt. In [RL18] wurde dies bereits mit Feedforward-Netzen erfolgreich durchgeführt. Bei der Untersuchung räumlicher Muster in den Modelldaten werden in dieser Arbeit neben Feedforward-Netzen zusätzlich Faltungsnetze eingesetzt.

Im Folgenden werden die Grundlagen zu den für diese Bachelorarbeit relevanten neuronalen Netzen erklärt. Außerdem wird ihr Einsatz bei der Kalibrierung von Ensembles beschrieben. Darüber hinaus werden die Implementierungen der Netze dargelegt und aufgetretene Probleme diskutiert und untersucht.

3.5.1. Grundlagen

Neuronale Netze sollen funktionale Zusammenhänge zwischen der Eingabemenge $\mathbb{X}$ und den Labels $\mathbb{Y}$ erlernen. Die Verarbeitung einer Eingabe geschieht dabei mittels linearer affiner Transformationen sowie Anwendung skalarer nichtlinearer Funktionen. Um beim Training gute Generalisierung zu erzielen, müssen die freien Parameter ϑ von neuronalen Netzen entsprechend angepasst werden. In diesem Kapitel werden die mathematischen Hintergründe dazu vorgestellt. Die Inhalte sind in [GBC16] und [Bis06] beschrieben.

3.5.1.1. Feedforward-Netze

Ein Feedforward-Netz (auch mehrlagiges Perzeptron genannt) besteht aus künstlichen Neuronen, die miteinander verbunden sind. Das künstliche Neuron ist dem biologischen Vorbild der Nervenzelle nachempfunden und verarbeitet mehrere Eingangssignale zu einem Ausgangssignal. Über die Verbindungen auf weitere Neuronen wird das Ausgangssignal eines Neurons auf andere Neuronen übertragen.

Die Signalverarbeitung der Neuronen wird mit einem sogenannten einfachen Perzeptron modelliert. Dort kann ein Neuron eine Aktivierung in Form einer reellen Zahl annehmen. Die Verbindungen zwischen den Neuronen sind unterschiedlich gewichtet und legen somit fest, wie sich die Aktivierung auf die folgenden Neuronen überträgt. Die

Berechnung einer Aktivierung wird in Abb. 3.6 gezeigt. Zunächst werden die Aktivierungen x_i der vorigen Neuronen mit den Gewichtungen w_i der zugehörigen Verbindungen multipliziert und anschließend aufsummiert. Außerdem wird ein Schwellenwert b (auch Bias genannt) dazuaddiert. Zuletzt wird eine Aktivierungsfunktion f auf die Summe angewendet. Damit ergibt sich insgesamt als Aktivierung h eines Neurons

$$h = f\left(\sum_{i=1}^{n} w_i x_i + b\right). \tag{3.18}$$

Die Aktivierungsfunktion ist typischerweise eine nichtlineare Funktion $f : \mathbb{R} \to \mathbb{R}$. Für diese Arbeit wird ausschließlich die Rectified-Linear-Unit-Aktivierungsfunktion (kurz ReLU)

$$f(z) = \max\{0, z\} \tag{3.19}$$

verwendet. Sie ist durch ihre einfache Ableitung[1] gut für den Einsatz bei neuronalen Netzen geeignet.

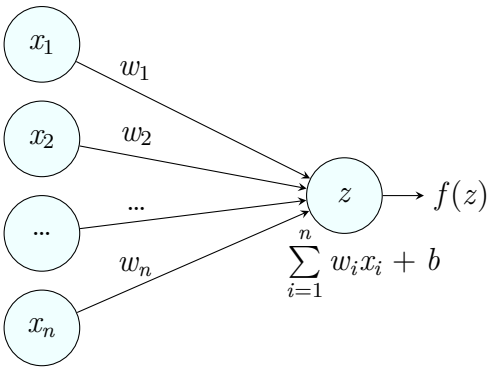

Abbildung 3.6.: Berechnung der Aktivierung eines Neurons

Bei Feedforward-Netzen liegt eine schichtweise Anordnung der Neuronen vor. Bei Vollvernetzung ist jedes Neuron einer Schicht mit jedem Neuron der folgenden Schicht verbunden. Die Eingabe wird somit nur in eine Richtung verarbeitet. Falls auch Verbindungen von Neuronen einer Schicht untereinander oder zu vorangegangenen Schichten existieren, wird von rekurrenten neuronalen Netze gesprochen. Diese sind für diese Arbeit nicht relevant. Sie werden in [GBC16] näher beschrieben. Am Anfang eines Feedforward-Netzes befindet sich eine Eingabeschicht. Sie enthält für jedes Feature genau ein Neuron. Am Ende des Netzes ist eine Ausgabeschicht vorzufinden. Die Anzahl der Neuronen in der Ausgabeschicht hängt von dem Problem ab, welches das neuronale Netz lösen soll. Oftmals entspricht sie der Dimensionalität der Labels, um Vorhersagen direkt mit den Labels vergleichen zu können. Zwischen der Eingabe- und der Ausgabeschicht können sich beliebig viele zusätzliche Schichten befinden. Diese werden versteckte Schichten

[1] aus numerischen Gründen ist die fehlende Differenzierbarkeit bei $z = 0$ vernachlässigbar

genannt, da sie außerhalb des neuronalen Netzes nicht sichtbar sind. Falls mehrere versteckte Schichten vorhanden sind, handelt es sich um ein tiefes neuronales Netz.

Die Berechnung einer Vorhersage eines Feedforward-Netzes kann mittels Matrizen und Vektoren dargestellt werden. Dazu werden die Aktivierungen der n Neuronen in einer Schicht als Vektor $h \in \mathbb{R}^n$ modelliert. Die Gewichtungen der Verbindungen zwischen einer Schicht mit n Neuronen und einer mit m Neuronen werden in einer Matrix $W \in \mathbb{R}^{m \times n}$ zusammengetragen. Dort entspricht der Eintrag w_{ij} dem Gewicht der Verbindung von Neuron j der einen Schicht zu Neuron i der nächsten Schicht. Bei dem Matrix-Vektor-Produkt $W \cdot h$ werden somit automatisch die Aktivierungen mit den richtigen Gewichtungen multipliziert und für die einzelnen Neuronen der nächsten Schicht aufsummiert. Die Bias-Werte können ebenfalls in einem Vektor $b \in \mathbb{R}^m$ dargestellt und zu den Aktivierungen hinzuaddiert werden. Abschließend wird die Aktivierungsfunktion elementweise auf den resultierenden Vektor angewendet. Für den Übergang von der i-ten in die $i+1$-te Schicht ergibt sich insgesamt

$$h_{i+1} = f_i \left(W_i h_i + b_i \right). \tag{3.20}$$

Dieser Übergang kann als Funktion $S_i \left(h_i \right)$ definiert werden. Dann lautet die Berechnungsvorschrift für die Vorhersage $\hat{y} = h_n$ zu einer Eingabe $x = h_0$ eines Feedforward-Netzes mit n Schichten

$$\hat{y} = h_n = S_{n-1} \left(\ldots S_1 \left(S_0 \left(x \right) \right) \ldots \right). \tag{3.21}$$

Feedforward-Netze wurden erstmals in [RL18] für die Ensemble-Kalibrierung meteorologischer Modelle zur Vorhersage von Temperatur eingesetzt. Bei der Kalibrierung von Ensembles sind die Ausgaben des neuronalen Netzes Wahrscheinlichkeitsverteilungen für probabilistische Vorhersagen. Da Temperaturvorhersagen als Normalverteilung modelliert werden können, müssen wie auch beim EMOS-Verfahren in Abschnitt 3.3 die Parameter der Normalverteilung geschätzt werden. Daher sind in der Ausgabeschicht zwei Neuronen vorgesehen. Dies ist in Abb. 3.7 veranschaulicht. Das erste dient als Schätzung für den Erwartungswert μ und das zweite für die Standardabweichung σ. Durch Hinzunahme des Labels als Beobachtung kann damit der CRPS (Abschnitt 3.2) als Verlustfunktion $J(\vartheta)$ eingesetzt und minimiert werden. Als Eingabefeatures des Netzes dienen verschiedene Variablen aus den zugrundeliegenden Modellen. Da zu den Modellvariablen für jedes Ensemblemember eigene Vorhersagen vorliegen, werden diese jeweils in [RL18] auf das Ensemblemittel und die Standardabweichung reduziert. So ergeben sich z. B. aus den Vorhersagen der Temperatur `T2` demnach der Mittelwert `T2_mean` und die Standardabweichung `T2_std` als Features für die neuronalen Netze.

3.5.1.2. Optimierung

Um eine Eingabe bezüglich einer Verlustfunktion $J(\vartheta)$ möglichst gut auf ihr Label abzubilden, müssen die freien Parameter und Gewichte im neuronalen Netz angepasst werden. Die gesamten Parameter eines neuronalen Netzes werden mit der Variablen ϑ ausgedrückt. Im Fall von Feedforward-Netzen handelt es sich um die Gewichtsmatrizen

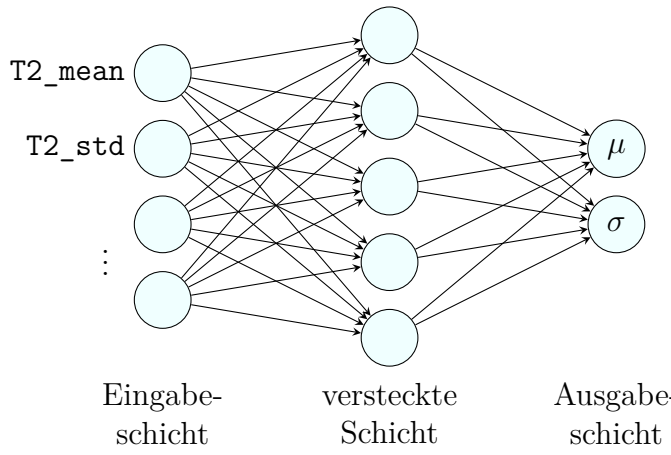

Abbildung 3.7.: Beispielhafte Struktur eines Feedforward-Netzes

W_i und Biasvektoren b_i. Verschiedene neuronale Netze unterscheiden sich jedoch wegen ihrer Struktur in der Anzahl ihrer Parameter. Je nach Ausprägung der Parameter ϑ stellt das neuronale Netz eine andere Wahrscheinlichkeitsverteilung $p_\vartheta(y \mid x)$ dar (vgl. Abschnitt 3.1).

Im Trainingsprozess werden die Schätzungen des neuronalen Netzes zu den Trainingsdaten ausgerechnet. Durch Hinzunahme der Labels kann dann der In-Sample-Fehler E_{in} auf den Trainingsdaten ausgerechnet werden. Dieser wird anschließend auf die Gewichte im neuronalen Netz zurückgeführt. Durch eine gezielte Anpassung der Parameter kann der Fehler des Netzes verringert werden. Ziel des Trainingsprozesses ist es, den In-Sample-Fehler E_{in} bezüglich der genutzten Verlustfunktion zu minimieren. Dies wird mithilfe eines Gradientenabstiegsverfahrens erreicht.

Für einen Gradientenabstieg muss pro Schritt der Gradient der Kostenfunktion $\nabla_\vartheta J(\vartheta)$ zur aktuellen Belegung der Parameter berechnet werden. In einem Abstiegsschritt werden die Parameter dann genau in Gegenrichtung dieses Gradienten angepasst, also

$$\vartheta_{i+1} = \vartheta_i - \eta \nabla_\vartheta J(\vartheta_i). \tag{3.22}$$

Die Lernrate $\eta \in \mathbb{R}^+$ beschreibt die Schrittweite eines Abstiegsschritts und bildet einen Hyperparameter beim Training von neuronalen Netzen. In der Regel wird sie vor dem Training festgelegt und im Laufe des Trainings verkleinert. Dies soll verhindern, dass Minima durch eine zu große Schrittweite übersprungen werden. Eine zu geringe Schrittweite würde den Trainingsprozess verlängern, da das Auffinden eines Minimums dann nur in kleinen Schritten möglich ist.

Da die Berechnung eines Abstiegsschritts auf dem gesamten Trainingsdatensatz zu aufwändig wäre, wird der Trainingsdatensatz in der Praxis zufällig in kleine, gleich große Gruppen, sogenannte Batches, unterteilt. Die Abstiegsrichtung wird dann jeweils nur auf Grundlage eines Batches ermittelt. Das Vollziehen eines solchen Abstiegsschritts heißt Iteration. Sobald alle Batches des Trainingsdatensatzes verarbeitet wurden, handelt es sich um eine Epoche. Die Gradienten, die auf Grundlage eines Batches ausgerechnet

wurden, bilden vor dem Hintergrund des gesamten Datensatzes lediglich eine Schätzung des echten Gradienten. Daher wird bei diesem Verfahren vom stochastischen Gradientenabstieg gesprochen. Die Größe der Batches ist wiederum ein weiterer Hyperparameter eines neuronalen Netzes. Es hat sich gezeigt, dass das stochastische Gradientenabstiegsverfahren zu einer besseren Generalisierung neuronaler Netze führt.

Da die Gradienten nur mithilfe von Batches geschätzt werden, kann sich der Fehler nach Durchführung eines Abstiegsschritts erhöhen und die Richtungen der Gradienten können stark variieren. Um dies zu vermeiden, wird häufig ein Trägheitsterm (Momentum) eingeführt. Zur Bestimmung der Abstiegsrichtung wird dann nicht nur der aktuelle Gradient in Betracht gezogen. Es werden auch die Gradienten der früheren Abstiegsschritte berücksichtigt. Vereinzelte, stark abweichende Gradienten haben somit einen geringen Einfluss auf den Trainingsprozess.

Die Lernrate η wirkt sich signifikant auf den Lernprozess aus. Daher ist es nicht leicht, sie vor dem Training geeignet festzulegen. Zudem ist der Einfluss jedes einzelnen Parameters aus ϑ auf die Verlustfunktion unterschiedlich stark. Adaptive Verfahren, die jedem Parameter eine eigene Lernrate bereitstellen und sie während des Trainings anpassen, sind demnach effizienter als der stochastische Gradientenabstieg aus Gleichung (3.22). Einige bekannte adaptive Optimierungsalgorithmen werden in [GBC16] beschrieben. Der AdaGrad-Algorithmus (von engl. *adaptive gradients*, [DHS11]) summiert alle quadrierten Gradienten im Laufe des Trainings auf, um die Lernrate schließlich antiproportional zur Quadratwurzel aus dieser Summe zu skalieren. Da die quadrierten Gradienten von Beginn an aufsummiert werden, wird die effektive Lernrate oftmals frühzeitig sehr klein. Dem AdaGrad-Algorithmus sehr ähnlich ist der RMSProp-Algorithmus (RMS von engl. *root mean square*, [Hin12]). Bei diesem werden allerdings die vorangegangenen quadrierten Gradienten nicht summiert, sondern exponentiell absteigend gemittelt. Dadurch ist der Algorithmus in der Lage Minima zu finden, die der AdaGrad-Algorithmus wegen seiner kleinen Lernrate gegen Ende des Trainings nicht gefunden hätte. Beim Adam-Algorithmus (von engl. *adaptive moment*, [KB14]) fließen zusätzlich Schätzungen des ersten und zweiten Moments des Gradienten zur Bestimmung der Abstiegsrichtungen und Schrittweiten ein. Er ist sehr robust bezüglich der Wahl seiner Hyperparameter. Trotzdem muss je nach Anwendungsfall die Lernrate angepasst werden. In dieser Arbeit wurde der Adam-Algorithmus beim Training der neuronalen Netze verwendet.

Um den Trainingsprozess von neuronalen Netzen besser untersuchen zu können, werden die Lernkurven der Netze betrachtet. Mit ihnen wird der Verlauf des Fehlers auf den Trainings- sowie Validierungsdaten in Abhängigkeit der Epoche in einem Plot dargestellt. Es sollte ein über die Epochen fallender Trainingsfehler zu erkennen sein. Im Optimalfall fällt auch der Fehler auf den Validierungsdaten zunächst und flacht zunehmend ab, je mehr das Netz gesättigt ist. Ist der Fehler auf den Validierungsdaten etwa so groß wie auf den Trainingsdaten, ist eine gute Generalisierung zu beobachten. Steigt jedoch der Validierungsfehler wieder an, so weist dies auf Overfitting hin.

3.5.1.3. Backpropagation-Algorithmus

Für die Durchführung eines Abstiegsschritts für den Gradientenabstieg, muss der Gradient der Verlustfunktion $\nabla_\vartheta J(\vartheta)$ berechnet werden. Der dafür genutzte Algorithmus heißt Backpropagation-Algorithmus [RHW86]. Der Gradient für die Parameter in ϑ kann analytisch berechnet werden. Die numerische Auswertung dieses Gradienten ist jedoch sehr rechenintensiv, da sich darin viele Unterausdrücke wiederholen. Je nach Implementierung können diese entweder mehrfach ausgerechnet werden, was einen erhöhten Rechen- und Zeitaufwand bedeutet, oder zwischengespeichert werden, was nur bei größerem Speicherbedarf möglich ist. Der Backpropagation-Algorithmus liefert eine sehr effiziente Berechnungsweise, die an das Prinzip der dynamischen Programmierung angelehnt ist. Das Einsatzgebiet dieses Algorithmus ist nicht nur auf das Training von neuronalen Netzen beschränkt.

Die Berechnung der Vorhersage eines neuronalen Netzes lässt sich in einem Berechnungsgraph darstellen, sodass jede einzelne Operation, die nacheinander auf die Eingabe des Netzes angewendet wird, als eigenständige Funktion betrachtet werden kann. Für ein Perzeptron ist dies in Abb. 3.8 zu sehen. Die Berechnung einer Vorhersage lässt sich demnach als Verkettung vieler einfacher Funktionen darstellen. Daher können die partiellen Ableitungen, die zur Berechnung des Gradienten benötigt werden, mithilfe der Kettenregel ausgerechnet werden. Für eine Verkettung $f \circ g$ mit $f : \mathbb{R}^n \to \mathbb{R},\ y \mapsto z$ und $g : \mathbb{R}^m \to \mathbb{R}^n,\ x \mapsto y$ gilt gemäß Kettenregel

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}. \tag{3.23}$$

Umgeschrieben in Vektorschreibweise ergibt sich

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^\top \nabla_y z \tag{3.24}$$

mit $\frac{\partial y}{\partial x} \in \mathbb{R}^{n \times m}$ als Jacobi-Matrix von g. Damit wird deutlich, dass sich der Gradient von z bezüglich x mithilfe des Matrix-Vektor-Produkts von der Jacobi-Matrix $\frac{\partial y}{\partial x}$ und dem Gradienten $\nabla_y z$ berechnen lässt. Der Backpropagation-Algorithmus besteht im Grunde aus der Durchführung solcher Matrix-Vektor-Produkte. Obwohl die Parameter in ϑ beliebige Dimensionen haben können, reicht die obige Kettenregel aus. Matrizen und Tensoren können nämlich in einen Vektor umgeformt werden, indem die Elemente neu

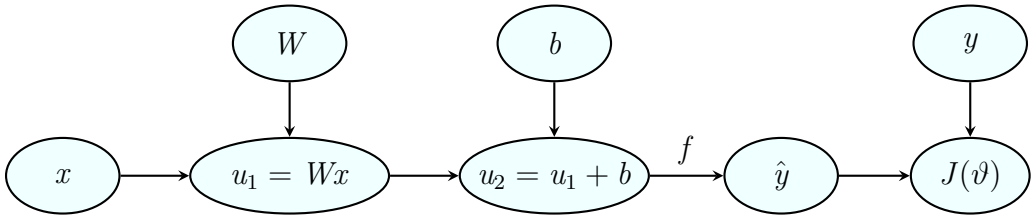

Abbildung 3.8.: Berechnungsgraph eines Perzeptrons

angeordnet werden. So lässt sich eine $n \times m$-Matrix z. B. zeilenweise in einen Vektor der Länge $n \cdot m$ überführen.

Um die Vorgehensweise des Backpropagation-Algorithmus zu beschreiben, wird ein Berechnungsgraph betrachtet, bei dem jede skalare Variable durch einen eigenen Knoten dargestellt wird. Demnach wird jedes einzelne Element einer Matrix oder eines Vektors mit einem eigenen Knoten beschrieben. Auch die Parameter aus ϑ sind im Berechnungsgraph enthalten. Die Indizierung der einzelnen Knoten $u^{(i)}$ ist so gewählt, dass durch die sequentielle Auswertung von $u^{(1)}$ bis $u^{(n)} = J(\vartheta)$ das gesamte Vorwärts-Ausrechnen des neuronalen Netzes inklusive der Anwendung der Verlustfunktion möglich ist. Der Algorithmus startet nun beim letzten Knoten $u^{(n)}$ des Graphen und berechnet die Ableitungen $\frac{\partial J}{\partial u^{(i)}}$ mittels Kettenregel in umgekehrter Reihenfolge bis zum ersten Knoten $u^{(1)}$. Durch die geschickte Indizierung der Knoten können dabei Zwischenergebnisse gespeichert und bei Anwendung der Kettenregel wiederverwendet werden. Eine vereinfachte Beschreibung des Algorithmus nach [GBC16] ist in Abb. 3.9 dargestellt. Dort wird stets die Ableitung $\frac{\partial J}{\partial u^{(i)}}$ in grad$[i]$ zwischengespeichert. Nach vollständiger Ausführung des Algorithmus liegt somit auch der benötigte Gradient $\nabla_\vartheta J(\vartheta)$ vor.

grad$[n] \leftarrow 1$
für jedes i von $n-1$ bis 1
$M \leftarrow \left\{ j \mid u^{(i)} \text{ ist Vorgänger von } u^{(j)} \right\}$
grad$[i] \leftarrow \sum_{j \in M} \text{grad}[j] \cdot \frac{\partial u^{(j)}}{\partial u^{(i)}}$

Abbildung 3.9.: Vereinfachte Darstellung des Backpropagation-Algorithmus in einem Struktogramm

In der Praxis repräsentieren die Knoten im Berechnungsgraph keine skalaren Variablen, sondern die im neuronalen Netz verarbeiteten Vektoren, Matrizen und Tensoren. Dann kann der Gradient von $u^{(n)} = J(\vartheta)$ zu jedem vorangegangenen Knoten $u^{(i)}$ durch Betrachtung des direkten Nachfolgers $u^{(j)}$ von $u^{(i)}$ einfach berechnet werden. Dazu muss lediglich der Gradient $\nabla_{u^{(j)}} u^{(n)}$, der aus einem früheren Schritt des Algorithmus bereits bekannt ist, mit der Jacobi-Matrix der Operation, die $u^{(j)}$ aus $u^{(i)}$ erzeugt hat, multipliziert werden. Auf diese Weise werden für jede vorangegangene Operation von $u^{(n)}$ die jeweiligen Jacobi-Matrizen aufmultipliziert, bis alle Teile des benötigten Gradienten $\nabla_\vartheta J(\vartheta)$ bestimmt worden sind. Sollte $u^{(n)}$ von $u^{(i)}$ aus über mehrere Pfade erreicht werden können, werden die Gradienten der bei $u^{(i)}$ ankommenden Pfade gemäß Kettenregel aufsummiert.

Viele Bibliotheken wie auch das in dieser Arbeit verwendete `tensorflow` [Aba+15] fügen beim Durchlaufen des Backpropagation-Algorithmus dem Berechnungsgraph zusätzliche Knoten hinzu, welche die einzelnen Ableitungen der Knoten symbolisch berechnen (vgl. Abb. 3.10). Das Ausrechnen der Gradienten mit konkreten Zahlenwerten kann dann zu einem späteren Zeitpunkt erfolgen. Vorteilhaft an diesem Prinzip ist, dass eine wiederholte Durchführung des Backpropagation-Algorithmus auf dem Graph die Berechnung höherer Ableitungen ermöglicht. Um die Ableitungen als Knoten in den

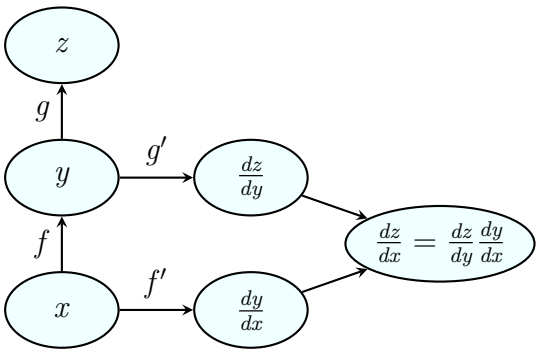

Abbildung 3.10.: Beispiel eines Berechnungsgraphen mit Ableitungen im skalaren Fall

Graph einfügen zu können, enthalten Bibliotheken zu jeder Operation, die auf den Tensoren durchgeführt werden kann, auch Methoden, die das Produkt aus Jacobi-Matrix und Vektor zu den jeweiligen Operationen gemäß Kettenregel vollziehen. Auf diese Weise muss für den Backpropagation-Algorithmus selbst zu keinem Zeitpunkt eine konkrete Ableitung bekannt sein. Der Algorithmus muss lediglich die entsprechenden Methoden der einzelnen Operationen aufrufen.

3.5.1.4. Faltungsnetze

Ein Faltungsnetz (engl. *convolutional neural network*, kurz CNN, [LeC+89]) enthält neben vollvernetzten Schichten (wie in Abschnitt 3.5.1.1 beschrieben) sogenannte Faltungsschichten (engl. *convolutional layers*). In einer solchen Schicht wird eine Faltung durchgeführt. In dieser Arbeit ist bei der Faltung nur der diskrete zweidimensionale Fall relevant. Neben der Eingabematrix A wird zur Faltung ein meist kleinerer und quadratischer Kernel $K \in \mathbb{R}^{n \times n}$ mit ungeradem n und Mittelpunkt k_{mm} benötigt. Ein Element der Ergebnismatrix B ergibt sich dann zu

$$B(i,j) = (K * A)(i,j) = \sum_{k=1}^{n} \sum_{l=1}^{n} A(i+m-k, j+m-l) K(k,l). \tag{3.25}$$

Dabei bilden A, B und K jeweils auf ihre Matrixelemente ab und sind an ungültigen Indizes 0. In einer Faltungsschicht wird anschließend eine Aktivierungsfunktion elementweise auf die Ergebnismatrix B angewendet. Der Übergang einer Faltungsschicht lässt sich demnach zu

$$H_{i+1} = f_i\left(K_i * H_i\right) \tag{3.26}$$

mit den Aktivierungen H als Matrix und dem Kernel K berechnen. Zur weiteren Verarbeitung der Aktivierungen in einem Feedforward-Netz gemäß Abschnitt 3.5.1.1 kann die Matrix H zeilen- oder spaltenweise in einen Vektor h überführt werden. Viele Implementierungen von Faltungsschichten führen statt einer Faltung eine Kreuzkorrelation durch. Sie entspricht einer Faltung mit gespiegeltem Kernel und ist in diesem Fall der Faltung gleichwertig, da der Kernel der vom Netz zu lernende Parameter ist.

Bei der Kreuzkorrelation wird der Kernel zeilenweise über die Eingabematrix bewegt. Der Kernel wird an jeder Stelle elementweise mit dem darunterliegenden Ausschnitt der Eingabematrix multipliziert. Die Summe dieser einzelnen Produkte werden anschließend an die entsprechende Stelle der Ausgabematrix geschrieben. Bei diesem Vorgang können die Kernelgröße sowie die Schrittweite des Kernels als Hyperparameter variiert werden. Außerdem ist es möglich, dass der Kernel am Rand der Eingabematrix überstehen darf. In diesem Fall wird am Rand der Eingabe ein Padding (z. B. bestehend aus Nullen) eingefügt. Außerdem kann der Kernel aus mehreren Schichten bestehen. Diese werden Filter genannt und ermöglichen das gleichzeitige Extrahieren verschiedener Informationen aus der Eingabe. Zudem besteht die Eingabe ebenfalls aus mehreren Schichten, sofern mehr als ein Feature vorhanden ist. Im Fall von Bilddaten können beispielsweise die verschiedenen Farbkanäle die Features bilden.

Der Einsatz von Faltungsschichten ist nicht auf zweidimensionale Daten beschränkt. Neben der hier vorgestellten Variante, die für den Einsatz bei der Bildverarbeitung geeignet ist, kann das Prinzip einer Faltungsschicht ebenso auf ein- oder dreidimensionale Daten angewendet werden. So lassen sich dann beispielsweise Audio- oder Videodaten mit dem Faltungsnetz verarbeiten.

Vorteilhaft an Faltungsschichten ist die Reduktion der Parameterzahl gegenüber eines Perzeptrons, da der Kernel wesentlich kleiner als eine Gewichtsmatrix ist. Dadurch kann ein Faltungsnetz effizienter trainiert und gespeichert werden. Darüber hinaus wird der Kernel an der gesamten Eingabematrix angewendet, wodurch die Eigenschaft der Äquivarianz nach [NG96] erreicht wird. Dies bedeutet, dass Verschiebungen in der Eingabe zu gleichartigen Verschiebungen der Ausgabe führen. Dies ist bei Feedforward-Netzen im Allgemeinen nicht der Fall.

Da Faltungsnetze wie in Abb. 3.11 auch in einem Berechnungsgraph dargestellt werden können, sind die in Abschnitt 3.5.1.2 vorgestellten Optimierungsverfahren und der in Abschnitt 3.5.1.3 beschriebene Backpropagation-Algorithmus ebenso auf ein Faltungsnetz anwendbar.

Faltungsnetze wurden bisher noch nicht für die Ensemble-Kalibrierung eingesetzt. Dies kann jedoch analog zu [RL18] und Abschnitt 3.5.1.1 durchgeführt werden. Zur Schätzung der Parameter μ und σ der Normalverteilung sind dazu entsprechend zwei Neuronen in der Ausgabeschicht enthalten. Dafür muss das Faltungsnetz zuvor in ein Perzeptron überführt worden sein. Die Eingaben für die Netze müssen allerdings pro Feature als zweidimensionale Matrizen vorliegen.

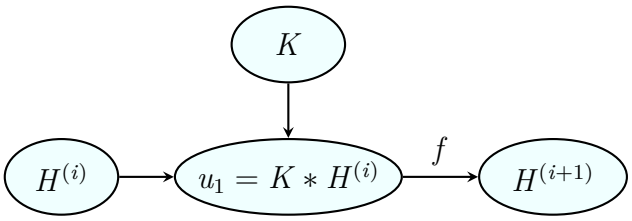

Abbildung 3.11.: Berechnungsgraph einer Faltungsschicht

3.5.1.5. Dropout-Schichten

Eine Dropout-Schicht nach [Sri+14] ist eine Möglichkeit zur Regularisierung und ist damit zur Verringerung von Overfitting von neuronalen Netzen geeignet. Sie biete hohe Effektivität bei geringem zusätzlichen Rechenaufwand. Im Grunde wird bei Anwendung von Dropout-Schichten ein sehr großes Ensemble von neuronalen Netzen (gemäß Bagging wie in Abschnitt 3.1.3 beschrieben) trainiert, indem während des Trainings in jedem Schritt zufällig ein Anteil der Neuronen in der Dropout-Schicht entfernt wird. Im Fall von Feedforward-Netzen und Faltungsnetzen geschieht dies durch Multiplikation der Aktivierungen der betroffenen Neuronen mit 0. Die Wahrscheinlichkeit, mit der ein Neuron ausfällt, muss vor dem Training festgelegt werden. Das Ensemble besteht dann aus den Netzen, die durch alle möglichen Kombinationen von entfernten und vorhandenen Neuronen in den betroffenen Schichten erzeugt werden können. Ein Beispiel dazu ist in Abb. 3.12 zu sehen.

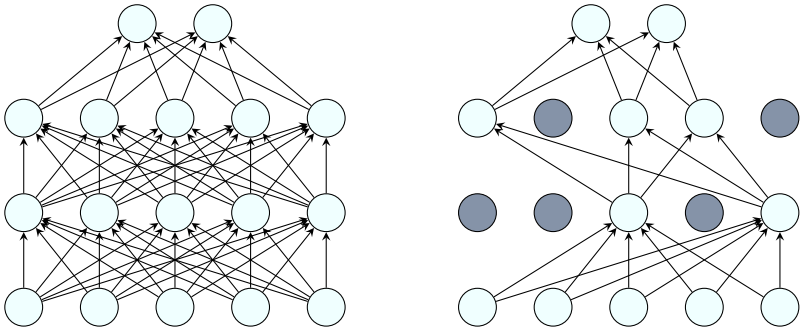

Abbildung 3.12.: Beispiel zur Anwendung von Dropout

Während beim Bagging die einzelnen Modelle im Ensemble unabhängig sind, teilen sie sich im Fall von Dropout ihre Parameter. Wegen der zufälligen Entscheidung, ob ein Neuron ausfällt oder nicht, wird in Wahrheit nur ein geringer Teil der im Ensemble enthaltenen Netze für meist nur einen Schritt explizit trainiert. Weil sich die Netze aber ihre Parameter teilen, erreichen auf diese Weise alle Netze am Ende eine gute Einstellung der Parameter. Dadurch bleibt auch der Speicher- und Rechenaufwand beim Training dieses Ensembles sehr gering.

Die Vorhersagen der im Ensemble enthaltenen Netze müssen am Schluss gemittelt werden. Statt auf das arithmetische Mittel zurückzugreifen, wird hier jedoch das geometrische Mittel verwendet. Da die Berechnung der einzelnen Netze allerdings sehr aufwändig wäre, wird in der Praxis zur Vorhersage nur einmal das komplette Netz mit allen Neuronen ausgewertet. Dabei werden dann bei den Neuronen, an denen im Training Dropout angewendet wurde, die ausgehenden Gewichtungen mit der Wahrscheinlichkeit, zu der die Neuronen bei Anwendung von Dropout erhalten bleiben, multipliziert. Damit wird die Erwartung der Ausgabe des Neurons im gesamten Ensemble simuliert.

Da Dropout eine Regularisierungstechnik ist, wird die Komplexität des Netzes durch

ihre Anwendung eingeschränkt. Daher ist die effektive Komplexität des neuronalen Netzes geringer, als durch die Struktur des Netzes gegeben ist.

3.5.2. Implementierung

Für die Implementierung der neuronalen Netze zur Ensemble-Kalibrierung wurde das Python-Modul `keras` [Cho+15] mit dem Backend `tensorflow` [Aba+15] eingesetzt. Es wurde die Python-Version 3.6.1 verwendet. Mit `keras` wird eine einfache und schnelle Implementierung von neuronalen Netzen ermöglicht, indem eine sehr benutzerfreundliche Programmierschnittstelle bereitgestellt wird. Außerdem kann `keras` aufgrund der Modularität leicht durch eigene Implementierungen ergänzt werden. Für die konkreten Berechnungen wird auf `tensorflow` zurückgegriffen. Dabei handelt es sich im eine Open-Source-Bibliothek, mit der hochperformante numerische Berechnungen sowohl auf CPUs als auch auf GPUs möglich sind. Besonders durch den Einsatz von GPUs kann das Training neuronaler Netze enorm beschleunigt werden. Daher wurden die Berechnungen der neuronalen Netze für diese Arbeit auf den verfügbaren GPUs des Systems JURON (Abschnitt 2.3) durchgeführt.

3.5.2.1. Netztopologien

In dieser Arbeit werden wie in [RL18] Feedforward-Netze eingesetzt. In der Ausgabeschicht sind jeweils zwei Neuronen enthalten, da damit wie in Abschnitt 3.5.1.1 beschrieben die Parameter der Normalverteilung geschätzt werden. Die Eingabefeatures der Netze sind ebenso wie in Abschnitt 3.5.1.1 beschrieben die Ensemblemittel und -standardabweichungen der einzelnen Modellvariablen. Aus einer Variablen ergeben sich somit zwei Features. Es werden die in Tabelle 2.1 auf Seite 15 aufgeführten Modellvariablen eingesetzt. Da das Modellgitter, die Höhe des Terrains sowie der Grundzustand des Luftdrucks bei allen Membern gleich ist, kann auf die Mittelwertbildung und die Berechnung der Standardabweichung der Variablen `XLAT`, `XLONG`, `HGT` und `PB` verzichtet werden. Stattdessen werden für diese Variablen die Werte eines Members übernommen. Ebenso ist bei den stationsspezifischen Informationen (Breitengrad `LAT`, Längengrad `LON` und Höhe `ALT`) keine weitere Behandlung nötig, da diese ohnehin für jede Station nur einmalig vorliegen. Somit ergeben sich aus 28 Features für die Modelldaten und 3 Features für die Stationsinformationen insgesamt 31 Features. Vor dem Training eines neuronalen Netzes werden die Features normiert, sodass sich pro Feature der Mittelwert 0 und die Standardabweichung 1 über alle Samples ergibt. Durch diese Skalierung der Features auf dieselbe Größenordnung können neuronale Netze besser mit den Daten trainiert werden.

Wie in [RL18] werden hier neuronale Netze implementiert, welche die Abbildung von den Modellwerten an den Messstationen und den stationsspezifischen Informationen auf die kalibrierten Vorhersagen lernen sollen. Um ein Verfahren zu erhalten, was am gesamten Modellgitter angewendet werden kann, werden im Gegensatz zu [RL18] die Stationskennungen nicht verarbeitet. Damit der Einfluss räumlicher Muster in den Modelldaten untersucht werden kann, werden zusätzlich neuronale Netze eingesetzt, welche die in Abschnitt 2.3 beschriebenen Modellumgebungen verarbeiten. Für die Netze, die keine

Umgebungen verarbeiten, können die Features der Modellwerte an den Stationen (Mittelpunkte der Umgebungen) direkt mit den stationsspezifischen Informationen zusammen in einem Array der Form `(n, 31)` mit `n` als Anzahl der Samples gehalten werden. So können problemlos die Feedforward-Netze trainiert werden. Wenn bei den Feedforward-Netzen auch Umgebungen verarbeitet werden sollen, müssen die zweidimensionalen Umgebungen zunächst in mehrere Features zerlegt werden. Dazu wird jeder einzelne Punkt in der Umgebung eines Features als eigenes Feature aufgefasst. Die Umgebungen der Form `(n, 11, 11, 28)` werden somit in die Form `(n, 3388)` überführt. Werden noch die Stationsinformationen hinzugefügt, ergibt sich ein Array der Größe `(n, 3391)`. Damit besitzen die betroffenen Netze 3391 Eingabefeatures und daher entsprechend viele Neuronen in der Eingabeschicht.

Bei den Faltungsnetzen werden zu den Umgebungen aus dem Modell punktweise Mittelwert und Standardabweichung über die Member gebildet und beides als zweidimensionale Matrix in das Netz hineingegeben. Diese können direkt über Faltungsschichten verarbeitet werden. Die stationsspezifischen Informationen in den Features sind jedoch skalar und können daher nicht zusammen mit den Modellvariablen in den Faltungsschichten verarbeitet werden. Um die Stationsinformationen dennoch in das neuronale Netz einfließen zu lassen, besitzen die Faltungsnetze zwei Eingabeschichten. Dies ist mit `keras` und der zugehörigen Functional API möglich. Ein Beispiel für den Aufbau eines solchen Netzes ist in Abb. 3.13 veranschaulicht. Die Umgebungen werden zunächst mit den Faltungsschichten (Conv2D) verarbeitet. Anschließend wird eine Dropout-Schicht eingefügt. Danach wird das Netz mittels Flatten in ein Perzeptron überführt. Die Stationsinformationen (Koordinaten der Station) werden in einem Perzeptron mit einer vollvernetzten

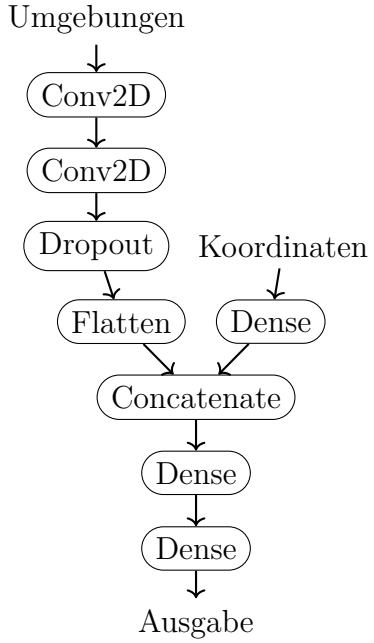

Abbildung 3.13.: Visualisierung eines Faltungsnetzes

Schicht (Dense) verarbeitet. Danach wird es mit dem Perzeptron von den Umgebungen zusammengeführt (Concatenate) und bis zur Ausgabeschicht als Feedforward-Netz weitergeführt. Die verschiedenen in dieser Arbeit eingesetzten Faltungsnetze unterscheiden sich hauptsächlich in der Anzahl und Größe der einzelnen Schichten.

3.5.2.2. CRPS als Verlustfunktion

Um die neuronalen Netze in `keras` wie in Abschnitt 3.5 beschrieben trainieren zu können, muss der CRPS als Verlustfunktion beim Kompilieren der Netze eingestellt werden. Da der CRPS nicht als Bibliotheksfunktion direkt von `keras` bereitgestellt wird, muss dieser als benutzerdefinierte Verlustfunktion zunächst eigenständig implementiert werden. Dabei sind ein paar Dinge zu beachten. Die Implementierung des CRPS für diese Arbeit wird in Listing 3.1 gezeigt.

Eine Verlustfunktion in `keras` ist eine Python-Funktion, die genau zwei Argumente entgegennimmt. Als erstes Argument erhält sie die Labels `y_true`, was in diesem Fall den Beobachtungen entspricht. Sie werden während des Trainings von `keras` als Tensor der Form `(n, 1)` mit `n` als Anzahl der Samples an die Funktion übergeben. Das zweite Argument `y_pred` sind die Vorhersagen des neuronalen Netzes. Da in diesem Fall das neuronale Netz zwei Neuronen in der Ausgabeschicht enthält (vgl. Abschnitt 3.5.1.1), werden die Vorhersagen als Tensor der Form `(n, 2)` bereitgestellt.

Um während des Trainings die Gradienten für die Gewichte der neuronalen Netze bilden zu können, muss jede Operation, die auf die Tensoren angewendet wird, für `keras` nachvollziehbar bleiben. Nur so kann später intern der Berechnungsgraph aufgestellt und der Backpropagation-Algorithmus (Abschnitt 3.5.1.3) durchgeführt werden. Daher dürfen nur Funktionen aus dem Backend von `keras` verwendet werden. Da `keras` verschiedene Implementierungen als Backends unterstützt, sind einige häufig genutzte und verfügbare Funktionen und Operationen für Tensoren unter `keras.backend` erreichbar. Das Backend wird hier in Z. 1 unter dem Namen `K` importiert.

Zur Berechnung des CRPS gemäß Gleichung (3.6) auf Seite 24 muss der standardisierte Vorhersagefehler bestimmt werden. Dazu wird in Z. 16 die Differenz zwischen den Labels `y_true` und den Erwartungswerten `y_pred[:, 0]` gebildet und schließlich durch die Standardabweichung geteilt. Dabei ist zu beachten, dass die Erwartungswerte durch das Slicing einen eindimensionalen Tensor bilden. Um die Differenz korrekt bilden zu können, müssen auch die Labels als solcher vorliegen. Daher werden sie mittels der Funktion `K.squeeze` in die gewünschte eindimensionale Form überführt. Um während des Trainings fehlerhafte CRPS-Werte durch negative Standardabweichungen zu vermeiden, wird in Z. 15 der Betrag der Standardabweichungen `y_pred[:, 1]` gebildet. Außerdem wird eine kleine Konstante hinzuaddiert, damit die Standardabweichung nicht Null wird, da dies später im CRPS zu `nan`-Werten führen würde.

Für den CRPS werden nach Definition in Gleichung (3.6) die Dichte und die Verteilungsfunktion der Standardnormalverteilung benötigt. Diese sind jedoch nicht in `keras` enthalten. Stattdessen muss hier auf das darunterliegende Backend `tensorflow` direkt zurückgegriffen werden. Dort wird die Normalverteilung als Klasse bereitgestellt und muss unter Angabe des Erwartungswerts und der Varianz initialisiert werden. Die Stan-

dardnormalverteilung ist in Z. 5 als globale Variable umgesetzt. Da der letzte Teil im CRPS $\frac{1}{\sqrt{\pi}}$ ein skalarer und konstanter Wert ist, der insbesondere von den Labels und Vorhersagen unabhängig ist, wird dieser in Z. 6 mittels `numpy` berechnet. Zum Schluss werden die berechneten CRPS-Werte schließlich in Z. 17 mittels `K.mean` über alle Samples gemittelt.

Listing 3.1: Implementierung des CRPS für Keras

```
from keras import backend as K
import tensorflow as tf
import numpy as np

phi = tf.distributions.Normal(0.0, 1.0)
c = 1/np.sqrt(np.pi)

def crps_keras(y_true, y_pred):
    """
    Mean Continuous Ranked Probability Score (CRPS) loss function
        ↪ for Keras

    :param y_true: True labels
    :param y_pred: Predictions
    """
    sigma = K.abs(y_pred[:, 1])+1e-14
    z = (K.squeeze(y_true, 1) - y_pred[:, 0])/sigma
    return K.mean(sigma*(z*(2*phi.cdf(z)-1)+2*phi.prob(z)-c), axis
        ↪ =0)
```

3.5.2.3. Ensembles zur Verringerung der Varianz

Bei ersten Testdurchläufen der neuronalen Netze konnten zwar schon gute Ergebnisse erzielt werden, jedoch hing das endgültige Ergebnis stark davon ab, zu welcher Epoche genau das Training beendet wurde. Um dies genauer zu untersuchen, wurden die Lernkurven betrachtet. Da in den ersten paar Epochen während des Trainings der Fehler meist noch sehr hoch ist, werden die Lernkurven in dieser Arbeit meist nicht direkt ab der ersten Epoche gezeigt. Dadurch sind die Lernkurven besser zu interpretieren. In der Lernkurve in Abb. 3.14 ist der Fehler auf den Trainingsdaten (In-Sample-Fehler E_{in}) blau und der Fehler auf den Validierungsdaten (Schätzer von E_{out}) rot eingezeichnet. Der Trainingsfehler verläuft stabil und verringert sich mit zunehmender Anzahl der Epochen. Der Validierungsfehler fällt insgesamt zwar auch, jedoch schwankt der konkrete Wert stark von Epoche zu Epoche. Daher kann es passieren, dass das Training zu einem Zeitpunkt beendet wird, an dem das Netz schlecht generalisiert. Die Schwankung und Instabilität des Validierungsfehlers weist auf eine zu hohe Varianz des neuronalen Netzes gemäß Abschnitt 3.1.1 hin. Um dies in den Griff zu bekommen, können Ensembles von neuronalen Netzen wie bei Bagging/Pasting aus Abschnitt 3.1.3 eingesetzt werden. Auch in [RL18] wurden Ensembles von neuronalen Netzen trainiert, um die Stabilität der

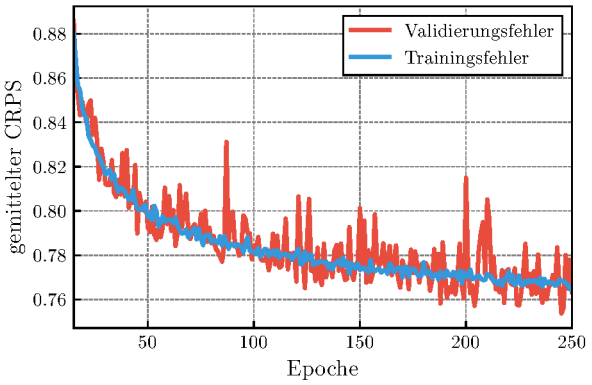

Abbildung 3.14.: Lernkurve bei hoher Varianz

Parameterschätzungen für μ und σ zu erhöhen. Die Ensembles von neuronalen Netzen sind dabei im Kontext des Machine Learning zu betrachten und bilden ein übliches Mittel zur Verringerung der Varianz eines Lernmodells. Ihr Einsatz ist unabhängig davon, dass sie für eine Ensemble-Kalibrierung eingesetzt werden, und sind daher getrennt vom Ensemble des meteorologischen Modells zu betrachten. Während beim meteorologischen Modell mit dem Ensemble die Unsicherheit der Vorhersage geschätzt wird, dienen die Ensembles von neuronalen Netzen der Verringerung der Varianz sowie einer möglichst genauen Schätzung der Parameter μ und σ.

Bei Ensembles von neuronalen Netzen müssen keine Bootstrap-Stichproben des Trainingsdatensatzes erzeugt werden, um unabhängige Modelle zu erhalten. Nach [GBC16] reicht es auch bei Wiederverwendung derselben Netzstruktur aus, für jedes der Netze eine andere Initialisierung der Parameter und Gewichte zu wählen. Neuronale Netze haben in der Regel eine große Vielfalt an möglichen Endparametrierungen, sodass allein die unterschiedliche zufällige Initialisierung ausreicht, um im Ensemble eine merkliche Verbesserung zu erzielen. Auch die zufällige Wahl von Batches beim stochastischen Gradientenabstieg trägt dazu bei, dass die einzelnen Netze in verschiedene Minima hineinlaufen und somit unterschiedlich generalisieren. Im Fall einer Normalverteilung können zur Bildung der Vorhersage die Schätzungen für μ und σ über die Netze gemittelt werden.

Das Trainieren eines Ensembles von neuronalen Netzen zieht einen größeren Speicherbedarf und eine erhöhte Laufzeit nach sich. Die Anzahl der im Ensemble enthaltenen Netze hat darauf direkten Einfluss. In dieser Arbeit wurden wie in [RL18] Ensembles von 10 neuronalen Netzen trainiert. Die Netze in einem Ensemble haben dabei dieselbe Struktur und unterscheiden sich nur in der Initialisierung ihrer Gewichte und sonstigen Zufallsprozessen während des Trainings. Um die Trainingszeit möglichst gering zu halten, werden die Netze eines Ensembles für diese Arbeit parallel trainiert. Dazu wurde das bereits existierende Programm mittels MPI (Message Passing Interface) unter Verwendung des Python-Moduls `mpi4py` [Dal+11] parallelisiert, sodass alle 10 Netze eines Ensembles gleichzeitig trainiert werden. Durch den von MPI vergebenen Rang kann jeder Prozess einen anderen Seed zur zufälligen Generierung der Parameterinitialisierungen wählen und damit je ein neuronales Netz trainieren. Die Gewichte der einzelnen Netze werden

schließlich abgespeichert, um später Vorhersagen aus den Netzen gewinnen zu können. Für die Berechnung einer Vorhersage müssen alle einzelnen Netze zunächst eine eigene Vorhersage bilden, um sie anschließend im Ensemble mitteln zu können.

In Abb. 3.15 sind die Lernkurven eines Ensembles zu sehen. Um alle Netze des Ensembles abzubilden, ist der Mittelwert der Lernkurven aller Netze als Linie und die Standardabweichung der Lernkurven aller Netze als transparente Fläche eingezeichnet. Obwohl alle im Ensemble enthaltenen Netze eine hohe Varianz aufweisen, konnte der Fehler des Gesamtergebnisses durch Verwendung des Ensembles erfolgreich und stabil verringert werden.

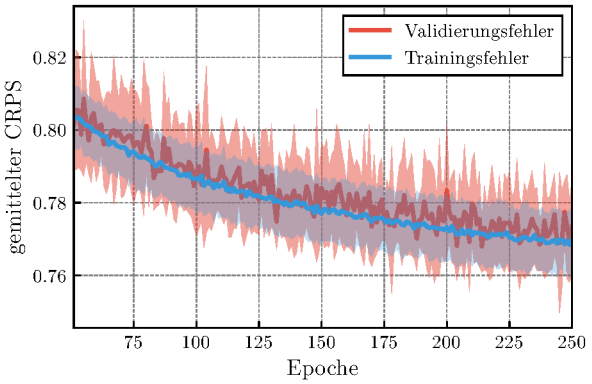

Abbildung 3.15.: Lernkurven eines Ensembles von neuronalen Netzen

3.5.2.4. Gesteuerte Anpassung der Lernrate

Nach Einführung der Ensembles wurden weitere neuronale Netze trainiert. Bei fast allen Lernkurven fiel dabei auf, dass die Netze nach den zuvor festgelegten 250 Epochen noch nicht genug gesättigt waren. Dies ist in Abb. 3.15 an dem noch fallenden Validierungsfehler zu sehen. Da nach Umstellung auf 4000 Epochen noch immer fallende Validierungsfehler zu sehen sind (siehe Abb. 3.16), mussten weitere Anpassungen am Training der Netze vorgenommen werden.

Um den Trainingsprozess zu verkürzen, aber gleichzeitig die Netze ausreichend zu sättigen, soll die Lernrate angepasst werden. Die Lernrate wurde bisher vor dem Training festgelegt und blieb konstant, da das eingesetzte Optimierungsverfahren Adam aus Abschnitt 3.5.1.2 die Skalierung der Lernrate während des Trainings adaptiv anpasst. Das Ändern der Lernrate vor dem Training hat keine Verbesserung erzielt. Daher soll die Lernrate im Laufe des Trainings manuell verkleinert werden. Dazu wird der `LearningRateScheduler` von `keras` eingesetzt. Dieser bietet die Möglichkeit, für jede Epoche im Training eine andere Lernrate zu wählen. Nach [GBC16] und Abschnitt 3.5.1.2 wird typischerweise zu Anfang des Trainings eine hohe Lernrate gewählt, die schließlich im Laufe des Trainings zunehmend verringert wird. Im Programm muss dazu eine Python-Funktion implementiert werden, welche die Ordnungszahl einer Epoche als ganze Zahl empfängt und die in dieser Epoche zu verwendenden Lernrate zurückgibt. Häufig ge-

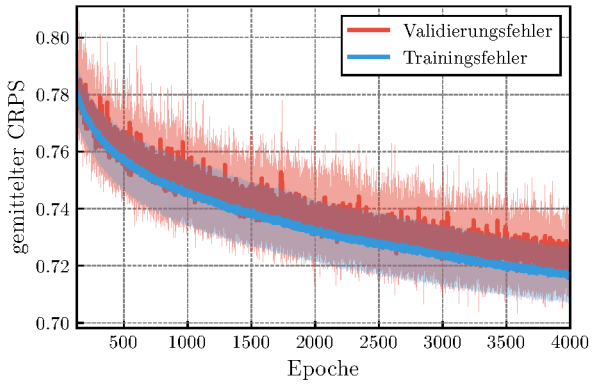

Abbildung 3.16.: Lernkurven eines Trainings mit 4000 Epochen

schieht die Anpassung der Lernrate dabei stückweise, d. h. eine gewählte Lernrate bleibt für eine bestimmte Anzahl von Epochen gleich, bevor sie weiterhin verringert wird. Für diese Arbeit wurde jeweils mit einer Startlernrate lr_0 begonnen, die alle *epochs_drop* Epochen mit dem Faktor $drop \in (0;1)$ multipliziert wird. Demnach wird die Anpassung der Lernrate gemäß der Gleichung

$$lr(epoch) = lr_0 \cdot drop^{\left\lfloor \frac{1+epoch}{epochs\_drop} \right\rfloor} \tag{3.27}$$

vollzogen. Dabei wird die aktuelle Epoche *epoch* ab 0 indiziert. Die konkreten Werte für die Startlernrate, den Faktor und die Dauer pro Lernrate sind für jedes der zu trainierenden neuronalen Netze geeignet anzupassen.

Um den `LearningRateScheduler` im Training einzusetzen, muss er als Callback fürs Training der Netze angegeben werden. Dazu wird in dem `fit`-Befehl, der das Training eines neuronalen Netzes durchführt, eine Liste von Callbacks mitgegeben. In dieser muss sich der `LearningRateScheduler` befinden, der zuvor unter Angabe der zu nutzenden Python-Funktion, die Gleichung (3.27) implementiert, initialisiert wurde. Dies ist in Listing 3.2 zu sehen.

Listing 3.2: Verwendung eines Callbacks (Code gekürzt)

```
import numpy as np
from crps import crps_keras
from neural_networks import fully_connected
from keras.callbacks import LearningRateScheduler

def step_decay(epoch):
    """
    returns learning rate for given epoch
    """
    init_lr = 0.1
    drop = 0.75
    epochs_drop = 20
```

```
    return init_lr*np.power(drop, np.floor((1+epoch)/epochs_drop))

model = fully_connected(n_features, 2)
model.compile(loss=crps_keras, optimizer='adam')

callbacks = [LearningRateScheduler(step_decay, 1)]
hist = model.fit(x_train, y_train, batch_size=1024, epochs=500,
                 verbose=2, validation_data=(x_val, y_val),
                 callbacks=callbacks)
```

In Abb. 3.17 ist erkennbar, dass die gesteuerte Anpassung der Lernrate gut dabei hilft, die Netze ausreichend zu sättigen. Bei jeder Veränderung der Lernrate ist ein Sprung der Verlustfunktion zu beobachten. Gleichzeitig kann mit der Anwendung des `LearningRateScheduler` auch die Varianz der einzelnen Netze verringert werden.

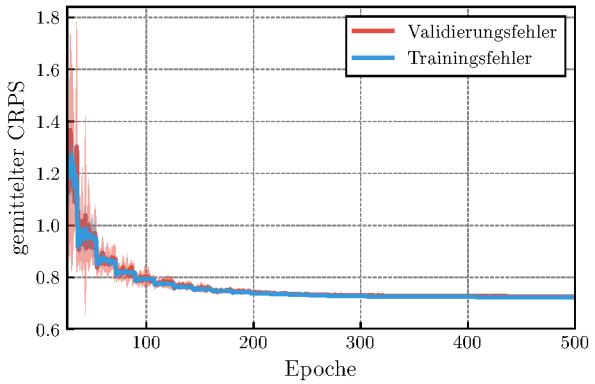

Abbildung 3.17.: Lernkurven bei gesteuerter Anpassung der Lernrate

4. Experiment

In diesem Kapitel wird die Durchführung des konkreten Experiments beschrieben. Anschließend werden die Ergebnisse diskutiert.

4.1. Vorgehen

Die Ensemble-Kalibrierung soll für Vorhersagen der Temperatur in 2 m Höhe durchgeführt werden. Aus dem vorliegenden Datensatz, der in Kapitel 2 beschrieben wurde, werden dazu zufällig 25 Prozent entnommen und gleichmäßig in einen Test- und Validierungsdatensatz aufgeteilt. Die restlichen Daten dienen als Trainingsdatensatz. Die Kalibrierung wird anhand des Trainingsdatensatzes mit verschiedenen Verfahren und Netzstrukturen durchgeführt. Das EMOS-Verfahren wird zum einen in der vereinfachten Variante, genannt `EMOS-mean`, und zum anderen in der Grundvariante, genannt `EMOS`, angewendet. Beide Verfahren nutzen zur Kalibrierung lediglich die Modellwerte an den Messstationen.

Daneben werden wie in [RL18] neuronale Netze eingesetzt, die keine versteckte Schicht besitzen. Dort sind die Neuronen der Eingabeschicht direkt mit den beiden Neuronen der Ausgabeschicht verbunden. Bei diesem Verfahren werden die Parameter der Normalverteilung demnach durch lineare Funktionen der Eingabefeatures geschätzt und können daher als eine Erweiterung von `EMOS-mean` angesehen werden. Im Gegensatz zu `EMOS-mean` sind hier die Schätzungen für beide Parameter (μ und σ) sowohl vom Ensemblemittel als auch von der Ensemblevarianz abhängig. Außerdem sind zusätzliche Features enthalten. Um den Einfluss der stationsspezifischen Informationen in den Features (vgl. Tabelle 2.1 auf Seite 15) auf die Ensemble-Kalibrierung zu untersuchen, wird diese Variante sowohl mit (genannt `LNN`, engl. *linear neural network*) als auch ohne die Stationsinformationen (genannt `LNN-mod`, `mod` für „Modellvariablen“) in den Features durchgeführt. Zur Untersuchung der Muster im Modell wird ebenfalls eine Variante eingesetzt, welche die kompletten Modellumgebungen mit allen Features verarbeitet. Diese heißt im Folgenden `LNN-umg` (`umg` für „Umgebung“).

Darüber hinaus werden Feedforward-Netze genutzt, die mindestens eine und höchstens vier versteckte Schichten beinhalten. Dies wird einerseits als `NN` (engl. *neural network*) nur unter Verwendung der Modellwerte an den Messstationen und andererseits als `NN-umg` unter Einbeziehung der Modellumgebungen durchgeführt. Im Gegensatz zu den Varianten von `LNN` sind bei diesen Netzen nun auch nichtlineare Beziehungen darstellbar. Da die Struktur des neuronalen Netzes an das Problem angepasst sein soll, werden in beiden Fällen verschiedene Netzstrukturen ausprobiert. Am Schluss wird jeweils dasjenige Netz ausgewählt, das den geringsten Fehler auf den Validierungsdaten vorweisen

konnte. Als Hyperparameter wird die Anzahl versteckter Schichten und die Anzahl der Neuronen in den Schichten variiert. Insgesamt werden je 8 verschiedene Netzstrukturen getestet.

Zuletzt wird ein Faltungsnetz eingesetzt, genannt `CNN` (engl. *convolutional neural network*), das die Modellumgebungen verarbeitet und so räumliche Merkmale in den Daten erkennen kann. Auch hier wird die Netzstruktur variiert, um ein möglichst geeignetes Netz zur Lösung des Problems zu finden. Die Netze enthalten bis zu vier Faltungsschichten und verschieden viele vollvernetzte Schichten. Auch die Anzahl der Neuronen in den vollvernetzten Schichten wird verändert. Zusätzlich werden die Faltungsnetze sowohl mit als auch ohne Dropout-Schicht angewendet. Insgesamt ergeben sich 12 verschiedene Faltungsnetze.

Eine Übersicht über alle Verfahren ist in Tabelle 4.1 zu finden. Für die neuronalen Netze wurden die in Abschnitt 3.5.2 beschriebenen Features eingesetzt. Außerdem wurden wie dort ebenso beschrieben Ensembles von jeweils 10 neuronalen Netzen trainiert, um die Varianz der Netze zu verringern. Die Größe dieser Ensembles von neuronalen Netzen wird bei ausgewählten Verfahren zusätzlich durch Hinzunahme weiterer Netze ins Ensemble genauer untersucht. Wenn bei einem Verfahren mehrere Netzstrukturen untersucht werden sollen, geschieht die Auswahl der besten Netzstruktur auf Grundlage der Validierungsdaten. Der Ablauf des Trainings eines Ensembles von neuronalen Netzen und der Wahl der Netzstruktur aufgrund des Validierungsfehlers ist in Abb. 4.1 in einem Struktogramm veranschaulicht.

Tabelle 4.1.: Übersicht über die eingesetzten Verfahren

Kürzel	**Beschreibung**
`EMOS-mean`	vereinfachte Variante des EMOS-Verfahrens
`EMOS`	EMOS-Grundverfahren
`LNN-mod`	neuronales Netz ohne versteckte Schicht und ohne Stationsinformationen in den Features
`LNN`	`LNN-mod` mit allen Features
`LNN-umg`	`LNN` mit Umgebungen als Eingabe
`NN`	neuronales Netz mit mind. einer versteckten Schicht und mit allen Features
`NN-umg`	`NN` mit Umgebungen als Eingabe
`CNN`	Faltungsnetz mit Umgebungen und allen Features

4.2. Ergebnisse

Es gibt keine absolute Größe, mit der die Güte eines Ensembles quantifiziert werden kann. Daher werden im Folgenden neben dem arithmetischen Mittel und der Standardabweichung der CRPS-Werte auch PIT-Histogramme sowie Boxplots zur Verteilung der CRPSS-Werte betrachtet, um die Ergebnisse der Kalibrierung interpretieren zu können.

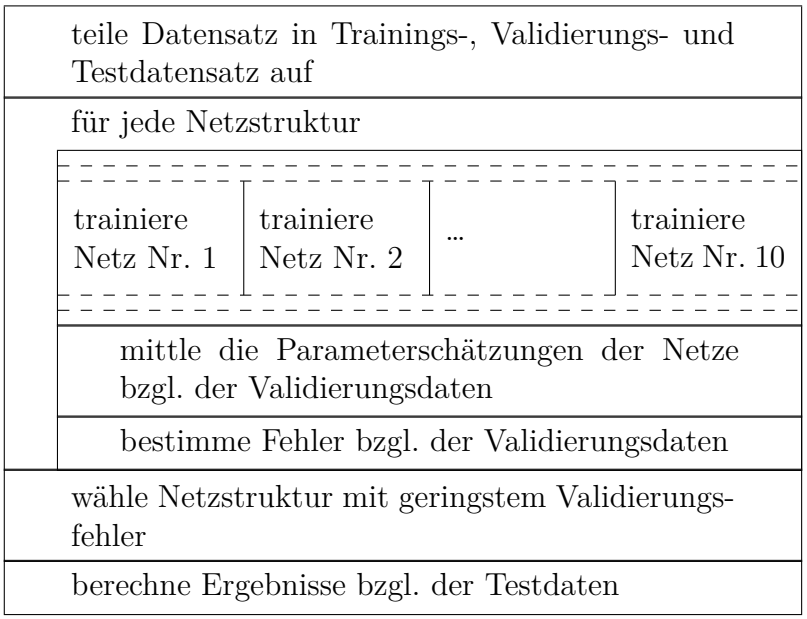

Abbildung 4.1.: Struktogramm zum Training der neuronalen Netze

In Tabelle 4.2 sind die Ergebnisse der einzelnen Verfahren auf dem Testdatensatz hinsichtlich CRPS (Abschnitt 3.2) eingetragen. Zusätzlich sind dort die Trainingszeiten der einzelnen Verfahren (ohne Laden der Daten und ohne Auswertung) sowie die Anzahl der gelernten freien Parameter aufgelistet. Neben den über die Samples gemittelten CRPS-Werten sind auch die Standardabweichungen der CRPS-Werte eingetragen. Außerdem sind in der ersten Zeile die Ergebnisse für das Modellensemble ohne Anwendung eines Verfahrens vorhanden. Dadurch wird erkennbar, dass jedes der Verfahren den CRPS gegenüber dem Modellensemble erfolgreich verringern konnte. Die beiden EMOS-Verfahren schneiden dabei am schlechtesten ab. Das Verfahren `EMOS` ist nur leicht besser als die vereinfachte Variante `EMOS-mean`. Bei den Verfahren `LNN-mod` und `LNN` sind kleinere CRPS-Werte erreicht worden. Durch die Hinzunahme der Stationsinformationen kann `LNN` dabei eine merkbare Verbesserung gegenüber `LNN-mod` erzielen. Das Verfahren `LNN-umg` erreicht durch Einbeziehung der räumlichen Umgebungen wiederum geringere CRPS-Werte. Das `NN`, das nur die Modellwerte an den Messstationen verarbeitet, konnte gegenüber allen Varianten des `LNN` bessere Ergebnisse erreichen. Dies lässt vermuten, dass nichtlineare Beziehungen zwischen den Features und der geschätzten Wahrscheinlichkeitsverteilung existieren. Unter den 8 betrachteten Netzen für `NN` konnte das Netz mit einer Schicht, die 70 Neuronen enthält, die geringsten CRPS-Werte auf dem Validierungsdatensatz hervorbringen. Bei Hinzunahme der Umgebungen im `NN-umg` verringert sich der CRPS gegenüber `NN` signifikant. Beim `NN-umg` konnte sich das Netz mit zwei Schichten (70 und 30 Neuronen) auf Grundlage des Validierungsdatensatzes durchsetzen. Zuletzt hat das `CNN` die besten Ergebnisse im Experiment hinsichtlich CRPS erzielen können. Dabei ist es nur leicht besser als das `NN-umg`. Beim `CNN` ist ein Faltungsnetz mit zwei Faltungsschichten ohne Dropout-Schicht eingesetzt worden. Weitere Implementierungsdetails zu den neuronalen Netzen sind in Tabelle A.1 im Anhang zu finden.

Tabelle 4.2.: Ergebnisse

Verfahren	**CRPS** Mittel	Standardabweichung	**Parameter**	**Laufzeit** (Training)
Modellensemble	1,41	1,33		
`EMOS-mean`	1,14	0,86	4	22 s
`EMOS`	1,13	0,83	33	8 min
`LNN-mod`	0,95	0,81	58	20 min
`LNN`	0,89	0,79	64	20 min
`LNN-umg`	0,78	0,66	6784	37 min
`NN`	0,70	0,61	2382	24 min
`NN-umg`	0,50	0,50	239 632	42 min
`CNN`	0,46	0,44	1 011 946	45 min

Bei den Varianten von `LNN` hat sich der CRPS durch die Ensemblebildung der neuronalen Netze gegenüber einem einzelnen Netz kaum verringern können. Es zeigt sich jedoch eine höhere Robustheit des Ensembles von neuronalen Netzen. Dies ist in Abb. 4.2 zu sehen. Dort sind die Lernkurven der einzelnen Netze des Ensembles von `LNN` abgebildet. Die Netze scheinen sich aufgrund ihrer CRPS-Werte in zwei Gruppen einteilen zu lassen. Manche Netze konnten am Ende des Trainings einen gemittelten CRPS von etwa 0,92 erreichen. Die übrigen Netze sind anscheinend in ein anderes Minimum hineingelaufen und haben einen CRPS von etwa 0,89 erzielt, was auch dem Gesamtergebnis des `LNN` entspricht. Die Ensemblebildung der neuronalen Netze hat demnach Stabilität in die Vorhersagen gebracht.

In Abb. 4.3 sind die Lernkurven des `NN-umg` und `CNN` zu sehen. Die Kurven der einzelnen Netze werden über ihr Mittel (als Linie) und ihre Standardabweichung (als Fläche) dargestellt. Bei beiden Verfahren ist gegen Ende des Trainings eine große Lücke zwischen dem Trainings- und Validierungsfehler zu beobachten. Die einzelnen Netze generalisieren demnach nicht so gut wie bei den anderen Verfahren (z. B. das `LNN` in Abb. 4.2). Beim

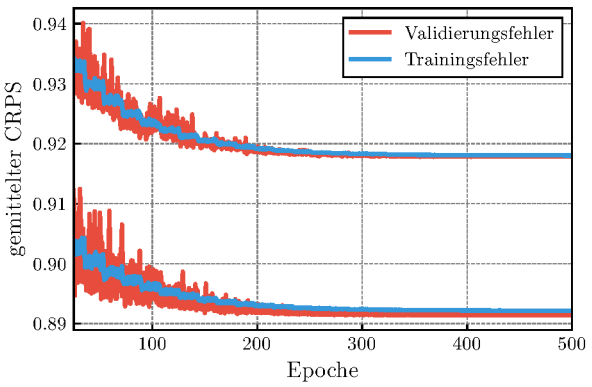

Abbildung 4.2.: Lernkurven von `LNN`

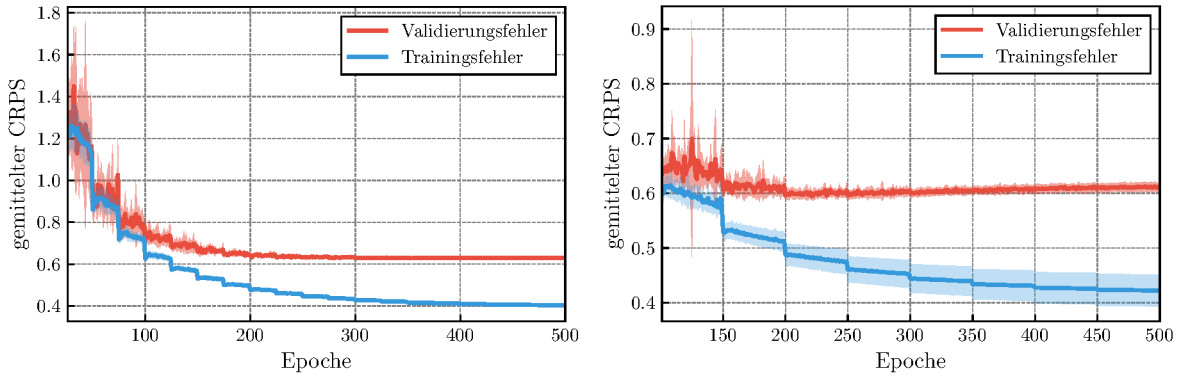

Abbildung 4.3.: Lernkurven von `NN-umg` (links) und `CNN` (rechts)

`CNN` ist die Lernkurve erst ab Epoche 100 zu sehen, damit der Anstieg des Validierungsfehlers erkennbar wird, der auf Overfitting hinweist. Die Lernkurven der restlichen Netze sind in Abb. A.2 bis A.4 im Anhang zu finden.

Bei den beiden Verfahren `NN-umg` und `CNN` konnte das Trainieren eines Ensembles von neuronalen Netzen signifikant bessere Ergebnisse erreichen als ein einzelnes Netz. Um den Einfluss der Größe dieses Ensembles von neuronalen Netzen zu untersuchen, wurden für diese beiden Netze größere Ensembles mit je 50 Netzen trainiert. In Abb. 4.4 ist jeweils der gemittelte CRPS gegen die Anzahl der im Ensemble enthaltenen Netze aufgetragen. Es ist erkennbar, dass der CRPS mit zunehmender Anzahl von Netzen verringert werden kann. Schon beim Einsatz weniger Netze (bis etwa 10 Stück) können deutliche Verbesserungen beobachtet werden. Ab einer Ensemblegröße von etwa 20 Netzen ist durch Hinzunahme weiterer Netze keine wesentliche Verbesserung mehr zu beobachten.

Um die Kalibrierung und die Schärfe der probabilistischen Vorhersagen auf dem Test-

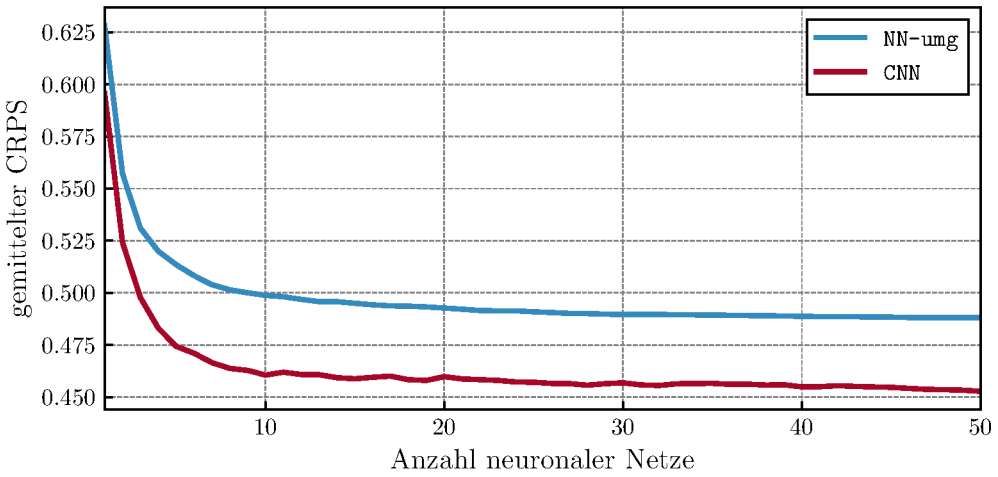

Abbildung 4.4.: Ergebnisse für größere Ensembles von `NN-umg` und `CNN`

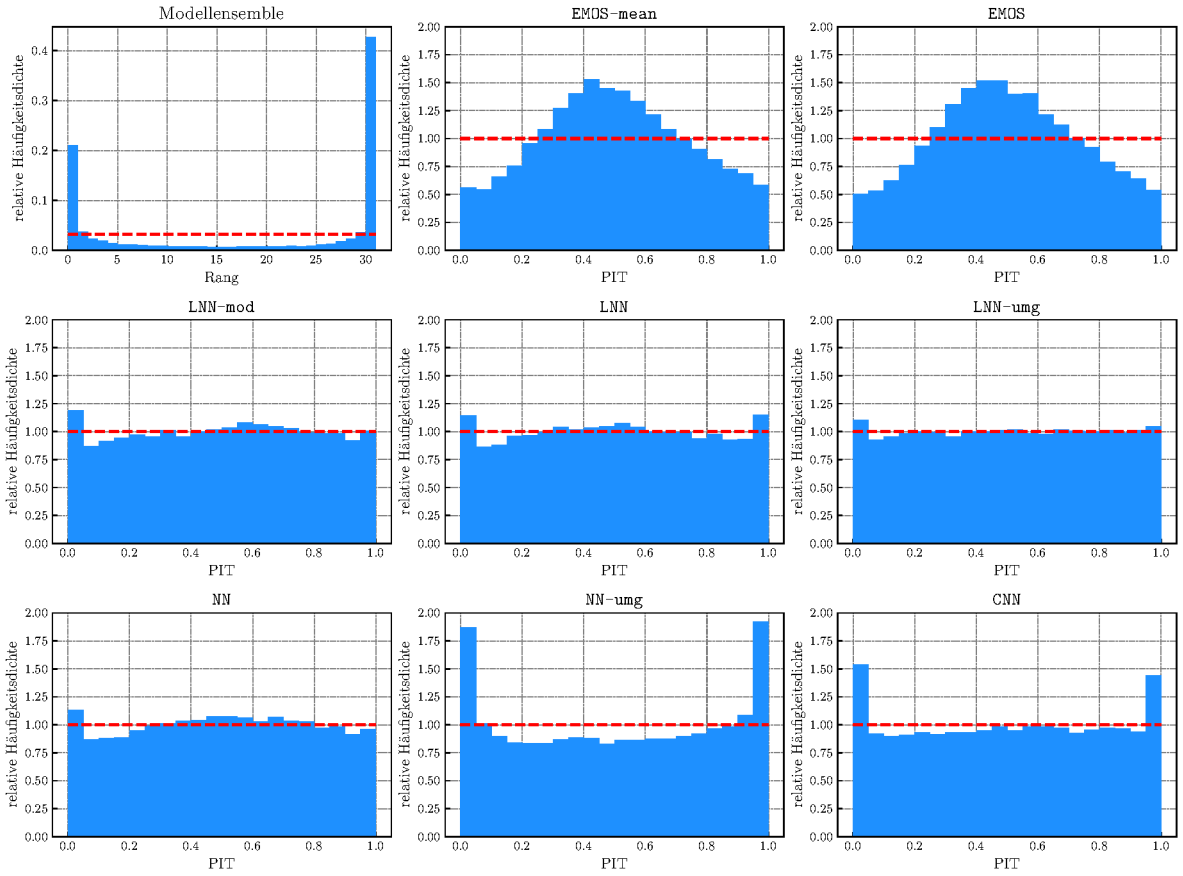

Abbildung 4.5.: Verification-Rank-Histogramm und PIT-Histogramme

datensatz weiterhin untersuchen zu können, sind das Verification-Rank-Histogramm des Modells und die PIT-Histogramme (Abschnitt 2.1.2) aller Verfahren in Abb. 4.5 dargestellt. Für das Modellensemble ist auf dem Testdatensatz eine U-Form zu erkennen, was auf eine zu geringe Varianz des Ensembles hinweist. Bei `EMOS-mean` und `EMOS` ist genau das Gegenteil zu beobachten. Dort wird die Varianz zu groß geschätzt, wodurch sich die Beobachtungen in der Mitte des Histogramms häufen. Bei den Varianten des `LNN` ergeben sich nahezu Gleichverteilungen. Die Kalibrierung und die Schärfe scheinen dort demnach adäquat eingestellt zu sein (vgl. Abschnitt 2.1.2). Die jeweils äußeren Balken der Histogramme sind leicht erhöht. Beim `NN` liegt ebenfalls fast eine Gleichverteilung vor. Das Histogramm des `NN-umg` weist eine leichte U-Form auf, was wieder eine zu gering geschätzte Varianz widerspiegelt. Das `NN-umg` schätzt die Temperatur insgesamt häufig zu hoch oder zu gering ein, wodurch die beiden äußeren Balken stark erhöht sind. Dies ist in einer etwas abgeschwächten Form auch beim `CNN` zu beobachten. Hier scheint der mittlere Teil des Histogramms näher an einer Gleichverteilung zu liegen als `NN-umg`. Insgesamt konnten alle Verfahren im Vergleich zum Modell wesentlich besser kalibrierte und schärfere Temperaturvorhersagen erzielen. Das Histogramm des `LNN-umg` liegt einer Gleichverteilung optisch am nächsten.

Die CRPSS-Werte (berechnet nach Gleichung (3.8)) der einzelnen Verfahren sind in Abb. 4.6 als Boxplots dargestellt. Um die Boxplots genauer untersuchen zu können, sind in Abb. 4.6 Ausreißer nicht dargestellt. Die Boxplots inklusive der Ausreißer sind in Abb. A.1 im Anhang zu finden. Befindet sich ein CRPSS-Wert oberhalb der rot gestrichelten Linie, so ist das dargestellte Verfahren an diesem Datenpunkt besser als das Referenzverfahren, das in der Überschrift des Plots genannt wird (vgl. Abschnitt 3.2). Ist ein Wert unterhalb der rot gestrichelten Linie, liegt eine Verschlechterung des Verfahrens gegenüber des Referenzverfahrens vor. In allen Boxplots ist ein steigender Trend zu beobachten, je besser das Verfahren hinsichtlich CRPS ist. Beim Vergleich von `EMOS-mean` mit `EMOS` ist aufgrund der kleinen Boxen zu sehen, dass die beiden Verfahren eine ähnliche Aussagekraft bezüglich des CRPS besitzen. Dies ist ebenfalls beim Vergleich von `LNN-mod` mit `LNN` zu beobachten. Außerdem fällt bei allen Plots auf, dass `NN-umg` sowie `CNN` deutlich bessere Vorhersagen als die übrigen Verfahren erzielen.

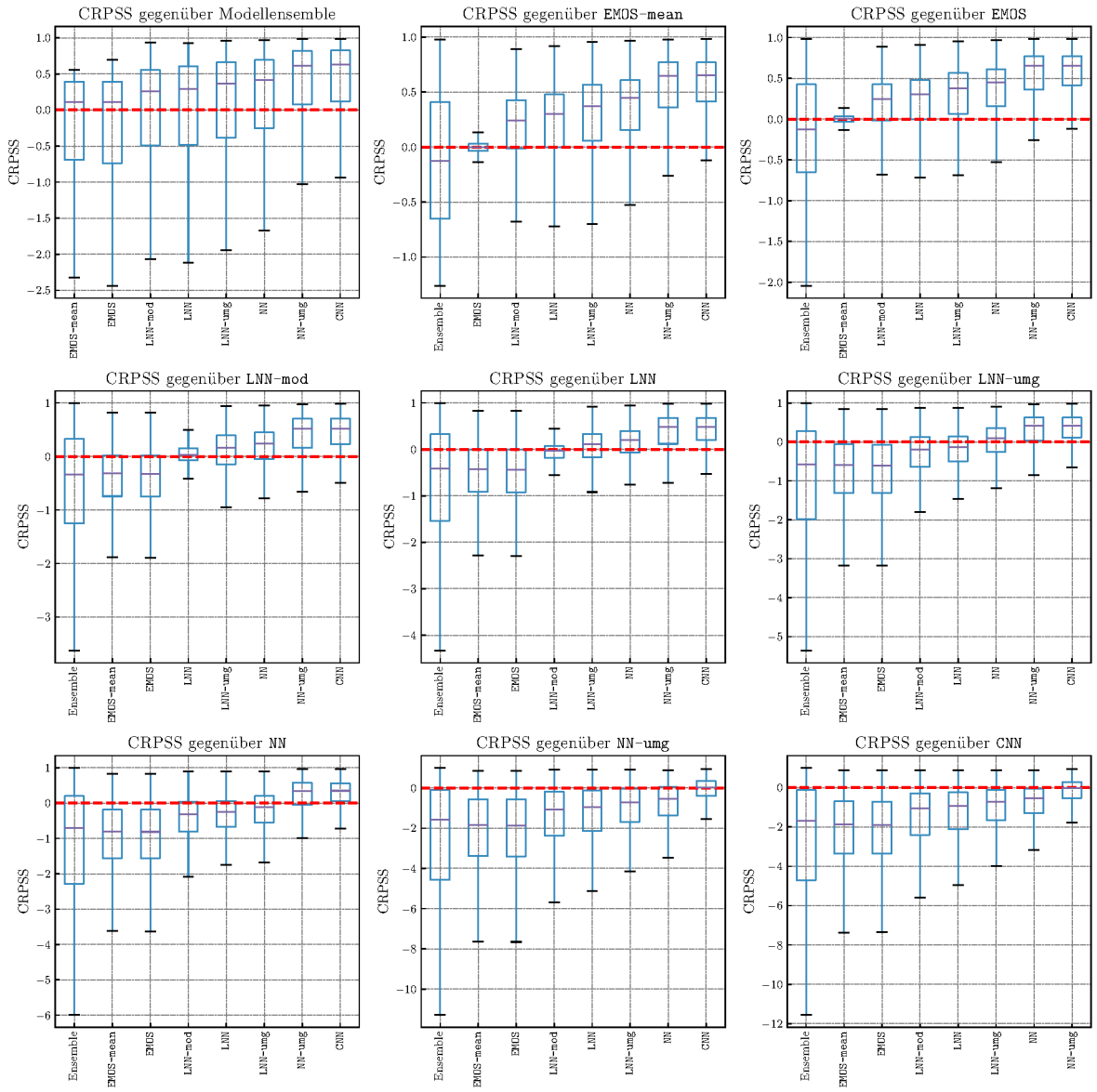

Abbildung 4.6.: Boxplots zu den CRPSS-Werten (ohne Ausreißer)

5. Fazit

Das Ensemble des WRF konnte erfolgreich mit tiefen neuronalen Netzen kalibriert werden. Die unterdispersiven Vorhersagen des Modells konnten durch die statistische Nachbearbeitung mit allen eingesetzten Verfahren verbessert werden. Die Vergleichsverfahren `EMOS-mean` und `EMOS` neigen dazu, die Varianz des Ensembles zu groß zu schätzen. Die neuronalen Netze wurden aufgrund von Instabilitäten in den Parameterschätzungen je als Ensemble trainiert und zeigen sowohl beim CRPS als auch bei den PIT-Histogrammen bessere Eigenschaften als das Modellensemble und die Varianten des EMOS-Verfahrens. Bei den neuronalen Netzen wird deutlich, dass die Hinzunahme der räumlichen Modellumgebungen der Messstationen zu besseren CRPS-Werten und damit zu besseren Vorhersagen führen konnte. Beim `CNN` konnte der CRPS verglichen zum Modell im Mittel etwa gedrittelt werden. In den PIT-Histogrammen zeigen sich bei `CNN` sowie `NN-umg` erhöhte äußere Balken. Das Histogramm zu `LNN-umg` weist optisch fast eine Gleichverteilung auf. Bei den Verfahren `NN-umg` und `CNN` konnte anhand der Lernkurven Overfitting festgestellt werden. Dies könnte in Zukunft durch Nutzung eines größeren Datensatzes oder die Einführung von Regularisierung verringert werden.

Zur genaueren Untersuchung der räumlichen Muster müssen weitere Experimente durchgeführt werden. Es könnte dabei untersucht werden, welchen Einfluss die Größe der verwendeten Modellumgebungen auf die Ergebnisse der Kalibrierung hat. Außerdem könnten weitere Vergleichsverfahren wie Boosting-Varianten des EMOS-Verfahrens oder Quantile Regression Forest implementiert werden, um einen besseren Vergleich der Verfahren zu ermöglichen. Darüber hinaus kann das Training der neuronalen Netze weiterhin optimiert werden. Um die zeitliche Reihenfolge der Daten besser zu berücksichtigen, könnte beispielsweise ein Zeitfenster eingeführt werden, wodurch zur Verbesserung der Vorhersagequalität eines bestimmten Tages nur Daten aus der Vergangenheit im Training eingesetzt werden. Zudem wäre es denkbar, zu diesem Zweck neue Netzstrukturen wie rekurrente neuronale Netze einzusetzen. Um die Kalibrierung auf beliebige Variablen zu erweitern, könnte ein neuronales Netz zudem eine Quantilregression durchführen, um nicht an eine zuvor festgelegte Wahrscheinlichkeitsverteilung gebunden zu sein und somit die Kalibrierung auf alle meteorologischen Variablen anwenden zu können.

A. Weitere Ergebnisse

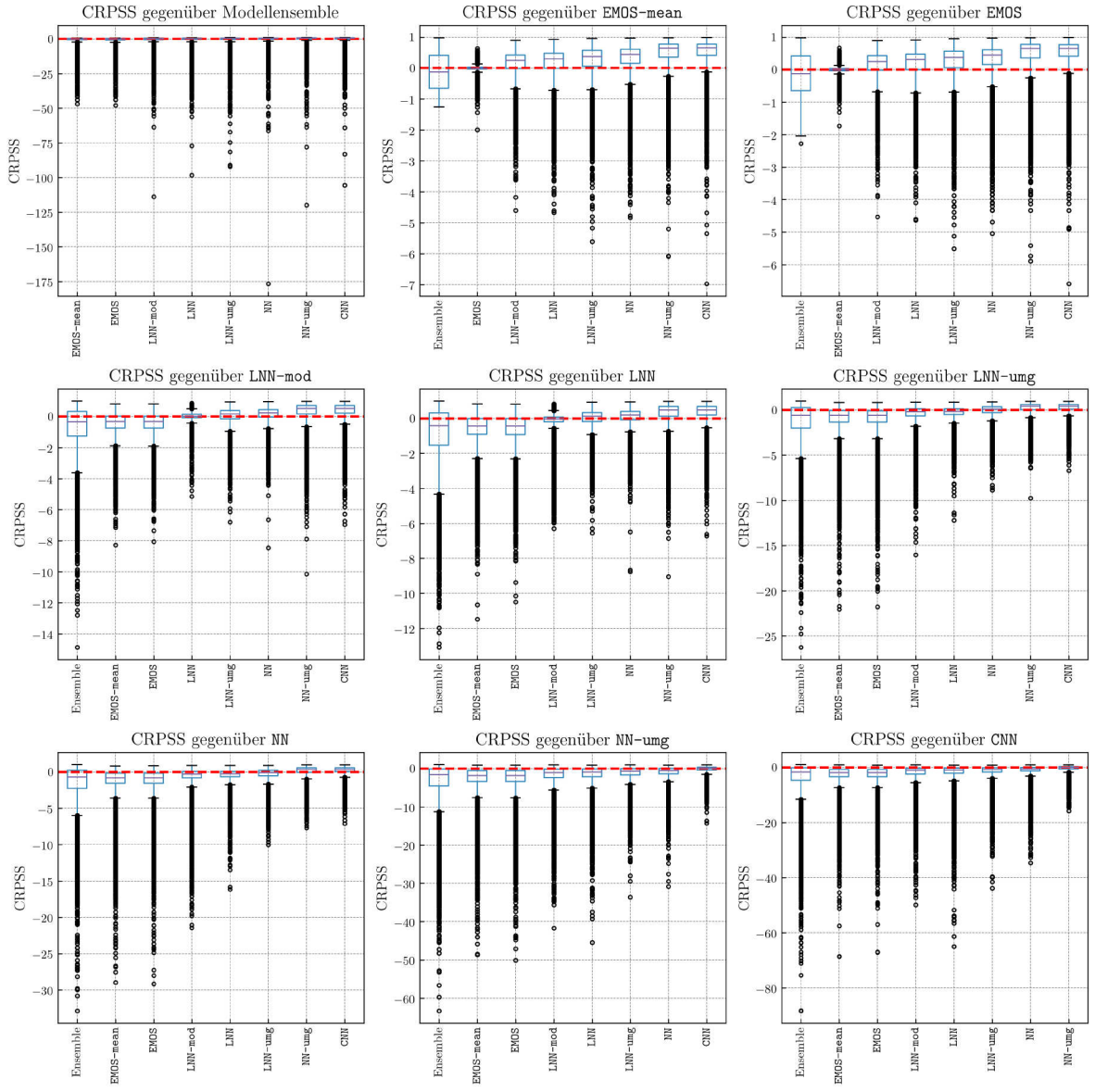

Abbildung A.1.: Boxplots zu den CRPSS-Werten (mit Ausreißern)

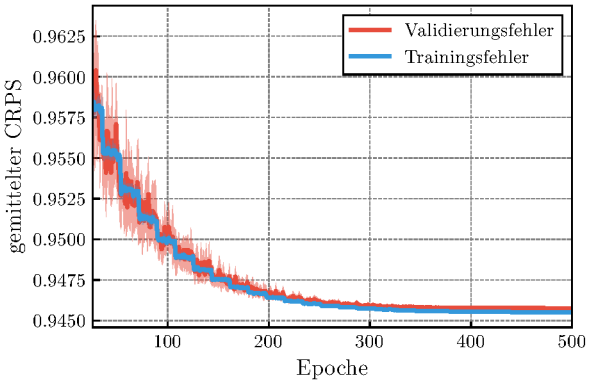

Abbildung A.2.: Lernkurven von `LNN-mod`

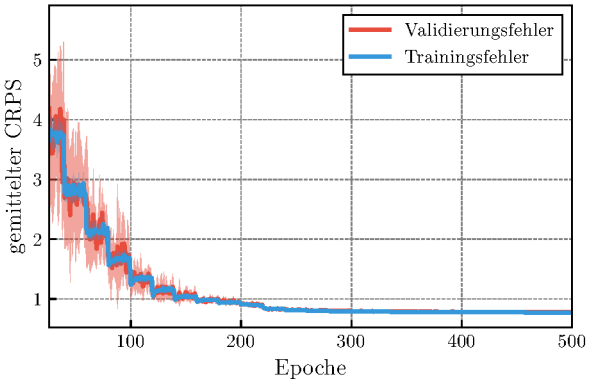

Abbildung A.3.: Lernkurven von `LNN-umg`

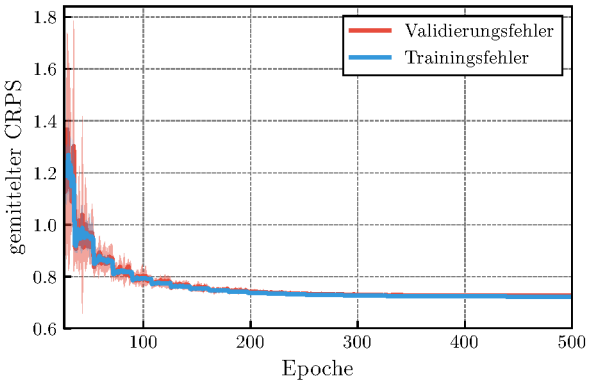

Abbildung A.4.: Lernkurven von `NN`

Tabelle A.1.: Implementierungsdetails zu den neuronalen Netzen. Angaben bzgl. der Lernraten gemäß Gleichung (3.27). Die eingeklammerte Neuronenanzahl beim `CNN` gilt nur für die Stationsinformationen

Verfahren	**Struktur**	$\mathbf{lr_0}$	**drop**	**epochs_drop**	**Epochen**
`LNN-mod`		0,1	0,75	18	500
`LNN`		0,1	0,75	18	500
`LNN-umg`		0,01	0,75	25	500
`NN`	Neuronen: 70	0,1	0,75	18	500
`NN-umg`	Neuronen: 70, 30	0,01	0,75	25	500
`CNN`	2 Faltungsschichten (3×3-Kernel, 32 Filter); Neuronen: (10,) 256	0,005	0,6	50	500

Literatur

[Aba+15] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (besucht am 31. 07. 2019).

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning*. 2. Aufl. The MIT Press, 2010. ISBN: 978-0-26-201243-0.

[Ber18] Jonas Berndt. „On the predictability of exceptional error events in wind power forecasting —an ultra large ensemble approach—". Englisch. Dissertation. Universität zu Köln, 2018. URL: https://kups.ub.uni-koeln.de/9098/ (besucht am 12. 08. 2019).

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Englisch. Springer Science+Business Media, 2006.

[BL15] Sándor Baran und Sebastian Lerch. „Log-normal distribution based EMOS models for probabilistic wind speed forecasting". Englisch. In: *Quarterly Journal of the Royal Meteorological Society* 141.691 (2015), S. 2289–2299. DOI: 10.1002/qj.2521. URL: https://arxiv.org/abs/1407.3252 (besucht am 17. 07. 2019).

[BL18] Sándor Baran und Sebastian Lerch. „Combining predictive distributions for statistical post-processing of ensemble forecasts". Englisch. In: *International Journal of Forecasting* 34 (Juni 2018), S. 477–496. DOI: 10.1016/j.ijforecast.2018.01.005. URL: https://www.researchgate.net/publication/305683521_Combining_predictive_distributions_for_statistical_post-processing_of_ensemble_forecasts (besucht am 23. 08. 2019).

[Bre+84] Leo Breiman u. a. *Classification and Regression Trees*. Englisch. Belmont: Wadsworth International Group, 1984.

[Bre01] Leo Breiman. „Random Forests". Englisch. In: *Machine Learning* 45.1 (2001), S. 5–32. ISSN: 1573-0565. URL: https://link.springer.com/article/10.1023/A:1010933404324 (besucht am 07. 08. 2019).

[Bre96] Leo Breiman. „Bagging Predictors". Englisch. In: *Machine Learning* 24.2 (Aug. 1996), S. 123–140. ISSN: 0885-6125. DOI: 10.1023/A:1018054314350. URL: http://dx.doi.org/10.1023/A:1018054314350 (besucht am 13. 08. 2019).

[Bro70] C. G. Broyden. „The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations“. Englisch. In: *Journal of Applied Mathematics* 6 (1 1. März 1970), S. 76–90. DOI: https://doi.org/10.1093/imamat/6.1.76. URL: https://academic.oup.com/imamat/article/6/1/76/746016 (besucht am 17. 07. 2019).

[Cho+15] François Chollet u. a. *Keras*. 2015. URL: https://keras.io (besucht am 31. 07. 2019).

[Dal+11] L. Dalcin u. a. „Parallel Distributed Computing using Python“. Englisch. In: *Advances in Water Resources* 34 (9 2011), S. 1124–1139. DOI: http://dx.doi.org/10.1016/j.advwatres.2011.04.013.

[DHS11] John Duchi, Elad Hazan und Yoram Singer. „Adaptive Subgradient Methods for Online Learning and Stochastic Optimization“. Englisch. In: *Journal of Machine Learning Research* 12 (2011), S. 2121–2159. ISSN: 1532-4435. URL: http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf (besucht am 30. 07. 2019).

[DMC] Ian Dewancker, Michael McCourt und Scott Clark. *Bayesian Optimization Primer*. URL: https://app.sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf (besucht am 10. 12. 2018).

[Eum] Eumetnet, Hrsg. *The Continuous Rank Probability Score (CRPS): Definition and use*. Englisch. URL: https://www.met-learning.eu/pluginfile.php/5277/mod_resource/content/6/www/english/msg/ver_prob_forec/uos3b/uos3b_ko1.htm (besucht am 06. 07. 2019).

[FS96] Yoav Freund und Robert E. Schapire. „Game Theory, On-line Prediction and Boosting“. Englisch. In: *Proceedings of the Ninth Annual Conference on Computational Learning Theory* (1996). URL: http://www.cs.cmu.edu/~ninamf/LGO10/wm-minimax.pdf (besucht am 13. 08. 2019).

[Gau+16] Laurent Gautier u. a. *Documentation for rpy2*. Englisch. 2016. URL: https://rpy2.readthedocs.io/en/version_2.8.x/ (besucht am 29. 07. 2019).

[GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[GBR07] Tilmann Gneiting, Fadoua Balabdaoui und Adrian E. Raftery. „Probabilistic forecast, calibration and sharpness“. Englisch. In: *Journal of the Royal Statistical Society* 69 (2 2007), S. 234–268. DOI: 10.1111/j.1467-9868.2007.00587.x. URL: https://www.stat.washington.edu/raftery/Research/PDF/Gneiting2007jrssb.pdf (besucht am 12. 06. 2019).

[Gér17] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn und TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems*. Englisch. Hrsg. von Nicole Tache. O'Reilly Media, März 2017. ISBN: 978-1-49-196229-9.

[Gne+04] Tilmann Gneiting u. a. „Calibrated Probabilistic Forecasting Using Ensemble Model Output Statistics and Minimum CRPS Estimation“. Englisch. In: *Monthly Weather Review* 133.5 (21. Sep. 2004), S. 1098–1118. DOI: 10.1175/MWR2904.1. URL: https://journals.ametsoc.org/doi/ref/10.1175/MWR2904.1 (besucht am 02. 04. 2019).

[Ham01] Thomas M. Hamill. „Interpretation of Rank Histograms for Verifying Ensemble Forecast“. Englisch. In: *Monthly Weather Review* 129 (März 2001), S. 550–559. DOI: https://doi.org/10.1175/1520-0493(2001)129<0550:IORHFV>2.0.CO;2. URL: https://journals.ametsoc.org/doi/full/10.1175/1520-0493%282001%29129%3C0550%3AIORHFV%3E2.0.CO%3B2 (besucht am 18. 07. 2019).

[Hin12] Geoffrey Hinton. *Neural Networks for Machine Learning*. Englisch. Foliensatz. 2012. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (besucht am 30. 07. 2019).

[Hor12] Kurt Hornik. „The Comprehensive R Archive Network“. Englisch. In: *WIREs Computational Statistics* 4.4 (2012), S. 394–398. DOI: 10.1002/wics.1212.

[JOP+01] Eric Jones, Travis Oliphant, Pearu Peterson u. a. *SciPy: Open source scientific tools for Python*. 2001. URL: http://www.scipy.org/ (besucht am 31. 07. 2019).

[Jül16] Forschungszentrum Jülich, Hrsg. *PCP Pilot Systems*. Englisch. 2016. URL: https://trac.version.fz-juelich.de/hbp-pcp/wiki/Public (besucht am 12. 08. 2019).

[Jül18] Forschungszentrum Jülich, Hrsg. *Institut für Energie- und Klimaforschung. Troposphäre (IEK-8)*. 2018. URL: www.fz-juelich.de/iek/iek-8/ (besucht am 30. 11. 2018).

[KB14] Diederik P. Kingma und Jimmy Lei Ba. *Adam: A Method for Stochastic Optimization*. Englisch. 2014. URL: https://arxiv.org/pdf/1412.6980v8.pdf (besucht am 30. 07. 2019).

[LeC+89] Y. LeCun u. a. „Backpropagation Applied to Handwritten Zip Code Recognition“. Englisch. In: *IEEE Communications Magazine* 27 (11 1989), S. 41–46. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf (besucht am 07. 08. 2019).

[Med] European Centre for Medium-Range Weather Forecasts, Hrsg. *Operational configurations of the ECMWF Integrated Forecasting System (IFS)*. Englisch. URL: https://www.ecmwf.int/en/forecasts/documentation-and-support (besucht am 19. 08. 2019).

[Mei06] Nicolai Meinshausen. „Quantile Regression Forests“. Englisch. In: *Journal of Machine Learning Research* 7 (2006), S. 983–999. URL: http://www.jmlr.org/papers/volume7/meinshausen06a/meinshausen06a.pdf (besucht am 07. 08. 2019).

[MMZ17] Jakob W. Messner, Georg J. Mayr und Achim Zeileis. „Nonhomogeneous Boosting for Predictor Selection in Ensemble Postprocessing“. Englisch. In: *Monthly Weather Review* 145.1 (2017), S. 137–147. DOI: 10.1175/MWR-D-16-0088.1. URL: https://doi.org/10.1175/MWR-D-16-0088.1 (besucht am 17. 07. 2019).

[NG96] Klas Nordberg und Gösta H. Granlund. „Equivariance and invariance-an approach based on Lie groups“. Englisch. In: *Proceedings of 3rd IEEE International Conference on Image Processing* 3 (1996), 181–184 vol.3. DOI: 10.1109/ICIP.1996.560414.

[Oli06] Travis E Oliphant. *A guide to NumPy*. Bd. 1. Trelgol Publishing USA, 2006. URL: https://numpy.org/ (besucht am 31. 07. 2019).

[Pro] Human Brain Project, Hrsg. *Human Brain Project*. Englisch. URL: https://www.humanbrainproject.eu/ (besucht am 12. 08. 2019).

[RHW86] David E. Rumelhart, Geoffrey Hinton und Ronald J. Williams. „Learning representations by back-propagating errors“. Englisch. In: *Nature* 323 (9. Okt. 1986), S. 533–536. URL: https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf (besucht am 07. 08. 2019).

[RL08] Computational Research Data Archive at the National Center for Atmospheric Research und Information Systems Laboratory, Hrsg. *NCEP ADP Global Upper Air and Surface Weather Observations*. Boulder CO, 2008. DOI: https://doi.org/10.5065/Z83F-N512.

[RL18] Stephan Rasp und Sebastian Lerch. „Neural networks for post-processing ensemble weather forecasts“. Englisch. In: *Monthly Weather Review* 146.11 (24. Mai 2018), S. 3885–3900. DOI: 10.1175/MWR-D-18-0187.1. URL: https://arxiv.org/abs/1805.09091 (besucht am 02. 04. 2019).

[SH15] Michael Scheuerer und Thomas M. Hamill. „Statistical Postprocessing of Ensemble Precipitation Forecasts by Fitting Censored, Shifted Gamma Distributions“. Englisch. In: *Monthly Weather Review* 143 (2015), S. 4578–4596. DOI: https://doi.org/10.1175/MWR-D-15-0061.1. URL: https://journals.ametsoc.org/doi/suppl/10.1175/MWR-D-15-0061.1 (besucht am 17. 07. 2019).

[Sha+16] Bobak Shahriari u. a. „Taking the Human Out of the Loop: A Review of Bayesian Optimization“. In: *Proceedings of the IEEE* 104 (1 Jan. 2016). DOI: 10.1109/JPROC.2015.2494218.

[Sri+14] Nitish Srivastava u. a. „Dropout: A Simple Way to Prevent Neuroal Networks from Overfitting“. Englisch. In: *Journal of Machine Learning Research* 15 (Juni 2014), S. 1929–1958. URL: http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf (besucht am 05. 08. 2019).

[Tai+16] Maxime Taillardat u. a. „Calibrated Ensemble Forecast Using Quantile Regression Forests and Ensemble Model Output Statistics“. Englisch. In: *Monthly Weather Review* 144 (Juni 2016), S. 2375–2393. DOI: https://doi.org/10.1175/MWR-D-15-0260.1. URL: https://journals.ametsoc.org/doi/pdf/10.1175/MWR-D-15-0260.1 (besucht am 08. 08. 2019).

[Tai+17] Maxime Taillardat u. a. *Forest-based methods and ensemble model output statistics or rainfall ensemble forecasting*. Englisch. 2017. DOI: 10.1175/WAF-D-18-0149.1. URL: https://arxiv.org/abs/1711.10937 (besucht am 19. 08. 2019).

[Tea15] Python Core Team, Hrsg. *Python. A dynamic, open source programming language*. Englisch. Python Software Foundation. 2015. URL: https://www.python.org/ (besucht am 20. 08. 2019).

[TG09] S. Theis und C. Gebhardt. „Grundlagen der Ensembletechnik und Wahrscheinlichkeitsaussagen“. In: *promet* 35 (1–3 2009): *Moderne Verfahren und Instrumente der Wettervorhersage im Deutschen Wetterdienst*. Hrsg. von DWD, S. 104–110. ISSN: 0340-4552. URL: https://www.dwd.de/DE/leistungen/pbfb_verlag_promet/pdf_promethefte/35_1_3_pdf.pdf?__blob=publicationFile&v=3 (besucht am 06. 07. 2019).

[TVS97] *Evaluation of probabilistic prediction systems*. Englisch. Workshop on Predictability (Shinfield Park, Reading, 20. Okt. 1997). ECMWF, 1997. URL: https://www.ecmwf.int/node/12555 (besucht am 21. 08. 2019).

[UCA18] UCAR, Hrsg. *Weather Research and Forecasting Model*. Englisch. 2018. URL: https://www.mmm.ucar.edu/weather-research-and-forecasting-model (besucht am 17. 12. 2018).

[Whi] Jeff Whitaker. *netCDF4 module*. Englisch. URL: https://unidata.github.io/netcdf4-python/netCDF4/index.html (besucht am 20. 08. 2019).

[Wol92] David H. Wolpert. „Stacked generalization“. Englisch. In: *Neural Networks* 5 (2 1992), S. 241–259. DOI: https://doi.org/10.1016/S0893-6080(05)80023-1.

[Yue+18] RA Yuen u. a. *ensembleMOS. Ensemble Model Output Statistics*. Englisch. 22. März 2018. URL: https://cran.r-project.org/package=ensembleMOS (besucht am 29. 07. 2019).