Genetische Algorithmen zur Hyperparameteroptimierung künstlicher neuronaler Netze für die Energiezeitreihenprognose

Tobias M. Fischer, Fabian Bauer, Silas A. Selzer, Peter Bretschneider

Fachgebiet Energieeinsatzoptimierung
Technische Universität Ilmenau
Gustav-Kirchhoff-Straße 5
98693 Ilmenau
E-Mail: {tobias-merlin.fischer, fabian.bauer, silas-aaron.selzer, peter.bretschneider}@tu-ilmenau.de

1 Einleitung

In der Energieversorgung spielen exakte Vorhersagen der Verbrauchslastgänge eine wesentliche Rolle für die optimale Planung und Steuerung des elektrischen Energiesystems. Aufgrund der Digitalisierung des Elektroenergiesystems wird die Anzahl der messtechnisch erfassten Lastgänge und exogenen Einflussgrößen signifikant ansteigen. [1] Zur möglichst genauen Vorhersage ist es notwendig, Modelle zu entwickeln und zu untersuchen, die dem wachsenden Mengengerüst Rechnung tragen und aufgrund der signifikant höheren Anzahl von Beobachtungen zu möglichst verbesserten Modellen führen. Maschinelles Lernen ist eine Anwendung der künstlichen Intelligenz, die Systeme in die Lage versetzt, aus Daten selbständig zu lernen. Bei Verfahren des maschinellen Lernens ist ein entscheidender Faktor die Optimierung der Hyperparameter. Hierzu werden im Rahmen der Arbeit die beiden Verfahren Zufallssuche und genetische Algorithmen untersucht. Dabei zeigen die, auf den Evolutionsprinzipien beruhenden genetischen Algorithmen bei der Suche nach dem globalen

DOI: 10.58895/ksp/1000138532-16 erschienen in:
Proceedings - 31. Workshop Computational Intelligence : Berlin, 25. - 26. November 2021
DOI: 10.58895/ksp/1000138532 | https://www.ksp.kit.edu/site/books/m/10.58895/ksp/1000138532/

Optimum nichtlinearer Problemstellungen vielversprechende Ergebnisse. Inden durchgeführten Untersuchungen werden Verfahren genetischer Algorithmen angewandt und mit gängigen Methoden verglichen.[2]

2 Hyperparameter

Künstliche neuronale Netze sind informationsverarbeitende Systeme und nutzen zum Training der Netzparameter Methoden des maschinellen Lernverfahrens. Jedes KNN verfügt über Hyperparameter und eine der grundlegendsten Aufgaben der Modelloptimierung ist es, die optimale Hyperparameterkombination über einen definierten Suchraum zu finden. Als Hyperparameter werden alle Parameter bezeichnet, die sich während eines Trainingszyklus des KNN konstant gehalten werden. Hierzu zählen bspw. die Anzahl der Schichten, die Anzahl der Knoten je Schicht oder auch die verwendeten Aktivierungsfunktionen. Besonders in den immer komplexer werdenden KNN-Architekturen ist die richtige Auswahl der Hyperparameter von entscheidender Bedeutung. So lässt sich mit einer effektiven Hyperparameteroptimierung der notwendige Rechenaufwand zum Trainieren des KNN verringern bei gleichzeitiger Steigerung der Performance der Modelle. [2, 3]
Die Frage, wie die verfügbaren Rechenkapazitäten effizient eingesetzt und die Suchräume effektive bearbeitet werden können, führte zu einer Vielzahl von Methoden der Hyperparameteroptimierung. Eine automatisierte Hyperparameteroptimierung bietet dabei mehrere Vorteile. Hyperparameterräume sind häufig komplex, bestehend aus einer Vielzahl von kontinuierlichen, diskreten und kategorischen Hyperparametern und werden deshalb häufig nur abgeschätzt oder heuristisch ermittelt. Dadurch besteht das Risiko, dass die Hyperparameter nicht vollständig optimiert sind und sich in einem lokalen Minimum befinden. Ebenso sind die Wechselwirkungen der Hyperparameter zumeist unbekannt. Durch die algorithmische Ermittlung der Hyperparameter können Unsicherheiten und Eingriffe des menschlichen Beobachters reduziert werden. In der Vergangenheit haben sich eine Vielzahl automatisierter Methoden zu Hyperparameteroptimierung entwickelt. [4, 5]

3 Untersuchte Optimierungsansätze

Die nachfolgenden Problemherleitung der Hyperparameteroptimierung orientiert sich an [6, 7]. Mit $X \subseteq \mathbb{R}^{d_x}$ als Eingaberaum und $Y \subseteq \mathbb{R}^{d_y}$ als Ausgaberaum ergibt sich als Ziel des überwachten Lernens eine Funktion h mit $h(x;\theta) : X \rightarrow Y$, wobei $x \in X$ gilt und h aus einer Schar von Funktionen parametrisiert mit $\theta \in \mathbb{R}^d$ kommt. Im Folgenden wird θ als die Bezeichnung für die Hyperparameter verwendet. Unter Hinzunahme einer Verlustfunktion

$$l(h(x;\theta);y) \tag{1}$$

kann eine Risikofunktion

$$R(\theta) = E_{x,y}[l(h(x;\theta);y)] \tag{2}$$

für ein gegebenes θ als der zu erwartende Fehler über die zugrunde liegende Wahrscheinlichkeitsverteilung $P(x,y)$ definiert werden. Mit Hilfe einer Anzahl μ der möglichen künstlichen neuronalen Netze und den jeweils zugehörigen Lösungen $\{\theta_j\}_{j=1}^{\mu}$ ergibt sich ein Minimierungsproblem der Funktion

$$J_\mu = \frac{1}{\mu}\sum_{j=1}^{\mu} R_n(\theta_j) = \frac{1}{\mu}\sum_{j=1}^{\mu}\left(\frac{1}{n}\sum_{i=1}^{n} l_i(\theta_j)\right), \tag{3}$$

welche den erwarteten durchschnittlichen empirischen Fehlers angibt.

3.1 Rastersuche

Die Rastersuche versucht dieses Minimierungsproblem zu lösen, indem jedem Hyperparameter eine Auswahl diskreter Werte, oder Vielfache dieser Werte, vorgegeben wird und der Algorithmus evaluiert das kartesische Produkt dieser Menge. Zur Rastersuche werden ebenso heuristische Verfahren gezählt, wie etwa die Auswahl der Hyperparameter nach Erfahrungswerten. Bei h Hyperparametern mit n Werten wächst die Anzahl der sequenziell verarbeiteten Trainings- und Bewertungsversuche mit h^n.

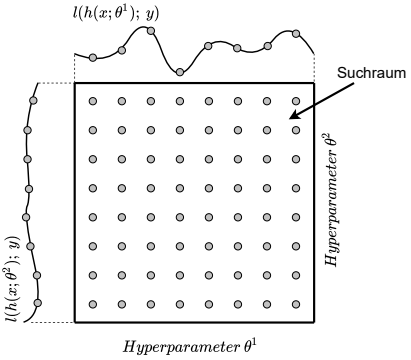

Bild 1: Darstellung der Suchraumabdeckung bei Anwendung der Rastersuche mit zwei Hyperparametern

Die Rastersuche ist deshalb nur für Anwendungen rentabel mit drei oder weniger Hyperparametern, hat jedoch den Vorteil einer gleichverteilten Suchraumabdeckung (siehe Abbildung 1). [3, 5]

3.2 Zufallssuche

Eine Erweiterung bilden zufallsbasierte Methoden der Hyperparameteroptimierung. Die Zufallssuche entspricht dem Vorgehen einer Monte-Carlo Simulation. [8] Im diskreten Hyperparameterraum werden mittels einer Randverteilung zufällige Hyperparameterkombinationen ausgewählt (siehe Abbildung 2). Im Gegensatz zur Rastersuche wird bei der Zufallssuche eine definierte Anzahl von Parameterkombinationen aus der spezifizierten Verteilung gezogen, was die Wahrscheinlichkeit verringert, viel Berechnungszeit in einem niedrig performanten Bereich des Hyperparameterraums aufzuwenden. [3] In einer unlimitierten Suche entspricht der Suchraum der Zufallssuche einer vollständigen Rastersuche, da jede Hyperparameterkombination mindestens einmal vorkommt und konvergiert damit automatisch zum globalen Optimum. In der Praxis bestehen allerdings limitierenden Faktoren wie die Rechenkapazität und -zeit, woraus der Vorteil der Zufallssuche resultiert. Bei nicht gleichverteilten Hyperparametern ermöglicht die Zufallssuche eine bessere Suchraumab-

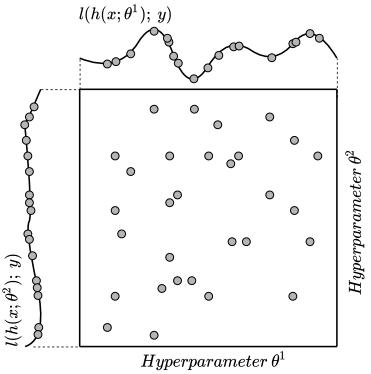

Bild 2: Darstellung der Suchraumabdeckung bei Anwendung der Zufallssuche mit zwei Hyperparametern

deckung als vergleichbare Optimierungsmethoden und steigert so die Wahrscheinlichkeit über die Optimierungszeit das globale Optimum zu finden. Als Nachteil ist zu nennen, dass die Zufallssuche ebenso wie die Rastersuche eine gewisse Anzahl unnötiger Evaluierungen durchführt, da performante Bereiche des Hyperparameterraumes nicht tiefergehend untersucht werden. [4, 5]

3.3 Genetischer Algorithmus

Die Konzeption und Struktur des in dieser Arbeit verwendeten genetischen Algorithmus orientiert sich an [7]. Für eine tiefergehende Analyse des Aufbaus verschiedener genetischer Algorithmen wird der interessierte Leser auf [3], [6] und [9] verwiesen. Die zentralen Themenfelder der Hyperparameteroptimierung sind eine möglichst große Suchabdeckung im Hyperparameterraum sowie die tiefergehende Untersuchung erschlossener lokaler Optima. Als Nebenbedingung sind die Optimierungszeit und Rechenkapazität aufzuführen. Die Zufallssuche, als Weiterentwicklung der Rastersuche, löst zwar das Problem der Suchraumabdeckung, bietet aber keine Möglichkeit regelnder Maßnahmen während der Optimierung.

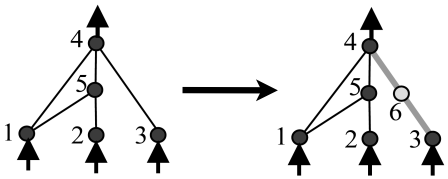

Bild 3: Mutation durch Entstehung eines neuen Neurons [10]

Der genetische Algorithmus löst diesen Zielkonflikt der Hyperparameteroptimierung, indem eine absteigende Einstufung der Verlustfunktionen

$$l\left(\theta_{1:\mu}\right) \leq l\left(\theta_{2:\mu}\right) \leq \ldots \leq l\left(\theta_{\mu:\mu}\right). \tag{4}$$

über die zufällig geschaffene Population $\Psi_\mu = \left\{\theta_1, \ldots, \theta_\mu\right\}$ mit $\theta_{k:\mu}$ als die k-besten Individuen und der Anzahl der Individuen μ erstellt wird.

Für ein Individuum ergeben sich alle Möglichkeiten an Hyperparametern innerhalb des Suchraumes

$$\Psi_\mu^h = \begin{pmatrix} \theta_1^1 & \cdots & \theta_\mu^1 \\ \vdots & \ddots & \vdots \\ \theta_1^h & \cdots & \theta_\mu^h \end{pmatrix}, \tag{5}$$

dargestellt als Matrix, wobei θ_μ^h den h-ten Hyperparameter des μ-ten Individuum bezeichnet. Für die weiteren Berechnungen wird zur Vereinfachung weiterhin Ψ als die Menge über die Population definiert. Zur Minimierung von J_μ verwendet der genetische Algorithmus Mutation, Selektion und Kreuzung. Die Mutation beschreibt die Wahrscheinlichkeit, dass ein Hyperparameter zufällig innerhalb seines Suchraumes verändert wird, siehe Abbildung 3.

Formal kann die Mutation durch die Vereinigung zweier Untermengen, der Menge der zufällig geschaffenen Ausgangspopulation

$$\Psi_{\mu,neu} = \Psi_{m\mu:\mu,alt} \cup \Psi_{1:m\mu,mutiert} \tag{6}$$

beschrieben werden, wobei m die Mutationsrate, $\Psi_{m\mu:\mu,alt}$

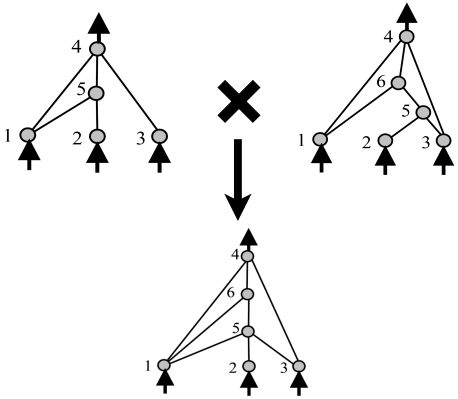

Bild 4: Die Kreuzung zweier Eltern zu einem Ableger [10]

den nicht-mutierten und $\Psi_{1:m\mu,mutiert}$ den mutierten Anteil der zufälligen Ausgangspopulation markiert. Jedoch ist zu beachten, dass die Mächtigkeit der Populationsmenge

$$\left|\Psi_{\mu,neu}\right| = \left|\Psi_{m\mu:\mu,alt}\right| + \left|\Psi_{1:m\mu,mutiert}\right| \tag{7}$$

erhalten bleibt.[7]

Als Selektion wird die Auswahl der ersten k-besten Individuen aus der Abfolge der Verlustfunktionen bezeichnet. Die Kreuzung stellt die Vermehrung von Genen innerhalb der Population dar. Aus jedem ausgewählten Individuenpaar werden zwei Nachfolger mittels der zufälligen Rekombinationsrate r erstellt. Abbildung 4 stellt die Kreuzung beispielhaft für zwei Eltern und einen Ableger dar. Für den genetischen Algorithmus ist es wichtig, dass die Gene vollständig und komplementär an die Ableger abgegeben werden, d.h. es gilt:

$$\left|\Psi^{h}_{\mu,kid1}\right| = \left|\Psi^{rh:h}_{\mu,parent1}\right| + \left|\Psi^{rh}_{\mu,parent2}\right| \tag{8}$$

sowie

$$\left|\Psi^{h}_{\mu,kid2}\right| = \left|\Psi^{rh:h}_{\mu,parent2}\right| + \left|\Psi^{rh}_{\mu,parent1}\right|. \tag{9}$$

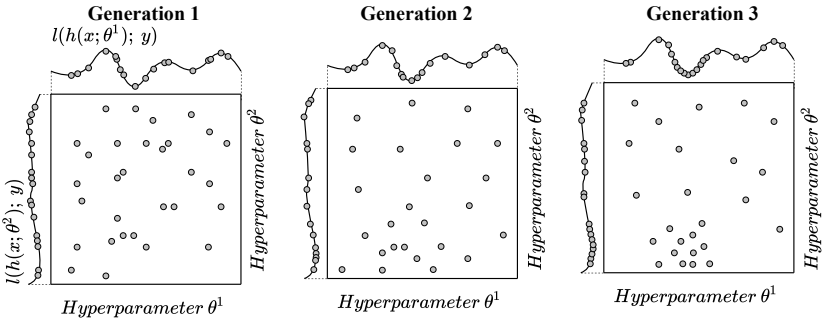

Bild 5: Darstellung der Suchraumabdeckung und Entwicklung der Hyperparametersuche über drei Generation mit genetischem Algorithmus für zwei Hyperparameter

Eine systematische Darstellung der Suchraumabdeckung und die Entwicklung der Hyperparameter über drei Generationen hinweg ist in Abbildung 5 exemplarisch dargestellt. Ziel des genetischen Algorithmus ist es, mittels der oben genannten evolutionären Techniken die Vorteile einer Zufallssuche zu erhalten, aber die Hyperparameteroptimierung dahingehend zu verbessern, dass die vorhandene Rechenkapazität auf vielversprechende Teil- oder Unterräume konzentriert wird. [7]

4 Modellierung

4.1 Datengrundlage

Die Datengrundlage für die durchgeführten Untersuchungen umfasst plausible und vollständige Energiezeitreihendaten im Umfang von drei Kalenderjahren mit einer Abtastzeit von 15 Minuten. Für die Energielastprognose ist es wichtig die Zusammenhänge zwischen verschiedenen Messgrößen als Muster zu identifizieren, weshalb die Untersuchung sowohl univariat als auch multivariat durchgeführt wurde. Für die multivariate Untersuchung standen zusätzlich die Zeitreihendaten für die Lufttemperatur, der Windgeschwindigkeit und der Solareinstrahlung zur Verfügung, ebenfalls mit einer Auflösung von

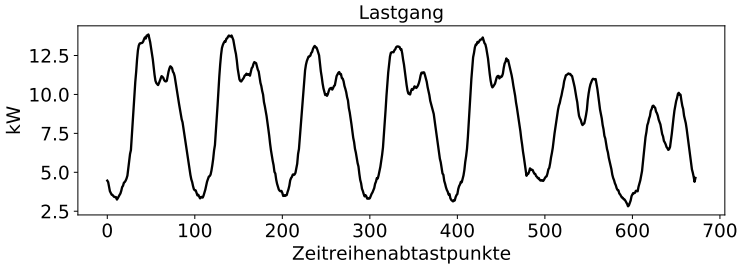

Bild 6: Auszugsweise Darstellung der Energielastzeitreihe über einen Zeitbereich von sieben Tagen

15-Minuten-Mittelwerten. Die Energielastzeitreihe ist auszugsweise in Abbildung 6 dargestellt. Deutlich zu sehen ist der Wochen- und Tageszyklus, wobei 672 Messpunkte einer Woche und 96 einem Tag entsprechen. Der verwendete Datenbestand wurde für das Anlernen der KNN-Modelle in Trainings- und Validierungsdaten aufgeteilt. Der Trainings- und Validierungsdatensatz umfasst die ersten beiden Kalenderjahre. Der anschließende Test der Prognosemodelle wird mit dem dritten Kalenderjahr durchgeführt.

4.2 Modellansatz

Als Modellansatz für die Prognosemodelle werden künstliche neuronale Netze mit den Architekturen Multi-Layer-Perceptron (MLP) und Temporal Convolutional Netzwork (TCN) verwendet.
In der MLP-Architektur bilden die künstlichen Neuronen die funktionellen Grundeinheiten. Sie sind in mehreren Schichten angeordnet und miteinander verbunden (siehe Abbildung 7). Die Schichten unterteilen sich in die Eingangsschicht, verdeckte Schicht und Ausgangsschicht. Die Neuronen einer Schicht sind mit den Neuronen der vorherigen und der nachfolgenden Schicht verbunden. Den Verbindungen zwischen den Neuronen werden Gewichte zugeordnet, welche die Relevanz einer Verbindung kennzeichnen. Die angelegten Signale der Eingabeschicht werden an die direkt verbundenen Knoten weitergegeben. In den Knoten werden die Gewichte aufsummiert, an eine Aktivierungsfunktion übergeben und der Ausgabewert berechnet. Die Informationsverarbeitung

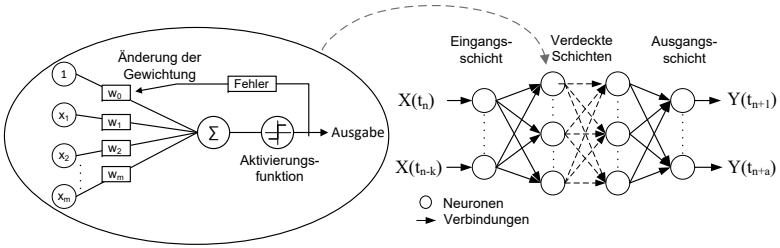

Bild 7: Aufbau eines künstlichen neuronalen Netzes in Anlehnung an [11]

erfolgt somit in den verdeckten Schichten von der Eingangs- zur Ausgangsschicht. Dabei sind die Anzahl der verdeckten Schichten, die Anzahl der enthaltenen Neuronen und die verwendete Aktivierungsfunktion frei zu wählende Hyperparameter bei der Konfiguration des KNN. [2]

Bei der TCN-Architektur hingegen wird die Eingabesequenz nicht aufsummiert, sondern mittels Faltungsoperationen verarbeitet. Der grundlegende Unterschied zwischen MLPs und TCNs ist, dass ein TCN nicht mehr nur aus Neuronen, sondern aus residualen Blöcken (RB) besteht. Diese Blöcke weisen eine eigene Netzstruktur auf, in der die Eingangssequenz zuerst eine Faltung (Dilated Causal Conv) durchläuft. Nach der Faltung erfolgt eine Gewichtsnormalisierung, darauffolgend die Verarbeitung über die Aktivierungsfunktion und zuletzt eine Regularisierung über das Dropout Verfahren. Die Gewichtsnormalisierung dient zur Rechenzeitbeschleunigung sowie dazu, explodierende Werte zu verhindern. Da sich die Struktur der Neuronen eines TCN von denen eines MLP unterscheiden, wird auch von Filtern gesprochen. Zur Bildung eines TCNs werden die residualen Blöcke aufeinandergestapelt. In Abbildung 8 ist der Aufbau eines zweischichtigen residualen Blocks zu sehen. Sollte die Eingabesequenz eine Länge unterschiedlich von der Ausgabe der Faltung haben, wird optional auf die Eingabesequenz eine 1x1 Faltung angewendet, die diese Differenz aufhebt. Durch die Faltung der Eingangsdaten können eine große Anzahl an Vergangenheitswerten berücksichtigt werden, wodurch sich TCNs besonders zur Zeitreihenprognose eignen. [12]

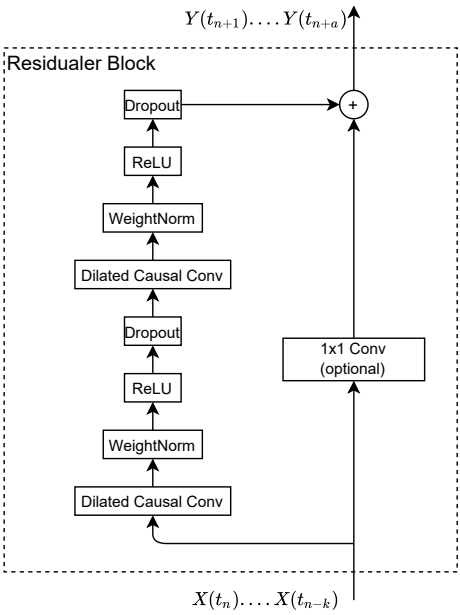

Bild 8: Aufbau eines residualen Blocks eines Temporal Convolutional Netzwork in Anlehnung an [12]

Für das Trainieren der beiden KNN-Modelle, also dem Ändern der gewichteten Verbindungen, wird der Backpropagation-Algorithmus als Lernverfahren verwendet. Dabei erfolgt die Korrektur der Netzgewichte mittels Gradientenabstiegsverfahren zur Fehlerminimierung. Für tiefergehende Erläuterungen wird auf [11] und [13] verwiesen.

Für die Lösung des Minimierungsproblems der Hyperparameteroptimierung wird in den nachfolgenden Untersuchungen der entworfene genetische Algorithmus und die Zufallssuche verwendet. Der genetische Algorithmus verwendet eine Anfangspopulation an Modellen und optimiert diese mittels Mutation, Selektion und Kreuzung.

Für alle Untersuchungsszenarien wird ein Day-Ahead Prognosehorizont festgelegt, was einer 36-Stunden-Prognose entspricht (siehe Abbildung 9).

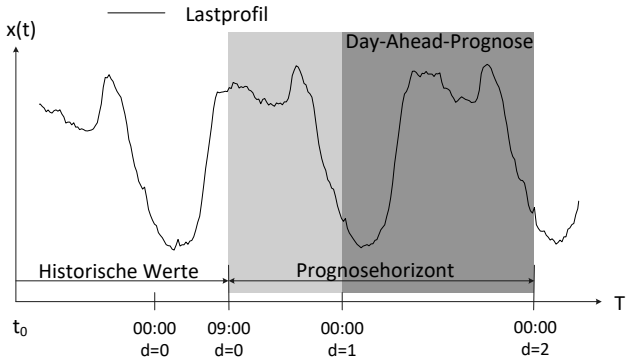

Bild 9: Prognosehorizont für die Untersuchungen

4.3 Hyperparameteroptimierung

Im Rahmen dieses Beitrags werden die Zufallssuche und ein genetischer Algorithmus über den gleichen Hyperparameterraum sowohl für eine univariate als auch eine multivariate Zeitreihenvorhersage verglichen. Als eingesetztes Framework dient Tensorflow [14], wobei die Zufallssuche mittels der Erweiterung Talos [15] durchgeführt wurde. Der Ablauf des genetischen Algorithmus ist in Abbildung 10 dargestellt. Eine zufällig generierte Startpopulation wird fünf Epochen trainiert, um in der Evaluierungsphase eine absteigende Auflistung der Tauglichkeit der Individuen zu erstellen. Von der Startpopulation werden die besten $k = 15$ % der Individuen selektiert. Um der Tatsache Rechnung zu tragen, dass weniger taugliche Modelle eventuell durch geringfügige Mutationen deutlich bessere Ergebnisse erzielen können und damit in der Rangfolge aufsteigen, wird eine Überlebensrate von 10 % eingesetzt. Diese gibt die Wahrscheinlichkeit an, dass ein eigentlich ausselektiertes Modell weiterverwendet wird. Nach der Selektion besteht die Population aus den 15 % besten Individuen und 10 % zufällig ausgewählten. Die Hyperparameter der zufällig ausgewählten Individuen werden mit der Mutationsrate $m = 30$ % zufällig verändert. Zur Erhaltung und Entwicklung der Population werden die besten Individuen zufällig gekreuzt, um Ableger zu rekombinieren. Jeweils zwei Individuen erstellen durch zufällige Rekombination zwei weitere Ableger, wobei der zweite Ableger die Hyperparameter komplementär zum ersten

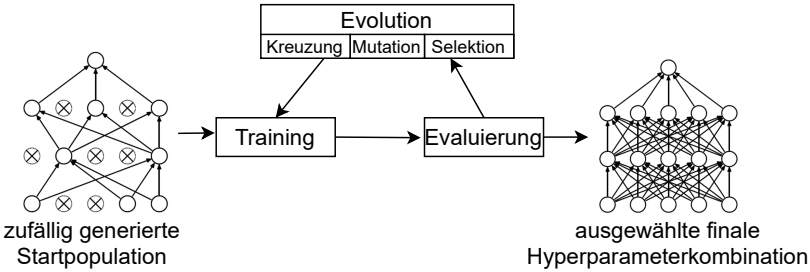

Bild 10: Schematischer Ablauf des genetischen Algorithmus mit Evolutionsstufen [16]

erhält. Die Hyperparameter beider Ableger werden ebenfalls mit $m = 30\ \%$ mutiert. Ein Durchlaufen dieser Schleife wird eine Generation genannt. Der genetische Algorithmus beendet die Hyperparameteroptimierung automatisch nach acht Generationen.

Für eine Speicherplatzoptimierung und Rechenbeschleunigung wird für die Speicherung der Individuen der Genotyp, für die Darstellung der Phänotyp verwendet. Der Genotyp ist die Gesamtheit der Gene eines Individuums bzw. dessen Erbanlagen, der Phänotyp beschreibt die sichtbaren Eigenschaften. Der Genotyp ist eine Liste, Genom genannt, mit allen Hyperparametern und deren Ausprägung eines Individuums. Dies senkt die Zugriffzeiten, da Selektion, Mutation und Kreuzung nicht in der für Tensorflow typischen als Graph dargestellten Netzarchitektur durchgeführt werden müssen. Der Phänotyp entspricht den als final Netzarchitektur dargestellten Hyperparametern, exemplarisch in Abbildung 10 dargestellt. [5, 17]

5 Simulative Untersuchung und Evaluierung

Die in Abschnitt 4 dargestellten und konzipierten Modelle mit den jeweiligen KNN-Architekturen wurden in umfangreichen simulativen Untersuchungen mit den dargestellten Hyperparameteroptimierungsmethoden (vgl. Abschnitt 3) bezüglich der Prognosequalität und der benötigten Rechenzeit miteinander verglichen. Für die Konfiguration des genetischen Algorithmus wurden folgende Einstellungen gewählt:

- Anzahl der Generationen $G = 8$
- Größe der Populationen $\mu = 30$
- Überlebensrate $s = 0,1$
- Selektionsrate $k = 0,15$
- Mutationsrate $m = 0,3$

Zur Sicherstellung der Vergleichbarkeit der Optimierungsergebnisse wird der Umfang bei der Zufallssuche auf 350 Stichproben festgesetzt.

Zur Bewertung der Modellgüte während der Zufallssuche sowie zur absteigenden Sortierung während des genetischen Algorithmus werden alle Modelle fünf Epochen trainiert.

Für die Bewertung werden die Optimierungszeit und die Prognosequalität herangezogen. Die dargestellten Fehlermaße sind der *RMSE* (Wurzel der mittleren Fehlerquadratsumme), *MAPE* (mittlerer absoluter prozentualer Fehler) und der *ME* (mittlere Fehler). Zur Erhöhung der Nachvollziehbarkeit erfolgt die Darstellung der Ergebnisse gegliedert nach der verwendeten Netzarchitektur.

MLPs verfügen als einfache vollverschaltete Netzarchitekturen über keine komplexen inneren Funktionen, wie rekursive Verbindungen oder Regularisierung. Daraus resultiert eine vereinfachte Hyperparameteroptimierung für die Zufallssuche. Hingegen ist die Netzarchitektur von TCNs aus Stapeln von residualen Blöcken aufgebaut. Folglich erhöht sich die Komplexität und Wechselwirkung der Hyperparameterauswahl. Dementsprechend wird die Performance des genetischen Algorithmus an dem einfachen Beispiel der MLP-Architektur getestet und anschließend auf das komplexere Problem der TCN-Architekturen ausgeweitet. [7, 12]

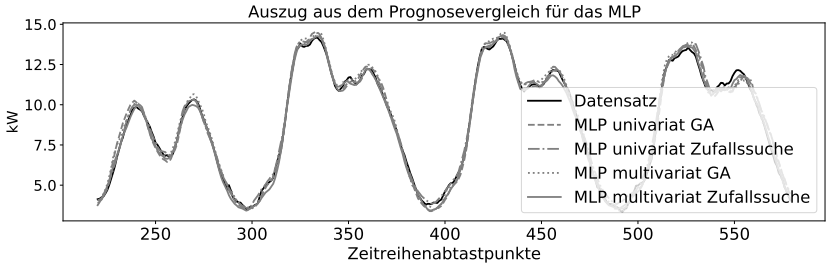

Bild 11: Vergleich der Prognosen der optimierten MLP-Netze anhand eines Ausschnitts der Energiezeitreihe von vier Tagen

5.1 MLP-Architektur

Die Hyperparameter θ werden zur Minimierung der Prognosegüte der Vorhersage mittels MLP-Architektur innerhalb des Suchraums

$$\theta \begin{cases} \text{Verdeckte Schichten} \in [1,\ldots,5] \\ \text{Neuronen} \in [1,\ldots,500] \\ \text{Aktivierungsfunktionen} \in [relu, elu, tanh, selu, sigmoid] \\ \text{Optimierer} \in [rmsprop, adam, sgd, adadelta] \end{cases}$$

variiert, wobei der Suchraum für die verschiedenen Hyperparameteroptimierungsalgorithmen, Zufallssuche und genetischer Algorithmus, identisch gewählt wird.

In Abbildung 11 sind die Prognosen beispielhaft an jeweils einem der optimierten MLP-Netze dargestellt. Die Vorhersagen stimmen zumeist gut mit der Zielzeitreihe überein. Abweichungen der Zeitreihenprognose werden vor allem an den lokalen Minima und Maxima der Energiezeitreihe deutlich.

Zur Abschätzung der Performance der Algorithmen der Hyperparameteroptimierung sind in Tabelle 1 die Prognosegüte anhand verschiedener Fehlermaße sowie die Berechnungszeit aufgetragen. Dabei ist der Durchschnittswert von drei verschiedenen Optimierungsdurchläufen aufgeführt.

Tabelle 1: Mittelwerte der Ergebnisse der unterschiedlichen Hyperparameteroptimierungen für die MLP-Netze

	Univariat		Multivariat	
Methode	Zufallssuche	gen. Algorithmus	Zufallssuche	gen. Algorithmus
RMSE	0,307	0,304	0,3285	0,327
MAPE	3,71	3,69	4,005	3,96
ME	−0,038	−0,033	−0,047	−0,053
Zeit (mm:ss)	45 : 06	33 : 03	56 : 35	52 : 02

Vergleicht man die unterschiedlichen Fehlermaße zunächst für die univariate Zeitreihenprognose, stellt man fest, dass die Fehlerwerte in der gleichen Größenordnung liegen. Dies spricht dafür, dass die beiden Optimierungsmethoden die Umgebung des Minimums des Gütegebirges finden. Es fällt dabei jedoch auf, dass die Vorhersagen der Netze optimiert mit dem genetischen Algorithmus jeweils Verbesserungen zeigen. Der Vorteil der genetischen Algorithmen wird beim Vergleich der benötigten Rechenzeit deutlich. Durch die Verwendung genetischer Algorithmen konnte die Optimierungszeit bei besserer Prognosegüte um mehr als 25 % reduziert werden.

Die Optimierungsergebnisse der multivariaten Zeitreihenvorhersage zeigen ein vergleichbares Verhalten. Auch hier sind die Absolutwerte der Fehlermaße *RMSE* und *MAPE* für den verwendeten genetischen Algorithmus im Vergleich zur Zufallssuche geringer. Der Absolutwert des mittleren Fehlers *ME* ist für die Vorhersage des Netzes optimiert mit dem genetisches Algorithmus minimal größer. Die Zeitersparnis durch Verwendung des genetischen Algorithmus beläuft sich auf ca. 8 %.

Bereits bei der Hyperparameteroptimierung der MLP-Architektur wird deutlich: Beide Algorithmen finden die Umgebung des Minimums der Hyperparameter, jedoch scheint der genetische Algorithmus im Vergleich zur Zufallssuche näher am Minimum zu liegen bzw. stabil zu konvergieren.

5.2 TCN-Architektur

Die Hyperparameter θ werden zur Minimierung der Prognosegüte der Vorhersage im folgenden Suchraum

$$\theta \begin{cases} \text{Filteranzahl} \in [1,\ldots,256] \\ \text{Filtergröße} \in [1,\ldots,16] \\ \text{Anzahl der Stapel} \in [1,\ldots,5] \\ \text{Dilatation} \in [[1,2],[1,2,4],[1,2,4,8],[1,2,4,8,16]] \\ \text{Aktivierungsfunktionen} \in [relu, elu, tanh, selu, sigmoid] \\ \text{Optimierer} \in [rmsprop, adam, sgd, adadelta] \end{cases}$$

verändert. Für die beiden Hyperparameteroptimierungsalgorithmen Zufallssuche und genetischer Algorithmus, wird auch für den Fall der TCN-Architektur der Suchraum konstant gehalten.

Zum Vergleich der Prognosen der unterschiedlich optimierten Netze sind die Vorhersagen beispielhaft an einem trainierten Netz in Abbildung 12 dargestellt. Unterschiede und Abweichungen der Vorhersagen untereinander stellt man vor allem an den lokalen Maxima und Minima der Energiezeitreihe fest.

Die Qualität der Algorithmen der Hyperparameteroptimierung kann durch die verschiedenen Fehlermaße der Prognosegüte sowie die Berechnungszeit aufgetragen in Tabelle 2 abgeschätzt werden. Wie schon für die MLP-Architektur sind die Durchschnittswerte von drei unterschiedlichen Optimierungsdurchläufen angegeben.

Sowohl bei der univariaten als auch bei der multivariaten Zeitreihenprognose stellt man vergleichbare Abhängigkeiten fest. Die Fehlermaße *RMSE* und *MAPE* der Prognose liegen für die Netze optimiert nach der Zufallssuche wie auch nach dem genetischen Algorithmus in einer gleichen Größenordnung, wobei die Fehlerwerte jeweils für die Netze optimiert nach dem genetischen Algorithmus geringer sind. Zunächst scheint der *ME* ein konträres Bild zu vermitteln. Die Absolutwerte der Fehlermaße der Netze optimiert nach der

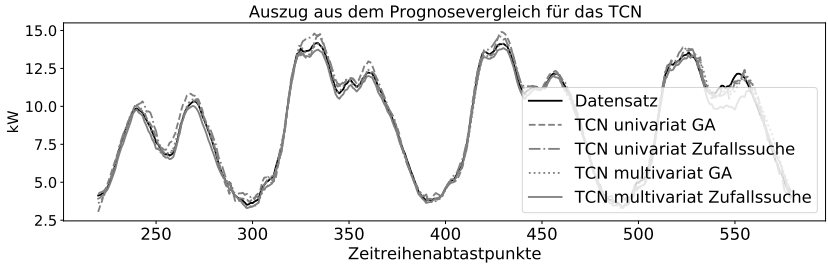

Bild 12: Vergleich der Prognosen der optimierten TCN-Netze anhand eines Ausschnitts der Energiezeitreihe von vier Tagen

Zufallssuche sind hier geringer. Dies lässt sich jedoch durch den Aufbau des Fehlermaß *ME* erklären. Anstatt die absoluten Abweichungen bzw. die quadrierten Werte zu mitteln, werden beim *ME* die Fehler vorzeichenbehaftet gemittelt. Dadurch können sich negative sowie positive Fehlerwerte ausgleichen. Folglich können rauschbehaftete Vorhersagen einen geringeren mittleren Fehler im Vergleich zu Vorhersagen mit einem Offset aufweisen. Gerade der Ausschnitt der Zeitreihe (siehe Abbildung 12) zeigt, dass die univariate Zeitreihenvorhersage optimiert nach dem genetischen Algorithmus die Zielzeitreihe überschätzt (TCN univariat GA). Durch den Offset entsteht ein vergleichsweise großer mittlerer Fehler. Gleiches gilt für die multivariate Zeitreihenvorhersage optimiert nach dem genetischen Algorithmus (TCN multivariat GA). Hingegen weisen die Prognosen mit der Optimierung nach der Zufallssuche keinen klaren Offset auf. Man kann eine Überschätzung und auch Unterschätzung der Zielzeitreihe feststellen, sodass der mittlere Fehler im Vergleich geringer ist.

Neben der Prognosegüte spielt auch die benötigte Rechenzeit der Hyperparameteroptimierung eine entscheidende Rolle. Durch die Verwendung des genetischen Algorithmus kann die Optimierungszeit bei der univariaten Zeitreihenprognose um ca. 25 % und bei der multivariaten Zeitreihenprognose um mehr als 55 % reduziert werden.

Auch anhand der TCN-Architektur konnte damit gezeigt werden: Durch die Verwendung eines genetischen Algorithmus zur Zeitreihenvorhersage kann die Prognosegüte in Bezug auf *RMSE* und *MAPE* bei gleichzeitiger signifikanter Reduktion der Optimierungszeit erhöht werden.

Tabelle 2: Mittelwerte der Ergebnisse der unterschiedlichen Hyperparameteroptimierungen für die TCN-Netze

	Univariat		Multivariat	
Methode	Zufallssuche	gen. Algorithmus	Zufallssuche	gen. Algorithmus
RMSE	0,464	0,443	0,497	0,493
MAPE	5,8	5,3	5,83	5,56
ME	−0,16	0,22	−0,056	0,195
Zeit (mm:ss)	2196 : 36	1647 : 11	2124 : 42	948 : 36

6 Zusammenfassung und Ausblick

Beim Entwickeln und Trainieren von Prognosemodellen basierend auf KNN ist ein entscheidender Faktor die Optimierung der Hyperparameter um leistungsfähige und präzise Modelle zu generieren. Bestehende automatisierte Verfahren zur Hyperparameteroptimierung verwenden zeitaufwändige Raster- oder Zufallssuchen. Speziell bei der Zufallssuche besteht das Risiko lediglich des Auffindens eines lokalen Extrempunktes. Demgegenüber wird bei der Rastersuche zwar der Suchraum größtmöglich abgedeckt, jedoch überschreitet die Optimierungszeit schnell ein akzeptables Maß bei einer steigenden Anzahl an Hyperparametern. Der in diesem Beitrag durchgeführte Vergleich der Methoden zur Hyperparameteroptimierung für KNN zur Energielastprognose hat gezeigt, dass durch die Verwendung von genetischen Algorithmen die Prognosegüte bei gleichzeitiger Reduzierung der Optimierungszeit gesteigert wird. Durch die Verwendung von Evolutionsprinzipien zur Optimierung der Hyperparameter erfolgt durch den Algorithmus eine Entwicklung in der Suchraumabdeckung über die Generationen hinweg, hin zu performanten Bereichen im Suchraum, ohne dabei die Vorteile der Zufallssuche zu verwerfen. In der dargestellten Anwendung der Energielastprognose trägt der genetische Algorithmus dazu bei, die bestmögliche Architektur bzw. Hyperparameterauswahl weitestgehend autonom zu ermitteln und somit die Modelle schneller, effektiver und effizienter zu trainieren. Damit kann dieses Verfahren einen wichtigen Beitrag

leisten hinsichtlich der Reproduzierbar-/ Vergleichbarkeit von Modellen und wissenschaftlichen Untersuchungen. Modellansätze basierend auf KNN können nur adäquat miteinander verglichen werden, wenn diese ein gleiches Maß an Feinabstimmung erhalten. Eine stabile, schnelle und zuverlässige Methode zur automatisierten Hyperparametereinstellung würde also nicht nur die Optimierung erleichtern und die Performance der Modelle steigern, sondern könnte auch durch eine einfachere Handhabbarkeit zur weiteren Verbreitung solcher Anwendungen führen.

Anknüpfungspunkte an die vorliegende Forschungsarbeit bestehen in der Ausweitung der Untersuchung auf weitere und größere Netzarchitekturen wie z.B. Ensemble Learning Verfahren und die Anwendung auf komplexere Aufgabenstellungen mit einer größeren Anzahl exogener Einflüsse. Eine Weiterentwicklung des genetischen Algorithmus könnte daraus bestehen, die trainierte und sortierte Anfangspopulation in Spezies zu unterteilen. Eine Spezie vereint dabei Individuen mit ähnlichen genetischen Eigenschaften und ähnlichen Verlustwerten. So könnte verhindert werden, dass performante Individuen die Population dominieren und gleichzeitig erhalten weniger performante eine längere Entwicklungszeit, da sie nur mit genetisch und leistungsfähig ähnlichen Individuen verglichen werden.

Literatur

[1] O. D. Doleski. „Herausforderung Utility 4.0.“. Wiesbaden: Springer Fachmedien Wiesbaden. 2017.

[2] I. Goodfellow, A. Courville und Y. Bengio. „Deep Learning: Das umfassende Handbuch: Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze“. Frechen: Verlags GmbH & Co. KG. 1. Auflage. 2018.

[3] F. Hutter, L. Kotthof und J. Vanschoren. „Automated machine learning: Methods, systems, challenges“. Springer International Publishing. 2019.

[4] T. Yu und H. Zhu. „Hyperparameter Optimization: A Review of Algorithms and Applications“. *arXiv-preprint* 2020. arXiv:2003.05689

[5] L. Yang und A. Shami. „On hyperparameter optimization of machine learning algorithms: Theory and practice“. In: *Neurocomputing*, Band 415, S. 295-316. 2020. https://doi.org/10.1016/j.neucom.2020.07.061

[6] I. Loshchilov und F. Hutter. „CMA-ES for Hyperparameter Optimization of Deep Neural Networks“. *arXiv-preprint* 2016. arXiv:1604.07269

[7] X. Cui, W. Zhang, Z. Tüske und M. Picheny. „Evolutionary Stochastic Gradient Descent for Optimization of Deep Neural Networks“. *arXiv-preprint* 2018. arXiv:1810.06773

[8] S. Raychaudhuri. „Introduction to Monte Carlo simulation“. In: *2018 Winter Simulation Conference*, S. 91-100. 2008. https://doi.org/10.1109/WSC.2008.4736059

[9] N. M. Aszemi and P. D. D. Dominic. „Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms“. In: *International Journal of Advanced Computer Science and Applications*, Band 10, Nr. 6, S. 269-278. 2019. http://dx.doi.org/10.14569/IJACSA.2019.0100638

[10] K. O. Stanley und R. Miikkulainen. „Evolving neural networks through augmenting topologies“. In: *Evolutionary Computation*, Band 10, Nr. 2, S. 99-127. 2002. https://doi.org/10.1162/106365602320169811

[11] S. Raschka. „Machine Learning mit Python: Das Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning“. Frechen: MITP. 1. Auflage. 2017.

[12] S. Bai, J. Z. Kolter und V. Koltun. „An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling“. *arXiv-preprint* 2018. arXiv:1803.01271v2

[13] G. D. Rey und K. F. Wender. „Neuronale Netze: Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung“. Bern: Huber. 2. Auflage. 2011.

[14] M. Abadi et al. „TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems“. https://www.tensorflow.org/. 2015.

[15] „Autonomio Talos“. http://github.com/autonomio/talos. 2020.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever und R. Salakhutdinov. „Dropout: a simple way to prevent neural networks from overfitting“. In: *The Journal of Machine Learning Research*, Band 15, Nr. 1, S. 1929-1958. 2014. https://doi.org/10.5555/2627435.2670313

[17] P. Taylor und R. Lewontin. „The Genotype/Phenotype Distinction“. In: *The Stanford Encyclopedia of Philosophy*, E. N. Zalta (ed.). 2021. https://plato.stanford.edu/archives/sum2021/entries/genotype-phenotype/