Carl von Ossietzky Universität Oldenburg
Abteilung für Digitalisierte Energiesysteme

Data Augmentation von Trainingsdatensätzen für Machine-Learning-Verfahren in Energiesystemen

Bachelorarbeit

vorgelegt von

Maximilian Böhm

Geboren am 09.12.1998 in Bocholt

Erstprüferin:	Prof. Dr.-Ing. Astrid Nieße
Zweitprüfer:	M. Sc. Thomas Wolgast
Betreuer:	M.Sc. Thomas Wolgast

Oldenburg, den 18. Oktober 2021

Bedingt durch die Entwicklung von Energienetzen kommen in diesen immer häufiger datengetriebene Verfahren wie das Machine-Learning zur Anwendung. Diese Verfahren benötigen umfangreiche Trainingsdaten, die im Energiebereich häufig nicht ausreichend vorhanden sind. Eine Möglichkeit, die Trainingsdaten durch synthetische Daten zu vergrößern, stellt die Data Augmentation dar. Ziel dieser Arbeit ist es zu untersuchen, ob die Ergebnisse eines Machine-Learning-Verfahrens aus dem Energiebereich durch Data Augmentation verbessert werden können. Zusätzlich ist zu fragen, ob verschiedene Ansätze zur Data Augmentation Unterschiede in der Auswirkung auf das Machine-Learning-Verfahren aufweisen. Dafür gilt es, verschiedene Ansätze zur Umsetzung der Data Augmentation vorzustellen und auszuwählen. Umgesetzt werden zwei weniger komplexe Ansätze: die transformationsbasierten Ansätze und ein komplexerer Machine-Learning-Ansatz, der Variational Autoencoder.

Abstract Die Datengrundlage liefert der Simbench-Datensatz, erweitert durch die Data-Augmentation-Ansätze. Für die Evaluation wird eine lineare Regression genutzt, welche die Ergebnisse einer Leistungsflussberechnung annähert. Mit den drei Ansätzen zur Data Augmentation werden verschiedene Experimente durchgeführt, aus denen Trainingsdaten für die lineare Regression hervorgehen. Die Bewertung der Ergebnisse der linearen Regression erfolgt durch die zwei Metriken Mean Absolute Error und R^2. Wie die Experimente zeigen, eignet sich eine Data Augmentation durch prozentuale Transformationen der Werte in den Trainingsdaten am besten für die lineare Regression hinsichtlich des Mean Absolute Error. Eine Data Augmentation durch eine Vertauschung von Sequenzen der Trainingsdaten oder durch den Variational Autoencoder empfiehlt sich in der vorliegenden Betrachtung nicht für die lineare Regression. Schließlich konnte auch keiner der Data-Augmentation-Ansätze zu einer Verbesserung der R^2-Metrik führen.

Inhaltsverzeichnis

Abkürzungen

DA	Data Augmentation
ML	Machine-Learning
GAN	Generative Adversarial Network
VAE	Variational Autoencoder
KNN	Künstliche neuronale Netze
MAE	Mittlere absolute Abweichung

Formelzeichen

Griechische Symbole

w Gewichte eines Neurons

φ Aktivierungsfunktion eines Neurons

b Bias eines Neurons

β Regressionskoeffizient

ϕ Gewichte eines neuronalen Netzes

Lateinische Symbole

G Generatornetz (Generative Adversarial Network)

D Discriminatornetz (Generative Adversarial Network)

KL Kullback-Leibler-Divergenz

1 Einleitung

Energiesysteme haben sich in den letzten Jahren stark verändert und mit ihnen auch die an sie gestellten Anforderungen und Herausforderungen. Bislang werden Energienetze durch zentrale Stromerzeugung von wenigen großen Erzeugungsanlagen dominiert. Aktuelle Trends weisen allerdings in die Richtung vieler kleinerer dezentraler Erzeugungsanlagen. Dabei soll viel Energie aus erneuerbaren Quellen gewonnen werden, wie zum Beispiel aus Windkraftanlagen, Photovoltaikanlagen oder Biogasanlagen. Laut Umweltbundesamt stieg der Anteil erneuerbarer Energien in Stromnetzen seit dem Jahr 2005 von etwa 10 Prozent auf über 45 Prozent (siehe Abbildung 1.1).

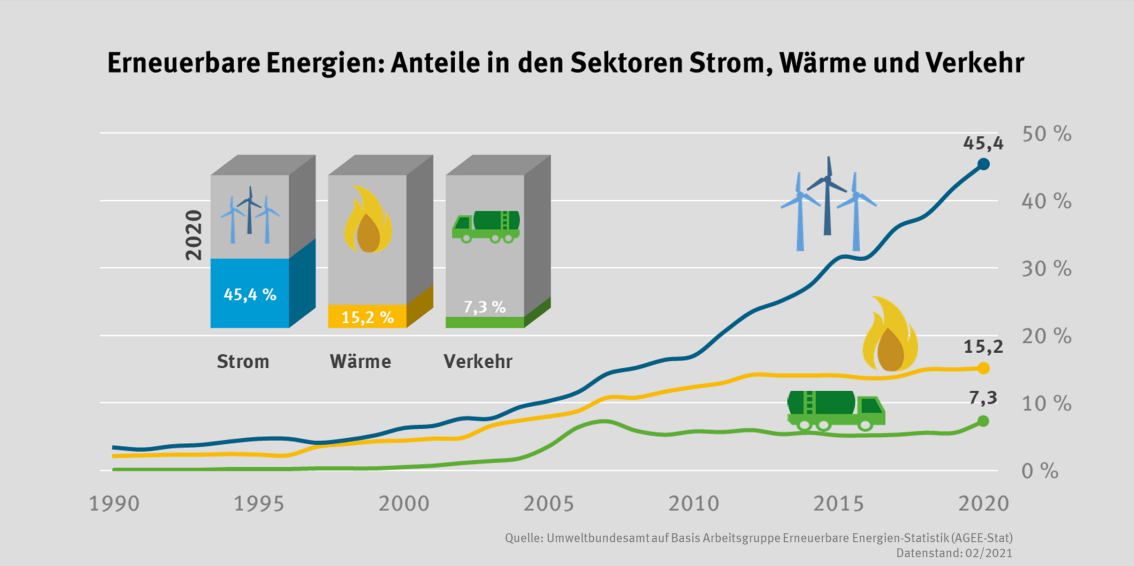


Abbildung 1.1: Entwicklung der Anteile erneuerbarer Energien (Februar 2021) [1]

Auf diese Weise können wichtige ökonomische und ökologische Vorteile erlangt sowie die Performanz und Effizienz gesteigert werden [2]. Des Weiteren führen diese Trends zu einer deutlich komplexeren Struktur der Energienetze, wodurch sich neue Herausforderungen in Bereichen wie der Verteilung im Energienetz, der Netzstabilität oder der Versorgungssicherheit ergeben. Diese komplexen und dezentralen Netze sind als intelligente Stromnetze[1]

[1] Intelligente Netze werden auch als Smart Grids bezeichnet [3].

angelegt. Diese Netze erstreben eine Optimierung und Überwachung der Netzbestandteile, durch eine Vernetzung dieser. Die Optimierung ist aufgrund der stärkeren Belastung durch erneuerbare Energiequellen wie Wind- oder Solarquellen auch nötig, da diese Quellen eine hohe Fluktuation aufweisen und so die Versorgungssicherheit gefährden können. Auch werden vermehrt Datensammler wie Smart Sensors in Smart Grids angewendet [4, 5], was den Trend zu datengetriebenen Verfahren in Energienetzen unterstützt.

Für die moderne Art der Optimierung von Netzen gibt es verschiedene Ansätze, die in ihrer Mehrheit das Datenangebot der gesammelten Daten nutzen [6]. Immer häufiger kommen Machine-Learning (ML)-Methoden[2] für diese Optimierung zum Einsatz, welche vielversprechende Ergebnisse erzielen [8]. ML-Methoden werden im Smart Grid unter anderem für Prognosen des Energieverbrauchs, Nachfragevorhersagen, Kostenvorhersagen, Lastprognosen, verschiedene Optimierungsaufgaben, Risikovorhersagen, Lagerplanungen und viele weitere Aufgaben eingesetzt [8]. ML-Methoden benötigen jedoch, eine große Anzahl historischer Daten, um gute Ergebnisse zu liefern. Die Anzahl historischer Energiedaten ist allerdings beschränkt. Grund dafür sind hoher Aufwand und hohe Kosten beim Sammeln dieser Daten sowie Datenschutz- und Sicherheitsgründe [9]. Ein Beispiel für einen solchen Datensatz ist der Simbench-Datensatz [10]. Dieser Datensatze bildet Zeitreihendaten eines Energienetzes für ein Jahr ab. Die Beschränkung auf nur ein Jahr führt zu Problemen, wenn ML-Methoden diesen Datensatz nutzen [11]. Auf den Simbench-Datensatz und das Problem der zeitlichen Beschränkung wird später noch genauer einzugehen sein. Um die ML-Methoden in Energiesystemen zu verbessern, ist es also nötig, eine größere Datengrundlage zu schaffen. Data Augmentation (DA) sind Methoden, die Datensätze durch kopierte und veränderte Daten oder durch synthetische Daten vergrößern können [12]. Durch DA werden neue Daten geschaffen, die sich an den bereits existierenden historischen Daten orientieren und so möglichst realistisch sind. In Abbildung 1.2 und Abbildung 1.3 werden Beispiele für Eingaben und Ausgaben von DA Methoden für Bilddaten dargestellt.

ML-Methoden können durch DA verbessert werden, indem die Datengrundlage für diese Methoden vergrößert wird. Auch spezielle ML-Probleme wie Overfitting[3] lassen sich durch DA verhindern oder abschwächen.

Es gibt verschiedene Ansätze zur Umsetzung von DA. In dieser Arbeit werden dabei die Gruppen transformationsbasierte Ansätze, statistische Ansätze und ML-Ansätze unterschieden. Gong et al. [16] zeigen, dass ML-Ansätze zu besseren Ergebnissen führen als einfache zufallsbasierte Verfahren. Generell lässt sich sagen, dass es im Bereich der DA

[2] Machine-Learning ist ein Teilbereich der künstlichen Intelligenz, wodurch sich auf Basis vorhandener Daten und Algorithmen Muster und Gesetzmäßigkeiten erkennen und Lösungen entwickeln lassen [7].

[3] Von Overfitting spricht man, wenn ein statistisches oder Machine-Learning-Modell auf Trainingsdaten bessere Ergebnisse erzielt als auf unbekannten Testdaten, da es keine ausreichende Generalisierung aus den Trainingsdaten erlernt hat [15].

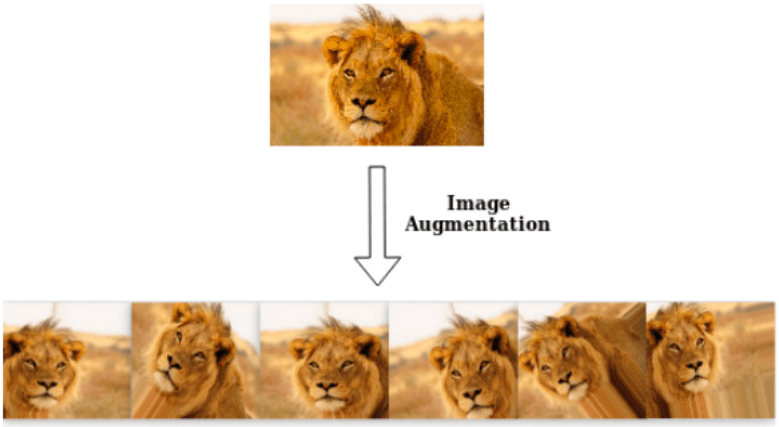


Abbildung 1.2: Data Augmentation bei Bildern: durch Verschieben, Spiegeln, Drehen, Strecken und Stauchen [13]

bereits viele erfolgreiche Durchbrüche gab. Jedoch haben diese Durchbrüche zu großen Teilen im Bereich der Image Augmentation stattgefunden. Image Augmentation ist ein Teilbereich der DA, der sich mit der Erweiterung von Bilddatensätzen befasst. Mit dem Bereich der Smart Grids hat sich die Forschung zu DA bislang eher weniger beschäftigt [17].

1.1 Zielsetzung

In dieser Arbeit soll untersucht werden, ob durch DA der Trainingsprozess eines ML-Verfahrens aus dem Energiebereich so verändert werden kann, dass bessere Endergebnisse entstehen. Außerdem soll der Einfluss verschiedener Ansätze der DA auf die Ergebnisse von ML-Verfahren verglichen werden. Hierzu wird eine lineare Regression implementiert, deren Ergebnisse zur Evaluation dienen. Dann werden zwei transformationsbasierte Ansätze und ein ML-Ansatz umgesetzt. Ziel ist es zu überprüfen, ob eine Verbesserung im Ergebnis der ML-Methode stattfindet und welchen Einfluss die jeweiligen Ansätze auf das Ergebnis haben. Es sollen folgende Forschungsfragen beantwortet werden:

RQ1: Ist es möglich, durch Data Augmentation das Training einer ML-Methode aus dem Energiebereich zu verbessern?

RQ2: Welche Data Augmentation Ansätze sind besonders geeignet, um Trainingsdaten einer Machine-Learning-Methode aus dem Energiebereich zu erweitern?

Abbildung 1.3: Eingaben und Ergebnisse der Data Augmentation bei Bildern, durch ein Nvidia Generative Adversarial Network[14]

1.2 Überblick

Zunächst werden die Grundlagen erläutert, die für das Verständnis dieser Arbeit relevant sind. Kapitel drei stellt verwandte Arbeiten zur DA im Allgemeinen und zur DA im Energiebereich vor. Im Anschluss wird die Auswahl der Methoden und deren Umsetzung dargestellt, wobei auf die verwendeten Technologien, die Datengrundlage und einige grundlegende Aspekte der Implementierung eingegangen wird. Im darauf folgenden Kapitel erfolgt eine Evaluation der Ergebnisse. Dazu werden zuerst die genutzten Metriken beschrieben. Dann werden die Ergebnisse dargestellt, miteinander verglichen und die daraus gewonnen Erkenntnisse diskutiert. Den Schluss bilden ein Fazit und ein Ausblick auf weitere Forschungmöglichkeiten.

2 Grundlagen

Das folgende Kapitel dient dazu, wichtige, zum wesentlichen Verständnis dieser Arbeit benötigte Grundlagen zu erläutern.

2.1 Machine-Learning

Maschinelles Lernen (ML) ist der Oberbegriff für einen Teilbereich der künstlichen Intelligenz, der sich mit der Entwicklung von IT-Systemen befasst, die durch ein sogenanntes *Training künstliches* Wissen aus Daten *erlernen* [18]. ML-Algorithmen lassen sich, abhängig davon, ob sie mit oder ohne menschlicher Unterstützung trainiert wurden, in Supervised Learning (unterstütztes Lernen), Unsupervised Learning (nicht unterstütztes Lernen), Semisupervised Learning (teilweise unterstütztes Lernen) und Reinforcement Learning (bestärkendes Lernen) unterteilen [18, 19]. Für diese Arbeit sind nur das Supervised Learning und das Unsupervised Learning relevant. Die Lineare Regression, welche später in diesem Kapitel vorgestellt wird, gehört zum Supervised Learning, da die Trainingsdaten die Zieldaten, also die gesuchten Lösungen enthalten und diese auch für das Training genutzt werden. Die Generativen Modelle, die in diesem Kapitel noch genauer vorgestellt werden, sind dem Unsupervised Learning zuzuordnen, da die Trainingsdaten keine Zieldaten enthalten. Generative Modelle nutzen Künstliche neuronale Netzes (KNNs).

2.1.1 Künstliche neuronale Netze

KNNs sind Algorithmen, die von der Funktionsweise des Gehirns inspiriert sind [19] und zum Bereich des ML gehören. In einem KNN sind Neuronen mit anderen Neuronen verbunden und bilden so ein Netz. Die Neuronen sind die hauptsächliche Verarbeitungseinheit von KNNs und nutzen folgende mathematische Operation, um aus den Eingaben x_i die Ausgabe y zu erzeugen:

$$y = \varphi(\sum_i (w_i x_i) + b) \tag{2.1}$$

[19]. Dabei steht w_i für die Gewichte, b für den Bias und φ für die Aktivierungsfunktion. Abbildung 2.1 zeigt das gesamte mathematische Modell eines künstlichen Neurons. Man sieht, wie ein künstliches Neuron einen Input x in einen Output o umwandelt.

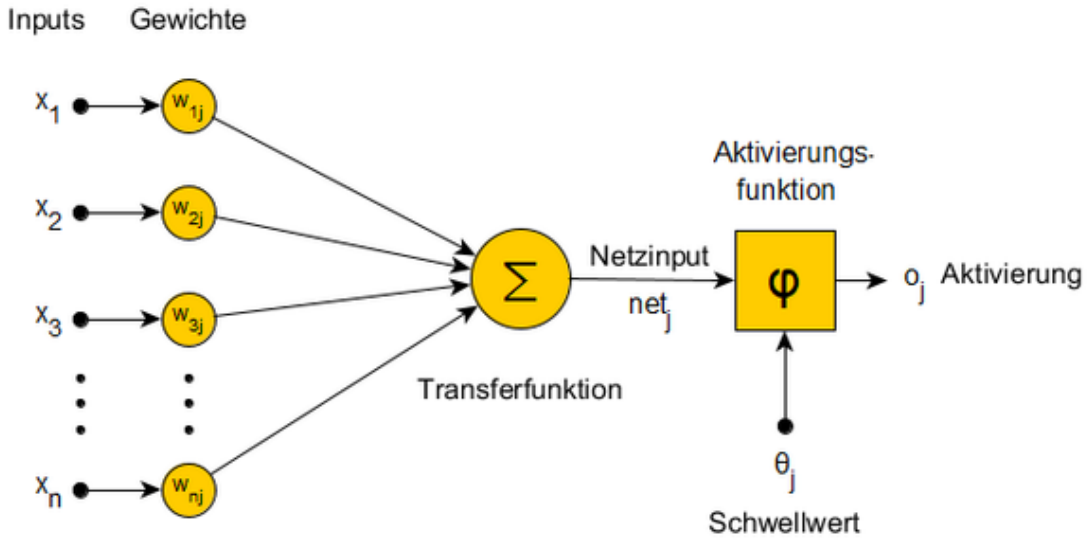


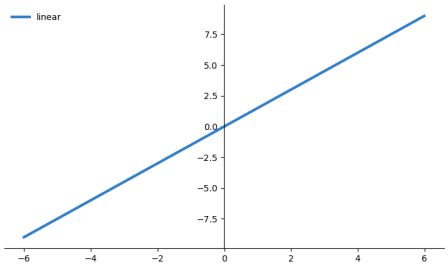
Abbildung 2.1: Das mathematische Modell eines künstlichen Neurons[20]

Der Input ist ein Vektor aus Aktivierungsstärken. Dieser Vektor wird gewichtet, indem er mit den Gewichten multipliziert wird. Dann werden die gewichteten Werte aufsummiert, sodass ein einzelner Wert als Netzinput in die Aktivierungsfunktion übergeben wird. Die Aktivierungsfunktion ordnet jedem Netzinput eine Outputstärke zu. Diese Aktivierungsfunktion kann beispielsweise ein lineare Funktion (siehe Gleichung 2.2, Abbildung 2.2(a)), eine Sigmoidfunktion (siehe Gleichung 2.3, Abbildung 2.2(b)) oder auch eine Rectified Linear Unit Funktion (ReLU) (siehe Gleichung 2.4, Abbildung 2.2(c)) sein [19].

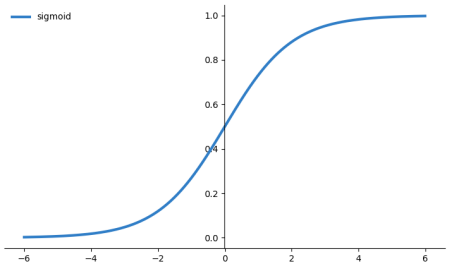
$$f_{linear}(z) = z \tag{2.2}$$

$$f_{sigmoid}(z) = \sigma(z) = \frac{1}{(1 + e^{-z})} \tag{2.3}$$

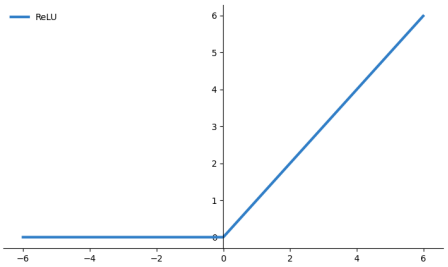
$$f_{relu}(z) = max(0, z) \tag{2.4}$$



(a) Linear



(b) Sigmoid



(c) ReLUl

Abbildung 2.2: Graphische Darstellung der Aktivierungsfunktionen

KNNs weisen die Strukturen gerichteter Graphen auf [21]. Die Neuronen bilden die Knoten in diesem gerichteten Graphen. Sie leiten Informationen durch das Netz und geben sie am Ende aus.

Es gibt verschiedene Architekturen für KNNs; für das Thema dieser Arbeit ist jedoch nur die Architektur der Multi Layer Perceptrons relevant. Diese sind eine Art von KNNs, die aus verschiedenen Schichten bestehen, einer Eingabeschicht (Input Layer) für die Eingabedaten, einer Ausgabeschicht (Output Layer) für die Ausgabedaten und einer oder mehreren verborgenen Schichten (Hidden Layer) [21]. Abbildung 2.3 visualisiert ein KNN mit zwei verborgenen Schichten. Dort sind die Neuronen jeweils mit den Nachfolgerneuronen in der nächsten Schicht verbunden.

Die Eingabeschicht eines KNN schleust die Daten in das Netz ein und gibt sie an die nächste Schicht weiter. Die verborgene Schicht gewichtet die empfangenen Daten in jeder Schicht neu und leitet sie ihrerseits ebenfalls an die nächste Schicht weiter. Dies vollzieht sich in jeder einzelnen der verborgenen Schichten. Die Ausgabeschicht ist die letze Schicht des KNN und liefert die Ausgabe des gesamten KNN.

Der Vorgang des Lernens funktioniert wie folgt: Vor dem ersten Netzdurchlauf werden die

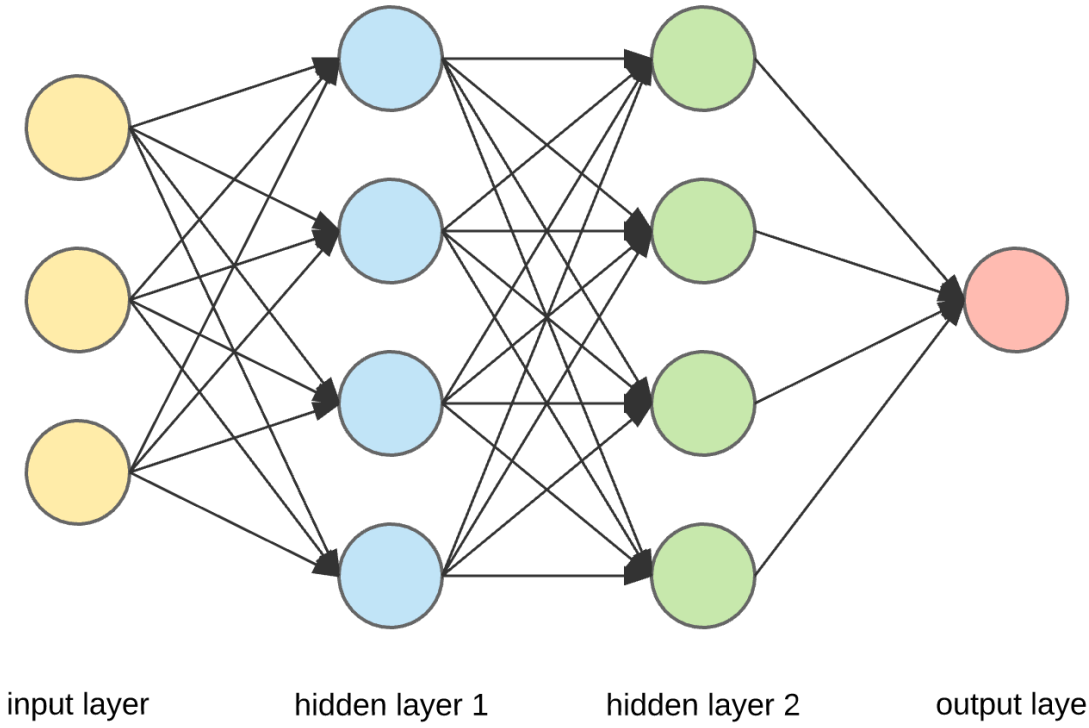


Abbildung 2.3: Künstliches neuronales Netz mit zwei Hidden Layern[22]

Gewichte des KNN initialisiert. Für diese Initialisierung gibt es verschiedene Möglichkeiten, so können beispielsweise Zufallswerte Anwendung finden. Die Daten durchlaufen das Netz und werden dann wieder ausgegeben. Die Ausgaben werden bewertet. Falls die Zieldaten verfügbar sind, kann der Fehler zwischen den Ausgaben und den Zieldaten berechnet werden. Mit Hilfe des Gradientenverfahrens[1] und der Backpropagation[2] erfahren die Gewichte des Netzes nach jedem Durchlauf eine solche Veränderung, sodass der Fehler der Ausgabe minimiert wird [19].

2.1.2 Lineare Regression

Die lineare Regression ist einer der wichtigsten und auch einfachsten Algorithmen für das ML [18]. Sie ist ein statistisches Verfahren, das das Verhältnis von zwei oder mehreren Variablen erklärt [24, 19]. Für ein lineares Verhältnis kann folgendes Modell verwendet

[1] Das Gradientenverfahren wird genutzt, um anhand von Steigungen Optimierungsprobleme zu lösen [23].
[2] Die Backporpagation ist ein Algorithmus, der in Verbindung mit dem Gradientenverfahren die Gewichte eines KNN verbessern kann.

werden:

$$y = \beta_0 + \beta_1 x + \epsilon. \tag{2.5}$$

Dabei ist y die abhängige Variable und x die unabhängige Variable. Die Variable ϵ steht für den Term des Fehlers im Modell. In diesem Kontext steht *Fehler* für einen statistischen Term, der zufällige Fluktuationen, Messfehler oder den Einfluss von Faktoren, über die keine Kontrolle besteht, beschreibt [24]. Mit den beobachteten Werten für x und y werden β_0 und β_1 geschätzt. Die abhängige Variable y wird oft von mehr als einer unabhängigen Variable beeinflusst [24]. Ein lineares Modell für diesen Fall ist das Folgende:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \epsilon. \tag{2.6}$$

Die Parameter $\beta_0, \beta_1, ..., \beta_k$ heißen Regressionskoeffizienten. Wie in Gleichung 2.5 steht ϵ für die zufällige Variation in y, die nicht durch die Variable x erklärt werden können. Ein Modell bietet ein theoretisches Framework, welches den Mechanismus repräsentiert, der die Beobachtungen erzeugt hat. Das allgemeine lineare Modell [25] betrachtet die Situation, bei der die abhängige Variable y kein Skalar, sondern ein Vektor ist. Das allgemeine Lineare Modell ist gegeben durch

$$y = XB + \epsilon. \tag{2.7}$$

Dabei ersetzt die Matrix B die β-Werte des klassischen linearen Modells. Bei der linearen Regression soll ein lineares Modell mit den Koeffizienten $\beta = (\beta_1, ..., \beta_p)$ so angepasst werden, dass die Quadratsumme der Residuen zwischen den beobachteten Werten für y und den von der linearen Regression approximierten y-Werten minimiert wird [26]. Das bedeutet, dass, wenn man eine Regressionslinie durch die Daten zieht, wie sie in Abbildung 2.4 graphisch abgebildet ist, die Distanz von jedem Datenpunkt zur Regressionsline ermittelt und quadriert wird und dann alle quadrierten Fehler-Distanzen aufsummiert werden. Die lineare Regression soll diese Endsummen der Fehler, zum Beispiel durch das Gradientenverfahren, welches den Fehler auf Basis der Steigung minimiert, verringern. Diese Endsumme der Fehler versucht die lineare Regression, zum Beispiel durch das Gradientenverfahren, welches den Fehler auf Basis der Steigung minimiert [23], zu reduzieren.

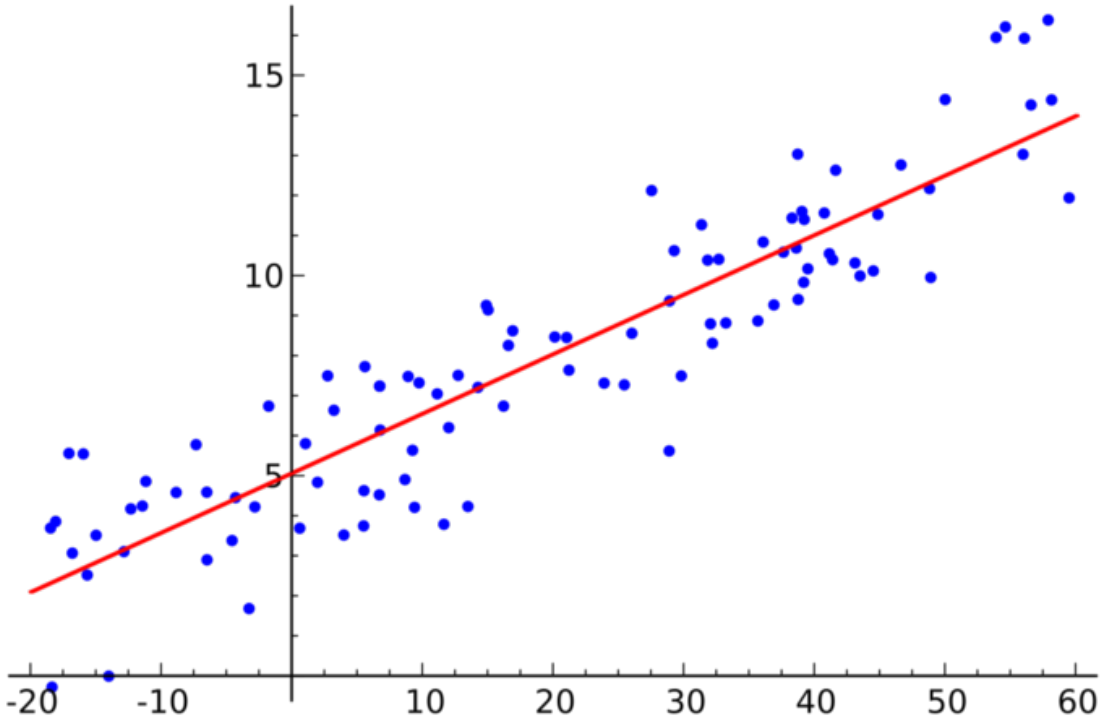


Abbildung 2.4: Beispiel für die lineare Regression [27]

2.1.3 Testen von Machine-Learning-Modellen

Da ML-Modelle in der Anwendung, nachdem sie trainiert wurden, unbekannte Daten verarbeiten müssen, sollte überprüft werden, ob das ML-Modell das erlernte Wissen entsprechend generalisieren kann. Dazu ist die Datengrundlage vor dem Training eines ML-Modells in einen Trainings- und einen Testdatensatz zu unterteilen [18]. Der Trainingsdatensatz wird für das Training, der Testdatensatz zur Überprüfung genutzt, so kann die Anwendbarkeit des ML-Modells schon im Entwicklungsprozess kontrolliert werden.

2.2 Generative Modelle

Generative Modelle erlernen mittels KNNs die Wahrscheinlichkeitsverteilung $P(Y|X)$ von Trainingsdaten. Diese multivariate Verteilung kann durch den Satz von Bayes zu $P(X|Y)$ umgeformt und so für diskriminative Aufgaben genutzt werden. Zusätzlich ist es möglich, neue (x, y) Paare zu erzeugen, was zu dem Ergebnis neuer Daten führt, die nicht Teil der ursprünglichen Trainingsdaten sind [28].

2.2.1 Generative Adversarial Networks

Generative Adversarial Networks [29] sind ein Ansatz für das generative Modellieren durch die Nutzung von Deep Learning[3]-Methoden wie die Convolutional Neural Networks (CNN). CNNs sind neuronale Netze mit mindestens einer Faltungsschicht[4].

Generative Adversarial Networks (GANs) wurden von Goodfellow et al. [29] eingeführt. Ein GAN besteht aus zwei KNNs, einem Discriminatornetz D und einem Generatornetz G. Das G hat das Ziel, Daten x so zu erzeugen, dass sie nicht von den ursprünglichen Trainingsdaten y unterschieden werden können. Dazu wird eine Input Noise Variable $p_z(z)$ genutzt, die eine Abbildung zum Datenraum $G(z;\phi_g)$ herstellt [29], wobei ϕ_g die Gewichte des KNN von G sind. D unterscheidet, ob Daten von G erzeugt wurden oder aus den Trainingsdaten y stammen [29]. In Abbildung 2.5 ist die Architektur eines GAN abgebildet. Der Input für das G ist dort ein Random Input Vector, der mit der Input Noise Variable $p_z(z)$ gebildet wird. Die durch G generierten Daten werden dann zusammen mit den realen Daten y in D übergeben. D ist durch $D(x;\phi_d)$ definiert [29]. Dabei bezeichnet ϕ_d die Gewichte des KNN von D und $D(x)$ die Wahrscheinlichkeit, dass x aus den Trainingsdaten x stammt und nicht von p_g. Die Wahrscheinlichkeitsverteilung für die realen Trainingsdaten ist p_r. Während des Trainings des GAN werden ϕ_d so angepasst, dass die Wahrscheinlichkeit für das richtige Klassifizieren von x maximiert wird. ϕ_g wird hingegen so trainiert, dass die Wahrscheinlichkeit von D zu erkennen, ob x von G erzeugt wurde, minimiert wird [29]. Die Loss-Funktion des GAN findet sich in Gleichung 2.8. Diese Loss-Funktion beschreibt ein Minmax-Spiel zwischen G und D. Wenn $p_g = p_r$ erreicht ist, hat das Minmax-Spiel das globale Optimum erreicht, also die Datenverteilung, welche von G erzeugt wird, die gleich der der realen Trainingsdaten ist [29].

$$min_G max_D V(D,G) = E_{x\ p_{data}(x)}[logD(x)] + E_{z\ p_z(z)}[log(1 - D(G(z))] \tag{2.8}$$

2.2.2 Varational Autoencoder

Der Variational Autoencoder (VAE) [32] ist eine Erweiterung des Autoencoder. Ein Autoencoder ist eine ML-Methode, die lernt, wie Eingabedaten in niedrigere Dimensionen codiert und dann die Daten erneut decodiert und rekonstruiert werden, um so nah wie möglich an die Eingabedaten heranzureichen. Er besteht aus drei Teilen: dem Encoder, welcher die Eingabedaten in eine Darstellung mit niedrigeren Dimensionen codiert, einer

[3] Deep Learning bezeichnet eine Methode des ML, die KNNs mit zahlreichen verborgenen Schichten nutzt.
[4] Die Faltungsschicht wird auch Convolutional Layer genannt, daher stammt der Name Convolutional Neural Network [30].

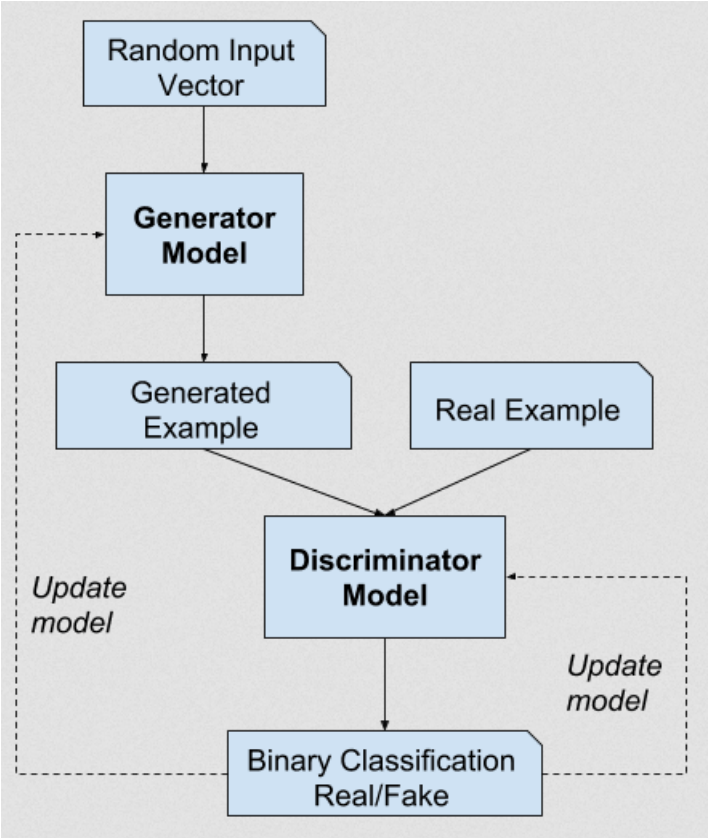


Abbildung 2.5: Beispiel der Architektur eines Generative Adversarial Network[31]

komprimierten Ebene, welche die codierte Darstellung und die niedrigste Dimension enthält, und dem Decoder, welcher die codierte Darstellung der Daten so rekonstruiert, dass sie sich möglichst nah an den realen Eingabedaten befinden. Der Encoder und der Decoder bestehen beide jeweils aus einem KNN [32].

Die Trainingsdaten können durch ein probalistisches Modell beschrieben werden. Für die Daten ergibt sich ein generativer Prozess mit den folgenden zwei Schritten: Zuerst wird eine latente Repräsentation z aus der Prior Verteilung $p(z)$ gezogen. Dann werden die Daten x aus der bedingten Wahrscheinlichkeitsverteilung $p(x|z)$ gezogen. Der Decoder kann durch $p(x|z)$ definiert werden, was die Verteilung der decodierten Daten bei gegebenen codierten Daten beschreibt. Der Encoder kann durch $p(z|x)$ definiert werden, was die Verteilung der codierten Daten bei gegebenen decodierten Daten beschreibt. Es wird angenommen, dass die codierte Representation z im latenten Raum der Prior Verteilung $p(z)$ folgt. Es gilt ebenfalls der Satz von Bayes (siehe Gleichung 2.9), der das Verhältnis zwischen dem Prior

$p(z)$ der Wahrscheinlichkeit $p(x|z)$ und dem Posterior $p(z|x)$ beschreibt [32]:

$$p(z|x) = \frac{p(x|z)p(z)}{px} = \frac{p(x|z)p(z)}{\int p(x|u)p(u)du} \tag{2.9}$$

Es wird angenommen, dass $p(z)$ eine Standard-Gauß-Verteilung ist und $p(x|z)$ eine Gaußverteilung, deren Mittelwert durch eine deterministische Funktion f der Variable z definiert ist und deren Kovarianzmatrix die Form einer positiven Konstanten c hat, welche die Identitätsmatrix I multipliziert [32]. Die Funktion F wird der Familie von Funktionen F zugeordnet. Auf Z ist später noch genauer einzugehen. Also gilt:

$$p(z) \equiv N(0, I) \tag{2.10}$$

$$p(x|z) \equiv N(f(z), cI) \qquad f \in F \qquad c > 0. \tag{2.11}$$

Aus $p(z)$ und $p(z|x)$ kann mit dem Satz von Bayes $p(z|x)$ berechnet werden. Im VAE wird $p(z|x)$ durch eine Gaußverteilung $q_x(z)$, deren Mittelwert und Kovarianz durch die Funktionen g und h der Parameter x definiert werden, beschrieben [32]. Diese Funktionen gehören zu der Familie der Gruppen G und H, auf welche ebenfalls später eingegangen wird. Also gilt:

$$q_x(z) \equiv N(g(x), h(x)) \qquad g \in G \qquad h \in H. \tag{2.12}$$

Nachdem somit eine Familie von Kandidaten für variationale Inferenz definiert wurde, muss die beste Aproximation innerhalb dieser Familie gefunden werden. Dafür werden die Funktionen g und h optimiert, um die Kullback-Leibler Divergenz[5] [33] zwischen der Approximation und dem Ziel $p(z|x)$ zu minimieren, damit die Wahrscheinlichkeitsverteilung der Approximation so nah wie möglich an dem Ziel $p(z|x)$ ist. Es wird also nach dem optimalen g^* und h^* gesucht, sodass gilt:

[5] Die Kullback-Leibler-Divergenz ist ein Maß für die Unterschiedlichkeit zweier Wahrscheinlichkeitsverteilungen.

$$(g*, h*) = \underset{(g,h)\in G\times H}{arg\,min}\ KL(q_x(z), p(z|x)) \tag{2.13}$$

$$= \underset{(g,h)\in G\times H}{arg\,min}\ (\mathbb{E}_{z\sim q_x}(logq_x(z)) - \mathbb{E}_{z\ q_x}(log\frac{p(x|z)p(z)}{p(x)})) \tag{2.14}$$

$$= \underset{(g,h)\in G\times H}{arg\,min}\ (\mathbb{E}_{z\sim q_x}(logq_x(z)) - \mathbb{E}_{z\ q_z} - \mathbb{E}_{z\ q_z}(logp(x|z)) + \mathbb{E}_{z\ q_z}(logp(x))) \tag{2.15}$$

$$= \underset{(g,h)\in G\times H}{arg\,max}\ (\mathbb{E}_{z\sim q_x}(logp(x|z)) - KL(q_x(z), p(z))) \tag{2.16}$$

$$= \underset{(g,h)\in G\times H}{arg\,max}\ (\mathbb{E}_{z\sim q_x}(-\frac{||x - f(z)||^2}{2c}) - KL(q_x(z), p(z))) \tag{2.17}$$

[34]. In der letzten Gleichung kann man den Tradeoff erkennen, der existiert, wenn der Posterior $p(z|x)$ aproximiert werden soll. Der Tradeoff besteht zwischen dem Maximieren der Likelihood der Beobachtungen (Maximierung der erwarteten Log-Likelihood des ersten Terms) und der Minimierung des Unterschieds zur Prior-Verteilung (Minimierung der KL Divergenz zwischen $q_x(z)$ und $p(z)$ für den zweiten Term). Dieser Tradeoff ist natürlich für Bayes'sche Inferenzprobleme und drückt die Balance zwischen dem Vertrauen in die Daten und dem Vertrauen in den Prior aus [34]. Es wurde gezeigt, dass anhand der bisherigen Annahmen der Posterior $p(z|x)$ mit der variationalen Inferenztechnik aproximiert werden kann [32]. Die Funktion f, die den Decoder definiert, ist in der Realität nicht bekannt und muss bestimmt werden. Die Performanz des Encoder-Decoder-Schemas hängt von der Wahl der Funktion f ab. Da $p(z|x)$ durch variationale Inferenz von $p(z)$ und $p(x|z)$ aproximiert werden kann und $p(z)$ eine einfache Gauß-Verteilung ist, sind die einzigen beiden Parameter, die im Modell optimiert werden müssen, c, welches die Varianz der Wahrscheinlichkeit definiert, und die Funktion f, welche den Mittelwert der Wahrscheinlichkeit definiert [34].

Es wird angenommen, dass man für jede f in F die beste Approximation von $p(z|x)$ als $q*_x(z)$ bezeichnet. Wir suchen nach einem Encoder-Decoder-Schema, das so effizient wie möglich ist. Dann soll die Funktion f gewählt werden, welche die erwartete Log-Likelihood von x mit gegebenen z, wenn z aus $q*_x(z)$ zufällig gezogen wird, maximiert [34]. Also suchen wir nach den optimalen f^* sodass gilt:

$$f* = \underset{f \in F}{arg\,max} \mathbb{E}_{z \sim q_x^*}(logp(x|z)) \tag{2.18}$$

$$= \underset{f \in F}{arg\,max} \mathbb{E}_{z \sim q_x^*}(-\frac{||x - f(z)||^2}{2c}), \tag{2.19}$$

wobei $q_x^*(z)$ von der Funktion f abhängt. Fasst man alles zusammen, so suchen wir nach dem optimalen $f*$, $g*$ und $h*$ sodass Gleichung 2.20 gilt [34]:

$$(f^*, g^*, h^*) = \underset{(f,g,h) \in F \times G \times H}{arg\,max} (\mathbb{E}_{z \sim q_x^*}(-\frac{||x - f(z)||^2}{2c}) - KL(q_x(z), p(z))). \tag{2.20}$$

In dieser Zielfunktion erkennt man den Rekonstruktionsfehler zwischen x und $f(z)$ und den Regularisierungsterm durch die KL-Divergenz zwischen $q_{x(z)}$ und $p(z)$. Außerdem erkennt man die Konstante c, welche das Verhältnis der beiden vorherigen Terme regelt.

Bislang hängt das probalistische Modell von den drei Funktionen f, g und h ab und drückt mit Hilfe der variationalen Inferenz das Optimierungsproblem aus, um $f*$, $g*$ und $h*$ zu ermitteln, welche das optimale Encoder-Decoder Schema ergeben. Da die Optimierung für die die Funktionen schwierig ist, werden f, g und h als KNNs umgesetzt. Dabei teilen sich g und h einen Teil ihrer Netzarchitektur und ihrer Gewichte, sodass:

$$g(x) = g_2(g_1(x)) \quad h(x) = h_2(h_1(x)) \quad g_1(x) = h_1(x) \tag{2.21}$$

gilt. Da $h(x)$ die Kovarianzmatrix von $q_x(z)$ definiert, ist $h(x)$ eine quadratische Matrix [34]. Um die Berechnung zu vereinfachen und die Anzahl der Parameter zu reduzieren, wird angenommen, dass die Approximation von $p(z|x)$ und $q_x(z)$ eine multidimensionale Gauß-Verteilung mit diagonaler Kovarianzmatrix ist. Mit dieser Annahme ist $h(x)$ einfach der Vektor der diagonalen Elemente der Kovarianzmatrix und hat die gleiche Größe wie $g(x)$ [34]. In Abbildung 2.6 ist der Encoder-Teil des VAE entsprechend abgebildet.

Im Gegensatz zum Encoder-Teil des Modells $p(z|x)$, für das eine Gauß-Verteilung mit dem Mittelwert und der Kovarianz, welches beides Funktionen von x (g und h) sind, betrachtet wurden, nimmt das Modell für $p(x|z)$ eine Gauß-Verteilung mit fester Kovarianz an [34]. Die Funktion f der Variable v, die den Mittelwert der Gauß-Verteilung definiert, wird durch ein KNN umgesetzt. Abbildung 2.7 zeigt den Decoder-Teil des VAE in diesem Sinne.

Die Gesamtarchitektur erlangt man nun, wenn der Encoder-Teil und der Decoder-Teil

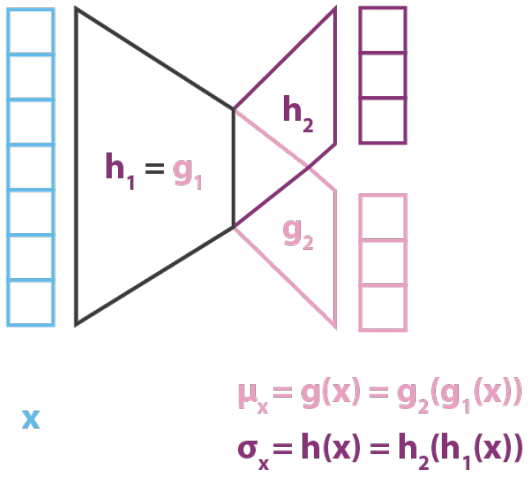


Abbildung 2.6: Encoder-Teil des Variational Autoencoders [34]

zusammengefügt werden. Dabei ist zu beachten, wie aus der Verteilung, die der Encoder als Ergebnis ausgibt, während des Trainings Stichproben gezogen werden können. Dieser Prozess muss so umgesetzt werden, dass das KNN durch Backpropagation den Fehler verringern kann. Dafür nutzt man das Reparameterisions-Verfahren (Reparametrisation Trick) [32], das ein Gradientenverfahren während des Trainings ermöglicht [34]. Das Reparameterisierungs-Verfahren funktioniert wie folgt: Angenommen, z ist eine kontinuierliche Zufallsvariable und $z \sim q_\phi(z|x)$ eine bedingte Verteilung. Dann ist es möglich, die Zufallsvariable z als deterministische Variable $z = g_\phi(\epsilon, x)$ auszudrücken. Dabei ist ϵ eine Hilfsvariable mit unabhängigem Randwert $p(\epsilon)$ und einer vektorwertigen Funktion $g_\phi(.)$, die durch $g_\phi(.)$ parameterisiert wird. Diese Reparameterisierung ist für den VAE sinnvoll, da sie genutzt werden kann, um eine Erwartung hinsichtlich $q_\phi(z|x)$ so umzuformulieren, dass die Monte-Carlo-Schätzung der Erwartung hinsichtlich ϕ differenzierbar ist [32]. In Abbildung 2.8 findet sich das Reparameterisierungs-Verfahren illustriert. Auf der linken Seite erkennt man die Probennahme ohne das Verfahren, sodass ein Problem für die Backpropagation entsteht, auf der rechten Seite erkennt man die Anwendung des Reparameterisierungs-Verfahrens.

Die Zielfunktion des VAE ist durch Gleichung 2.20 gegeben, in der die theoretische Erwartung durch eine Monte-Carlo-Approximation ersetzt wird. Wenn man diese Approximation betrachtet und $C = \frac{1}{2c}$ definiert, erhält man die Zielfunktion, welche aus einem Rekonstruktionsterm, einem Regularisierungsterm und einer Konstanten, welche die relativen Werte der beiden Terme definiert, besteht [34]. In Abbildung 2.9 ist diese Zielfunktion und die Gesamtarchitektur des VAE abgebildet.

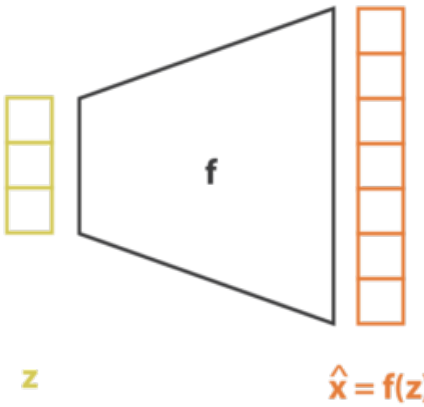


Abbildung 2.7: Decoder Teil des Variational Autoencoders [34]

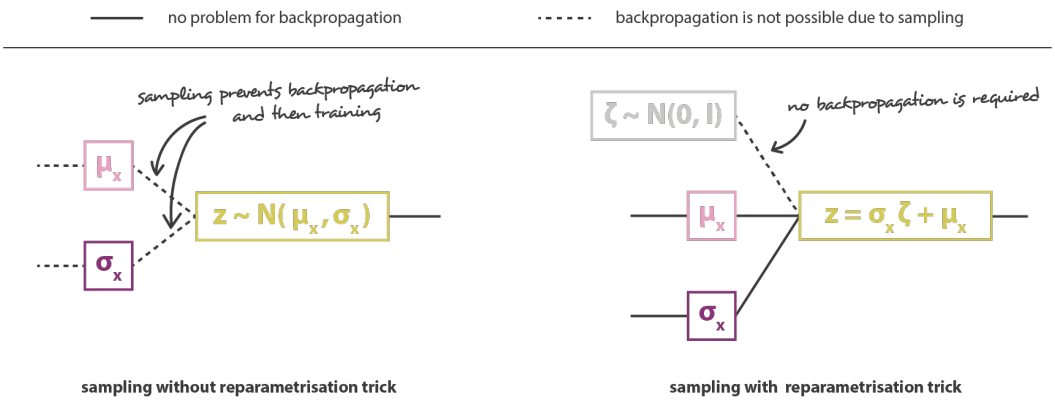


Abbildung 2.8: Illustration des Reparametrisierungs-Tricks [34]

Abbildung 2.10 zeigt ebenfalls die gesamte Architektur des VAE. Dort wird der VAE genutzt, um Bilder von Zahlen zu codieren und dann wieder zu decodieren. Es ist dort zu erkennen, wie der Encoder und der Decoder durch jeweils ein KNN umgesetzt sind. Darüber hinaus sieht man, wie der Encoder in der Mitte den Mittelwert und die Varianz ausgibt, woraufhin aus diesen Stichproben gezogen werden, die dann als Eingabe für den Decoder dienen, um das ursprüngliche Bild zu rekonstruieren.

2.3 Leistungsflussberechnung

Die Leistungsflussberechnung dient der Ermittlung des elektrischen Status eines Systems bei vorgegebener Belastung[36], wobei aus gegebenen Einspeiseleistungen und Verbraucher-

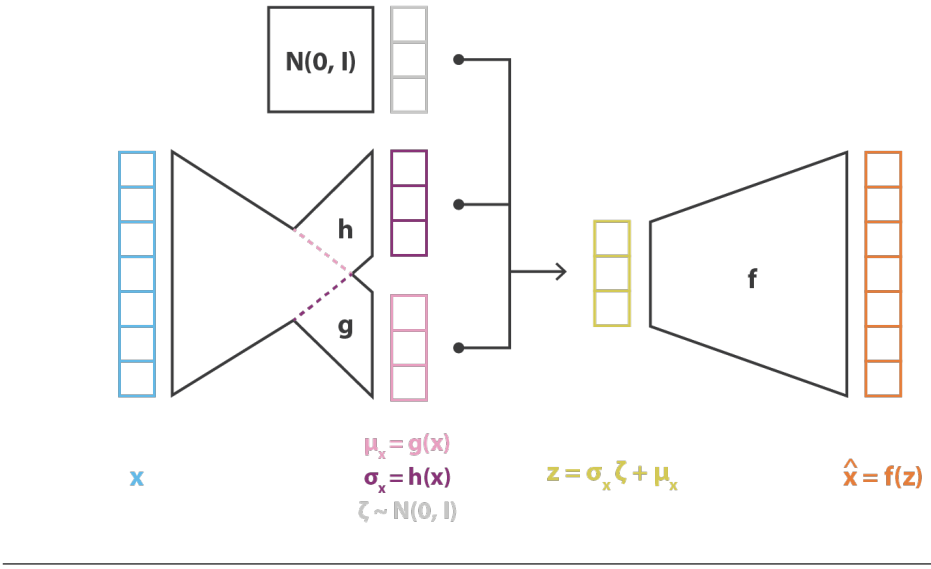


Abbildung 2.9: Abbildung des gesamten Variational Autoencoders[34]

lasten Wirk- und Blindleistungsflüsse ermittelt werden. Das Wissen über diese Leistungsflüsse ist relevant für eine optimale Netzplanung, für Netzbetrieb, Netzoptimierung und die Wirtschaftlichkeit des Netzes [37]. Die Leistungsflussberechnung erzeugt die Ergebnisse für die Knotenspannungen und Ströme der Betriebsmittel, die Wirk- und Blindleistungen auf den Leitungen und die Übertragungsverluste in Leitungen und Transformatoren [37]. Nach [37] werden Knoten in drei Arten unterteilt: Lastknoten, Speiseknoten und Bilanzknoten. Diese sind mit den gegebenen und gesuchten Werten in Tabelle 2.1 dargestellt. Jeder Knoten kann durch die Werte für die Wirkleistung P, die Blindleistung Q, den Spannungsbetrag $|U|$ und den Phasenwinkel der Spannung δ gekennzeichnet werden.

Knotenart	Gegeben	Gesucht
Leistungsknoten	P, Q	$\|U\|$, δ
Speiseknoten	P, $\|U\|$	Q, δ
Bilanzknoten	$\|U\|$, δ	P, Q

Tabelle 2.1: Gegebene und gesuchte Werte für verschiedene Knoten

Die Stromiteration und das Newton-Raphson-Verfahren sind zwei Verfahren, die bei der Leistungsflussberechnung zum Einsatz kommen [37]. Sind die Knotenströme zu Beginn bekannt, wird häufig die Stromiteration eingesetzt. Falls diese unbekannt sind, wird ein

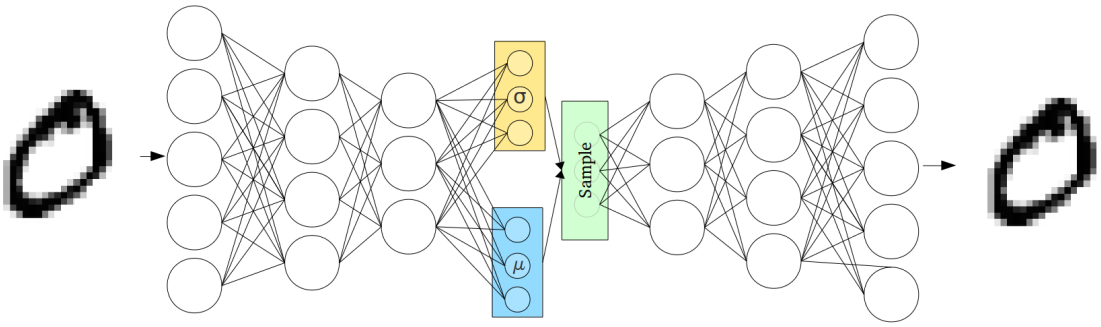


Abbildung 2.10: Beispiel der Architektur eines Variational Autoencoders für Bildaten von Zahlen [35]

Startvektor der Knotenspannungen geschätzt. Anhand dieser und der gegebenen Leistungswerte werden die Belastungsströme ermittelt. Dann werden iterativ verbesserte Knotenspannungen identifiziert, und mit diesen Knotenspannungen wird das Verfahren wiederholt. Sind die Knotenströme zu Beginn unbekannt, kommt das Newton-Raphson-Verfahren zur Anwendung. Hierbei errechnet man die Knotenspannungen direkt aus den Leistungen [37]. Das Newton-Raphson-Verfahren gilt als Grundlage für die meisten Verfahren, die nichtlineare algebraische Gleichungen lösen[38]. Genauer ist es ein Verfahren, um iterativ bessere Annäherungen an die Nullstellen von Funktionen zu finden [39]. Das Newton-Raphson-Verfahren funktioniert nach [39] wie folgt: Gegeben sei eine Funktion f über die reellen Zahlen x und ihre Ableitung f'. Zuerst wird eine Vermutung x_0 für die Nullstellen der Funktion f erhoben. Eine bessere Annäherung für x_1 ist:

$$x_1 = x_0 - \frac{f(x_0}{f'(x_0}. \tag{2.22}$$

Aus geometrischer Sicht ist $(x_1, 0)$ die Schnittstelle mit der x-Achse der Tangente zu dem Graphen von f bei $(x_0, f(x_0))$. Der Prozess wird als:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.23}$$

weitergeführt, bis ein ausreichend genauer Wert erreicht ist.

3 Verwandte Arbeiten

In [40] diagnostizieren Lai et al., dass es wenig Literatur zur Anwendung von DA für das Load Forecasting gebe. Zu Beginn dieses Kapitels werden daher zwei Publikationen vorgestellt, die eine Übersicht über die Forschung zu ML-Methoden in Energiesystemen vermitteln. In dieser Arbeit wird DA in die Teilbereiche transformationsbasierte Ansätze, statistische Ansätze und ML-Ansätze unterteilt und die betrachtete Forschungsliteratur in diese drei Bereiche eingeordnet.

Mosavi et al. [8] präsentieren aktuelle ML-Methoden in Energiesystemen und deren Entwicklung über die letzten zwanzig Jahre. In Abbildung 3.1 ist das Wachstum solcher Arbeiten abgebildet. Man erkennt einen stetigen Anstieg an Veröffentlichungen, der ab 2015 sehr stark zunimmt.

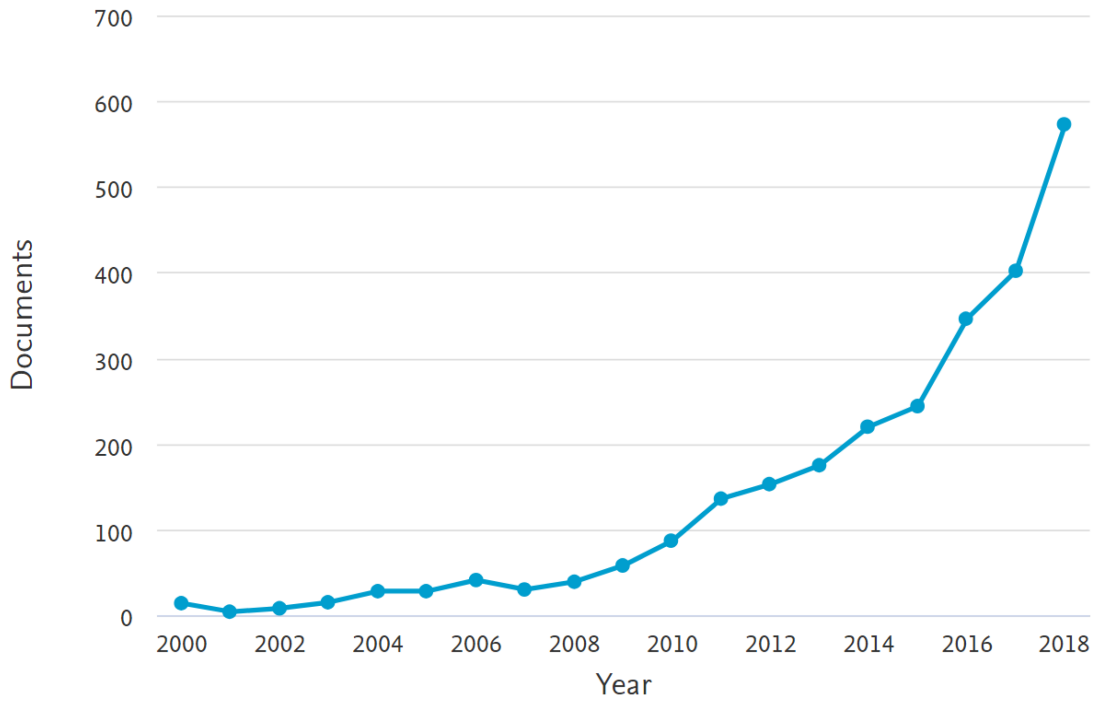


Abbildung 3.1: Der Wachstum der Anzahl an Artikeln der letzten zwei Jahrzehnte, die sich mit Machine-Learning-Methoden in Energiesystemen befassen [8].

Ruan et al. [41] stellen Arbeiten zur Optimierung von Energiesystemen vor. Diese Arbeiten basieren sehr häufig auf lerngestützten Methoden, wie in Abbildung 3.2 zu erkennen ist. Abbildung 3.2 zeigt den Forschungstrend von lerngestützter Optimierung in Energiesystemen und den Anteil der Publikationen, die sich mit der Optimierung in Energiesystemen befassen. Ab 2018 ist dort ein starker Anstieg an Veröffentlichungen zu erkennen. Der Anteil der lerngestützten Optimierung nimmt in den Jahren 2016 und 2017 leicht ab, steigt in 2018 und 2019 jedoch wieder an. [11] vergleicht verschiedene ML-Verfahren für die datengetriebene Annäherungen an Energiesysteme. Dabei wird das Problem beschrieben, dass die verfügbaren Daten auf ein Jahr beschränkt sind und Teile des Jahres nicht für das Training der ML-Verfahren genutzt werden können, da sie als Testdaten herangezogen werden müssen. [11] löst das Problem mit einer 12-fold-Cross-Validation, wobei das Verfahren zwölfmal wiederholt werden muss. Data Augmentation, wie sie hier vorgestellt wird, ist eine weitere Möglichkeit zur Lösung dieser Problematik.

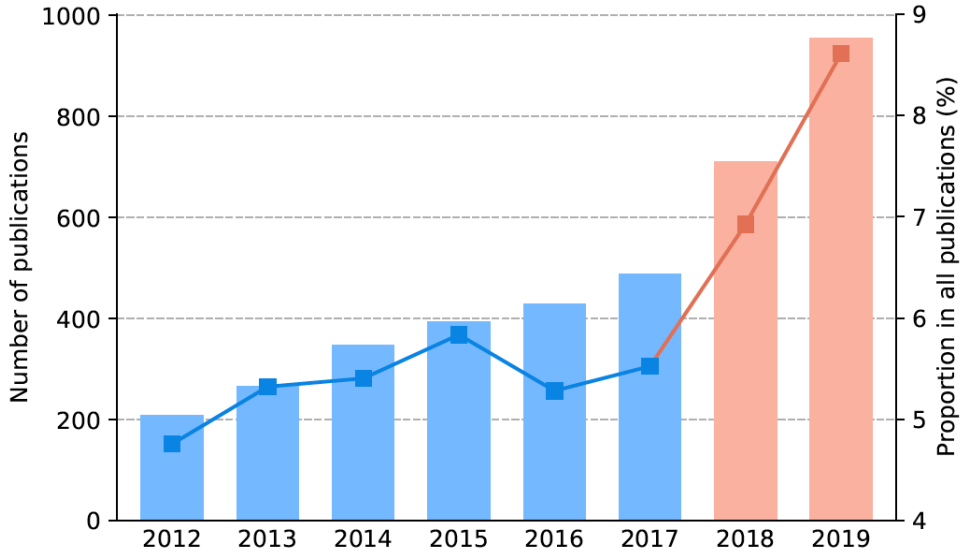


Abbildung 3.2: Forschungstrend lerngestützter Optimierung in Energiesystemen. Die Balken zeigen die Anzahl an Publikationen zu diesem Thema und die Linie den Anteil dieser Publikationen an allen Publikationen, die die Optimierung in Energiesystemen behandeln [41].

3.1 Transformationsbasierte Ansätze

Transformationsbasierte Ansätze beruhen auf dem Prinzip der Kopie und Transformation der Eingabedaten. Bei Bildern ist dies sehr intuitiv. Die Bilder werden, wie in Abbildung 1.2 zu sehen ist, verschoben, gespiegelt, rotiert, gestreckt oder gestaucht [42]. Diese Transformationen können teilweise für Energiedaten, wie sie in dieser Arbeit genutzt werden, übernommen werden. Numerische Werte würden dann vergleichsweise positiv oder negativ verschoben. Zeitreihen könnten in Sequenzen eingeteilt und vertauscht werden, um neue Zeitreihen zu generieren. Transformationsbasierte Ansätze haben den Vorteil, dass sie simpel umzusetzen sind. Der Nachteil dieser Ansätzen ist, dass sie Domänenkenntnisse vorrausetzen, da festgelegt werden muss, ob die Transformationen realistische Daten erzeugen, und dies sehr zeitaufwändig ist [43]. Auch wenn transformationsbasierte Ansätze zufällig angewendet werden, sollte der Zufall so gesteuert werden, dass er realistische Daten generiert. Ratner et al. [43] präsentieren einen Ansatz, um die Auswahl der geeigneten Transformationen zu automatisieren und so die manuelle Arbeit und den verbundenen Zeitaufwand zu reduzieren. In einer weiteren Arbeit von DeVries et al. [44] wird der Ansatz von simplen Transformationen angewendet. Allerdings nehmen die Autoren die Transformationen hier nicht in den Eingabedaten, sondern in einem erlernten Merkmalsraum vor.

3.2 Generative Modelle

Anstatt simple Transformationen zu nutzen, die zum Teil zufallsbasiert sind oder die historischen Daten mischen, ist es möglich, Daten aus Funktionsverteilungen mit generativen Modellen zu genwinnen. Diese generativen Modelle werden hier in zwei Kategorien, die ML-Ansätze und die statistischen Ansätze, unterteilt.

3.2.1 Machine-Learning-Ansätze

Die ML-Ansätze in der Data Augmentation nutzen ML-Methoden wie KNNs, um synthetische Daten aus historischen Daten zu erzeugen. Es hat in den letzten Jahren vielversprechende Durchbrüche in der, auf ML basierenden, DA gegeben. Viele dieser Enwicklungen haben im Bereich der Bilderkennung stattgefunden, wie in der Arbeit von Moe et al. [45] oder Pestei et al. [42]. Aber auch in vielen anderen Bereichen, wie der Text Classification[46], der Erzeugung von medizinischen Zeitreihendaten[47] und auch im Energiebereich, wie in den Arbeiten von Zheng et al. [48], Zhang et al. [17] oder Gong et al. [16]. Als besonders erfolgreiche Methoden haben sich die Generative Adversarial Nets (GANs)[29] und der Variational Autoencoder (VAE)[32] erwiesen. Diese beiden Methoden sind weit verbreitet

und werden auch im Energiebereich häufig zur Erzeugung synthetischer Daten genutzt [49, 16, 50, 51, 52, 17].

GANs werden sehr häufig für Bilddaten verwendet, wenngleich es auch bereits Forschung zur Anwendung von GANs in anderen Bereichen gegeben hat. Zum Beispiel werden von Mogren [53] Continous Recurrent Neuronal Networks mit Adversarial Training genutzt, um Musik zu generieren. Yu et al. [52] nutzen sequenzielle Generative Adversarial Networks (SeqGANs), um Gedichte, Sprache und Musik zu generieren. Diese SeqGANs haben einen Reinforcement Learning Generator, um das Problem von GANs, sequenzielle Daten zu generieren, zu umgehen. Fekri et al. [49] haben in ihrer Arbeit GANs zu Recurrent Generative Adversarial Networks (R-GANs) weiterentwickelt. Durch Experimente demonstrieren sie, dass die so erzeugten Daten zum Training von Energieprognosemodellen verwendet werden können. In R-GANs werden die Convolutional Neuronal Networks (CNN) aus dem GAN durch Recurrent Neuronal Networks[1] ersetzt, da diese in der Lage sind, zeitliche Abhängigkeit zu behandeln, wie sie häufig im Energiebereich zu finden sind. Zheng et al. [48] stellen einen Ansatz vor, bei dem ein GAN genutzt wird, um realistische Phasor Measurement Unit (PMU)-Datenströme zu erzeugen. Eine Besonderheit dieses Ansatzes ist, dass die synthetisch erzeugten PMU-Daten, anders als die in GANs häufig genutzten Bilddaten, bestimmten physischen Bedingungen, wie den Kirchhoffschen Regeln, unterliegen müssen. Aus der, in der Arbeit durchgeführten, Evaluation erkennt man, dass die synthetischen Daten erfolgreich in datengetriebenen Methoden verwendet werden können. In einer anderen Arbeit von Zhang et al. [17] wird das Problem, ML-Methoden in Smart Grids anzuwenden, durch einen weiteren GAN-Ansatz, der auf Zeitreihendaten, ausgelegt ist, angegangen. Der Test erfolgt statistisch und durch Anwendung auf ML-Aufgaben, wie Time Series Clustering und Load Prediction.

VAEs wurden bislang, genau wie GANs, sehr häufig zur Data Augmentation von Bilddaten genutzt[54, 55, 56, 57]. Aber auch in anderen Bereichen wie zur DA von Sprachdaten [50, 58] oder medizinischen Daten [42] kommen VAEs zum Einsatz. Im Energiebereich ist die Anwendung von VAEs aktuell noch geringer. Gong et al. [16] nutzen einen VAE dazu, Trainingsdaten für ein Deep Neuronal Network (DNN) mit künstlich generierten Daten zu vergrößern. Das DNN soll trainiert werden, um Strom-Diebstähle in Energiedaten zu erkennen. Das Encodernetz haben Gong et al. durch ein CNN realisiert, da sich ein CNN gut für eine Merkmalsextraktion eignet [59]. Pan et al. [60] nutzen einen VAE, um Lastenprofile für Elektrofahrzeuge zu erzeugen. So kann die Wahrscheinlichkeitsverteilung der historischen Daten, die zur Erzeugung der Lastenprofile nötig ist, automatisch erlernt werden, ohne dass ein manuelles Eingreifen nötig ist. In der Arbeit von Chakraborty et

[1] Als Recurrent Neuronal Networks werden neuronale Netze bezeichnet, deren Neuronen Verbindungen von einer Schicht zu Neuronen derselben oder einer vorangegangenen Schicht besitzen.

al. [61] wird ein VAE genutzt, um Modelle von virtuellen Batterien zu generieren. Der VAE bildet mit gegebenen Sensor- und Zählerdaten Punktschätzungen der Parameter der virtuellen Batterien und Konfidenzintervalle um diese Werte.

Da in VAEs statistische Methoden eine wichtige Rolle spielen, könnte man überlegen, diese auch im nächsten Teil den statistischen Ansätzen zuzuordnen. In dieser Arbeit werden sie aber aufgrund der Verwendung von KNNs zu den ML-Ansätzen gezählt.

3.2.2 Statistische Ansätze

Bei den statistischen Ansätzen versucht man, Regeln zu finden, welche die Eingabedaten durch statistische Werte beschreiben, und aufgrund dieser Regeln neue Daten zu erzeugen. Dieses Verfahren ähnelt dem des VAE, allerdings automatisiert der VAE dieses Verfahren durch die Nutzung von KNNs. Bei den statistischen Ansätzen ist in der Regel eine manuelle Bestimmung der Regeln nötig. Zhou et al. [62] wenden die Patient Rule Induction Space Methode (Prism)[63] an, um Regeln aus den Eingabedaten zu ziehen, die sie dann zur Erzeugung von Daten, zur Identifikation von transienter Stabilität und instabielen Generatoren von Energiesystemen nutzen. Da bei ML-Methoden zur Analyse der transienten Stabilität die Anzahl instabiler Instanzen immer geringer als die der stabilen Instanzen ist, kommt es zum Underfitting[2]. Zhou et al. können dieses Problem durch die von ihnen vorgestellte DA-Methode verhindern. In einer Arbeit von Ngoko et al. [64] werden Markov-Matrizen genutzt, um aus realen aufgezeichneten Solardaten neue synthetische Solar-Daten zu generieren. Chen und Rabiti [65] haben synthetische Windgeschwindigkeitsszenarien durch die Anwendung von Fourierreihen und dem Autoregressive Moving Average Model auf reale aufgezeichnete Daten generiert. Dabei wurden die Fourierreihen herangezogen, um saisonale Trends zu erkennen, während das Autoregressive Moving Average Model die Autokorrelation zwischen übrigen Rückständen ermöglicht.

[2]Beim Underfitting hat ein statistisches Modell nicht alle Eigenschaften der Daten erlernt und liefert dadurch schlechte Generalisierung und unzuverlässige Prognosen.

4 Implementierung

4.1 Auswahl der Methoden

Nach Betrachtung der verwandten Arbeiten muss es in diesem Kapitel darum gehen, die Methoden auszuwählen, die daraufhin überprüft werden sollen, ob DA ein ML-Verfahren aus dem Energiebereich verbessern kann. Dabei werden mehrere Methoden betrachtet, um zu vergleichen, wie sich verschiedene DA-Methoden auf das Endergebnis auswirken.

Aufgrund des Erfolges von ML-Ansätzen zur DA, ihrer breiten Anwendungsmöglichkeiten und ihres hohen Automatisierungsgrades wird in dieser Arbeit ein ML-Ansatz implementiert. Die beiden erfolgreichsten Ansätze hierbei sind die GANs und die VAEs. Im Rahmen dieser Arbeit wird ein VAE umgesetzt, weil GANs den VAEs zwar oft in der Generierung von Daten überlegen, aber auch schwerer umzusetzen sind[66]. Da es das Ziel dieser Arbeit ist zu überprüfen, ob ML-Verfahren durch DA verbessert werden können, reicht ein VAE hier aus.

Zum Vergleich verschiedener DA-Ansätze werden ebenfalls zwei transformationsbasierte Ansätze ausgewählt, davon soll ein Ansatz die Werte der Daten ins Positive oder Negative verschieben und der andere die Daten in Sequenzen einteilen und vermischen. Ein wichtiger Grund für die Auswahl der beiden transformationsbasierten Ansätze ist, dass sie sehr leicht umzusetzen und nachzuvollziehen sind. Bei dem Vergleich mit dem VAE kann dann diskutiert werden, ob sich der Mehraufwand in der Umsetzung eines VAEs lohnt. Die transformationsbasierten Ansätze werden ebenfalls daraufhin untersucht, ob sie das Ergebnis eines ML-Verfahren verbessern können. Dazu werden die Auswirkungen der beiden transformationsbasierten Verfahren mit denen des VAE verglichen. So kann auch die zweite Forschungsfrage beantwortet und der Entwicklungsaufwand, der Aufwand für ihre Anwendung und die Flexibilität der Anwendung verglichen und diskutiert werden. Zur Überprüfung der drei Umsetzungen für die DA dient eine lineare Regression. Die lineare Regression ist eines der einfachsten ML-Verfahren und kann vielseitig angewendet werden. Da der Schwerpunkt dieser Arbeit auf der Untersuchung der Auswirkungen von DA auf ML-Verfahren liegt, reicht ein einfaches ML-Verfahren wie die lineare Regression für diese Untersuchung aus. Genauer soll hingegen der Einfluss von DA auf ML-Verfahren aus dem

Energiebereich untersucht werden. Deshalb wird die lineare Regression genutzt, um die Leistungsflussberechnung zu *erlernen*. Die Leistungsflussberechnung ist ein wichtiges Verfahren im Energiebereich und Grundlage vieler Analyseaufgaben. Die lineare Regression erlernt, aus den Eingabedaten für die Leistungsflussberechnung die entsprechenden Ergebnisse der Leistungsflussberechnung zu erzeugen. Dabei ist zu erwarten, dass die lineare Regression deutlich schnellere Ergebnisse für die Leistungsflussberechnung liefert als die Durchführung der Leistungsflussberechnung selbst. Die durch die lineare Regression erzeugten Ergebnisse werden dann mit den Ergebnissen der eigentlichen Leistungsflussberechnung verglichen. Es ist davon auszugehen, dass die Ergebnisse der linearen Regression zwar nicht exakt denen der Leistungsflussberechnung entsprechen, aber dennoch ausreichend präzise sind. Wichtig ist jedoch, dass es noch Verbesserungspotenzial gibt, welches durch die DA ausgenutzt werden kann. Anhand der erzeugten Ergebnisse ohne die DA und der Ergebnisse mit den verschiedenen DA-Ansätzen können diese evaluiert werden.

4.2 Programmiersprache und Frameworks

Die Implementierung wird in dieser Arbeit mit Python umgesetzt. Python ist zur populärsten Programmiersprache für viele ML-Anwendungen geworden [27]. Es handelt sich um eine verständliche und gut lesbare Sprache, die einen zuverlässigen Code liefert. Ihre Einfachheit hilft bei der Implementierung von komplexen Algorithmen und vielseitigen Workflows, wie sie im ML-Bereich häufig umgesetzt werden. Außerdem gilt Python als die flexibelste Programmiersprache für ML-Anwendungen [67]. Diese Flexibilität gibt Entwickler*innen mehr Kontrolle und Komfort und reduziert die Wahrscheinlichkeit von Fehlern. Des Weiteren bietet Python viele Bibliotheken und Frameworks, die für die Implementierung von ML-Anwendungen sehr wichtig sind, sowie eine Plattformunabhängigkeit.

Zur Beschaffung der Daten und zur Umsetzung von Energieanalyseaufgaben werden in dieser Arbeit die beiden Frameworks Simbench[1] und Pandapower[2] genutzt. Simbench ist ein Forschungsprojekt zur Entwicklung eines Benchmark-Datensatzes, um innovative Lösungen im Bereich der Netzanalyse, Netzplanung und Netzbetriebsführung zu vergleichen. Simbench wurde als Teil des German Federal Government's 6th Energy Research Program Research for an Environmentally Friendly, Reliable and Affordable Energy Supply über dreieinhalb Jahre geführt. Der Simbench-Datensatz ist als Hauptergebnis des Projekts vollständig und öffentlich zugänglich. Der Datensatz beinhaltet elektrische Parameter zur statischen Modellierung von Stromnetzen und umfasst Spannungsebenen von der Niederbis zur Höchstspannung. In dieser Arbeit werden im Besonderen die Daten für die Lasten

[1] https://simbench.de/de/ – Abgerufen am 30.03.2021
[2] http://www.pandapower.org/ – Abgerufen am 05.04.2021

Technologie	Version
Python	3.8.8
Simbench	1.2.0
Pandapower	2.5.0
Numpy	1.19.2
Pandas	1.2.3
Scikit-learn	0.24.1
Tensorflow	2.4.1

Tabelle 4.1: Tabellarische Darstellung des verwendeten Technologiestacks

und Einspeisungen verwendet. Diese Daten liegen als Zeitreihen vor.

Pandapower ist ein auf Python basierendes, BSD-lizensiertes Tool für die Analyse von Energiesystemen. Es zielt auf die Automatisierung von statischer und quasi-statischer Analyse und die Optimierung von balancierten Energiesystemen ab. Unter anderem enthält Pandapower eine Implementierung für die Leistungsflussberechnung, welche in dieser Arbeit zur Erzeugung von Trainings- und Testdaten verwendet wird. Außerdem beinhaltet Pandapower bereits einige Simbench-Energienetze.

Für die Verwaltung und die Analyse der Daten werden die beiden Bibliotheken Pandas[3] und Numpy[4] genutzt. Pandas enthält Datenstrukturen und Operatoren für den Zugriff auf numerische Tabellen und Zeitreihen. Pandapower nutzt und vereint Pandas mit dem Pypower[5]-Framework, welches die Berechnung des Leistungsflusses und des optimalen Leistungsflusses ermöglicht. Numpy liefert eine einfache Handhabung von Vektoren, Matrizen oder großen mehrdimensionalen Arrays. Neben den Datenstrukturen bietet Numpy auch effiziente Funktionen für numerische Berechnungen an.

Zur Umsetzung des VAE wird das Framework TensorFlow[6] genutzt, ein Framework zur datenstromorientierten Programmierung, dass sehr häufig im ML-Bereich Anwendung findet. Ebenfalls wird zur Umsetzung des VAE, der linearen Regression und der Auswertung der Ergebnisse das Scikit-learn-Framework[7] genutzt. Scikit-learn ist eine Bibliothek für das Machine Learning in Python mit verschiedenen Klassifikations-, Regressions- und Clustering-Algorithmen.

Tabelle 4.1 führt die verwendeten Technologien und die entsprechenden Versionen auf.

[3] https://pandas.pydata.org/ – Abgerufen am 25.03.2021
[4] https://numpy.org – Abgerufen am 25.03.2021
[5] https://github.com/rwl/PYPOWER – Abgerufen am 25.03.2021
[6] https://tensorflow.org/ – Abgerufen am 05.04.2021
[7] scikit-learn.org/ – Abgerufen am 05.04.2021

4.3 Datengrundlage

Die Algorithmen und Ideen zu vielen erfolgreichen ML-Methoden der letzen Jahre existieren schon länger. Ausschlaggebend für den Erfolg dieser Methoden waren jedoch die neue Verfügbarkeit großer Datenmengen und neuere Ressourcen, um diese Datatenmengen zu verarbeiten [68]. In dieser Arbeit werden Energiedaten benötigt, um mithilfe der linearen Regression die Leistungsflussberechnung für ein Energienetz zu lernen. Dafür werden die Frameworks Simbench und Pandapower genutzt.

Die hier herangezogenen Daten stammen aus einem Simbench-Netz. Listing 4.1 bietet eine Übersicht über die Komponenten des Simbench-Netzes "1-MV-rural–0-sw". Simbench nutzt Codes zur Bennenung der Energienetze. Der Code des verwendeten Energienetzes kann wie folgt interpretiert werden: Es wird die Simbench-Version 1 genutzt. Es ist ein Mittelspannungsnetz eines ländlichen Raums abgebildet. Es wird ein gegenwertiges Netz dargestellt (keines, das etwa einer zukünftigen Entwicklung entsprechen würde).

```
This pandapower network includes the following parameter tables:
   - bus (97 elements)
   - load (96 elements)
   - sgen (102 elements)
   - switch (204 elements)
   - ext_grid (1 element)
   - line (99 elements)
   - trafo (2 elements)
   - measurement (37 elements)
   - bus_geodata (97 elements)
   - substation (1 element)
   - loadcases (6 elements)
```

Listing 4.1: Abruf der Komponenten des verwendeten Simbench-Netzes 1-MV-rural–0-sw über Pandapower

Die Komponenten des Energienetzes besitzen mehrere Attribute, beispielsweise sind die Attribute für die Lasten in Abbildung 4.1 abgebildet.

Für die Berechnung des Leistungsflusses nutzt Pandapower nur die Daten für die Wirkleistung und für die Blindleistung des Lastenprofils sowie die Wirkleistung des Erzeugungsprofils. Diese Daten sind unter *load* und *sgen* des Netzes, jeweils als Pandas-DataFrame, für die Wirkleistung und die Blindleistung zu finden. Wobei für das Erzeugungsprofil nur die Wirkleistung angegeben ist. Die drei Tabellen bestehen aus etwa 35.000 Zeilen. Jede Zeile steht für einen 15-Minuten-Abschnitt eines Jahres. Insgesamt ist genau ein Jahr abgebildet. Abbildung 4.2 zeigt beispielsweise den Pandas DataFrame für die Wirkleistung des Lastenprofils.

Für die Bezeichnung der DataFrames wird die folgende Konvention verwendet: Zum einen

	name	bus	p_mw	q_mvar	const_z_percent
0	HV1_MV1.101_load	2	0.230	0.0909	0.0
1	MV1.101 Load 2	4	0.080	0.0316	0.0
2	MV1.101 Load 3	5	0.080	0.0316	0.0
3	MV1.101 Load 4	6	0.080	0.0316	0.0
4	MV1.101 Load 5	7	0.080	0.0316	0.0
..	...	...	...	...	...
91	MV1.101 MV Load 1	15	0.350	0.1383	0.0
92	MV1.101 MV Load 2	54	0.560	0.2213	0.0
93	MV1.101 MV Load 3	64	0.450	0.1779	0.0
94	MV1.101 MV Load 4	27	0.270	0.1067	0.0
95	MV1.101 MV Load 5	94	0.295	0.1166	0.0

Abbildung 4.1: Ausschnitt der Lasten des Pandapower-Netzes

wird der Name des Profils abgekürzt genutzt, hier zum Beispiel *load*, zum anderen wird die Einheit als *p_mw* für die Wirkleistung und *q_mvar* für die Blindleisung angegeben.

Als Ergebnis der Lastflussberechnung fügt Pandapower weitere Parameter zum Netz hinzu und versieht diese Parameter mit dem Präfix *res_*. Für die Umsetzung der linearen Regression werden in dieser Arbeit die Ergebnistabellen für die Spannungsgrößen der Busse und die Auslastung der Leitungen genutzt. Abbildung 4.3 gibt die Tabelle zum Ergebnis für die Spannungsgrößen der Busse wieder.

Die Daten lassen das Problem hervortreten, das als Motivation für die Anwendung der DA dient: Die verfügbaren Daten beschränken sich lediglich auf ein Jahr. Bei ML-Verfahren ist es allerdings üblich, die Datengrundlage in mindestens einen Trainings- und einen Testdatensatz zu unterteilen, da man die Testdaten benötigt, um ML-Verfahren zu bewerten. Wenn man nun die Daten aus einem Jahr aufteilen muss, geht ein Teil des Jahres für das Training verloren. So hat das entsprechende ML-Verfahren nie die Möglichkeit, den fehlenden Teil des Jahres für das Training zu nutzen. Mit DA kann man hier ansetzen und durch eine Vergrößerung des Datensatzes den negativen Einfluss der fehlenden Daten verringern oder mit einer entsprechend guten DA-Methode die fehlenden Daten annähernd realistisch generieren.

4.4 Implementierungsarchitektur

Abbildung 4.4 zeigt eine Übersicht über die gesamte Architektur der Implementierung. Sie ist in die vier Bereiche unterteilt, auf die im Folgenden einzugehen sein wird: Datengenerierung (Simbench/Pandapower), Datenvorbereitung (Data Preparation), Data Augmentation und Datenevaluation (Data Evaluation).

{('load', 'p_mw'):	0	1	2	3	4	5	93	94	95
0	0.080769	0.019711	0.019711	0.019711	0.019711	0.032463	0.032463	0.032463	6613
1	0.080400	0.027662	0.027662	0.027662	0.027662	0.023630	0.023630	0.023630	7829
2	0.092308	0.032082	0.032082	0.032082	0.032082	0.028326	0.028326	0.028326	3685
3	0.089231	0.032017	0.032017	0.032017	0.032017	0.023973	0.023973	0.023973	6613
4	0.089231	0.032734	0.032734	0.032734	0.032734	0.025785	0.025785	0.025785	5633
...	...	...	...	...	...	...	...	...	...
35131	0.082708	0.025873	0.025873	0.025873	0.025873	0.021096	0.021096	0.021096	3933
35132	0.094246	0.024076	0.024076	0.024076	0.024076	0.020901	0.020901	0.020901	0273
35133	0.084246	0.017258	0.017258	0.017258	0.017258	0.020200	0.020200	0.020200	8809
35134	0.071539	0.019816	0.019816	0.019816	0.019816	0.019823	0.019823	0.019823	2953
35135	0.094246	0.020116	0.020116	0.020116	0.020116	0.021106	0.021106	0.021106	3685

Abbildung 4.2: Ausschnitt des Pandas DataFrame, der die Werte für die Wirkleistung des Lastenprofils darstellt.

4.5 Datengenerierung

Für die Datengenerierung werden die bereits erwähnten Frameworks Simbench und Pandapower verwendet. Simbench ermöglicht es, die Energieprofile des Netzes zu laden. Für die Leistungsflussberechnung sind von Pandapower wiederum die Daten für die Wirkleistung und für die Blindleistung des Lastenprofils sowie die Wirkleistung des Erzeugungsprofils erforderlich. Diese drei Daten-Tabellen werden aus den Energieprofilen gespeichert und dann in der Datenvorbereitung (Data Preparation) für die weitere Nutzung vorbereitet.

4.6 Datenvorbereitung

Im ersten Schritt der Datenvorbereitung unterteilt man die drei Daten-Tabellen in Trainings- und Testdaten. Diese Unterteilung ist wichtig, denn die Trainingsdaten werden vom VAE für das Training und für das Generieren synthetischer Daten genutzt. Es ist zu beachten, dass der VAE nur die Trainingsdaten nutzt und keinen Zugriff auf die Testdaten hat. Nur so kann sichergestellt werden, dass die spätere Evaluation nicht verfälscht wird. Die Trainingsdaten stehen ebenfalls der linearen Regression zur Verfügung, um ein Training auf Trainingsdaten durchzuführen, die nicht durch DA erweitert wurden und so zum Ver-

	vm_pu	va_degree	p_mw	q_mvar
0	1.025000	0.000000	8.088519	-5.211553
1	1.025000	0.000000	0.000000	0.000000
2	1.013663	1.095404	-1.770000	0.090900
3	1.013663	1.095404	0.000000	0.000000
4	1.014625	1.125712	-0.080000	0.031600
..	...	...	...	...
92	1.006111	1.611881	-0.080000	0.031600
93	1.005814	1.620095	0.057000	0.079800
94	1.005631	1.624784	0.352000	0.196400
95	1.005629	1.627076	0.057000	0.079800
96	1.005661	1.628346	-0.080000	0.031600

Abbildung 4.3: Ergebnistabelle für die Spannungsgrößen für einen Zeitabschnitt.

gleich in der Evaluation dienen können. Die Testdaten werden erst in der Evaluation der lineare Regression wiederverwendet. So kann sicher überprüft werden, wie das Verfahren bei unbekannte Daten funktioniert. Denn Ziel des Trainings der linearen Regression ist es, für unbekannte Daten eine möglichst genaue Annäherung an die Ergebnisse, die eine Lastflussberechnung ergeben würde, zu liefern.

Ausgewählt werden 70 Prozent der Daten für die Trainingsdaten, die restlichen 30 Prozent der Daten als Testdaten. Dabei werden die Daten jedoch vorher nicht gemischt. Das bedeutet, dass die Trainingsdaten die Zeitschritte für die ersten 70 Prozent des Jahres widerspiegeln und die Testdaten die letzten 30 Prozent des Jahres. Ein alternativer Ansatz wäre hier, die Daten vorher zu mischen, um so Zeitschritte des gesamten Jahres in den Trainingsdaten zu berücksichtigen. Dies würde vermutlich das Endergebnis des hier umgesetzten Verfahrens verbessern, jedoch keine Aussage darüber erlauben, ob durch DA ML-Verfahren für unbekannte Zeitabschnitte verbessert werden können.

Dieses Problem wird beispielsweise in [11] beschrieben. Nach der Unterteilung in Test- und Trainingsdaten ergeben sich sechs Tabellen, da jede der bereits erwähnten drei Tabellen entsprechend in Test- und Trainingsdaten zu unterteilen ist. Im Folgenden werden die Eingabedaten für die Lastflussberechnung mit x bezeichnet, die Ergebnisse der Lastflussberechnung mit y. Falls zusätzlich die entsprechenden Test- oder Trainingsdaten von x oder y gemeint sind, werden x und y entsprechen als *x_test/ y_test* oder *x_train/ y_train* gekennzeichnet. Für *x_test* wird dann mit Pandapower die Lastflussberechnung durchgeführt. *x_test* und die Ergebnisse *y_test* der Lastflussberechnung werden später im Evaluationsschritt (Data Evaluation) wiederverwendet. Zur Lastflussberechnung von

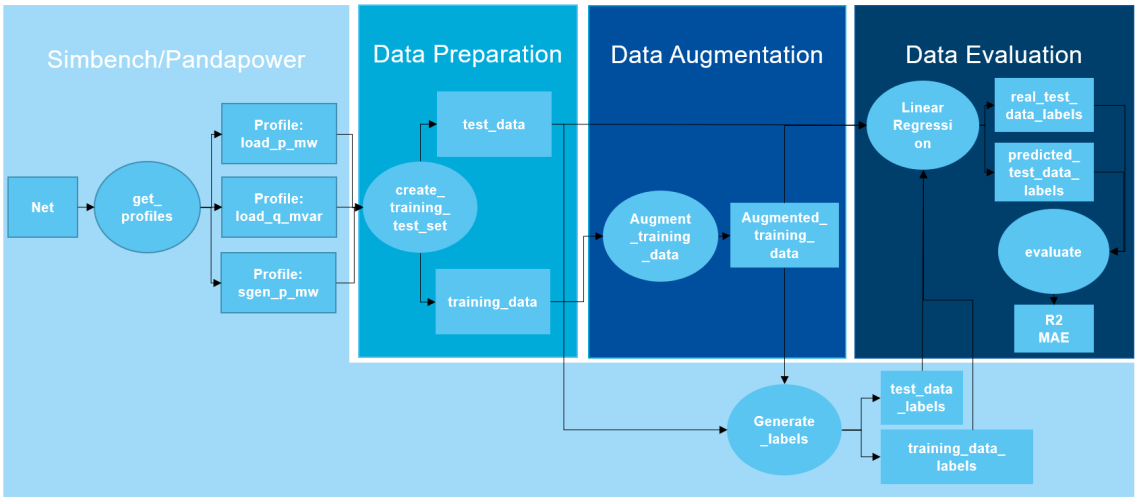


Abbildung 4.4: Architektur der Implementierung

Zeitreihen verwendet Pandapower die Time Series Simulation[8]. Dabei werden Controller genutzt, um die Werte für verschiedene Elemente in jedem Zeitschritt zu aktualisieren, insgesamt drei Controller an der Zahl, einen für jede Tabelle von x. Diese Controller werden dem ausgewählten Netz (das aus dem die Energieprofile bereits gezogen wurden) hinzugefügt. Mit der Pandapower-Funktion *run_timeseries(net, timesteps)* kann dann die Lastflussberechnung für alle Zeitschritte durchgeführt werden, wobei *net* das ausgewählte Netz repräsentiert und *timesteps* die Anzahl der Zeitschritte, in unserem Fall 10540 (30 Prozent von 35136), was 30 Prozent eines Jahres entspricht. Die Ergebnisse werden dann als *.csv* Datei abgespeichert und enthalten ebenso viele Zeitschritte wie *x_test*. Für die noch nicht weiterverwendeten *x_train*-Daten wird die Lastflussberechnung noch nicht durchgeführt. *x_train* wird erst im nächsten Schritt der Data Augmentation weiterverarbeitet.

4.7 Data Augmentation

Die DA wird in dieser Arbeit im Rahmen dreier verschiedener Ansätze umgesetzt, die später zum Vergleich zur Verfügung stehen. Die *x_train* Daten werden in diesem Schritt durch synthetische Daten erweitert oder ersetzt. Danach wird für die Ausgabe der DA, also die neuen *x_train* Daten, genauso wie bei *x_test* die Lastflussrechnung mit Pandapower durchgeführt. *x_train* kann allerdings durch die synthetischen Daten größer sein als vorher, weshalb sich entsprechend auch die *timesteps* erhöhen. Im Folgenden wird die Implementierung der drei Ansätze zur DA genauer dargestellt. Die ersten beiden Ansätze

[8] https://pandapower.readthedocs.io/en/v2.2.2/timeseries.html – Abgerufen am 01.04.2021

gehören zu den transformationsbasierten Ansätzen, der dritte Ansatz ist ein VAE, der zu den ML-Ansätzen zählt.

4.7.1 Transformation durch prozentuale Änderungen der Werte der Trainingsdaten

Bei diesem transformationsbasierten Ansatz werden alle Werte von *x_train* kopiert und um einen ausgewählten Prozentsatz zufällig positiv oder negativ verschoben. Dazu werden sie entweder um einen Prozentsatz ihrer eigenen Werte addiert oder subtrahiert. Es finden für die Evaluation verschiedene Prozentsätze Verwendung. Der Prozentsatz wird für jeden Wert zufällig neu aus einem Intervall von 0 bis zu dem ausgewählten Prozentsatz ausgewählt. Dabei wird überprüft, ob der resultierende Wert größer als der größte Wert aus *x_train* oder kleiner als der kleinste Wert aus *x_train* ist. Wenn dies der Fall ist, setzt man den Wert entsprechend gleich dem größten oder dem kleinsten Wert aus *x_train*. Die erzeugten synthetischen *x_train* entsprechen der gleichen Tabellenform (Zeilen, Spalten) wie die ursprünglichen *x_train*, wobei aber die Werte transformiert wurden. Dieser Prozess wird mehrfach wiederholt. Die entstandenen neuen *x_train* werden zu einem gesamten *x_train* zusammengefügt und dann für die Erzeugung der *y_train* durch die Leistungsflussberechnung zur Verfügung gestellt. In Listing 4.2 ist beispielhaft zu erkennen, wie die DA bei diesem Ansatz funktioniert. Man sieht, dass die Zahlen aus *a* zufällig um einen Prozentwert zwischen 0 und 10 Prozent positiv oder negativ transformiert wurden. Bei zwei der Zahlen aus *a* wäre die resultierende Zahl größer beziehungsweise kleiner als die größte oder kleinste Zahl gewesen. Deshalb wurden die auf die entsprechenden größten oder kleinsten Werte gesetzt. In *a* wurden die Werte auf die ursprünglichen Werte gesetzt, da diese die bereits größten oder kleinsten Werte in *a* waren.

```
a = [1, 1, 2, 2, 3, 3]
a_aug = aug_prozentuale_transformationen(daten=a, prozent= 0.1)
a_aug
    [1.05, 1.0, 1.8, 1.85, 3.0, 2.7]
```

Listing 4.2: Beispiel für die Transformation durch prozentuale Änderungen

4.7.2 Transformation durch zufälliges Vertauschen von Sequenzen der Trainingsdaten

In diesem transformationsbasierten Ansatz geht es nicht um eine Veränderung der einzelnen Werte, sondern um eine der Reihenfolge der Zeitreihe. Dazu werden die Zeitreihen von *x_train* in Sequenzen eingeteilt. Für diese Sequenzen wird zuerst eine Größe festgelegt. Für die Auswertung werden verschiedene Größen für die Sequenzen überprüft. Die Tabellen von *x_train* werden in kleinere Tabellen, die Sequenzen, aufgeteilt, zufällig ver-

tauscht und wieder zusammengefügt. Dadurch entstehen neue Tabellen für *x_train*, die eine andere Reihenfolge der Zeitschritte aufweisen als die ursprünglichen *x_train*-Tabellen. Da die drei *x_train*-Tabellen unabhängig voneinander gemischt werden, entstehen neue Wertekombinationen für die Zeitabschnitte. Deshalb ergeben sich weiterführend auch in der Leistungsflussberechnung neue Ergebnisse *y_train* für diese neuen Wertekombinationen, und so hat man neue synthetische Trainingsdaten. In Listing 4.3 ist das Einteilen in Sequenzen und das Vertauschen beispielhaft dargestellt. Es wird eine Sequenzgröße von zwei verwendet. Dadurch wird *a* in drei Sequenzen unterteilt, die dann in ihrer Reihenfolge vertauscht und wieder zusammengefügt werden. Die Simbenchdaten werden anhand der Zeitschritte in Sequenzen unterteilt. In Listing 4.3 würden dann die Zahlen 1 bis 6 in *a* beispielhaft für jeweils einen Zeitschritt stehen.

```
a = [[1, 2, 3, 4, 5, 6]]
a_sequenzen = sequenzen_bilden(daten=a, sequenz_groesse=2)
a_sequenzen
    [[1, 2], [3, 4], [5, 6]]
a_aug = mischen_und_zusammenfuegen(daten=a)
a_aug
    [[3, 4, 5, 6, 1, 2]]
```

Listing 4.3: Beispiel für die Transfrormation durch zufälliges Mischen von Sequenzen

4.7.3 Variational Autoencoder

Die Implementierung des VAE in dieser Arbeit basiert auf der Keras VAE-Implementierung[9], der VAE-Implementierung[10] von Marco Cerliani und der Autoencoder-Implementierung[11] von Jason Brownlee.

Zu Beginn werden die Trainingsdaten *x_train* mit dem MinMax-Scaler[12] des Scikit-Learn-Frameworks auf Werte zwischen 0 und 1 skaliert. Der VAE besteht aus zwei Keras KNNs. Ein Beispiel für die Implementierung eines KerasKNN-Model ist in Listing 4.4 dargestellt. Die Keras Model Klasse gruppiert die Schichten (Layer) von KNNs in ein Objekt mit Trainings- und Inferenz-Eigenschaften.

```
from tensorflow import keras
from tensorflow.keras.layers import Dense
```

[9] https://github.com/keras-team/keras-io/blob/master/examples/generative/vae.py – Abgerufen am 01.04.2021
[10] https://github.com/cerlymarco/MEDIUM_NoteBook/tree/master/VAE_TimeSeries – Abgerufen am 01.04.2021
[11] https://machinelearningmastery.com/autoencoder-for-regression/ – Abgerufen am 01.04.2021
[12] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html – Abgerufen am 01.04.2021

```
# input layer
model_inputs = keras.Input(shape=(input_dimension,))
# hidden layers
h = Dense(hidden_dimension)(model_inputs)
h = Dense(hidden_dimension)(h)
# output layer
model_outputs = Dense(output_dimension, activation='activation_function')(h)
model = keras.Model(model_inputs, model_outputs)

model.compile(...)
```

Listing 4.4: Beispiel für die Implementierung eines Keras-KNN

Der Encoder hat ein Input-Layer mit einer Dimension, die der Dimension von *x_train* entspricht. Darauf folgen drei Dense-Layer. Ein Dense-Layer ist eine Schicht des KNN, bei der alle Neuronen mit allen Inputs und allen Outputs verbunden sind. Dann folgen zwei Dense-Layer mit der latenten Dimension, eins für den Mittelwert und eins für die Varianz. Diese beiden Dense-Layer dienen als Input für ein Sampling-Layer. Dieses Sampling-Layer implementiert den Reparameterisierungstrick, um Stichproben aus der latenten Verteilung, also dem Output der beiden vorherigen Dense-Layer, zu ziehen. Die endgültige Ausgabe des Encoders sind die Outputs der beiden Dense-Layer für die latente Verteilung und die Ausgabe des Sampling Layers.

Das Input-Layer des Decoders entspricht der Dimension der latenten Verteilung. Darauf folgen zwei Dense-Layer und ein Output-Layer, welches der Dimension der ursprünglichen *x_train* Daten entspricht.

Der Encoder und der Decoder werden in einem weiteren Keras-Modell zusammengeführt, welches so zum VAE wird. Das VAE-Modell hat einen angepassten Trainingsschritt, der bei jeder Iteration des Trainings aufgerufen wird. Im Trainingsschritt des VAE wird die vom Encoder durch das Sampling-Layer erzeugte,Probe aus der latenten Verteilung in den Decoder gegeben und so eine Rekonstruktion der ursprünglichen Eingabe *x_train* erzeugt. Dann bildet man den Fehler zwischen dem ursprünglichen *x_train* und dem rekonstruierten *x_train* durch die mittlere quadratische Abweichung. Außerdem wird ein weiterer Fehler durch die Kullback-Leibler-Divergenz zwischen den Ausgaben für den Mittelwert und der Varianz aus dem Encoder berechnet. Addiert man zuletzt die beiden Fehler, ergibt sich der Gesamtfehler des VAE. Diese Fehler werden für das Training des VAE genutzt. Der VAE wird über 100 Epochen mit einer Batchgröße von 128 trainiert. In Listing 4.5 ist die Funktion für das Training eines Keras-Modells dargestellt.

```
history = model.fit(x_train, y_train, epochs=100, batch_size=128)
```

Listing 4.5: Beispiel für das Trainieren eines Keras-Modells

Nachdem das Training des VAE abgeschlossen ist, kann der Decoder des VAE für die Erzeugung synthetischer Daten genutzt werden. Listing 4.6 zeigt, wie der Encoder und der Decoder des VAE zur Generierung synthetischer Daten genutzt werden. Der Encoder gibt die Probe z für *x_train* aus. Mit z wird *x_train* dann durch den Decoder rekonstruiert. Die Ausgabe des Decoders *result* ist eine Tabelle, die dem Format von *x_train* entspricht. Die in Listing 4.6 dargestellte Anwendung des VAE kann beliebig oft wiederholt werden, um so immer wieder neue synthetische Daten zu generieren.

Die Ergebnisse des VAE werden dann für die Leistungsflussberechnung herangezogen um *y_train* zu erzeugen.

```
z_mean, z_log_var, z = vae.encoder.predict(x_train)
result = vae.decoder.predict(z)
```

Listing 4.6: Beispiel für die Erzeugung neuer Daten mit dem trainierten VAE

4.8 Evaluation

Für die Evaluation werden nun die gesamten x- und y-Daten benötigt. Die drei Tabellen für jeweils *x_train*, *y_train*, *x_test* und *y_test* werden zusammengefügt, sodass sich insgesamt vier Tabellen ergeben, mit denen man die lineare Regression durchführt. Die lineare Regression wird mit der Scikit-Learn-Klasse Linear Regression umgesetzt. Vor Anwendung der linearen Regression werden *x_train* und *y_train* mit dem MinMax-Scaler auf Werte zwischen 0 und 1 skalliert. Listing 4.7 stellt die Implementierung der linearen Regression dar. *x_train* und *y_train* werden für das Training der linearen Regression genutzt. Die lineare Regression lernt also, die Ergebnisse *y_train* der Leistungsflussberechnung bei gegebenen *x_train* anzunähern. Die Testdaten *x_test*, die dem linearen Model noch unbekannt sind, werden dem Modell übergeben, das seinerseits dann die entsprechenden Annäherungen für die Ergebnisse ausgibt. Diese Annäherungen werden dann mit den eigentlichen *y_test*, die aus der Leistungsflussberechnung stammen, verglichen, und es wird ihre mittlere absolute Abweichung und R^2 berechnet.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x_train, y_train)
prediction = model.predict(x_test)
```

Listing 4.7: Beispiel für die Anwendung der linearen Regression

5 Evaluation

In diesem Kapitel soll es darum gehen, die verschiedenen Umsetzungen der DA anhand der Forschungsfragen aus Abschnitt 1.1 zu evaluieren. Es wird auf die Form der Auswertung, die verwendeten Metriken und die Ergebnisse dieser Arbeit eingegangen. Die umgesetzten Ansätze für die DA werden verglichen und in ihrer Anwendbarkeit diskutiert.

5.1 Form der Auswertung

Die Auswertung erfolgt nach dem Prinzip *train on synthetic test on real* (TSTR) [49]. Das bedeutet, dass für das Training der linearen Regression nur die Daten genutzt werden, die durch die DA generiert wurden, während die eigentlichen Daten, die für die DA genutzt wurden, explizit ausgeschlossen bleiben. Für das Testen der linearen Regression werden dann ausschließlich die Testdaten genutzt, die vor der Anwendung der DA ausgeschlossen wurden. So kann die Qualität der durch die DA erzeugten Daten überprüft werden.

5.2 Auswahl der Metriken

Die hier ausgewählten Metriken bewerten die Ergebnisse der linearen Regression. Sie vergleichen die echten Ergebnisse der Leistungsflussberechnung mit denen, von der linearen Regression angenäherten. Dazu wird hier die Mittlere absolute Abweichung (MAE) ausgewählt. MAE ist eine weit verbreitete Metrik zur Evaluation von Modellen [69]. Ihre Berechnung lautet wie folgt:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|e_i|. \tag{5.1}$$

Dabei steht ein niedriger MAE-Wert für ein besseres Ergebnis als ein hoher. Zusätzlich wurde R^2 ausgewählt, ebenfalls eine weit verbreitete und einfach zu interpretierende Metrik für lineare Modelle [70]. Sie zeigt den prozentualen Anteil der durch die lineare Regression erklärten Quadratsumme an der zu erklärenden totalen Quadratsumme. Falls die Daten

jedoch eine große Streuung aufweisen, kann R^2 ein schlechtes Ergebnisse anzeigen, obwohl die lineare Regression die Daten bestmöglich erklärt. R^2 wird wie folgt berechnet:

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \overline{y})^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}. \tag{5.2}$$

Der bestmögliche Wert für R^2 ist 1. Ein Wert von 0 bedeutet, dass das lineare Modell eine Regressionsline bildet, die dem Durchschnitt der Werte der Zieldaten entspricht. Ein negativer Wert drückt aus, dass die Vorhersagen des linearen Modells schlechter sind als die Durchschnitts-Regressionslinie.

5.3 Ergebnisse

Zur Generierung der Ergebnisse wird die lineare Regression mit verschiedenen Trainingsdaten trainiert, auf die Testdaten angewendet und mit den vorgestellten Metriken bewertet. Dabei gilt das in Kapitel 5 vorgestellte Prinzip TSTR.

Tabelle 5.1 und Tabelle 5.2 zeigen die Ergebnisse der durchgeführten Experimente. Es wurden in den Experimenten verschiedene DA-Ansätze mit verschiedenen Parametern untersucht. Ein wichtiges Parameter für die Untersuchungen war die Größe des erzeugten Trainingsdatensatzes. Für die Experimente wurde die Variable *times_to_augment* genutzt, die angibt, um wievielmal die Trainingsdaten durch einen DA-Ansatz vergrößert wurden. Tabelle 5.1 bildet die Ergebnisse ab, die mit einem Wert von $times\_to\_augment = 5$ erzeugt wurden, also einem synthetischen Trainingsdatensatz, der fünfmal so groß ist wie der ursprüngliche Trainingsdatensatz. Tabelle 5.2 zeigt die Ergebnisse bei $times\_to\_augment = 10$. Beide Tabellen weisen die Ergebnisse der linearen Regression anhand der beiden Metriken *MSE* und R^2 aus. Die Ergebnisse sind nach den verwendeten DA-Ansätze unterteilt, mit denen die Trainingsdaten generiert wurden. Zum Vergleich sind in beiden Tabellen die Ergebnisse der linearen Regression angegeben, bei der die realen nicht synthetischen Trainingsdaten für das Training genutzt wurden. Da für diese Trainingsdaten keine DA durchgeführt wurde, ist in den Tabellen für den Data-Augmentation-Ansatz *keine DA* angegeben. Außerdem fanden Veränderungen weiterer Parameter der Ansätze statt. Für den DA-Ansatz, bei dem die Trainingsdaten prozentual verändert werden, wurden verschiedene Prozentsätze untersucht. Für den DA-Ansatz, bei dem Sequenzen vertauscht werden, wurden verschiedene Sequenzgrößen untersucht. Für den VAE wurde in den Experimenten nur das Parameter für die Größe des erzeugten Trainingsdatensatzes verändert. Für den VAE existieren auch keine weiteren Parameter, die es zu untersuchen gäbe. Denn dies ist einer der in Abschnitt 4.1 genannten Gründe für die Entscheidung zur Umsetzung des VAE.

Der VAE hat einen hohen Automatisierungsgrad und macht eine manuelle Einstellung der Parameter unnötig, was einen Vorteil gegenüber anderen Methoden darstellt. Aufgrund der probalistischen Natur der Experimente kann es zu Schwankungen der Ergebnisse kommen, wenn diese mit den gleichen Parametern mehrfach durchgeführt werden. Aus diesem Grund wurde jedes der hier präsentierten Experimente fünfmal durchgeführt. Die in Tabelle 5.1 und Tabelle 5.2 vorgestellten Ergebniswerte sind jeweils die Durchschnittswerte der fünf Ergebnisse aus dem fünfmaligen Ausführen der jeweiligen Experimente.

Data Augmentation Ansatz	MSE	R^2
Keine DA	0.162	0.902
Transformation prozentuale Änderung (5%)	0.161	0.897
Transformation prozentuale Änderung (10%)	0.160	0.898
Transformation vertauschen von Sequenzen (Sequenzgröße: 50)	0.182	0.865
Transformation vertauschen von Sequenzen (Sequenzgröße: 100)	0.189	0.851
VAE	0.904	-4.554

Tabelle 5.1: Ergebnisse der linearen Regression, bei times_to_augment=5

Data Augmentation Ansatz	MAE	R^2
Keine DA	0.162	0.902
Transformation prozentuale Änderung (5%)	0.160	0.878
Transformation prozentuale Änderung (10%)	0.159	0.881
Transformation vertauschen von Sequenzen (Sequenzgröße: 50)	0.190	0.831
Transformation vertauschen von Sequenzen (Sequenzgröße: 100)	0.196	0.824
VAE	0.731	-2.701

Tabelle 5.2: Ergebnisse der linearen Regression, bei times_to_augment=10

Die Ergebnissen aus Tabelle 5.1 und Tabelle 5.2 machen deutlich, dass der DA-Ansatz mit den Transformationen durch prozentuale Änderungen zu einer Verbesserung der linearen Regression bezüglich des MAE geführt hat. Bei $times\_to\_aug = 10$ und und einem Höchstprozentsatz von 10 hat der MAE-Wert eine positive Veränderung von 0.162 auf bis zu 0.159 erfahren (siehe Tabelle 5.2). Dies entspricht einer Verbesserung von etwa zwei Prozent. Für die Versuche dieses Ansatzes, bei denen anderen Parameter verwendet wurden, zeigt sich ebenfalls eine Verbesserung der linearen Regression, allerdings liegt die Verbesserung dort nur bei etwa einem Prozent und weniger. Der R^2 verschlechtert sich bei allen Versuchen innerhalb dieses Ansatzes. Es lässt sich erkennen, dass bei der Nutzung eines größeren Trainingsdatensatzes ($times\_to\_aug=10$) der R^2-Wert schlechter wird. Er sinkt von ursprünglich 0.902 auf bis zu 0.878 ab. Bei $times\_to\_aug = 10$ sinkt der Wert beispielsweise von 0.897 auf 0.878 ab. Außerdem ist festzuhalten, dass die Nutzung von

einem Höchstprozentsatz von 10 zu besseren Ergebnissen führt als die Nutzung von einem Höchstprozentsatz von 5. Der Wert für den MAE und R^2 ist bei einem Wert von 10 Prozent besser. Beispielsweise sinkt der MAE-Wert von 0.161 auf 0.160, und der R^2-Wert steigt von 0.897 auf 0.898. Aus diesen Experimenten ergibt sich demnach, dass die Verwendung der DA anhand von Transformationen durch prozentuale Änderungen zu einer Verbesserung der linearen Regression führt, wobei ein größerer synthetischer Trainingsdatensatz bessere Ergebnissen erbringt als ein kleinerer Trainingsdatensatz. Außerdem führt die Veränderung von bis zu 10 Prozent zu einem besseren Resultat als die mit bis zu 5 Prozent. Aus den Ergebnissen mit der Verwendung der DA durch Transformationen, bei der die Trainingsdaten in Sequenzen vertauscht wurden, erkennt man, dass dies zu keiner Verbesserung der linearen Regression in Bezug auf beide Metriken führt. Alle Experimenten zu diesem Ansatz zeigen schlechte Ergebnisse für die lineare Regression. Die Ergebnisse haben sich für den MAE-Wert von 0.162 auf bis zu 0.190 verschlechtert, für den R^2-Wert von 0.902 auf 0.831. Der beste MAE-Wert von 0.182 und der beste R^2-Wert von 0.865 ist bei einer Sequenzgröße von 50 und *times_to_aug=5* entstanden. Auch im Vergleich zu dem vorherigen Ansatz verhält sich dieser Ansatz in den Experimenten schlechter. Die Experimente für die Verwendung des VAE haben Folgendes gezeigt: Der VAE führt zu deutlich schlechten Ergebnissen der linearen Regression. Es kommt zu einer starken Verschlechterung der linearen Regression in Bezug auf beide Metriken. Tabelle 5.2 zeigt, dass der MAE-Wert von 0.162 um etwa 560 Prozent auf 0.904 ansteigt. Der r^2-Wert wird sogar negativ. Bei *times_to_aug=10* scheint der VAE bessere Werte zu liefern als bei *times_to_aug=5*. Der MAE-Wert beträgt 0.731 und der R^2-Wert -2.701. Diese Werte sind jedoch immer noch deutlich schlechter als die Vergleichswerte, die aus der linearen Regression ohne DA stammen. Auch im Vergleich zu den anderen DA-Ansätzen zeigt sich, dass der VAE im Rahmen der in dieser Arbeit unternommenen Untersuchung nicht für die DA geeignet ist. Abbildung 5.1 zeigt einen Ausschnitt der Trainingsdaten und der entsprechenden Rekonstruktion durch den VAE. Es ist deutlich zu erkennen, dass der VAE die originalen Daten nicht ausreichend rekonstruieren kann. Die rekonstruierten Daten scheinen sich lediglich, mit leichten Schwankungen, um einen Wert von 2,4 zu verteilen, unabhängig davon wie die Werte der originalen Daten sind. Dies unterstützt die Erkenntnis aus den Ergebnissen der linearen Regression: Der VAE kann die Trainingsdaten für die DA nicht ausreichend rekonstruieren und deshalb keine geeigneten Trainingsdaten für das Training der linearen Regression erzeugen. Abbildung 5.2 zeigt die Verteilung der Trainingsdaten als Boxplots. Bei Betrachtung von Abbildung 5.2 fällt auf, dass die Verteilung der originalen Trainingsdaten der der prozentualen Transformation annähernd entspricht. Dies war zu erwarten, da die Daten lediglich um ein paar Prozent verschoben wurden. Erneut ist eine schlechte Rekonstruktion des VAE erkennbar. Bei den Verteilungen ergeben sich deutliche Unterschiede, wobei die des VAE im Vergleich

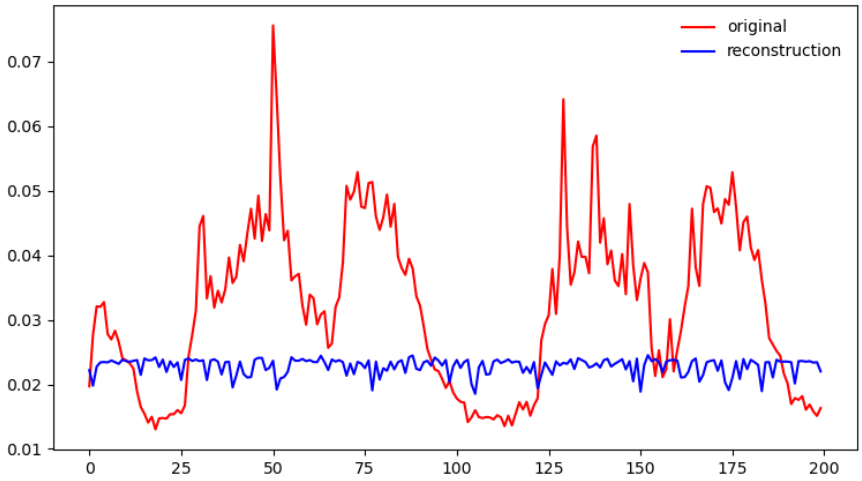


Abbildung 5.1: Ausschnitt der Trainingsdaten und der Rekonstruktion dieser Daten durch den VAE

zu den originalen Daten eine geringere Streuung aufweist, was sich bereits in Abbildung 5.1 gezeigt hat.

Obwohl die Experimente zeigen, dass die DA mit dem VAE zu deutlich schlechteren Ergebnissen führt als die mit den transformationsbasierten Ansätzen, lässt sich erkennen, dass das Training des VAE erfolgreich funktioniert. In Abbildung 5.3 ist der Verlauf des Fehlers (Loss) des VAE über das Training dargestellt. Der Fehler des VAE setzt sich, wie bereits in Unterabschnitt 2.2.2 und Unterabschnitt 4.7.3 erläutert, aus dem Rekonstruktionsfehler und dem Fehler der KL-Divergenz zusammen. Es ist zu erkennen, dass diese Fehler erfolgreich minimiert werden.

Da das Training des VAE zwar funktioniert, aber dieser Trainingsdaten generiert, die zu schlechten Ergebnissen der linearen Regression führen, ist zu vermuten, dass die Datengrundlage für die erfolgreiche Anwendung des VAE zu klein ist. Der VAE schafft es also nicht, aus der hier genutzten Datengrundlage eine ausreichende Rekonstruktion der Trainingsdaten zu erlernen.

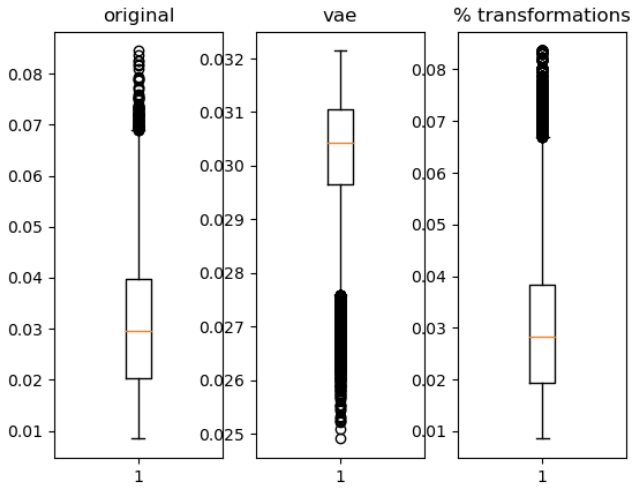


Abbildung 5.2: Boxplots der Gleichen Spalte für die originalen Trainingsdaten, die aus dem VAE und die aus der DA durch prozentuale Transformationen.

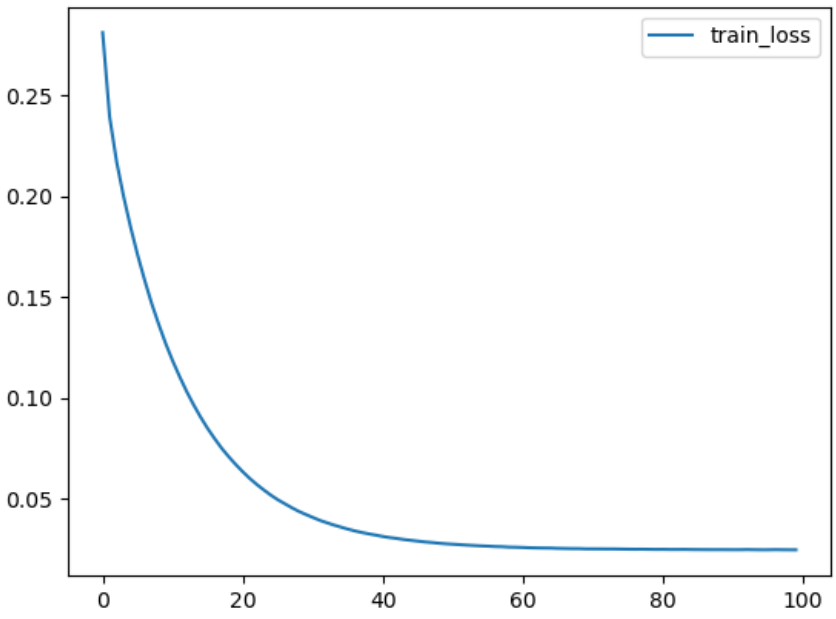


Abbildung 5.3: Loss des VAE

6 Fazit

Ziel dieser Arbeit war es zu untersuchen, ob DA von Trainingsdaten ML-Verfahren in Energiesystemen verbessern kann. Dazu wurden zunächst verschiedene Ansätze für die DA und verwandte Arbeiten vorgestellt und dann zwei transformationsbasierte Ansätze und ein ML-Ansatz für die Umsetzung der DA ausgewählt und implementiert. Diese Ansätze wurden genutzt, um die Trainingsdaten für eine lineare Regression zu vergrößern. Die lineare Regression hat in dieser Arbeit die Leistungsflussberechnung erlernt. Experimente erbrachten verschiedene Ergebnisse für die lineare Regression, anhand derer die verschiedenen DA-Ansätze evaluiert und verglichen wurden. In Hinblick auf die Forschungsfragen ergeben sich folgende Erkenntnisse und Handlungsempfehlungen:

RQ1: *Ist es möglich, durch DA das Training einer ML-Methode aus dem Energiebereich zu verbessern?*

Die Ergebnisse für die DA mit Transformationen durch prozentuale Änderungen haben gezeigt, dass die lineare Regression für die Leistungsflussberechnung bessere Ergebnisse annähert als ohne die Nutzung von DA. Damit lässt sich sagen, dass es möglich ist, das Training einer ML-Methode aus dem Energiebereich durch DA zu verbessern.

RQ2: *Welche DA-Ansätze sind besonders geeignet, Trainingsdaten einer ML-Methode aus dem Energiebereich zu erweitern?*

Die Ergebnisse dieser Arbeit zeigen, dass sich für die Erweiterung von Trainingsdaten einer ML-Methode aus dem Energiebereich transformationsbasierte Ansätze in diesem Fall besser als ML-Ansätze eignen. Dies lässt den Schluss zu, dass sich der Mehraufwand für die Umsetzung eines komplexen ML-Ansatzes oft nicht lohnt und ein simpler transformationsbasierter Ansatz bereits ausreicht. Dabei ist jedoch zu beachten, dass die Parameter für die Umsetzung eines transformationsbasierten Ansatzes manuell festgelegt werden müssen.

Aufgrund der Ergebnisse dieser Arbeit ist für die DA für ML-Verfahren aus dem Energiebereich ein transformationsbasierter Ansatz zu empfehlen, der die Trainingsdaten durch

prozentuale Änderungen transformiert, wie sie in Unterabschnitt 4.7.1 vorgestellt wurden. Es gilt aber auch in Betrachtung des zugrundeliegenden Falls abzuwägen, ob der Aufwand für die Umsetzung und Nutzung von DA durch eine Verbesserung des Endergebnisses von etwa zwei Prozent gerechtfertigt ist.

Dass der VAE in der vorliegenden Betrachtung zu schlechten Ergebnissen führte, verwandte Forschungsarbeiten diesen jedoch erfolgreich angewendet haben, legt die Vermutung nahe, dass die Datengrundlage für einen VAE noch zu gering ist. Diese Datengrundlage zu betrachten wäre Gegenstand einer fortführenden Analyse. Eine weitere Möglichkeit wäre die Untersuchung von Variationen in der Implementierung für den VAE, beispielsweise durch das Nutzen von Convolutional Neural Networks. Ebenfalls könnte ein anderer ML-Ansatz für die DA umgesetzt werden. Die in dieser Arbeit vorgestellten GANs könnten daraufhin untersucht werden, ob sie sich für die DA in diesem Fall eignen. Die zum VAE erzeugten Ergebnisse dienen dann für einen Vergleich.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Lewicki, P.: Erneuerbare Energien in Zahlen. Publisher: Umweltbundesamt (2013). https://www.umweltbundesamt.de/themen/klima-energie/erneuerbare-energien/erneuerbare-energien-in-zahlen Accessed 2021-03-21

[2] Dietrich, D.: Demand Side Management: Demand Response, Intelligent Energy Systems, and Smart Loads. Accessed 2020-11-02

[3] Energie, B.f.W.u.: Intelligente Netze. https://www.bmwi.de/Redaktion/DE/Artikel/Energie/intelligente-netze.html Accessed 2021-03-21

[4] Hosseini Imani, M., Zalzar, S., Mosavi, A., Shamshirband, S.: Strategic Behavior of Retailers for Risk Reduction and Profit Increment via Distributed Generators and Demand Response Programs. Energies **11**(6), 1602 (2018). doi:10.3390/en11061602. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute. Accessed 2020-11-02

[5] Torabi, M., Hashemi, S., Saybani, M.R., Shamshirband, S., Mosavi, A.: A Hybrid clustering and classification technique for forecasting short-term energy consumption. Environmental Progress & Sustainable Energy **38**(1), 66–76 (2019). doi:10.1002/ep.12934. Accessed 2020-11-02

[6] Dineva, A., Mosavi, A., Faizollahzadeh Ardabili, S., Vajda, I., Shamshirband, S., Rabczuk, T., Chau, K.-W.: Review of Soft Computing Models in Design and Control of Rotating Electrical Machines. Energies **12**(6), 1049 (2019). doi:10.3390/en12061049. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute. Accessed 2020-11-02

[7] Was ist Machine Learning? https://www.bigdata-insider.de/was-ist-machine-learning-a-592092/ Accessed 2021-04-05

[8] Mosavi, A., Salimi, M., ardabili, S., Rabczuk, T., Band, S., Varkonyi-Koczy, A.: State of the Art of Machine Learning Models in Energy Systems, a Systematic Review. Energies **12** (2019). doi:10.3390/en12071301

[9] Cheng, L., Yu, T., Zhang, X., Yin, L.: Machine Learning for Energy and Electric Power Systems: State of the Art and Prospects. Dianli Xitong Zidonghua/Automation of Electric Power Systems **43**, 15–31 (2019). doi:10.7500/AEPS20180814007

[10] admin: Datensätze (2019). https://simbench.de/de/datensaetze/ Accessed 2020-11-12

[11] Balduin, S., Westermann, T., Puiutta, E.: Evaluating different machine learning techniques as surrogate for low voltage grids. Energy Informatics **3**(1), 24 (2020). doi:10.1186/s42162-020-00127-3. Accessed 2020-11-16

[12] Dyk, D.A.v., Meng, X.-L.: The Art of Data Augmentation. Journal of Computational and Graphical Statistics **10**(1), 1–50 (2001). doi:10.1198/10618600152418584. Publisher: Taylor & Francis _eprint: https://doi.org/10.1198/10618600152418584. Accessed 2020-11-13

[13] Goyal, S.: MachineX: Image Data Augmentation Using Keras (2019). https://towardsdatascience.com/machinex-image-data-augmentation-using-keras-b459ef87cd22 Accessed 2021-03-21

[14] Karras, T., Laine, S., Aila, T.: A Style-Based Generator Architecture for Generative Adversarial Networks. arXiv:1812.04948 [cs, stat] (2019). arXiv: 1812.04948. Accessed 2021-03-21

[15] Perez, L., Wang, J.: The Effectiveness of Data Augmentation in Image Classification using Deep Learning. arXiv:1712.04621 [cs] (2017). arXiv: 1712.04621. Accessed 2020-11-20

[16] Gong, X., Tang, B., Zhu, R., Liao, W., Song, L.: Data Augmentation for Electricity Theft Detection Using Conditional Variational Auto-Encoder. Energies **13**(17), 4291 (2020). doi:10.3390/en13174291. Number: 17 Publisher: Multidisciplinary Digital Publishing Institute. Accessed 2020-10-26

[17] Zhang, C., Kuppannagari, S.R., Kannan, R., Prasanna, V.K.: Generative Adversarial Network for Synthetic Time Series Data Generation in Smart Grids. In: 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), pp. 1–6. IEEE, Aalborg (2018). doi:10.1109/SmartGridComm.2018.8587464. https://ieeexplore.ieee.org/document/8587464/ Accessed 2020-11-03

[18] Géron, A.: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. Ö'Reilly Media, Inc.", ??? (2019). Google-Books-ID: HHetDwAAQBAJ

[19] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, Global Edition, 3. edition edn. Addison Wesley, Boston Columbus Indianapolis New York San Francisco Upper Saddle River Amsterdam, Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo (2016)

[20] Neuronale Netze – eLearning - Methoden der Psychologie - TU Dresden. https://methpsy.elearning.psych.tu-dresden.de/mediawiki/index.php/Neuronale_Netze Accessed 2021-03-25

[21] Kruse, R.: Neuronale Netze: Eine Einführung in die Neuroinformatik. Springer, ??? (2013). Google-Books-ID: TR6WBwAAQBAJ

[22] Gong, S.: How does a Neural Network work intuitively in code? (2019). https://medium.com/@gongster/how-does-a-neural-network-work-intuitively-in-code-f51f7b2c1e3f Accessed 2021-03-25

[23] Bottou, L.: Stochastic Gradient Descent Tricks. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade vol. 7700, pp. 421–436. Springer, Berlin, Heidelberg (2012). Series Title: Lecture Notes in Computer Science

[24] Rencher, A.C., Schaalje, G.B.: Linear Models in Statistics, 2nd ed edn. Wiley-Interscience, Hoboken, N.J (2008). OCLC: ocn144331522

[25] Andres, J.: Das Allgemeine Lineare Modell. doi:10.25521/HQM15. Publisher: Universitätsbibliothek Mannheim. Accessed 2021-03-31

[26] sklearn.linear_model.LinearRegression — scikit-learn 0.24.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html Accessed 2021-03-31

[27] Müller, A.C., Guido, S.: Introduction to Machine Learning with Python: A Guide for Data Scientists. Ö'Reilly Media, Inc.", ??? (2016)

[28] A. Y., N., Jordan, M.I.: On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. Advances in neural information processing systems, 841–848 (2002)

[29] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates, Inc., ??? (2014). http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf Accessed 2020-10-21

[30] O'Shea, K., Nash, R.: An Introduction to Convolutional Neural Networks. arXiv:1511.08458 [cs] (2015). arXiv: 1511.08458. Accessed 2021-03-22

[31] Brownlee, J.: A Gentle Introduction to Generative Adversarial Networks (GANs) (2019). https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/ Accessed 2021-03-25

[32] Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. arXiv:1312.6114 [cs, stat] (2014). arXiv: 1312.6114. Accessed 2020-10-25

[33] Hershey, J.R., Olsen, P.A.: Approximating the Kullback Leibler Divergence Between Gaussian Mixture Models. In: 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07, vol. 4, pp. 317–320 (2007). doi:10.1109/ICASSP.2007.366913. ISSN: 2379-190X

[34] Rocca, J.: Understanding Variational Autoencoders (VAEs) (2021). https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 Accessed 2021-03-30

[35] Shafkat, I.: Intuitively Understanding Variational Autoencoders (2018). https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf Accessed 2021-03-29

[36] Polster, S., Renner, H.: Berechnung elektrischer Energienetze, 34

[37] Schwab, A.J.: Elektroenergiesysteme: Erzeugung, Übertragung und Verteilung Elektrischer Energie. Springer, ??? (2017). Google-Books-ID: Gq80DwAAQBAJ

[38] Ypma, T.J.: Historical Development of the Newton–Raphson Method. SIAM Review **37**(4), 531–551 (1995). doi:10.1137/1037125. Accessed 2021-04-01

[39] Akram, S., ul Ann, Q.: Newton Raphson Method. International Journal of Scientific & Engineering Research **Volume 6** (2015). Accessed 2021-04-01

[40] Lai, C.S., Mo, Z., Wang, T., Yuan, H., Ng, W.W.Y., Lai, L.L.: Load Forecasting based on Deep Neural Network and Historical Data Augmentation, 11

[41] Ruan, G., Zhong, H., Zhang, G., He, Y., Wang, X., Pu, T.: Review of Learning-Assisted Power System Optimization. arXiv:2007.00210 [cs, eess, math] (2020). arXiv: 2007.00210. Accessed 2020-10-07

[42] Pesteie, M., Abolmaesumi, P., Rohling, R.N.: Adaptive Augmentation of Medical Data Using Independently Conditional Variational Auto-Encoders. IEEE Transactions on Medical Imaging **38**(12), 2807–2820 (2019). doi:10.1109/TMI.2019.2914656. Conference Name: IEEE Transactions on Medical Imaging

[43] Ratner, A.J., Ehrenberg, H., Hussain, Z., Dunnmon, J., Ré, C.: Learning to Compose Domain-Specific Transformations for Data Augmentation, 11

[44] DeVries, T., Taylor, G.W.: DATASET AUGMENTATION IN FEATURE SPACE, 12 (2017)

[45] Mao, X., Li, Q., Xie, H., Lau, R.Y.K., Wang, Z., Smolley, S.P.: Least Squares Generative Adversarial Networks, 9

[46] Yu, S., Yang, J., Liu, D., Li, R., Zhang, Y., Zhao, S.: Hierarchical Data Augmentation and the Application in Text Classification. IEEE Access **7**, 185476–185485 (2019). doi:10.1109/ACCESS.2019.2960263. Conference Name: IEEE Access

[47] Esteban, C., Hyland, S.L., Rätsch, G.: Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs. arXiv:1706.02633 [cs, stat] (2017). arXiv: 1706.02633. Accessed 2020-10-12

[48] Zheng, X., Wang, B., Kalathil, D., Xie, L.: Creation of Synthetic Networked PMU Data: A Generative Adversarial Network Approach. arXiv:1908.08180 [eess] (2020). arXiv: 1908.08180. Accessed 2020-10-07

[49] Fekri, M.N., Ghosh, A.M., Grolinger, K.: Generating Energy Data for Machine Learning with Recurrent Generative Adversarial Networks. Energies **13**(1), 130 (2020). doi:10.3390/en13010130. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute. Accessed 2020-10-12

[50] Sadeghi, M., Leglaive, S., Alameda-PIneda, X., Girin, L., Horaud, R.: Audio-visual Speech Enhancement Using Conditional Variational Auto-Encoders. arXiv:1908.02590 [cs, eess] (2020). arXiv: 1908.02590. Accessed 2020-10-26

[51] Yijiang, W., Chen, L., Ganjun, W., Xiaosheng, P., Taiwei, L., Yunzheng, Z.: Partial Discharge Data Augmentation of High Voltage Cables based on the Variable

Noise Superposition and Generative Adversarial Network. In: 2018 International Conference on Power System Technology (POWERCON), pp. 3855–3859 (2018). doi:10.1109/POWERCON.2018.8602223

[52] Yu, L., Zhang, W., Wang, J., Yu, Y.: SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient, 7

[53] Mogren, O.: C-RNN-GAN: Continuous recurrent neural networks with adversarial training. arXiv:1611.09904 [cs] (2016). arXiv: 1611.09904. Accessed 2020-10-23

[54] Hou, X., Shen, L., Sun, K., Qiu, G.: Deep Feature Consistent Variational Autoencoder. In: 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 1133–1141 (2017). doi:10.1109/WACV.2017.131

[55] Vahdat, A., Kautz, J.: NVAE: A Deep Hierarchical Variational Autoencoder. arXiv:2007.03898 [cs, stat] (2021). arXiv: 2007.03898. Accessed 2021-03-23

[56] Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., Carin, L.: Variational Autoencoder for Deep Learning of Images, Labels and Captions, 9

[57] Imran, A., Terzopoulos, D.: Multi-adversarial Variational Autoencoder Networks. In: 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), pp. 777–782 (2019). doi:10.1109/ICMLA.2019.00137

[58] Hsu, W., Zhang, Y., Glass, J.: Unsupervised domain adaptation for robust speech recognition via variational autoencoder-based data augmentation. In: 2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), pp. 16–23 (2017). doi:10.1109/ASRU.2017.8268911

[59] Xin, R., Zhang, J., Shao, Y.: Complex network classification with convolutional neural network. Tsinghua Science and Technology **25**(4), 447–457 (2020). doi:10.26599/TST.2019.9010055. Conference Name: Tsinghua Science and Technology

[60] Pan, Z., Wang, J., Liao, W., Chen, H., Yuan, D., Zhu, W., Fang, X., Zhu, Z.: Data-Driven EV Load Profiles Generation Using a Variational Auto-Encoder. Energies **12**(5), 849 (2019). doi:10.3390/en12050849. Number: 5 Publisher: Multidisciplinary Digital Publishing Institute. Accessed 2021-03-23

[61] Chakraborty, I., Nandanoori, S.P., Kundu, S., Kalsi, K.: Stochastic Virtual Battery Modeling of Uncertain Electrical Loads Using Variational Autoenco-

der*. In: 2020 American Control Conference (ACC), pp. 1305–1310 (2020). doi:10.23919/ACC45564.2020.9147609. ISSN: 2378-5861

[62] Zhou, Y., Guo, J., Guo, Q., Sun, H., Lu, Y., Xu, X.: Rule Extraction-based Data Augmentation Method for Transient Instability Identification of Power Systems Using Machine Learning. In: 2019 IEEE Sustainable Power and Energy Conference (iSPEC), pp. 1687–1692 (2019). doi:10.1109/iSPEC48194.2019.8975153

[63] Cendrowska, J.: PRISM: An algorithm for inducing modular rules. International Journal of Man-Machine Studies **27**(4), 349–370 (1987). doi:10.1016/S0020-7373(87)80003-2. Accessed 2020-10-27

[64] Ngoko, B.O., Sugihara, H., Funaki, T.: Synthetic generation of high temporal resolution solar radiation data using Markov models. Solar Energy **103**, 160–170 (2014). doi:10.1016/j.solener.2014.02.026. Accessed 2020-10-23

[65] Chen, J., Rabiti, C.: Synthetic wind speed scenarios generation for probabilistic analysis of hybrid energy systems. Energy **120**, 507–517 (2017). doi:10.1016/j.energy.2016.11.103. Accessed 2020-11-13

[66] Researcher, M.S. PhD: GANs vs. Autoencoders: Comparison of Deep Generative Models (2020). https://towardsdatascience.com/gans-vs-autoencoders-comparison-of-deep-generative-models-985cf15936ea Accessed 2021-03-25

[67] Why is Python Used for Machine Learning? | Hacker Noon. https://hackernoon.com/why-python-used-for-machine-learning-u13f922ug Accessed 2021-03-25

[68] Buchanan, B.G.: A (Very) Brief History of Artificial Intelligence. AI Magazine **26**(4), 53–53 (2005). doi:10.1609/aimag.v26i4.1848. Number: 4. Accessed 2021-03-24

[69] Chai, T., Draxler, R.R.: Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. Geoscientific Model Development **7**(3), 1247–1250 (2014). doi:10.5194/gmd-7-1247-2014. Publisher: Copernicus GmbH. Accessed 2021-04-04

[70] Reisinger, H.: The impact of research designs on R2 in linear regression models: an exploratory meta-analysis, 12

Erklärung

Ich versichere, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichungen, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Oldenburg, den 18. Oktober 2021 Maximilian Böhm