Hochschule für Technik und Wirtschaft Berlin

Ausarbeitung

KI in der Robotik

RNN & LSTM für Zeitreihenprognose

von

Matthias Titze

s0563413

25.09.2020

Hochschule für Technik und Wirtschaft Berlin

University of Applied Sciences

Betreuer: Patrick Baumann, M.Sc

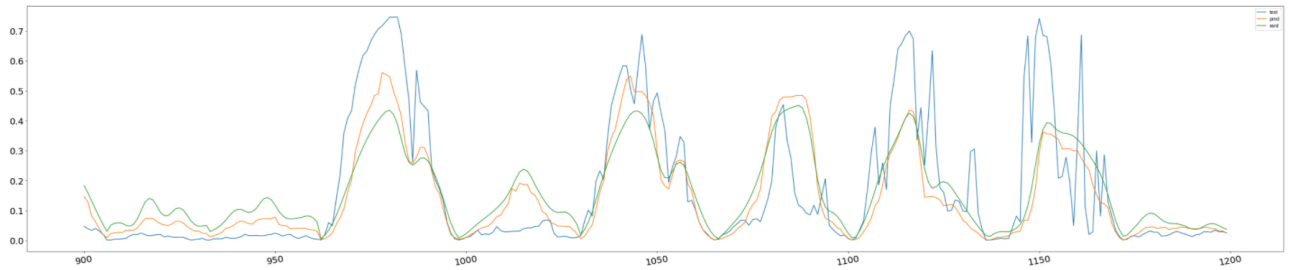
Kurzfassung

Für die Optimierung der Nutzung von lokal erzeugter Energie aus regenerativen Quellen werden heute vermehrt IoT-Netzwerke und Maschine-Learning-Algorithmen eingesetzt. Dabei spielt die Prognose der Stromerzeugung und des zu erwartenden Verbrauchs eine entscheidende Rolle. Diese Aufgabenstellungen lassen sich aus der Sicht des Data-Scientists als Zeitreihenprobleme begreifen. Die Arbeit prüft für derartige Vorhersagen den Einsatz von rekurrenten Netzwerken. Das einfache RNN und das darauf aufsetzende LSTM werden hier in Bezug auf Performanz verglichen. Daneben erfolgt eine Gegenüberstellung mit dem Random-Forest-Regressor-Algorithmus.

1. Einleitung

1.1 Motivation

Modelle auf der Basis von Machine-Learning-Algorithmen können eine zentrale Rolle bei der strategischen Steuerung von Energiekreisläufen spielen. Sie werden mit dem Ziel eingesetzt, die Energieproduktion zu prognostizieren, um insbesondere die Nutzung der oft volatilen, regenerativen Quellen zu optimieren. Dabei steht die zeitliche Entzerrung des Verbrauchs, sowie die Speicherung und Umwandlung von elektrischer Energie im Vordergrund.



[**Abbildung 1:** Prognosekurven für die Leistung einer PV-Anlage]

Als Beispiel für einen Stromproduzenten mit erheblichen Leistungsschwankungen sei hier Photovoltaik genannt. Es liegt im Interesse des Investors in eine derartige Anlage, den dort produzierten Strom auch direkt zu nutzen. Daraus ergibt sich ein ökonomischer Vorteil, wenn die Einnahmen für die Einspeisung ins Netz unter den Kosten für den Bezug von Strom liegen. Die Leistung von PV-Paneelen ist jedoch durch den Tageszyklus und das Wetter bedingt, so dass die Abnahme nach Möglichkeit strategisch zu planen ist. Ein Beispiel für einen intelligent gesteuerten Verbrauch ist die Manipulation von Ladevorgängen bei elektrischen Fahrzeugen, welche sich aus dem Teilnetz bedienen. Wird der Ladeprozess entsprechend der variablen Stromproduktion der PV-Anlage angepasst, dann kann vorwiegend diese lokal erzeugte, regenerative Energie gespeichert werden. Der Strombezug aus dem Netz wird dadurch reduziert und die Basislast beim Netzbetreiber gesenkt.

Eine derartige Steuerung ist jedoch mit dem Kunden zu verhandeln, da dieser eventuell längere Ladezeiten in Kauf nehmen muss. Daher ist es entscheidend Prognosen, für die etwaige Ladeleistung abzugeben. Dafür müssen wiederum die Produktion der Photovoltaikanlagen und die sonstigen Verbrauchsströme vorhergesagt werden. Die dafür eingesetzten ML-Modelle sind Schwerpunkt dieser Arbeit.

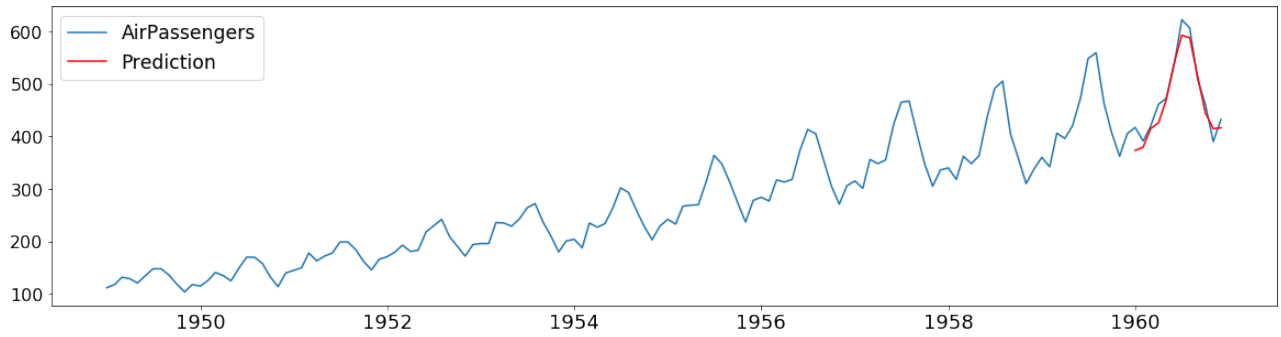
1.2 Problemstellung

Die Prognose von Energieströmen fällt in die Kategorie der Zeitreihenprobleme, d.h. wir möchten den Verlauf einer Größe - hier die elektrische Leistung in Kilowatt - über die Zeit vorhersagen. Ähnliche Fragestellungen sind auch in anderen Themengebieten zu finden, wie zum Beispiel bei der Prognose von Börsenkursen oder der Passagierzahlen an Flughäfen. Als Eingabedaten für entsprechende ML-Modelle kommen Zeitreihen zum Tragen, also Datensätze, die vorhandene Werte mit Zeitstempeln paaren. Jedoch lassen sich zwei Obergruppen für die Lösungsstrategien ausmachen:

A. Zeitgleiche Daten: Bei diese Ansatz wird über einen Feature-Vektor X(t) von Werten zum Zeitpunkt t aus einer Domäne auf eine zu prognostizierenden Variable y(t) geschlossen. Dabei ist das Feature y nicht in X enthalten, der Zeitstempel aber identisch. Dies ist die eher klassische Strategie bei Regressionsmodellen

oder FNNs (Feedforward Neural Networks). So könnte zum Beispiel über die Wettervorhersage (Temperatur, Sonnenstand, Wolkenbedeckung) die Solarstromproduktion vorhergesagt werden. Die Problemstellung für eine Zeitreihenprognose liegt hier in der Art der Eingabedaten: Soll in die Zukunft geschaut werden, dann sind auch die genutzten Daten oft Prognosen und somit fehlerbehaftet. Schlägt das Wetter wider Erwarten um, so wird die PV-Anlage ebenso andere Leistungswerte zeigen.

B. Historische Daten: Der Gegenentwurf ist die Nutzung von historischen Daten als Eingabe für die Vorhersage. Damit ist nicht das Modelltraining beschrieben, welches meist auf historischen Daten X mit bekannten Label-Werten y basiert, sondern der eigentliche Prognosedurchlauf: Wir möchten aus einer Vergangenheit direkt auf eine Zukunft schließen, die Kurve also mittels Mustererkennung fortschreiben. Dabei kann sich die Zahl der Features soweit reduzieren, dass am Ende nur der zurückliegende Verlauf von y(t - n) für die Extrapolation der Werte y(t + m) verwendet wird. In diesem Fall liegen Ein- und Ausgabe im gleichen Feature-Raum. Ein solcher Ansatz ist dann attraktiv, wenn keine zukünftigen Werte aus anderen Features X(t + m) als sinnvolle Eingaben zur Verfügung stehen oder wenn diese nur unpräzise Aussagen liefern.



[**Abbildung 2:** Prognosekurven für Passagiere via LSTM]

Für den zweiten Ansatz finden - neben verschieden Schiebefenster-Methoden - neuronale Netze Verwendung, welche sequenzielle Daten verarbeiten können. Zu diesen gehören u.a. das einfache RNNs (Recurrent Neural Network) und die darauf aufbauende Architektur LSTM (Long Short-Term Memory). Diese Arbeit soll die Grundlagen dieser Netze erläutern und untersuchen, inwieweit sie sich für die Vorhersage von Stromverbrauchszyklen eignen.

1.3 Vorgehen und Aufbau der Arbeit

DSR: Für diese Arbeit wird der Design-Science-Research-Methodik gefolgt: Vor dem Hintergrund der konkreten Aufgabenstellung, werden experimentelle Machine-Learning-Modelle entwickelt und analysiert. Damit wird durch praktische Umsetzung eines Lösungsvorschlags und dessen Evaluierung Wissen generiert.

Die Ausarbeitung gliedert sich wie folgt:

Grundlagen: In diesem Kapitel werden zunächst die wesentlichen Merkmale der gewählten ML-Modelle erläutert. Dies sind das einfache RNN und die Erweiterung LSTM. Die mathematischen Grundlagen sollen hier angerissen werden, um auf Vorteile und Schwächen dieser beiden Architekturen zu schließen. Gleichzeitig wird die folgende Implementierung hier theoretisch untermauert.

Konzeption: In diesem Abschnitt werden die Anforderungen gelistet und der technische Lösungsvorschlag vorgestellt. Das Anwendungsfeld wird präzisiert und die vorhandenen Daten beschrieben. Aus diesen

Vorgaben erschließt sich die Wahl der ML-Modelle und Frameworks für die Implementation. Zuletzt müssen hier die Strategien zur Evaluation und der Aufbau von Vergleichsexperimenten erläutert werden.

Implementation: Dieses Kapitel beschreibt und dokumentiert die Implementation. Der Code, welcher vorwiegend in Form von kommentierten Jupyter Notebooks vorliegen wird, soll hier überblickshaft erläutert werden. Auf interessantere Bausteine wird im Detail eingegangen.

Evaluation: Da es sich bei der Implementation um eine Reihe von Data-Science-Experimenten handelt, sollen die generierten Prognosedaten in diesem Abschnitt evaluiert werden. Es erfolgt eine Bewertung anhand von Metriken sowie eine qualitative Inspektion der Ergebnisse.

Fazit: Zum Abschluss werden die innerhalb dieses Forschungsprojekts gewonnenen Erkenntnisse zusammengefasst. Insbesondere Schwierigkeiten und ungelöste Fragen werden erläutert. Ein Ausblick auf weiterführende Ansätze für eine zukünftige Lösung der offenen Problemstellungen rundet die Arbeit ab.

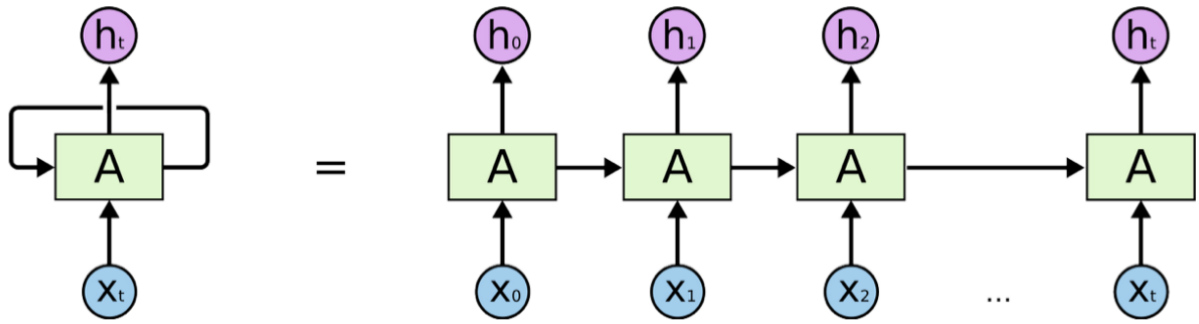
2. Grundlagen

In diesem Kapitel soll der grundsätzliche Aufbau von zwei rekurrenten neuronalen Netzen beschrieben werden: Im ersten Schritt wird das einfache RNN erläutert, welches der Ausgangspunkt für viele fortgeschrittene Architekturen ist. Eine dieser Erweiterungen ist das LSTM, welches sich im zweiten Teil findet.

2.1 Recurrent Neural Network

2.1.1 Was ist ein RNN?

Die Abkürzung RNN steht für rekurrentes neuronales Netz. Es handelt sich dabei um eine Netzarchitektur, die eine lange Folge von Daten verarbeiten kann. Die Besonderheit bei der sequenziellen Betrachtung des Datenstroms, liegt im Erinnerungsvermögen: Das Modell kennt nicht nur den aktuellen Datenpunkt sondern bezieht auch den durch vorangegangene Eingaben erreichten Zustand in jeden neuen Berechnungsschritt ein. Es handelt sich also um eine Generalisierung eines Feedforward-Netzes, welche eine Form von Gedächtnis besitzt. Das RNN vollführt die gleiche Kalkulation für jeden Input-Vektor X_t, erhält aber als zusätzliche Eingabe auch die Ausgaben h_{t-1} der letzten Berechnung - der Mechanismus ist demnach rekursiv. Die sequenzielle Verknüpfung der Berechnungen macht hier den charakteristischen Unterschied aus. Im strukturellen Vergleich zu einem herkömmlichen FNN lässt sich erkennen, das ein RNN das FNN gedanklich mehrfach kopiert und den Zustand einer jeden Instanz in die folgende überträgt. In dieser entfalteten Perspektive wird die Verkettung deutlich. Die Kopie wird in der Realität jedoch vermieden und durch einen schleifenförmigen Datenfluss ersetzt.



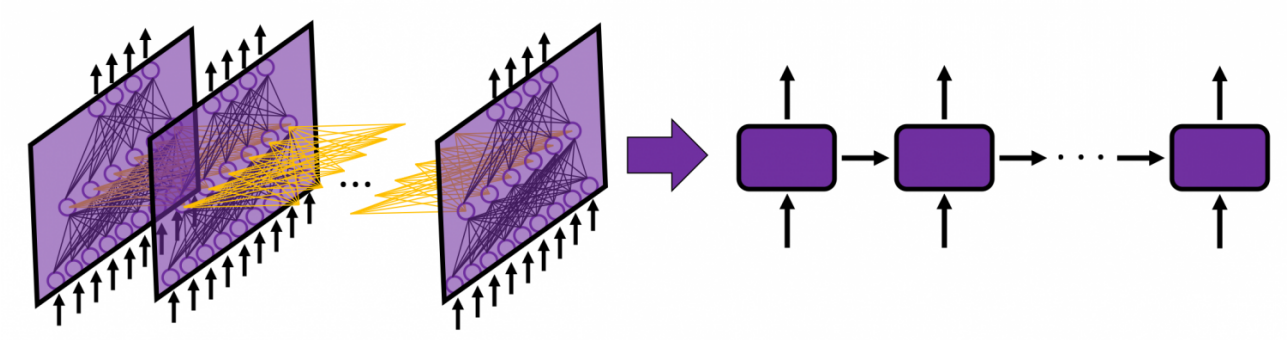
[**Abbildung 3:** RNN in zyklischer und entfalteter Systemdarstellung]

Mathematisch lässt sich dieser Aufbau wie folgt fassen:

$$o^t = f(h^t; \theta)$$
$$h^t = g(h^{t-1}, x^t; \theta)$$

[**Formel 1:** Mathematische Beschreibung eines RNN]

Der Ausgabewert o_t ergibt sich durch die Funktion f. Diese arbeitet über den Zustand h_t der versteckten Schichten mit den Parametern θ. Die zweite Formel zeigt, das h_t sich aber als eine Funktion der Eingabewerte x_t sowie des Zustands des vorangegangenen Berechnungsschritts h_{t-1} darstellt. In Abbildung 4 ist die eigentliche Vernetzung der Neuronen mit den entsprechenden Querverbindungen (gelb) zwischen den Zuständen der inneren Schichten illustriert. Hier wird deutlich, dass jede Zelle A ein vollständiges Fully Connected Network ist.



[**Abbildung 4:** RNN Netzwerkverbindung und Transfer zum Systemdiagramm]

Anmerkung: In den unterschiedlichen Quellen kommt es zu diversen Aussagen darüber, ob es sich bei den übertragenen Werten h_{t-1} um die Ausgabe des voran gegangenen Berechnungsschrittes o_t, oder um einen inneren Zustand handelt. Viele vereinfachende Datenflussdiagramme legen ersteres nahe, doch ist in der Mathematik und in Abbildung 4 zu erkennen, dass die Verknüpfung zur Ausgabeschicht o_t anders gestaltet sein können als die horizontalen Verbindungen im inneren des Netzes. Es sind dementsprechend nicht die Ausgabewerte, die beim nächsten Berechnungsschritt erneut in das Netz gespeist werden. h_t und o_t gleich zu setzen, ist eine unzulässige Verkürzung.

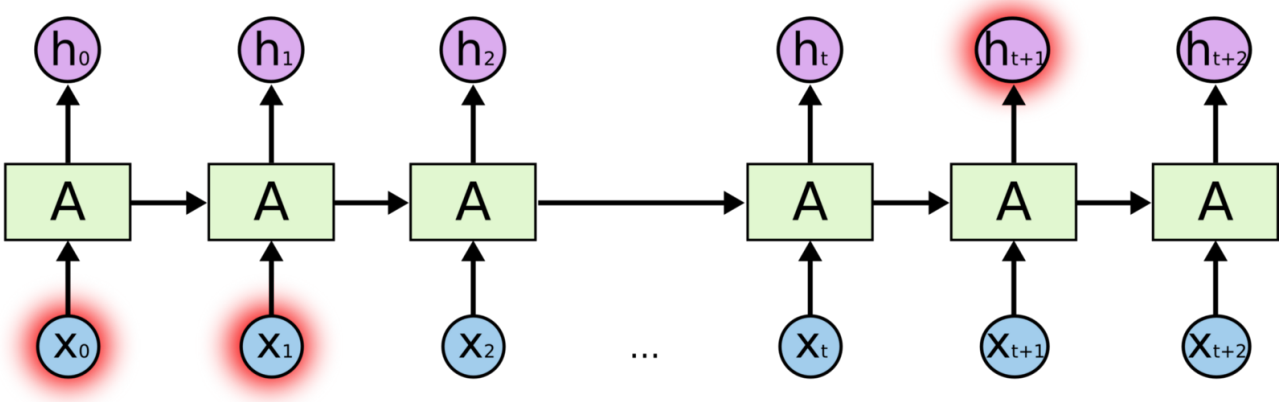
2.1.2 Backpropagation über Zeit (BPTT)

Das Training von RNN ist dem eines FNN sehr ähnlich: Auch hier erhalten wir über die Kostenfunktion eine mehrdimensionale "Kostenlandschaft", die via Gradientenabstieg zum Minimum hin beschritten werden kann. Für die Ermittlung der entsprechenden Gradienten benötigen wir wiederum das Backpropagation-Verfahren, diesmal aber über die Zeit. Diese Version arbeitet konzeptionell über das Ausrollen (s.o.) des Netzes. Daraus ergibt sich eine Form, die einem FNN gleicht, aber alle Eingaben x_t als Input-Schicht und alle Ausgaben o_t als eine Output-Schicht betrachtet. Der genaue Mechanismus ist natürlich noch etwas komplexer und beinhaltet auch Optimierungsstrategien wie das Teacher Forcing. Dies soll beim gegebenen Umfang der Arbeit aber nicht im Detail erläutert werden. Das Grundkonzept der BPTT ist jedoch für das Verständnis einer Gradientenproblematik relevant.

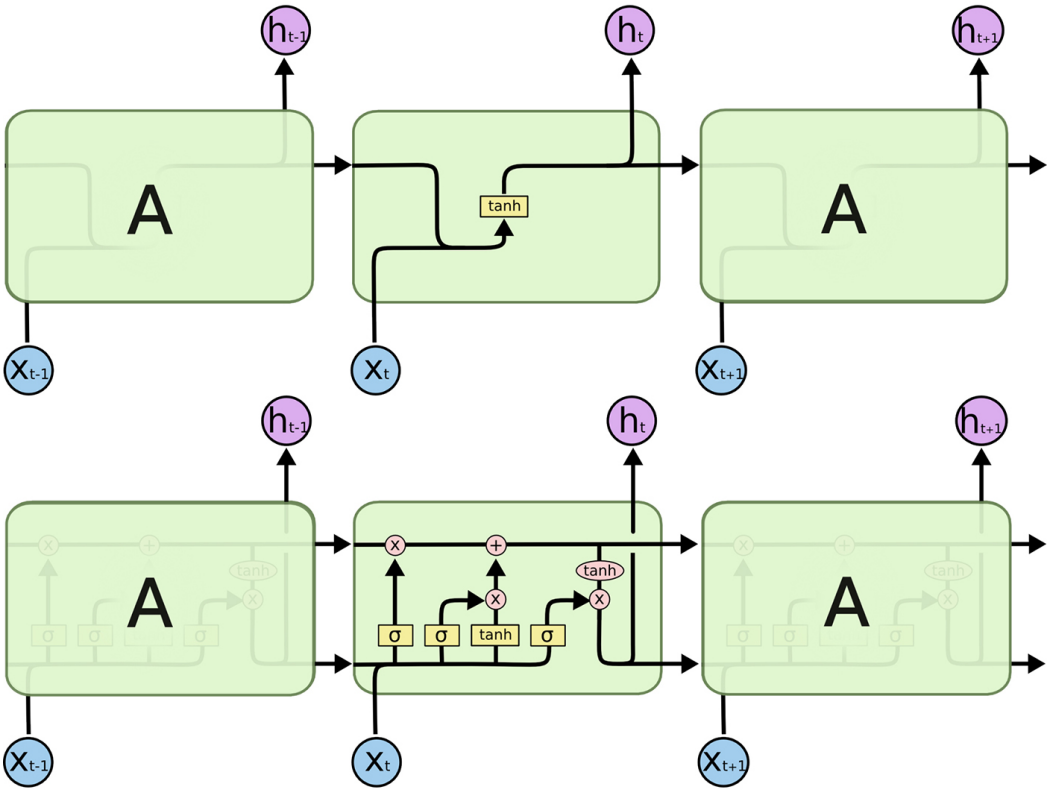
2.1.3 Vanishing Gradient

Durch das Berechnen des Gradienten in die Vergangenheit entsteht eine zum Teil extrem lange Kette von Neuronen, die jeweils eine partielle Ableitung nach sich ziehen. Bei der Verknüpfung dieser Teilableitungen

mittels der Kettenregel werden die Gewichte verkürzt gesprochen wiederholt miteinander multipliziert. Sind nun diese Gewichte W im Schnitt kleiner als 1, dann erhalten wir eine Multiplikationskette von Werten < 1. Dies bedeutet, dass der Gradient gegen 0 gehen wird und praktisch verschwindet. Gerade bei sigmoiden Aktivierungsfunktionen sind aber Ausgaben < 1 zu erwarten. Diese Problemstellung ist auch aus sehr tiefen neuronalen Netzen bekannt. In der ResNet-Architektur wird ihr u.a. über Sprungverbindungen begegnet. Da RNNs in dieser Hinsicht als extrem tiefe Netze zu werten sind und die einzelnen Schichten zudem als verwandte Variationen entstehen, ist diese Entwicklung hier dramatisch. Bei Gewichten, die > 1 sind, kommt es dagegen zu einem explodierenden Gradienten. Daher ist auch ein Wechsel der Aktivierungsfunktion (z.B. ReLu) nicht immer zielführend. Ein verschwindender Gradient führt schließlich zu einer Stagnation des Lernfortschritts beim Training des Modells. Information geht über die Zeit verloren und das Gedächtnis des Netzes bleibt kurz.



[**Abbildung 5:** RNN Vanishing Gardient]



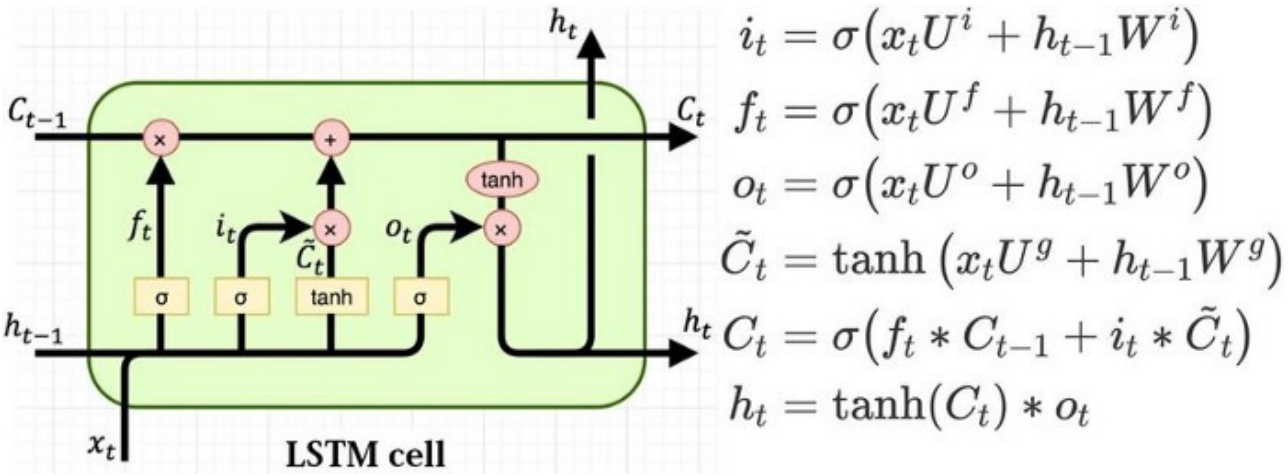
[**Abbildung 6:** Verschaltung RNN vs LSTM]

2.2 LSTM Long Short-Term Memory

2.2.1 LSTM Memory Zelle

Der Herausforderung des verschwindenden Gradienten begegnen fortschrittlichere Netzwerkarchitekturen wie das LSTM, durch eine umfassende Erweiterung der RNN-Zelle (in Abbildung 3 mit A bezeichnet). Die übergeordnete Struktur gleicht zunächst einem einfachen rekurrenten Netzwerk: In Abbildung 5 ist zu erkennen, das auch die modifizierte Zelle als Eingaben den aktuellen Datenvektor x_t sowie die vorangegangene Ausgabehypothese h_{t-1} erhält. Die berechnete Ausgabe wird äußerlich auch wie gehabt an die Folgezelle weiter gereicht. Daneben wird aber als eine Art Schnellstraße für das Gedächtnis eine zusätzlicher Zustand Ct eingeführt, der ebenso horizontal durch das Netzwerk wandert.

2.2.2 LSTM Gates



[**Abbildung 7:** LSTM-Zelle mit Gates als Funktionen]

Im inneren der Zelle steigt die Komplexität jedoch deutlich: Neue Funktionen, die als Gates bezeichnet werden, erlauben es, Information zu C addieren (erinnern) oder aus C zu entfernen (vergessen).

f_t Vergessen: Die Reduktion (forget) von Information kann durch eine Multiplikation der entsprechenden Matrizenposition mit einem Wert im Intervall [0, 1] erreicht werden - dies ist vom Vanishing Gradient her bekannt. Diese Bremswerte werden durch das Einspeisen der Eingaben x_t und h_{t-1} in die Sigmoid-Funktion erzeugt. Die entsprechende Verknüpfung löscht demnach selektiv das Gedächtnis C.

i_t Erinnern: Das Einfügen (insert) von Information kann einfach durch positionsweise Addition erfolgen. Die an diesen Strängen vorgeschalteten sigmoiden Funktionen passen lediglich die Wertebereiche günstig an, um Werteexplosionen zu verhindern.

o_t Ausgabe: Auch die Ausgabe (out) wird durch entsprechende Gates manipuliert. Das Ergebnis wird via Multiplikation zusätzlich durch den aktuellen Gedächtniszustand beeinflusst, bevor es als Prognose hervor geht und wie schon beim klassischen RNN auch in die Folgezelle wandert.

2.2.3 Vorteile in der Praxis

Aus der Beschreibung das Aufbaus einer LSTM-Zelle wird vielleicht erkenntlich, das ein derartiges Netzwerk eine deutlich bessere Modulation des Gedächtnisprozesses ermöglicht. In der Praxis hat sich gezeigt, dass LSTMs in vielen Fällen tatsächlich viel geringere Schwierigkeiten haben, Informationen auch über lange Sequenzfolgen zu erhalten und diese zudem selektiv in weitere Berechnungen einzubringen. Ein mathematischer Nachweis für die Kapazität, dem verschwindenden Gradienten entgegen zu wirken, würde den Umfang dieser Arbeit sprengen. Daher soll an dieser Stelle auf ein Zitat zurück gegriffen werden. In der Quelle ist der Beweis entsprechend nachzuvollziehen.

„LSTMs solve the problem by creating a connection between the forget gate activations and the gradients computation, this connection creates a path for information flow through the forget gate for information the LSTM should not forget.“ [Arb18]

3. Konzeption

3.1 Schwerpunkt des Experiments

Für die Überprüfung der Eignung von rekurrenten neuronalen Netzen für die Vorhersage von Energieproduktion- bzw. Verbrauchskurven stehen zwei Aufgabenbereiche zur Verfügung:

A. Photovoltaik: Die Prognose der Produktion von Strom durch Photovoltaik (gemessen als Leistung in KW) ist von hohem Interesse. Sie gibt einen Schätzwert für die regenerative Energie an, welche in einem Teilnetz voraussichtlich zur Verfügung steht. PV-Produktion richtet sich neben den fixen Eigenschaften der jeweiligen Anlage, hauptsächlich nach der Wetterlage. Da diese in Form von Wettervorhersagen des DWD (Deutscher Wetterdienst) auch als zukünftiger Datensatz vorliegt, kann für diese Fragestellung ein zeitgleiches Modell (siehe Kapitel 2) genutzt werden. Dieses wird aber vermutlich einen erheblichen Fehler aufweisen, da Wetterprognosen selbst starke Ungenauigkeiten enthalten. Zur Verbesserung wäre eine Kombinationsmodell, welches ein RNN als Nachkorrektur einbindet sicher interessant. Damit könnte die zum Prognosezeitpunkt bekannte historische PV-Produktion der jüngsten Vergangenheit einen Beitrag zur Steigerung der Präzision leisten. Eine derart komplexe Architektur, die zeitgleiche und historische Sequenzen in ein Modell speist, ist für diese Arbeit jedoch zu aufwendig.

B. Hausanschluss: Mit dem Hausanschluss ist der allgemeine Stromverbrauch eines Quartiers (in diesem Fall ein Komplex von Bürogebäuden) bezeichnet. Diese Basislast ist interessant, da sie im Regelfall von der Produktion der PV-Anlagen abzuziehen ist. Nur der Rest steht für strategische Verwendungen wie das Laden von elektrischen Fahrzeugen zur Verfügung. Der Grundverbrauch von Bürogebäuden wird aller Vermutung nach stark durch den Rhythmus der Wochentage bestimmt. Es ist anzunehmen, dass durch Kühlung bzw. Heizung, IT und personellem Lasteintrag an den Werktagen ein deutlich gesteigerter Energiebedarf besteht. Dieser variiert dann wiederum über die Tageszeit. Das Wetter spielt aufgrund des thermischen Eintrags ins Gebäude sicher auch eine Rolle. Eine erste Exploration der Daten zeigt jedoch schnell, dass der Einfluss nicht so unmittelbar ist wie etwa bei der Photovoltaik. Es bietet sich somit nicht an Wettervorhersagen als Ausgangspunkt für eine Prognose des Stromverbrauchs zu nutzen.

Somit ist der Hausanschluss ein guter Anwendungsfall für Prognosemodelle, die zeitliche Strukturmuster im historischen Stromverbrauch analysieren und auf dieser Basis die Zukunft fortschreiben. Dieser Anwendungsfall soll in der Implementation experimentell geprüft werden.

3.2 Datenlage

A. Historischer Stromverbrauch: Als Ausgangsdaten für das Experiment steht der historische Stromverbrauch eines Bürokomplexes mit bekanntem Standort (Niedersachsen) zur Verfügung. Die Daten liegen als Zeitreihen vor, welche die abgenommene Leistung in KW über die Jahre 2018-2019 sowie das erste Quartal 2020 darstellen. Die Auflösung ist durch eine zeitliche Distanz von 15 Minuten zwischen den gemessenen Werten gegeben. Die Konsistenz der Daten ist zu großen Teilen gewährleistet, auch wenn ein Monat zwischen den Jahren nicht erfasst wurde. Dieses Intervall ist dementsprechend auszuschließen.

B. Historische Wetterdaten: Für die Jahre 2018-2019 liegen historische Wetterdaten vor. Dies sind reale Messungen von Temperatur, Sonneneinstrahlung, Luftdruck und Feuchtigkeit. Die Zeitstempel lassen sich mit denen der Zeitreihen aus A zur Deckung bringen, da die Auflösung identisch ist. Es ist anzumerken, dass für die Prognose keine historischen Daten verwendet werden können, da diese für die Zukunft nicht existieren. Die Wettervorhersagen vom DWD haben jedoch abweichende Datenfelder für die Sonneneinstrahlung. Daher muss hier gegebenenfalls eine Umrechnung vorgenommen werden.

C. Wettervorhersagen DWD: Für 2020 wurden auch die Wettervorhersagen des DWD aufgenommen. Diese historischen Prognosedaten würden sich dementsprechend mit den Eingaben in das fertige Modell decken. Aus diesem Grund kann auch ein Training auf diesen im Vergleich zu den realen Messungen weniger genauen Daten interessant werden.

Die Vorhanden Daten müssen vor dem Training und Testen der Modelle gesichtet, bereinigt und auf Konsistenz geprüft werden. In Fällen, bei denen die Wetterdaten einbezogen werden, ist für eine Verknüpfung und Synchronisation mit den Stromverbrauchskurven Sorge zu tragen.

3.3 Vergleichsexperiment

In dieser Arbeit werden konkret die Modell RNN und LSTM mit einer sequenziellen Eingabe über ein Feature evaluiert. Für Training und Evaluation, sollen in der Implementierung zunächst nur die historischen Stromverbrauchswerte genutzt werden. In diesem Szenario werden die Informationen über die Wetterlage ignoriert. Dies ist der direkteste Anwendungsfall für ein rekurrentes Netz. Sequenzielle Informationen über ein Feature werden durch das Modell analysiert und fortgeschrieben.

Um die Qualität der rekurrenten Modelle vor dem Hintergrund einer Alternative zu beurteilen, wird in der Implementation ein Vergleichsexperiment auf Basis des Entscheidungswalds erstellt. Dafür eignet sich der RFR (Random Forest Regressor), da dieser oft gut auf kategorische Werte wie Wochentage oder Monate (über Indexierung numerisch dargestellt) reagiert.

Die Gegenüberstellung dieser Modell eröffnet hypothetisch einen guten Einblick in die Vorteile und Schwächen der jeweiligen Algorithmen. Insbesondere für die Qualitative Einschätzung jenseits der gängigen Metriken scheint dieser Ansatz geeignet.

4. Implementation

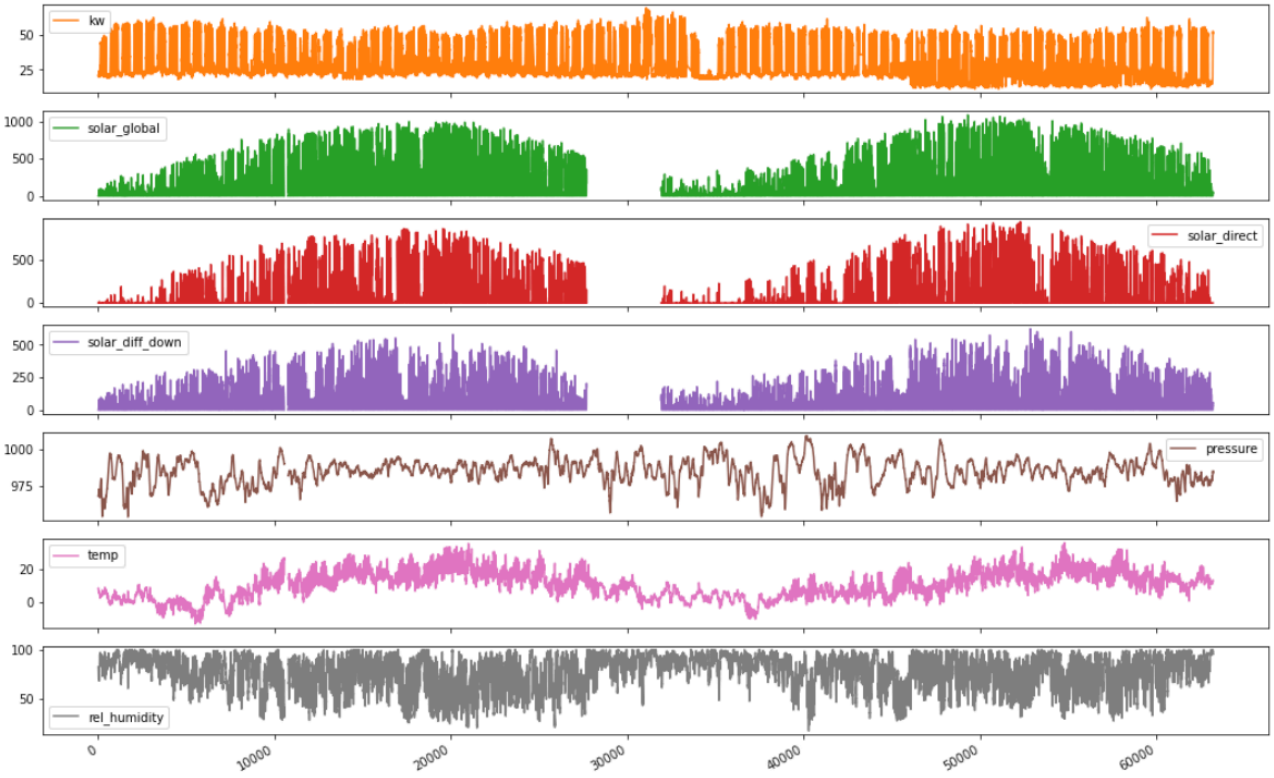
4.1 Werkzeuge und Frameworks

Die datenwissenschaftlichen Experimente wurden in Form von Jupyter Notebooks in Python implementiert. Für die Exploration und Bereinigung der Daten kamen die Bibliotheken numpy, pandas und matplotlib zum

Einsatz. RNN und LSTM-Modelle konnten über das Framework Keras erstellt werden, welches eine komfortable API zu Googles Tensorflow ist. Für die Vergleichsstudie wurde der Random Forest Regressor aus den SciKit-Learn-Paketen eingebunden.

4.2 Datenaufbereitung

Konsistenz: Für das Training und Testen der Modelle wurde die in verschiedenen CSV- und Json-Dateien vorliegenden Daten in einen Dataframe vereint. Abbildung 8 zeigt, das die Zeitreigen zu großen Teilen vollständig sind. In den Monaten Dezember 2018 und Januar 2019 wurden jedoch keine Solarwerte dokumentiert. Daher ist dieser Intervall in jenen Fällen auszuschließen, wo diese Features eingesetzt werden. Für die sequenziellen Modelle RNN und LSTM, kann eine derartige Unterbrechung der Kontinuität des Zeitstrahls besonders problematisch werden. In den entsprechenden Experimenten wurden die Wetterdaten aber ignoriert. Lediglich in der Vergleichsstudie (DRF) nutzte auch die Sonneneinstrahlung. Der Random Forest verarbeitet die Daten aber ohne Verkettung der Zeitstempel.



[**Abbildung 8:** Zeitreihendaten Wetter und Stromverbrauch]

ts	kw	solar_global	solar_direct	solar_diff_down	pressure	temp	rel_humidity	month	day	hour	minute	weekday	holiday
2019-10-21 07:00:00	53.0	31.666667	0.0	31.666667	983.80	12.90	97.00	10	21	7	0	0	0
2019-10-21 07:15:00	51.0	28.333333	0.0	28.333333	984.25	13.25	95.25	10	21	7	15	0	0
2019-10-21 07:30:00	52.0	58.333333	0.0	58.333333	984.40	13.00	95.50	10	21	7	30	0	0
2019-10-21 07:45:00	52.0	35.833333	0.0	35.833333	984.95	13.00	96.30	10	21	7	45	0	0
2019-10-21 08:00:00	52.0	46.666667	0.0	46.666667	985.20	13.20	96.20	10	21	8	0	0	0

[**Tabelle 1:** Auszug aus dem angereichertem Trainingsdatensatz]

Anreicherung: Insbesondere Für den Random Forest Regressor ist eine Aufspaltung des Zeitstempels in einzelne Spalten vorteilhaft. Die String-Darstellung lässt sich für diesen Algorithmus nicht als Kontinuum interpretieren. Eine Umrechnung in einen einzigen Integer etwa durch Konvertierung in die Unix-Millis-Zeit

ist jedoch auch nicht ideal, da dieses Format keinen Bezug zum Wochen- oder Tagesrhythmus herstellt. Aus diesem Grund wurden Monate, Wochen, Tage und Stunden getrennt indexiert. Die Tage konnten schließlich in die Wochentage [1-7] überführt und eine zusätzliche Spalte für Feiertage angefügt werden. So wurde aus dem Zeitstempel Information generiert, die eine stärkere Korrelation zwischen Stromverbrauch und Aktivitätszyklus im Bürogebäude impliziert.

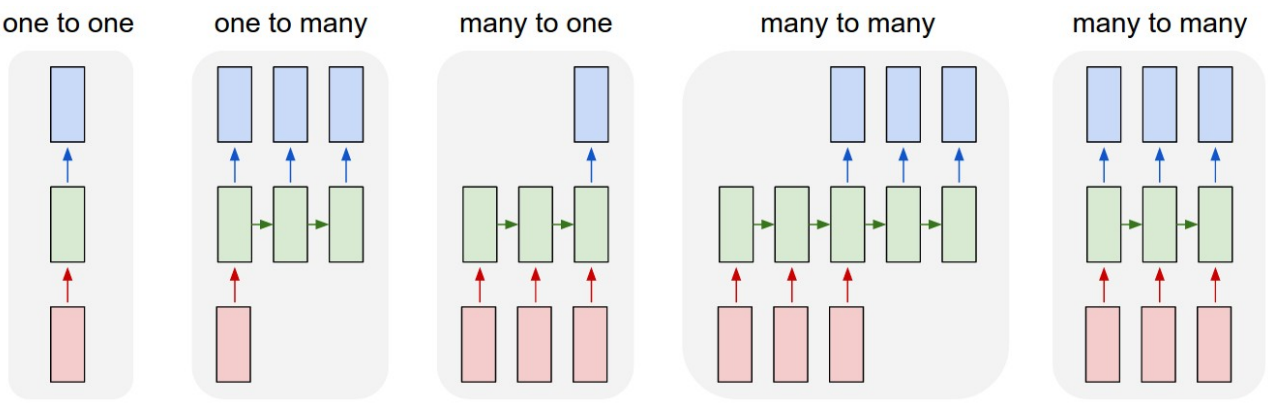
Normalisierung: Für die Eingabe in neuronale Netze sind die Daten in der Regel zu normalisieren. Für den Entscheidungsbaum ist dieser Prozess hingegen nicht ratsam. Auch in der Regressorvarienate kommt der DRF besser mit besser unterscheidbaren Ausprägungen der Features zurecht, da sich bei Semi-kathegorischen Werten tendeziell weniger Äste im Baum ausbilden. Ein Explodieren der Werte innerhalb der Berechnungen ist hier nicht zu erwarten.

4.3 Datenteilung

Für die Teilung in Trainings- und Testdaten wurde eine Schwelle von 80% angestrebt. In einer weitere Separierung der 20% Testdaten in Test- und Validierungsblöcke wurde in diesen Experimenten verzichtet. Eine unvoreingenommene Bewertung der Ergebnisse durch den Test mit Daten, die nicht bei Modellselektion und Hyperparamtertuning beteiligt sind, ist im Produktiveinsatz anzuraten, für diese Studie aber nicht essenziell.

DRF: Für das Vergleichsmodell wurden zwei unterschiedliche Teilungsmethoden getestet: Der erste Durchlauf wurde über ein zufälliges Samplen der Trainigsdaten gespeist. Dies hat im Schnitt ein effektiveres Training zur Folge, da der ganze Zeitrahmen erfasst wird. Allerdings sind die restlichen Daten für das Testen nicht über einen dichten, kontinuierlichen Zeitraum verteilt, was eine visuelle Bewertung der Plots - z.B. durch Gegenüberstellung mit den Wochentagen - deutlich erschwert. Daher wurde eine weitere Teilung durch Abschneiden von ganzen Wochen vorgenommen. Die Vorhersage arbeitet hier vom Featureraum der Wetter- und Zeitspalten X hin zum KW y (zeitgleiches Model). Also ist dieK KW-Spalte als Label abzulösen.

RNN (many-to-one): Die Zusammenstellung eines Datensatzes für ein rekurrentes Netzwerk gestaltet sich etwas komplexer. Hier sind die Eingabevektoren Sequenzen (in diesem Fall mit nur einem Feature - KW). Somit mussten aus den Rohdaten ganze Zeitintervalle extrahiert werden. Diese Vektoren können dann zu einer Matrix "zusammengeschoben werden". Die Sequenzen von gleicher Länge werden somit gestapelt. Als Label ist die letzte Spalte dieser Matrix zu isolieren, da Eingabe- und Zieldaten sich nur durch die zeitliche Abfolge unterscheiden. Bei diesem Vorgehen wird über eine Eingabesequenz X_i nur eine Zielzeitpunkt y_i berechnet - es handelt es sich somit um eine Many-to-One-Prognose.



[**Abbildung 9:** Problemkonfigurationen bei RNNs]

RNN (many-to-many): Alternativ können mehrere Spalten am Ende der Matrix abgeschnitten werden, um eine Many-to-Many-Konfiguration zu erzielen. In diesem Fall ist aber auch ein neues Modell zu erzeugen, das aus einer einzelnen Sequenz eine Folgesequenz generiert. Dieses Netz unterscheidet sich deutlich von einem Many-to-One-Aufbau, da die Ausgabeschicht (Denke) nicht nur mehrere Knoten aufweist, sondern zeitlich entkoppelt, wiederholt angewendet wird. Daher besitzt dieses Netz deutlich mehr Parameter und provoziert eine lange Trainingsdauer. Um dem entgegen zu wirken, wurde die Auflösung der Daten auf stündliche Intervalle runter gerechnet. Es ist dann möglich eine Zeitraum von zwei Wochen mit 336 statt vorher 1344 Samples abzubilden. Zwei Wochen sind vermutlich eine gute Sequenzlänge, da sich entscheidende Werktagszyklen über diesen Zeitraum darstellen.

4.4 Aufbau und Training RNN-Modell (many-to-one)

Ein RNN-Modell lässt sich in Keras relativ einfach zusammensetzen, da das Framework bereits vorgefertigte RNN-Layer bereit stellt. In diesem Bespiel wurden drei Hidden-Layer mit jeweils 40 Neuronen gewählt. Die Eingabesequenz ist 20 Zeitstempel lang. Der Tangens hyperbolicus als Aktivierungsfunktion findet sich in vielen RNN-Anwendungen, da er en Explodieren der Daten verhindert - nicht aber den Gradientenschwund. Der Aufbau wird durch einen Dense-Layer abgeschlossen, der den einzelnen Ausgabewert y_{t+1} produziert.

```
rnn_model = Sequential()
rnn_model.add(SimpleRNN(40, activation='tanh', return_sequences=True, input_shape=(X_train.shape[1], 1)))
rnn_model.add(Dropout(0.15))
rnn_model.add(SimpleRNN(40, activation='tanh', return_sequences=True))
rnn_model.add(Dropout(0.15))
rnn_model.add(SimpleRNN(40, activation='tanh', return_sequences=False))
rnn_model.add(Dropout(0.15))
rnn_model.add(Dense(1))
rnn_model.summary()
```

[**Auszug Code 1:** Keras Konstruktion RNN-Modell (many-to-one)]

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_9 (SimpleRNN)     (None, 20, 40)            1680
_________________________________________________________________
dropout_12 (Dropout)         (None, 20, 40)            0
_________________________________________________________________
simple_rnn_10 (SimpleRNN)    (None, 20, 40)            3240
_________________________________________________________________
dropout_13 (Dropout)         (None, 20, 40)            0
_________________________________________________________________
simple_rnn_11 (SimpleRNN)    (None, 40)                3240
_________________________________________________________________
dropout_14 (Dropout)         (None, 40)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 41
=================================================================
Total params: 8,201
Trainable params: 8,201
Non-trainable params: 0
_________________________________________________________________
```

[**Tabelle 2:** Keras Zusammenfassung RNN-Modell (many-to-one)]

Nach Kompilieren des Modell wurde das Training über 10 Epochen bei einer Batch-Größe von 1000 Sequenzen angestoßen. Das Training reduzierte den Mean-Squared-Error-Loss via Adam-Optimierer stringent von 0,1313 auf 0,0056.

4.5 Aufbau und Training LSTM-Modell (many-to-one)

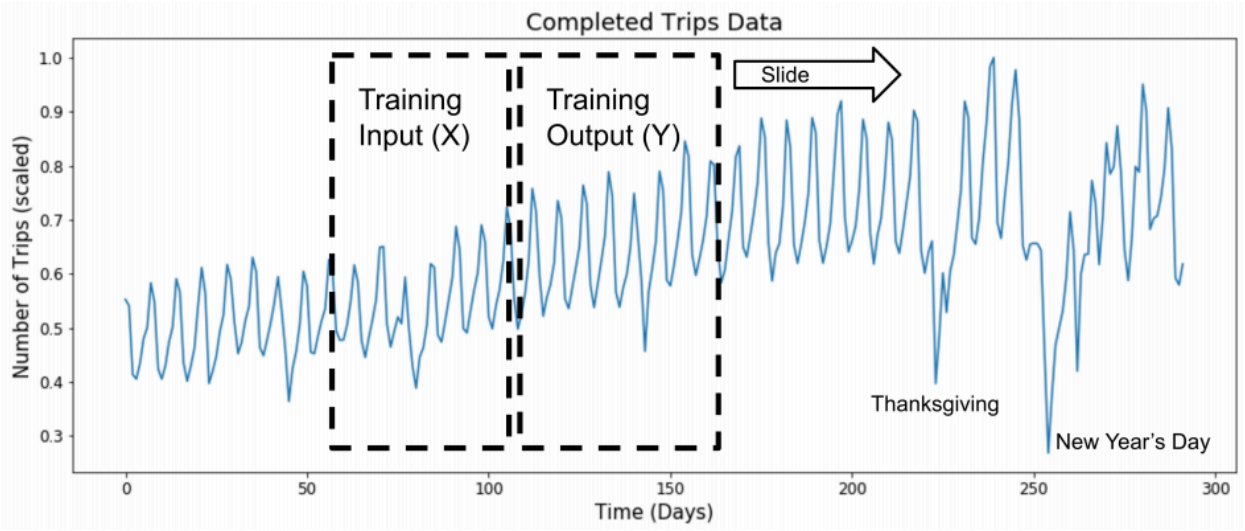
Das LSTM-Modell lässt sich in Keras in analoger Weise zum RNN zusammensetzen. Wir verwenden hier jedoch den LSTM-Layer. Interessant ist, das dieses Modell bei gleicher Neuronenzahl über ca. die 4-fache Anzahl an Parametern verfügt. Das Training bracht dementsprechend auch die 3-fache Zeit. Es reduziert den Kosten von 0,0335 auf 0,0040.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 20, 40)            6720
_________________________________________________________________
dropout_15 (Dropout)         (None, 20, 40)            0
_________________________________________________________________
lstm_4 (LSTM)                (None, 20, 40)            12960
_________________________________________________________________
dropout_16 (Dropout)         (None, 20, 40)            0
_________________________________________________________________
lstm_5 (LSTM)                (None, 40)                12960
_________________________________________________________________
dropout_17 (Dropout)         (None, 40)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 41
=================================================================
Total params: 32,681
Trainable params: 32,681
Non-trainable params: 0
_________________________________________________________________
```

[**Tabelle 3:** Keras Zusammenfassung LSTM-Modell (many-to-one)

4.6 Aufbau und Training rekurrente Modelle (many-to-many)

Der Aufbau einer Many-to-Many-Konfiguration kann in Keras mit dem TimeDistributed-Layer erreicht werden. Dieser sorgt dafür, das der Dense-Layer für die Ausgabe auch tatsächlich bei jedem Zeitstempelschritt neu angewendet wird. Würden wir die dichte Ausgabeschicht nicht derart entkoppeln, dann würde sie auf alle Ausgaben des LSTM-Layers in einem Verbund angewendet. Dies entspräche dann aber nicht mehr der rekurrenten Logik, sondern mehr einer Schiebefenster-Systematik mit klassischem Feedforward-Netz. Im diesem letzten Fall wären nur zeitversetzte Ein- und Ausgaben über ein dichtes Netz verknüpft. Das LSTM hingegen soll mehrfach einzelne Ausgabewerte rekurrent berechnen, um die Sequenz zu erstellen.



[**Abbildung 10:** Schiebefenster Konfiguration - FNN oder LSTM (many-to-many)]

Interessant wird auch die Zahl der Neuronen im Hidden-Layer (im Codebeispiel 2: 320). Im Experiment wurde diese Zahl stark variiert, da sie das Ergebnis massiv beeinflusst. Es ist hierbei in keiner Form zwingend, die Anzahl der verstecken Neuronen gemäß der Länge der Input-Sequenz auszurichten.

```
lstm_model_mtm = Sequential()
lstm_model_mtm.add(LSTM(320, input_dim=1, return_sequences=True))
lstm_model_mtm.add(TimeDistributed(Dense(1)))
lstm_model_mtm.add(Activation('linear'))
lstm_model_mtm.summary()
```

[**Auszug Code 2:** Keras Konstruktion LSTM-Modell (many-to-many)]

4.6 Training Random Forest Regressor

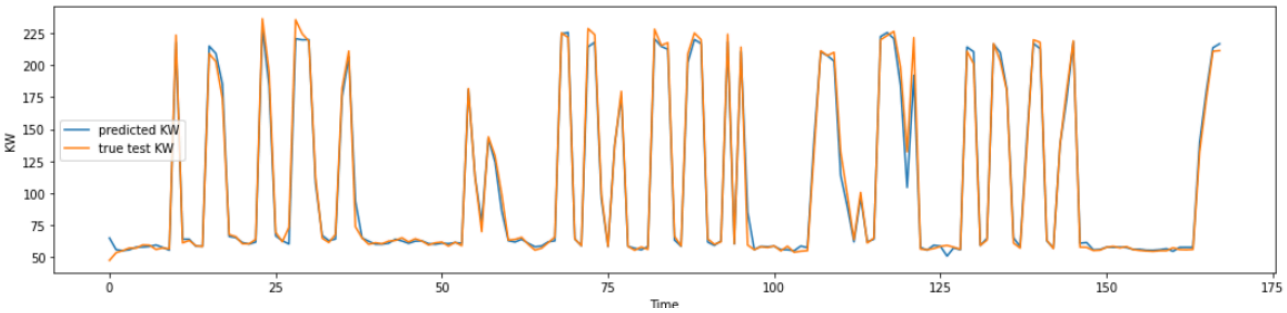
Der Random Forest Regressor kann direkt aus der SciKit-Learn-Bibliothek instanziiert werden. Dem Modell sind nur die passenden Hyperparameter zu übergeben.

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=30, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=50, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

[**Tabelle 4:** SciKit-Learn Zusammenfassung Random Forrest Regressor]

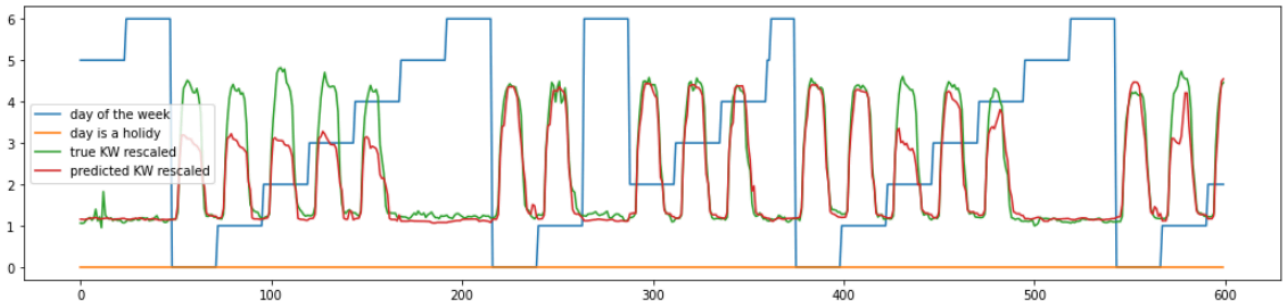
5. Evaluierung

Für diese Arbeit wurden eine Reihe von Experimenten mit unterschiedlichen Modellen durchgeführt. Somit steht neben den verwandten Modelle, RNN und LSTM, auch das auf dem Entscheidungsbaum basierende Regressionsverfahren zum Vergleich. Wir evaluieren also neben den Varianten rekurrenter Netze auch den Past-to-Future-Ansatz als solchen. Der Vergleich muss hier aus Platzgründen kurz gehalten werden. Daher wird vor allen auf die qualitative Gegenüberstellung der wesentlichsten Plots gebaut. Ausschweifende Listen von Metriken und Fehlerquotienten sollen hier vermieden werden. Zur besseren Einsicht der Merkmale wurden zwei Wochen aus den Prognosekurven selektiert.



[**Abbildung 11:** zufällige Prognose-Samples - Random Forest Regressor]

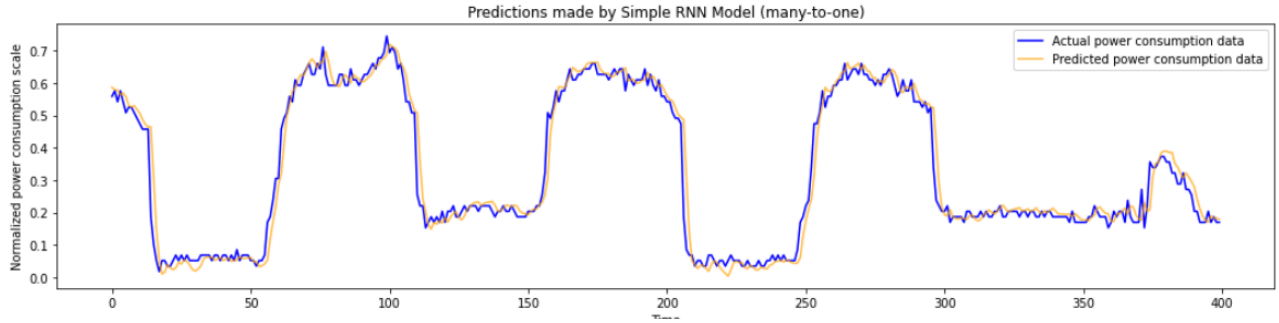
5.1 Random Forest Regressor



[**Abbildung 12:** 2 Wochen Prognose-Sequenz mit Werktagezyklus - Random Forest Regressor]

Der Random Forest Regressor funktioniert erstaunlich gut: Scheinbar lassen sich durch die via Datenanreicherung erzeugten Zeitmerkmale sehr gut einzelne Muster für den Stromverbrauch ableiten. Insbesondere die plane Niedriglast am Wochenende wird sehr sicher vorhergesagt. Jedoch ist zu erwähnen, das ein zufälliges Samplen der Trainingsdaten hier einen deutlich geringeren Fehler erzeugte (nRSME von 2,58%). Beim Abtrennen von kompakten Wochen kommt es zu größeren Abweichung: Ganze Wochen werden durchgängig mit gleichem, zu niedrigem Verbrauch angesetzt (nRSME von 10,02%). Es entsteht der Eindruck, dass sich historische Muster zu direkt in die Prognose stempeln. Dies wäre aber noch im Detail zu prüfen.

5.2 Many-to-One RNN und LSTM



[**Abbildung 13:** 2 Wochen Prognose-Sequenz - RNN (many-to-one)]

Bei der Gegenüberstellung von Prognose- und Testdaten beim RNN in der Many-to-One-Konfiguration, kommt zunächst Begeisterung auf. Die Kurven liegen sehr dicht beieinander, nur eine leichter zeitlicher Versatz ist sichtbar. Die Prognosedaten schleppen hinterher. Die Ergebnisse des LSTMs unterscheiden sich in diesem Experiment nur marginal: Es ist sogar eine minimale Verschlechterung durch das erweiterte Gedächtnis zu beobachten.

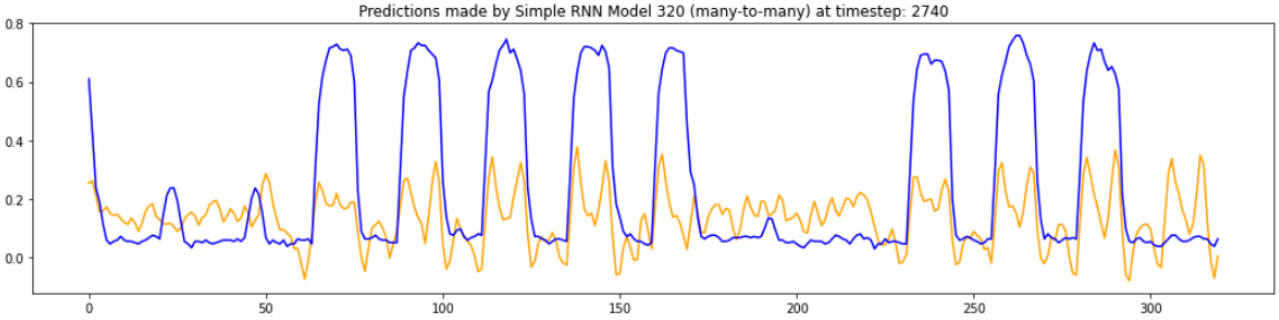
Bei näherer Überlegung wird jedoch klar, das die Modelle für den Anwendungsfall wenig leisten: Schließlich kann zu einem Zeitpunkt t in der Produktionsumgebung nur ein einziger Wert für t+1 berechnet werden. Dies ist verhältnismäßig einfach, wenn der Zustand der jüngsten Vergangenheit bekannt ist. Der Versatz zeigt deutlich, dass das Modell vor allem auf schon real gewordene Veränderungen reagiert. Die Mustererkennung über einen Zeitraum von zwei Wochen kann zunächst nicht nachgewiesen werden. Eine Sequenz entsteht in diesem Experiment nur, weil die Testdaten als Paket eine kontinuierliche Sequenz erzeugen. Das einzelne Testdatum hat hingegen kaum Prognosekraft im Sinne der Aufgabenstellung.

Anmerkung: Die große Anzahl der Beiträge im Internet, die genau einen solchen Aufbau für die Vorhersage von Sequenzen vorstellen, zeigt, dass eine passende Kurve leicht zu Fehleinschätzungen führen kann: Ohne

technische Durchdringung der Problemstellung wird hier aufgrund einer vermeintlichen Genauigkeit das falsche Verfahren gewählt, welches keine Zeitreihe sondern einen Einzelwert erzeugt.

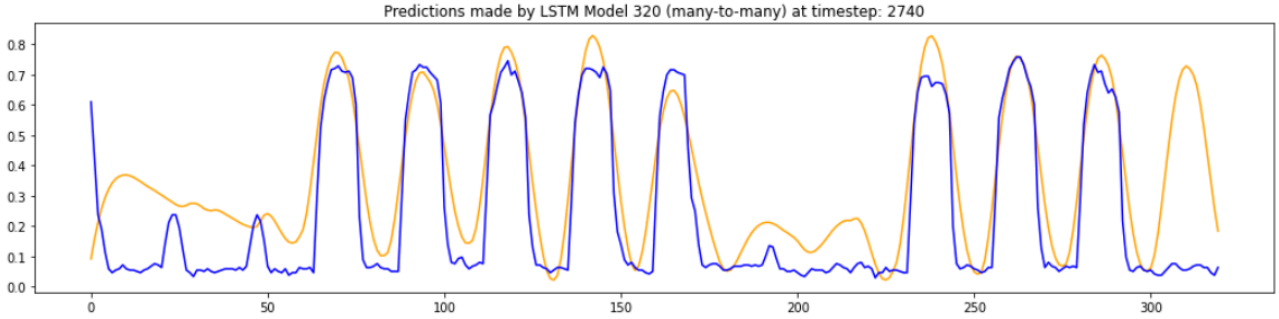
5.3 Many-to-Many RNN und LSTM

Die Many-To-Many-Konfiguration von rekurrenten Netzen fasst nun die Problemstellung: Für jede Sequenz der Eingabe wir eine gleichlange Sequenz in die Zukunft projiziert. Hier wird die intelligente Mustererkennung richtig gefordert: Dementsprechend zeichnet sich beim RNN zunächst ein schlechteres Bild. Auch wenn eine Aufnahme der Rhythmik zu erkennen ist, sind doch die Amplituden stark verfälscht.

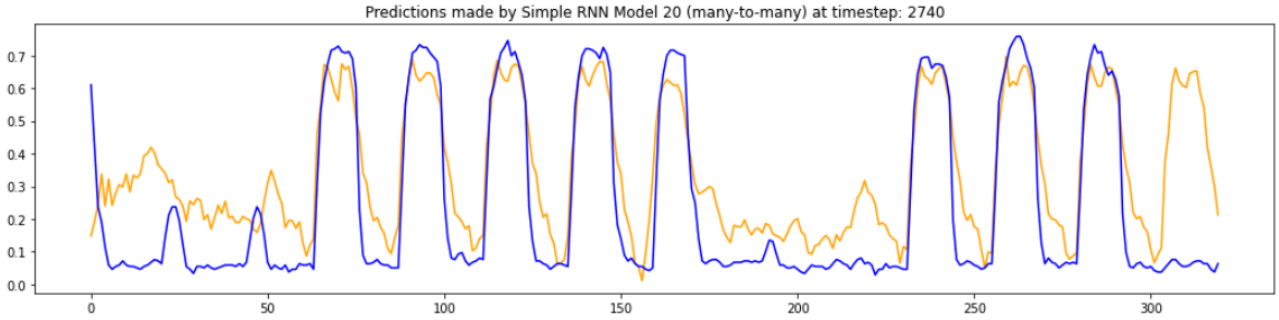


[**Abbildung 14:** 2 Wochen Prognose-Sequenz - RNN (many-to-many) - 320 Hidden Neurons]

Das LSTM scheint hier seine längeres Gedächtnis auszuspielen, um den eigentlichem Zyklus der Wochentage zu erkennen. Die Kurve trifft die Amplituden deutlich besser und beschreibt auch die Talsohle am Wochenende im Prinzip. Interessant ist aber, das der Graph deutlich weniger Mikroamplituden zeigt. Das LSTM scheint die Eigenschaften etwas zu stark zu glätten, was insbesondere bei den steilen Anstiegen am Wochenbeginn und -ende zu größeren Fehlern führt. Dennoch ist das LSTM dem RNN zu diesem Zeitpunkt klar überlegen.



[**Abbildung 15:** 2 Wochen Prognose-Sequenz - LSTM (many-to-many) - 320 Hidden Neurons]



[**Abbildung 16:** 2 Wochen Prognose-Sequenz - RNN (many-to-many) - 20 Hidden Neurons]

Ein überraschendes Resultat erbringt schließlich die Reduktion der Neuronen im Hidden-Layer des RNNs von 320 auf 20. Bei dieser Zusammensetzung, kann selbst das RNN nun relativ Präzise den

Amplitudenzyklus der Wochentage replizieren. Dies zeigt, dass ein RNN dem LSTM nicht grundsätzlich unterlegen sein muss. Vielmehr spielt der genaue Aufbau in Zusammenhang mit der Problemstellung und den vorhandenen Daten eine entscheidende Rolle. Es muss hierbei erwähnt werden, dass die Breite an Möglichkeiten von Einstellungen und Kombinationen in dieser Arbeit kaum berührt werden konnte.

6. Fazit

6.1 Zusammenfassung

Die Reihe an Experimenten mit den rekurrenten Netzen, RNN und LSTM, hat gezeigt, dass eine Zeitreihenanalyse über die Mustererkennung in Sequenzen zumindest sinnvolle Prognosen liefert. Interessant sind diese Modelle insbesondere für Anwendungsfälle, wo keine weiteren Daten als der historische Verlauf der fortzuschreibenden Kurve gegeben sind. Dabei können RNN und LSTM durchaus auch mit einem größeren Featureraum umgehen und dann durch diese anderen Merkmale noch komplexere Muster und Zusammenhänge identifizieren.

Ein klarer Gewinner in der Gegenüberstellung von RNN und LSTM ist nicht festgestellt worden. Für den Einsatz zur Vorhersage von Stromverbrauchszyklen scheinen beide Versionen im Grundsatz geeignet, auch wenn eine sehr genaue Anpassung der einzelnen Schichten vorzunehmen ist. Das LSTM liefert gerade beim schnellen Testen ohne Tuningdurchläufe zuverlässiger akzeptable Ergebnisse. Die Anzahl der Parameter liegt hier aber deutlich höher, was das Training und auch die Reaktionszeit bei der Vorhersage verlängert. Die Genauigkeit des RNNs in der Many-to-Many-Konfiguration schwankte wesentlich stärker, wenn die Neuronenzahl im Inneren variiert wurde. Allerdings wären genaue Aussagen, bei der Menge an getesteten Zusammenstellungen verfrüht.

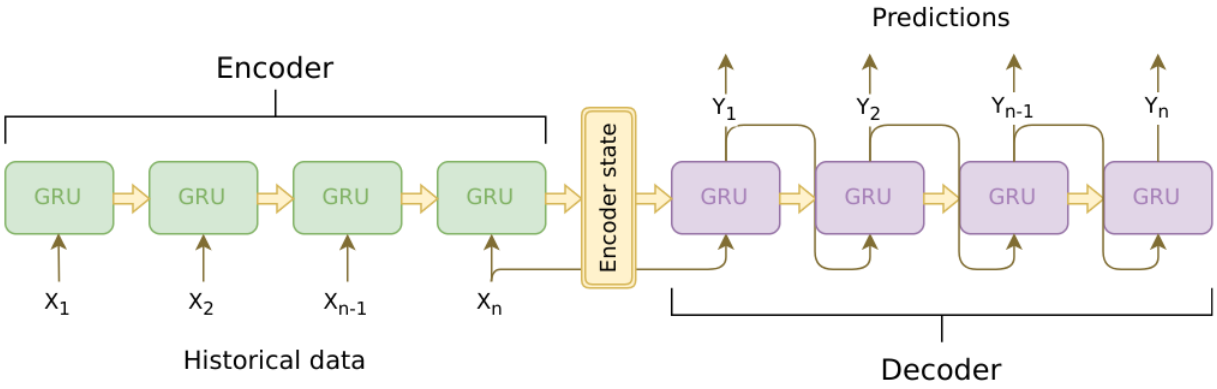
Als überraschend präzise erwies sich der Random Forest Regressor im Zusammenspiel mit einer inhaltlich fundierten Datenanreicherung. Es muss hier noch einmal ausdrücklich erwähnt werden, dass eine Spaltung der Zeitstempel in semi-kathegorische Werte für zyklische Zeitvariablen (Wochentag, Tagesstunde, Feiertage) die Prognose entscheidend unterstützt. Diese Variablen bilden einen Feature-Raum, der bei einem zeitgleichen Experiment ein Stempeln von vergangenen Sequenzen in die Zukunft ermöglicht. Es ist zu vermuten, das dieser Ansatz scheitert, wenn sich Muster nicht mit den Schleifen dieser repetitiv Zeitwerte decken. In dieser Hinsicht ist der Stromverbrauch eines Bürokomplexes ein Idealfall. Auch bleibt zu untersuchen, ob die genutzten Wetterdaten tatsächlich einen eher geringen Einfluss auf den Entscheidungswald hatten.

6.2 Ausblick

Diese Arbeit stellt eine erste Grundsatzuntersuchung für die Eignung der rekurrenten neuronalen Netze für die Prognose von Stromverbrauchsdaten dar. Sie muss bei dem gegebenen Umfang an der Oberfläche bleiben. Dennoch wäre eine deutlich genauere Untersuchung der angedeuteten Zusammenhänge denkbar:

[1] Es bliebt zu klären ob eine Konfiguration von RNN und LSTM gefunden werden kann, die die Eigenschaften der Daten noch besser erfasst. Dies ist besonders interessant, da für den Stromverbrauch durch Domänenwissen schon intuitive Annahmen getroffen werden können. Korrelationen mit den Tageszyklen durch Arbeitsaktivitat im Büro sind doch sehr offensichtlich. Desto spannender ist die Frage, wann ein Modell diesen einfachen Zusammenhang lernen kann.

[2] Bei der Many-to-Many-Konfiguration gibt es noch offene Fragen. Der genaue Mechanismus der Zeitreihengeneration ist zu überprüfen, da die wiederholte Anwendung des Dense-Layers noch etwas unpassend erscheint. Die Erforschung von Encoder-Decoder-System sollte hierbei weitere Erkenntnisse liefern. Die wirkliche Entkopplung des Lese- und Schreibprozesses ist eventuell relevant, wenn eine Sequenz vollständig analysiert werden muss, um diese präzise fortzuschreiben.



[**Abbildung 17:** Encoder-Decoder Systematik]

7. Anhang

7.1 Abbildungsverzeichnis

7.2 Formeln und Graphen

7.3 Tabellenverzeichnis

7.4 Online Quellen

Anmerkung: Bei der Erstellung dieser Arbeit wurden ausschließlich Online-Quellen benutzt. Eine gesonderte Literaturangabe erfolgt aus diesem Grund nicht. Zitate finden sich nur an einer Stelle im Text, so dass hier vor allem die zur Recherche genutzten Seiten referenziert sind.

[Arb18] Nir Abel, *How do LSTM networks solve the problem of vanishing gradients?*, 2018

URL: https://mc.ai/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients/,

zuletzt besucht: 25.09.2020

[Fer19] Patrice Ferlet, *How to work with Time Distributed data in a neural network*, 2019

URL: https://medium.com/smileinnovation/how-to-work-with-time-distributed-data-in-a-neural-network-b8b39aa4ce00,

zuletzt besucht: 25.09.2020

[Ker20] Keras API reference

URL: https://keras.io/api/layers/recurrent_layers/time_distributed/,

zuletzt besucht: 25.09.2020

[Kum20] *Harshid Kumar, The gradient problem in RNN*, 2020

URL: https://kharshit.github.io/blog/2019/01/04/the-gradient-problem-in-rnn,

zuletzt besucht: 25.09.2020

[Mel18] Kathrin Melcher, “Once Upon A Time … “ by LSTM Network, 2018

URL: https://www.knime.com/blog/text-generation-with-lstm,

zuletzt besucht: 25.09.2020

[Bro17] Jason Brownlee,
Difference Between Return Sequences and Return States for LSTMs in Keras, 2017

URL: https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/,

zuletzt besucht: 25.09.2020

[Yas19] Yasuto Tamura, Prerequisites for understanding RNN at a more mathematical level, 2019

URL: https://data-science-blog.com/blog/2020/06/01/prerequisites-for-understanding-rnn-at-a-more-mathematical-level/,

zuletzt besucht: 25.09.2020

7.5 Quellen Abbildungen

[**Abbildung 1:** Prognosekurven für die Leistung einer PV-Anlage], Quelle: eigene

[**Abbildung 1:** Prognosekurven für die Leistung einer PV-Anlage], Quelle: eigene

[**Abbildung 2:** Prognosekurven für Passagiere via LSTM]

URL: https://medium.com/swlh/a-quick-example-of-time-series-forecasting-using-long-short-term-memory-lstm-networks-ddc10dc1467d, zuletzt besucht: 25.09.2020

[**Abbildung 3:** RNN in zyklischer und entfalteter Systemdarstellung]

URL: https://www.knime.com/blog/text-generation-with-lstm, zuletzt besucht: 25.09.2020

[**Abbildung 4:** RNN Netzwerkverbindung und Transfer zum Systemdiagramm]

URL: https://data-science-blog.com/blog/2020/06/01/prerequisites-for-understanding-rnn-at-a-more-mathematical-level/], zuletzt besucht: 25.09.2020

[**Abbildung 6:** Verschaltung RNN vs LSTM]

URL: https://github.com/topics/gtzan-dataset, zuletzt besucht: 25.09.2020

[**Abbildung 7:** LSTM-Zelle mit Gates als Funktionen]

URL: https://towardsdatascience.com/multiclass-text-classification-using-lstm-in-pytorch-eac56baed8df, zuletzt besucht: 25.09.2020

[**Abbildung 8:** Zeitreihendaten Wetter und Stromverbrauch]

[**Abbildung 9:** Problemkonfigurationen bei RNNs]

URL: https://stackoverflow.com/questions/43034960/many-to-one-and-many-to-many-lstm-examples-in-keras, zuletzt besucht: 25.09.2020

[**Abbildung 10:** Schiebefenster Konfiguration - FNN oder LSTM (many-to-many)]

URL: https://machinelearningmastery.com/lstm-model-architecture-for-rare-event-time-series-forecasting/, zuletzt besucht: 25.09.2020

[**Abbildung 11:** 2 Wochen Prognosesamples - Random Forest Regressor], Quelle: eigene

[**Abbildung 12:** 2 Wochen Prognose mit Werktagzyklus - Random Forest Regressor], Quelle: eigene

[**Abbildung 13:** 2 Wochen Prognose - RNN (many-to-one)] Quelle: eigene

[**Abbildung 14:** 2 Wochen Prognose - RNN (many-to-many) - 320 Hidden Neurons], Quelle: eigene

[**Abbildung 15:** 2 Wochen Prognose - LSTM (many-to-many) - 320 Hidden Neurons], Quelle: eigene

[**Abbildung 16:** 2 Wochen Prognose - RNN (many-to-many) - 20 Hidden Neurons], Quelle: eigene

[**Abbildung 17:** Encoder-Decoder Systematik]

URL: https://jeddy92.github.io/JEddy92.github.io/ts_seq2seq_intro/, zuletzt besucht: 25.09.2020