



Fachhochschule Südwestfalen
Fachbereich Ingenieur- und Wirtschaftswissenschaften
Neuronal Network and Deep Learning (Prof. Dr. Thomas Kopinski und
Felix Neubürger)
Gutachter: Prof. Dr. Thomas Kopinski und Felix Neubürger

Vorhersage der Stromerzeugung basierend auf Self Learning Activation Functions und Liquid Neuronal Networks

Vitali Krilov, Vladislav Stasenko

Zusammenfassung

Bei der Vorhersage von Zeitreihen kann die Anpassung an die aktuelle Situation eine große Rolle spielen. So i.d.R. ob z.B. Umwelteinflüsse oder Pandemien eine große Veränderung in einer Zeitreihe herbeiführen. Der erste Ansatz für eine Vorhersage von Zeitreihen basiert in dieser Arbeit auf den Liquid Neuronal Networks (LNN). Die LNNs sind dynamisch und passen sich an die aktuelle Situation an, lernen also immer weiter. Als eine Ergänzung zu den LNNs wird die Self Learning Activation Function (SLAF) implementiert. Diese approximiert die Parametrierung der richtigen Aktivierungsfunktion. Das Neuronale Netz wird auf den Datensatz der Stromerzeugung in den Jahren 2004-2018 vom Unternehmen American Electric Power angewandt.

Keywords: SLAF, LNN, Forecasting, Electric Power

Meschede
9. September 2024

Wir geben Impulse



Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt und indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde weder einer anderen Prüfungsbehörde vorgelegt noch veröffentlicht.

Ich weiß, dass die Arbeit in digitalisierter Form daraufhin überprüft werden kann, ob unerlaubte Hilfsmittel verwendet wurden und ob es sich – insgesamt oder in Teilen – um ein Plagiat handelt. Zum Vergleich meiner Arbeit mit existierenden Quellen darf sie in eine Datenbank eingestellt werden und nach der Überprüfung zum Vergleich mit künftig eingehenden Arbeiten dort verbleiben.

Meschede, 9. September 2024.

Vitali Krilov

MatNr: 30329089

Email: curie.marie@fh-swf.de

Corresponding Author

Vladislav Stasenko

MatNr: 87654321

Email: curie.pierre@fh-swf.de

Inhaltsverzeichnis

1 Einleitung	3
2 Methodik	3
2.1 Liquid Time-Constant Neuronal Networks (LTCNNs)	3
2.2 Self Learnable Activation Functions (SLAFs)	5
2.3 Auto Regressive-Moving Average (ARIMA)	5
2.4 PROPHET: Forecasting at a scale	5
3 Datengrundlage und Datenanalyse	5
4 Implementierung	11
4.1 SLAF Integration in LTC Nodes	11
4.2 Training	13
5 Vorhersage Ergebnisse	14
6 Fazit <- Zusammen alles	14

1 Einleitung

Geht es um Vorhersagen von Zeitreihen mit komplexen saisonalen Mustern, so sind wohl die bekanntesten statistischen Modellierungen Autoregressive Integrated Moving Average (ARIMA), Exponential Smoothing oder das Prophet Modell. Im Bereich der Neuronalen Netze ist Long-Short-Term-Memory (LSTM), welches eine spezielle Art von Recurrent Neural Network RNN ist, das bekannte Neuronale Netz für komplexe Zeitreihen. Ein aktuell neuer Ansatz für Vorhersage von komplexen Zeitreihen ist ein weiteres RNN, das Liquid Neuronal Network (LNN). Es geht darum mit weniger Rechenleistung und geringerem Datensatz ein kompakteres Modell zu erzeugen, welches sich z.B. in Echtzeit an die Datenlage anpassen kann. So kann z.B. ein LSTM 100.000 - 1.000.000 Parameter haben, wobei das LNN mit bis zu 10.000 Parametern auskommt und ähnliche Resultate liefert. So ist eine Anbindung an elektronische Messgeräte oder Rasberry PI mit weniger Rechneleistung im Gegensatz zu LSTM möglich. In dieser Arbeit wird ein LNN auf einen Datensatz von der Stromerzeugung vom Unternehmen American Electric Power angewandt. Das Ziel ist es eine Vorhersage für den nächsten Tag "day-ahead" zu treffen. Bei den Neuronalen Netzen steht immer die Frage im Raum, welche Aktivierungsfunktion genutzt werden soll. Als Ergänzung zu dem LNN wird ein self learning activation function (SLAF) Modell implementiert. SLAF soll das vanishing gradient Problem dezimieren, aber auch eine approximierte Aktivierungsfunktion für das LNN finden.

2 Methodik

Eine der bekanntesten Modelle für eine Zeitreihe ist das ARIMA Model. Prophet als ein weiteres Benchmark-Model, welches sich an komplexe Saisonalität anpasst.

2.1 Liquid Time-Constant Neuronal Networks (LTCNNs)

Liquid Neural Networks, insbesondere LTCNNs, sind eine neuartige Form von neuronalen Netzen, die speziell für dynamische, zeitabhängige Daten entwickelt wurden. Im Gegensatz zu klassischen neuronalen Netzen, bei denen die Verbindungen zwischen den Neuronen statisch sind, passen sich die Verbindungen in LTCNs kontinuierlich über die Zeit an. Dies ermöglicht es den Modellen, dynamisch auf unterschiedliche Eingabeströme zu reagieren, was sie besonders geeignet für die Verarbeitung von zeitlichen Daten macht, wie z.B. in Zeitreihenanalysen oder bei Echtzeitanwendungen.

Der Kernmechanismus von LTCNs basiert auf der Verwendung von Differentialgleichungen zur Beschreibung der Neuronenaktivität. Jedes Neuron $x_i(t)$ wird durch eine Differentialgleichung beschrieben, die seinen Zustand in Abhängigkeit von der Zeit und den Eingangsgrößen steuert:

$$\dot{x}_i(t) = -\frac{1}{\tau_i}x_i(t) + \sum_j w_{ij} \cdot \sigma(x_j(t)) + I_i(t)$$

In dieser Gleichung steht:

- $x_i(t)$ für den Zustand des Neurons i zu einem bestimmten Zeitpunkt t ,
- τ_i ist die zeitabhängige Konstante, die die Dynamik des Neurons bestimmt,
- w_{ij} ist das Gewicht der Verbindung zwischen Neuron j und Neuron i ,
- $\sigma(x_j(t))$ ist die Aktivierungsfunktion (z.B. eine nichtlineare Funktion wie die Sigmoid- oder ReLU-Funktion),
- $I_i(t)$ repräsentiert den externen Input, der auf das Neuron einwirkt,
- $\dot{x}_i(t)$ bezeichnet die zeitliche Ableitung des Neuronenzustandes $x_i(t)$.

Die Zeitkonstante τ_i spielt eine zentrale Rolle, da sie festlegt, wie schnell das Neuron auf Änderungen in den Eingangsdaten reagiert. Diese Konstante wird während des Trainings dynamisch angepasst, was den Begriff "Liquid" (flüssig) erklärt: Die Netzwerkkonfiguration ist nicht statisch, sondern verändert sich kontinuierlich im Laufe der Zeit.

Die Architektur eines LTCNs ist relativ simpel aufgebaut: Jedes Neuron besitzt eine zeitabhängige Aktivierung, die durch die oben genannte Differentialgleichung gesteuert wird. Durch die ständige Anpassung der Zeitkonstanten können LTCNs flexibel auf Veränderungen in den Daten reagieren. Diese dynamischen Modelle sind besonders geeignet für komplexe, zeitlich veränderliche Muster.

Besonders für die Verarbeitung von Zeitreihen oder die Analyse von Signalen in Echtzeit sind LTCNs vielversprechend. Durch die flexiblen Zeitkonstanten können sowohl kurzfristige als auch langfristige Abhängigkeiten in den Daten erfasst werden. Dies macht sie besonders nützlich in Anwendungsbereichen wie der Vorhersage von Finanzdaten, der Wetterprognose oder der Analyse biologischer Signale (z.B. EEG- oder EKG-Daten), wo das Erkennen von zeitlichen Mustern entscheidend ist.

Ein wichtiger Durchbruch in der Forschung zu Liquid Neural Networks wurde durch das Paper von Hasani et al. (2021) eingeführt, welches die grundlegende Architektur der Liquid Time Constant Networks beschreibt und deren Funktionsweise detailliert erklärt und demonstriert, dass LTCNs in der Lage sind, mit weniger Parametern eine hohe Effizienz und Genauigkeit bei der Modellierung dynamischer Systeme zu erreichen. Diese Netzwerke zeigten überragende Ergebnisse in der Modellierung von dynamischen Umgebungen und zeitraumgreifenden Aufgaben, was sie von herkömmlichen rekurrenten Netzen unterschied.

In einer späteren Anwendung auf Zeitreihendaten wurde in einem Paper von Gilpin et al. (2021) gezeigt, dass LTCNs signifikante Verbesserungen in der Vorhersagegenauigkeit von Finanzzeitreihen erzielten, indem sie sowohl kurzfristige Schwankungen als auch langfristige Trends erfassten. In einer Studie demonstrierten Pasandi et al. (2022), dass LTCNs zur Analyse von EEG-Signalen eingesetzt werden können, um epileptische Anfälle in Echtzeit mit einer hohen Genauigkeit zu erkennen. In beiden Fällen ließen die Stärken der flüssigen Zustandsübergänge und der dynamischen Zeitkonstanten als wesentliche Vorteile gegenüber traditionellen Netzwerken.

Hasani, R., Lechner, M., Amini, A., Rus, D., & Grosu, R. (2021). "Liquid Time-constant Networks." Nature Machine Intelligence. DOI: 10.1038/s42256-021-00302-4.

2.2 Self Learnable Activation Functions (SLAFs)

Aktivierungsfunktionen haben einen großen Einfluss auf die Modellierung im Bereich der Neuronalen Netze. Dabei reagieren Neuronen auf verschiedene Aktivierungsfunktionen unterschiedlich und beschreiben damit, ob ein Neuron gerade den Status "aktiv" oder "nicht aktiv" hat. Ist ein bestimmter Schwellwert in einer dieser Funktionen überschritten, so ändert das Neuron seinen Status. Die wohl bekanntesten Aktivierungsfunktionen sind sigmoid, relu, tanh und eine lineare Aktivierung. Dabei wurde nach **Ramachandran2017** die swish-Funktion aufgedeckt:

$$f(x) = x * \text{sigmoid}(\beta * x)$$

In der Arbeit von **Ramachandran2017** wurde die swish-Funktion als die beste Aktivierungsfunktion empirisch bestimmt. Dabei wurde vor allem mit der ReLU-Funktion verglichen.

2.3 Auto Regressive-Moving Average (ARIMA)

2.4 PROPHET: Forecasting at a scale

- Lagged Values <- ARIMA Bezug / ACF-PACF <- Vitali
- Train / Testdatensatz <- egal wer, nur paar Sätze

3 Datengrundlage und Datenanalyse

Auch für Neuronale Netze muss Datenanalyse betrieben werden. Es ist wichtig den Datensatz komplett zu durchdringen und die Erkenntnisse aus der Analyse in der Implementierung des Neuronalen Netzen mit zu berücksichtigen. Der rohe Datensatz der Stromerzeugung hat eine stündliche Granularität in der Zeitspanne 01.10.2004 - 03.08.2018 (5054 Tage). Untersucht man den Datensatz auf Äquidistanz, so fallen insgesamt 40 fehlende Beobachtungen in der stündlichen Auflösung. Die Abbildung 1 repräsentiert den rohen Datensatz. Die fehlenden Werte (rot) wurden mit dem letzten vorhandenen Wert ersetzt. Dies ist damit zu begründen, da der Datensatz groß genug ist und es keine drastischen Auswirkungen auf das Neuronale Netz haben wird. Nach der Säuberung der Daten hat der Datensatz korrekterweise 121.296 (24 * 5054) Beobachtungen. Ein positiver oder negativer Trend ist für die Stromerzeugung nicht zu erkennen.

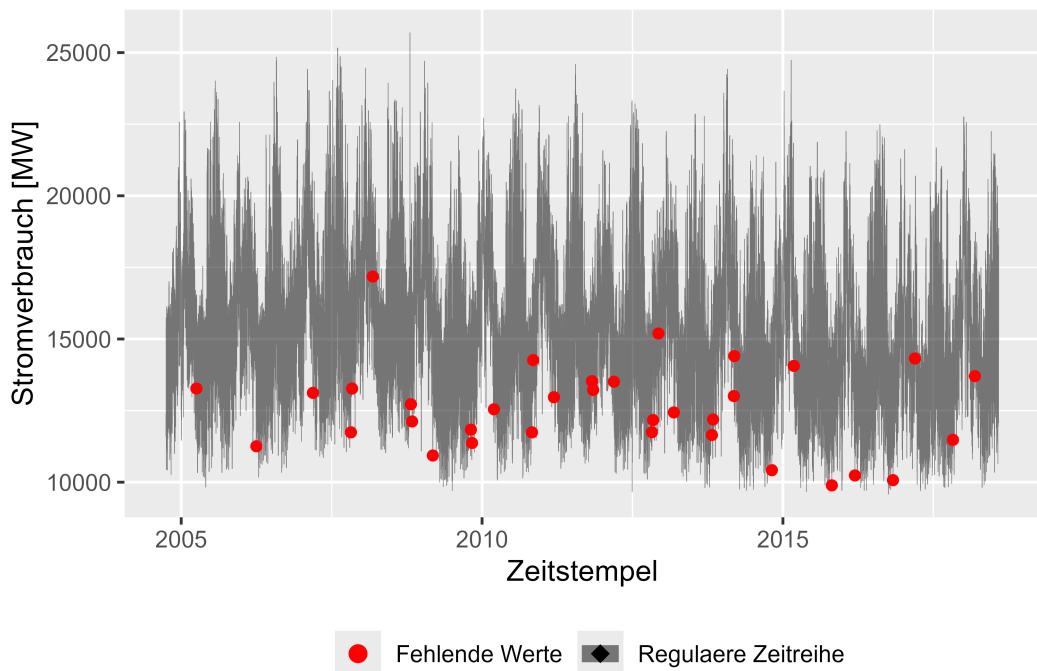


Abbildung 1: Roher Datensatz zur Stromerzeugung (01.10.2004 - 03.08.2018) vom Unternehmen American Electric Power

Wird der Datensatz der Stromerzeugung in die einzelnen Jahre aufgespalten, so ist deutlich ein moustache-Muster (Schnurrbart) zu erkennen. Dieses Muster kann in der Abbildung 2 entnommen werden.

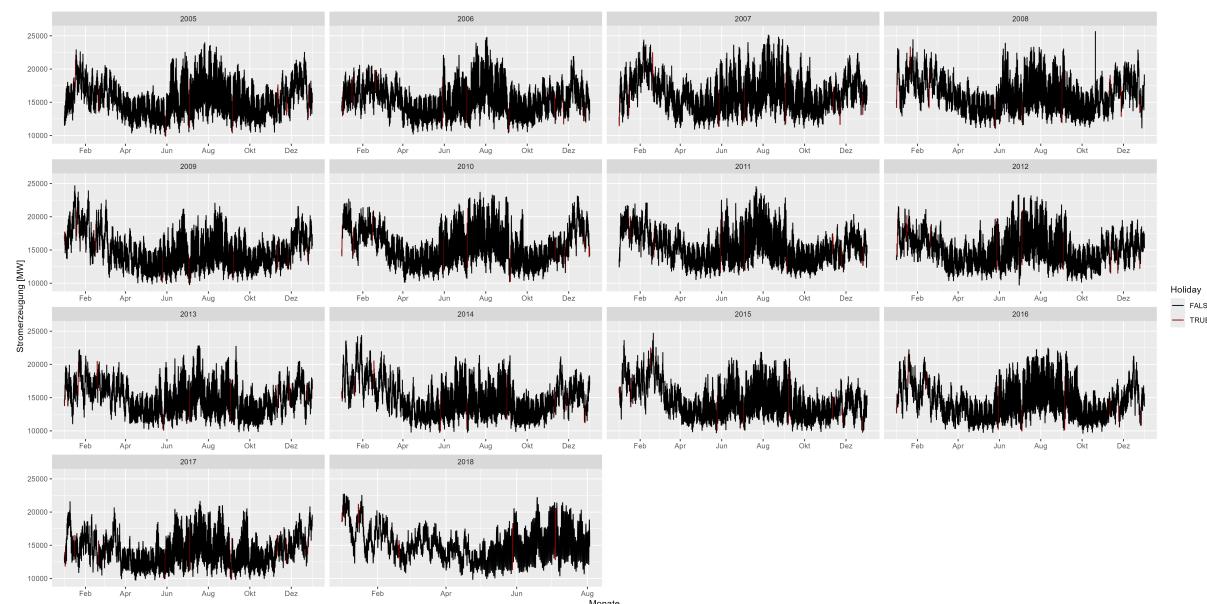


Abbildung 2: Stromerzeugung aufgespalten in einzelne Jahre

Auch die Verteilung der Daten kann eine Rolle bei der Modellierung der Neuronalen Netze spielen. Die Stromerzeugung besitzt ein minimalen Wert von 9581 MW, einen maximalen Wert

von 25.695 MW und einer Durchschnitt von 15.499 MW. Die Abbildung 10 zeigt die Verteilung der Stromerzeugung (Normalverteilt).

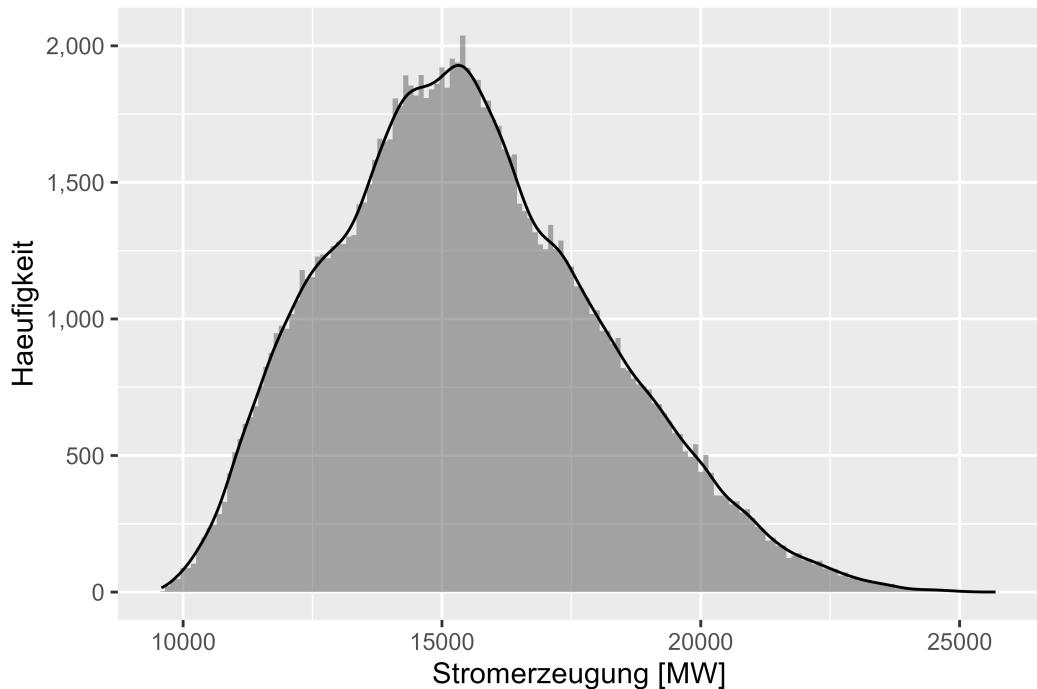


Abbildung 3: Verteilung der Stromerzeugung

Wird der Datensatz pro Stunde aggregiert, so ist auch ein weiteres Muster innerhalb der einzelnen Tage zu erkennen. So hat die Stromerzeugung in der Nacht (~23:00 - 08:00) runtergefahren und in der Tageszeit (~08:00 - 23:00) hochgefahren. Die Abbildung ?? stellt dabei die aggregierte Darstellung für jede Stunde dar.

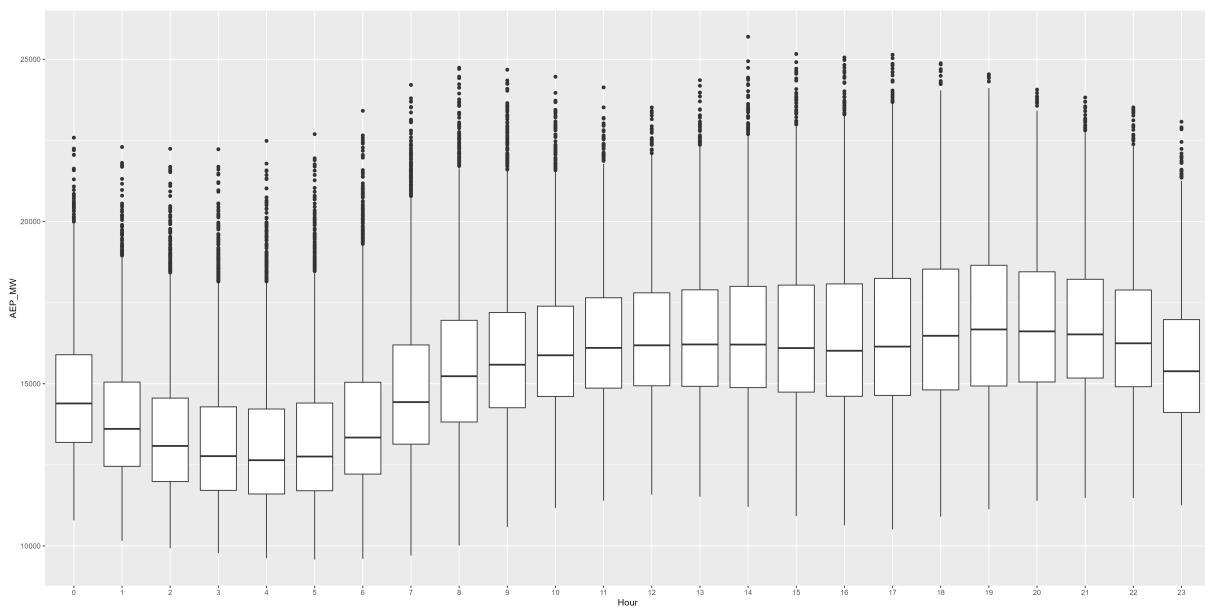


Abbildung 4: Stündlich aggregierte Darstellung der Stromerzeugung

Für ein tieferes Verständnis ist es sinnvoll die Wochentage, Werkstage und die Feiertage zu betrachten. Als erstes die wöchentliche Betrachtung der Stromerzeugung. Die Abbildung 5 stellt dabei die einzelnen Wochentage über die Jahre dar. Anzumerken ist, dass der Samstag und der Sonntag sich von den regulären Wochentagen in der Stromerzeugung unterscheiden. So haben die beiden Tage im Durchschnitt ca. 2000 MW weniger an Stromerzeugung.

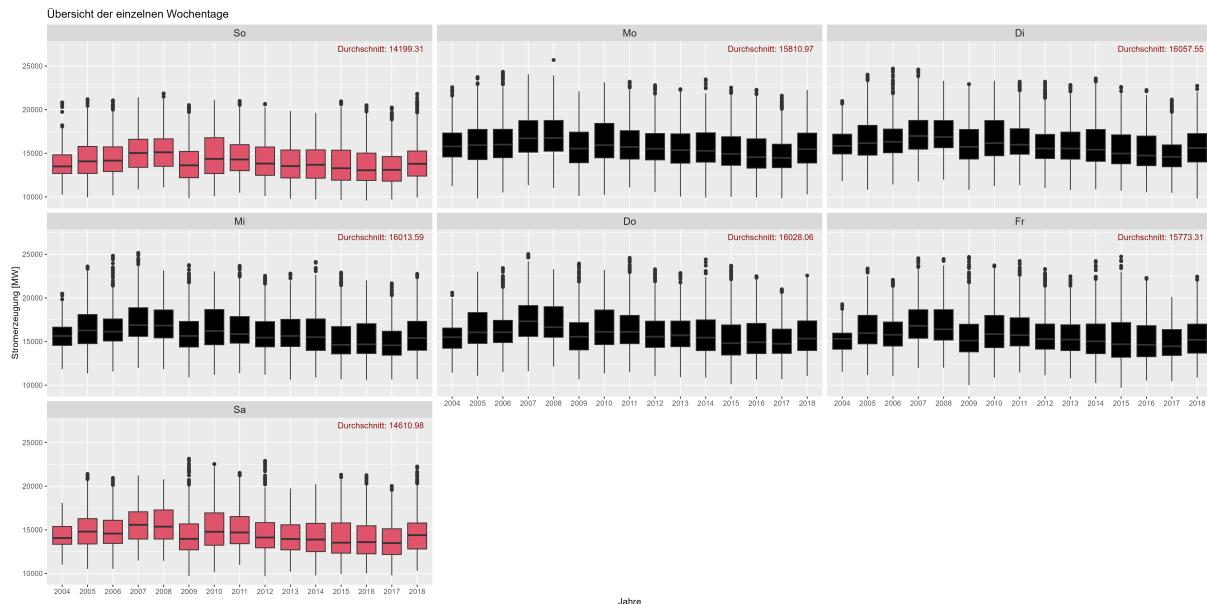


Abbildung 5: Wöchentlich aggregierte Darstellung der Stromerzeugung

Auch die einzelnen Feiertage unterscheiden sich von einem regulären Tag. In der Abbildung 6 sind die einzelnen Feiertage abgebildet. Ähnliche wie ein Wochenende sinkt an den meisten Feiertagen die Stromerzeugung. Bis auf den Martin Luther Kind, Jr. Day, New Year's Day, Veterans Day und den Washington's Birthday verhalten sich die Feiertage wie ein Wochenende unter der Betrachtung vom Durchschnitt.

Vorhersage der Stromerzeugung basierend auf Self Learning Activation Functions und Liquid Neuronal Networks

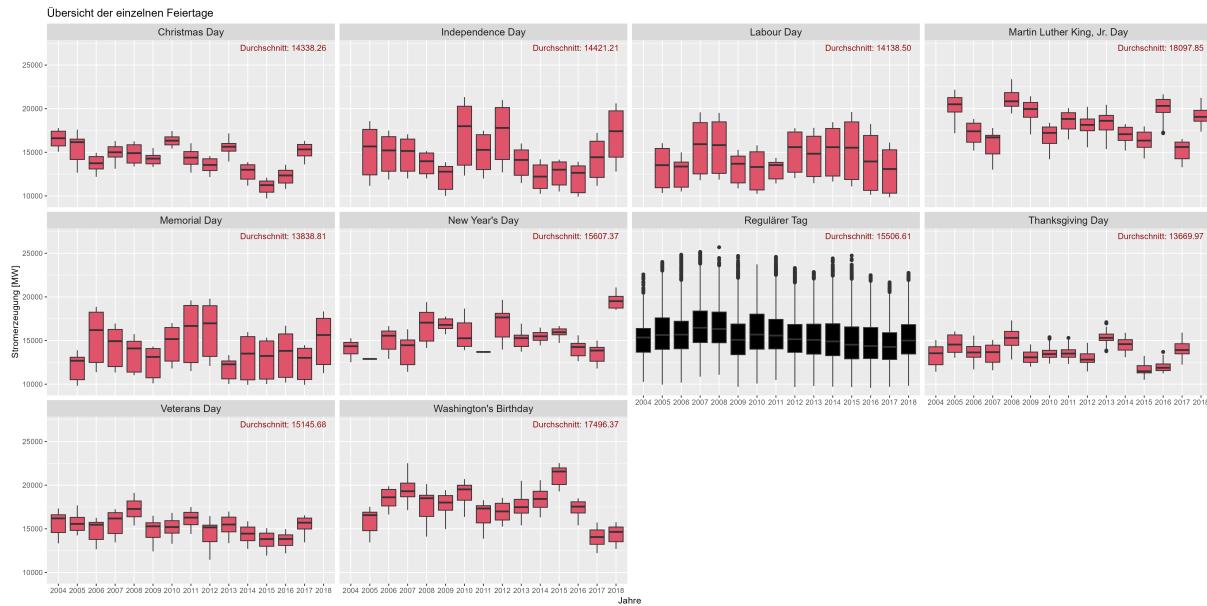


Abbildung 6: Feiertage, aggregierte Darstellung der Stromerzeugung

So kann eine dummy variable erzeugt werden, die die Feiertage, aber auch die Wochenenden beinhaltet. Die Abbildung 7 stellt dabei die Verteilung der Werktag und der nicht Werktag dar. Eine Verschiebung für die nicht Werktag in eine geringere Stromerzeugung.

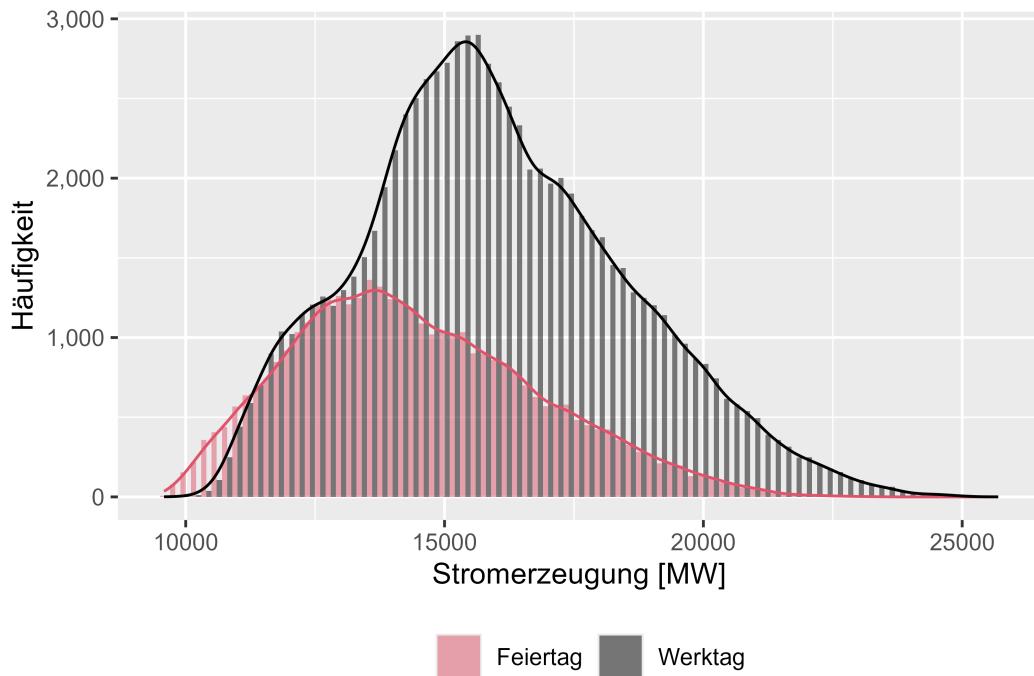


Abbildung 7: Verteilung der Werktag und nicht Werktag

Um den Begriff der komplexen saisonalität zu verdeutlichen zeigt die Abbildung 8 die Autokorrelation. Dabei werden die aktuellen Beobachtungen mit den Vergangenen gegenübergestellt und die Korrelation zwischen diesen ermittelt. So z.B. für lag 12 wird ein shift des Datensatzes

um 12 Zeiteinheiten nach links verschoben. Es wird die Korrelation zwischen dem vorschobenen Datensatz und dem Originalen ermittelt. So in der Abbildung 8 eine komplexe saisonalität, da es sowohl tägliche, wöchentliche aber auch jährliche Muster im Datensatz zu finden sind.

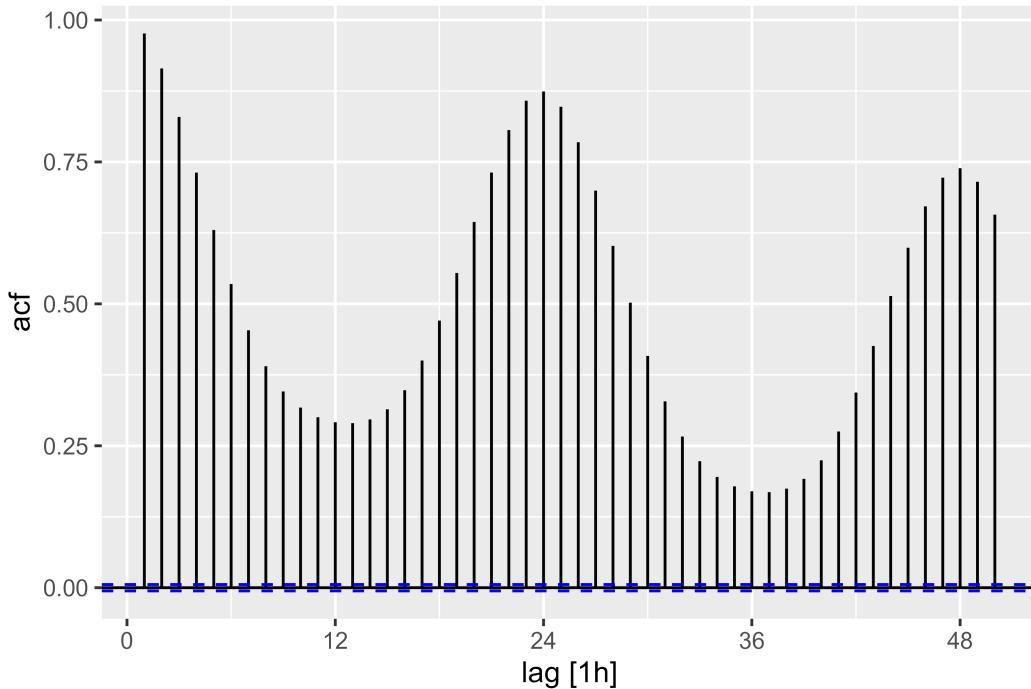


Abbildung 8: Autokorrelation für die Stromerzeugung

Als letzter Punkt in der Datenanalyse ist die Betrachtung der verschiedenen vergangenen Beobachtungen. Durchschnitt der letzten Woche, Minimum vom letzten Tag, Maximum vom letzten Tag und Durchschnitt der letzten zwei Tage. Die Abbildung 9 stellt dabei die Korrelation zwischen den einzelnen Werten dar. Es ist zwischen TRUE (Werktag) und FALSE (kein Werktag) zu unterscheiden. Eine mittelstarke (ca. 0.5) Korrelation für die Stromerzeugung und die unterschiedlichen Variablen.

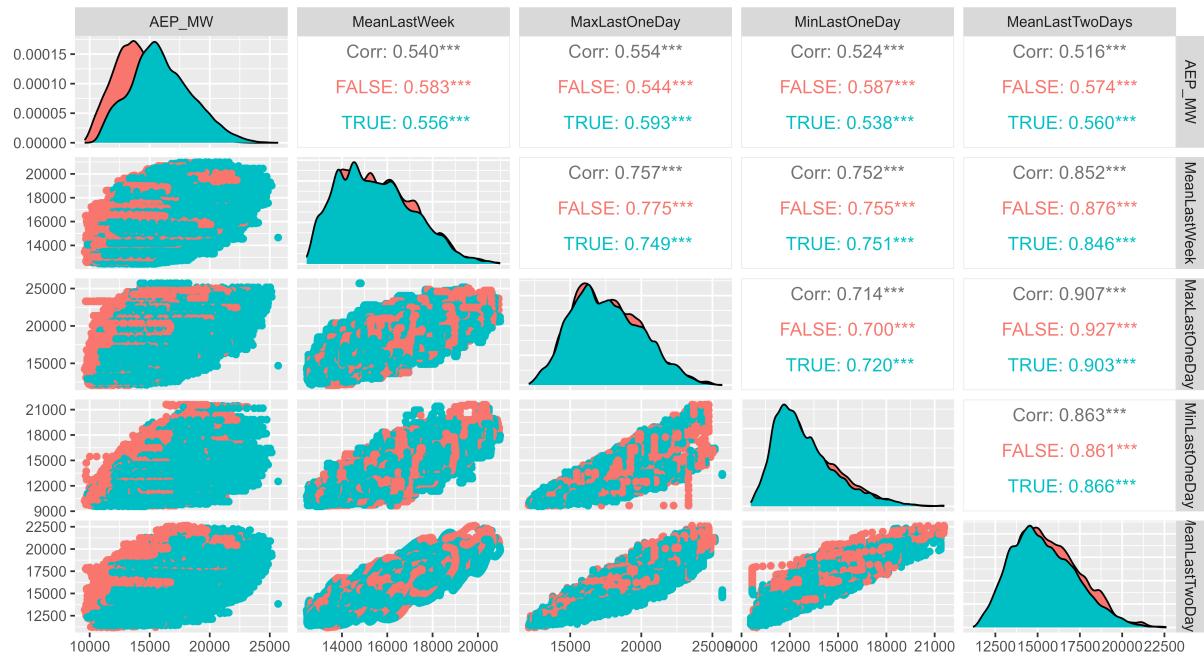


Abbildung 9: Korrelation zwischen den vergangenen Beobachtungen und der Stromerzeugung

- Statistische Methoden <- Vitali
- Feature Engineering <- Vitali (Analytisch) & Vlad (PCA)
- “Calender Effect” <- Vitali

4 Implementierung

4.1 SLAF Integration in LTC Nodes

Wie schon erwähnt, die Swish Funktion wird in diesem Projekt anstatt der klassischen Sigmoid-Aktivierungsfunktion angewendet. Wir erweitern die Funktionalität von “NCPs” Package um das Benutzen von Swish Funktion zu ermöglichen.

Die Aktivierungsfunktion wird ersetzt durch das Einstellen von dem optionalen boolischen Parameter `use_swish_activation` bei der “LTC” Modelle Definition:

```
class LTC(nn.Module):
    def __init__(
        self,
        input_size: int,
        units,
        return_sequences: bool = True,
        batch_first: bool = True,
        mixed_memory: bool = False,
        input_mapping="affine",
        output_mapping="affine",
        ode_unfolds=6,
```

```
    epsilon=1e-8,  
    implicit_param_constraints=True,  
    use_swish_activation=False,  
):  
    ...
```

Der Parameter wird später in jede LTC Knoten übergeben, welcher durch eine weitere Python Klasse repräsentiert ist:

```
class LTCCell(nn.Module):  
    def __init__(  
        self,  
        wiring,  
        in_features=None,  
        input_mapping="affine",  
        output_mapping="affine",  
        ode_unfolds=6,  
        epsilon=1e-8,  
        implicit_param_constraints=False,  
        use_swish_activation=False,  
):  
    ...
```

Falls der Parameter als True gesetzt ist, wird einen neuen Parameter für PyTorch Modelle definiert, welcher ein zufälliger Skalarwert im Zahlenbereich von 0,5 bis 1,5 ist:

```
def add_weight(self, name, init_value, requires_grad=True):  
    param = torch.nn.Parameter(init_value, requires_grad=requires_grad)  
    self.register_parameter(name, param)  
    return param  
  
def _get_init_value(self, shape, param_name):  
    minval, maxval = self._init_ranges[param_name]  
    if minval == maxval:  
        return torch.ones(shape) * minval  
    else:  
        return torch.rand(*shape) * (maxval - minval) + minval  
  
    if self._use_swish_activation:  
        self._params["swish_beta"] = self.add_weight(  
            "swish_beta",  
            init_value=self._get_init_value(  
                (1,), "swish_beta"  
            ),  
        )
```

Der Skalar wird dann später verwendet, um einen Matrizen Produkt zu berechnen:

```
def _sigmoid(self, v_pre, mu, sigma, beta):
    v_pre = torch.unsqueeze(v_pre, -1) # For broadcasting
    mues = v_pre - mu
    x = sigma * mues
    if self._use_swish_activation:
        return x * torch.sigmoid(beta * x)
    return torch.sigmoid(x)
```

Der Parameter ist einer Anpassung bei einer Error Backpropagation fällig, und darüber hinaus, darf von dem initialen Wert während des Trainings unterscheiden.

4.2 Training

Durch die pytorch-Implementierung lässt sich das Training sowohl auf einer CPU, als auch einer GPU ohne große Anpassungen stattfinden. Um das Training stabiler zu gestalten, waren das Hardware von FH-SWF Cluster genutzt.

Das Training lässt sich sowohl mittels cmd-Befehl als auch Jupyter Notebook anstoßen, allein nur die Parameter in config.py sind entscheidend:

```
PATH = "data/csv/AEP_hourly_cleaned.csv"
STATION = "AEP_MW"
FEATURES_LIST = [
    "WorkDay",
    "LastDayWasHolodiayAndNotWeekend",
    "NextDayIsHolidayAndNotWeekend",
    "MeanLastWeek",
    "MeanLastTwoDays",
    "MaxLastOneDay",
    "MinLastOneDay"
]
FEATURES_2_SCALE = [
    "value",
    "MeanLastWeek",
    "MeanLastTwoDays",
    "MaxLastOneDay",
    "MinLastOneDay"
]
YEAR_SHIFT = 365
WEEK_SHIFT = 7
VALUES_PER_DAY = 24
FILTER_DT_FROM = "2014-01-01 00:00:00"
FILTER_DT_TILL = "2017-01-01 00:00:00"
```

```
GRID_SEARCH = True  
  
BATCH_SIZE = 7  
NUM_WORKERS = 128  
NUM_LNN_UNITS = [16, 8, 32]  
  
USE_SWISH_ACTIVATION = [False, True]  
INIT_LR = [0.01, 0.0001]  
NUM_EPOCHS = [10, 50, 100]
```

Außer Datenpfad, Energiestation und Liste relevanten Features, lässt sich auch einen Grid Search mit dem Parameter-Pool einstellen. Die wichtigsten Hyperparameter, die für LNN eine Rolle spielen, sind es Anzahl von Knoten, eine initiale Learning Rate und Anzahl von Epochen. Durch die Konfiguration kann man auch die Schranken für den Trainingsdatensatz definieren, in unserem Fall haben wir uns für 3 Jahren Daten zwischen 2014 und 2017 entschieden. Für die Auswertung nutzen wir die Daten aus danach folgender Woche.

Bevor das eigentliche Training startet, werden die Daten normalisiert auf Bereich zwischen 0 und 1 mit dem `sklearn.preprocessing.MinMaxScaler` Objekt. Für die Evaluation der Ergebnisse bzw. produktives Nutzen kann der Skaler die Daten denormalisieren. Welche Merkmale normalisiert werden müssen, kann auch mittels Konfig eingestellt werden, da die boolische Variablen brauchen keine Skalierung.

Neben der initialen Variablen, die am Start gelesen werden, hinzufügen wir zwei weitere Spalten, die wöchentliche und jährliche Werten Verzögerung der abhängigen Variable einführen. Für Time-Series Daten ist eine normale Praktik, die wir hier auch verfolgen.

Für das unkomplizierte Training wird die `Trainer` Klasse aus der `pytorch_lightning` Bibliothek benutzt, welche einen Data Scientist vom Boiler Code befreit.

- Cluster <- Vlad
- Normalisierung / Skalierung <- Vlad

5 Vorhersage Ergebnisse

- Metriks <- MAE / MAPE (X) / LOSS-FUNCTION <- Vlad
- Plots der Ergebnisse <- Vlad

6 Fazit <- Zusammen alles

- Probleme
- Lösungen
- Zusammenfassung

- Schlussfolgerung und letztes Wort

test (**hasani2021liquid**)

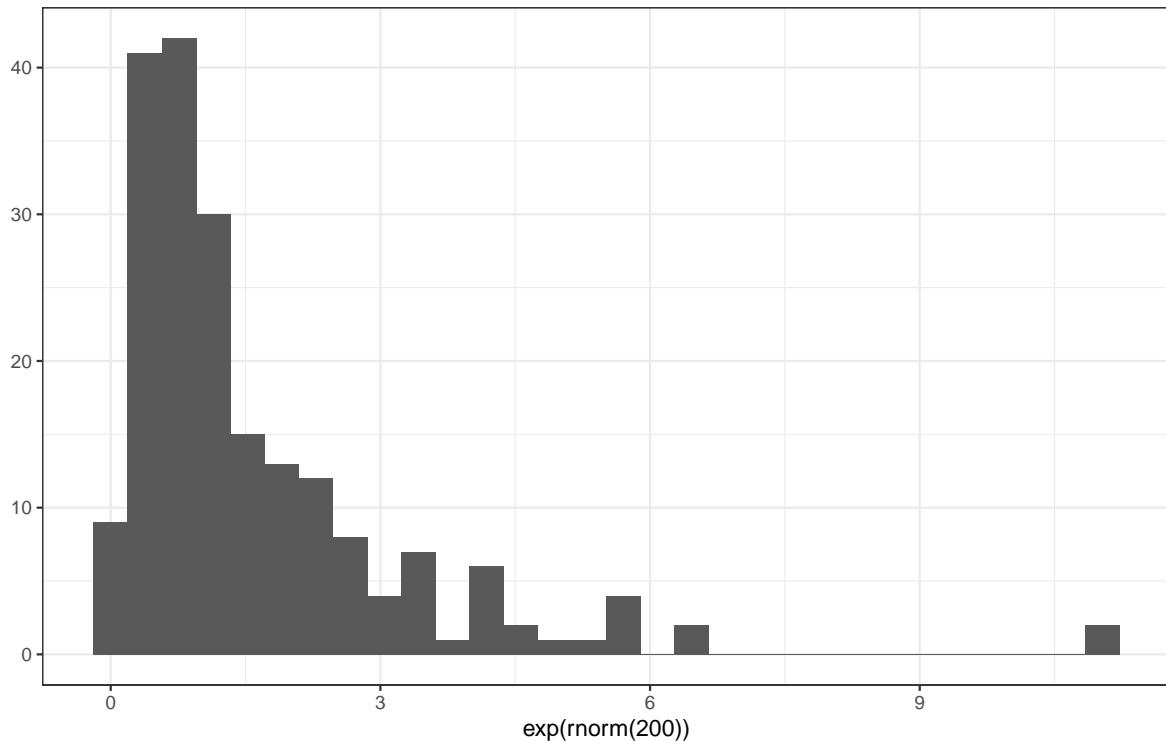


Abbildung 10: Nice histogram.