



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

Stromprognose mittels Self Learning Activation Functions & 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
11. 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, 11. September 2024.

Vitali Krilov

MatNr: 30329089

Email: krilov.vitali@fh-swf.de

Corresponding Author

Vladislav Stasenko

MatNr: 87654321

Email: stasenko.vladislav@fh-swf.de

Inhaltsverzeichnis

1 Einleitung	4
2 Methodik	6
2.1 Liquid Time-Constant Neuronal Networks (LTCNNs)	6
2.2 Self Learnable Activation Functions (SLAFs)	7
2.3 Auto Regressive-Moving Average (ARIMA)	8
2.4 PROPHET: Forecasting at a scale	8
2.5 Metriken	9
3 Datengrundlage und Datenanalyse	10
4 Implementierung	17
4.1 SLAF Integration in LTC Nodes	17
4.2 Feature Engineering	19
4.3 Training	19
5 Vorhersage Ergebnisse	22
6 Fazit	28
7 Anhang	28

Abbildungsverzeichnis

1	Swish-Funktion, ReLU Repräsentation	7
2	Roher Datensatz zur Stromerzeugung (01.10.2004 - 03.08.2018) vom Unternehmen American Electric Power	10
3	Stromerzeugung aufgespalten in einzelne Jahre	11
4	Verteilung der Stromerzeugung	11
5	Stündlich aggregierte Darstellung der Stromerzeugung	12
6	Wöchentlich aggregierte Darstellung der Stromerzeugung	12
7	Feiertage, aggregierte Darstellung der Stromerzeugung	13
8	Verteilung der Werktagen und nicht Werktagen	13
9	Autokorrelation für die Stromerzeugung	14
10	Korrelation zwischen den vergangenen Beobachtungen und der Stromerzeugung	15
11	Nach 10 Epochen mit und ohne SLAF, Ausschnitt für Januar	21
12	Nach 10 Epochen mit und ohne SLAF, ganzes Jahr 2017	21
13	Prophet, Ausschnitt für Januar	22
14	Prophet, Vorhersage gegenüber realen Beobachtungen	23
15	Nach 100 Epochen mit und ohne SLAF, Ausschnitt für Januar	24
16	Nach 100 Epochen mit und ohne SLAF, ganzes Jahr 2017	24
17	Nach 100 Epochen LNN, Vorhersage gegenüber realen Beobachtungen	25
18	Nach 100 Epochen LNN + SLAF, Vorhersage gegenüber realen Beobachtungen	26
19	MAPE Verlust in den einzelnen Schritten (eine Epoche sind mehrere Schritte)	26

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 eine Vorhersage von komplexen Zeitreihen ist ein weiteres RNN, das Liquid Time-Constant Neuronal Network oder kurz 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 Raspberry Pi mit weniger Rechenleistung 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 verstärkt dezimieren, aber auch eine approximierte Aktivierungsfunktion für das LNN finden.

Um den roten Faden besser folgen zu können, werden die einzelnen Kapitel kurz erläutert:

Methodik: Im Kapitel der Methodik wird auf die theoretischen Grundlagen eingegangen. Im ersten Abschnitt wird auf die Liquid Time-Constant Neuronal Networks (LTCNNs) eingegangen. Hier wird beschrieben, was ein LTCNN ist und wie die mathematische Zusammensetzung aussieht. Der Begriff "Luquid" und das Lernverhalten wird in Bezug auf das neuronale Netz erläutert.

Darauf folgt die Erklärung, was eine Self Learnable Activation Function (SLAF) ist. Es wird auf das Konzept hinter SLAFs prägnant eingegangen und erläutert. Der Begriff der swish-Funktion wird eingeführt und beschrieben.

Als Letztes wird vollständigkeitshalber auf das Auto Regressive-Moving Average (ARIMA), das Prophet-Modell und die verwendeten Metriken eingegangen.

Datengrundlage und Datenanalyse: Im Kapitel der Datengrundlage und Datenanalyse wird auf den verwendeten Datensatz eingegangen. Dieser wird in seiner Rohform präsentiert und daraufhin analytisch untersucht. Dabei wird auf Datenkonsistenz (fehlende Werte und doppelte Beobachtungen) eingegangen. Das Verfahren zur Interpolation der fehlenden Daten und der Umgang mit doppelten Beobachtungen wird erläutert. Daraufhin wird der gesäuberte Datensatz in Bezug auf das Jahr, Woche und Stunde untersucht. Gegen Ende wird auf den Effekt der

Feiertage eingegangen und es werden Features generiert, die in das neuronale Netz einfließen sollen.

Implementierung: Im Kapitel der Implementierung wird erläutert, wie das Verfahren und welche Ressourcen zur Implementierung verwendet wurden. So wird stückweise auf den Code eingegangen. Darüber hinaus wird erläutert, wie Feature Engineering betrieben wurde. Wo und wie das Modell trainiert wurde.

Vorhersage Ergebnisse: Im Kapitel der Vorhersage der Ergebnisse wird auf die statistischen Modelle als Referenz und auf die Vorhersage der LNNs eingegangen. Es wird der Unterschied zwischen den statistischen Modellen und den LNNs aufgezeigt. Mittels MAPE wird das beste Modell ermittelt. Gegen Ende wird der Trainingsverlauf dargestellt und die Ergebnisse nochmal zusammengefasst.

Fazit: Im Kapitel Fazit wird die Arbeit rückblickend betrachtet und die wichtigsten Ergebnisse werden zusammengefasst. Es werden verschiedene Forschungsfragen formuliert und ein Ausblick dargestellt.

2 Methodik

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:

$$\frac{dx(t)}{dt} = - \left[\frac{1}{\tau} + f(x(t), I(t), t, \theta) \right] x(t) + f(x(t), I(t), t, \theta) A$$

Hier ist $x(t)$ der verborgene Zustand, τ ist die Zeitkonstante, und $f(x(t), I(t), t, \theta)$ ist eine Funktion, die sich mit der Eingabe $I(t)$, der Zeit t und den Parametern θ entwickelt.

Die Zeitkonstante τ_i spielt eine zentrale Rolle, da sie bestimmt, 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:

$$\tau_{\text{sys}} = \frac{\tau}{1 + \tau f(x(t), I(t), t, \theta)}.$$

Um diese Netze zu trainieren, verwenden die Autoren eine Technik namens Backpropagation through Time (BPTT) in Kombination mit einem benutzerdefinierten Solver, um die Gleichungen des Netzes effizient zu lösen. Dieser Solver stellt ein Gleichgewicht zwischen Stabilität und Geschwindigkeit her. Die Aktualisierungsregel lautet wie folgt:

$$x(t + \Delta t) = x(t) + \Delta t \frac{f(x(t), I(t), t, \theta) A}{1 + \Delta t \left(\frac{1}{\tau} + f(x(t), I(t), t, \theta) \right)}$$

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.

In praktischen Tests übertrafen LTCs traditionelle Modelle wie Long Short-Time Memory Cells (LSTMs) und neuronale Ordinary Differential Equation (ODEs). Ob es um die Erkennung von

Gesten oder die Vorhersage menschlicher Aktivitäten ging, in den meisten Fällen lieferten sie bessere Ergebnisse. Wie andere neuronale Netze haben sie jedoch aufgrund des Problems des verschwindenden Gradienten Schwierigkeiten mit langfristigen Abhängigkeiten. Außerdem benötigen sie im Vergleich zu einfacheren Modellen mehr Rechenressourcen.

Diese Ausgewogenheit von Flexibilität und Ausdrucksstärke macht LTCs zu einer spannenden Entwicklung in der Welt der rekurrenten neuronalen Netze, auch wenn es noch einige Herausforderungen zu bewältigen gibt.

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. In der Arbeit von **Ramachandran2017** wurde die swish-Funktion aufgedeckt:

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

In dieser Arbeit wurde die swish-Funktion als die beste Aktivierungsfunktion alternativ zu der ReLU Funktion empirisch bestimmt. Die swish-Funktion soll dabei glätttere Verläufe ermöglichen, was zu einer stabileren und effizienteren Modelloptimierung führen kann. Der Verlauf der swish-Funktion ist in Abbildung 1 dargestellt.

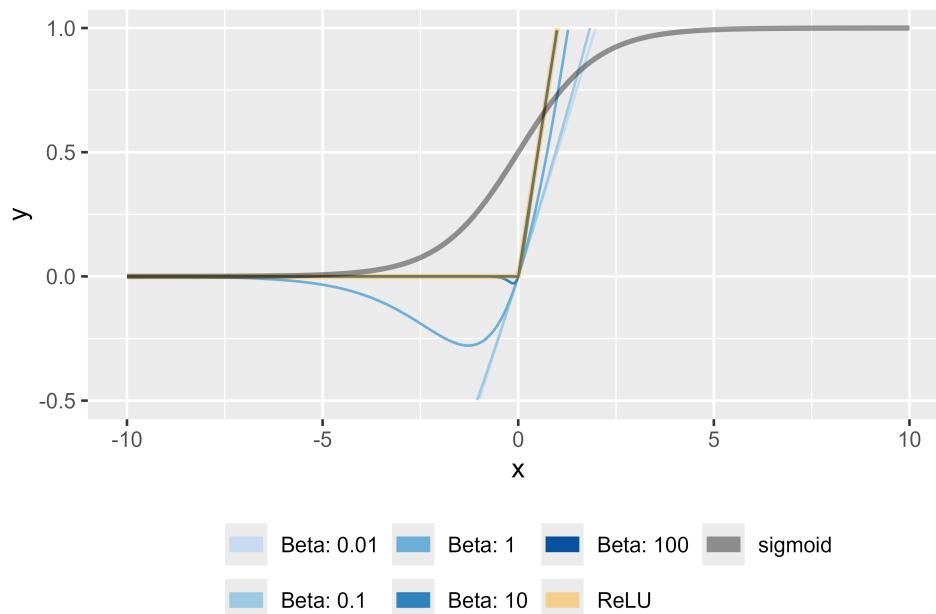


Abbildung 1: Swish-Funktion, ReLU Repräsentation

Geht das $\beta \rightarrow \infty = \text{ReLU}()$ so repräsentiert die Swish-Funktion die aktuelle ReLU-Funktion. Bei schon relativ hohem β erreicht die swish-Funktion approximiert die ReLU-Funktion. Geht das $\beta \rightarrow 0 = x$, so entspricht die Funktion x.

2.3 Auto Regressive-Moving Average (ARIMA)

Bei dem (S)ARIMA Modell handelt es sich um ein additives statistisches Modell, welches sich an komplexe Zeitreihen anpasst. Dabei beschreiben die Thermen PDQ (saisonall) und pdq (nicht saisonall) das Modell. Folgende Backshift Notation beschreibt das Modell:

$$\Phi(B)\phi(B)(1 - B)^d(1 - B)^Dy_t = c + \Theta(B)\theta(B)\epsilon_t$$

Dabei wird die Zeitreihe d-mal und D-mal differenziert. Die Thermen p, d, P, Q werden mittels des geringsten AICc ermittelt. Dieses Modell gilt als eines der Grundlagenmodelle zum Vergleich in dieser Arbeit. [HyndmanR](#)

2.4 PROPHET: Forecasting at a scale

Das Prophet-Model wurde von Facebook [taylor2018forecasting](#) entwickelt und findet Anwendung beim Vorhersagen von Zeitreihen mit komplexer Saisonalität. Speziell designt, um Feiertag-Muster zu erkennen. Dabei handelt es sich wie bei dem ARIMA Modell um ein additives Modell. Im Vergleich zum ARIMA Modell ist das Prophet-Model vor allem im Training deutlich schneller. Hinzu kommt, dass dieses Modell im Gegensatz zum ARIMA Modell den Effekt der Feiertage deutlich besser erfassen kann. Folgende Formel beschreibt das Modell:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

$g(t)$: Der schrittweise lineare Trend

$s(t)$: Die saisonalen Muster

$h(t)$: Feiertag-Mustererfassung

ϵ_t : Der Fehler-Term (Rauschen)

2.5 Metriken

Als ein Auswahlkriterium für eine Vorhersage spielen Metriken (in dieser Arbeit) wie Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE) und Root Mean Squared Error (RMSE) eine Rolle. Folgend werden die einzelnen Metriken kurz beschrieben.

MAE beschreibt die durchschnittliche absolute Abweichung zwischen den vorhergesagten Werten \hat{y}_i und den tatsächlichen Werten y_i . Folgende Formel wird dabei verwendet:

$$MAE = \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{n}$$

MAPE beschreibt die durchschnittliche prozentuale Abweichung zwischen den vorhergesagten Werten \hat{y}_i und den tatsächlichen Werten y_i . Folgende Formel wird dabei verwendet.

$$MAPE = 100 \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

RMSE beschreibt die Durchschnittliche quadratische Abweichung zwischen den vorhergesagten Werten \hat{y}_i und den tatsächlichen Werten y_i . Folgende Formel wird dabei verwendet.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

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 Netzes mitzuberücksichtigen. Der rohe Datensatz der Stromerzeugung hat eine stündliche Granularität in der Zeitspanne 01.10.2004 - 03.08.2018 (5054 Tage). Wird der Datensatz auf Äquidistanz untersucht, so fallen insgesamt 40 fehlende Beobachtungen in der stündlichen Auflösung. Die Abbildung 2 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. Die ganze Analyse und Datensäuberung wurde in dieser Arbeit mit R gemacht.

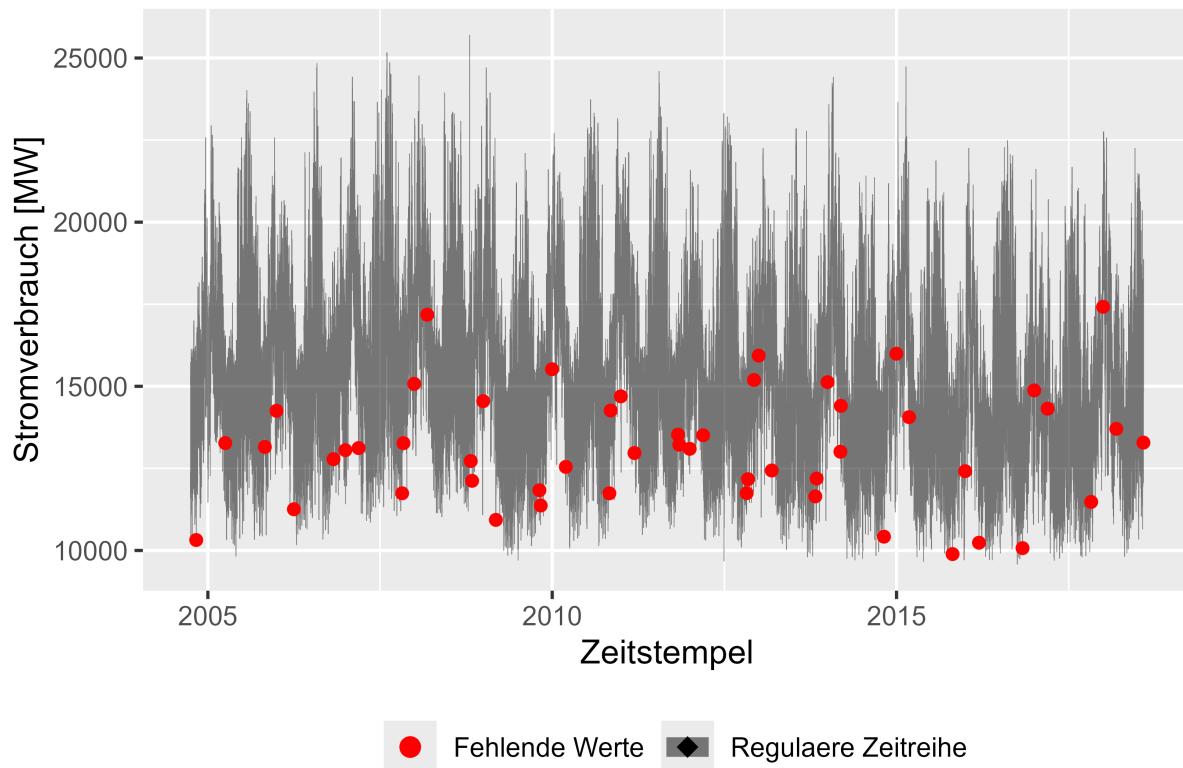


Abbildung 2: Roher Datensatz zur Stromerzeugung (01.10.2004 - 03.08.2018) vom Unternehmen American Electric Power

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 3 entnommen werden.

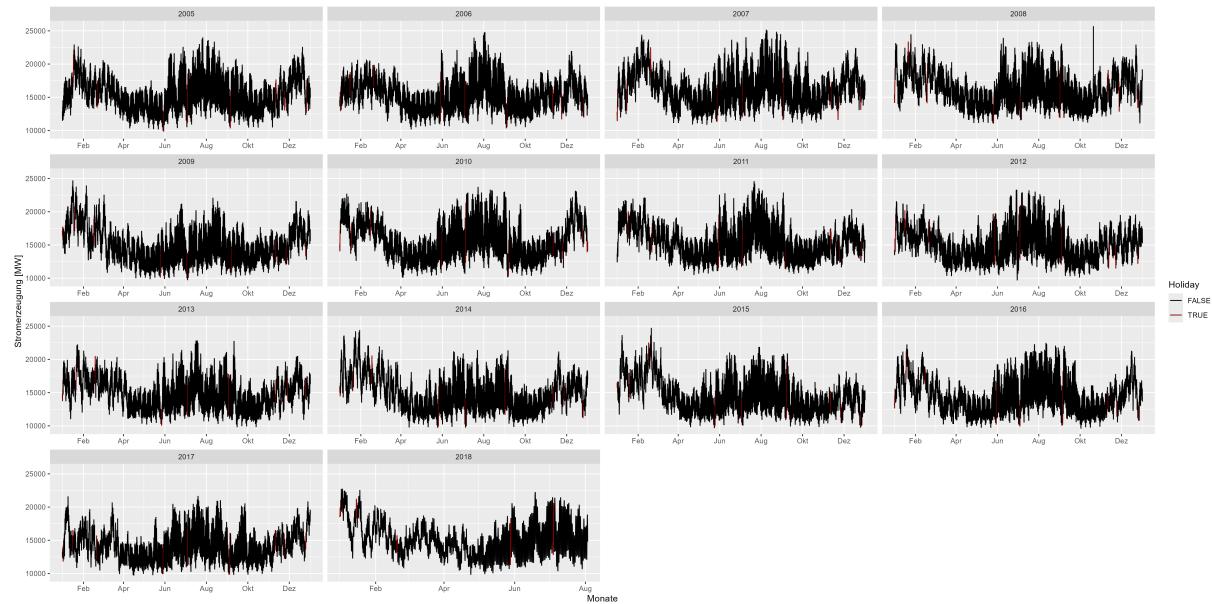


Abbildung 3: Stromerzeugung aufgespalten in einzelne Jahre

Auch die Verteilung der Daten kann eine Rolle bei der Modellierung der neuronalen Netze spielen. Die Stromerzeugung besitzt einen minimalen Wert von 9581 MW, einen maximalen Wert von 25.695 MW und einen Durchschnitt von 15.499 MW. Die Abbildung 4 zeigt die Verteilung der Stromerzeugung (normalverteilt).

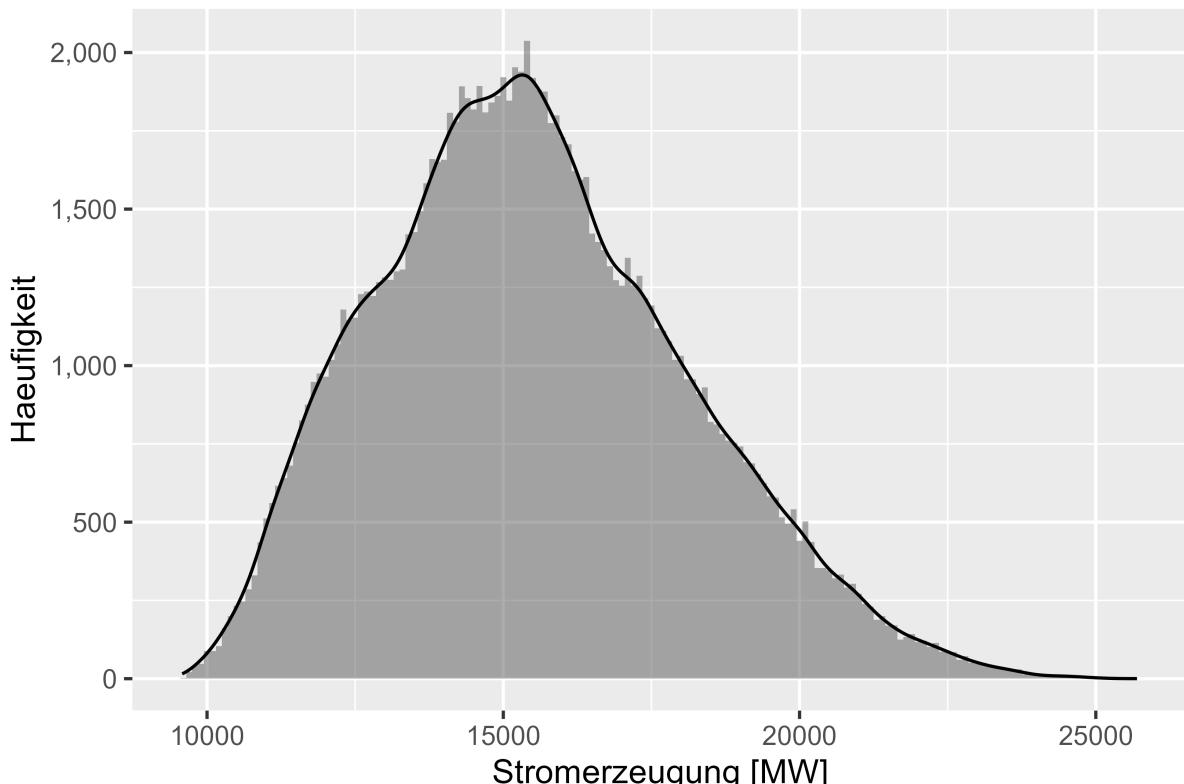


Abbildung 4: Verteilung der Stromerzeugung

Wird der Datensatz pro Stunde aggregiert, so ist auch ein weiteres Muster innerhalb der einzelnen Tage zu erkennen. So wird die Stromerzeugung in der Nacht (~23:00 - 08:00) heruntergefahren und in der Tageszeit (~08:00 - 23:00) hochgefahren. Die Abbildung 5 stellt dabei die aggregierte Darstellung für jede Stunde dar.

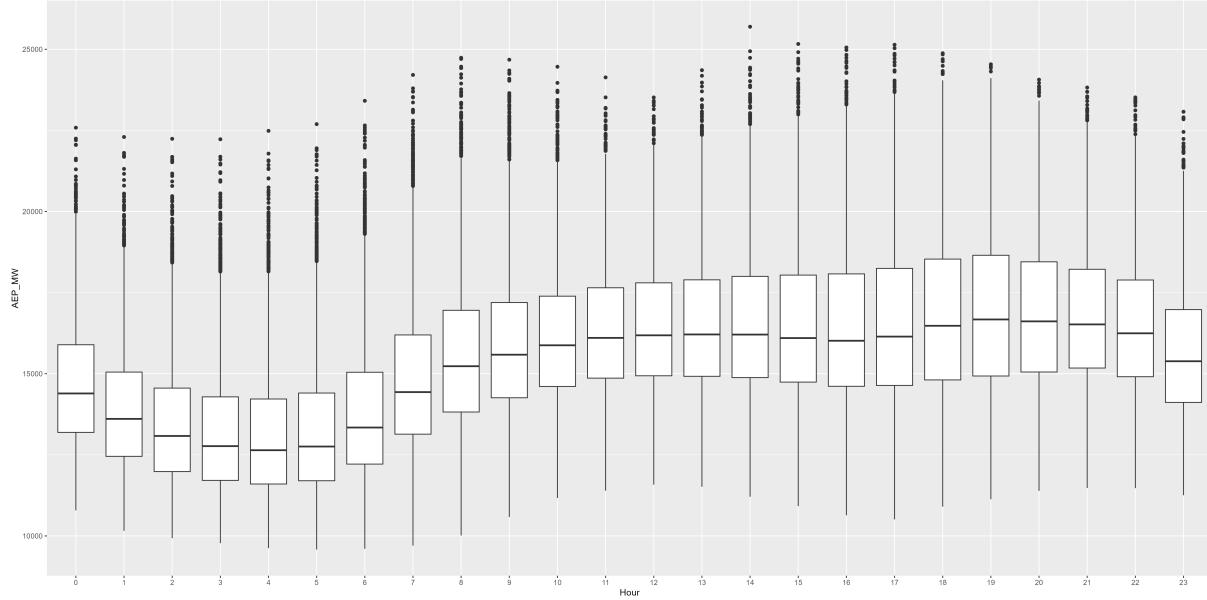


Abbildung 5: Stündlich aggregierte Darstellung der Stromerzeugung

Für ein tieferes Verständnis ist es sinnvoll, die Wochentage, Werkstage und die Feiertage zu betrachten. Als Erstes die wöchentliche Betrachtung der Stromerzeugung. Die Abbildung 6 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.

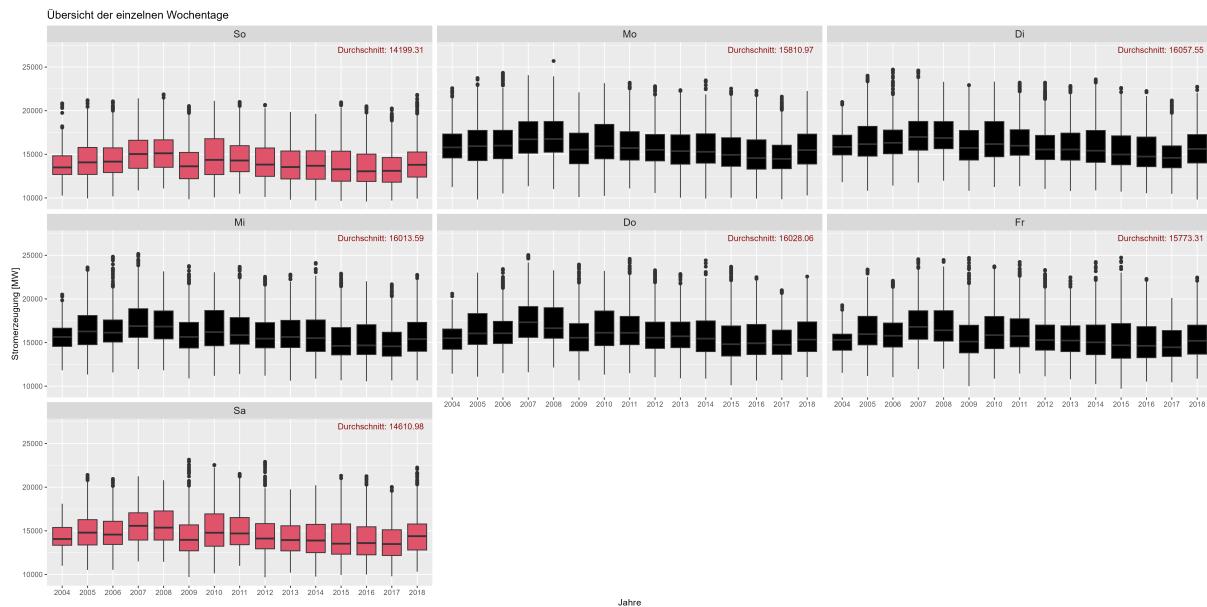


Abbildung 6: Wöchentlich aggregierte Darstellung der Stromerzeugung

Auch die einzelnen Feiertage unterscheiden sich von einem regulären Tag. In der Abbildung 7 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.

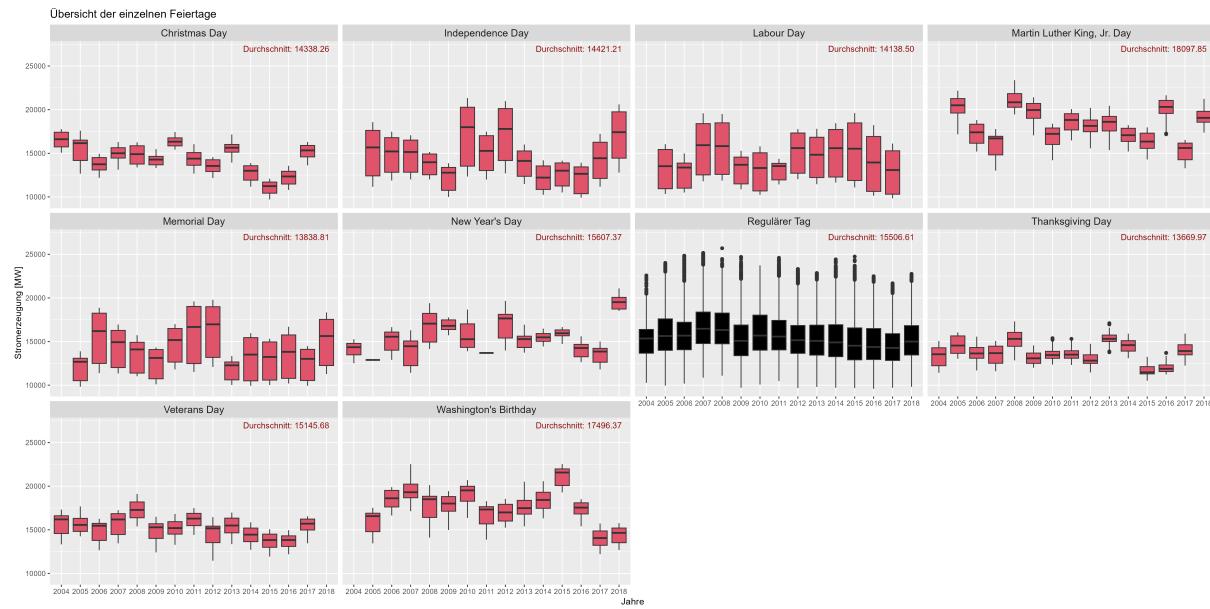


Abbildung 7: Feiertage, aggregierte Darstellung der Stromerzeugung

So kann eine dummy variable erzeugt werden, die die Feiertage, aber auch die Wochenenden beinhaltet. Die Abbildung 8 stellt dabei die Verteilung der Werktag und der nicht Werktag dar. Eine Verschiebung für die nicht Werktag in eine geringere Stromerzeugung ist deutlich sichtbar.

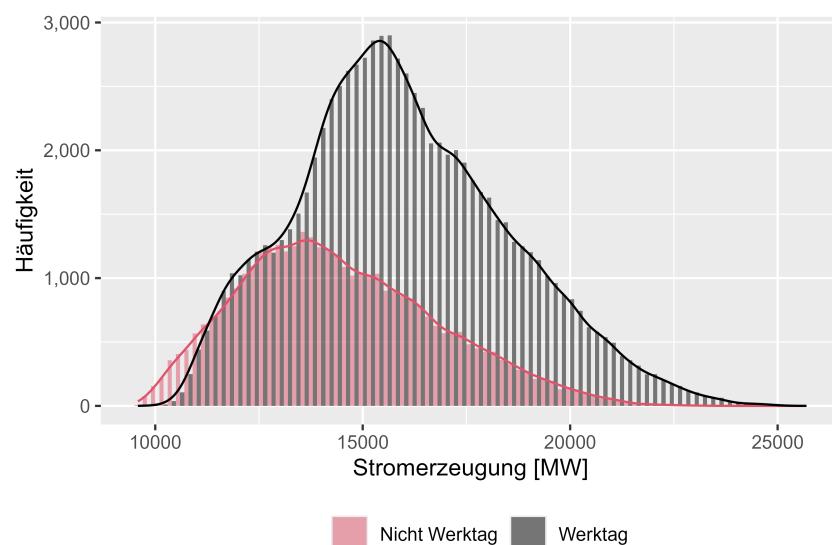


Abbildung 8: Verteilung der Werktag und nicht Werktag

Um den Begriff der komplexen Saisonalität zu verdeutlichen, zeigt die Abbildung 9 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 verschobenen Datensatz und dem Originalen ermittelt. So in der Abbildung 9 ist eine komplexe Saisonalität abgebildet, da es sowohl tägliche, wöchentliche als auch jährliche Muster im Datensatz zu finden sind.

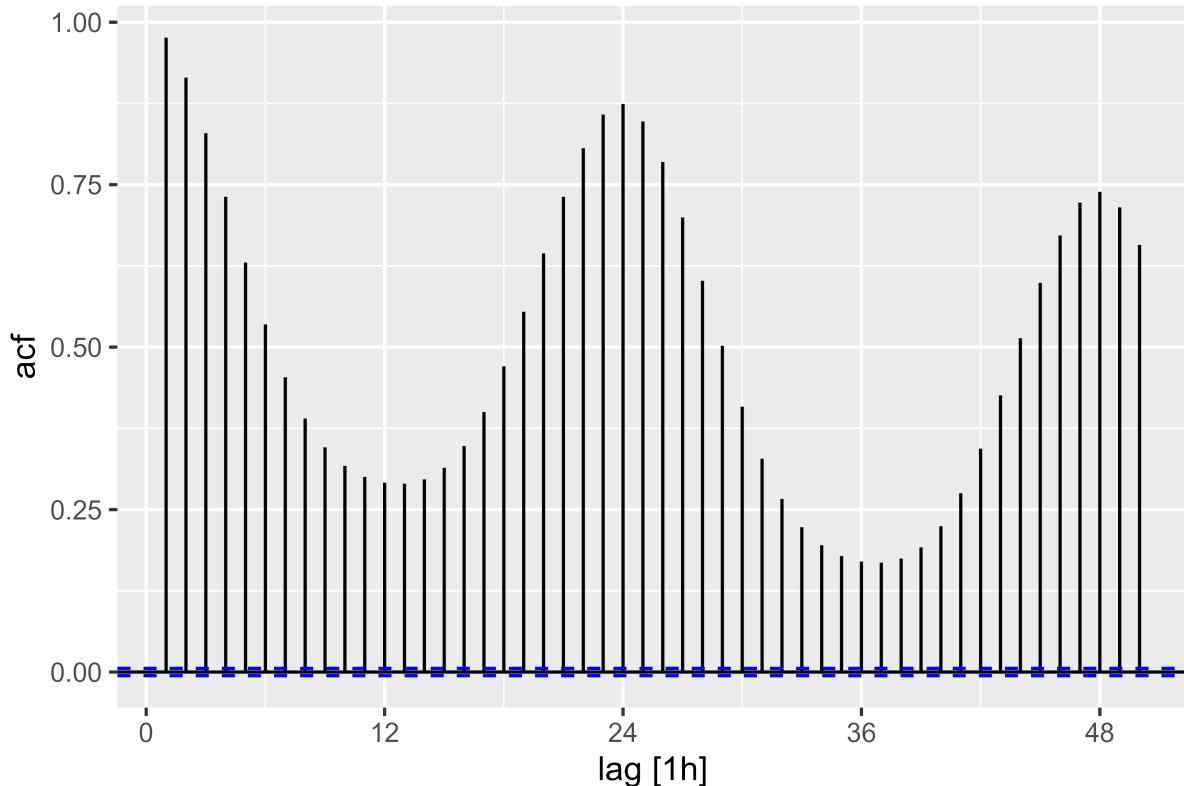


Abbildung 9: Autokorrelation für die Stromerzeugung

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 10 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.

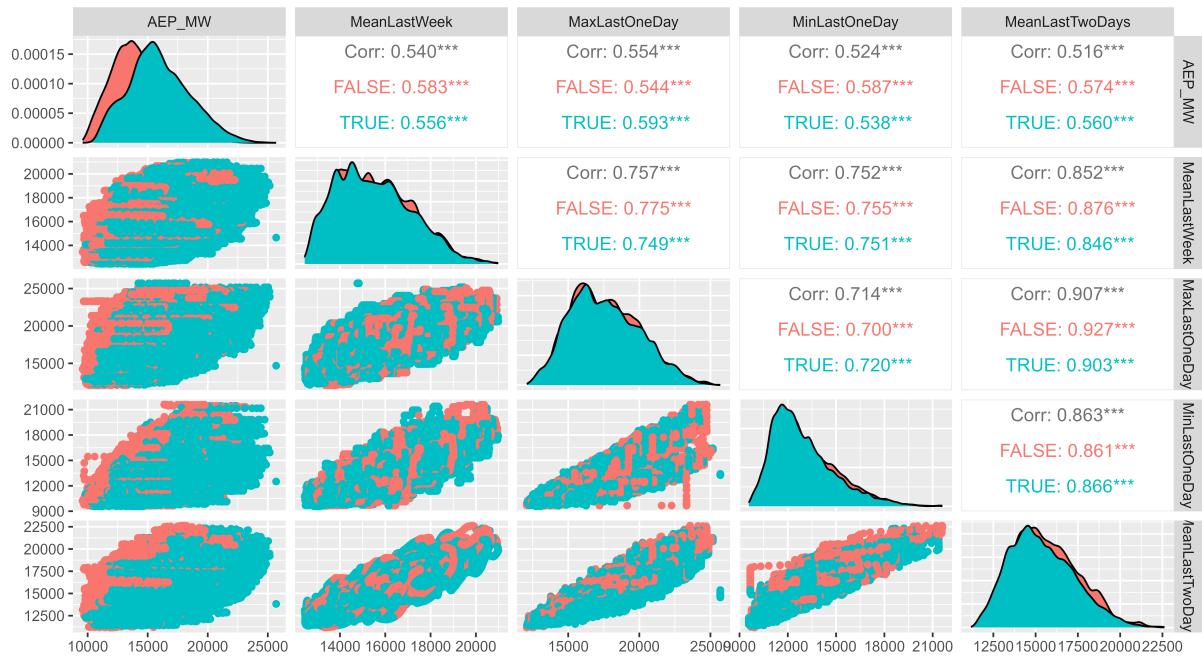


Abbildung 10: Korrelation zwischen den vergangenen Beobachtungen und der Stromerzeugung

Damit wurde der Datensatz nach der Analyse angepasst und kann folgend beschrieben werden:

- Datetime: Zeitstempel
- AEP_MW: Stromerzeugung in Megawatt
- Weekday: Wochentag (Mo, Di, Mi, Do, Fr, Sa, So)
- Date: Datum (YYYY-MM-DD)
- Year: Jahr
- Week: Kalenderwoche
- Hour: Stunde
- Month: Monat
- WorkDay: Ob es ein Arbeitstag ist (TRUE/FALSE)
- Mo: Montag (TRUE/FALSE)
- Di: Dienstag (TRUE/FALSE)
- Mi: Mittwoch (TRUE/FALSE)
- Do: Donnerstag (TRUE/FALSE)
- Fr: Freitag (TRUE/FALSE)
- Sa: Samstag (TRUE/FALSE)
- So: Sonntag (TRUE/FALSE)
- Holiday: Ob es ein Feiertag ist (TRUE/FALSE)
- LastDayWasNotWorkDay: Ob der letzte Tag kein Arbeitstag war (TRUE/FALSE)
- LastDayWasNotWorkDayAndNowWorkDay: Ob der letzte Tag kein Arbeitstag war und der heutige Arbeitstag ist (TRUE/FALSE)
- NextDayIsNotWorkDayAndNowWorkDay: Ob der nächste Tag kein Arbeitstag ist und der heutige Arbeitstag ist (TRUE/FALSE)

- LastDayWasHolidayAndNotWeekend: Ob der letzte Tag ein Feiertag war und kein Wochenende (TRUE/FALSE)
- NextDayIsHolidayAndNotWeekend: Ob der nächste Tag ein Feiertag ist und kein Wochenende (TRUE/FALSE)
- HolidayName: Name des Feiertags
- MeanLastWeek: Durchschnitt der letzten Woche
- MeanLastTwoDays: Durchschnitt der letzten zwei Tage
- MaxLastOneDay: Maximum des letzten Tages
- MinLastOneDay: Minimum des letzten Tages

Anmerkung: Es wurden weitere Feature erzeugt, die z.B. den Kalender-Effekt besser beschreiben sollen. So kann es sein, dass nach einem Feiertag die Menschen Urlaub nehmen und dadurch die Stromerzeugung sinkt. So wird auch untersucht, ob der letzte Tag z.B. ein Feiertag war und nun ein Werktag (z.B. das Feature "LastDayWasNotWorkDayAndNowWorkDay").

4 Implementierung

4.1 SLAF Integration in LTC Nodes

Wie schon erwähnt, die swish-Funktion wird in diesem Projekt anstatt der klassischen Sigmoid-Aktivierungsfunktion angewandt. Es wird die Funktionalität vom “NCPs” Package um die swish-Funktion erweitert.

Die Aktivierungsfunktion wird durch das Einstellen von dem optionalen booleschen Parameter `use_swish_activation` ersetzt. Bei der “LTC” Modelle Definition sieht der Code folgend aus:

```
1 class LTC(nn.Module):
2     def __init__(
3         self,
4         input_size: int,
5         units,
6         return_sequences: bool = True,
7         batch_first: bool = True,
8         mixed_memory: bool = False,
9         input_mapping="affine",
10        output_mapping="affine",
11        ode_unfolds=6,
12        epsilon=1e-8,
13        implicit_param_constraints=True,
14        use_swish_activation=False,
15    ):
16        ...
```

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

```
1 class LTCCell(nn.Module):
2     def __init__(
3         self,
4         wiring,
5         in_features=None,
6         input_mapping="affine",
7         output_mapping="affine",
8         ode_unfolds=6,
9         epsilon=1e-8,
10        implicit_param_constraints=False,
11        use_swish_activation=False,
12    ):
13        ...
```

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

```
1 def add_weight(self, name, init_value, requires_grad=True):
2     param = torch.nn.Parameter(init_value, requires_grad=requires_grad)
3     self.register_parameter(name, param)
```

```
4     return param
5
6 def _get_init_value(self, shape, param_name):
7     minval, maxval = self._init_ranges[param_name]
8     if minval == maxval:
9         return torch.ones(shape) * minval
10    else:
11        return torch.rand(*shape) * (maxval - minval) + minval
12
13 if self._use_swish_activation:
14     self._params["swish_beta"] = self.add_weight(
15         "swish_beta",
16         init_value=self._get_init_value(
17             (1,), "swish_beta"
18         ),
19     )
```

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

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

Der Parameter (swish beta) wird während des Trainings kontinuierlich verändert. Das passiert wenn die Fehler ins Netz zurückgespielt werden.

4.2 Feature Engineering

Neben den initialen Variablen, die aus der Datenanalyse erzeugt wurden, werden zwei weitere Spalten hinzugefügt. Nämlich die wöchentliche und jährliche Verzögerung der abhängigen Variable. Für Time-Series Daten ist es eine normale Praktik, die wir hier auch verfolgen. Somit ergeben sich ca. 30 Features, die einen möglichen Einfluss auf das Modell haben können.

- $x: x_{t-24}$ Verschiebung um einen Tag zurück.
- $x_shifted_week: x_{t-24*7}$ Verschiebung um eine Woche zurück.
- $x_shifted_year: x_{t-24*365}$ Verschiebung um ein Jahr zurück.

Wegen dieser großen Dimensionalität des Datensatzes möchten wir mithilfe der PCA-Analyse zeigen, wie viele Variablen benötigt werden, um die Varianz der Daten zu erklären. Folgend eine Tabelle, wie viele Komponenten gebraucht werden, um die Varianz des Datensatzes zu beschreiben:

Principal Component	Merkmal	Kummulierte Varianz
PC1	x	0.61522145
PC2	x_shifted_week	0.80951859
PC3	x_shifted_year	0.86591735
PC4	MaxLastOneDay	0.90337726
PC5	WorkDay	0.9391192
PC6	MeanLastWeek	0.96688936
PC7	MinLastOneDay	0.98807356
PC8	MeanLastTwoDays	0.99384472
PC9	LastDayWasHolidayAndNotWeekend	0.99713582
PC10	NextDayIsHolidayAndNotWeekend	1.0

Tabelle 1: PCA: Beschreibung der Varianz

Die Komponenten sind in der Reihenfolge nach deren Wichtigkeit (Feature Importance) platziert. Jedoch zeigt die Tabelle, dass die numerischen Variablen höher platziert sind als die booleschen Merkmale, weil diese stärkere Korrelation mit abhängigen Variablen aufweisen.

Trotzdem hat sich im Verlauf des Trainings herausgestellt, dass die verzögerten Werte eher das Modell verwirren als fördern. Aufgrund des enormen Zeitaufwands, um alle Möglichkeiten der ca. 30 Features durchzugehen (Grid-Search für Features) wurde in dieser Arbeit der Ansatz der Datenanalyse verfolgt. Dabei wurde versucht, logisch die Variablen herauszupicken und ins Modell hereinfließen zu lassen. Das Ganze untermauert mit der Datenanalyse. Im folgenden Unterkapitel beschreibt die FEATURES_LIST die ausgewählten Features. Dabei war es auch die Idee für den Anfang einen etwas einfacheren Ansatz zu untersuchen, um die Trainingszeit zu minimieren.

4.3 Training

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

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

```
1 PATH = "data/csv/AEP_hourly_cleaned.csv"
2 STATION = "AEP_MW"
3 FEATURES_LIST = [
4     "WorkDay",
5     "LastDayWasHolodiayAndNotWeekend",
6     "NextDayIsHolidayAndNotWeekend",
7     "MeanLastWeek",
8     "MeanLastTwoDays",
9     "MaxLastOneDay",
10    "MinLastOneDay"
11 ]
12 FEATURES_2_SCALE = [
13     "value",
14     "MeanLastWeek",
15     "MeanLastTwoDays",
16     "MaxLastOneDay",
17     "MinLastOneDay"
18 ]
19
20 YEAR_SHIFT = 365
21 WEEK_SHIFT = 7
22 VALUES_PER_DAY = 24
23 FILTER_DT_FROM = "2014-01-01 00:00:00"
24 FILTER_DT_TILL = "2017-01-01 00:00:00"
25
26 GRID_SEARCH = True
27
28 BATCH_SIZE = 7
29 NUM_WORKERS = 128
30 NUM_LNN_UNITS = [16, 8, 32]
31
32 USE_SWISH_ACTIVATION = [False, True]
33 INIT_LR = [0.01, 0.0001]
34 NUM_EPOCHS = [10]
```

Außer Datenpfad, Energiestation und Liste relevanter Features lässt sich auch einen Grid Search mit dem Parameter-Pool einstellen. Die wichtigsten Hyperparameter, die für LNN eine Rolle spielen, sind die Anzahl von Knoten, eine initiale Learning-Rate und die Anzahl von Epochen. In der Konfiguration werden auch die Schranken für den Trainingsdatensatz definiert. In diesem Fall war es sinnvoll, die letzten 3 Jahre zum Vorhersagejahr zu nehmen (hier 2014 und 2017). Dies ist damit zu begründen, dass das Modell sowohl das jährliche Muster, wöchentliche Muster als auch das tägliche Muster lernen soll.

Bevor das eigentliche Training startet, werden die Daten im Bereich zwischen 0 und 1 mit dem `sklearn.preprocessing.MinMaxScaler` Objekt normalisiert. Für die Evaluation der Ergebnisse bzw. produktives Nutzen kann der Scaler die Daten denormalisieren. Welche Features normalisiert werden, wird mittels `FEATURES_2_SCALE` eingestellt, da z.B. die booleschen Variablen keine Skalierung brauchen.

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

Der Grid Search für die Parameter (nicht Features) wurde auf 10 Epochen pro Parameterkombination begrenzt, um das Potenzial von den Modellen analytisch zur 10-ten Epoche zu betrachten. Dafür wurden insgesamt 36 Modelle miteinander verglichen. Zuerst wurde für jeden Kandidaten ein Mean Average Percentage Error berechnet und eine grafische Darstellung der Vorhersagen als ein Anhaltspunkt genommen. Die meist vielversprechenden Parameterkombinationen wurden dann für längeres Training (100 Epochen) ausgewählt.

Nach 10 Epochen hatten die besten Modelle einen MAPE Loss zwischen 19% und 25% gezeigt:

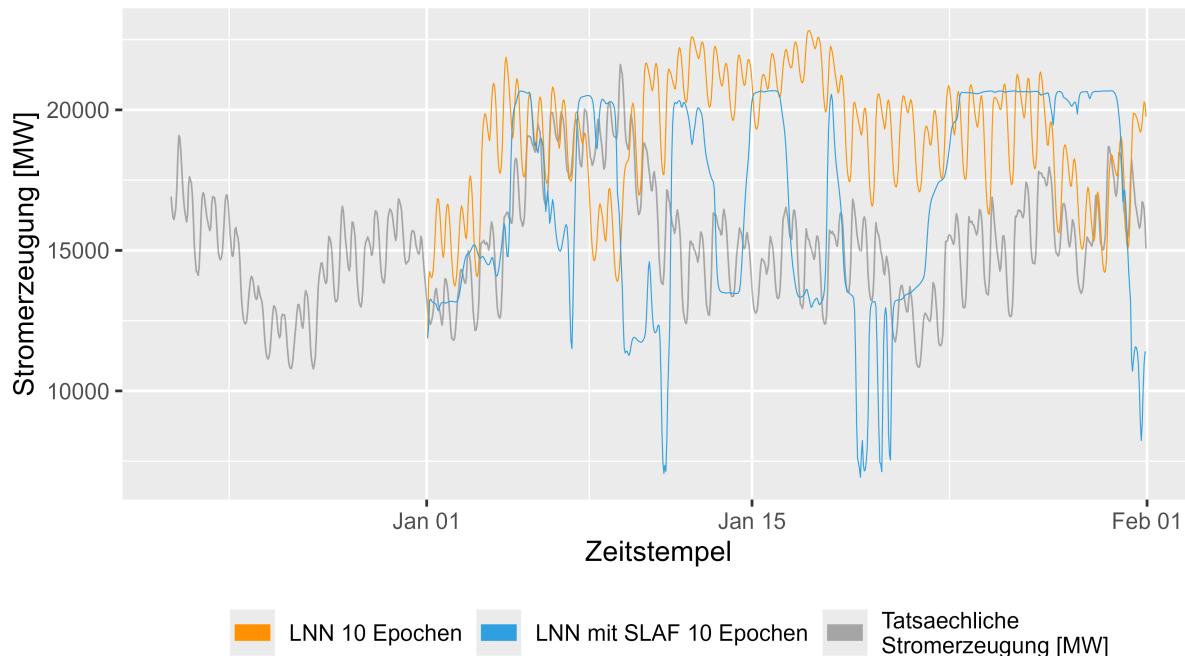


Abbildung 11: Nach 10 Epochen mit und ohne SLAF, Ausschnitt für Januar

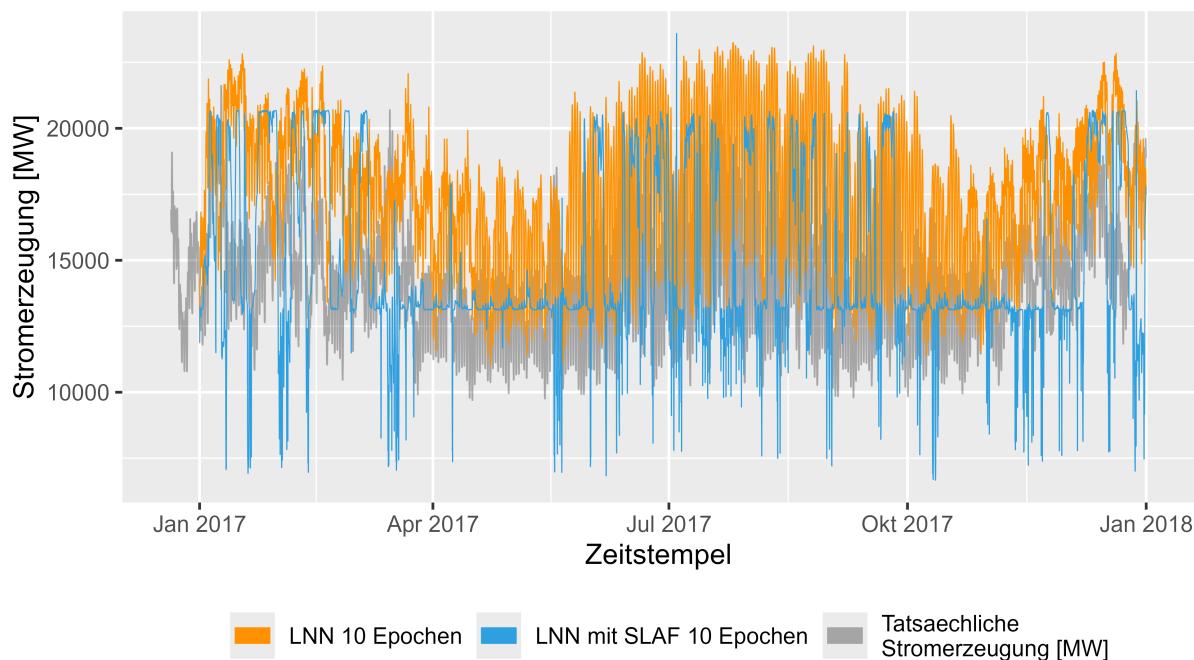


Abbildung 12: Nach 10 Epochen mit und ohne SLAF, ganzes Jahr 2017

5 Vorhersage Ergebnisse

Ein Vergleichskriterium für die verschiedenen Modelle ist der MAPE. Es wurden verschiedene statistische Modelle und Methoden implementiert und ausprobiert. ARIMA Modell, NAIVE Methode, MEAN Methode, DRIFT Methode und das Prophet-Modell. Alle Modelle wurden vereinfacht implementiert und nicht auf den Datensatz optimiert. Schneidet das neuronale Netz schlechter ab als eins dieser Modelle, so fliegt es aus der Betrachtung für eine Vorhersage raus. Im Folgenden werden die Vorhersagen für das Jahr 2017, jeweils immer für die nächsten 24 Stunden, vorgestellt. D.h. jedes Modell sagt nur einen Tag voraus und das über das ganze Jahr.

Zuerst wird das beste statistische Modell vorgestellt, das Prophet Modell, mit einem MAPE von 7.49%. Die Abbildung 13 zeigt die Vorhersage. Eine Anmerkung hier zu den statistischen Modellen ist das Vorhersageintervall. Denn bei den neuronalen Netzen ist ein Vorhersageintervall eher schwieriger darzustellen und zu berechnen. Das Prophet-Modell erkennt das Muster relativ gut und ist auch unter der 10% Grenze, die in der Energiewirtschaft für Energieprognosen oft als ein Ausschluss und Brauchbarkeitskriterium gesetzt wird.

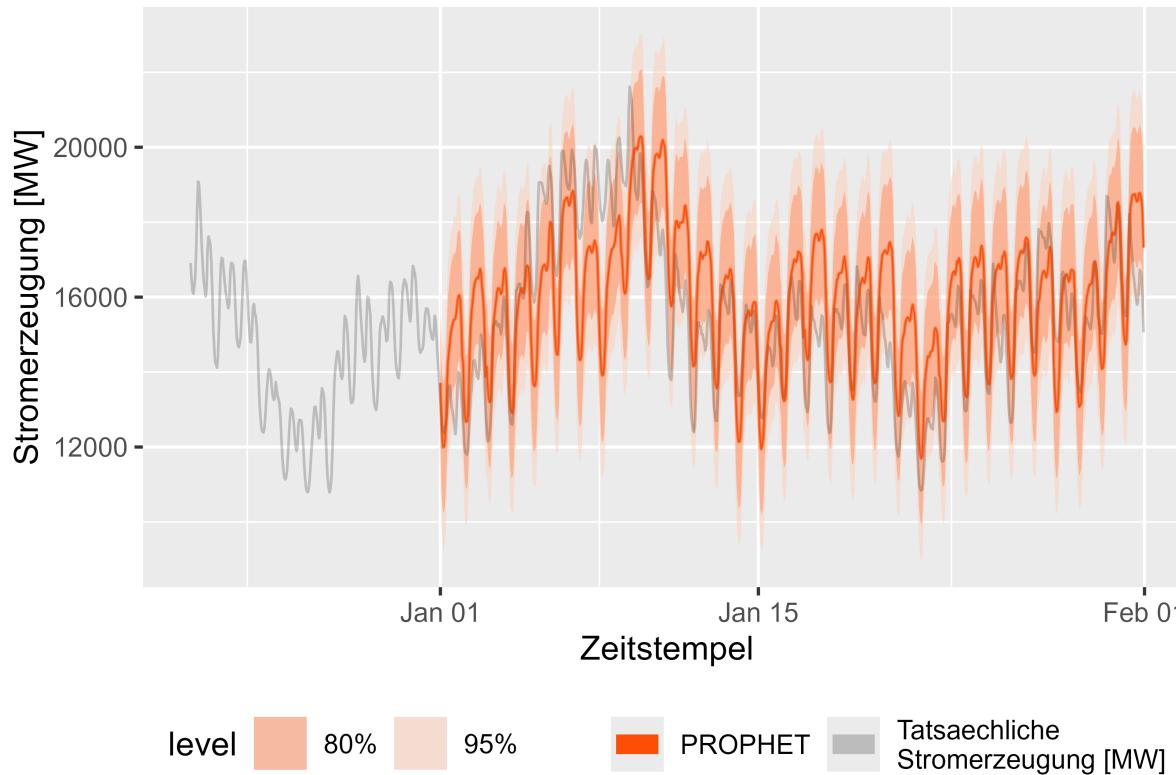


Abbildung 13: Prophet, Ausschnitt für Januar

Die Abbildung 14 stellt die Vorhersage gegenüber den realen Beobachtungen dar. Deutlich zu erkennen, dass die Vorhersage werte für das Prophet-Modell sich um die schwarze Gerade streuen. Diese Gerade soll eine perfekte Vorhersage darstellen. Die Streuung um diese Gerade kann auch als Residueen angesehen werden.

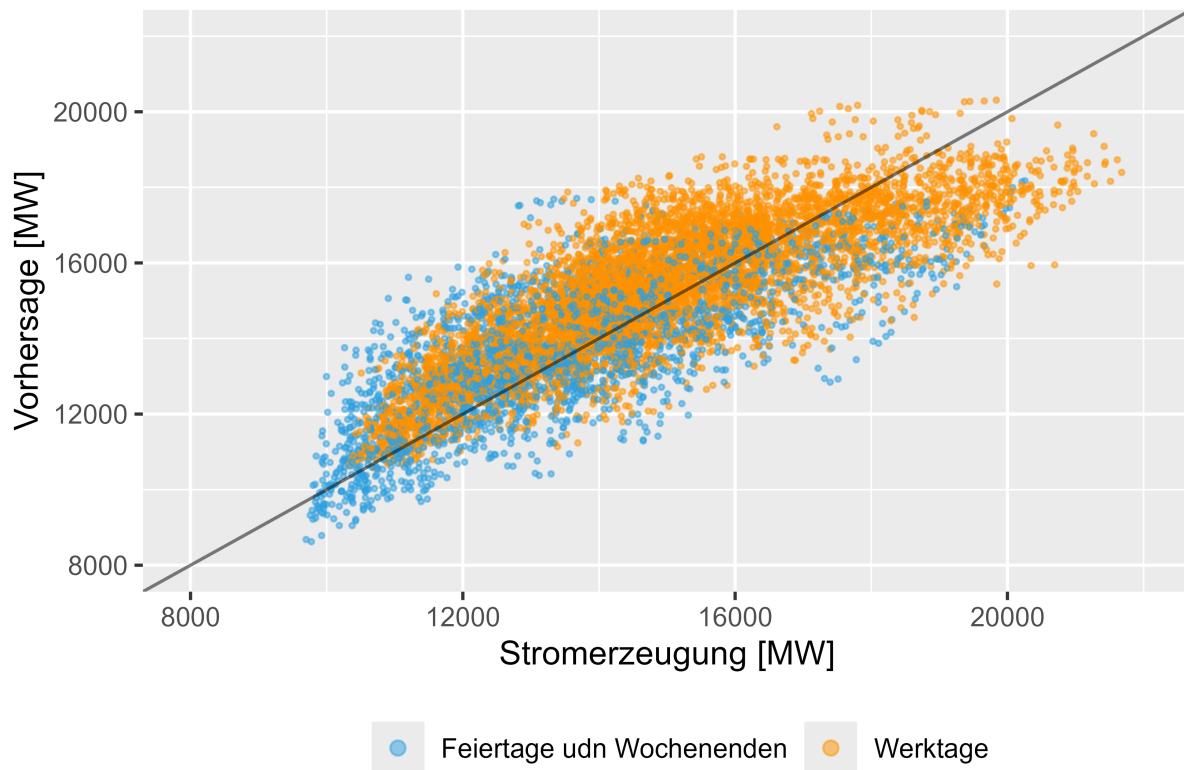


Abbildung 14: Prophet, Vorhersage gegenüber realen Beobachtungen

Das neuronale Netz ist mit einem MAPE von 6.62% besser als das Prophet-Modell. Dabei ist zwischen "LNN mit SLAF" und "LNN" zu unterscheiden. LNN mit SLAF konnte leider nicht so gut performen wie LNN (ohne SLAF). Ein Grund dafür kann z.B. sein, dass das LNN nicht für ein SLAF geeignet ist, oder dass SLAF für Zeitreihen nicht anwendbar ist. Die Abbildung 15 zeigt die Vorhersage für den Januar 2017 nach 100 Epochen. Hier wird gezeigt, dass LNN sich besser an das saisonale Muster anpassen kann als ein LNN mit SLAF. LNN mit SLAF hat zwar an einigen Stellen das Muster erkennen können, aber nicht über den ganzen Datensatz hinweg.

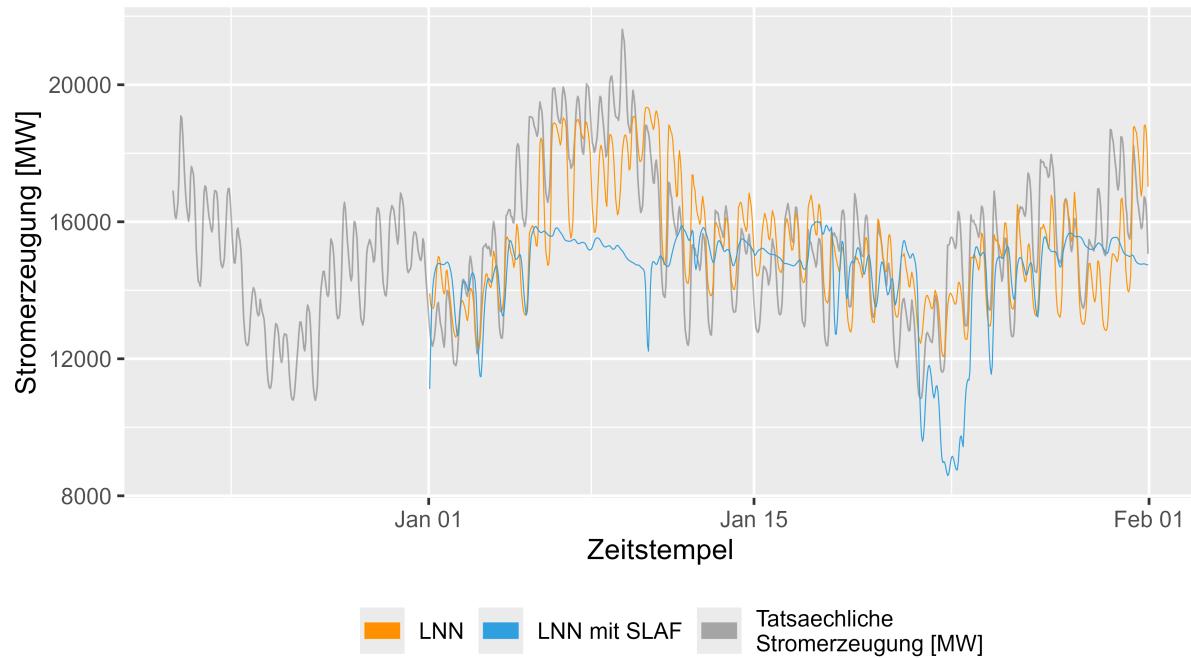


Abbildung 15: Nach 100 Epochen mit und ohne SLAF, Ausschnitt für Januar

Auch in der Jahresbetrachtung sieht LNN begründet aus. LNN passt sich an das moustache-Muster über das ganze Jahr sehr gut an und ist damit auch für eine Vorhersage von Stromerzeugung bzw. von Zeitreihen geeignet. LNN mit SLAF war in dem Fall nicht so erfolgreich. Zwar folgt es halbwegs dem Verlauf, erkennt aber das saisonale Muster nicht so gut wie LNN ohne SLAF. Die Abbildung 16 zeigt dabei die Vorhersage der beiden Modelle für das Jahr 2017 nach 100 Epochen.

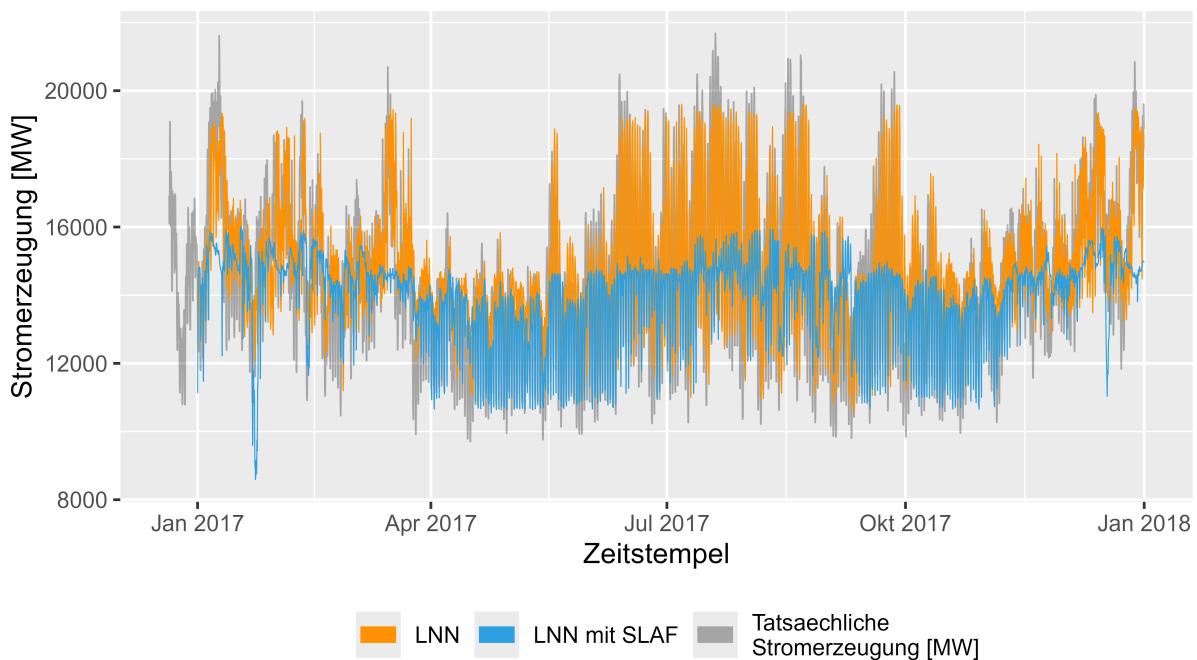


Abbildung 16: Nach 100 Epochen mit und ohne SLAF, ganzes Jahr 2017

Der Vergleich zwischen Abbildung 14 und 17 zeigt, dass beim LNN ein ähnliches Muster entsteht wie beim Prophet Modell für die Vorhersage werte gegenüber den echten Beobachtungen. Anzumerken hier ist, dass LNN eine künstliche Grenze im oberen Bereich gezogen hat. Diese Grenze konnte mit Normalisierung zwar gedämmt werden, konnte jedoch nicht komplett aufgelöst werden. Dies zeigt, dass LNNs noch mehr Potenzial haben kann. Wobei das LNN noch weiter optimiert werden muss.

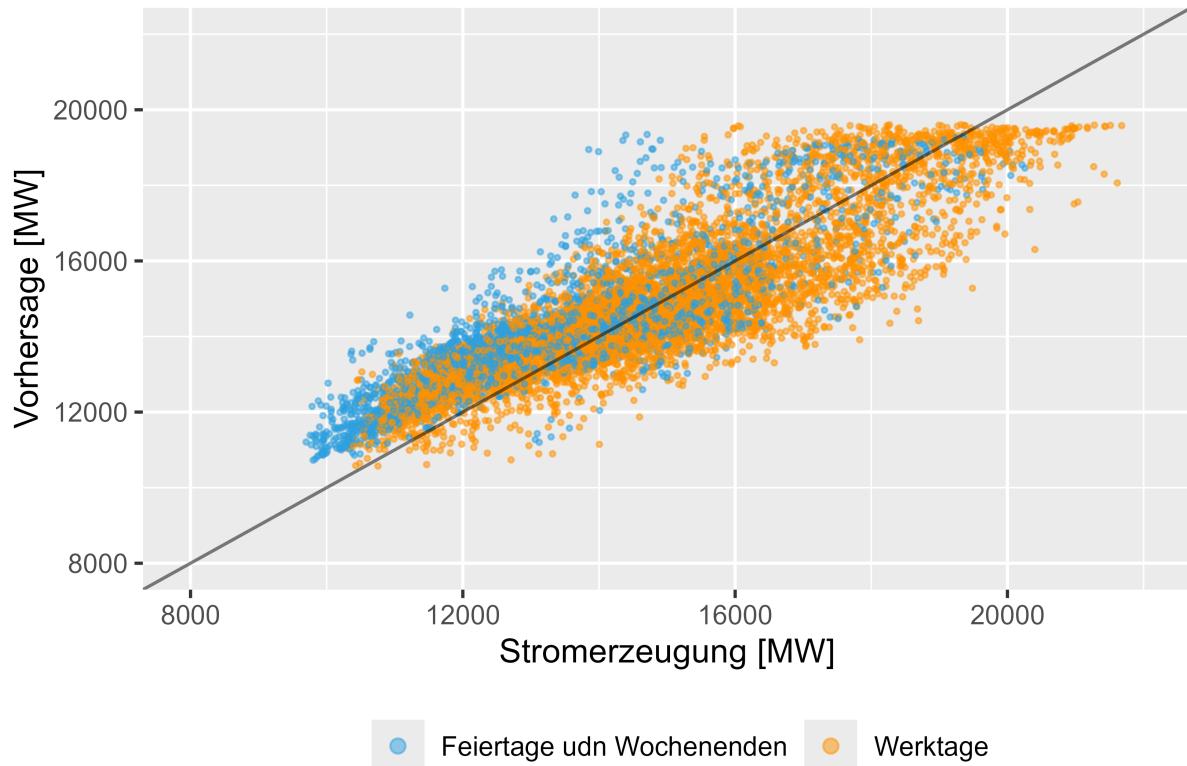


Abbildung 17: Nach 100 Epochen LNN, Vorhersage gegenüber realen Beobachtungen

LNN mit SLAF in Abbildung 18 sieht im Gegensatz dazu nicht so gut aus. LNN mit SLAF hat eine stärkere Unter- und Obergrenze gezogen. Die Datenwolke ist konzentrierter und breiter um die perfekte Vorhersage gestreut. Wieder deutlich, dass LNN mit SLAF nicht so gut geeignet ist wie LNN ohne SLAF.

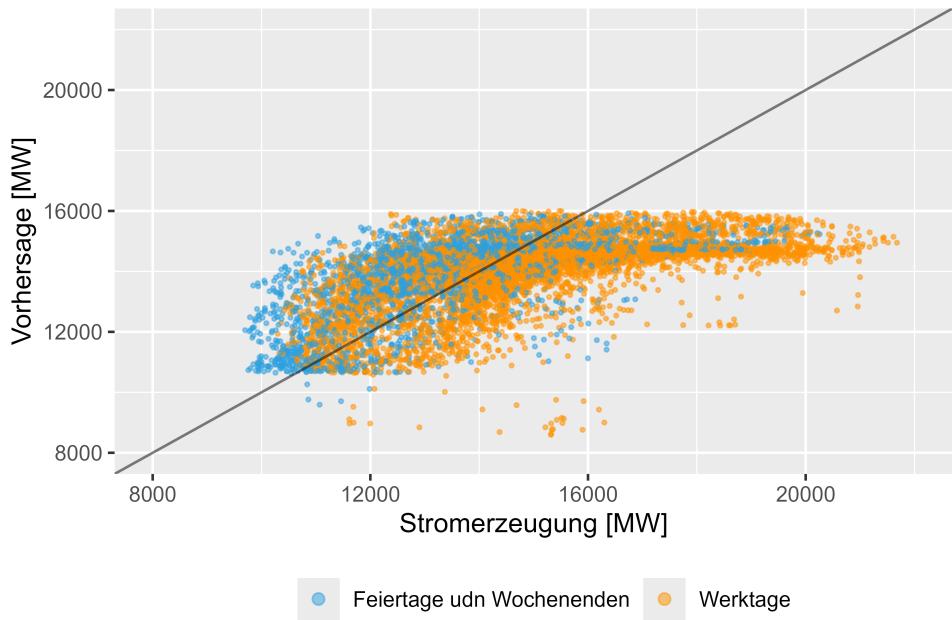


Abbildung 18: Nach 100 Epochen LNN + SLAF, Vorhersage gegenüber realen Beobachtungen

Die Trainings-Verluste zeigen jedoch, dass im Training sich LNN mit SLAF an den Verlust vom Training ohne SLAF annähert. Genau dieses Verhalten war auch zu erwarten, da SLAFs mit swish nur eine Approximation der echten ReLU-Funktion ist.

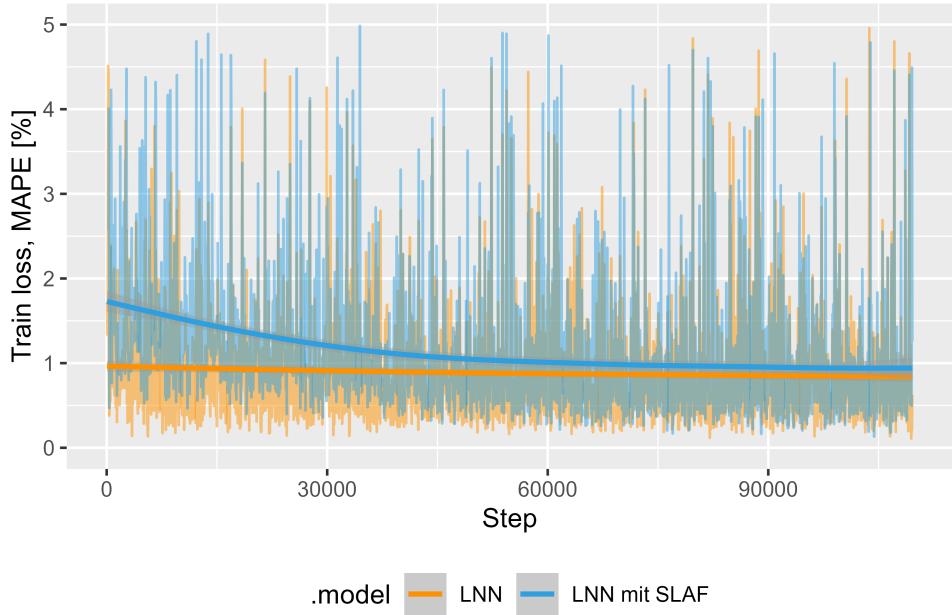


Abbildung 19: MAPE Verlust in den einzelnen Schritten (eine Epoche sind mehrere Schritte)

Die Tabelle 2 fasst nochmal alle Ergebnisse zusammen. Dabei wurden die MAPEs, MAEs und RMSEs für die Vorhersage vom ganzen Jahr 2017 berechnet. Wie besprochen ist LNN mit einem MAPE von 6.62% ein sehr gutes Ergebnis für die Stromerzeugung und geeignet für eine Vorhersage. Anzumerken ist hier, dass das ARIMA Modell und das Prophet-Modell nicht optimiert wurden. Ebenfalls konnte LNN nicht vollständig durch die enorme Trainingszeit (2 Studen ca. für 100 Epochen) optimiert werden.

Nr.	Model	RMSE	MAPE	MAE
1	LNN	1170	6.62	937
2	PROPHET	1351	7.49	1071
3	ARIMA	1421	7.68	1118
4	LNN mit SLAF	1895	9.63	1443
5	NAIVE	2423	12.60	1887
6	DRIFT	2592	13.26	2025
7	MEAN	2322	13.72	1888
8	LNN mit SLAF 10 Epochen	3721	19.83	2865
9	LNN 10 Epochen	3779	23.57	3307

Tabelle 2: Vergleich der Modelle mit RMSE, MAPE und MAE

6 Fazit

In dieser Arbeit wurde gezeigt, dass LNNs ohne SLAF für eine Vorhersage von Zeitreihen im Bereich der Energiewirtschaft geeignet sind. Durch eine umfassende Datenanalyse konnten die notwendigen Features für die Modellierung extrahiert werden. Der Vergleich zu den herkömmlichen statistischen Modellen zeigt ein großes Potenzial im Bereich von Zeitreihen für LNNs auf. Schon in wenigen Schritten kann ein LNN so optimiert werden, dass es ein unoptimiertes statistisches Modell übertrumpft.

Des weiteren unterscheidet sich ein LLN in der Größe zu den statistischen Modellen enorm. Dort, wo ein LNN nur ca. 100 KB braucht, braucht ein statistisches Modell ca. 1 MB. Dies zeigt nochmal, dass LNNs kompakter sind und weniger Rechenleistung für eine Vorhersage brauchen. So kann ein Prophet-Modell ca. 1 Minute brauchen, um eine Vorhersage zu treffen und ein LNN nur wenige Sekunden im Gegensatz.

Eine mögliche Optimierung des LNNs ist die Betrachtung der unterschiedlichen Feiertage. So kann für jeden Feiertag eine Dummy-Variable erstellt werden, die in das Modell einfließen soll. Eine weitere Möglichkeit ist es, den Trainingsdatensatz zu erhöhen. Anstatt die letzten 3 Jahre zum Vorhersagejahr zu nehmen, soll der gesamte Datensatz als Trainingsdatensatz genommen werden und das letzte Jahr als ein Testdatensatz.

Die saisonalen Muster werden genauso gut von dem LNN erkannt wie von einem statistischen Modell. Der einzige Nachteil eines LNNs gegenüber einem statistischen Modell sind die Vorhersageintervalle. Diese werden bei einem LNN nicht automatisch mit erzeugt, können jedoch durch weitere Verfahren ermittelt und für LNNs angepasst werden (Forschungsfrage). LNNs mit SLAF performen zwar nicht so gut, können aber vielleicht weiter angepasst werden. Denn LNNs sind von Grund auf nicht für SLAF optimiert worden. Dabei könnte eine weitere Forschungsfrage entstehen, und zwar wie SLAF am besten für LNNs optimiert werden kann und welche Vorteile das dann mit sich bringt.

Im Großen und Ganzen sollten LNNs im Bereich der Zeitreihen weiter untersucht werden und tiefergehende Modellierungen und Optimierungen erzeugt werden.

7 Anhang

Git-Repo: <https://github.com/vladislavv/ncps-slaf>.

Datensatz: AEP_hourly_cleaned.csv ist der verwendete Datensatz fürs Training.

Im Ordner ncps-slaf ist nochmal der verwendete Code beigelegt.

Im Ordner ncps-slaf/analysis ist die Datenanalyse zu finden (R).

Im Ordner ncps-slaf/notebooks und ncps-slaf/project ist unser Code für LNN zu finden (Python).