# Nutzerleitfaden zur Erstellung von Fernwärmelastprognosen mittels Machine Learning Verfahren

*Dieser Nutzerleitfaden bezieht sich auf die unter der nachfolgenden Internetadresse veröffentlichten Materialien (u.a. Python-Quellcode, Testdaten, Anleitungen):*
***https://github.com/deepDHC/deepDHC-user-guide***

## Motivation

Zeitreihenprognosen können eine leistungsfähige Methode zur Vorhersage des Wärmebedarfs eines Fernwärmenetzes darstellen. Durch die Analyse historischer Daten über den Energieverbrauch und andere relevante Faktoren wie Wetterdaten kann ein Neuronales Netz Muster und Trends erkennen, die ihm helfen, genaue Vorhersagen über den künftigen Bedarf zu treffen. Auf diese Weise kann der Netzbetreiber die Ressourcennutzung der Anlage optimieren, Kosten einsparen und Emissionen reduzieren und sicherstellen, dass die Wärme effizient und kostengünstig an die Kunden geliefert wird. Darüber hinaus können genaue Prognosen dem Betreiber bei der Planung von Wartungs- und Reparaturarbeiten helfen und sicherstellen, dass das Netz auch in Zeiten hoher Nachfrage zuverlässig und funktionsfähig bleibt. Insgesamt kann die Zeitreihenprognose dazu beitragen, dass Fernwärmenetze effizienter arbeiten und Geld sparen, während sie den Kunden einen guten Service bieten.

Eine mögliche Deep Learning Methode, die für die Zeitreihenprognose verwendet werden kann, ist das LSTM. Ein LSTM-Netzwerk (Long Short-Term Memory) ist ein Rekurrentes Neuronales Netzwerk (RNN), das sich besonders für die Verarbeitung und Vorhersage sequenzieller Daten eignet. Es verwendet eine komplexe Architektur von Zellen, die sich Werte über lange Zeiträume hinweg merken können, so dass es Abhängigkeiten und Muster in sequentiellen Daten erfassen kann. In diesem kurzen Leitfaden wird anhand eines Beispiels gezeigt, wie thermische Last- und Wetterdaten verwendet werden können, um ein LSTM-Modell zu trainieren. Dieses Modell wird dann gespeichert und von der Festplatte geladen, um Lastvorhersagen für einen anderen Zeitraum zu machen. Darüber hinaus wird gezeigt, wie die Fehler der Vorhersagen berechnet werden können, wenn der tatsächliche Lastbedarf bekannt ist, wodurch ein kurzer Überblick über alle grundlegenden Komponenten gegeben wird, die für die Erstellung eines Modells zur Vorhersage des Fernwärmebedarfs erforderlich sind.

Dieser Leitfaden wurde von der Hochschule Kempten im Rahmen des Forschungsprojekts "deepDHC" erstellt. Weitere Informationen zum Forschungsprojekt sind unter deepDHC.de zu finden. DeepDHC wurde vom Bundesministerium für Wirtschaft und Klimaschutz unter der Fördernummer 03EN3017 gefördert. Weitere finanzielle und sachliche Unterstützung erfolgte durch die Projektpartner AGFW, Fernwärme Ulm GmbH und ZAK Energie GmbH. Die Verantwortung für die Inhalte der Arbeit liegt bei den Autoren.

## Installation

Um das Beispiel ausführen zu können, wird eine beliebige Version von Python 3.7 benötigt. Alle Abhängigkeiten können in einer globalen Python-Umgebung installiert werden, es wird jedoch empfohlen, eine virtual env zu verwenden. Alle benötigten Packages können mit folgendem Befehl installiert werden:

```
pip install -r requirements.txt
```

## Beispiel

Das folgende Beispiel befindet sich in `samples/lstm.py`

Zunächst liest der Code eine CSV-Datei ein, die den gesamten für das Beispiel benötigten Datensatz enthält. Es werden dabei nur die Basisdaten geladen. Es gibt viele weitere Features, die aus diesem Basisdatensatz extrahiert werden können, wie zum Beispiel der Jahrestag aus dem Datum im Datensatz. Dies geschieht mit Hilfe von `add_derived_features` , `shfit_n_loads` bzw. `shift_forecast` . Durch die Verschiebung der Lasten und der Vorhersage können dem Modell Informationen über den Lastbedarf vor dem Zeitpunkt der Vorhersage oder über die Wetterlage zu diesem Zeitpunkt geliefert werden.

```
data_path = '../data/data-clean.csv'
data = pd.read_csv(data_path)

data = dp.add_derived_features(data)
data = dp.shift_n_loads(data, 6)
data = dp.shift_forecast(data, 72)
```

Danach definiert der Code einen Zeitraum, der für das Training des Modells verwendet wird, und weitere Variablen, die festlegen, welcher Zeitraum später im Beispiel vorhergesagt werden soll.

```
time_frame = [TimeFrame(datetime(year=2014, month=9, day=15, hour=0),
                        datetime(year=2021, month=12, day=31, hour=0))]
prediction_start = datetime(year=2022, month=1, day=1, hour=0)
prediction_end = datetime(year=2022, month=9, day=30, hour=0)
```

Der folgende Code trainiert dann ein LSTM-Modell auf den Trainingsdaten mit einem Cut-off, der auf dem Beginn der Vorhersagen basiert, die später im Beispiel gemacht werden. Dies ist wichtig, um Data Leakage zu vermeiden. Die Ergebnisse sind nur anwendbar, wenn das Modell die Daten, die es während der Prognose vorhersagen wird, noch nie gesehen hat. Wenn das Training abgeschlossen ist, wird der mittlere absolute Fehler (MAE) des trainierten Modells, der seine Genauigkeit angibt, für den Benutzer ausgegeben.

```
cut_off_date = pd.Timestamp(prediction_start).tz_localize('utc')
train_data = data[data['MESS_DATUM'] < cut_off_date]
model, result = train(train_data, plotting=True)
print(f"training finished with a mae of: {result.mae}")
```

Der Code speichert dann das trainierte LSTM-Modell in einer Datei ab und lädt es anschließend wieder. Dieses Beispiel unterstützt der Einfachheit halber nur das Speichern in einer Datei, aber das Modell kann auch in einer Datenbank oder einer Cloud gespeichert werden.

```
save_model('lstm.model', model, 'multi')
model = load_model('lstm.model')
```

Anhand des geladenen Modells werden dann für den zuvor definierten Prognosezeitraum Prädiktionen erstellt. Da die tatsächlichen Lasten für diesen Zeitraum bekannt sind, gibt der Code nicht einfach die vorhergesagten Werte zurück. Er vergleicht sie mit dem tatsächlichen Lastbedarf in diesem Zeitraum und berechnet mehrere Fehler, um die Genauigkeit der Vorhersage zu ermitteln. Schließlich werden diese Fehler für den Benutzer ausgegeben.

```
pred, true = get_comparison_data_for_time_frame(prediction_start,
                                                prediction_end,
                                                data,
                                                model)
nmae, mape, rmse, mae = calc_all_errors(true_values=true, predicted_values=pred)
print(f'nmae: {nmae}, mape : {mape}, rmse: {rmse}, mae: {mae}')
```