Hochschule Kaiserslautern
University of Applied Sciences

Angewandte Ingenieurwissenschaften Kaiserslautern

elab2go
Mobile Engineering Lab

Was ist Maschinelles Lernen? Konzepte, Methoden, Tools

Prof. Dr. Eva Maria Kiss

Offene Digitalisierungsallianz Pfalz

Wir sind dabei!

Digitaltag 2022

© elab2go // Kaiserslautern, 2022

Übersicht

- Machine Learning: Konzepte, Methoden, Tools
- Lebenszyklus des Überwachten Lernens
- Künstliche Neuronale Netzwerke
- Anwendung: Prognose von Stromverbrauchsdaten

© elab2go // Kaiserslautern, 2022

Was ist Maschinelles Lernen?

Eine Maschine **lernt** aus der **Erfahrung E** hinsichtlich einer Klasse von **Aufgaben T** und dem **Performance-Maß P**, falls die Performance P hinsichtlich T mit E sich verbessert.

[Tom Mitchell, Machine Learning, 1997]

Informell:

Maschinelles Lernen ist ein Teilgebiet der Künstlichen Intelligenz, das es Systemen ermöglicht, auf Basis von Trainingsdaten automatisch zu lernen.

Maschinelles Lernen befasst sich mit der Entwicklung lernfähiger Systeme und Algorithmen.

© elab2go // Kaiserslautern, 2021

Was ist Maschinelles Lernen?

Maschinelles Lernen ist ein Teilgebiet der Künstlichen Intelligenz, das es Systemen ermöglicht, automatisch zu lernen und hinzuzulernen.

Maschinelles Lernen befasst sich mit der Entwicklung lernfähiger Systeme und Algorithmen, und verwendet dabei Konzepte und Methoden der Statistik und der Informationstheorie.

Maschinelles Lernen bietet eine vereinheitlichte Beschreibung existierender Lernverfahren, zu denen unter anderem **Entscheidungsbaum-Lernen**, Lernen mit Hilfe **Künstlicher Neuronaler Netzwerke** oder **Bayessches Lernen** gehören.

© elab2go // Kaiserslautern, 2022

Historische Entwicklung

Maschinelles Lernen hat seit seinen Anfängen um 1950 mehrere Auf-und-ab-Phasen erlebt. Datenzentrisches Maschinelles Lernen ist seit 1990 eine Disziplin der Datenanalysten und Statistiker.

Seit 2005 hat **Deep Learning** die Forschung beflügelt und zu einem neuen Hype geführt.

2005 **Nature-Artikel „Deep Learning“**
2010 **Kaggle** geht online
2012 **Google Brain-Forschung:**
Spracherkennung, Fotosuche,
YouTube-Videoempfehlungen, Robotik
2014 **Facebook DeepFace:**
Gesichtserkennung in Fotos

. . .

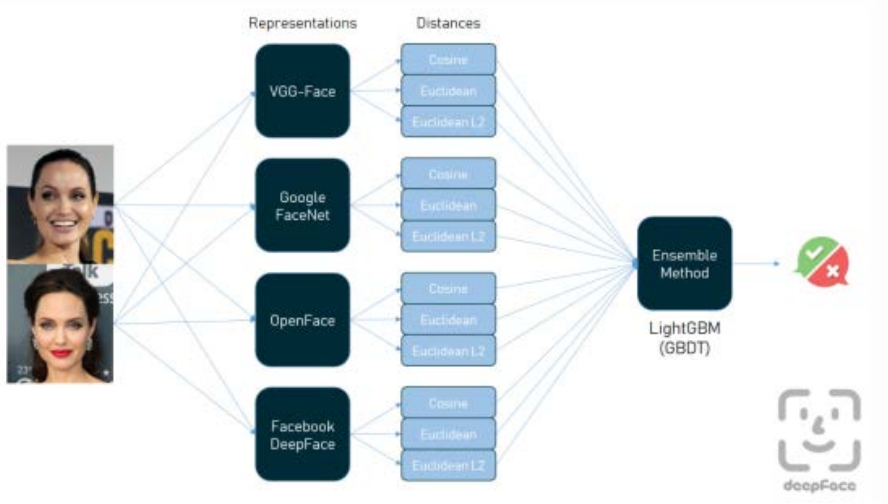


Bild: DeepFace-Projekt auf pypi.org

© elab2go // Kaiserslautern, 2022

Aktueller Hype bedingt durch Big Data

Durch die verstärkte Speicherung anonymisierter Nutzerdaten im Internet und die Digitalisierung in der Industrie (Industrie 4.0) sind **riesige Datenspeicher** entstanden**, die als Basis für lernende Systeme verwendet werden können**.

Der Bedarf, aus den schnelllebigen und (semi)-strukturierten Daten sinnvolle Informationen zu extrahieren (Spracherkennung, Bilderkennung) hat zur Weiterentwicklung der Sprachen und Bibliotheken für Maschinelles Lernen geführt.

Heute wird Angewandtes Maschinelles Lernen verstärkt auch von Informatikern und Ingenieuren eingesetzt.

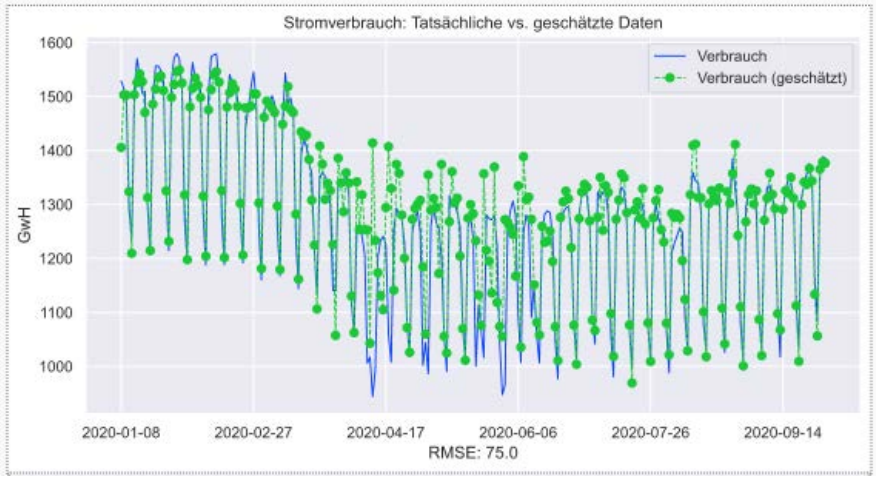
© elab2go // Kaiserslautern, 2022

Machine Learning: Ziele und Anwendungen

Machine Learning-Algorithmen haben im weitesten Sinne das Ziel, aus Input-Daten **sinnvolle Zusammenhänge zu erkennen** und daraus Regeln abzuleiten. Anwendungen sind z.B.

Muster erkennen, Prognosen erstellen, Trends vorhersagen, Daten nach bestimmten Kriterien gruppieren.

Datum	Verbrauch	Wind	Solar	Wind+Solar
2014-01-01	1080.08	220.08	30.49	250.58
2014-01-02	1343.10	304.17	13.68	317.85
2014-01-03	1379.78	340.64	23.51	364.15
...	...	...	...	...
2017-12-29	1295.09	584.28	29.85	614.13
2017-12-30	1215.45	721.25	7.47	728.71
2017-12-31	1107.11	721.18	19.98	741.16



© elab2go // Kaiserslautern, 2022

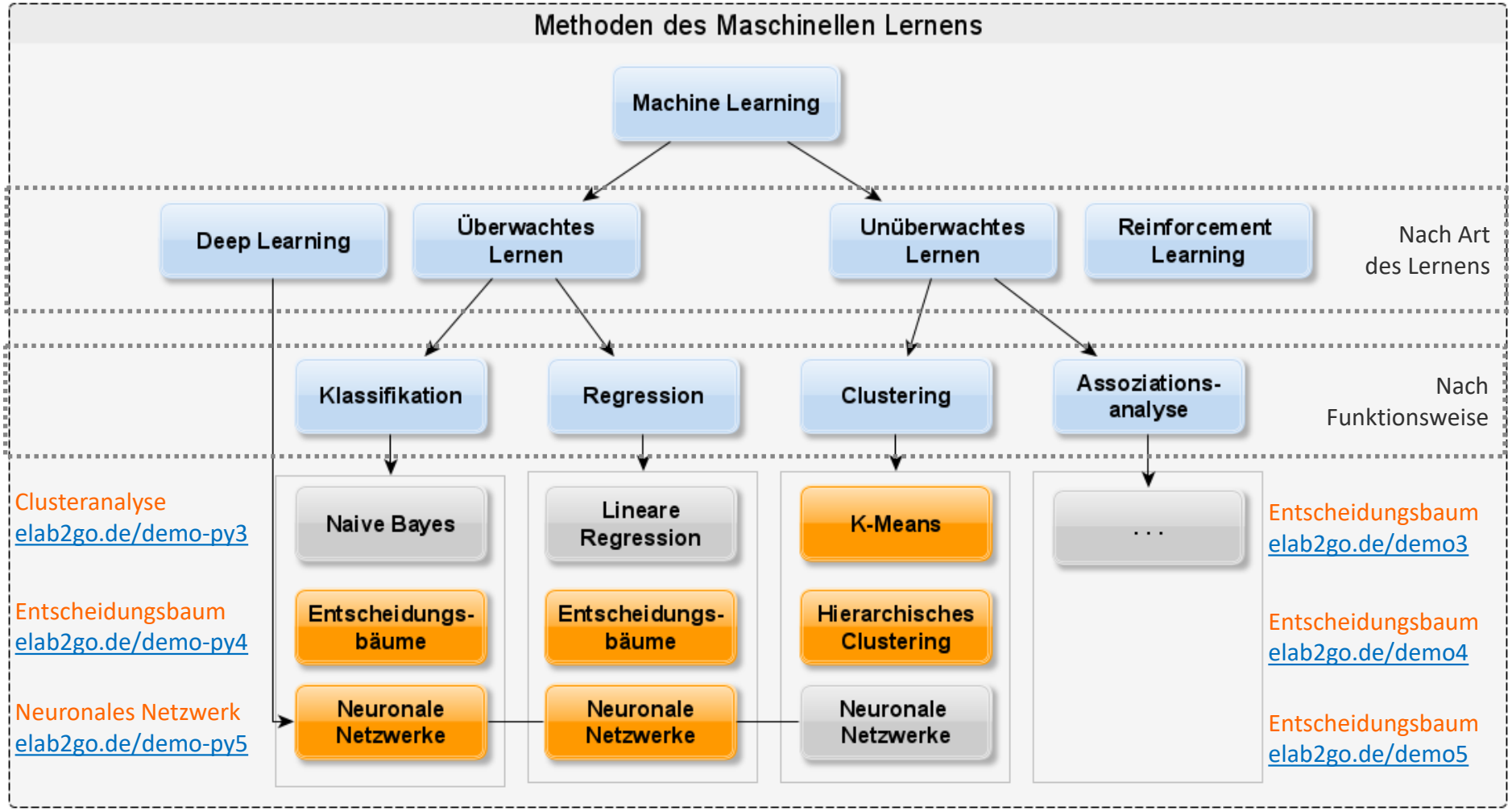
Machine Learning-Anwendungen in der Industrie

Anwendungsgebiete in der Industrie sind:

- **Predictive Maintenance**
  - Klassifikation von Ausfällen
  - Prognose der Entwicklung von Sensordaten
- **Predictive Analytics**
  - Analytisches Customer Relationship Management: Produkt-empfehlungen, Optimierung von Marketing-Kampagnen
  - Betrugsaufdeckung: Ist eine E-Mail Spam oder nicht?
- **Weitere Anwendungen:**
  - Bild- und Texterkennung, Text-To-Speech, Videoüberwachung
  - Virtuelle Assistenten, Stimmungsanalyse

© elab2go // Kaiserslautern, 2022

Machine Learning: Methoden und Algorithmen



© elab2go // Kaiserslautern, 2022

Übersicht der Lernverfahren

Man unterscheidet beim Maschinellen Lernen zwischen verschiedenen Lernverfahren, die jeweils unterschiedliche Abläufe und Anwendungen haben.

- **Überwachtes Lernen (Supervised Learning):** Bewertete Input-Daten vorhanden, Modell wird durch Training erstellt
- **Unüberwachtes Lernen (Unsupervised Learning):** Keine Bewertung der Input-Daten vorhanden, Modell wird durch Gruppierung oder Mustererkennung erstellt
- **Bestärkendes Lernen (Reinforcement Learning):** Lernen durch Interaktion mit Umgebung + Belohnung
- **Transfer Learning:** Lernen durch passende vortrainierte Modelle

© elab2go // Kaiserslautern, 2022

Machine Learning #1: Überwachtes Lernen

Beim **überwachten Lernen** liegt für jeden Datensatz der Input-Daten eine Bewertung vor, d.h. die Daten sind schon in Kategorien unterteilt.

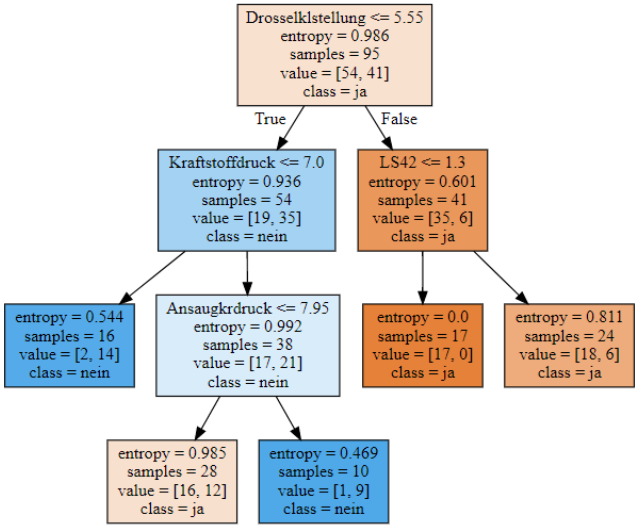
Beispiel: **Entscheidungsbaum-Klassifikationsmodell** für die Vorhersage von Ausfällen an einem Automotive-Datensatz

Automotive Datensatz:
oben Trainings-Daten, unten: Testdaten

Messungsnummer	Ausfall	KuehlmitteltemperaturEinspritzmengeKurzzeit			Luftdruck	KatalysatortempKategorie
M_001	nein	2,6	,9		2,8	normal
M_002	ja	9,2	22,7		2,8	hoch
M_003	nein	3,2	28,5		,1	normal
M_004	ja	2,8	2,6	...	,2	normal
M_005	ja	4,9	14,4		3,0	hoch
M_006	nein	,8	5,5		2,9	hoch
M_007	nein	6,2	26,4		,1	normal

⋮

Messungsnummer	Ausfall	KuehlmitteltemperaturEinspritzmengeKurzzeit	EinspritzmengeLangzeit	Kraftstoffdruck
M_137	?	2,6	,9	4,2	13,8
M_138	?	9,2	22,7	15,3	8,6
M_139	?	3,2	28,5	8,7	4,9
M_140	?	2,8	2,6	5,9	7,4



© elab2go // Kaiserslautern, 2021

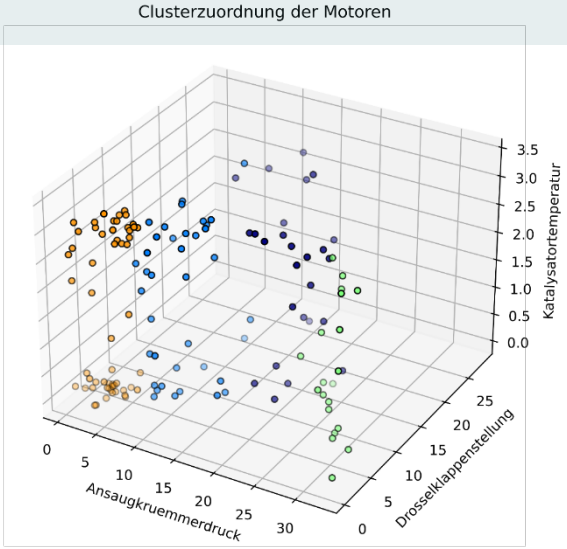
Machine Learning #2: Unüberwachtes Lernen

Beim **unüberwachten Lernen** liegen für die Input-Daten **keine Bewertungen** vor, d.h. das Lernverfahren muss ein Modell ohne Zielvorgaben erstellen, indem es Muster in den Daten erkennt.

Beispiel: **Clusteranalyse** für die Gruppierung der Ausfälle an einem Automotive-Datensatz

Die Frage, die hier mit einer Cluster-analyse beantwortet werden soll, lautet:

"Welche Clusterzuordnung der Motoren erfolgt bzgl. der drei Merkmale, die den stärksten Einfluss auf den Ausfall hatten?"



© elab2go // Kaiserslautern, 2021

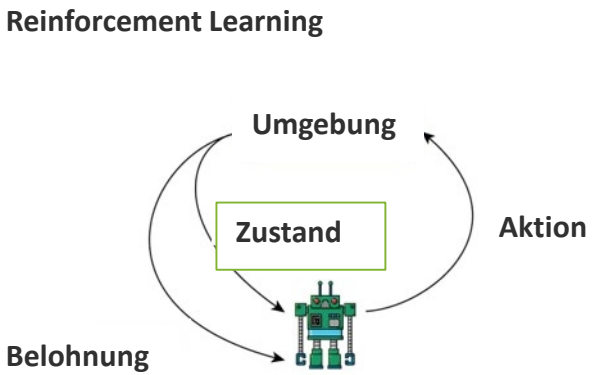
Machine Learning #3: Bestärkendes Lernen

Beim **bestärkenden Lernen (Reinforcement Learning)** werden Agenten eingesetzt, die mit ihrer Umgebung interagieren und Strategien erlernen, um erhaltene Belohnungen zu maximieren.

RL-Anwendungen: Robotersteuerung in der Automatisierung, Autonomes Fahren, Automatische Übersetzungen

Bei dieser Form des Lernens werden vorab keine / wenig Input-Daten benötigt.

Die Daten werden während der Interaktion mit der Umgebung erzeugt.



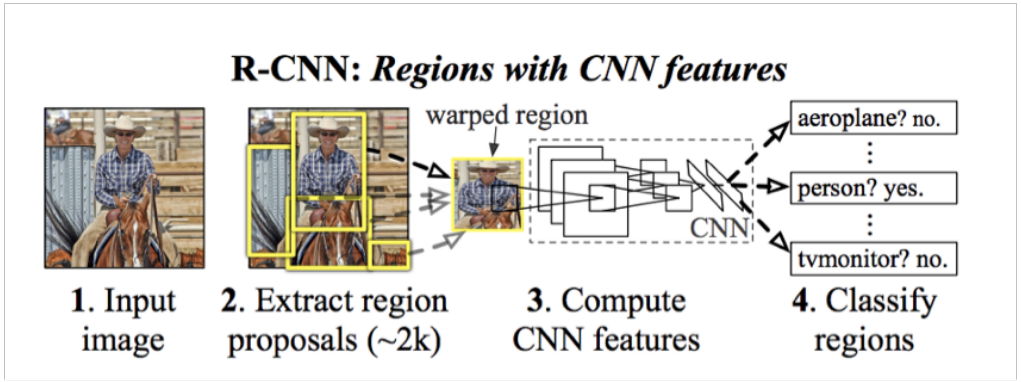
© elab2go // Kaiserslautern, 2022

Machine Learning #4: Deep Learning

Deep Learning ist ein Teilgebiet des maschinellen Lernens, das sich mit komplexen Algorithmen beschäftigt, die aus den Daten auch die relevanten Merkmale lernen und durch diesen Aufbau Probleme wie Objekterkennung lösen können.

Deep Learning-Anwendungen: Autonomes Fahren, Objekt-Klassifikation -Lokalisierung, -Erkennung, Stimmungsanalyse

Speziell für Objekterkennung gibt es verschiedene Frameworks und vortrainierte Modelle (R-CNN, YOLO, SSD, RetinaNet).



© elab2go // Kaiserslautern, 2022

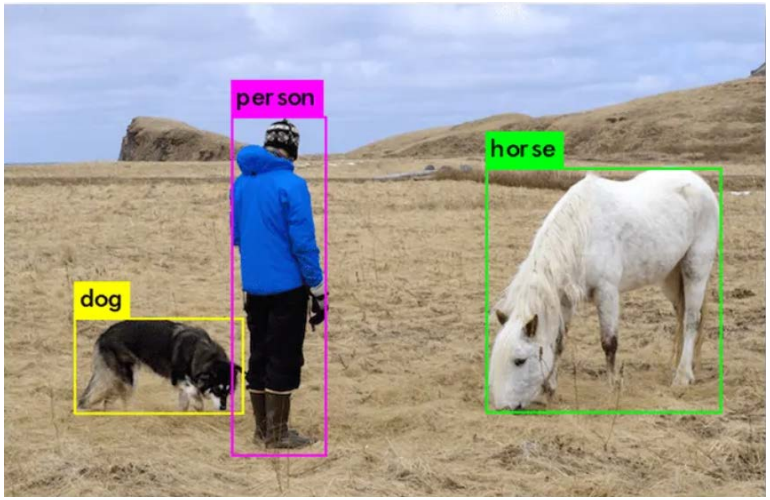
Anwendung: Objekterkennung mittels Deep Learning

Objekterkennung ist eine Teilaufgabe verschiedener Anwendungen und muss oft in Echtzeit durchgeführt werden.

Die Objekterkennung besteht aus eines Klassifikation (Was ist das Objekt?) und einer Lokalisierung (Wo, in welcher "Bounding Box", befindet sich das Objekt?)

Es werden einstufige oder zweistufige Verfahren des Deep Learning eingesetzt:

- Einstufig: YOLO (You look only once), SSD (Single Shot Multibox Detector)
- Mehrstufig (zuerst lokalisieren, dann klassifizieren): R-CNN



© elab2go // Kaiserslautern, 2022

Tools und Sprachen für Maschinelles Lernen

Für die Entwicklung einer ML-Anwendung werden drei Sprachen mit den dazu passenden Entwicklungsumgebungen verwendet:

- R und RStudio ➔ **elab2go Demo2, Demo3, Demo4**
  - Die klassische Lösung (open source)
  - Wird bevorzugt von Datenanalysten in Offline-Szenarien eingesetzt
- Python ➔ **elab2go Demo-PY3, Demo-PY4, Demo-PY5**
  - Die innovative Lösung (open source)
  - Wird im Internet-Umfeld eingesetzt, in Online- oder Echtzeit-Szenarien, von Einsteigern und Programmierern
- MATLAB ➔ **elab2go Demo5, Demo-MAT4**
  - Die enterprise Lösung (kommerziell)
  - Wird bevorzugt von Ingenieuren eingesetzt

© elab2go // Kaiserslautern, 2022

Anwendung im Energiesektor: Analyse von Stromverbrauch mittels Deep Learning

Drei Apps aus dem Werkzeugkasten für Datenanalyse im Energiesektor:

Demo-PY2 und Demo-R2 beantworten die Frage:

„Wie hat sich Verbrauch und Produktion von Strom insgesamt, Solar- und Windenergie in den vergangenen Jahren entwickelt?“

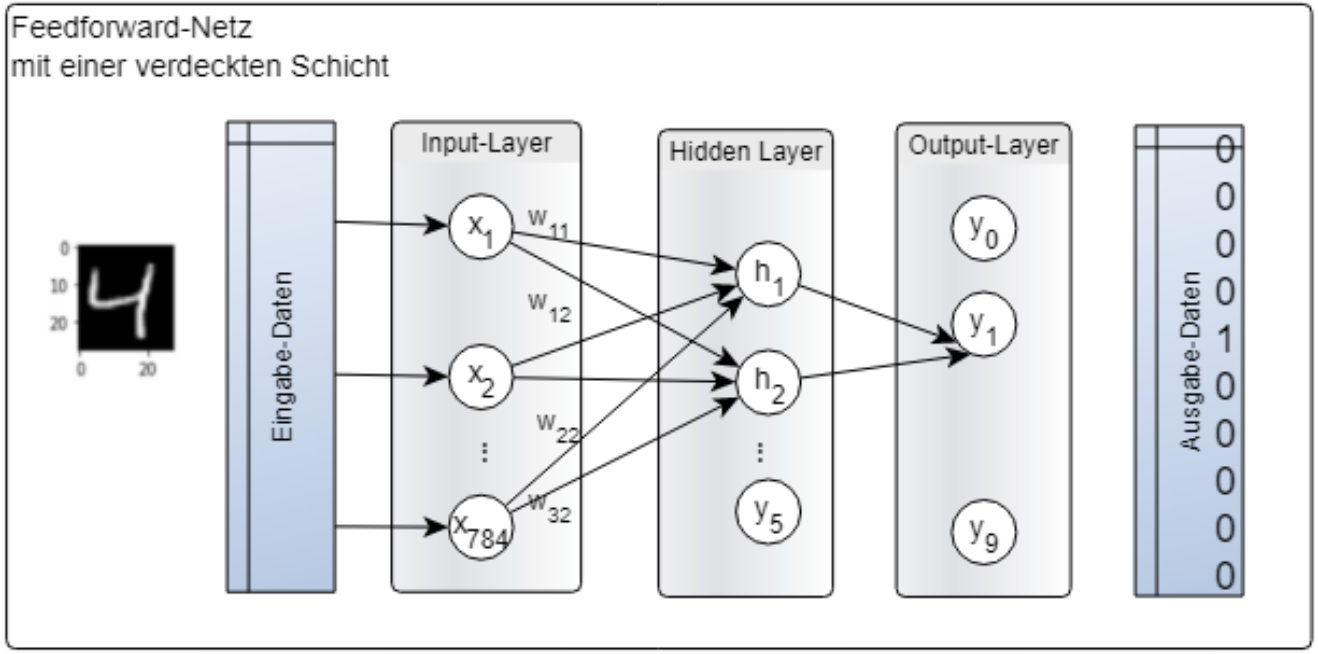
Demo-PY5 beantwortet die Frage:

„Wie wird sich der Stromverbrauch in der Zukunft entwickeln? “

© elab2go // Kaiserslautern, 2021

Was ist ein Künstliches Neuronales Netzwerk?

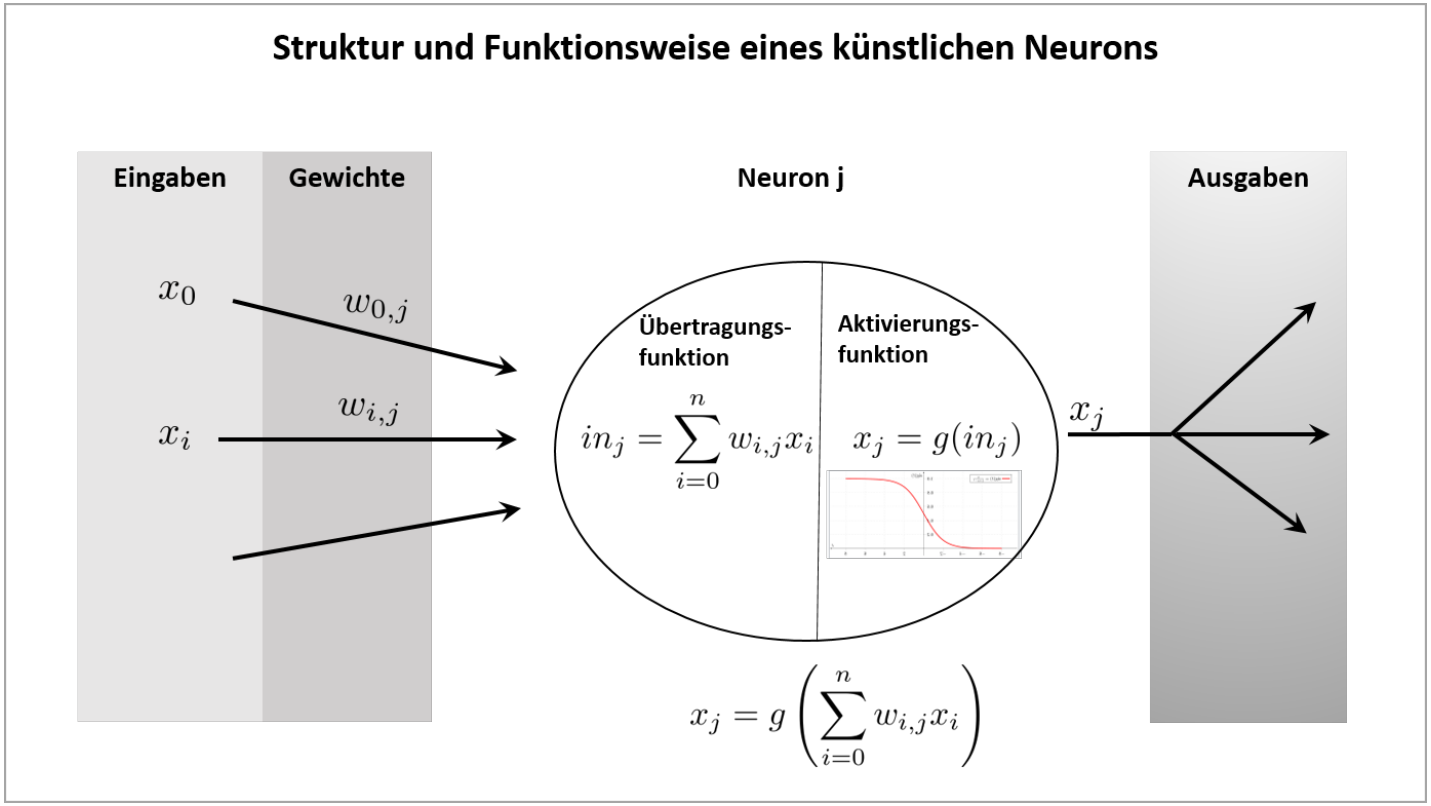
- Ein Netzwerk verbundener künstlicher Neuronen, die in Schichten angeordnet sind.



© elab2go // Kaiserslautern, 2021

Was ist ein Künstliches Neuron?

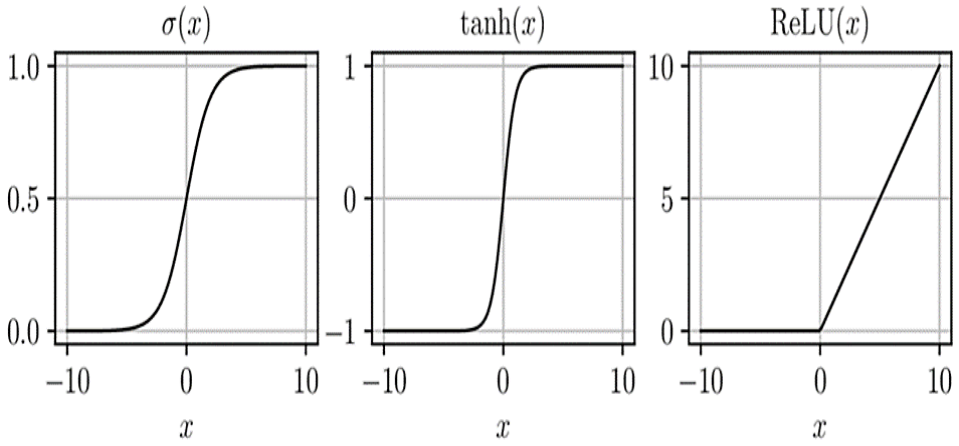
- Der elementare Baustein eines neuronalen Netzwerks



© elab2go // Kaiserslautern, 2021

Aktivierungsfunktionen: Nichtlinear und linear

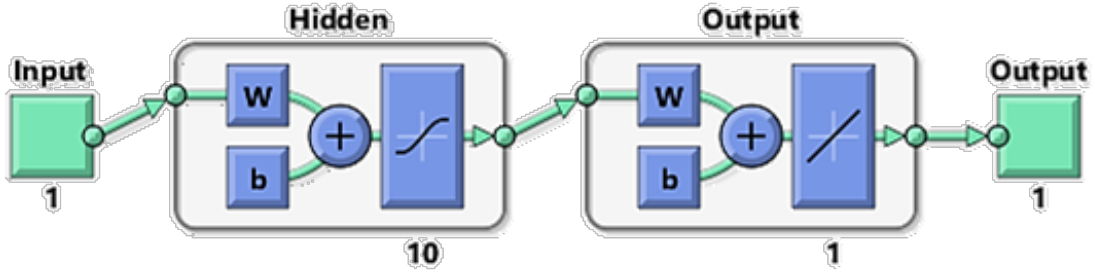
- **Sigmoidfunktion**
  - Bereich zwischen 0 und 1
  - $y = \frac{1}{1+e^{-x}}$
- **Tanh-Funktion**
  - Bereich von [-1, 1]
  - $y = 2\sigma(2x) - 1$
- **ReLU (Rectified Linear Unit)**
  - Ersetzt negative Werte mit 0
  - $y = \max(0, x)$



© elab2go // Kaiserslautern, 2021

Netzwerk-Architekturen: Feed-Forward vs. Rekurrent

- Feed-Forward: Keine Rückkopplungen möglich



- Rekurrente Netzwerke (RNN) : Rückkopplungen möglich

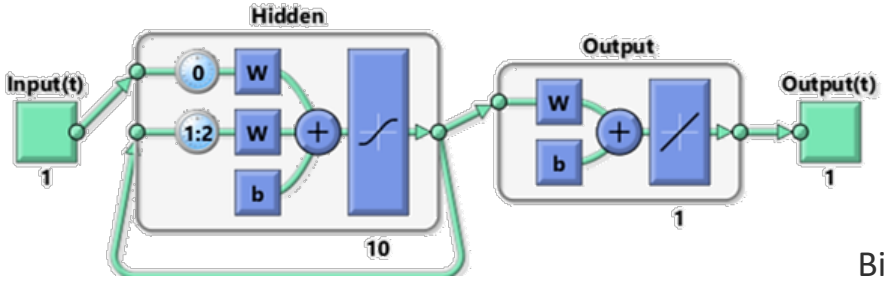


Bild: MATLAB

- RNNs ermöglichen die Modellierung zeitabhängiger und sequentieller Datenaufgaben wie Zeitreihenanalyse.

© elab2go // Kaiserslautern, 2021

Wie lernt ein Neuronales Netzwerk?

- Das Netzwerk entspricht einer Funktion, die die Eingaben x über die Gewichte w in die Ausgaben y überführt: $y = f(w, x)$
- Beim überwachten Lernen führt eine Menge von Trainingsdaten $(x_1, y_1), (x_2, y_2), \ldots, (x_p, y_p)$ ausgehend vom Startgewicht w_0 zur Bestimmung der Gewichte $w_1, w_2, \ldots, w_p$
- Das Startgewicht w_0 wird zufällig festgelegt.
- Das nächste Gewicht w_1 wird aus (x_1, y_1, w_0) durch Minimieren der Verlustfunktion $w \mapsto E(f(w, x_1), y_1)$ bestimmt.
- Hier müssen Gradientabstiegsverfahren verwendet werden, d.h. der Gradient der Verlust- bzw. Fehlerfunktion wird benötigt.

© elab2go // Kaiserslautern, 2021

Tiefe Rekurrente Netzwerk-Architekturen

- Tiefe Rekurrente Netzwerkarchitekturen mit vielen versteckten Schichten leiden unter dem **Problem des Verschwindens von Gradienten:**
  - Durch Matrixmultiplikation kleiner Werte werden die Gradientenwerte immer kleiner.

- Die Gradienten enthalten Informationen, die bei der RNN-Parameteraktualisierung verwendet werden, und wenn der Gradient immer kleiner wird, werden Aktualisierungen unbedeutend, was bedeutet, dass das System nicht mehr lernt.

© elab2go // Kaiserslautern, 2021

LSTM (Long Short-Term Memory) Netzwerke

- Ein LSTM (Long Short-Term Memory) Netzwerk besteht aus komplexeren Bausteinen, die sowohl Rückkopplung erlauben, als auch das "Vergessen" eines Anteils des erhaltenen Inputs.
- Vorteil:
  - Lösen das Problem des verschwindenden Gradienten.
  - Damit können tiefe Netzwerke mit vielen versteckten Schichten erstellt werden.

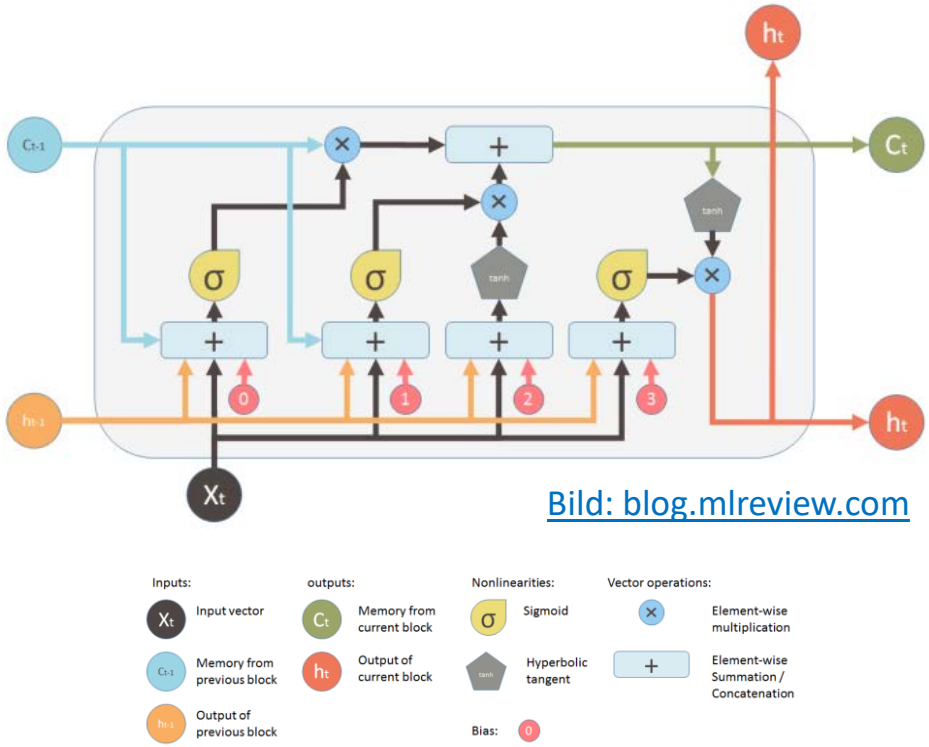
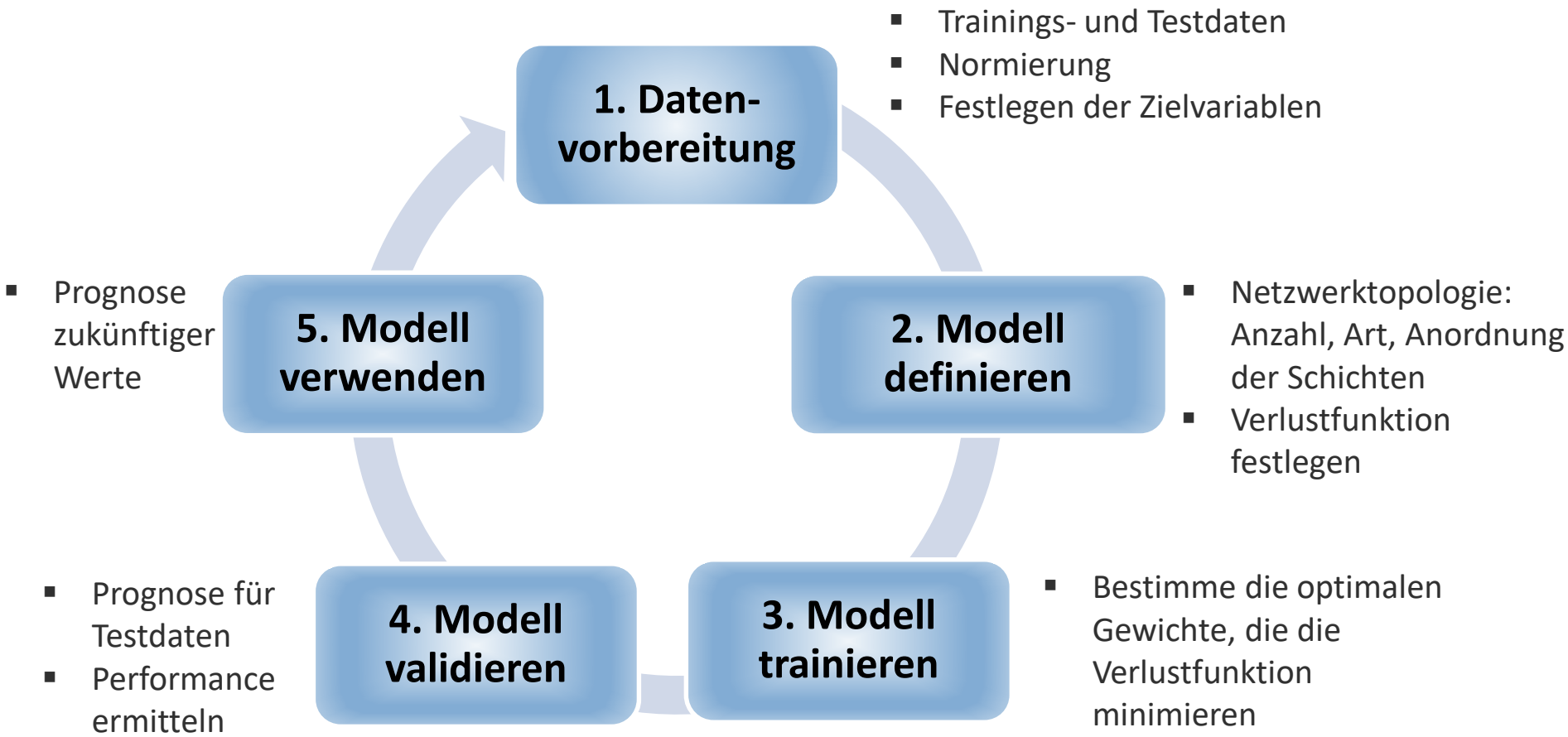


Bild: blog.mlreview.com

© elab2go // Kaiserslautern, 2021

Lebenszyklus des überwachten Lernens

Lebenszyklus des Überwachten Lernens mit einem Neuronalen Netzwerk



© elab2go // Kaiserslautern, 2021

Demo-PY5: Deep Learning mit Keras und Tensorflow
https://www.elab2go.de/demo-py5/

Demo-PY5 zeigt, wie ein Künstliches Neuronalen Netz mit mehreren Long Short-Term Memory (LSTM)-Schichten mit Hilfe der Python-Bibliotheken Keras und Tensorflow trainiert, validiert und damit eine Prognose über die zukünftige Entwicklung der Verbrauchsdaten erstellt wird.

Ein Zeitreihen-Datensatz zum Stromverbrauch in Deutschland aus der Open-Power-System-Data (OPSD)-Plattform dient uns dabei als Anwendungsbeispiel. Die Daten werden in der interaktiven, webbasierten Anwendungsumgebung Jupyter Notebook analysiert.

© elab2go // Kaiserslautern, 2022

Die Fragestellung

Wie ändert sich Verbrauch und Produktion von Strom insgesamt, Solar- und Windenergie über die Jahre?

Reicht die Energiegewinnung über erneuerbare Energien aus, um den Verbrauch zu decken?

Um diese Fragen zu beantworten, wird ein Zeitreihen-Datensatz zum Stromverbrauch in Deutschland aus der Open-Power-System-Data (OPSD)-Plattform zunächst grafisch analysiert und dann wird mit Hilfe eines tiefen künstlichen Neuronalen Netzwerks eine Prognose für die zukünftige Entwicklung der Verbrauchsdaten erstellt.

© elab2go // Kaiserslautern, 2022

Datensatz für Stromverbrauch und -Produktion

- CSV-Datei mit Daten zum Stromverbrauch sowie zur Stromproduktion von Solar- und Windenergie in Deutschland
- 01.01.2016 bis 30.09.2020
- Auflösung: 60 Minuten
- 41641 Datensätze

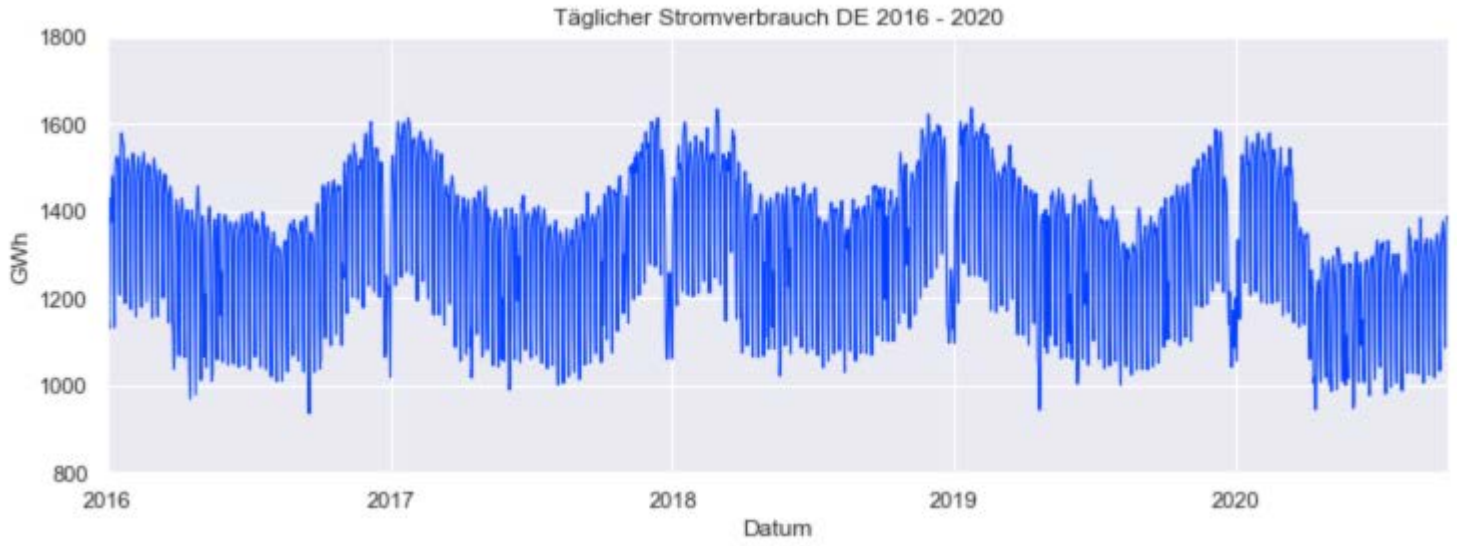
time_series_60min_2016-2020.csv

```
1 TimestampUTC,TimestampCET,Verbrauch,Solar(Kapazität),Solar,Wind(Kapazität),Wind
2 2016-01-01T00:00,2016-01-01T01:00,40126,38631,0,32812,8579
3 2016-01-01T01:00,2016-01-01T02:00,38429,38631,0,32812,8542
4 2016-01-01T02:00,2016-01-01T03:00,37484,38631,0,32812,8443
```

© elab2go // Kaiserslautern, 2022

Die Zeitreihe: Täglicher Stromverbrauch in DE 2016 bis 2020

Nach Aggregation der Daten auf Tagesbasis erhält man eine Zeitreihe, die keinen Trend, jedoch eine klare Saisonalität aufweist.



© elab2go // Kaiserslautern, 2022

Vorbereitung: Sprachen und Tools bereitstellen

- Programmiersprache: **Python**
- Entwicklungs- und Paketverwaltungsplattformen
  - **Anaconda**, **Spyder** und **Jupyter Notebook**

- Das Modell (ein tiefes LSTM-Netzwerk) wurde zunächst mit der Entwicklungsumgebung Spyder entwickelt, mit Hilfe der Python-Pakete Tensorflow, Keras und Pandas.
- Das funktionierende Modell wurde danach auch als Jupyter Notebook erstellt und mit Markdown-Erläuterungen versehen.

© elab2go // Kaiserslautern, 2022

Python als Programmiersprache für Datenanalyse

https://www.elab2go.de/demo-py1/

Die Programmiersprache **Python** wurde um 1990 in Amsterdam entwickelt. Die aktuelle Version Python 3.0 wird vor allem im Umfeld des Machine Learning verwendet, aber auch bei der Programmierung eingebetteter Systeme (Raspberry Pi).

Python kommt in Kombination mit der Plattform **Anaconda** (für Anwendungs- und Paketverwaltung) und **Spyder** oder **Jupyter Notebook** als Entwicklungsumgebungen zum Einsatz.

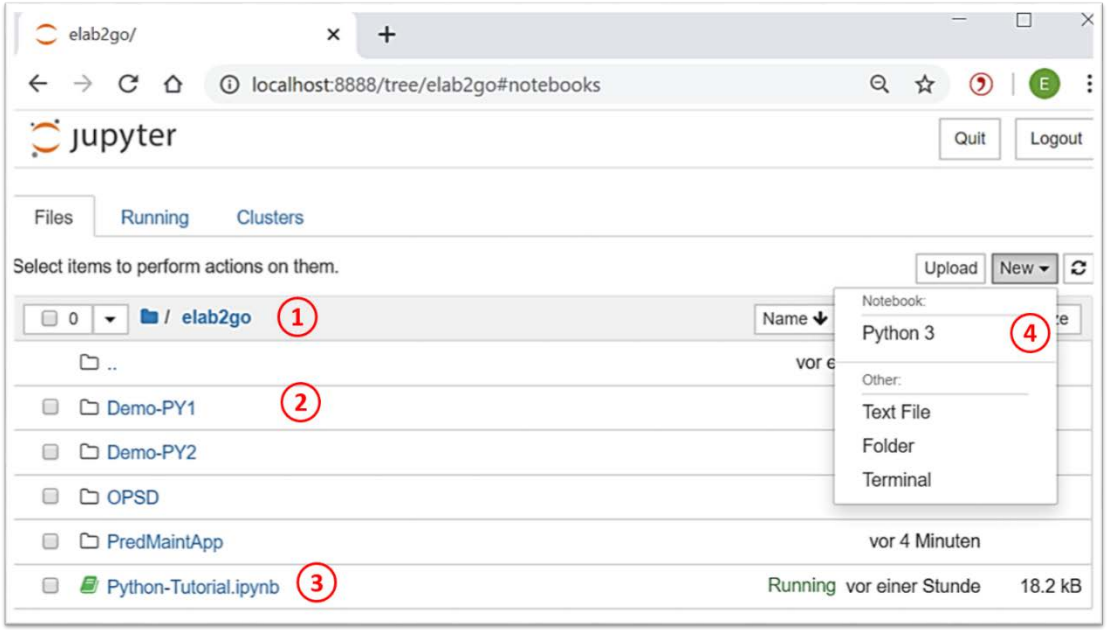
Vorteil: Leichter Einstieg in die Datenanalyse durch Verwendung der kostenlosen Python-Programmpakete für Datenverwaltung, -Modellierung und -Analyse.

© elab2go // Kaiserslautern, 2022

Jupyter Notebook als Lernumgebung

https://www.elab2go.de/demo-py1/jupyter-notebooks.php

Jupyter Notebook ist eine webbasierte Umgebung, die das Erstellen, Dokumentieren und Teilen von Tutorials und Demonstratoren unterstützt, und zwar insbesondere im Umfeld der Datenanalyse.

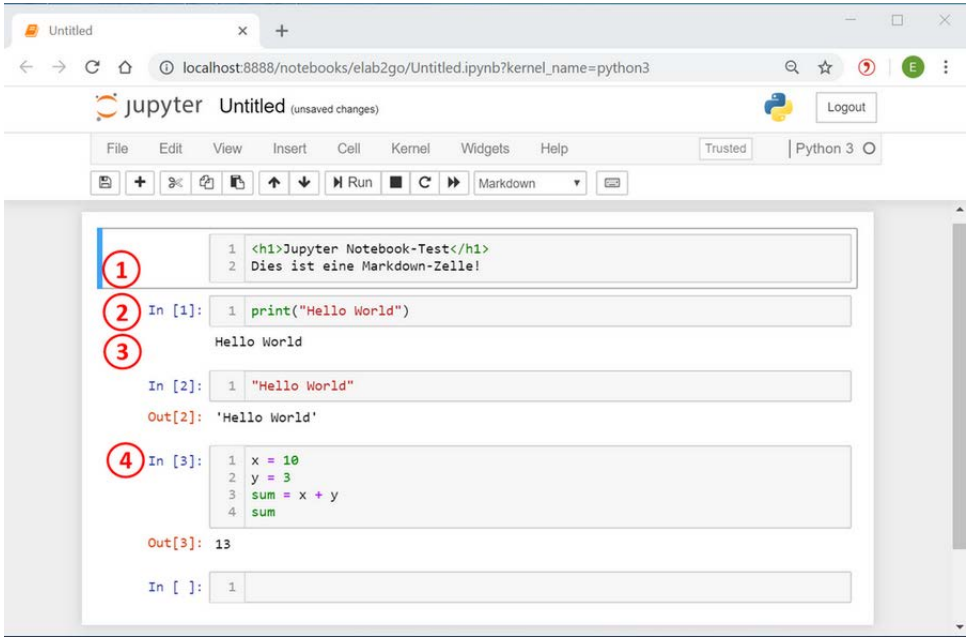


© elab2go // Kaiserslautern, 2022

Jupyter Notebook: Funktionalität: Dokumentierten Code schreiben, ausführen und teilen

In einem Jupyter Notebook kann man Code schreiben und ausführen, Daten visualisieren, und diesen Code auch mit anderen teilen. Code und Markup (Beschreibung des Codes) werden in unabhängige Zellen geschrieben und können separat ausgeführt werden.

Notebooks können in andere Formate exportiert werden, z.B. als HTML, LaTeX, PDF, Slideshow.



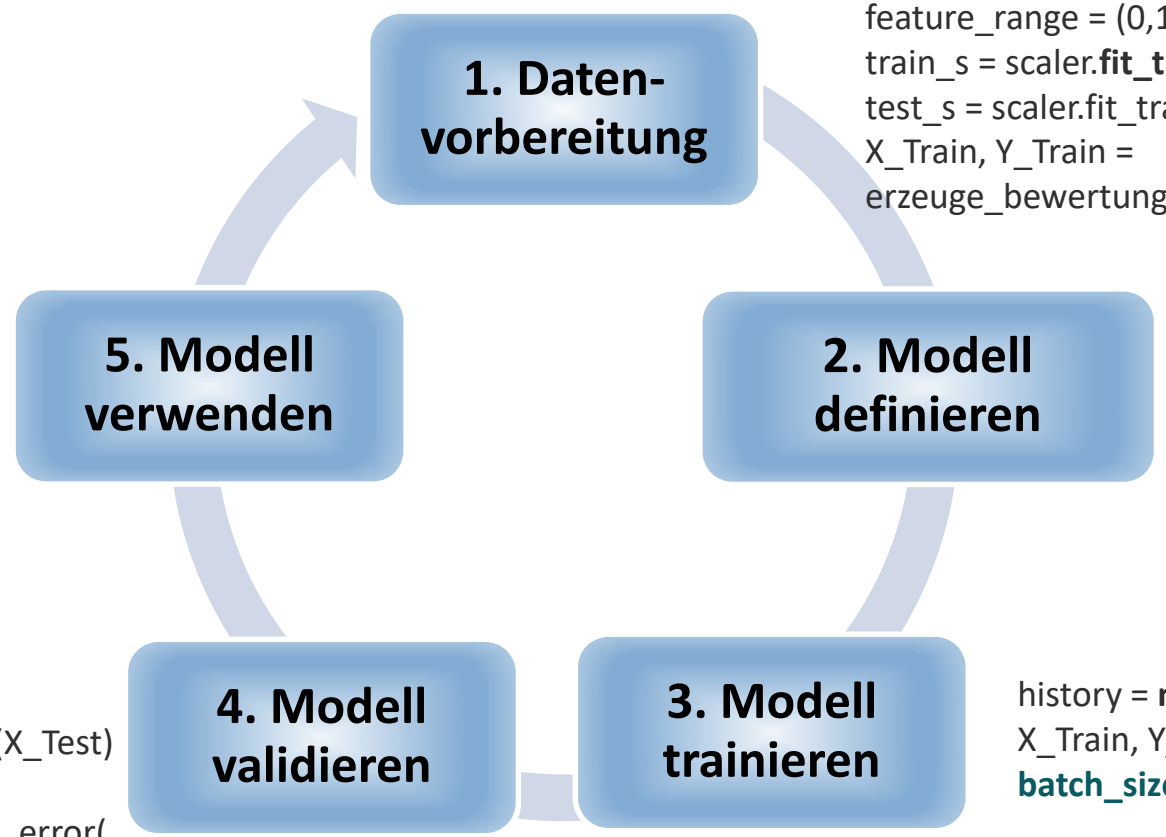
© elab2go // Kaiserslautern, 2022

Python-Bibliotheken für Maschinelles Lernen

- **Tensorflow** ist ein von Google entwickeltes Framework für maschinelles Lernen und künstliche Intelligenz, das unter Open-Source-Lizenz verfügbar ist.
- **Keras** ist eine Open-Source-Python-Bibliothek zum Entwickeln und Bewerten von Machine Learning-Modellen, die als benutzerfreundliche Schnittstelle zu den Frameworks Tensorflow und Theano verwendet wird.
- **Pandas** bietet leistungsstarke, benutzerfreundliche Datenstrukturen und Funktionen für den Zugriff auf Datentabellen.

© elab2go // Kaiserslautern, 2022

Umsetzung in Python: compile, fit, predict




```
scaler = MinMaxScaler(
feature_range = (0,1) )
train_s = scaler.fit_transform(train)
test_s = scaler.fit_transform(test)
X_Train, Y_Train =
erzeuge_bewertung(train_s, 7)
```

```
model = Sequential()
model.add(layer1);
model.add(layer2);
model.compile(
optimizer = "adam",
loss = "mse"])
```

```
history = model.fit(
X_Train, Y_Train, epochs=400,
batch_size=64)
```

```
Y_Pred =
model.predict(X_Test)
mse =
mean_squared_error(
Y_Test, Y_Pred)
```

© elab2go // Kaiserslautern, 2022

Schritt 1-1: Datenvorbereitung: Aufteilung

10% der Trainingsdaten werden für die Validierung verwendet.

```
1  VALIDATION_SPLIT = 0.1
2  train_size = int(len(series) * (1-VALIDATION_SPLIT))
3  test_size = len(series) - train_size
4  train = series.iloc[0:train_size,:]
5  test = series.iloc[train_size:len(series)]
```

Trainingsdaten:

Datum	Verbrauch
2016-01-01	1007.11
2016-01-02	1147.89
...	...
2019-08-05	1278.04
2019-08-06	1315.79

1314 rows × 1 columns

Testdaten:

Datum	Verbrauch
2019-08-07	1324.10
2019-08-08	1326.62
...	...
2019-12-30	1200.05
2019-12-31	1124.93

147 rows × 1 columns

Trainings- und Testdaten

© elab2go // Kaiserslautern, 2022

Schritt 1-2: Datenvorbereitung: Normierung

Skaliere die Daten auf den Wertebereich (0, 1)

Ggf. auch auf Standardnormalverteilung (Mittelwert 0, Stdabw. 1)

```
1 scaler = MinMaxScaler(
2                 feature_range = (0,1) )
3
4 # train_s: skalierte Trainingsdaten
5 train_s = scaler.fit_transform(train)
6
7 # test_s: skalierte Testdaten
8 test_s = scaler.fit_transform(test)
```

Trainingsdaten (unskaliert)

Datum	Verbrauch
2016-01-01	1007.11
2016-01-02	1147.89
2016-01-03	1130.63

Trainingsdaten (skaliert)

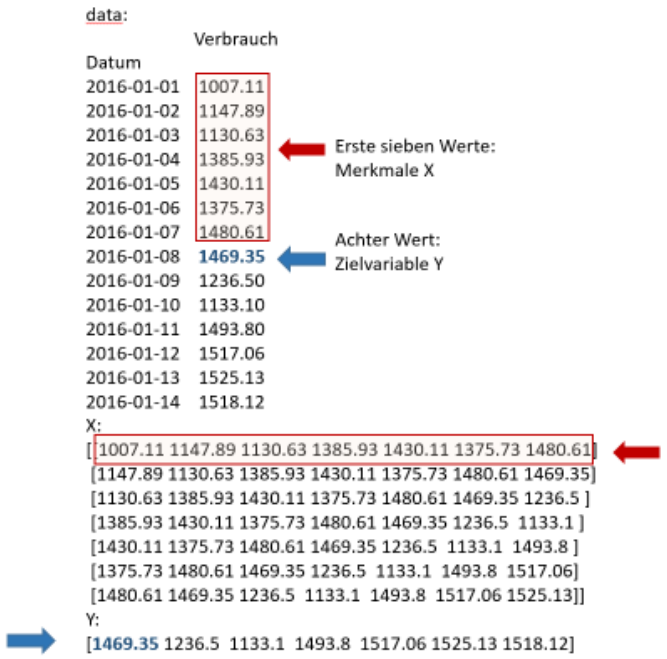
Datum	Verbrauch
2016-01-01	0.102986
2016-01-02	0.303656
2016-01-03	0.279054

© elab2go // Kaiserslautern, 2022

Schritt 1-2: Datenvorbereitung: Zielvariable

Das Problem muss als Modell des überwachten Lernens formuliert werden: Die Zeitreihe wird aufgeteilt in Merkmale X und eine Zielvariable Y, die die Bewertung der Merkmale enthält.

```
1  X_Train, Y_Train =
2  erzeuge_bewertung(train_s, 7)
3  X_Test, Y_Test =
4  erzeuge_bewertung(test_s, 7)
```



© elab2go // Kaiserslautern, 2022

Schritt 2: Modell erstellen: Code

Ein Modell wird erstellt, indem seine Netzwerk-Topologie festgelegt wird, d.h. die Anzahl, Art und Anordnung der Schichten, und die Verlustfunktion angegeben wird, die für das Ermitteln der optimalen Gewichtswerte verwendet werden soll.

```
model = Sequential()
# Füge erste LSTM-Schicht hinzu
layer = LSTM(units=10, input_shape=(7,1), return_sequences=True)
model.add(layer)
# Füge zweite LSTM-Schicht hinzu (...)
model.add(LSTM(units = 10, input_shape=(7,1)))
# Füge Dense-Schicht hinzu
model.add(Dense(units = 1))
# Erstelle das Modell
model.compile(optimizer = "adam", loss = "mse",
              metrics=['mean_squared_error'])
```

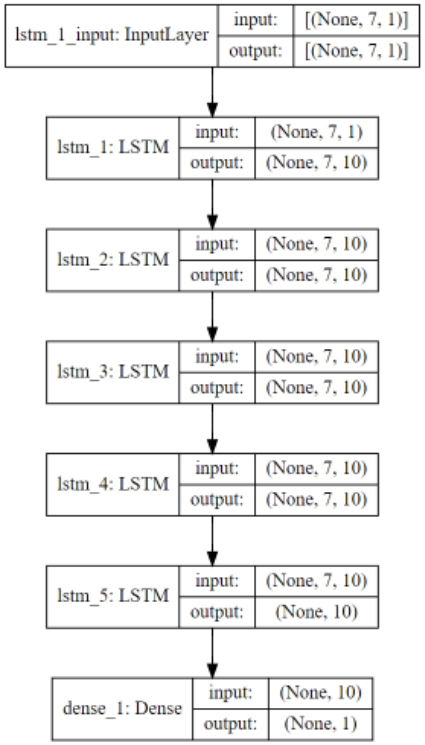
© elab2go // Kaiserslautern, 2022

Schritt 2: Modell erstellen: Ausgabe

Hier wird ein neuronales Netzwerk mit einer **Eingabeschicht**, einer **Ausgabeschicht** und fünf versteckten **LSTM-Schichten**. Als **Verlustfunktion** wird "mse" festgelegt, d.h. die mittlere quadratische Abweichung.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 7, 10)	480
lstm_2 (LSTM)	(None, 7, 10)	840
lstm_3 (LSTM)	(None, 7, 10)	840
lstm_4 (LSTM)	(None, 7, 10)	840
lstm_5 (LSTM)	(None, 10)	840
dense_1 (Dense)	(None, 1)	11

Total params: 3,851
Trainable params: 3,851
Non-trainable params: 0



© elab2go // Kaiserslautern, 2021

Schritt 3: Modell trainieren

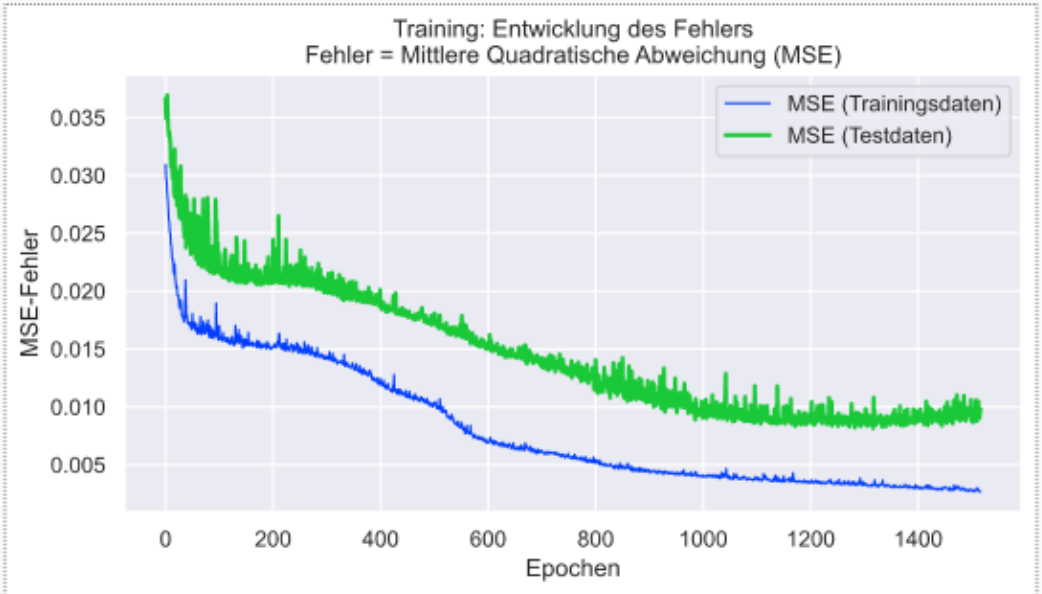
Ein Modell wird trainiert, indem unter Verwendung des Optimierungsalgorithmus der Wert der Verlustfunktion während der Trainingsphase minimiert wird. Das Trainieren des Modells geschieht mit Hilfe der Funktion **fit()**.

```
# Erstelle Callback für Stop-Kriterium
from keras.callbacks import EarlyStopping
cb_stop = EarlyStopping(monitor='val_loss', mode='min',
                        verbose=1, patience=200)
# X_Train erhält eine zusätzliche Dimension
X_Train = np.reshape(X_Train,
                     (X_Train.shape[0], X_Train.shape[1], 1))
# Trainiere das Modell mit Hilfe der Funktion fit()
history = model.fit(X_Train, Y_Train,
                    epochs=4000, batch_size=64,
                    validation_split=0.1, verbose=2,
                    callbacks=[cb_stop])
```

© elab2go // Kaiserslautern, 2021

Schritt 3: Modell trainieren: Visualisierung der Trainingsphase

Der MSE-Fehler in den Trainingsdaten sinkt und erreicht schließlich ein stabiles Plateau. Der MSE-Fehler der Testdaten fällt bis zu einer Epoche E1 gleichzeitig mit dem MSE der Trainings-daten, steigt dann wieder, hier greift das Stopp-Kriterium.



Entwicklung des MSE-Fehlers über die ersten 1400 Epochen

© elab2go // Kaiserslautern, 2021

Schritt 4: Modell validieren

Mit Hilfe des Testdatensatzes wird das Modell validiert, d.h. es wird mit Hilfe der Funktion predict() eine Prognose erstellt und für diese werden Performancemetriken (MSE, RMSE) ausgewertet.

```
# Vorhersage für skalierte Testdaten
Y_Pred = model.predict(X_Test)

# Reskaliere die Daten
y_Test = pd.DataFrame(scaler.inverse_transform(Y_Test))
y_Pred = pd.DataFrame(
    scaler.inverse_transform(Y_Pred))

# Berechne RMSE der Validierungsdaten
mse = mean_squared_error(y_Test, y_Pred)
rmse = np.round(np.sqrt(mse))
```

```
Validierungs-Fehler:

MSE:
 5181.14

RMSE:
 72.00
```

© elab2go // Kaiserslautern, 2021

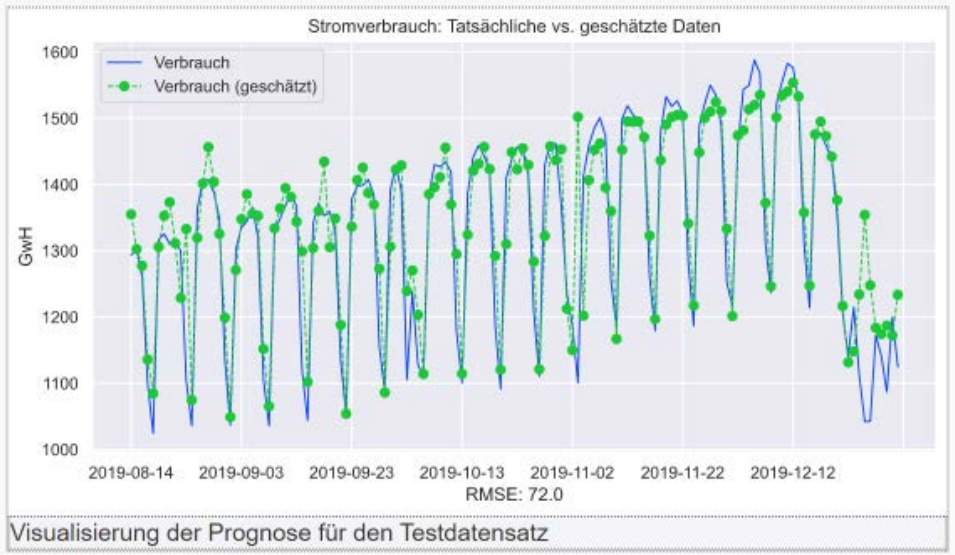
Schritt 4: Performance der Validierung visualisieren

Berechne die maximale Abweichung zwischen tatsächlichen und geschätzten Testdaten und visualisiere diese.

	y_Test	y_Pred	Error	Error (%)
	0	0		
RMSE: 72.0				
2019-08-14	1293.6	1355.1	61.4	4.7
2019-08-15	1302.1	1302.7	0.5	0.0
2019-08-16	1261.1	1277.6	16.5	1.3
...	...	...	...	...
2019-12-29	1087.1	1187.2	100.1	9.2
2019-12-30	1200.0	1172.1	27.9	2.3
2019-12-31	1124.9	1233.7	108.8	9.7

140 rows × 4 columns

Fehlertabelle für die Validierung mit RMSE, absolutem und relativem Fehler

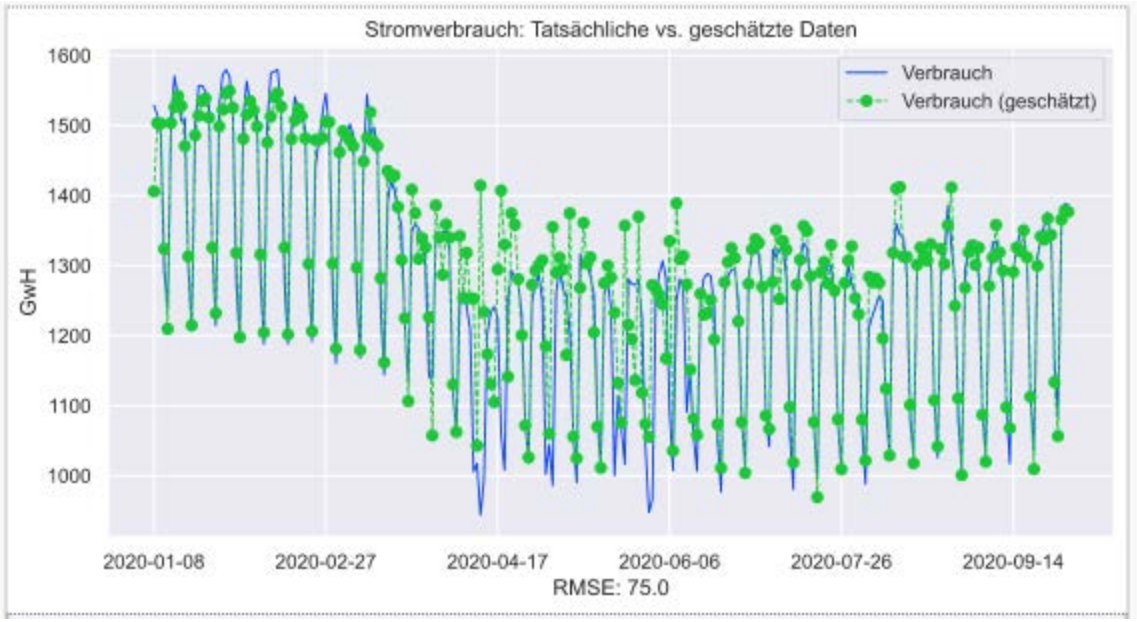


Visualisierung der Prognose für den Testdatensatz

© elab2go // Kaiserslautern, 2021

Schritt 5: Prognose für 2020

Die Prognose wird mit Hilfe der Funktion predict erstellt, ähnlich wie bei der Validierung. Performance-Metriken und Visualisierung zeigen eine akzeptable Prognose, mit einem RMSE-Fehler von 75 der wie erwartet größer ist als der in den Validierungsdaten.



© elab2go // Kaiserslautern, 2021

Konfigurationsparameter auf einen Blick

- **Konfigurationsparameter einer Schicht**
  - Dimension des Eingabe-Vektors: input_shape=(7,1)
  - Ausgabeeinheiten einer Schicht: Units: 10 / 1
  - Ausgabe versteckter Schichten in früheren Schichten wieder verwenden?
    - return_sequences=True / False
- **Konfigurationsparameter der Modellebene**
  - Optimierungsalgorithmus: Der Adam-Algorithmus ist ein stochastisches Gradienten-Abstiegsverfahren
  - Verlustfunktion (loss)
    - Mittlere Quadratische Abweichung (engl. mean squared error, MSE)
- **Konfigurationsparameter der Trainingsphase**
  - Anzahl Epochen: 4000, mit Early Stopping-Regel
  - Größe eines Batch: 1 / 32 / 64

© elab2go // Kaiserslautern, 2021

Demo-PY5: Fazit

Bei der Verwendung Neuronaler Netzwerke für Zeitreihen-prognosen ist es wichtig, die statistischen Eigenschaften der Zeitreihe zu untersuchen.

Die Güte des Modells wird von der Konfiguration des Netzes (Anzahl der Schichten und Ausgaben einer Schicht) beeinflusst, sowie der Einstellung der Konfigurationsparameter für die einzelnen Schichten, das Modell, und die Trainingsphase.

Die optimale Anzahl der Epochen / Iterationen für ein gutes Modell, das weder unter- noch überangepasst ist, kann mit Hilfe eines Stop-Kriteriums herausgefunden werden.

© elab2go // Kaiserslautern, 2021

Zusammenfassung

Maschinelles Lernen ist die Entwicklung lernfähiger Systeme:

- durch **überwachtes / unüberwachtes Lernen**
- mit unterschiedlichen Modellen: **Entscheidungsbaum, Clusteranalyse, Künstliche Neuronale Netze, Deep Learning**
- Programmiersprachen und Tools: **Python, R, MATLAB (…)**

Auf elab2go decken wir aktuell die wichtigsten Tools mit zwei Anwendungsfällen aus dem Automotive- und Energiesektor ab.
Weitere Erläuterungen und Demos sind auf elab2go zu finden.

Wir freuen uns über Ihr Feedback!

© elab2go // Kaiserslautern, 2022

elab2go: Demos und Tutorials rund um Python

- Python-Tutorial
  - https://www.elab2go.de/demo-py1/
- Jupyter Notebook verwenden
  - https://www.elab2go.de/demo-py1/jupyter-notebooks.php
- Jupyter Notebook Widgets erstellen
  - https://www.elab2go.de/demo-py1/jupyter-notebook-widgets.php

- Datenanalyse und Viualisierung mit Pandas
  - https://www.elab2go.de/demo-py2
- Predictive Maintenance mit scikit-learn (Entscheidungsbaum, Automotive)
  - https://www.elab2go.de/demo-py4/
- Machine Learning mit Python (Künstliches Neuronales Netzwerk, Energie)
  - https://www.elab2go.de/demo-py5/

© elab2go // Kaiserslautern 2021

Weitere Informationen

Kontakt elab2go Mobile Engineering Lab

Prof. Dr. Eva Maria Kiss
Studiengangsleiterin Elektrotechnik
Offene Digitalisierungsallianz Pfalz

Hochschule Kaiserslautern
Fachbereich Angewandte Ingenieurwissenschaften
Schoenstrasse 11, 67659 Kaiserslautern

E-Mail:
evamaria.kiss(at)hs-kl.de

www.elab2go.de

© elab2go // Kaiserslautern, 2022