RNN & LSTM

Time Series Forecasts

KI in der Robotik

Author: Matthias Titze - s0563413 | Betreuer: Patrick Baumann M.Sc.

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Überblick

0 Motivation: Intelligente Steuerung von Energienutzung

Ziel: intelligente Steuerung von Energieverbrauch

- Regenerative Energieproduktion (durch PV-Anlagen) ist sehr volatil.
- Es besteht eine Interesse, die produzierte Energie direkt zu nutzen.
- Wirtschaftlicher Vorteil: Die Einspeisung ins Netz bringt weniger Gewinn.
- ➳ Verbrauch intelligent steuern, d.h. zeitlich entzerren oder umschichten
- Bsp.: Ladevorgänge bei E-Fahrzeugen gemäß Stromproduktion planen.

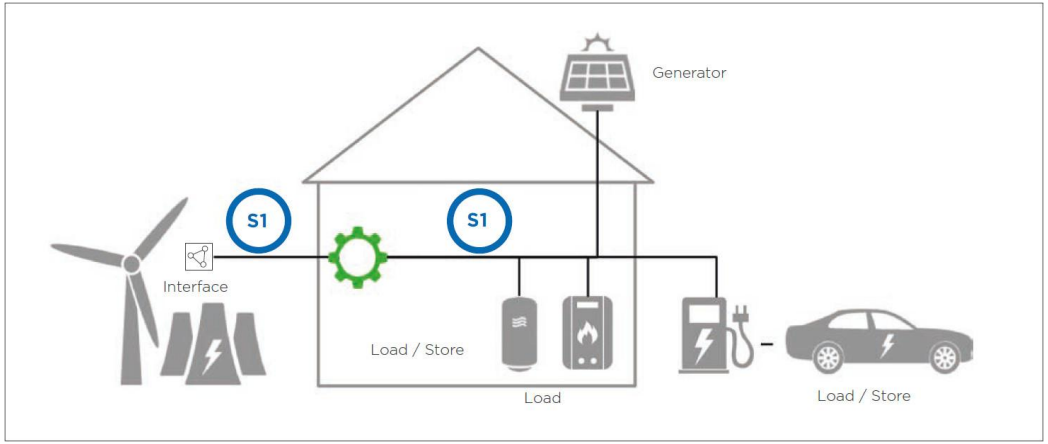


Abb. 1: Solarstrom und Elektromobilität

0 Motivation: Prognose PV-Production und Stromverbrauch

PV-Produktion:

- Für die Planung muss die PV-Stromproduktion über die Zeit vorausgesagt werden.
- Solarstrom ist stark abhängig vom Wetter - Daten dazu sind auch Vorhersagen.

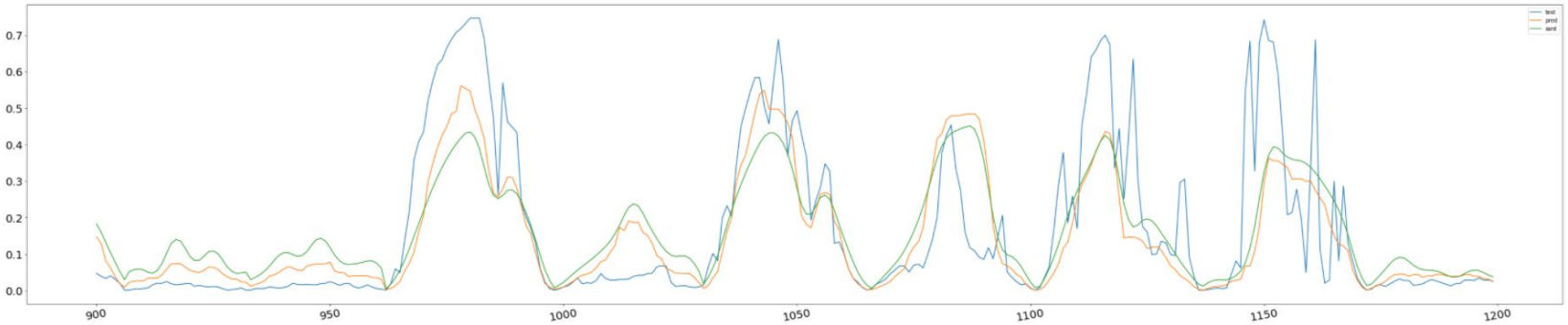
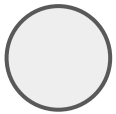


Abb. 2: Prognosekurve PV-Produktion

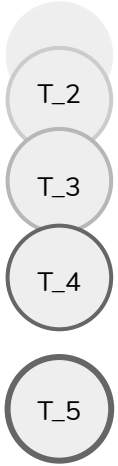
Stromverbrauch (Hausanschluss):

- Auch der erwartete Stromverbrauch im Quartier muss für die Planung bekannt sein.
- Bei Bürogebäuden hängt dieser primär von den Aktivitätszyklen bzw. Arbeitszeiten ab.

1 Zeitreihen & Sequenzen

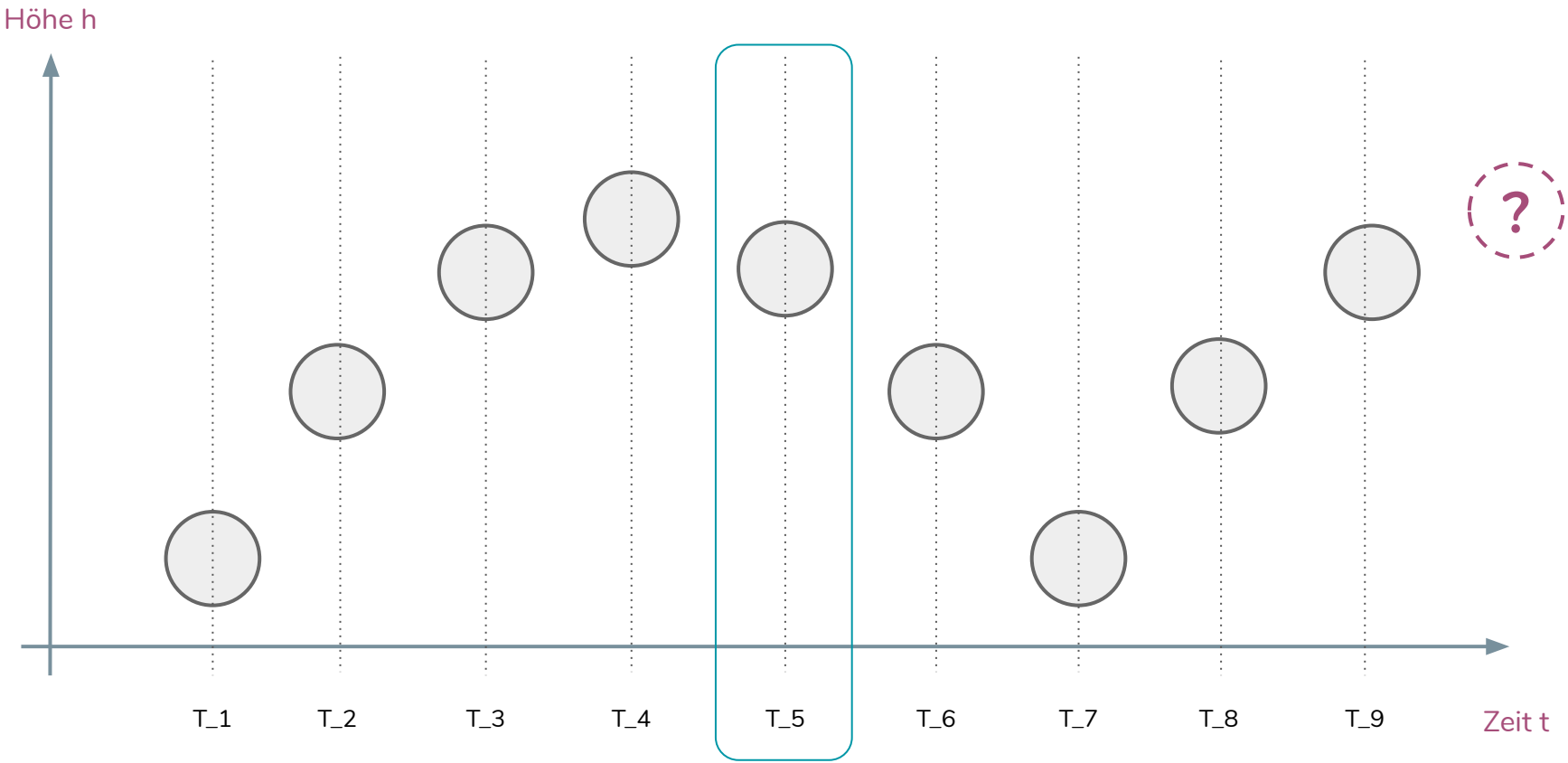


Fällt der Ball ?

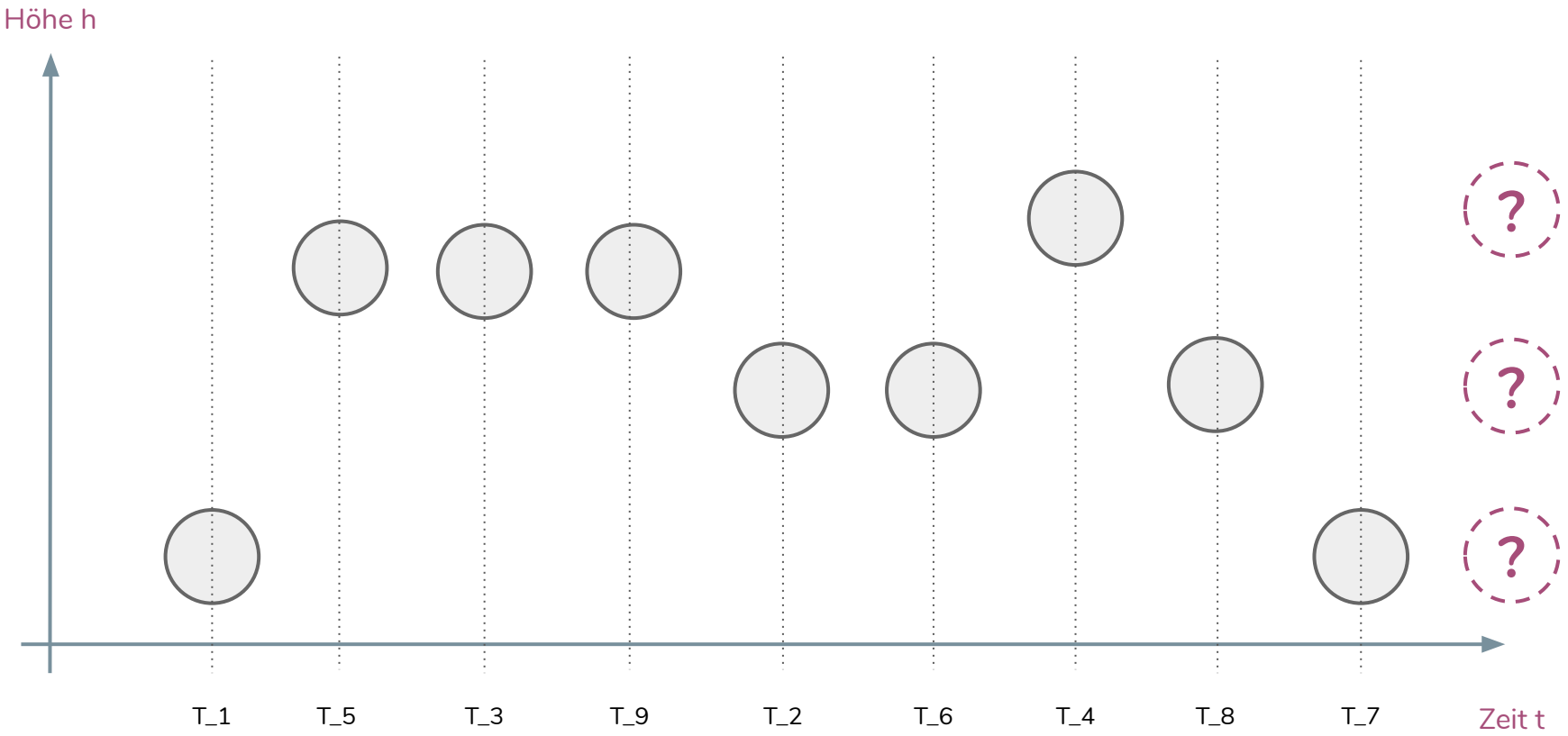


Nun, fällt der Ball ?

1 Problemstellung: Zeitreihen und Sequenzen



1 Problemstellung: Zeitreihen - nicht sequenziell verarbeitet



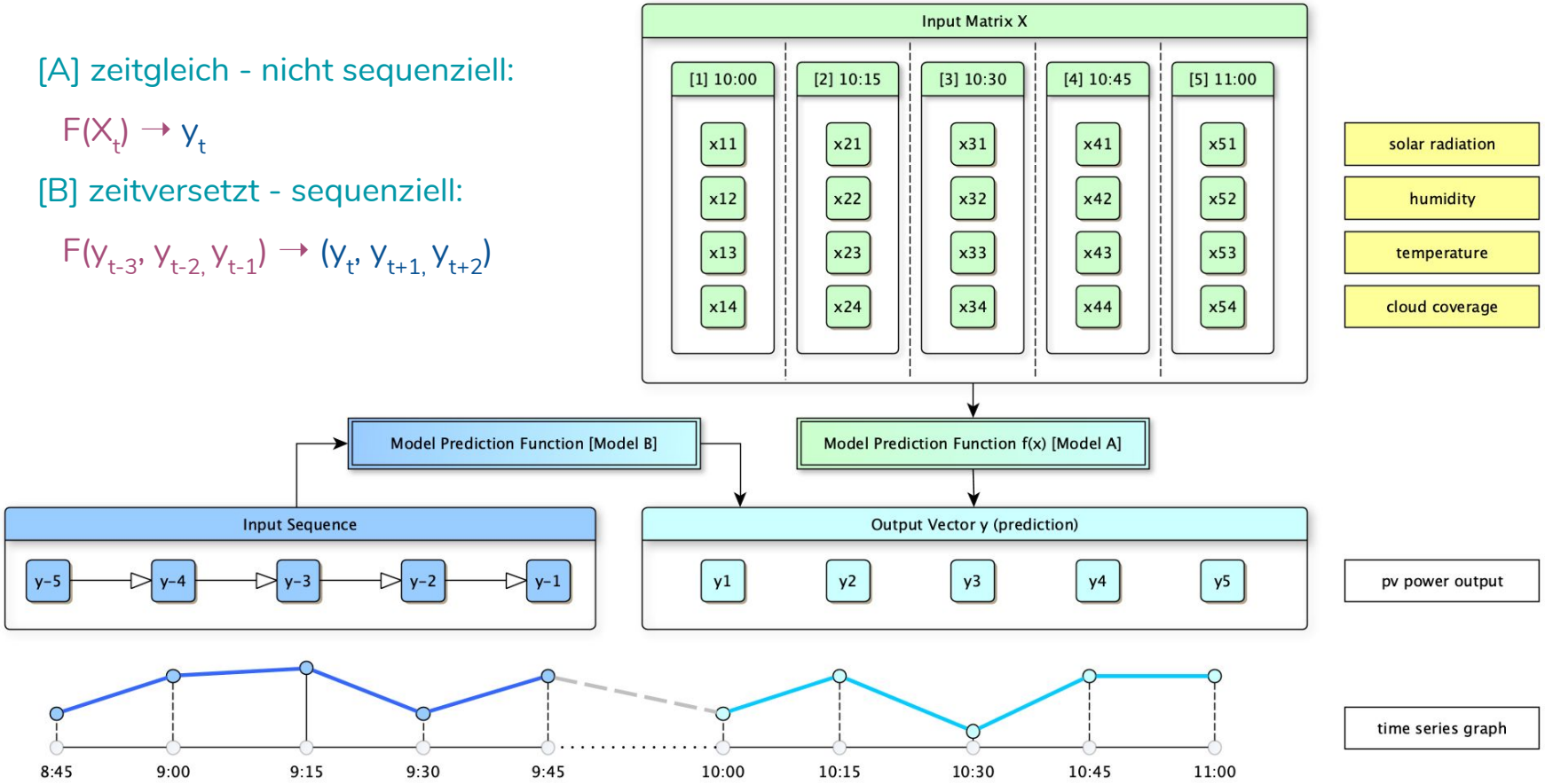
1 Problemstellung: zeitgleich vs. zeitversetzt

[A] zeitgleich - nicht sequenziell:

$F(X_t) \rightarrow y_t$

[B] zeitversetzt - sequenziell:

$F(y_{t-3}, y_{t-2}, y_{t-1}) \rightarrow (y_t, y_{t+1}, y_{t+2})$



1 Problemstellung: Zeitreihen und Sequenzen

Zeitreihen: kartieren den Verlauf eines Features y über die Zeit t, also [Zeitstempel -> Wert]

Sequenz: zusätzliche Information liegt in der Abfolge von Datenpunkten verborgen

Sequenzielles Modell: kann den Abfolge-Zusammenhang zwischen den Datenpunkten nutzen

➳ Analyse historischer Sequenzen kann zur Prognose des weiteren Verlaufs

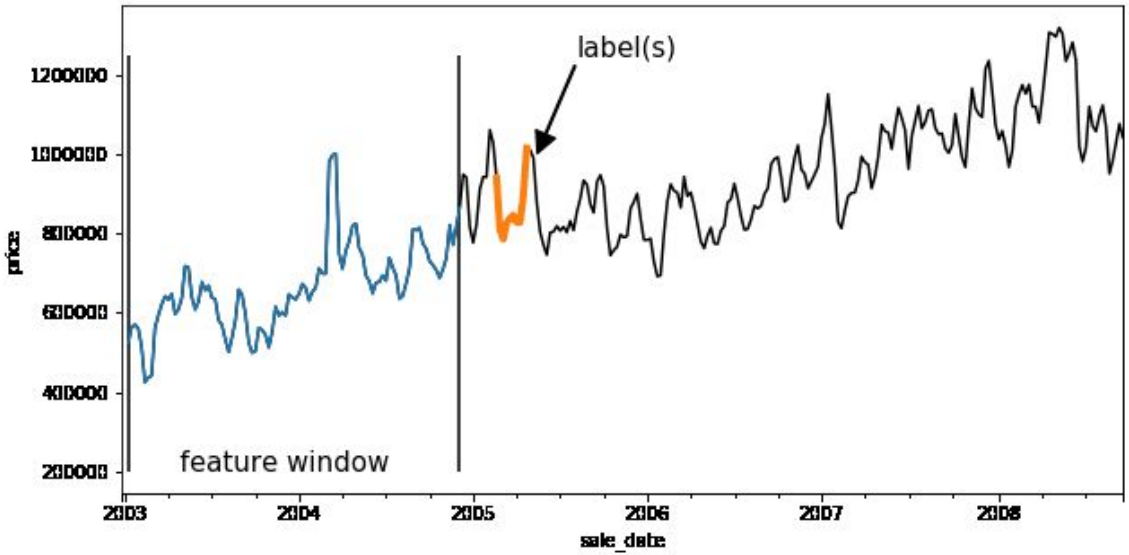


Abb. 3: Prognose von Feature X via historischer Verlauf von X

Beispiele:

- Börsenkurse
- Passagierzahlen
- Stromverbrauch

1 Problemstellung: Prognose Stromverbrauch via RNN, LSTM und RFR

Problemstellung Konkret

- Vergleich von Modellen zur Vorhersage des Stromverbrauchs eines Bürokomplexes

[A] Rekurrente Modelle

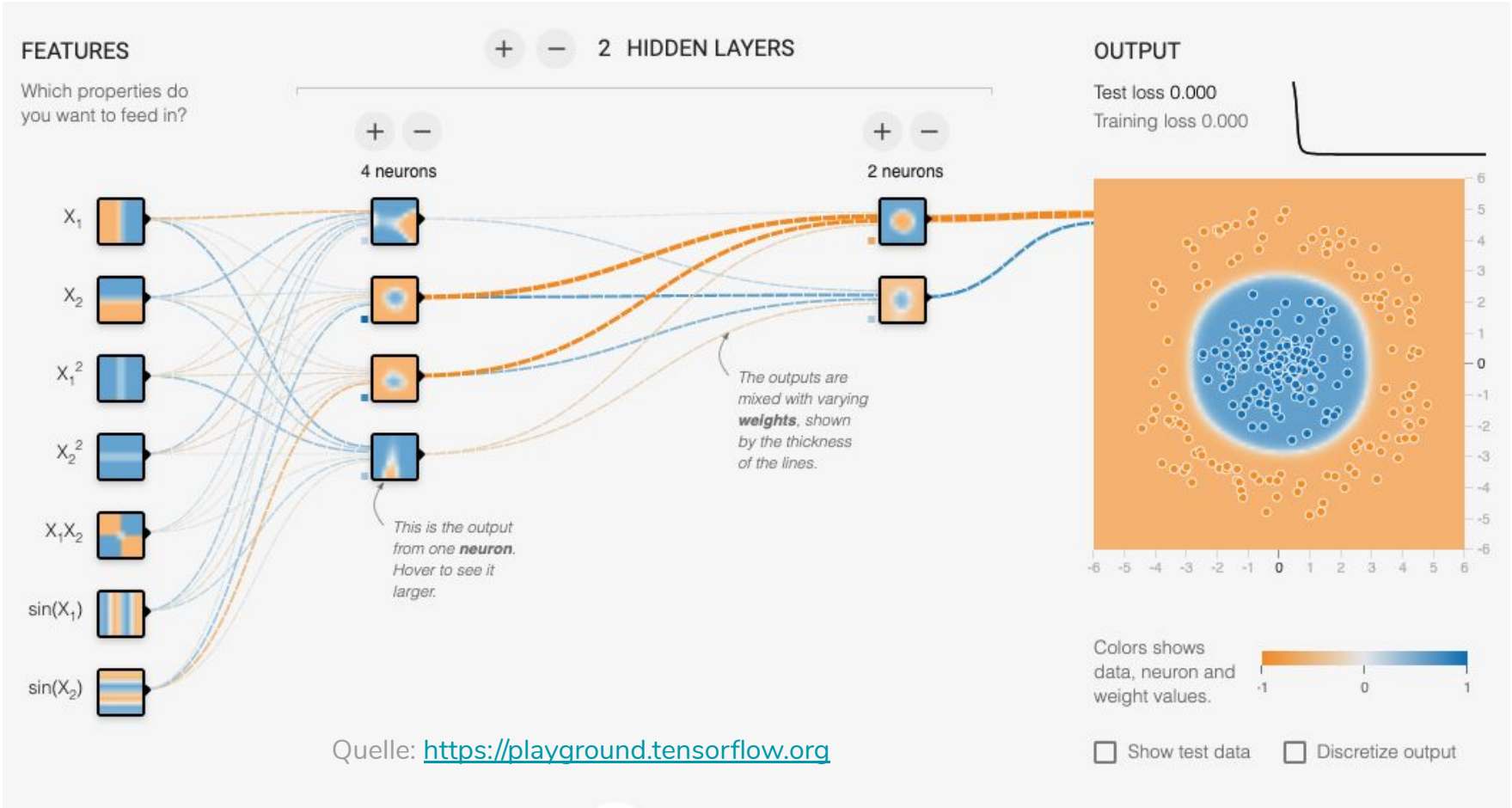
- Historische Stromverbrauchsdaten (Sequenzen) in die Zukunft fortschreiben
- RNN und LSTM in Bezug auf Präzision vergleichen
- Anmerkung: Dies sind die zeitversetzten Modell

[B] Klassische (zeitgleiche) Modelle

- Vergleichsexperiment mit zeitgleichem Modell: Random Forest Regressor
- Nutzung von Wetterdaten die auf Verbrauchsdaten schließen
- Spezifisches Feature-Engineering für verborgene Zeitinformationen

2 Grundlagen RNN & LSTM

2 Grundlagen: Feed-Forward-Netz (Zelle A)



Quelle: https://playground.tensorflow.org

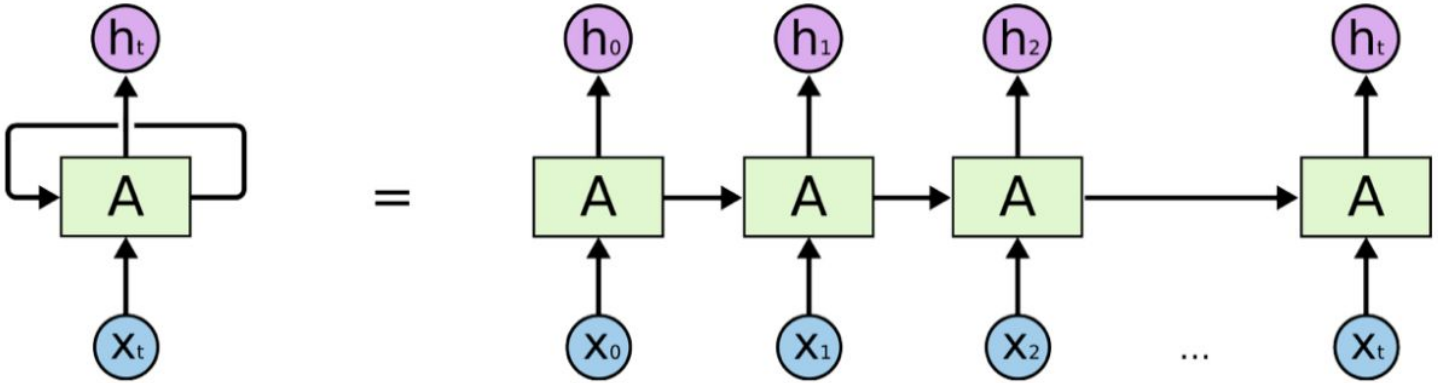


Abb. 3: RNN in zyklischer und entfalteter Systemdarstellung

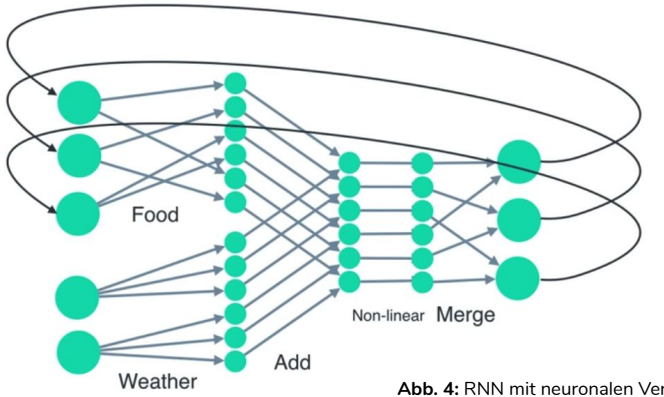


Abb. 4: RNN mit neuronalen Vernetzungen

$$o^t = f(h^t; \theta)$$

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

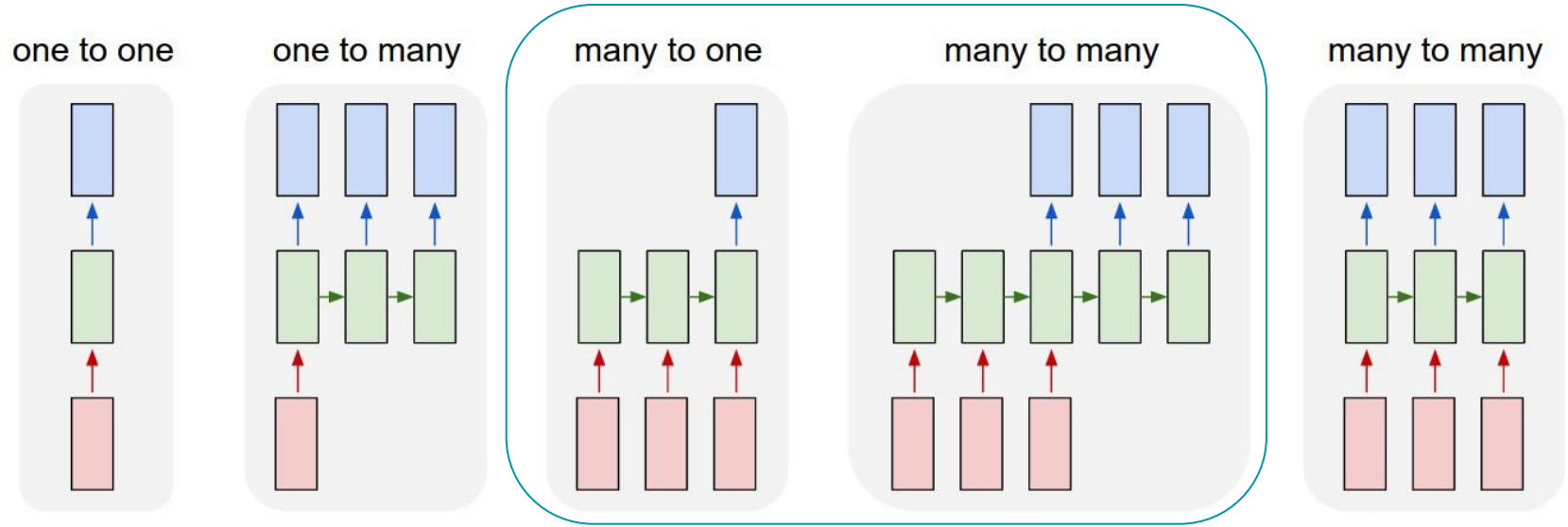


Abb. 5: Problemkonfigurationen bei RNNs

- Für rekurrente Netzwerke gibt es verschiedenste Anwendungsszenarien.
- Diese unterscheiden sich im Mapping der Anzahl von Eingabe und Ausgabewerten.
- Die Vielfalt führt schnell zu Verwirrung beim Aufbau rekurrenter Modelle. Vorsicht!

2 Grundlagen: Backpropagation über die Zeit (BPTT)

- Lernen erfolgt auch bei RNNs über den Gradientenabstieg auf der Kostenlandschaft.
- Dafür verwenden wir das Backpropagation Verfahren (Partielle Ableitungen & Kettenregel).
- Bei rekurrenten Netzen reichen die Neuronen-Ketten durch die Zeitstempel-Sequenz.
- Je länger die zu verarbeitende Sequenz, desto tiefer wird das Netz.

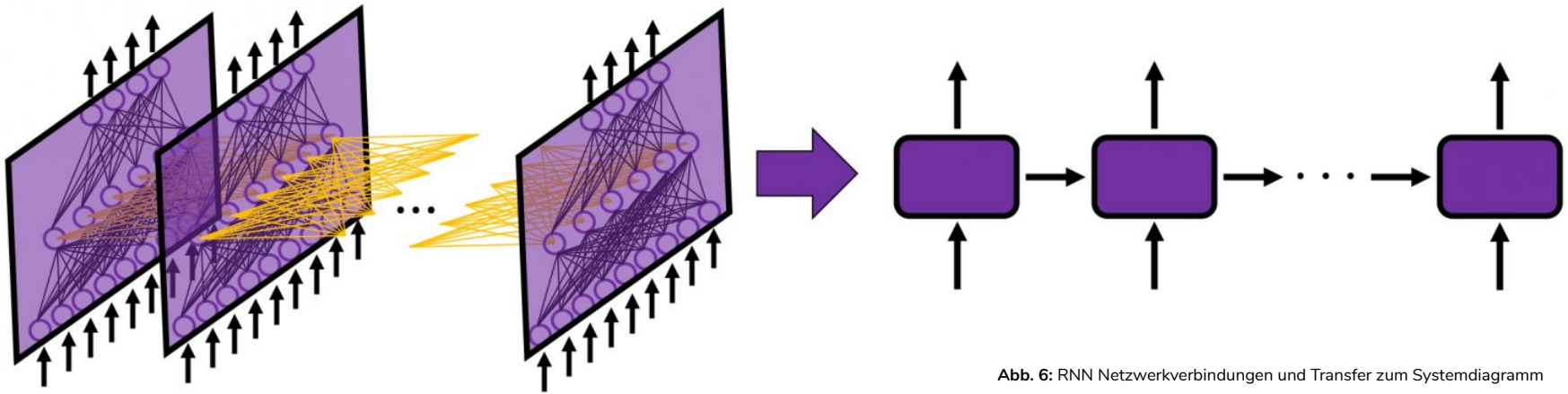


Abb. 6: RNN Netzwerkverbindungen und Transfer zum Systemdiagramm

2 Grundlagen: Vanishing Gradient

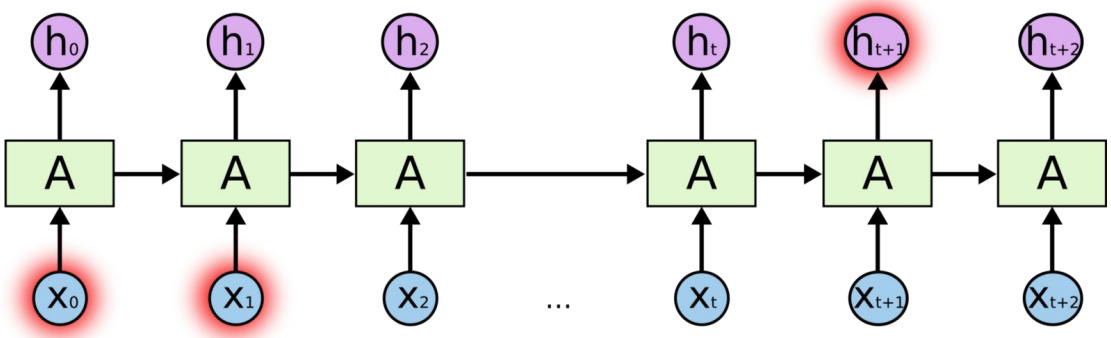


Abb. 7: RNN Vanishing Gradient

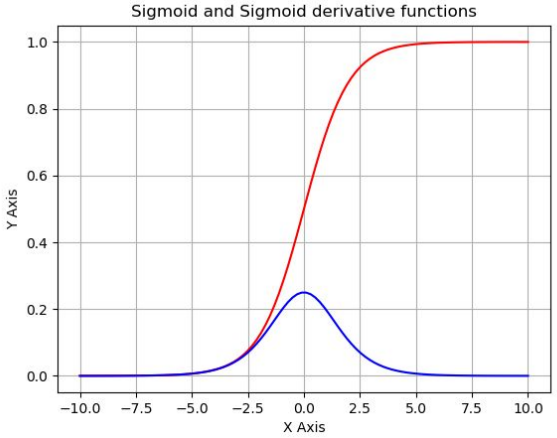


Abb. 8

- Bei BPTT entstehen sehr lange Ketten von Partiellen Ableitungen - durch die Sequenz.
- Bei sigmoiden Aktivierungsfunktionen ist die Ableitung aber sehr dicht bei 0 (< 0.3).
- Große Änderungen in den Eingabewerten erzeugen kleine Deltas in der Ausgabe.
- In der Kettenregel multiplizieren wir sehr oft wiederholt Werte [0 << 1].
- Daher geht der Gradient of sehr schnell gegen 0 und verschwindet.

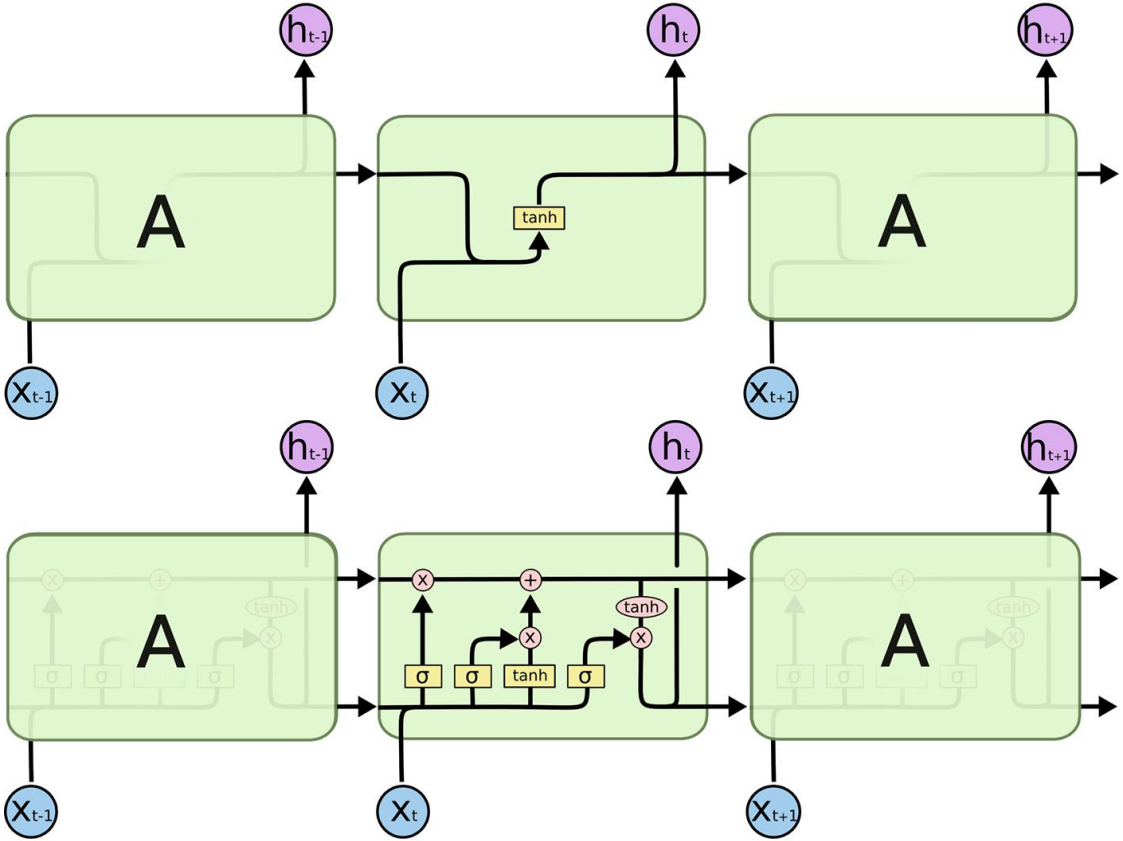
Lösungsansätze Vanishing Gradient:

- ändere Aktivierungsfunktion
- Residual Connections (Resnet)

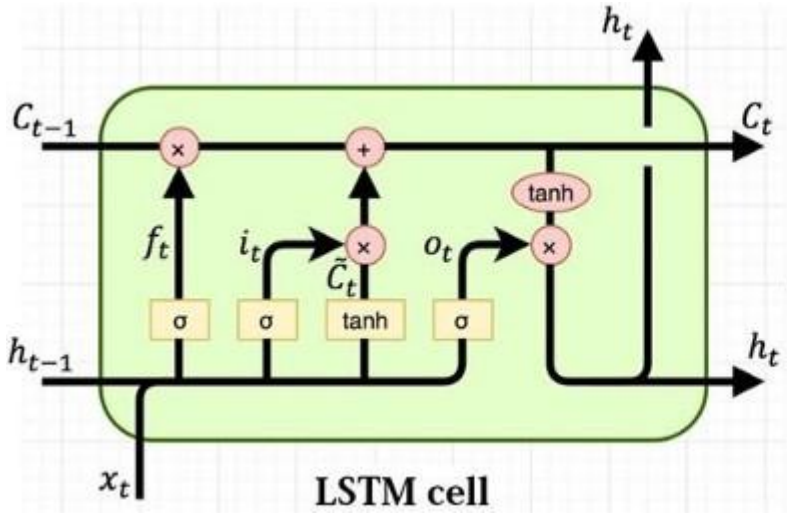
LSTM:

- Erweitert die RNN-Zelle
- Gedächtnis-Autobahn via zusätzlichem Cell-State C
- Hinzufügen und Löschen von Informationen aus C via Gate-Funktionen

Abb. 9: Verschaltung RNN vs. LSTM



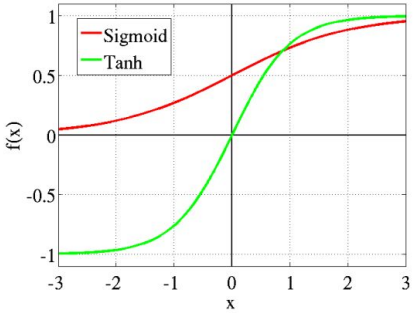
2 Grundlagen: LSTM-Zelle und Gates



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Abb. 10: LSTM-Zelle mit Gates als mathematische Funktionen

- Vergessen f: Multipliziere C mit Wert < 1 aus der Sigmoid-Funktion
- Erinnern i: Addiere Wert auf C (Produkt Sigmoid und tanh)
- Output o: Verknüpfe Cell-State C mit Input über Multiplikation
- Tangens hyperbolicus normalisiert, Sigmoid skaliert



3 Konzept & Daten

3 Konzept & Daten: Ausgangsdaten

Als Ausgangsdaten stehen folgende Zeitreihendaten zur Verfügung:

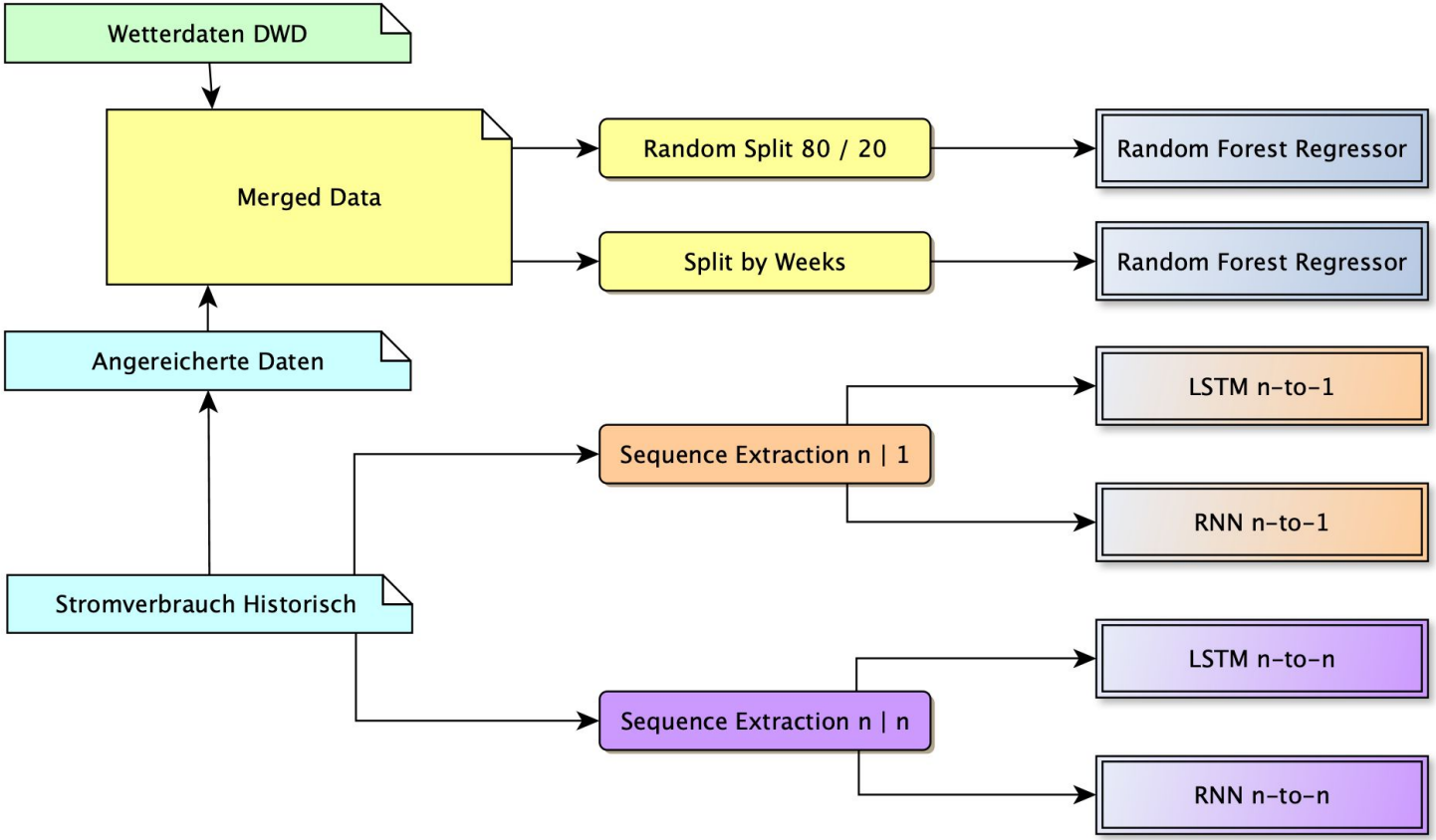
Wetterdaten:

- historische Wetterdaten von Deutschen Wetterdienst (DWD) für die Jahre 2018 - 2019
- archivierte Wettervorhersagen vom DWD für das erste Quartal 2020

Stromverbrauch Quartier PxC:

- historischer Stromverbrauch für den Bürokomplex (PxC) für die Jahre 2018 - 2019
- historischer Stromverbrauch für das erste Quartal 2020
- Diese Daten sind sequenzielle Zeitreihen, d.h. Kurven: [Zeitstempel, KW].

3 Konzept & Daten: Vergleichsexperimente



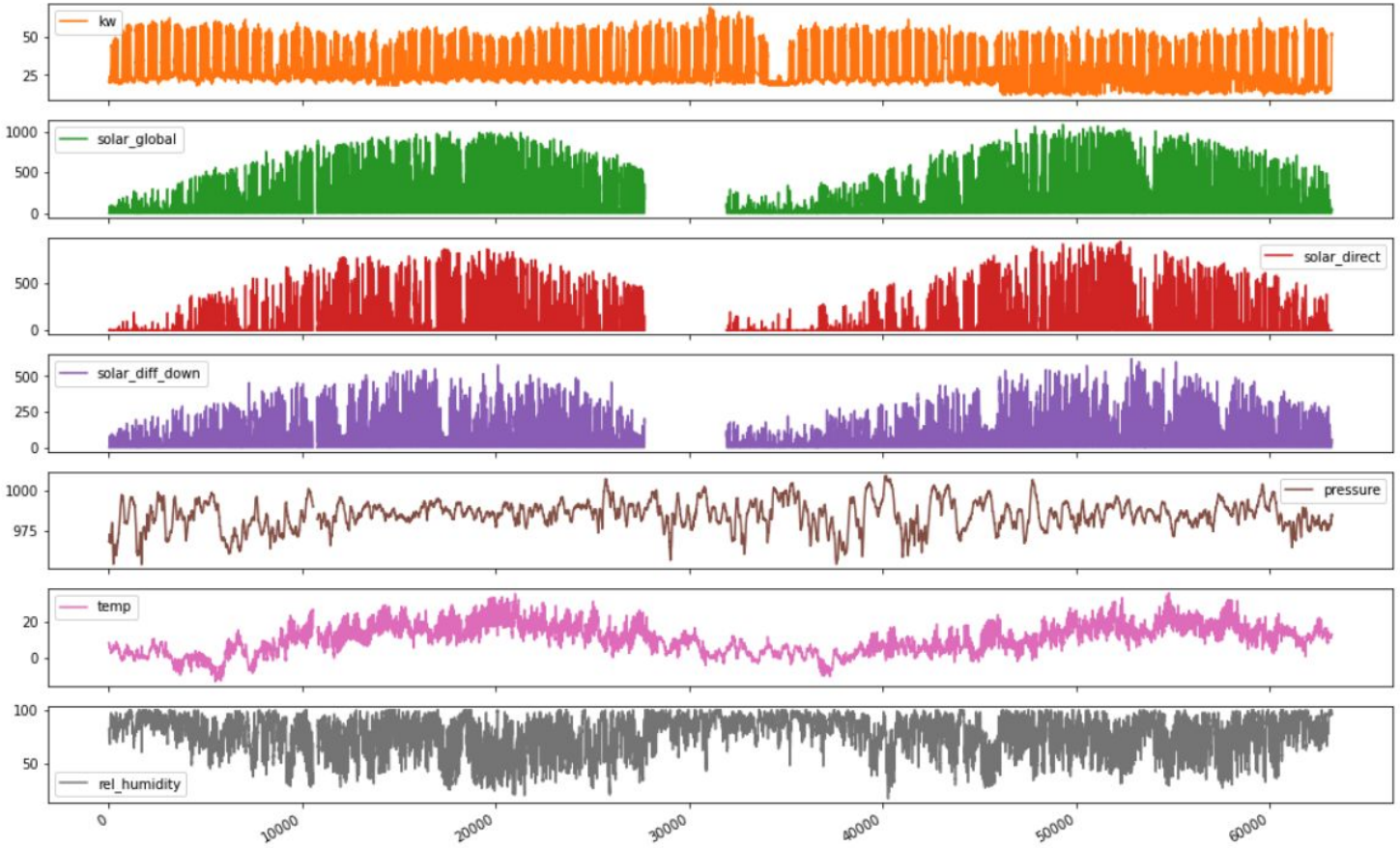


Abb. 11: Zeitreihendaten Wetter und Stromverbrauch

3 Konzept & Daten: Decision Tree & Random Forest Regressor

Abb. 12: Decision Tree

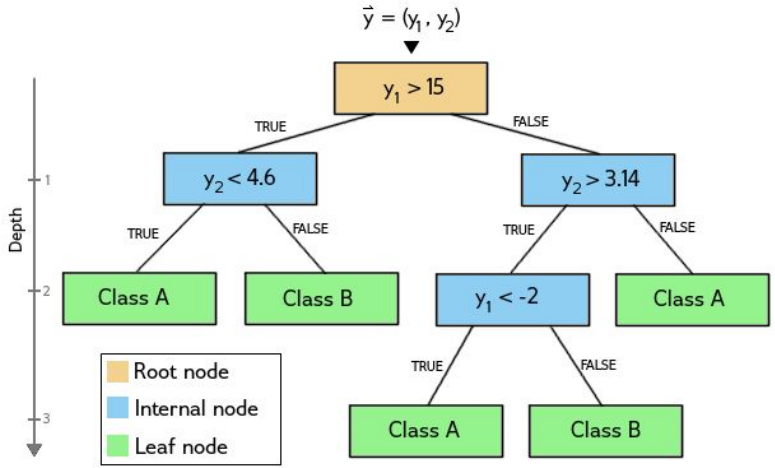
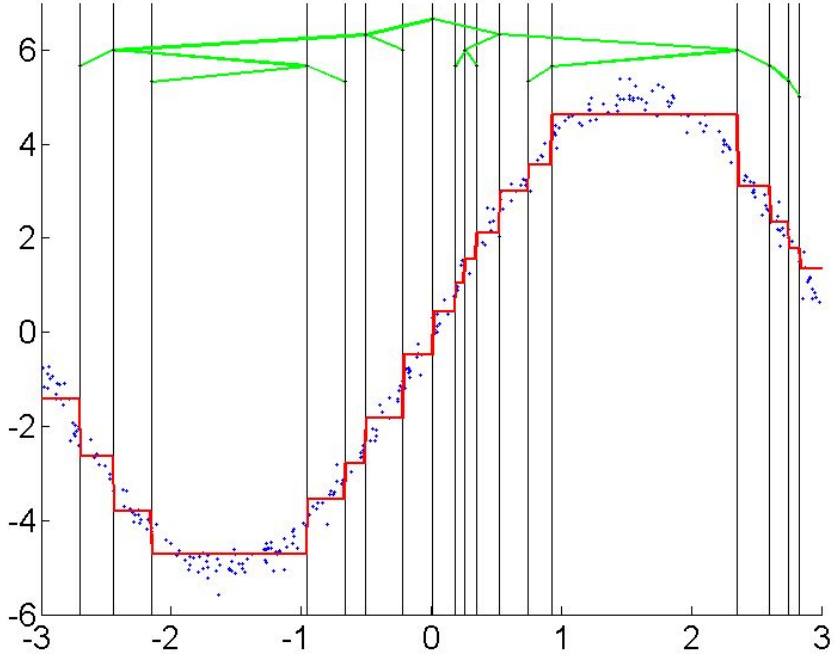
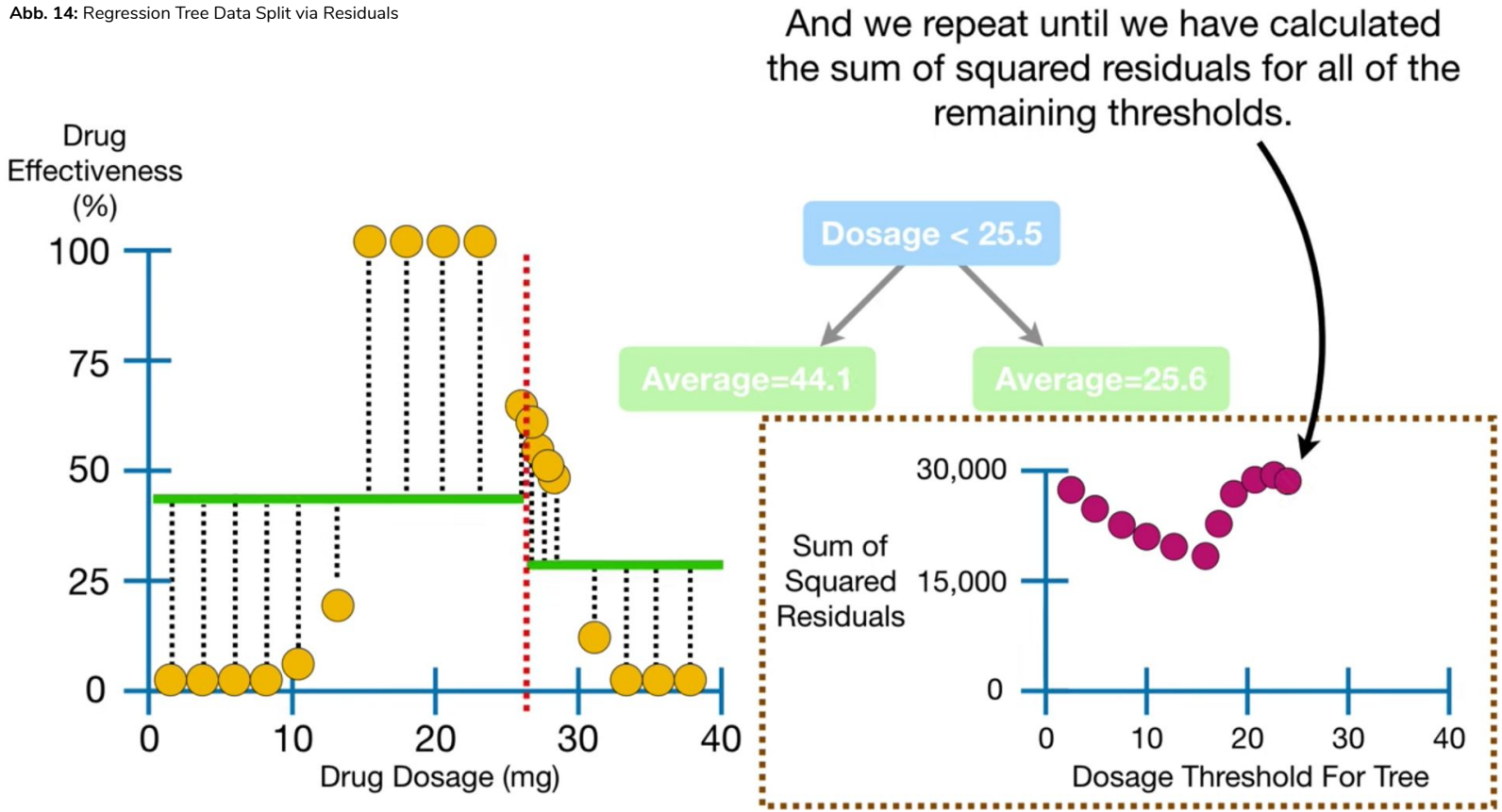


Abb. 13: Vorhersage kontinuierlicher Daten - Stufengraph



- Decision Trees nutzen wir eher für Klassifikationsprobleme
- Regression Trees eignen sich aber auch für Regressionsprobleme (kontinuierliche Daten) - etwas stufiger Prognose-Graph

Abb. 14: Regression Tree Data Split via Residuals



3 Konzept & Daten: Anreicherung der Daten für RFR

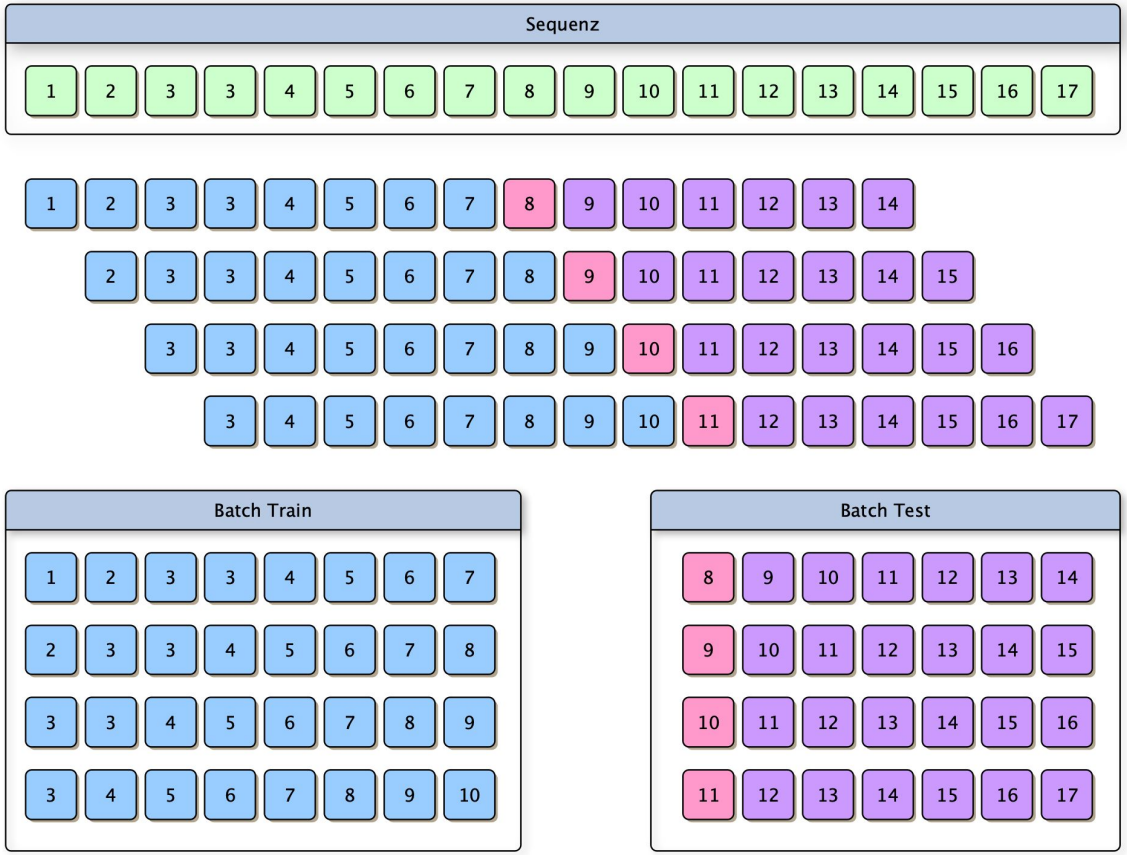
ts	kw	solar_global	solar_direct	solar_diff_down	pressure	temp	rel_humidity	month	day	hour	minute	weekday	holiday
2019-10-21 07:00:00	53.0	31.666667	0.0	31.666667	983.80	12.90	97.00	10	21	7	0	0	0
2019-10-21 07:15:00	51.0	28.333333	0.0	28.333333	984.25	13.25	95.25	10	21	7	15	0	0
2019-10-21 07:30:00	52.0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

Tabelle 1:: Auszug aus dem angereichertem Datensatz

- Der Zeitstempel bildet nur ein einziges Kontinuum und ist wenig informativ.
- Er verbirgt wiederkehrende Zyklen wie den Rhythmus der Wochentage oder die 24 Tagesstunden. Wir vermuten aber, dass diese Zusatzinformation interessant ist.
- Der Decision Tree Regressor soll über Zeitkomponenten, die Daten trennen.
- Daher ist es sinnvoll, Werte zu kreieren, die die zyklische Zeitinformation abbilden.

4 Implementation

4 Implementation: Erzeugung Input-Daten für RNNs



Aus der Time Series:

1. Teilsequenzen selektieren
2. Teilsequenzen in vorderen Trainingsabschnitt und hinteren Testabschnitt zerschneiden
3. Teile zu Batch-Matrizen zusammenschieben
4. Bei Many-To-One nur einen Datenpunkt nach der Trainingssequenz abtrennen

4 Implementation: Aufbau und Training RNN (many-to-one)

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

many-to-one

- Die Keras API stellt einen RNN-Layer und auch LSTM-Schichten zur Verfügung.
- Die inneren Vernetzung werden aber bei PyTorch offensichtlicher.
- LSTM hat mindestens 4x mehr Parameter.

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

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

4 Implementation: Aufbau und Training LSTM (many-to-many)

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

many-to-many

- Bei der Many-to-Many-Konfiguration muss am Ende eine Sequenz ausgegeben werden.
- Dafür benötigen wir den Time-Distributed-Layer damit der Dense-Layer für jeden Eingabeschritt wiederholt und von den vorangegangenen isoliert ausgeführt wird.
- Ohne ihn, würden wir eine Netz erhalten, die früheren Outputs, von späteren Berechnungen abhängig. Dies entspräche dann eher einem normalen FNN.
- Bei den Many-to-Many-Netzen wurden die Hidden-Layer experimentell variiert.
- Als Optimizer fand der Adam-Oprimzer über dem MSE Verwendung.

4 Implementation: Aufbau und Training Random Forest Regressor

```
# Train with split a.
regressor_a = RandomForestRegressor(n_estimators=n_estimators, max_depth=max_depth, random_state=random_seed)
regressor_a.fit(X_train_a, y_train_a)
```

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

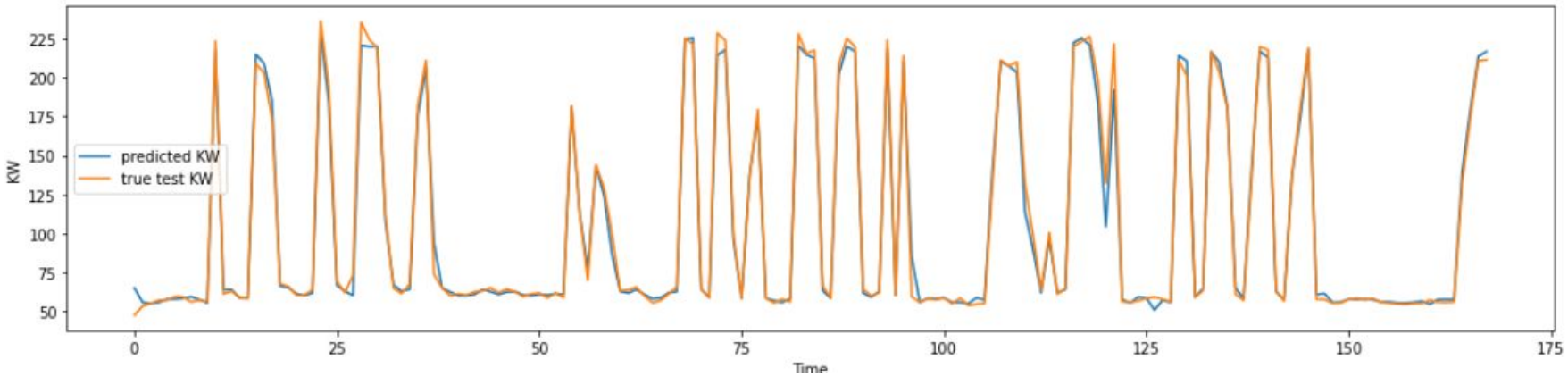
```
# Predict with split a.
y_pred_a = regressor_a.predict(X_test_a)
```

- Random Forest Regressor ist Teil des SciKit-Learn Frameworks.
- einfach mit wenigen Zeilen Code zu instanziieren
- Relativ wenige Hyperparamter - gute Ergebnisse auch ohne viel Tuning
- Trainiert erheblich schneller als Neuronale Netze
- entscheidend ist das Data-Engineering

5 Auswertung

5 Auswertung: Random Forest Regressor

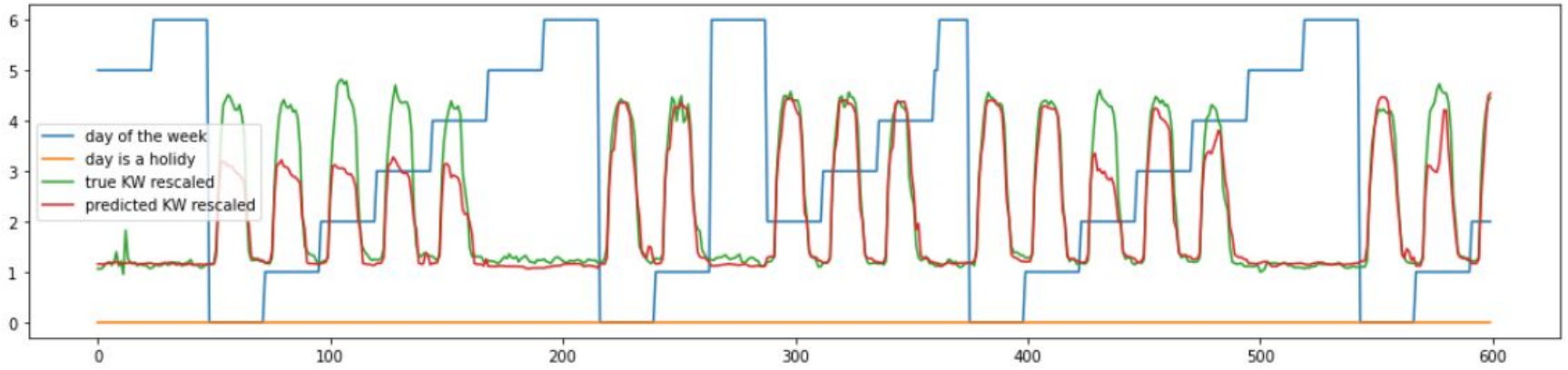
Abb. 15: Ausschnitt Plot Prognose RFR Random Split 80/20



- Die Vorhersagen des Random Forest Regressors sind sehr genau.
- Der RMSE ist nur 2,59 %.
- Bei zufälligen Samplen der Testdaten mit 80% / 20% Split lassen sich aber nicht die entscheidenden Zusammenhänge mit dem Verlauf der Werktage ausmachen.

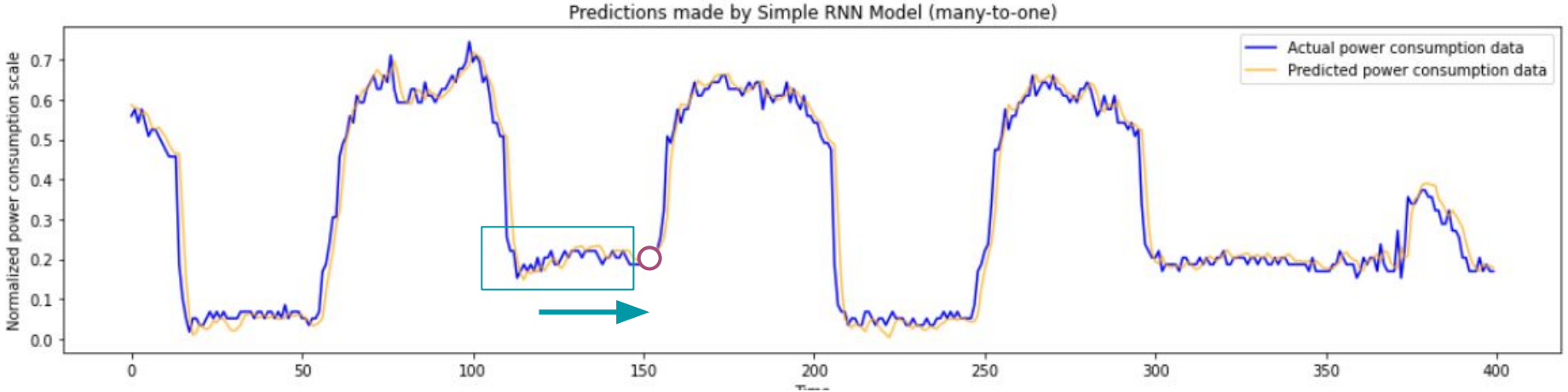
5 Auswertung: Random Forest Regressor

Abb. 16: Ausschnitt Plot Prognose RFR Split nach Wochen



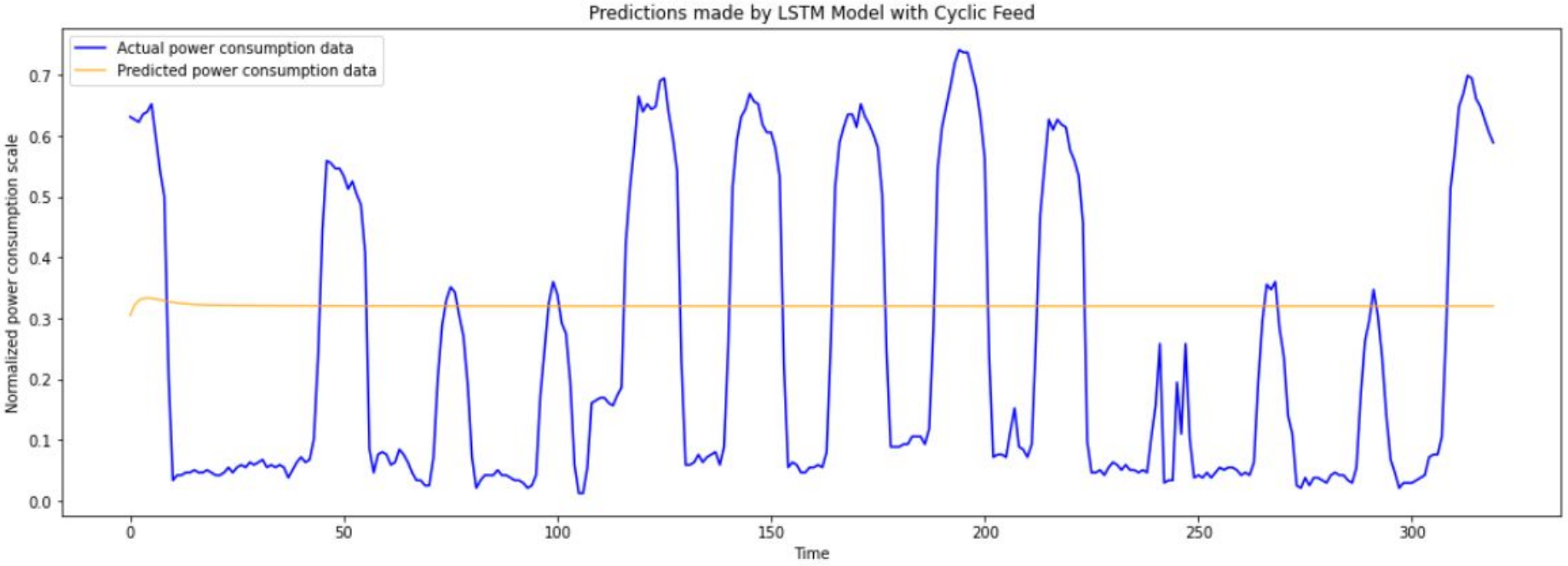
- Wurden die Testdaten als Block vom Ende des Datensatzes abgeschnitten, dann lassen sich in den Vorhersagen auch zusammenhängende Wochen abbilden.
- Hier ist gut zu erkennen, dass der RFR sauber zwischen Wertagen und Wochenende trennt.
- Der stufige Charakter der Prognosen des Algorithmus hat keine Schwierigkeiten mit abrupten Wechseln oder Anstiegen. Am Wochenende bleibt die Prognose entschieden flach.

5 Auswertung: Many-to-One RNN und LSTM



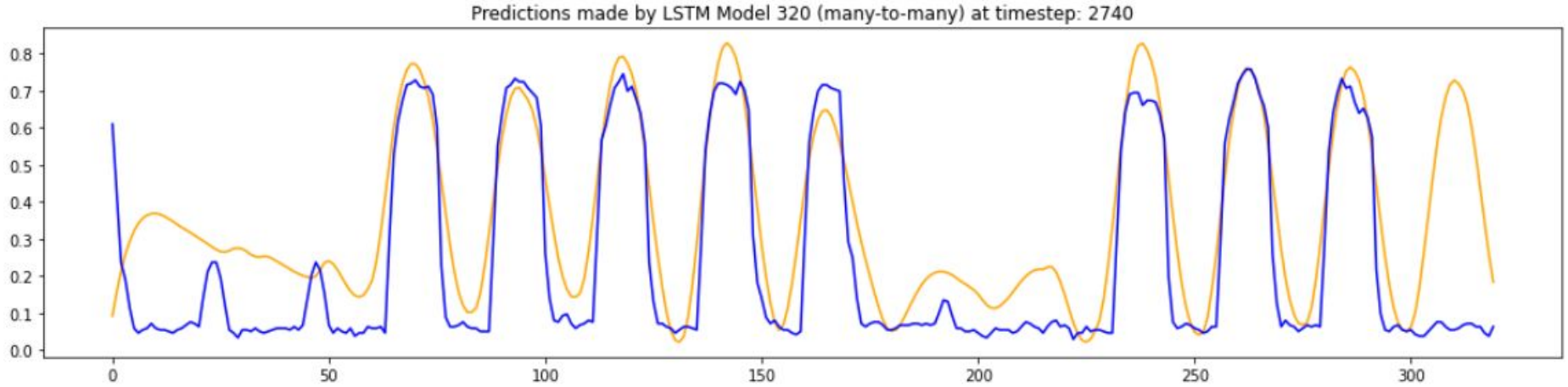
- Die Präzision der Many-to-One-Modelle scheint zunächst sehr beeindruckend.
- Der Prognose-Graph folgt sehr direkt dem realen Test-Label y_{Test}
- Nur ein leichtes Nachziehen (Rechts-Shift) ist durgängidg zu beobachten.
- Dieses Resultat ist aber sehr verständlich, wenn wir bedenken, dass wir jeweils nur einen Zeitstempel in die Zukunft blicken. Die Sequenz entsteht durch Batch-Prognosen.

5 Auswertung: Many-to-Many RNN via Cycle Feed



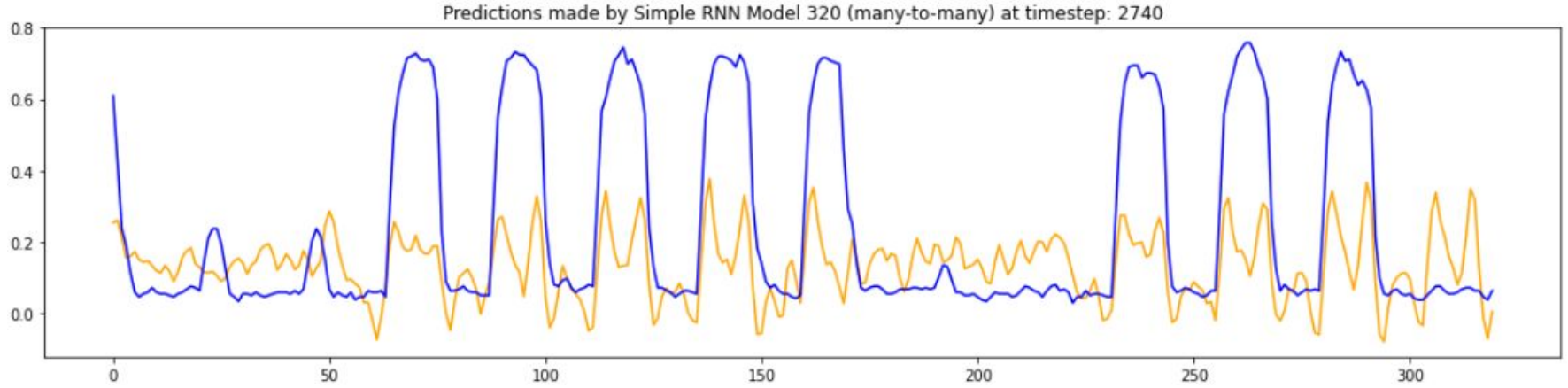
- Der Versuch ein Many-To-Many RNN zu erzeugen, indem die Prognose-Punkte an die Eingabe für die nächsten Zeitschritt gehängt wird, schlägt nach kurzer Zeit fehl.

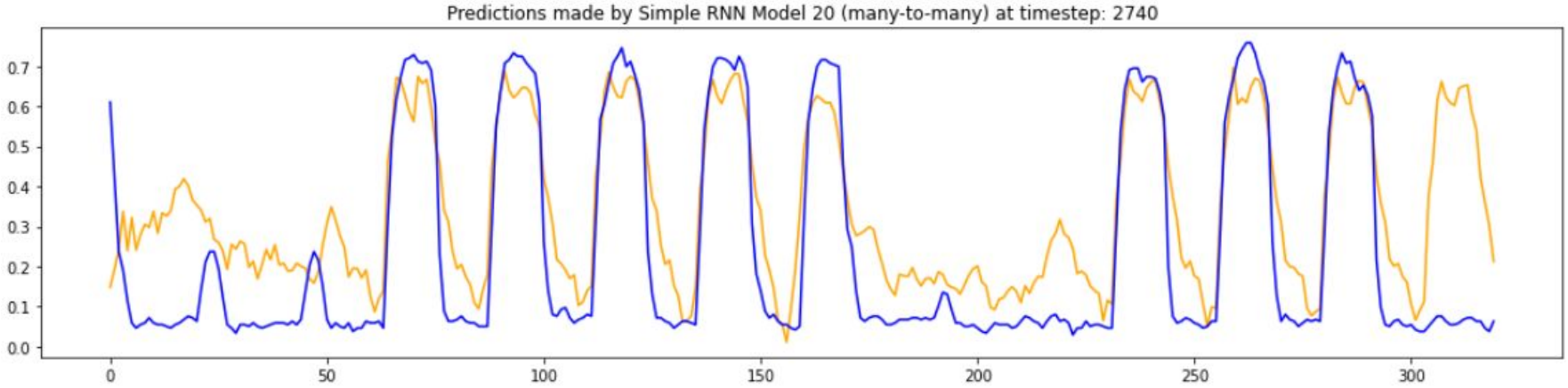
5 Auswertung: Many-to-Many LSTM



- Die Vorhersagen des LSTM-Netzes sind relativ gut.
- Die entscheidenden Zyklen werden zumindest erfasst.
- Dies ist bemerkenswert, da das Netz nicht über Wetterdaten oder zusätzliche Zeit-Features verfügte. Einzige Datenquelle ist dern historische Stromberbrauch.
- Die Kurve leidet aber unter zu starkem Smoothing.

5 Auswertung: Many-to-Many RNN





6 Fazit & Ausblick

6 Fazit: Rekurrente Netzwerke eignen sich für Time Series Forecasts

Random Forest Regressor:

- Bestes Modell in der Vergleichsreihe. Verfügte aber über Wetterdaten.
- Das spezielle Feature-Engineering erlaubte vor der Problemstellung eine ideale Datenteilung. Dieser Effekt lässt sich bei weniger regelmäßigen Vorgängen nicht reproduzieren.

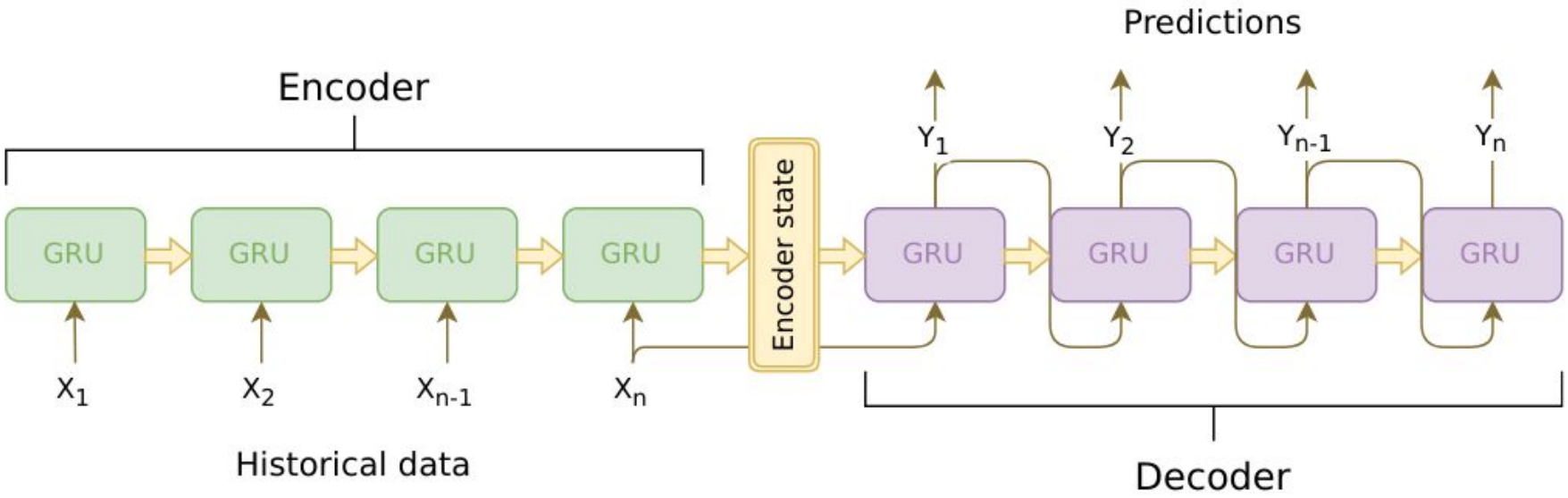
LSTM:

- Das LSTM ist in seine Aussagequalität schon ausreichend.
- Die Kurve ist aber zu geglättet (underfitting) und verpasst starke Wechsel.
- Die Prognosen bleiben bei unterschiedlichen Neuronenzahlen relativ stabil.

RNN:

- Das einfache RNN muss nicht immer die schlechtere Wahl sein.
- Bei exakten Einstellungen ist es ähnlich genau wie das LSTM und hat weniger Underfit.
- Die Qualität schwankt aber mit der Neuronenzahl im Hidden-Layer enorm.

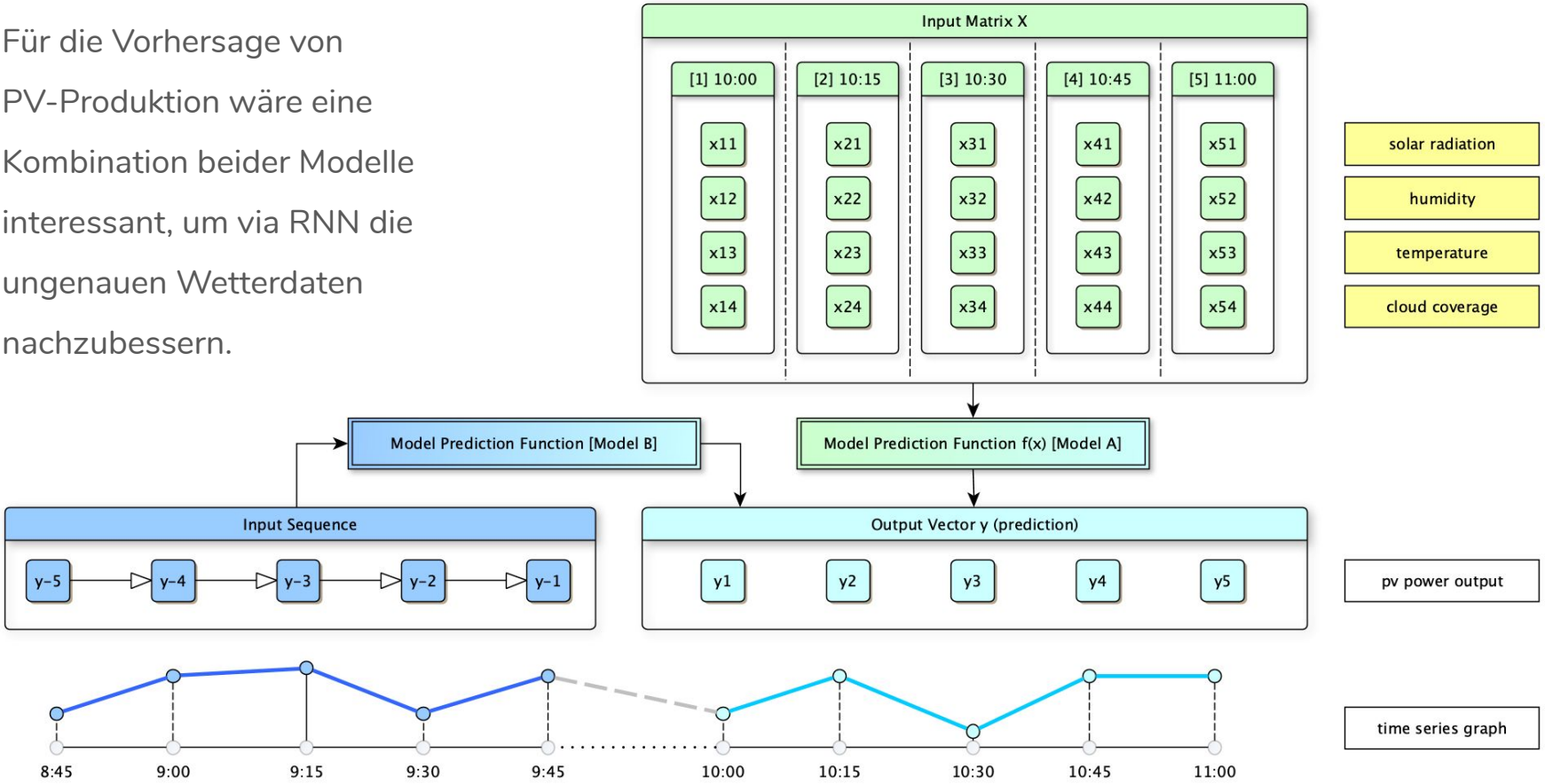
6 Ausblick: Encoder-Decoder Systematik



- Das Encoder-Decoder-System scheint für die Aufgabenstellung geeigneter.
- Hier wird sichergestellt, dass die komplette Eingabesequenz gelesen wird, bevor die Prognosewerte geschrieben werden.

6 Ausblick: Kombinationsmodelle beider Ansätze für PV

Für die Vorhersage von PV-Produktion wäre eine Kombination beider Modelle interessant, um via RNN die ungenauen Wetterdaten nachzubessern.



Fin