Artificial Intelligence and Machine Learning for Connected Industries

Class 13 - Lab Time series

Killian Cressant

Time series with RNN

RNN to predict the future

- RNN learns past patterns in the data
  - It is able to use its knowledge to forecast the future
    - Assuming of course that past patterns still hold in the future
- Analyze time series data
  - The number of daily active users on your website
  - The hourly temperature in your city
  - Home's daily power consumption
  - The trajectories of nearby cars

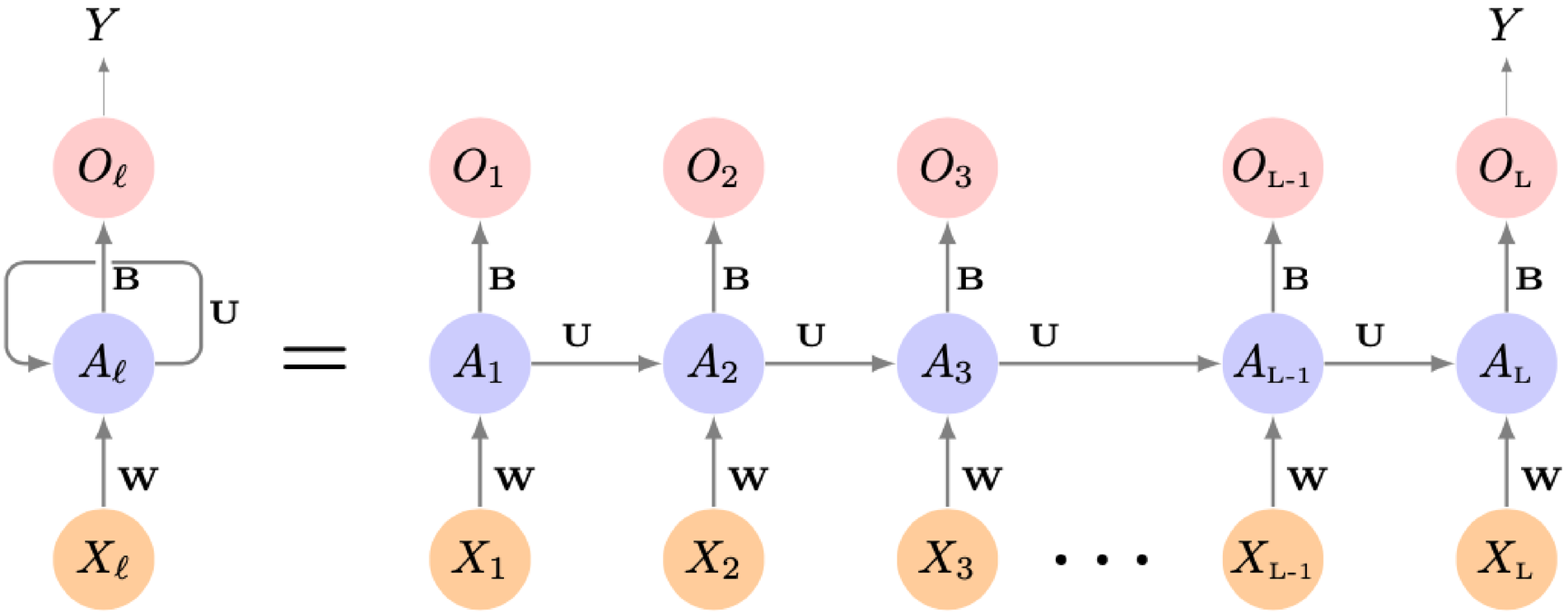
RNN to predict the future

- RNNs can work on sequences of arbitrary lengths
  - Rather than on fixed-sized inputs
- They can take
  - Sentences
  - Documents
  - Audio samples as input
    - Making them extremely useful for
      - Natural language processing applications
        - Automatic translation or speech-to-text

RNN example

- RNNs are designed to accommodate and take advantage
  - The sequential nature of such input objects
  - Similar to convolutional neural networks
    - Take advantage of the spatial structure of image inputs
- The output Y can be
  - A sequence
    - Language translation
  - A scalar
    - The binary sentiment label of a movie review document

A simple RNN example

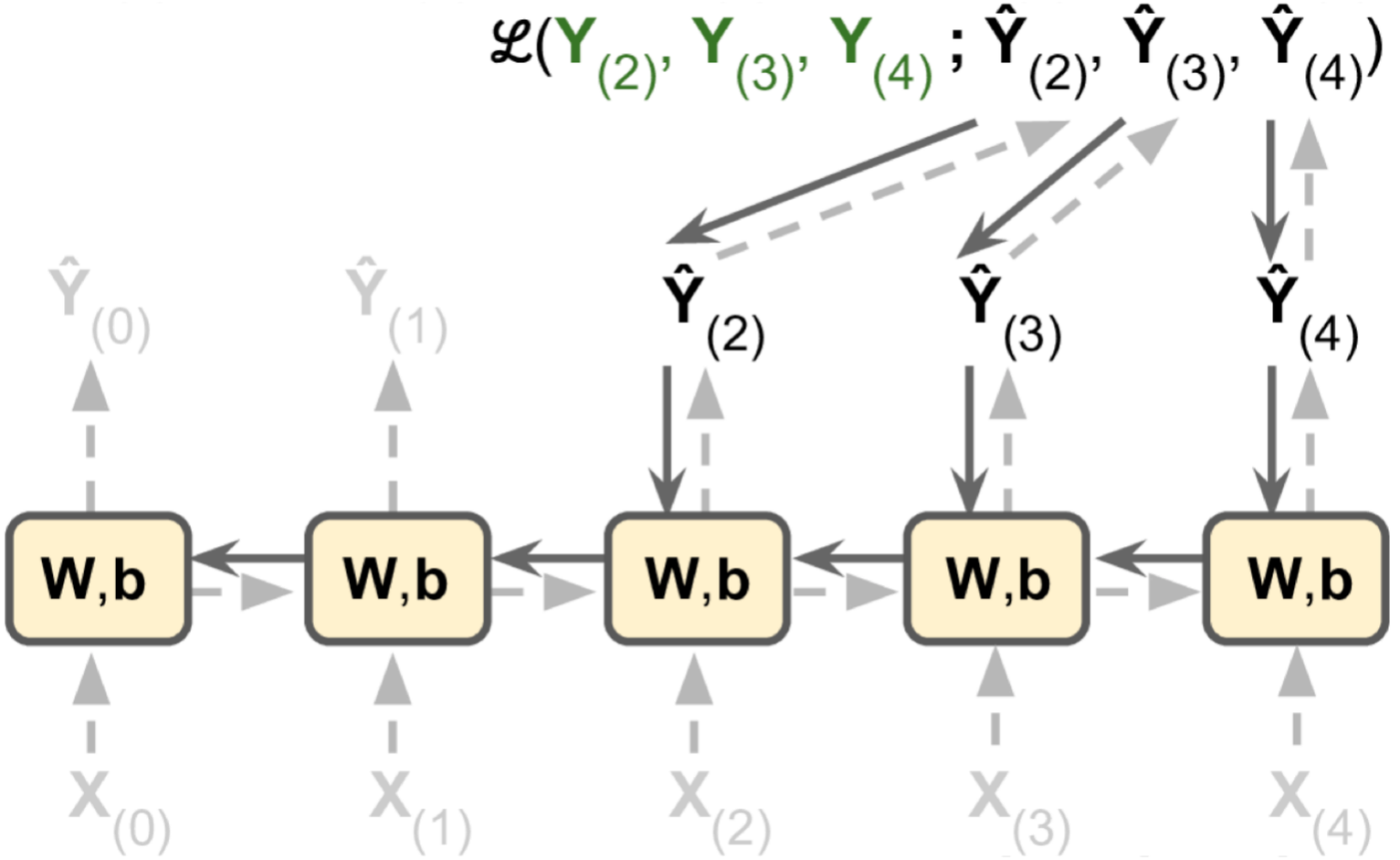


source: James, G., Witten, D., Hastie, T., Tibshirani, R. "An Introduction to Statistical Learning"

Memory cell

- The output of a recurrent neuron at time step t
  - A function of all the inputs from previous time steps
    - It has a form of memory
- Memory cell
  - A part of a neural network that preserves some state across time steps
- A very basic cell capable of learning only short patterns
  - A single recurrent neuron
  - A layer of recurrent neurons
    - It varies depending on the task
    - Typically about 10 steps long

Training



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Forecasting a Time Series

Forecasting

- Predicting future values
  - The most typical task when dealing with time series

- Other tasks include
  - Imputation
    - Filling in missing past values
  - Classification
  - Anomaly detection
  - Among others

Bus and rail — number of passengers

- Build a model capable of forecasting the number of passengers
  - Will ride on bus and rail the next day
- We have access to daily ridership data since 2001
- Download the dataset

	service_date	day_type	bus	rail_boardings	total_rides
0	2001-01-01	U	297192	126455	423647
1	2001-01-02	W	780827	501952	1282779
2	2001-01-03	W	824923	536432	1361355
3	2001-01-04	W	870021	550011	1420032
4	2001-01-05	W	890426	557917	1448343

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Load the dataset to a Pandas DataFrame

```
import pandas as pd
from pathlib import Path

path = Path("datasets/ridership/CTA_-_Ridership_-_Daily_Boarding_Totals.csv")
df = pd.read_csv(path, parse_dates=["service_date"])
df.columns = ["date", "day_type", "bus", "rail", "total"]  # shorter names
df = df.sort_values("date").set_index("date")
df = df.drop("total", axis=1)  # no need for total, it's just bus + rail
df = df.drop_duplicates()  # remove duplicated months (2011-10 and 2014-07)
```

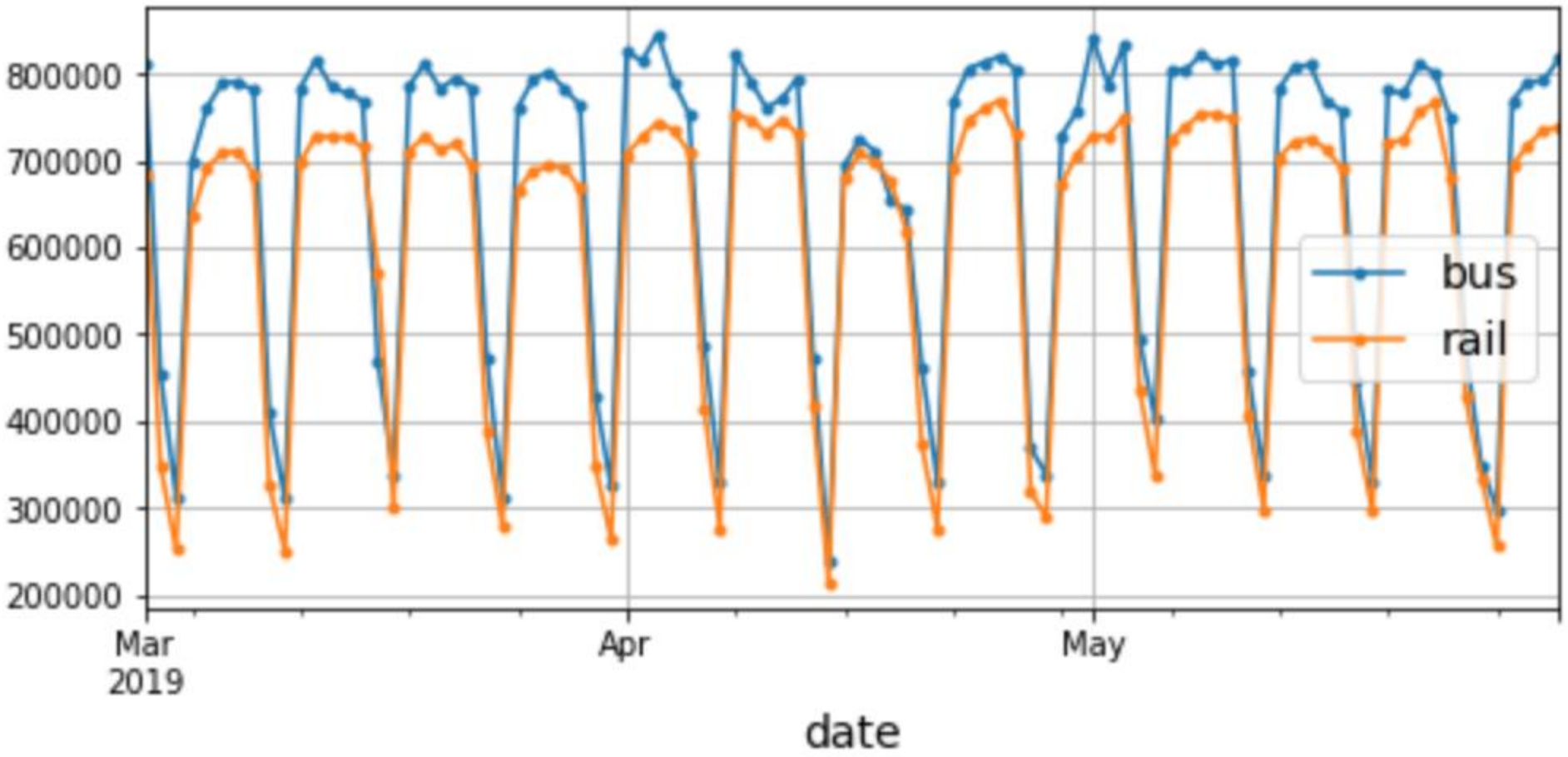
source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Let’s take a look at our dataset

date	day_type	bus	rail
2001-01-01	U	297192	126455
2001-01-02	W	780827	501952
2001-01-03	W	824923	536432
2001-01-04	W	870021	550011
2001-01-05	W	890426	557917

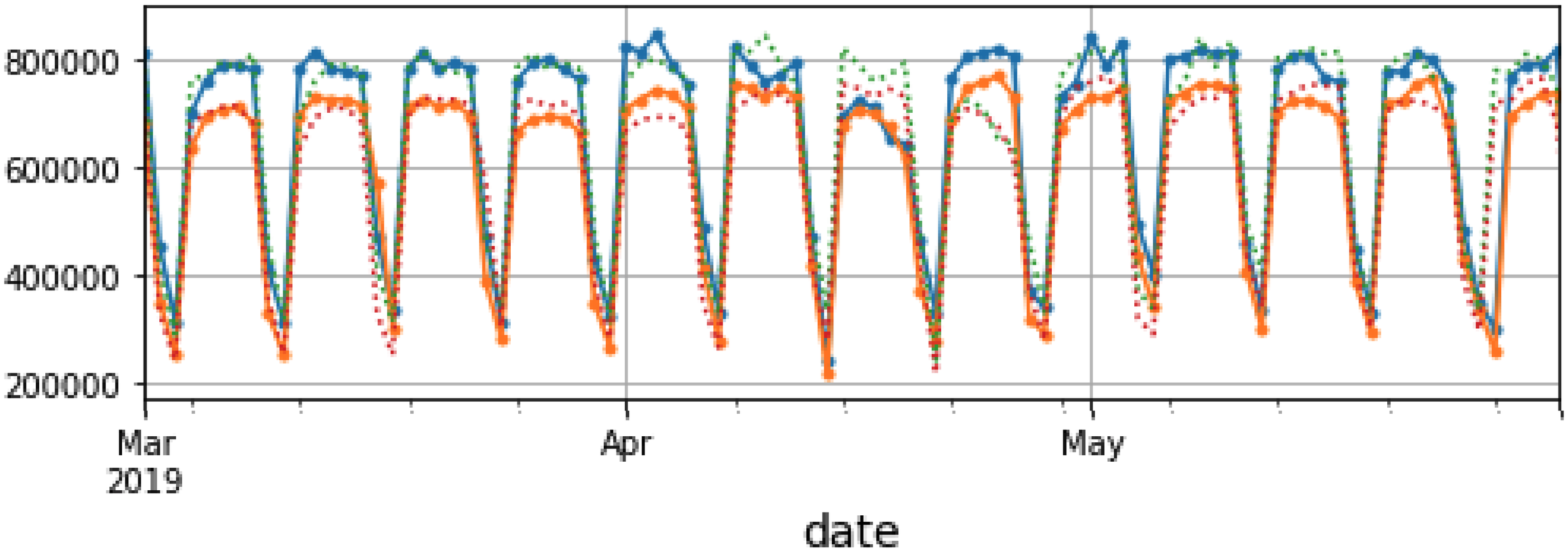
source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

The first months of 2019



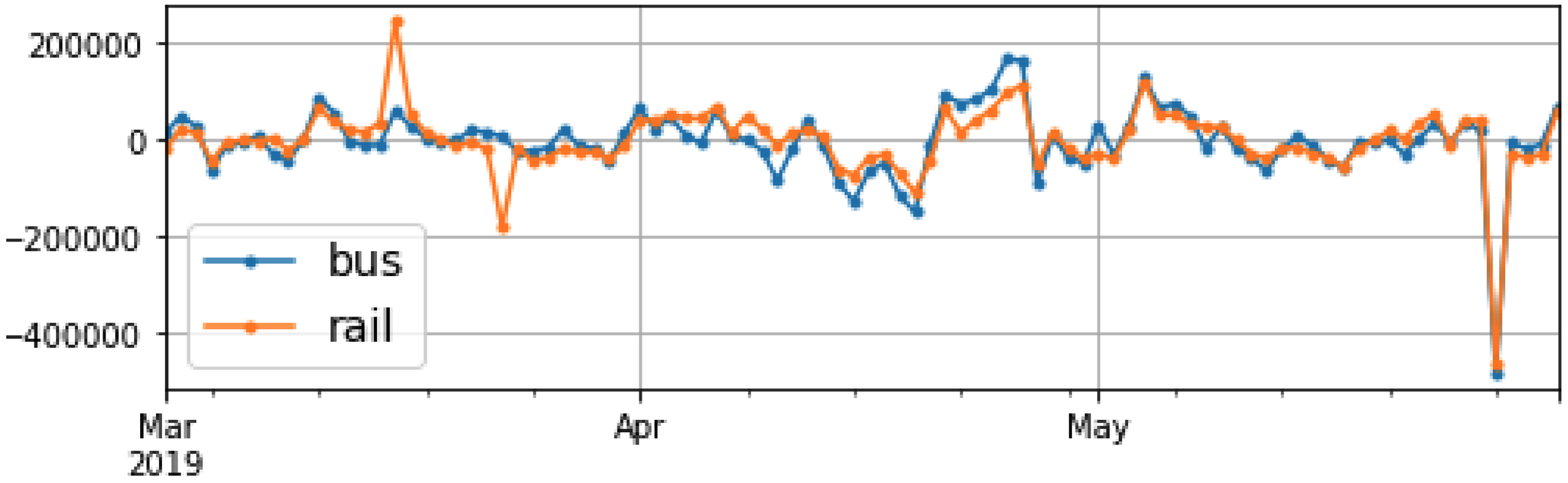
source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Naive forecasting



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Naive forecasting




```
['A', 'U', 'U']
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Measuring the MAE

```
diff_7.abs().mean()
```

```
bus     43915.608696
rail    42143.271739
dtype: float64
```

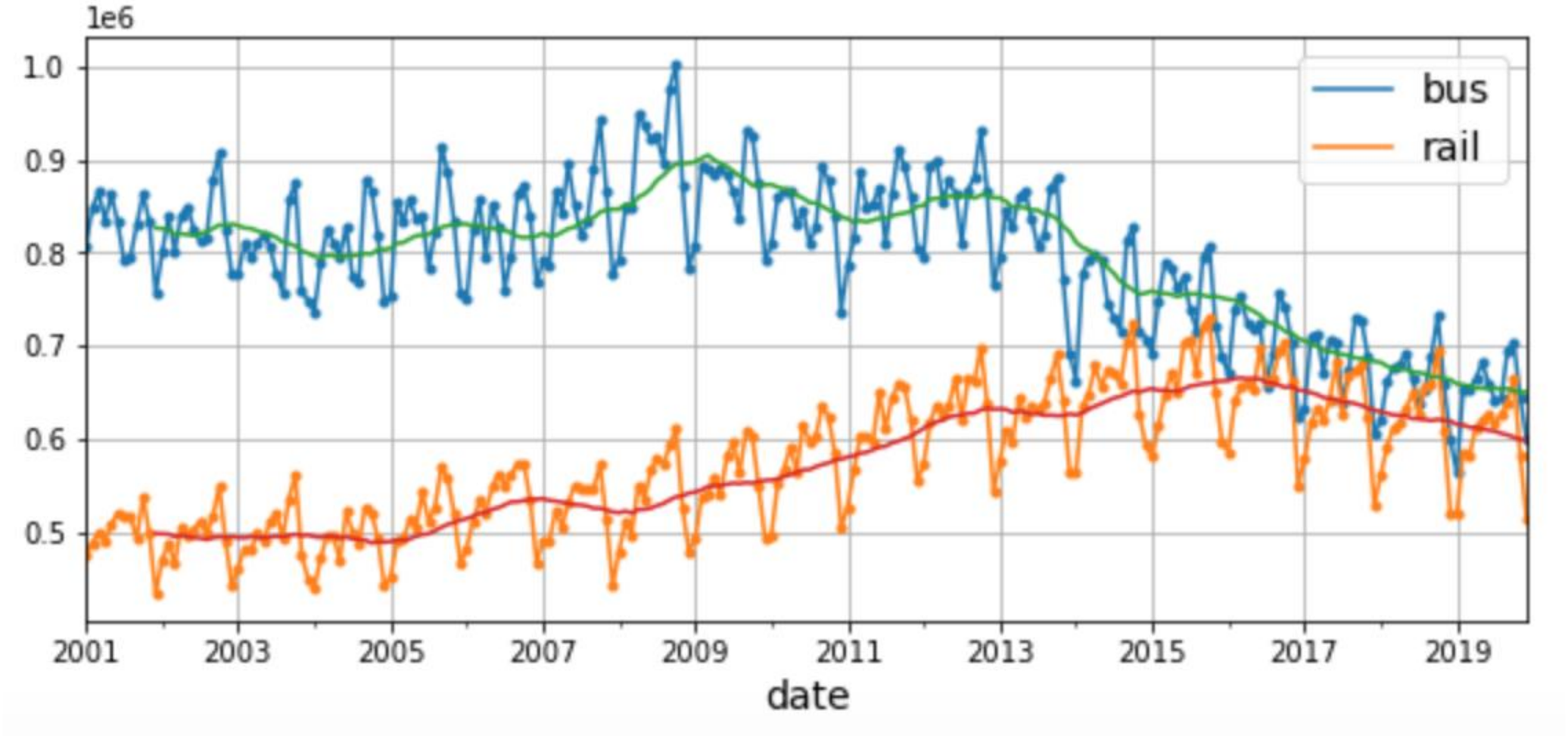
mean absolute percentage error (MAPE)

```
targets = df[["bus", "rail"]]["2019-03":"2019-05"]
(diff_7 / targets).abs().mean()
```

```
bus     0.082938
rail    0.089948
dtype: float64
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

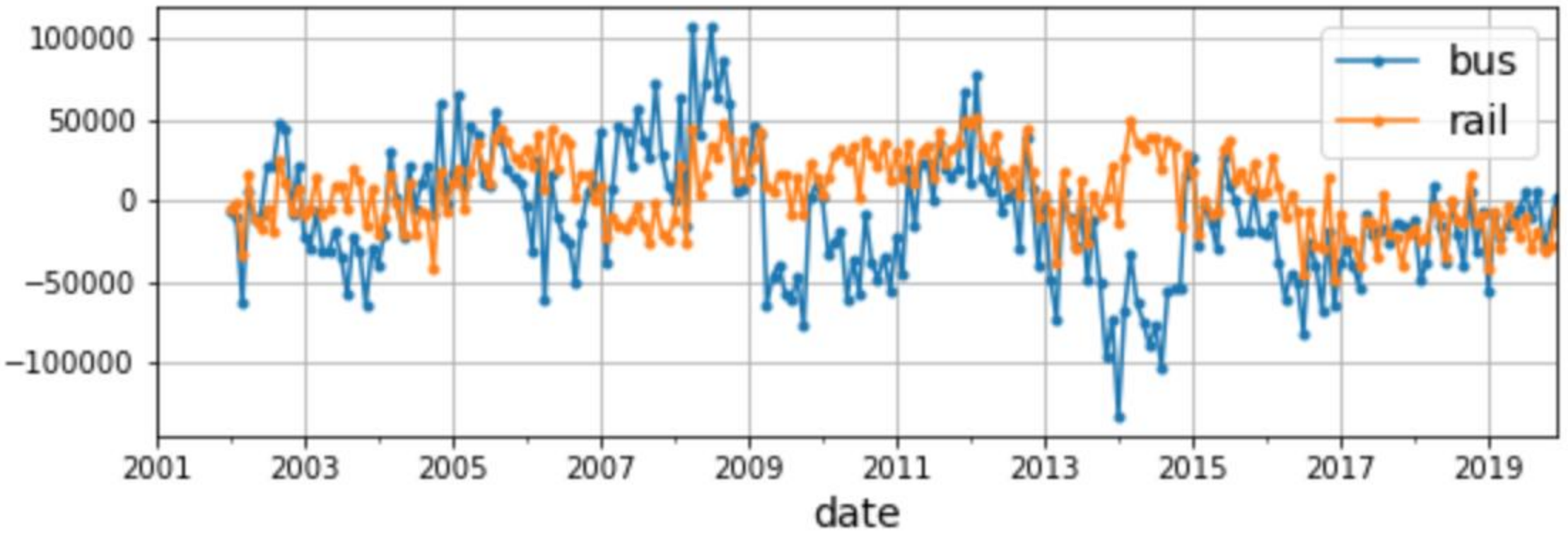
Looking for year seasonality



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

12-month difference

```
df_monthly.diff(12)[period].plot(grid=True, marker=".", figsize=(8, 3))
plt.show()
```



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Differencing

- A common technique used to remove trend and seasonality
  - From a time series
- It is easier to study a stationary time series
  - Statistical properties remain constant over time
    - Without any seasonality or trends

The ARMA Model Family

- AutoRegressive Moving Average (ARMA) model
  - by Herman Wold in the 1930s
- It computes its forecasts
  - Using a simple weighted sum of lagged values
  - Corrects these forecasts by adding a moving average

$$\hat{y}_{(t)} = \sum_{i=1}^{p} \alpha_i \, y_{(t-i)} + \sum_{i=1}^{q} \theta_i \, \epsilon_{(t-i)}$$
$$\text{with } \epsilon_{(t)} = y_{(t)} - \hat{y}_{(t)}$$

The ARMA Model Family

- AutoRegressive Moving Average (ARMA) model
  - by Herman Wold in the 1930s
- It computes its forecasts
  - Using a simple weighted sum of lagged values
  - Corrects these forecasts by adding a moving average

$$\hat{y}_{(t)} = \sum_{i=1}^{p} \alpha_i \, y_{(t-i)} + \sum_{i=1}^{q} \theta_i \, \epsilon_{(t-i)}$$
$$\text{with } \epsilon_{(t)} = y_{(t)} - \hat{y}_{(t)}$$

Differencing

- Using differencing over a single time step
  - Produces an approximation of the derivative of the time series
    - It will give the slope of the series at each time step
  - This means that it will eliminate any linear trend
    - Transforming it into a constant value
  - Examples
    - [3, 5, 7, 9, 11] —> [2, 2, 2, 2]
    - [1, 4, 9, 16, 25, 36] —> [3, 5, 7, 9, 11] —> [2, 2, 2, 2]

Differencing

- Running *d* consecutive rounds of differencing
  - Computes an approximation of the *d*-th order derivative
  - It eliminates polynomial trends up to degree *d*
  - The hyperparameter *d*
    - The order of integration

ARIMA model

- AutoRegressive Integrated Moving Average
  - Introduced in 1970 by George Box and Gwilym Jenkins
- Differencing is the central contribution
  - This model runs *d* rounds of differencing
    - Make the time series more stationary
    - Then it applies a regular ARMA model
- When doing forecast
    - It adds back the terms that were subtracted by differencing
    - After applying the ARMA model

Seasonal ARIMA — SARIMA

- It additionally models a seasonal component for a given frequency
  - Using the exact same ARIMA approach
- It has a total of *7* hyperparameters
  - The same p, d, and q, hyperparameters as ARIMA
  - Additional P, D, and Q hyperparameters to model the seasonal pattern
  - The period s of the seasonal pattern
- The hyperparameters P, D, and Q are just like p, d, and q
  - They are used to model the time series at $t - s$, $t - 2s$, $t - 3s$, etc.

Using SARIMA

```
from statsmodels.tsa.arima.model import ARIMA

origin, today = "2019-01-01", "2019-05-31"
rail_series = df.loc[origin:today]["rail"].asfreq("D")
model = ARIMA(rail_series,
              order=(1, 0, 0),
              seasonal_order=(0, 1, 1, 7))
### your code
y_pred = model.forecast()
```

(p, d, q)

(P, D, Q, s)

```
y_pred[0]  # ARIMA forecast
```

```
427758.62631318445
```

```
df["rail"].loc["2019-06-01"]  # target value
```

```
379044
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Forecasting for 3 months

```
origin, start_date, end_date = "2019-01-01", "2019-03-01", "2019-05-31"
time_period = pd.date_range(start_date, end_date)
rail_series = df.loc[origin:end_date]["rail"].asfreq("D")
y_preds = []
for today in time_period.shift(-1):
    model = ARIMA(rail_series[origin:today],  # train on data up to "today"
                  order=(1, 0, 0),
                  seasonal_order=(0, 1, 1, 7))
    model = model.fit()  # note that we retrain the model every day!
    y_pred = model.forecast()[0]
    y_preds.append(y_pred)

y_preds = pd.Series(y_preds, index=time_period)
mae = (y_preds - rail_series[time_period]).abs().mean()
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Choosing the hyper parameters

- There are several methods
- The simplest to understand and to get started is the brute-force approach
  - Run a grid search

Using RNN to forecast

Our goal

- Predict tomorrow's ridership
  - Based on the ridership of the past 8 weeks (56 days)
- The inputs to our model will be sequences
  - A single sequence per day once the model is in production
  - Each containing 56 values from time steps $t - 55$ to t
- For each input sequence
  - The model will output a single value
    - The forecast for time step $t + 1$

The training set

- We will use every 56-day window from the past as training data
- The target for each window will be the value immediately following it
- Keras has a utility function to prepare our dataset
  - tf.keras.utils.timeseries_dataset_from_array()

```
my_series = [0, 1, 2, 3, 4, 5]
my_dataset = tf.keras.utils.timeseries_dataset_from_array(
    my_series,
    targets=my_series[3:],  # the targets are 3 steps into the future
    sequence_length=3,
    batch_size=2
)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Our dataset

```
list(my_dataset)
```

```
[(<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
  array([[0, 1, 2],
         [1, 2, 3]], dtype=int32)>,
  <tf.Tensor: shape=(2,), dtype=int32, numpy=array([3, 4], dtype=int32)>),
 (<tf.Tensor: shape=(1, 3), dtype=int32, numpy=array([[2, 3, 4]], dtype=int32)>,
  <tf.Tensor: shape=(1,), dtype=int32, numpy=array([5], dtype=int32)>)]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

window() method

```
for window_dataset in tf.data.Dataset.range(6).window(4, shift=1):
    for element in window_dataset:
        print(f"{element}", end=" ")
    print()
```

```
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5
4 5
5
```

drop_remainder=True

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

The window() method

- Returns a nested dataset
  - Analogous to a list of lists
- This is useful when you want to transform each window
  - By calling its dataset methods
  - To shuffle them or batch them
- We cannot use a nested dataset directly for training
  - Our model will expect tensors as input
    - Not datasets

flat_map() method

- It converts a nested dataset into a flat dataset
  - Contains tensors
- Can take a function as an argument
  - Allows us to transform each dataset
    - In the nested dataset before flattening

window() + flat_map()

```
dataset = tf.data.Dataset.range(6).window(4, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window_dataset: window_dataset.batch(4))
for window_tensor in dataset:
    print(f"{window_tensor}")
```

```
[0 1 2 3]
[1 2 3 4]
[2 3 4 5]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Creating our sets

- Creating our own function

```
def to_windows(dataset, length):
    dataset = dataset.window(length, shift=1, drop_remainder=True)
    return dataset.flat_map(lambda window_ds: window_ds.batch(length))
```

- Splitting the dataset into train, validation, and test sets

```
rail_train = df["rail"]["2016-01":"2018-12"] / 1e6
rail_valid = df["rail"]["2019-01":"2019-05"] / 1e6
rail_test = df["rail"]["2019-06":] / 1e6
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Creating the training set

```
seq_length = 56
train_ds = tf.keras.utils.timeseries_dataset_from_array(
    rail_train.to_numpy(),
    targets=rail_train[seq_length:],
    sequence_length=seq_length,
    batch_size=32,
    shuffle=True,
    seed=42
)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Creating the validation set

```
valid_ds = tf.keras.utils.timeseries_dataset_from_array(
    rail_valid.to_numpy(),
    targets=rail_valid[seq_length:],
    sequence_length=seq_length,
    batch_size=32
)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Using RNN

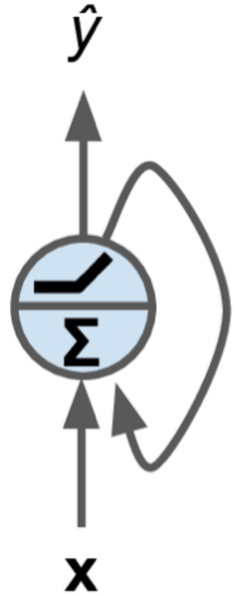
A simple RNN

Univariate sequences of any length

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(1, input_shape=[None, 1])
])
```

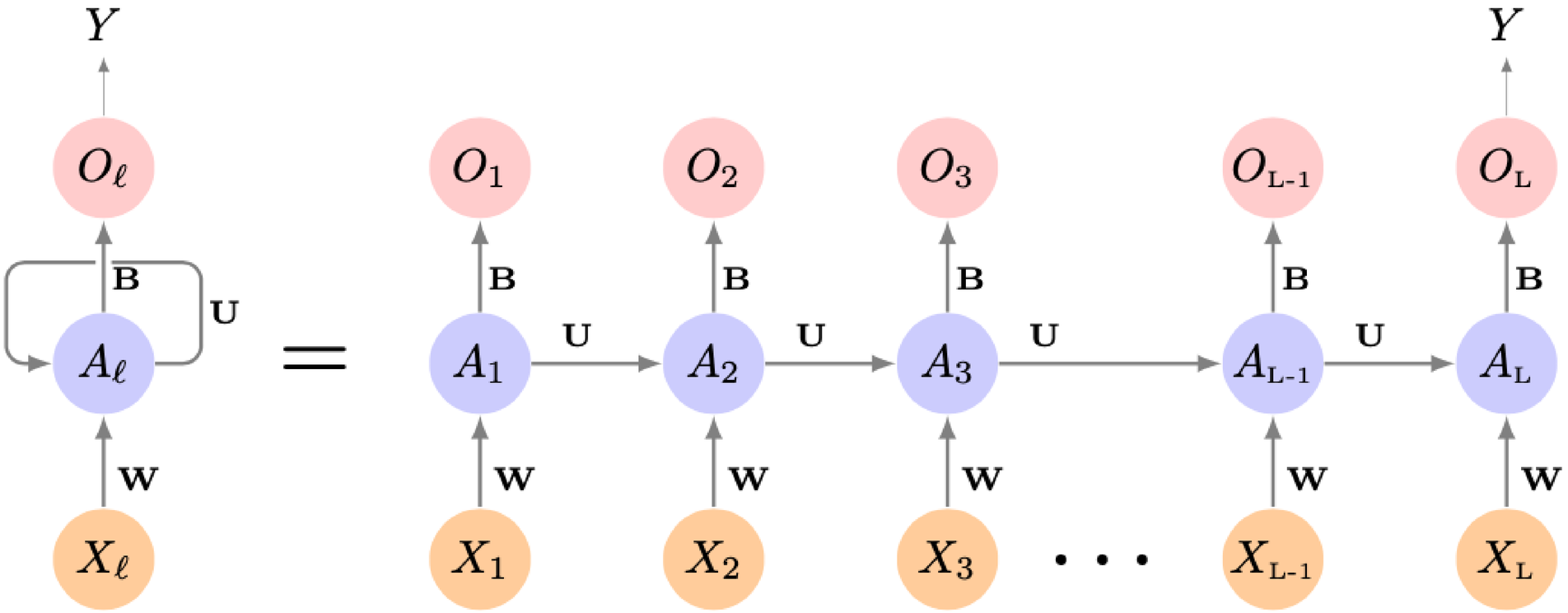
source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

- All recurrent layers in Keras expect 3-dimensional inputs of shape
  - [batch size, time steps, dimensionality]
- Dimensionality
  - Univariate time series —> 1
- Time steps —> None —> any size



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

A simple RNN example



source: James, G., Witten, D., Hastie, T., Tibshirani, R. "An Introduction to Statistical Learning"

Fit and evaluate function

```
def fit_and_evaluate(model, train_set, valid_set, learning_rate, epochs=500):
    early_stopping_cb = tf.keras.callbacks.EarlyStopping(
        monitor="val_mae", patience=50, restore_best_weights=True)
    opt = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
    model.compile(loss=tf.keras.losses.Huber(), optimizer=opt, metrics=["mae"])
    history = model.fit(train_set, validation_data=valid_set, epochs=epochs,
                        callbacks=[early_stopping_cb])
    valid_loss, valid_mae = model.evaluate(valid_set)

    return valid_mae * 1e6
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Bad result

- This result was to be expected for two reasons
  - A single recurrent neuron
    - Only two values at each step
      - The current value of the sequence
      - The previous output
    - It has only 3 parameters
  - The activation function tahn (Hyperbolic tangent)
    - Output values from -1 and 1
    - Time series contains values from 0 to 1.4

Let’s create our Deep RNN model

```
tf.random.set_seed(42)  # extra code – ensures reproducibility

# Add 3 Recurrent layers of 32 neurons each
# The first two --> return_sequences=True
# One dense layer with a single neuron

deep_model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(32, return_sequences=True, input_shape=[None, 1]),
    tf.keras.layers.SimpleRNN(32, return_sequences=True),
    tf.keras.layers.SimpleRNN(32),
    tf.keras.layers.Dense(1)
])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Forecasting Multivariate Time Series

Forecast the rail time series

- Using the bus and rail data
- Using the day type (Weekday, sAturday, sUnday)
  - Shift the day type series
    - One day into the future
      - Use this as tomorrow's day type as input
- We can use the same architecture

Creating our multivar dataset

```
# First creat the dataset with the bus and rail information
# Scale (1e6)

df_mulvar = df[["bus", "rail"]] / 1e6

# Insert in the dataset the "next_day_type" column

df_mulvar["next_day_type"] = df["day_type"].shift(-1)

df_mulvar = pd.get_dummies(df_mulvar)  # one-hot encode the day type
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Splitting our dataset

```
# Let's split our dataset - train, validation, and test sets

mulvar_train = df_mulvar["2016-01":"2018-12"]

mulvar_valid = df_mulvar["2019-01":"2019-05"]

mulvar_test = df_mulvar["2019-06":]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Creating our series training set

```
tf.random.set_seed(42)  # extra code – ensures reproducibility

train_mulvar_ds = tf.keras.utils.timeseries_dataset_from_array(
    mulvar_train.to_numpy(),  # use all 5 columns as input
    targets=mulvar_train["rail"][seq_length:],  # forecast only the rail series
    sequence_length=seq_length,
    batch_size=32,
    shuffle=True,
    seed=42
)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Creating our series validation set

```
valid_mulvar_ds = tf.keras.utils.timeseries_dataset_from_array(
    mulvar_valid.to_numpy(),
    targets=mulvar_valid["rail"][seq_length:],
    sequence_length=seq_length,
    batch_size=32
)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Let’s create our multi-task model

```
# Let's create our multivariate model with
# SimpleRNN with 32 neurons
# One dense layaer with a single neuron on top of it

tf.random.set_seed(42)

multask_model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(32, input_shape=[None, 5]),
    tf.keras.layers.Dense(2)
])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Forecasting Several Steps Ahead

Forecasting Several Time Steps Ahead

- Predicting several steps ahead
  - Just change the target values to the day we want

What if we want to predict the next 14 values?

First option

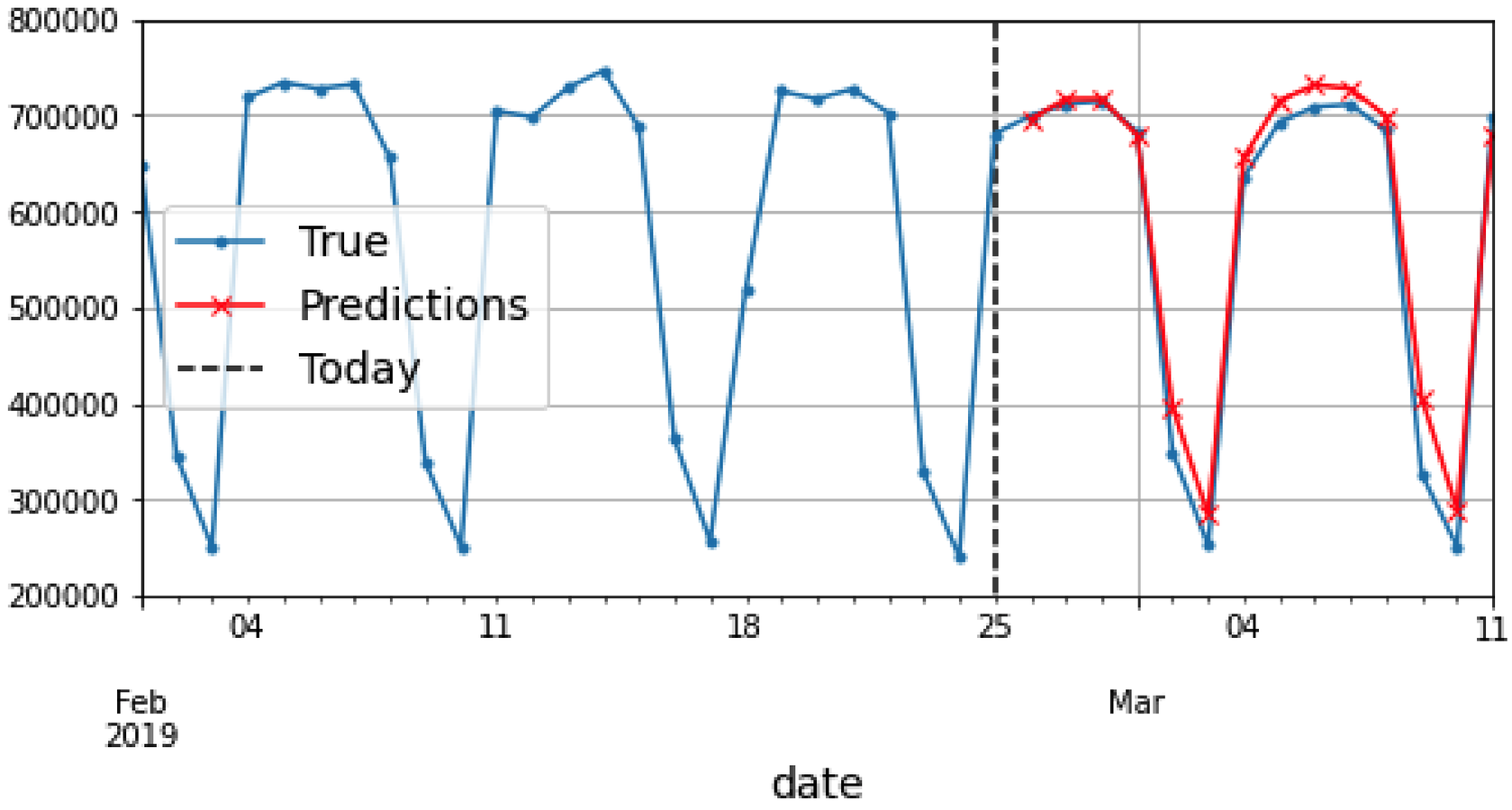
```
import numpy as np

X = rail_valid.to_numpy()[np.newaxis, :seq_length, np.newaxis]
for step_ahead in range(14):
    y_pred_one = univar_model.predict(X)
    X = np.concatenate([X, y_pred_one.reshape(1, 1, 1)], axis=1)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

- X —> 3D array of shape [1, 56, 1]
  - Recall that recurrent layers expect 3D inputs
- Repeatedly forecast the next value
  - Appending it to the input series

Plotting the results



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Second option

- Creating a RNN that predicted the 14 values at once
- We can still use a sequence-to-vector model
  - It will output 14 values instead of 1
- We first need to change the targets to be vectors
  - Containing the next 14 values
  - We can use timeseries_dataset_from_array()

Creating our datasets

```
tf.random.set_seed(42)  # extra code – ensures reproducibility

def split_inputs_and_targets(mulvar_series, ahead=14, target_col=1):
    return mulvar_series[:, :-ahead], mulvar_series[:, -ahead:, target_col]
```

```
ahead_train_ds = tf.keras.utils.timeseries_dataset_from_array(
    mulvar_train.to_numpy(),
    targets=None,
    sequence_length=seq_length + 14,
    batch_size=32,
    shuffle=True,
    seed=42
).map(split_inputs_and_targets)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Let’s create our model

```
tf.random.set_seed(42)

# Let's create our multivariate model with
# SimpleRNN with 32 neurons
# One dense layaer with 14 neurons on top of it

ahead_model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(32, input_shape=[None, 5]),
    tf.keras.layers.Dense(14)
])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Using a Sequence-To-Sequence Model

- We can train it to forecast the next 14 values
  - At each and every time step
- The advantage of this technique
  - The loss will contain a term for the output of the RNN
    - At each and every time step
    - Not just for the output at the last time step
- This will both stabilize and speed up training

Using a Sequence-To-Sequence Model

- At time step 0 the model will output
  - A vector containing the forecasts for time steps 1 to 14
- At time step 1 the model will forecast
  - Time steps 2 to 15 and so on
- The targets are sequences of consecutive windows
  - Shifted by 1 time step at each time step
- The target is not a vector anymore
  - A sequence of the same length as the inputs
  - Containing a 14-dimensional vector at each step

Creating our training set

```
my_series = tf.data.Dataset.range(7)
dataset = to_windows(to_windows(my_series, 3), 4)
list(dataset)
```

```
[<tf.Tensor: shape=(4, 3), dtype=int64, numpy=
 array([[0, 1, 2],
        [1, 2, 3],
        [2, 3, 4],
        [3, 4, 5]])>,
 <tf.Tensor: shape=(4, 3), dtype=int64, numpy=
 array([[1, 2, 3],
        [2, 3, 4],
        [3, 4, 5],
        [4, 5, 6]])>]
```

sequence

predictions -1

[0, 1, 2, 3, 4, 5, 6,]

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Creating inputs and targets

```
dataset = dataset.map(lambda S: (S[:, 0], S[:, 1:]))
list(dataset)
```

```
[(<tf.Tensor: shape=(4,), dtype=int64, numpy=array([0, 1, 2, 3])>,
  <tf.Tensor: shape=(4, 2), dtype=int64, numpy=
  array([[1, 2],
         [2, 3],
         [3, 4],
         [4, 5]])>),
 (<tf.Tensor: shape=(4,), dtype=int64, numpy=array([1, 2, 3, 4])>,
  <tf.Tensor: shape=(4, 2), dtype=int64, numpy=
  array([[2, 3],
         [3, 4],
         [4, 5],
         [5, 6]])>)]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Artificial Intelligence and Machine Learning for Connected Industries

Class 13 - Lab Time series

Killian Cressant