04_multivariate_timeseries

September 29, 2021

1 Multivariate Time Series Regression

So far, we have limited our modeling efforts to single time series. RNNs are naturally well suited to multivariate time series and represent a non-linear alternative to the Vector Autoregressive (VAR) models we covered in Chapter 8, Time Series Models.

1.1 Imports & Settings

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
%matplotlib inline

from pathlib import Path
import numpy as np
import pandas as pd
import pandas_datareader.data as web

from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import minmax_scale

import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, LSTM
import tensorflow.keras.backend as K

import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
if gpu_devices:
    print('Using GPU')
    tf.config.experimental.set_memory_growth(gpu_devices[0], True)
else:
    print('Using CPU')
```

```
Using CPU
```

[4]:
```
sns.set_style('whitegrid')
np.random.seed(42)
```

[5]:
```
results_path = Path('results', 'multivariate_time_series')
if not results_path.exists():
    results_path.mkdir(parents=True)
```

1.2 Load Data

For comparison, we illustrate the application of RNNs to modeling and forecasting several time series using the same dataset we used for the VAR example, monthly data on consumer sentiment, and industrial production from the Federal Reserve's FRED service in Chapter 8, Time Series Models:

[6]:
```
df = web.DataReader(['UMCSENT', 'IPGMFN'], 'fred', '1980', '2019-12').dropna()
df.columns = ['sentiment', 'ip']
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 480 entries, 1980-01-01 to 2019-12-01
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   sentiment  480 non-null    float64
 1   ip         480 non-null    float64
dtypes: float64(2)
memory usage: 11.2 KB
```

[7]:
```
df.head()
```

[7]:

DATE	sentiment	ip
1980-01-01	67.0	46.8770
1980-02-01	66.9	47.9757
1980-03-01	56.5	48.4793
1980-04-01	52.7	47.0662
1980-05-01	51.7	45.6995

1.3 Prepare Data

1.3.1 Stationarity

We apply the same transformation—annual difference for both series, prior log-transform for industrial production—to achieve stationarity that we used in Chapter 8 on Time Series Models:

[8]:
```
df_transformed = (pd.DataFrame({'ip': np.log(df.ip).diff(12),
                                'sentiment': df.sentiment.diff(12)})
                  .dropna())
```

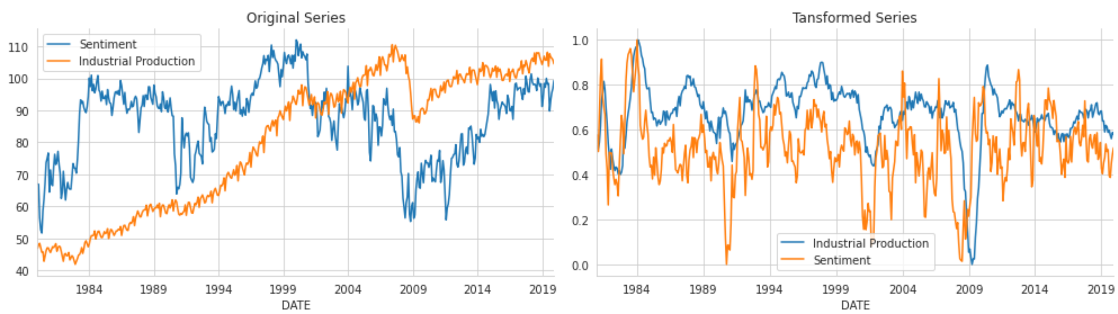
1.3.2 Scaling

Then we scale the transformed data to the [0,1] interval:

[9]:
```
df_transformed = df_transformed.apply(minmax_scale)
```

1.3.3 Plot original and transformed series

[10]:
```
fig, axes = plt.subplots(ncols=2, figsize=(14,4))
columns={'ip': 'Industrial Production', 'sentiment': 'Sentiment'}
df.rename(columns=columns).plot(ax=axes[0], title='Original Series')
df_transformed.rename(columns=columns).plot(ax=axes[1], title='Tansformed␣
↪Series')
sns.despine()
fig.tight_layout()
fig.savefig(results_path / 'multi_rnn', dpi=300)
```



1.3.4 Reshape data into RNN format

We can reshape directly to get non-overlapping series, i.e., one sample for each year (works only if the number of samples is divisible by window size):

[11]:
```
df.values.reshape(-1, 12, 2).shape
```

[11]:
```
(40, 12, 2)
```

However, we want rolling, not non-overlapping lagged values. The create_multivariate_rnn_data function transforms a dataset of several time series into the shape required by the Keras RNN layers, namely `n_samples` x `window_size` x `n_series`, as follows:

[12]:
```
def create_multivariate_rnn_data(data, window_size):
    y = data[window_size:]
    n = data.shape[0]
    X = np.stack([data[i: j]
                  for i, j in enumerate(range(window_size, n))], axis=0)
    return X, y
```

We will use window_size of 24 months and obtain the desired inputs for our RNN model, as follows:

[13]:
```
window_size = 18
```

[14]:
```
X, y = create_multivariate_rnn_data(df_transformed, window_size=window_size)
```

[15]:
```
X.shape, y.shape
```

[15]:
```
((450, 18, 2), (450, 2))
```

[16]:
```
df_transformed.head()
```

[16]:
```
                  ip  sentiment
DATE
1981-01-01  0.526669   0.576214
1981-02-01  0.513795   0.502513
1981-03-01  0.542863   0.670017
1981-04-01  0.613397   0.832496
1981-05-01  0.731775   0.914573
```

Finally, we split our data into a train and a test set, using the last 24 months to test the out-of-sample performance, as shown here:

[17]:
```
test_size =24
train_size = X.shape[0]-test_size
```

[18]:
```
X_train, y_train = X[:train_size], y[:train_size]
X_test, y_test = X[train_size:], y[train_size:]
```

[19]:
```
X_train.shape, X_test.shape
```

[19]:
```
((426, 18, 2), (24, 18, 2))
```

1.4 Define Model Architecture

We use a similar architecture with two stacked LSTM layers with 12 and 6 units, respectively, followed by a fully-connected layer with 10 units. The output layer has two units, one for each time series. We compile them using mean absolute loss and the recommended RMSProp optimizer, as follows:

[20]:
```
K.clear_session()
```

[21]:
```
n_features = output_size = 2
```

[22]:
```
lstm_units = 12
dense_units = 6
```

[23]:
```
rnn = Sequential([
    LSTM(units=lstm_units,
         dropout=.1,
         recurrent_dropout=.1,
         input_shape=(window_size, n_features), name='LSTM',
         return_sequences=False),
    Dense(dense_units, name='FC'),
    Dense(output_size, name='Output')
])
```

The model has 1,268 parameters, as shown here:

[24]:
```
rnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
LSTM (LSTM)                  (None, 12)                720
_________________________________________________________________
FC (Dense)                   (None, 6)                 78
_________________________________________________________________
Output (Dense)               (None, 2)                 14
=================================================================
Total params: 812
Trainable params: 812
Non-trainable params: 0
_________________________________________________________________
```

[25]:
```
rnn.compile(loss='mae', optimizer='RMSProp')
```

1.5 Train the Model

We train for 50 epochs with a batch_size value of 20 using early stopping:

[26]:
```
lstm_path = (results_path / 'lstm.h5').as_posix()

checkpointer = ModelCheckpoint(filepath=lstm_path,
                               verbose=1,
                               monitor='val_loss',
                               mode='min',
                               save_best_only=True)
```

[27]:
```
early_stopping = EarlyStopping(monitor='val_loss',
                              patience=10,
                              restore_best_weights=True)
```

[28]:
```
result = rnn.fit(X_train,
                 y_train,
                 epochs=100,
                 batch_size=20,
                 shuffle=False,
                 validation_data=(X_test, y_test),
                 callbacks=[early_stopping, checkpointer],
                 verbose=1)
```

```
Epoch 1/100
19/22 [========================>…] - ETA: 0s - loss: 0.2743
Epoch 00001: val_loss improved from inf to 0.04285, saving model to
results/multivariate_time_series/lstm.h5
22/22 [==============================] - 1s 25ms/step - loss: 0.2536 - val_loss:
0.0429
Epoch 2/100
20/22 [==========================>…] - ETA: 0s - loss: 0.1013
Epoch 00002: val_loss improved from 0.04285 to 0.03912, saving model to
results/multivariate_time_series/lstm.h5
22/22 [==============================] - 0s 13ms/step - loss: 0.0991 - val_loss:
0.0391
Epoch 3/100
20/22 [==========================>…] - ETA: 0s - loss: 0.0956
Epoch 00003: val_loss did not improve from 0.03912
22/22 [==============================] - 0s 12ms/step - loss: 0.0941 - val_loss:
0.0404
Epoch 4/100
19/22 [========================>…] - ETA: 0s - loss: 0.0965
Epoch 00004: val_loss improved from 0.03912 to 0.03764, saving model to
results/multivariate_time_series/lstm.h5
22/22 [==============================] - 0s 14ms/step - loss: 0.0945 - val_loss:
0.0376
Epoch 5/100
18/22 [=======================>…] - ETA: 0s - loss: 0.0910
Epoch 00005: val_loss did not improve from 0.03764
22/22 [==============================] - 0s 12ms/step - loss: 0.0918 - val_loss:
0.0504
Epoch 6/100
21/22 [===========================>..] - ETA: 0s - loss: 0.0903
Epoch 00006: val_loss improved from 0.03764 to 0.03714, saving model to
results/multivariate_time_series/lstm.h5
22/22 [==============================] - 0s 13ms/step - loss: 0.0898 - val_loss:
0.0371
Epoch 7/100
20/22 [==========================>…] - ETA: 0s - loss: 0.0898
Epoch 00007: val_loss did not improve from 0.03714
22/22 [==============================] - 0s 12ms/step - loss: 0.0885 - val_loss:
```

```
0.0376
Epoch 8/100
19/22 [========================>…] - ETA: 0s - loss: 0.0908
Epoch 00008: val_loss did not improve from 0.03714
22/22 [==============================] - 0s 13ms/step - loss: 0.0884 - val_loss:
0.0491
Epoch 9/100
19/22 [========================>…] - ETA: 0s - loss: 0.0899
Epoch 00009: val_loss did not improve from 0.03714
22/22 [==============================] - 0s 12ms/step - loss: 0.0876 - val_loss:
0.0418
Epoch 10/100
19/22 [========================>…] - ETA: 0s - loss: 0.0906
Epoch 00010: val_loss improved from 0.03714 to 0.03557, saving model to
results/multivariate_time_series/lstm.h5
22/22 [==============================] - 0s 13ms/step - loss: 0.0892 - val_loss:
0.0356
Epoch 11/100
19/22 [========================>…] - ETA: 0s - loss: 0.0916
Epoch 00011: val_loss did not improve from 0.03557
22/22 [==============================] - 0s 13ms/step - loss: 0.0894 - val_loss:
0.0463
Epoch 12/100
18/22 [=======================>…] - ETA: 0s - loss: 0.0883
Epoch 00012: val_loss did not improve from 0.03557
22/22 [==============================] - 0s 13ms/step - loss: 0.0877 - val_loss:
0.0389
Epoch 13/100
18/22 [=======================>…] - ETA: 0s - loss: 0.0882
Epoch 00013: val_loss did not improve from 0.03557
22/22 [==============================] - 0s 13ms/step - loss: 0.0873 - val_loss:
0.0451
Epoch 14/100
18/22 [=======================>…] - ETA: 0s - loss: 0.0879
Epoch 00014: val_loss improved from 0.03557 to 0.03552, saving model to
results/multivariate_time_series/lstm.h5
22/22 [==============================] - 0s 14ms/step - loss: 0.0867 - val_loss:
0.0355
Epoch 15/100
20/22 [===========================>…] - ETA: 0s - loss: 0.0854
Epoch 00015: val_loss improved from 0.03552 to 0.03534, saving model to
results/multivariate_time_series/lstm.h5
22/22 [==============================] - 0s 12ms/step - loss: 0.0837 - val_loss:
0.0353
Epoch 16/100
19/22 [========================>…] - ETA: 0s - loss: 0.0864
Epoch 00016: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 13ms/step - loss: 0.0841 - val_loss:
```

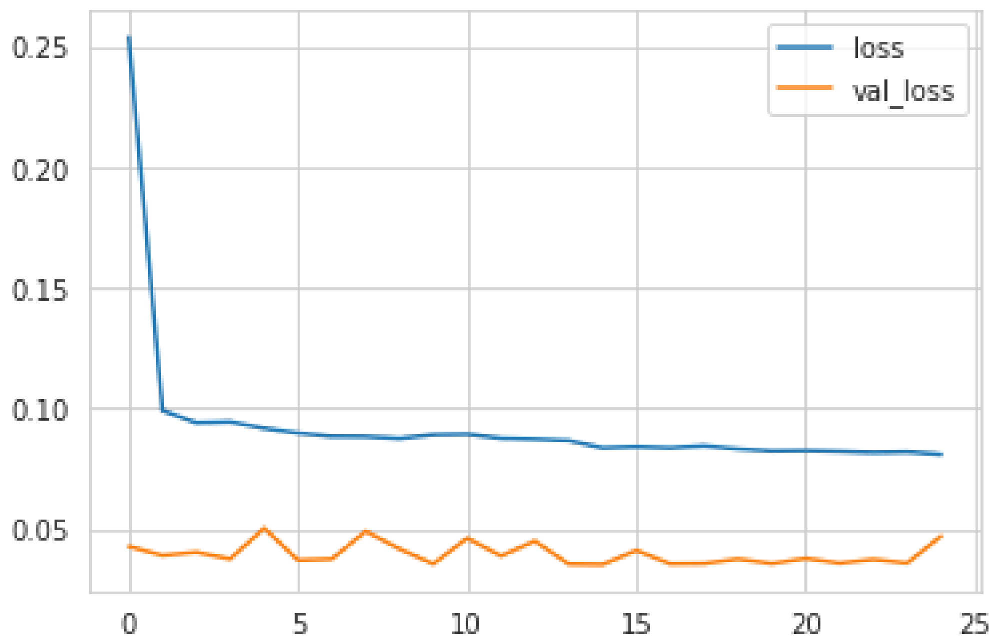
```
0.0412
Epoch 17/100
22/22 [==============================] - ETA: 0s - loss: 0.0837
Epoch 00017: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 14ms/step - loss: 0.0837 - val_loss:
0.0356
Epoch 18/100
20/22 [==========================>…] - ETA: 0s - loss: 0.0859
Epoch 00018: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 15ms/step - loss: 0.0845 - val_loss:
0.0357
Epoch 19/100
20/22 [==========================>…] - ETA: 0s - loss: 0.0845
Epoch 00019: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 14ms/step - loss: 0.0832 - val_loss:
0.0376
Epoch 20/100
20/22 [==========================>…] - ETA: 0s - loss: 0.0837
Epoch 00020: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 13ms/step - loss: 0.0824 - val_loss:
0.0357
Epoch 21/100
18/22 [=======================>…] - ETA: 0s - loss: 0.0839
Epoch 00021: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 14ms/step - loss: 0.0825 - val_loss:
0.0379
Epoch 22/100
21/22 [===========================>..] - ETA: 0s - loss: 0.0827
Epoch 00022: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 14ms/step - loss: 0.0822 - val_loss:
0.0359
Epoch 23/100
22/22 [==============================] - ETA: 0s - loss: 0.0818
Epoch 00023: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 13ms/step - loss: 0.0818 - val_loss:
0.0375
Epoch 24/100
21/22 [===========================>..] - ETA: 0s - loss: 0.0823
Epoch 00024: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 15ms/step - loss: 0.0820 - val_loss:
0.0359
Epoch 25/100
18/22 [=======================>…] - ETA: 0s - loss: 0.0823
Epoch 00025: val_loss did not improve from 0.03534
22/22 [==============================] - 0s 13ms/step - loss: 0.0810 - val_loss:
0.0471
```

1.6 Evaluate the Results

Training stops early after 22 epochs, yielding a test MAE of 1.71, which compares favorably to the test MAE for the VAR model of 1.91.

However, the two results are not fully comparable because the RNN model produces 24 one-step-ahead forecasts, whereas the VAR model uses its own predictions as input for its out-of-sample forecast. You may want to tweak the VAR setup to obtain comparable forecasts and compare their performance:

[29]:
```
pd.DataFrame(result.history).plot();
```



[30]:
```
y_pred = pd.DataFrame(rnn.predict(X_test),
                      columns=y_test.columns,
                      index=y_test.index)
y_pred.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 24 entries, 2018-01-01 to 2019-12-01
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ip         24 non-null     float32
 1   sentiment  24 non-null     float32
dtypes: float32(2)
memory usage: 384.0 bytes
```

[31]:
```
test_mae = mean_absolute_error(y_pred, y_test)
```

[32]:
```
print(test_mae)
```

```
0.03533523602534612
```

[33]:
```
y_test.index
```

[33]:
```
DatetimeIndex(['2018-01-01', '2018-02-01', '2018-03-01', '2018-04-01',
               '2018-05-01', '2018-06-01', '2018-07-01', '2018-08-01',
               '2018-09-01', '2018-10-01', '2018-11-01', '2018-12-01',
               '2019-01-01', '2019-02-01', '2019-03-01', '2019-04-01',
               '2019-05-01', '2019-06-01', '2019-07-01', '2019-08-01',
               '2019-09-01', '2019-10-01', '2019-11-01', '2019-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None)
```

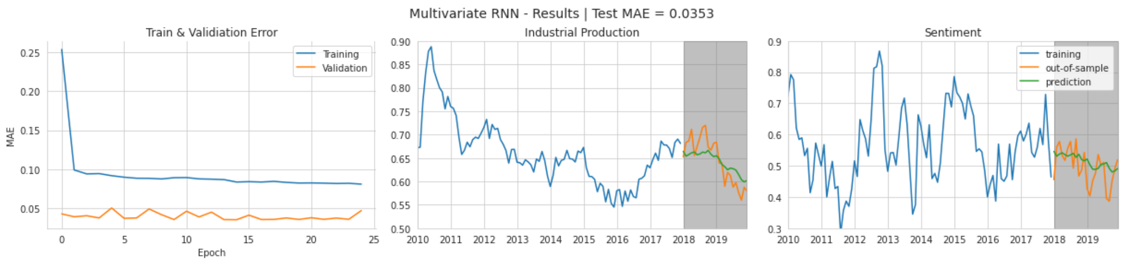
[34]:
```
fig, axes = plt.subplots(ncols=3, figsize=(17, 4))
pd.DataFrame(result.history).rename(columns={'loss': 'Training',
                                              'val_loss': 'Validation'}).
 ↪plot(ax=axes[0], title='Train & Validiation Error')
axes[0].set_xlabel('Epoch')
axes[0].set_ylabel('MAE')
col_dict = {'ip': 'Industrial Production', 'sentiment': 'Sentiment'}

for i, col in enumerate(y_test.columns, 1):
    y_train.loc['2010':, col].plot(ax=axes[i], label='training',␣
 ↪title=col_dict[col])
    y_test[col].plot(ax=axes[i], label='out-of-sample')
    y_pred[col].plot(ax=axes[i], label='prediction')
    axes[i].set_xlabel('')

axes[1].set_ylim(.5, .9)
axes[1].fill_between(x=y_test.index, y1=0.5, y2=0.9, color='grey', alpha=.5)

axes[2].set_ylim(.3, .9)
axes[2].fill_between(x=y_test.index, y1=0.3, y2=0.9, color='grey', alpha=.5)

plt.legend()
fig.suptitle('Multivariate RNN - Results | Test MAE = {:.4f}'.format(test_mae),␣
 ↪fontsize=14)
sns.despine()
fig.tight_layout()
fig.subplots_adjust(top=.85)
fig.savefig(results_path / 'multivariate_results', dpi=300);
```



Multivariate RNN - Results | Test MAE = 0.0353
Train & Validiation Error
Training
Validation
MAE
Epoch
Industrial Production
Sentiment
training
out-of-sample
prediction