PS21

January 26, 2024

1 Mini-project #2 (RNN exercise)

1.1 Task: Weather prediction

1.2 Loading Jena climate dataset & propreprocessing

[1]:
```
import pandas as pd
data = pd.read_csv('jena_climate_2009_2016.csv')

# fill up the missing entries with the mean
wv = data['wv (m/s)']
wv_missing_idx = (wv == -9999.00)
wv_mean = wv[~wv_missing_idx].mean()
wv[wv_missing_idx] = wv_mean

max_wv = data['max. wv (m/s)']
missing_idx = (max_wv == -9999.00)
max_wv_mean = max_wv[~missing_idx].mean()
max_wv[missing_idx] = max_wv_mean

# revmoe 'data time' column
data.pop('Date Time')

# downsampling
data = data[0::6] # m=70,092
```

```
C:\Users\chsuh\AppData\Local\Temp\ipykernel_33720\391320505.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  wv[wv_missing_idx] = wv_mean
C:\Users\chsuh\AppData\Local\Temp\ipykernel_33720\391320505.py:13:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
 max_wv[missing_idx] = max_wv_mean
```

1.3 Normalization & splitting

[2]:
```python
features = data
labels = data[['T (degC)']]

# normalization
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
features = std_scaler.fit_transform(features)

# 7:2:1 splitting
from sklearn.model_selection import train_test_split
X_rest, X_test, y_rest, y_test = train_test_split(features,
                                                  labels,
                                                  test_size=0.1,
                                                  shuffle=False)
X_train, X_val, y_train, y_val = train_test_split(X_rest,
                                                  y_rest,
                                                  test_size=2/9,
                                                  shuffle=False)
```

1.4 Time series data generation: Construct $\{(x_T^{(i)}, y_T^{(i)})\}_{i=1}^{m_T}$

[3]:
```python
T = 24
batch_size = 16

from tensorflow.keras.preprocessing import timeseries_dataset_from_array
# Train batch dataset
dataset_train = timeseries_dataset_from_array(X_train[:-T],
                                              y_train[T:],
                                              sequence_length = T,
                                              sequence_stride = 1,
                                              batch_size = batch_size,
                                              shuffle = True)
# validation batch dataset
dataset_val = timeseries_dataset_from_array(X_val[:-T],
                                            y_val[T:],
                                            sequence_length = T,
                                            sequence_stride = 1,
                                            batch_size=batch_size,
                                            shuffle = False)
# test batch dataset
dataset_test = timeseries_dataset_from_array(X_test[:-T],
                                             y_test[T:],
                                             sequence_length = T,
                                             sequence_stride = 1,
                                             batch_size=batch_size,
                                             shuffle = False)
```

1.5 DNN model

[4]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,Flatten,Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
```

[5]:
```
# DNN model construction
inputs = Input(shape=(T,14))
x = Flatten()(inputs)
x = Dense(units=32,activation='relu')(x)
outputs = Dense(units=1)(x)
dnn_model = Model(inputs=inputs,outputs=outputs)
dnn_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 24, 14)]          0

 flatten (Flatten)           (None, 336)               0

 dense (Dense)               (None, 32)                10784

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 10,817
Trainable params: 10,817
Non-trainable params: 0
_________________________________________________________________
```

[6]:
```
# Early stopping & learning rate decay
es_callback = EarlyStopping(monitor='val_loss',patience=10)
def scheduler(epoch,lr):
    if epoch in [10,20,30]: lr = 0.1*lr
    return lr
lrs_callback = LearningRateScheduler(scheduler)
```

[7]:
```
# Optimizer
opt = Adam(learning_rate=0.001,
           beta_1=0.9,
           beta_2=0.999)
```

[8]:
```python
# Compile
from tensorflow.keras.metrics import RootMeanSquaredError
dnn_model.compile(loss='mean_squared_error',
                 metrics=RootMeanSquaredError(),
                 optimizer=opt)
```

[9]:
```python
# Training
hist= dnn_model.fit(dataset_train,
                    epochs=30,
                    validation_data=dataset_val,
                    callbacks=[es_callback,lrs_callback])
```

```
Epoch 1/30
3064/3064 [==============================] - 16s 4ms/step - loss: 4.0462 -
root_mean_squared_error: 2.0115 - val_loss: 0.9511 -
val_root_mean_squared_error: 0.9752 - lr: 0.0010
Epoch 2/30
3064/3064 [==============================] - 14s 5ms/step - loss: 0.8718 -
root_mean_squared_error: 0.9337 - val_loss: 0.6878 -
val_root_mean_squared_error: 0.8294 - lr: 0.0010
Epoch 3/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.7640 -
root_mean_squared_error: 0.8740 - val_loss: 0.6770 -
val_root_mean_squared_error: 0.8228 - lr: 0.0010
Epoch 4/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.7125 -
root_mean_squared_error: 0.8441 - val_loss: 0.7528 -
val_root_mean_squared_error: 0.8676 - lr: 0.0010
Epoch 5/30
3064/3064 [==============================] - 14s 4ms/step - loss: 0.6894 -
root_mean_squared_error: 0.8303 - val_loss: 0.6323 -
val_root_mean_squared_error: 0.7952 - lr: 0.0010
Epoch 6/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.6596 -
root_mean_squared_error: 0.8122 - val_loss: 0.6292 -
val_root_mean_squared_error: 0.7932 - lr: 0.0010
Epoch 7/30
3064/3064 [==============================] - 14s 4ms/step - loss: 0.6408 -
root_mean_squared_error: 0.8005 - val_loss: 0.5819 -
val_root_mean_squared_error: 0.7628 - lr: 0.0010
Epoch 8/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.6311 -
root_mean_squared_error: 0.7944 - val_loss: 0.5787 -
val_root_mean_squared_error: 0.7607 - lr: 0.0010
Epoch 9/30
3064/3064 [==============================] - 12s 4ms/step - loss: 0.6204 -
```

```
root_mean_squared_error: 0.7877 - val_loss: 0.5522 -
val_root_mean_squared_error: 0.7431 - lr: 0.0010
Epoch 10/30
3064/3064 [==============================] - 14s 4ms/step - loss: 0.5998 -
root_mean_squared_error: 0.7745 - val_loss: 0.6124 -
val_root_mean_squared_error: 0.7826 - lr: 0.0010
Epoch 11/30
3064/3064 [==============================] - 15s 5ms/step - loss: 0.5054 -
root_mean_squared_error: 0.7109 - val_loss: 0.5235 -
val_root_mean_squared_error: 0.7235 - lr: 1.0000e-04
Epoch 12/30
3064/3064 [==============================] - 15s 5ms/step - loss: 0.4998 -
root_mean_squared_error: 0.7069 - val_loss: 0.5214 -
val_root_mean_squared_error: 0.7221 - lr: 1.0000e-04
Epoch 13/30
3064/3064 [==============================] - 14s 5ms/step - loss: 0.4980 -
root_mean_squared_error: 0.7057 - val_loss: 0.5211 -
val_root_mean_squared_error: 0.7219 - lr: 1.0000e-04
Epoch 14/30
3064/3064 [==============================] - 11s 4ms/step - loss: 0.4963 -
root_mean_squared_error: 0.7045 - val_loss: 0.5212 -
val_root_mean_squared_error: 0.7219 - lr: 1.0000e-04
Epoch 15/30
3064/3064 [==============================] - 12s 4ms/step - loss: 0.4947 -
root_mean_squared_error: 0.7033 - val_loss: 0.5221 -
val_root_mean_squared_error: 0.7226 - lr: 1.0000e-04
Epoch 16/30
3064/3064 [==============================] - 14s 5ms/step - loss: 0.4939 -
root_mean_squared_error: 0.7028 - val_loss: 0.5209 -
val_root_mean_squared_error: 0.7217 - lr: 1.0000e-04
Epoch 17/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.4932 -
root_mean_squared_error: 0.7023 - val_loss: 0.5204 -
val_root_mean_squared_error: 0.7214 - lr: 1.0000e-04
Epoch 18/30
3064/3064 [==============================] - 12s 4ms/step - loss: 0.4920 -
root_mean_squared_error: 0.7014 - val_loss: 0.5191 -
val_root_mean_squared_error: 0.7205 - lr: 1.0000e-04
Epoch 19/30
3064/3064 [==============================] - 14s 5ms/step - loss: 0.4912 -
root_mean_squared_error: 0.7008 - val_loss: 0.5173 -
val_root_mean_squared_error: 0.7193 - lr: 1.0000e-04
Epoch 20/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.4909 -
root_mean_squared_error: 0.7006 - val_loss: 0.5163 -
val_root_mean_squared_error: 0.7186 - lr: 1.0000e-04
Epoch 21/30
3064/3064 [==============================] - 15s 5ms/step - loss: 0.4798 -
```

```
root_mean_squared_error: 0.6927 - val_loss: 0.5106 -
val_root_mean_squared_error: 0.7145 - lr: 1.0000e-05
Epoch 22/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.4790 -
root_mean_squared_error: 0.6921 - val_loss: 0.5104 -
val_root_mean_squared_error: 0.7144 - lr: 1.0000e-05
Epoch 23/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.4787 -
root_mean_squared_error: 0.6919 - val_loss: 0.5103 -
val_root_mean_squared_error: 0.7143 - lr: 1.0000e-05
Epoch 24/30
3064/3064 [==============================] - 14s 5ms/step - loss: 0.4786 -
root_mean_squared_error: 0.6918 - val_loss: 0.5102 -
val_root_mean_squared_error: 0.7143 - lr: 1.0000e-05
Epoch 25/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.4785 -
root_mean_squared_error: 0.6917 - val_loss: 0.5101 -
val_root_mean_squared_error: 0.7142 - lr: 1.0000e-05
Epoch 26/30
3064/3064 [==============================] - 14s 4ms/step - loss: 0.4784 -
root_mean_squared_error: 0.6916 - val_loss: 0.5099 -
val_root_mean_squared_error: 0.7141 - lr: 1.0000e-05
Epoch 27/30
3064/3064 [==============================] - 14s 5ms/step - loss: 0.4782 -
root_mean_squared_error: 0.6915 - val_loss: 0.5098 -
val_root_mean_squared_error: 0.7140 - lr: 1.0000e-05
Epoch 28/30
3064/3064 [==============================] - 13s 4ms/step - loss: 0.4781 -
root_mean_squared_error: 0.6915 - val_loss: 0.5098 -
val_root_mean_squared_error: 0.7140 - lr: 1.0000e-05
Epoch 29/30
3064/3064 [==============================] - 14s 5ms/step - loss: 0.4780 -
root_mean_squared_error: 0.6914 - val_loss: 0.5098 -
val_root_mean_squared_error: 0.7140 - lr: 1.0000e-05
Epoch 30/30
3064/3064 [==============================] - 14s 5ms/step - loss: 0.4779 -
root_mean_squared_error: 0.6913 - val_loss: 0.5097 -
val_root_mean_squared_error: 0.7139 - lr: 1.0000e-05
```

[10]:
```
# Evaluate normalized RMSE
eval_rmse = dnn_model.evaluate(dataset_test)[1]
eval_nrmse = eval_rmse/y_test.std()
print(eval_nrmse)
```

```
436/436 [==============================] - 2s 4ms/step - loss: 0.4932 -
root_mean_squared_error: 0.7023
T (degC)    0.089477
dtype: float64
```

1.6 LSTM model

[11]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Flatten, Dense,LSTM
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
```

[12]:
```
# LSTM model construction
inputs = Input(shape=(T,14))
x = LSTM(units=32,return_sequences=True)(inputs)
x = LSTM(units=32)(x)
outputs=Dense(units=1)(x)
rnn_model = Model(inputs=inputs, outputs=outputs)
rnn_model.summary()
```

```
Model: "model_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 24, 14)]          0

 lstm (LSTM)                 (None, 24, 32)            6016

 lstm_1 (LSTM)               (None, 32)                8320

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 14,369
Trainable params: 14,369
Non-trainable params: 0
_________________________________________________________________
```

[13]:
```
# Early stopping & learning rate decay
es_callback = EarlyStopping(monitor='val_loss',patience=10)
def scheduler(epoch,lr):
    if epoch in [10,20,30]: lr = 0.1*lr
    return lr
lrs_callback = LearningRateScheduler(scheduler)
```

[14]:
```
# Optimizer
opt = Adam(learning_rate=0.001,
          beta_1=0.9,
          beta_2=0.999)
```

[15]:
```
# Compile
from tensorflow.keras.metrics import RootMeanSquaredError
```

```
rnn_model.compile(loss='mean_squared_error',
                  metrics=RootMeanSquaredError(),
                  optimizer=opt)
```

[16]:
```
# Training
history = rnn_model.fit(dataset_train,
                        epochs=30,
                        validation_data=dataset_val,
                        callbacks=[es_callback,lrs_callback])
```

```
Epoch 1/30
3064/3064 [==============================] - 40s 11ms/step - loss: 7.4458 -
root_mean_squared_error: 2.7287 - val_loss: 0.7951 -
val_root_mean_squared_error: 0.8917 - lr: 0.0010
Epoch 2/30
3064/3064 [==============================] - 31s 10ms/step - loss: 0.6251 -
root_mean_squared_error: 0.7906 - val_loss: 0.5424 -
val_root_mean_squared_error: 0.7365 - lr: 0.0010
Epoch 3/30
3064/3064 [==============================] - 31s 10ms/step - loss: 0.5638 -
root_mean_squared_error: 0.7509 - val_loss: 0.5172 -
val_root_mean_squared_error: 0.7191 - lr: 0.0010
Epoch 4/30
3064/3064 [==============================] - 34s 11ms/step - loss: 0.5471 -
root_mean_squared_error: 0.7397 - val_loss: 0.5013 -
val_root_mean_squared_error: 0.7080 - lr: 0.0010
Epoch 5/30
3064/3064 [==============================] - 31s 10ms/step - loss: 0.5363 -
root_mean_squared_error: 0.7323 - val_loss: 0.5083 -
val_root_mean_squared_error: 0.7130 - lr: 0.0010
Epoch 6/30
3064/3064 [==============================] - 33s 11ms/step - loss: 0.5289 -
root_mean_squared_error: 0.7273 - val_loss: 0.5260 -
val_root_mean_squared_error: 0.7253 - lr: 0.0010
Epoch 7/30
3064/3064 [==============================] - 33s 11ms/step - loss: 0.5193 -
root_mean_squared_error: 0.7206 - val_loss: 0.4923 -
val_root_mean_squared_error: 0.7016 - lr: 0.0010
Epoch 8/30
3064/3064 [==============================] - 34s 11ms/step - loss: 0.5124 -
root_mean_squared_error: 0.7158 - val_loss: 0.4806 -
val_root_mean_squared_error: 0.6932 - lr: 0.0010
Epoch 9/30
3064/3064 [==============================] - 33s 11ms/step - loss: 0.5062 -
root_mean_squared_error: 0.7114 - val_loss: 0.4914 -
val_root_mean_squared_error: 0.7010 - lr: 0.0010
Epoch 10/30
3064/3064 [==============================] - 32s 11ms/step - loss: 0.5001 -
```

```
root_mean_squared_error: 0.7071 - val_loss: 0.5031 -
val_root_mean_squared_error: 0.7093 - lr: 0.0010
Epoch 11/30
3064/3064 [==============================] - 34s 11ms/step - loss: 0.4567 -
root_mean_squared_error: 0.6758 - val_loss: 0.4541 -
val_root_mean_squared_error: 0.6739 - lr: 1.0000e-04
Epoch 12/30
3064/3064 [==============================] - 33s 11ms/step - loss: 0.4514 -
root_mean_squared_error: 0.6719 - val_loss: 0.4521 -
val_root_mean_squared_error: 0.6724 - lr: 1.0000e-04
Epoch 13/30
3064/3064 [==============================] - 32s 11ms/step - loss: 0.4491 -
root_mean_squared_error: 0.6701 - val_loss: 0.4512 -
val_root_mean_squared_error: 0.6717 - lr: 1.0000e-04
Epoch 14/30
3064/3064 [==============================] - 30s 10ms/step - loss: 0.4478 -
root_mean_squared_error: 0.6692 - val_loss: 0.4524 -
val_root_mean_squared_error: 0.6726 - lr: 1.0000e-04
Epoch 15/30
3064/3064 [==============================] - 32s 11ms/step - loss: 0.4464 -
root_mean_squared_error: 0.6682 - val_loss: 0.4510 -
val_root_mean_squared_error: 0.6716 - lr: 1.0000e-04
Epoch 16/30
3064/3064 [==============================] - 31s 10ms/step - loss: 0.4453 -
root_mean_squared_error: 0.6673 - val_loss: 0.4512 -
val_root_mean_squared_error: 0.6717 - lr: 1.0000e-04
Epoch 17/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4443 -
root_mean_squared_error: 0.6666 - val_loss: 0.4510 -
val_root_mean_squared_error: 0.6715 - lr: 1.0000e-04
Epoch 18/30
3064/3064 [==============================] - 31s 10ms/step - loss: 0.4437 -
root_mean_squared_error: 0.6661 - val_loss: 0.4511 -
val_root_mean_squared_error: 0.6716 - lr: 1.0000e-04
Epoch 19/30
3064/3064 [==============================] - 31s 10ms/step - loss: 0.4426 -
root_mean_squared_error: 0.6653 - val_loss: 0.4508 -
val_root_mean_squared_error: 0.6714 - lr: 1.0000e-04
Epoch 20/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4417 -
root_mean_squared_error: 0.6646 - val_loss: 0.4503 -
val_root_mean_squared_error: 0.6711 - lr: 1.0000e-04
Epoch 21/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4364 -
root_mean_squared_error: 0.6606 - val_loss: 0.4491 -
val_root_mean_squared_error: 0.6702 - lr: 1.0000e-05
Epoch 22/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4358 -
```

```
root_mean_squared_error: 0.6601 - val_loss: 0.4492 -
val_root_mean_squared_error: 0.6702 - lr: 1.0000e-05
Epoch 23/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4356 -
root_mean_squared_error: 0.6600 - val_loss: 0.4493 -
val_root_mean_squared_error: 0.6703 - lr: 1.0000e-05
Epoch 24/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4354 -
root_mean_squared_error: 0.6599 - val_loss: 0.4493 -
val_root_mean_squared_error: 0.6703 - lr: 1.0000e-05
Epoch 25/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4353 -
root_mean_squared_error: 0.6598 - val_loss: 0.4494 -
val_root_mean_squared_error: 0.6704 - lr: 1.0000e-05
Epoch 26/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4352 -
root_mean_squared_error: 0.6597 - val_loss: 0.4494 -
val_root_mean_squared_error: 0.6704 - lr: 1.0000e-05
Epoch 27/30
3064/3064 [==============================] - 31s 10ms/step - loss: 0.4351 -
root_mean_squared_error: 0.6596 - val_loss: 0.4494 -
val_root_mean_squared_error: 0.6704 - lr: 1.0000e-05
Epoch 28/30
3064/3064 [==============================] - 33s 11ms/step - loss: 0.4349 -
root_mean_squared_error: 0.6595 - val_loss: 0.4495 -
val_root_mean_squared_error: 0.6704 - lr: 1.0000e-05
Epoch 29/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4349 -
root_mean_squared_error: 0.6594 - val_loss: 0.4495 -
val_root_mean_squared_error: 0.6704 - lr: 1.0000e-05
Epoch 30/30
3064/3064 [==============================] - 32s 10ms/step - loss: 0.4348 -
root_mean_squared_error: 0.6594 - val_loss: 0.4495 -
val_root_mean_squared_error: 0.6705 - lr: 1.0000e-05
```

[17]:
```
# Evaluate normalized RMSE
eval_rmse = rnn_model.evaluate(dataset_test)[1]
eval_nrmse = eval_rmse/y_test.std()
print(eval_nrmse)
```

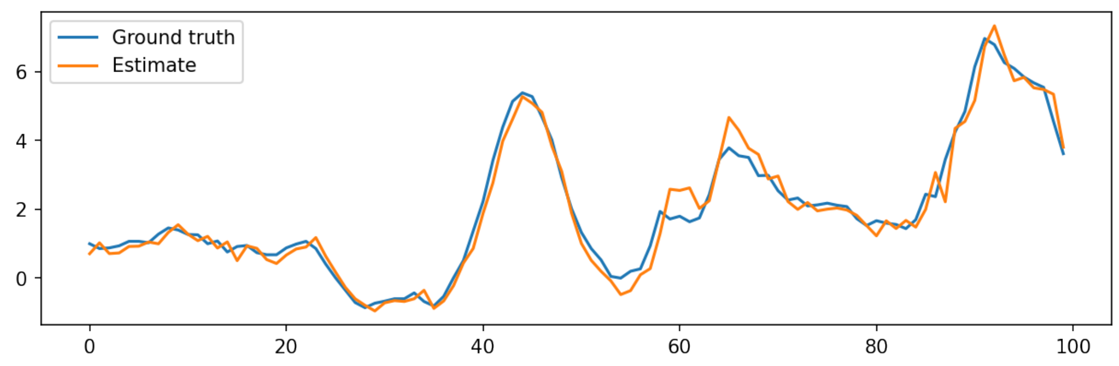
```
436/436 [==============================] - 3s 6ms/step - loss: 0.4399 -
root_mean_squared_error: 0.6633
T (degC)    0.084508
dtype: float64
```

[18]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,3), dpi=150)
plt.plot(y_test[T:100+T].values)
estimated = rnn_model.predict(dataset_test)
plt.plot(estimated[:100])
plt.legend(['Ground truth', 'Estimate'])
```

```
436/436 [==============================] - 3s 5ms/step
```

```
[18]: <matplotlib.legend.Legend at 0x2176e043c10>
```



```
[ ]:
```