PS22

January 26, 2023

1 Mini-project #2 (RNN exercise)

1.1 Task: Weather prediction

1.2 Loading Jena climate dataset & propreprocessing

[1]:

```python
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_43892/391320505.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_43892/391320505.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 $\{(x_T^{(i)}, y_T^{(i)})\}_{i=1}^{m_T}$

[9]:

```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

[10]:

```
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
```

[13]:

```
# 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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 24, 14)]          0

 flatten_1 (Flatten)         (None, 336)               0

 dense_2 (Dense)             (None, 32)                10784

 dense_3 (Dense)             (None, 1)                 33

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

[14]:

```
# 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)
```

[15]:

```
# Optimizer
opt = Adam(learning_rate=0.001,
          beta_1=0.9,
          beta_2=0.999)
```

[19]:

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

[17]:

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

```
Epoch 1/30
3064/3064 [==============================] - 3s 895us/step - loss: 5.3481 -
root_mean_squared_error: 2.3126 - val_loss: 1.0391 -
val_root_mean_squared_error: 1.0193 - lr: 0.0010
Epoch 2/30
3064/3064 [==============================] - 2s 764us/step - loss: 0.9471 -
root_mean_squared_error: 0.9732 - val_loss: 1.2296 -
val_root_mean_squared_error: 1.1089 - lr: 0.0010
Epoch 3/30
3064/3064 [==============================] - 2s 758us/step - loss: 0.7985 -
root_mean_squared_error: 0.8936 - val_loss: 0.8527 -
val_root_mean_squared_error: 0.9234 - lr: 0.0010
Epoch 4/30
3064/3064 [==============================] - 2s 775us/step - loss: 0.7408 -
root_mean_squared_error: 0.8607 - val_loss: 0.6701 -
val_root_mean_squared_error: 0.8186 - lr: 0.0010
Epoch 5/30
3064/3064 [==============================] - 2s 763us/step - loss: 0.7042 -
root_mean_squared_error: 0.8392 - val_loss: 0.6997 -
val_root_mean_squared_error: 0.8365 - lr: 0.0010
Epoch 6/30
3064/3064 [==============================] - 2s 771us/step - loss: 0.6797 -
root_mean_squared_error: 0.8245 - val_loss: 0.7347 -
val_root_mean_squared_error: 0.8571 - lr: 0.0010
Epoch 7/30
3064/3064 [==============================] - 2s 765us/step - loss: 0.6591 -
root_mean_squared_error: 0.8119 - val_loss: 0.6716 -
val_root_mean_squared_error: 0.8195 - lr: 0.0010
Epoch 8/30
3064/3064 [==============================] - 2s 766us/step - loss: 0.6446 -
```

```
root_mean_squared_error: 0.8028 - val_loss: 0.6347 -
val_root_mean_squared_error: 0.7967 - lr: 0.0010
Epoch 9/30
3064/3064 [==============================] - 2s 794us/step - loss: 0.6365 -
root_mean_squared_error: 0.7978 - val_loss: 0.6149 -
val_root_mean_squared_error: 0.7842 - lr: 0.0010
Epoch 10/30
3064/3064 [==============================] - 2s 783us/step - loss: 0.6231 -
root_mean_squared_error: 0.7894 - val_loss: 0.6145 -
val_root_mean_squared_error: 0.7839 - lr: 0.0010
Epoch 11/30
3064/3064 [==============================] - 2s 789us/step - loss: 0.5215 -
root_mean_squared_error: 0.7221 - val_loss: 0.5309 -
val_root_mean_squared_error: 0.7286 - lr: 1.0000e-04
Epoch 12/30
3064/3064 [==============================] - 2s 768us/step - loss: 0.5138 -
root_mean_squared_error: 0.7168 - val_loss: 0.5274 -
val_root_mean_squared_error: 0.7262 - lr: 1.0000e-04
Epoch 13/30
3064/3064 [==============================] - 2s 766us/step - loss: 0.5118 -
root_mean_squared_error: 0.7154 - val_loss: 0.5264 -
val_root_mean_squared_error: 0.7255 - lr: 1.0000e-04
Epoch 14/30
3064/3064 [==============================] - 2s 790us/step - loss: 0.5089 -
root_mean_squared_error: 0.7134 - val_loss: 0.5236 -
val_root_mean_squared_error: 0.7236 - lr: 1.0000e-04
Epoch 15/30
3064/3064 [==============================] - 2s 795us/step - loss: 0.5068 -
root_mean_squared_error: 0.7119 - val_loss: 0.5321 -
val_root_mean_squared_error: 0.7295 - lr: 1.0000e-04
Epoch 16/30
3064/3064 [==============================] - 2s 787us/step - loss: 0.5058 -
root_mean_squared_error: 0.7112 - val_loss: 0.5222 -
val_root_mean_squared_error: 0.7227 - lr: 1.0000e-04
Epoch 17/30
3064/3064 [==============================] - 2s 761us/step - loss: 0.5041 -
root_mean_squared_error: 0.7100 - val_loss: 0.5263 -
val_root_mean_squared_error: 0.7255 - lr: 1.0000e-04
Epoch 18/30
3064/3064 [==============================] - 2s 781us/step - loss: 0.5027 -
root_mean_squared_error: 0.7090 - val_loss: 0.5209 -
val_root_mean_squared_error: 0.7217 - lr: 1.0000e-04
Epoch 19/30
3064/3064 [==============================] - 2s 761us/step - loss: 0.5012 -
root_mean_squared_error: 0.7080 - val_loss: 0.5213 -
val_root_mean_squared_error: 0.7220 - lr: 1.0000e-04
Epoch 20/30
3064/3064 [==============================] - 2s 786us/step - loss: 0.5011 -
```

```
root_mean_squared_error: 0.7079 - val_loss: 0.5238 -
val_root_mean_squared_error: 0.7237 - lr: 1.0000e-04
Epoch 21/30
3064/3064 [==============================] - 2s 765us/step - loss: 0.4897 -
root_mean_squared_error: 0.6998 - val_loss: 0.5145 -
val_root_mean_squared_error: 0.7173 - lr: 1.0000e-05
Epoch 22/30
3064/3064 [==============================] - 2s 782us/step - loss: 0.4886 -
root_mean_squared_error: 0.6990 - val_loss: 0.5139 -
val_root_mean_squared_error: 0.7169 - lr: 1.0000e-05
Epoch 23/30
3064/3064 [==============================] - 2s 800us/step - loss: 0.4883 -
root_mean_squared_error: 0.6988 - val_loss: 0.5139 -
val_root_mean_squared_error: 0.7169 - lr: 1.0000e-05
Epoch 24/30
3064/3064 [==============================] - 2s 805us/step - loss: 0.4881 -
root_mean_squared_error: 0.6986 - val_loss: 0.5135 -
val_root_mean_squared_error: 0.7166 - lr: 1.0000e-05
Epoch 25/30
3064/3064 [==============================] - 2s 791us/step - loss: 0.4879 -
root_mean_squared_error: 0.6985 - val_loss: 0.5134 -
val_root_mean_squared_error: 0.7165 - lr: 1.0000e-05
Epoch 26/30
3064/3064 [==============================] - 3s 819us/step - loss: 0.4878 -
root_mean_squared_error: 0.6985 - val_loss: 0.5134 -
val_root_mean_squared_error: 0.7166 - lr: 1.0000e-05
Epoch 27/30
3064/3064 [==============================] - 2s 801us/step - loss: 0.4876 -
root_mean_squared_error: 0.6983 - val_loss: 0.5134 -
val_root_mean_squared_error: 0.7166 - lr: 1.0000e-05
Epoch 28/30
3064/3064 [==============================] - 2s 806us/step - loss: 0.4874 -
root_mean_squared_error: 0.6982 - val_loss: 0.5132 -
val_root_mean_squared_error: 0.7164 - lr: 1.0000e-05
Epoch 29/30
3064/3064 [==============================] - 3s 814us/step - loss: 0.4873 -
root_mean_squared_error: 0.6981 - val_loss: 0.5130 -
val_root_mean_squared_error: 0.7163 - lr: 1.0000e-05
Epoch 30/30
3064/3064 [==============================] - 2s 800us/step - loss: 0.4872 -
root_mean_squared_error: 0.6980 - val_loss: 0.5130 -
val_root_mean_squared_error: 0.7162 - lr: 1.0000e-05
```

[18]:

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

```
436/436 [==============================] - 1s 2ms/step - loss: 0.5136 -
root_mean_squared_error: 0.7167
T (degC)    0.091312
dtype: float64
```

1.6 LSTM model

[21]:

```python
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
```

[22]:

```python
# 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_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 24, 14)]          0

 lstm_2 (LSTM)               (None, 24, 32)            6016

 lstm_3 (LSTM)               (None, 32)                8320

 dense_5 (Dense)             (None, 1)                 33

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

[26]:

```python
# 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)
```

[27]:

```python
# Optimizer
opt = Adam(learning_rate=0.001,
          beta_1=0.9,
          beta_2=0.999)
```

[32]:

```python
# Compile
from tensorflow.keras.metrics import RootMeanSquaredError
rnn_model.compile(loss='mean_squared_error',
                 metrics=RootMeanSquaredError(),
                 optimizer=opt)
```

[33]:

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

```
Epoch 1/30
3064/3064 [==============================] - 25s 7ms/step - loss: 6.9035 -
root_mean_squared_error: 2.6275 - val_loss: 0.7795 -
val_root_mean_squared_error: 0.8829 - lr: 0.0010
Epoch 2/30
3064/3064 [==============================] - 23s 7ms/step - loss: 0.6391 -
root_mean_squared_error: 0.7994 - val_loss: 0.5613 -
val_root_mean_squared_error: 0.7492 - lr: 0.0010
Epoch 3/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.5728 -
root_mean_squared_error: 0.7568 - val_loss: 0.5412 -
val_root_mean_squared_error: 0.7356 - lr: 0.0010
Epoch 4/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.5520 -
root_mean_squared_error: 0.7430 - val_loss: 0.5379 -
val_root_mean_squared_error: 0.7335 - lr: 0.0010
Epoch 5/30
3064/3064 [==============================] - 25s 8ms/step - loss: 0.5380 -
root_mean_squared_error: 0.7335 - val_loss: 0.5168 -
val_root_mean_squared_error: 0.7189 - lr: 0.0010
Epoch 6/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.5286 -
root_mean_squared_error: 0.7270 - val_loss: 0.5096 -
val_root_mean_squared_error: 0.7139 - lr: 0.0010
Epoch 7/30
3064/3064 [==============================] - 25s 8ms/step - loss: 0.5192 -
root_mean_squared_error: 0.7205 - val_loss: 0.5084 -
val_root_mean_squared_error: 0.7130 - lr: 0.0010
Epoch 8/30
3064/3064 [==============================] - 26s 8ms/step - loss: 0.5133 -
```

```
root_mean_squared_error: 0.7164 - val_loss: 0.4952 -
val_root_mean_squared_error: 0.7037 - lr: 0.0010
Epoch 9/30
3064/3064 [==============================] - 27s 9ms/step - loss: 0.5066 -
root_mean_squared_error: 0.7118 - val_loss: 0.5506 -
val_root_mean_squared_error: 0.7420 - lr: 0.0010
Epoch 10/30
3064/3064 [==============================] - 26s 9ms/step - loss: 0.4996 -
root_mean_squared_error: 0.7068 - val_loss: 0.5169 -
val_root_mean_squared_error: 0.7190 - lr: 0.0010
Epoch 11/30
3064/3064 [==============================] - 28s 9ms/step - loss: 0.4544 -
root_mean_squared_error: 0.6741 - val_loss: 0.4550 -
val_root_mean_squared_error: 0.6745 - lr: 1.0000e-04
Epoch 12/30
3064/3064 [==============================] - 29s 10ms/step - loss: 0.4493 -
root_mean_squared_error: 0.6703 - val_loss: 0.4525 -
val_root_mean_squared_error: 0.6727 - lr: 1.0000e-04
Epoch 13/30
3064/3064 [==============================] - 29s 9ms/step - loss: 0.4470 -
root_mean_squared_error: 0.6686 - val_loss: 0.4516 -
val_root_mean_squared_error: 0.6720 - lr: 1.0000e-04
Epoch 14/30
3064/3064 [==============================] - 28s 9ms/step - loss: 0.4458 -
root_mean_squared_error: 0.6677 - val_loss: 0.4521 -
val_root_mean_squared_error: 0.6724 - lr: 1.0000e-04
Epoch 15/30
3064/3064 [==============================] - 26s 9ms/step - loss: 0.4449 -
root_mean_squared_error: 0.6670 - val_loss: 0.4514 -
val_root_mean_squared_error: 0.6719 - lr: 1.0000e-04
Epoch 16/30
3064/3064 [==============================] - 26s 9ms/step - loss: 0.4439 -
root_mean_squared_error: 0.6663 - val_loss: 0.4524 -
val_root_mean_squared_error: 0.6726 - lr: 1.0000e-04
Epoch 17/30
3064/3064 [==============================] - 28s 9ms/step - loss: 0.4429 -
root_mean_squared_error: 0.6655 - val_loss: 0.4510 -
val_root_mean_squared_error: 0.6716 - lr: 1.0000e-04
Epoch 18/30
3064/3064 [==============================] - 27s 9ms/step - loss: 0.4419 -
root_mean_squared_error: 0.6647 - val_loss: 0.4535 -
val_root_mean_squared_error: 0.6734 - lr: 1.0000e-04
Epoch 19/30
3064/3064 [==============================] - 26s 9ms/step - loss: 0.4412 -
root_mean_squared_error: 0.6642 - val_loss: 0.4518 -
val_root_mean_squared_error: 0.6722 - lr: 1.0000e-04
Epoch 20/30
3064/3064 [==============================] - 26s 8ms/step - loss: 0.4401 -
```

```
root_mean_squared_error: 0.6634 - val_loss: 0.4505 -
val_root_mean_squared_error: 0.6712 - lr: 1.0000e-04
Epoch 21/30
3064/3064 [==============================] - 26s 9ms/step - loss: 0.4353 -
root_mean_squared_error: 0.6598 - val_loss: 0.4477 -
val_root_mean_squared_error: 0.6691 - lr: 1.0000e-05
Epoch 22/30
3064/3064 [==============================] - 26s 9ms/step - loss: 0.4344 -
root_mean_squared_error: 0.6591 - val_loss: 0.4475 -
val_root_mean_squared_error: 0.6689 - lr: 1.0000e-05
Epoch 23/30
3064/3064 [==============================] - 27s 9ms/step - loss: 0.4342 -
root_mean_squared_error: 0.6589 - val_loss: 0.4473 -
val_root_mean_squared_error: 0.6688 - lr: 1.0000e-05
Epoch 24/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.4340 -
root_mean_squared_error: 0.6588 - val_loss: 0.4472 -
val_root_mean_squared_error: 0.6688 - lr: 1.0000e-05
Epoch 25/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.4338 -
root_mean_squared_error: 0.6587 - val_loss: 0.4473 -
val_root_mean_squared_error: 0.6688 - lr: 1.0000e-05
Epoch 26/30
3064/3064 [==============================] - 25s 8ms/step - loss: 0.4337 -
root_mean_squared_error: 0.6586 - val_loss: 0.4472 -
val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
Epoch 27/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.4336 -
root_mean_squared_error: 0.6585 - val_loss: 0.4471 -
val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
Epoch 28/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.4335 -
root_mean_squared_error: 0.6584 - val_loss: 0.4471 -
val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
Epoch 29/30
3064/3064 [==============================] - 24s 8ms/step - loss: 0.4334 -
root_mean_squared_error: 0.6583 - val_loss: 0.4472 -
val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
Epoch 30/30
3064/3064 [==============================] - 25s 8ms/step - loss: 0.4333 -
root_mean_squared_error: 0.6583 - val_loss: 0.4471 -
val_root_mean_squared_error: 0.6687 - lr: 1.0000e-05
```

[34]:

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

```
436/436 [==============================] - 1s 3ms/step - loss: 0.4410 -
root_mean_squared_error: 0.6640
T (degC)    0.084609
dtype: float64
```

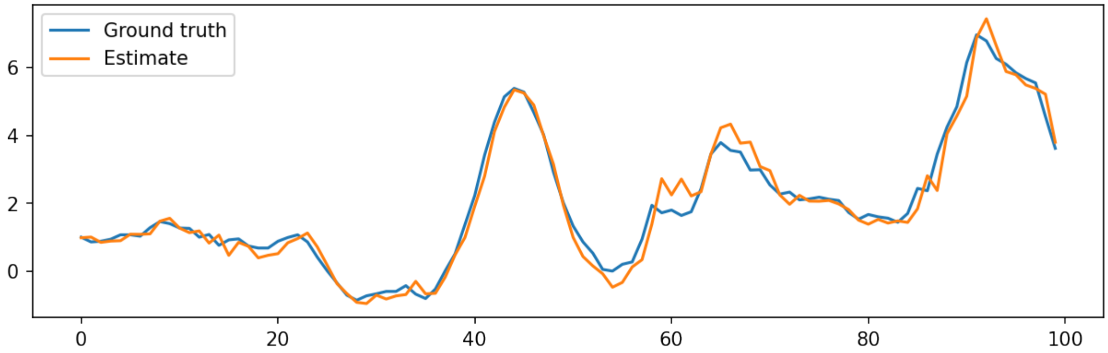
[35]:

```
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'])
```

[35]:

```
<matplotlib.legend.Legend at 0x1511526fd30>
```



[ ]: