```
"""
プログラム１：
気象データセットをダウンロードし、展開しておく。
"""
import os
import urllib
import zipfile

dir_path = "/content/drive/My Drive/Python/RNN/"
os.chdir(dir_path)

if not os.path.exists('jena_climate'):
    os.mkdir('jena_climate')
os.chdir('./jena_climate')

url = 'https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip'
urllib.request.urlretrieve(url, "jena_climate_2009_2016.csv.zip")

f = zipfile.ZipFile('./jena_climate_2009_2016.csv.zip','r')
for file in f.namelist():
    f.extract(file,"./")


"""
プログラム２：
データを正規化し、
訓練、検証、テストに使用するジェネレータを準備し、
GRUベースのモデルの訓練と評価をする
"""

import os
import numpy as np
import pprint
from keras.models import Sequential
from keras.layers import Dense, GRU
from keras.optimizers import RMSprop
import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt
from keras.callbacks import ModelCheckpoint

dir_path = "/content/drive/My Drive/Python/RNN/"
os.chdir(dir_path)

#  気象データセットのデータ調査
data_dir = '/content/drive/My Drive/Python/RNN/jena_climate'
fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')

f = open(fname)
data = f.read()
f.close()

lines = data.split('¥n')
header = lines[0].split(',')
header = [x[1: -1] for x in header]
lines = lines[1:]

pprint.pprint(header)

#  データの解析
float_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(',')[1:]]
    float_data[i, :] = values

print(float_data.shape)

#  データの正規化
mean = float_data[:200000].mean(axis=0)
float_data -= mean
std = float_data[:200000].std(axis=0)
float_data /= std

#  時系列サンプルとそれらの目的値を生成するジェネレータ
def generator(data, lookback, delay, step, min_index, max_index,
              batch_size, shuffle=False):
    if
```

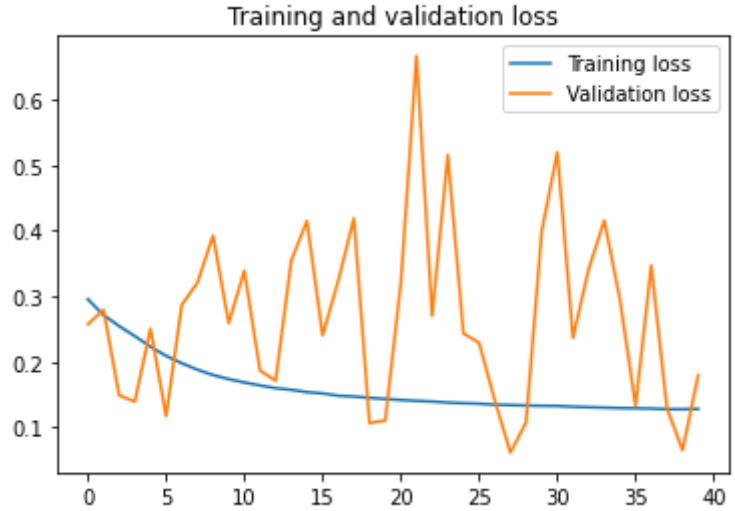
```
    if max_index is None:
        max_index = len(data) - delay
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(min_index + lookback, max_index, size=batch_size)
        else:
            if i >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)

        samples = np.zeros((len(rows), lookback // step, data.shape[-1]))
        targets = np.zeros((len(rows),))
        for j, row in enumerate(rows):
            indices = range(row - lookback, row, step)
            samples[j] = data[indices]
            targets[j] = data[row + delay][1]
        yield samples, targets

#  訓練、検証、テキストに使用するジェネレータの準備
lookback = 1440
delay = 144
step = 6
batch_size = 256

train_gen = generator(float_data,
                      lookback=lookback,
                      delay=delay,
                      step=step,
                      min_index=0,
                      max_index=200000,
                      batch_size=batch_size,
                      shuffle=True)

val_gen = generator(float_data,
                    lookback=lookback,
                    delay=delay,
```

```
                      step=step,
                      min_index=200000,
                      max_index=300000,
                      batch_size=batch_size)

test_gen = generator(float_data,
                     lookback=lookback,
                     delay=delay,
                     step=step,
                     min_index=300000,
                     max_index=None,
                     batch_size=batch_size)

#  全結合モデルの訓練と評価
train_steps = ((200000 - 0 - lookback) // batch_size) + 1

val_steps = ((300000 - 200000 - lookback) // batch_size) + 1

test_steps = ((len(float_data) - delay - 300000 - lookback) // batch_size) + 1

model = Sequential()
model.add(GRU(64, dropout=0.02, recurrent_dropout=0.02, input_shape=(None, float_data.shape[-1])))
model.add(Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')

filepath = 'weights_min.hdf5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, ¥
    save_best_only=True, mode='auto')

history = model.fit_generator(train_gen,
                              steps_per_epoch=train_steps,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps,
                              callbacks=[checkpoint])

#  結果をプロットと保存
```

```
# 結果をプロットと保存
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(loss, label='Training loss')
plt.plot(val_loss, label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()

model.load_weights('weights_min.hdf5')
score = model.evaluate_generator(test_gen, steps=test_steps, verbose=1)
print("loss=", score)

model.save("RNN_model.h5")
```

```
Using TensorFlow backend.
['Date Time',
 'p (mbar)',
 'T (degC)',
 'Tpot (K)',
 'Tdew (degC)',
 'rh (%)',
 'VPmax (mbar)',
 'VPact (mbar)',
 'VPdef (mbar)',
 'sh (g/kg)',
 'H2OC (mmol/mol)',
 'rho (g/m**3)',
 'wv (m/s)',
 'max. wv (m/s)',
 'wd (deg)']
(420551, 14)
Epoch 1/40
776/776 [==============================] - 459s 592ms/step - loss: 0.2957 - val_loss: 0.2576

Epoch 00001: val_loss improved from inf to 0.25756, saving model to weights_min.hdf5
Epoch 2/40
776/776 [==============================] - 465s 599ms/step - loss: 0.2716 - val_loss: 0.2794

Epoch 00002: val_loss did not improve from 0.25756
Epoch 3/40
776/776 [==============================] - 464s 597ms/step - loss: 0.2547 - val_loss: 0.1492

Epoch 00003: val_loss improved from 0.25756 to 0.14920, saving model to weights_min.hdf5
Epoch 4/40
776/776 [==============================] - 458s 590ms/step - loss: 0.2394 - val_loss: 0.1396

Epoch 00004: val_loss improved from 0.14920 to 0.13964, saving model to weights_min.hdf5
Epoch 5/40
776/776 [==============================] - 458s 590ms/step - loss: 0.2233 - val_loss: 0.2508

Epoch 00005: val_loss did not improve from 0.13964
Epoch 6/40
776/776 [==============================] - 455s 587ms/step - loss: 0.2093 - val_loss: 0.1176

Epoch 00006: val_loss improved from 0.13964 to 0.11757, saving model to weights_min.hdf5
Epoch 7/40
```

```
Epoch 00038: val_loss did not improve from 0.06192
Epoch 39/40
776/776 [==============================] - 463s 596ms/step - loss: 0.1279 - val_loss: 0.0654

Epoch 00039: val_loss did not improve from 0.06192
Epoch 40/40
776/776 [==============================] - 459s 591ms/step - loss: 0.1284 - val_loss: 0.1797

Epoch 00040: val_loss did not improve from 0.06192
```




```
465/465 [==============================] - 57s 123ms/step
loss= 0.5328634977340698
```