프로그램 설명

① #1은 코랩에서는 필요 없습니다. 오류가 발생하면, 메모리 확장을 설정합니다([step37_01], [그림 2.9] 참조).

② [step25_02]의 Reuters 이진 벡터 분류를 간단한 1차원 CNN 모델로 구현합니다.

③ #2는 빈도수가 높은 top_words = 1000 단어를 훈련 데이터(x_train, y_train), 테스트 데이터(x_test, y_test)에 로드합니다. x_train, x_test를 단어 인덱스 위치에 1, 나머지는 0으로 이진 벡터로 인코딩합니다. x_train.shape = (8982, 1000), x_test.shape = (2246, 1000)입니다. y_train, y_test는 원-핫 인코딩하여 y_train.shape = (8982, 46), y_test.shape = (2246, 46)입니다.

④ #3은 Conv1D 층 입력을 위해 x_train의 모양을 (8982, 1000, 1)로 변경하고, x_test의 모양을 (2246, 1000, 1)로 변경합니다.

⑤ #4는 Conv1D, BatchNormalization, MaxPool1D, Dropout, Flatten, Dense 층 등으로 모델을 생성합니다. 출력층인 Dense 이전에 Flatten 층으로 평탄화합니다. 출력층은 activation = 'softmax' 활성화 함수, units = 46(분류 개수)의 뉴런을 갖는 Dense 층입니다.

⑤ #5는 손실함수 loss = 'categorical_crossentropy'로 학습합니다. 훈련 데이터의 정확도(train_acc)는 95.22%입니다. 테스트 데이터의 정확도(test_acc)는 74.76%입니다.

step39_03 1차원 CNN 모델: 스마트폰의 가속도계 센서 데이터 셋 분류 3903.py

```
'''
ref1(dataset): http://www.cis.fordham.edu/wisdm/dataset.php
ref2(paper): http://www.cis.fordham.edu/wisdm/includes/files/sensorKDD-2010.pdf
ref3: https://towardsdatascience.com/human-activity-recognition-har-tutorial-with-keras-and-core-ml-part-1-8c05e365dfa0
ref4: https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf
'''
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

#1
##gpus = tf.config.experimental.list_physical_devices('GPU')
##tf.config.experimental.set_memory_growth(gpus[0], True)

#2
def parse_end(s):
    try:
        return float(s[-1])
    except:
        return np.nan

def read_data(file_path):
# columns: 'user', 'activity', 'timestamp', 'x-accl', 'y-accl', 'z-accl';
    labels = {'Walking'    :0,
              'Jogging'    :1,
              'Upstairs'   :2,
              'Sitting'    :3,
              'Downstairs' :4,
              'Standing'   :5}
    data = np.loadtxt(file_path, delimiter = ",",
                      usecols = (0,1, 3, 4, 5),          # without timestamp
```

```
                converters={1: lambda name: labels[name.decode()],
                            5: parse_end})
    data = data[~np.isnan(data).any(axis = 1)]          # remove rows with np.nan
    return data

# Load data set containing all the data from csv
data = read_data("./DATA/WISDM_ar_v1.1/WISDM_ar_v1.1_raw.txt")
##print("user:", np.unique(data[:, 0]))                 # 36 users
##print("activity:", np.unique(data[:, 1]))             # 6 activity

#3: normalize x, y, z
mean = np.mean(data[:, 2:], axis = 0)
std  = np.std(data[:, 2:], axis = 0)
data[:,2:] = (data[:, 2:] - mean) / std
##data[:,2:] = (data[:, 2:]) / np.max(data[:, 2:], axis = 0) # [ -1, 1]
##print(np.mean(data[:, 2:], axis = 0))                 # [0, 0, 0]
##print(np.std(data[:, 2:], axis = 0))                  # [1, 1, 1]

# split data into x-train and x_test
x_train = data[data[:, 0] <= 28]                        # [28, 36]
x_test = data[data[:, 0] > 28]

#4: segment data and reshape (-1, TIME_PERIODS, 3)
TIME_PERIODS = 80                                       # length
STEP_DISTANCE = 40                                      # if STEP_DISTANCE = TIME_PERIODS, then no overlap
def data_segments(data):
    segments = []
    labels = []
    for i in range(0, len(data)-TIME_PERIODS, STEP_DISTANCE):
        X = data[i:i + TIME_PERIODS, 2:].tolist()       # x, y, z

        # label as the most activity in this segment
        values, counts = np.unique(data[i:i+TIME_PERIODS, 1], return_counts = True)
        label = values[np.argmax(counts)]               # from scipy import stats: stats.mode()

        segments.append(X)
        labels.append(label)

    # reshape (-1, TIME_PERIODS, 3)
    segments = np.array(segments, dtype = np.float32).reshape(-1, TIME_PERIODS, 3)
    labels = np.asarray(labels)
    return segments, labels

x_train, y_train = data_segments(x_train)
x_test, y_test = data_segments(x_test)
print("x_train.shape=", x_train.shape)
print("x_test.shape=",  x_test.shape)

# one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
##print("y_train=", y_train)
##print("y_test=", y_test)

#5: build a model with 1D CNN
model = tf.keras.Sequential()
```

```
model.add(tf.keras.layers.Input(shape = (TIME_PERIODS,3)))       # shape = (80, 3)
model.add(tf.keras.layers.Conv1D(filters = 100,
                                 kernel_size = 11, activation = 'relu'))
model.add(tf.keras.layers.MaxPool1D())
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv1D(filters = 10, kernel_size = 5, activation = 'relu'))
model.add(tf.keras.layers.MaxPool1D())
model.add(tf.keras.layers.Dropout( rate = 0.5))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units = 6, activation = 'softmax'))
model.summary()

#6: train and evaluate the model
opt = tf.keras.optimizers.RMSprop(learning_rate = 0.01)
model.compile(optimizer = opt, loss = 'categorical_crossentropy', metrics = ['accuracy'])
ret = model.fit(x_train, y_train, epochs = 100, batch_size = 400,
                validation_data = (x_test, y_test), verbose = 2)      # validation_split=0.2
train_loss, train_acc = model.evaluate(x_train, y_train, verbose = 2)
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose = 2)

#7: plot accuracy and loss
plt.title("Accuracy")
plt.plot(ret.history['accuracy'], "b-", label = "train accuracy")
plt.plot(ret.history['val_accuracy'], "r-", label = "test accuracy")
plt.plot(ret.history['loss'], "g-", label = "train loss")
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(loc = "best")
plt.show()

#8: draw sample activity
activity = ('Walking','Jogging','Upstairs', 'Sitting','Downstairs','Standing')
train_label = np.argmax(y_train, axis = 1)
plot_data = []
n = 1
for i in range(6):
    plot_data.append(np.where(train_label == i)[0][n]) # n-th data

fig, ax = plt.subplots(6, sharex = True, sharey = True)
fig.tight_layout()
for i in range(6):
    k = plot_data[i]
    ax[i].plot(x_train[k], label = activity[i])
    ax[i].set_title(activity[i])
plt.show()
```

▼ 실행 결과

```
x_train.shape= (20868, 80, 3)
x_test.shape= (6584, 80, 3)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 70, 100)           3400
```

```
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 35, 100)          0
_________________________________________________________________
batch_normalization (BatchNo (None, 35, 100)          400
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 31, 10)           5010
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 15, 10)           0
_________________________________________________________________
dropout (Dropout)            (None, 15, 10)           0
_________________________________________________________________
flatten (Flatten)            (None, 150)              0
_________________________________________________________________
dense (Dense)                (None, 6)                906
=================================================================
Total params: 9,716
Trainable params: 9,516
Non-trainable params: 200
_________________________________________________________________
653/653 - 1s - loss: 0.0042 - accuracy: 0.9996
206/206 - 0s - loss: 1.4047 - accuracy: 0.8507
```

프로그램 설명

① ref1, ref2의 안드로이드 스마트폰의 가속도계 센서 데이터 셋 WISDM을 이용한 6가지 사람의 활동(Walking, Jogging, Upstairs, Sitting, Downstairs, Standing) 분류를 ref3과 ref4를 참고하여 구현합니다. 데이터는 20Hz로 샘플링한 데이터입니다. 각 데이터 간격은 1 / 20 = 0.05초입니다. TIME_PERIODS = 80은 80 * 0.05 = 4초 입니다. 예제에서는 데이터를 4초 간격의 3-채널(x, y, z)로 분할하여 (batch, steps, channels) = (batch, 80, 3)의 모양의 다채널 데이터를 1차원 CNN 모델에 입력합니다.

② #1은 코랩에서는 필요 없습니다. 필자의 컴퓨터는 GPU 메모리 확장을 설정하지 않아도 오류가 발생하지 않습니다. 오류가 발생하면, #1의 주석을 해제하고 메모리 확장을 설정합니다([step37_01], [그림 2.9] 참조).

③ #2에서 data = read_data("./DATA/WISDM_ar_v1.1/WISDM_ar_v1.1_raw.txt")는 ref1에서 다운로드한 데이터 파일로부터 넘파이 배열 data에 데이터를 읽습니다. 데이터 파일의 각 라인의 데이터는 콤마로 구분되고, 마지막은 세미콜론이 붙어 있습니다. 일부 한 줄에 2개의 데이터가 입력된 경우는 메모장에서 줄을 변경합니다.

```
[user], [activity], [timestamp], [x-accel], [y-accel], [z-accel];
```

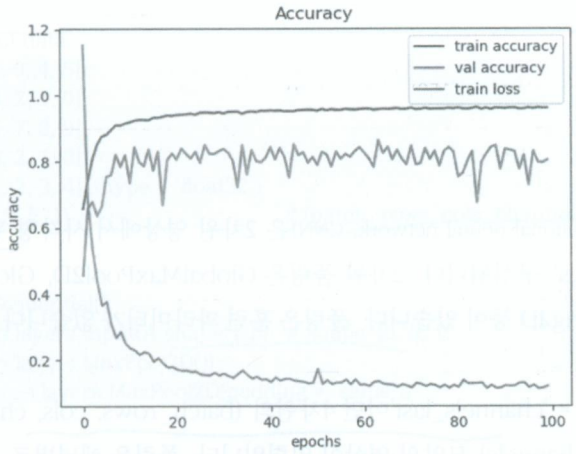
④ read_data() 함수는 file_path의 파일을 np.loadtxt()로 읽어 넘파이 배열 data에 읽어 반환합니다. 구분자는 delimiter = ","이고, usecols = (0, 1, 3, 4, 5)만을 사용하여 2열의 timestamp는 읽지 않습니다. converters를 사용하여 1열의 활동(activity) 문자열은 labels의 숫자로 변환합니다. 5열은 parse_end 함수를 사용하여 float(s[-1])로 세미콜론을 제거하고, 제거할 수 없으면 np.nan으로 변환합니다. data = data[~np.isnan(data).any(axis = 1)]은 np.nan이 있는 행의 데이터를 삭제합니다.

⑤ #3은 data의 센서 데이터([x, y, z]) 부분 data[:, 2:]를 axis = 0 방향으로 평균 mean, 표준편차 std를 계산하여 정규화합니다. mean.shape = (3,), std.shape = (3,)입니다. user는 1에서 36까지입니다. 1에서 28까지를 x_train 데이터, 29에서 36까지를 x_test 데이터로 분할합니다.

⑥ #4에서 data_segments() 함수는 STEP_DISTANCE 만큼씩 움직이며 TIME_PERIODS 길이의 센서 데이터([x, y, z])를 segments 리스트에 추가하고, 레이블은 가장 많은 활동을 계산하여 labels 리스트에 추가합니다. segments를 넘파이 배열로 변경하고, CNN 입력을 위해 3차원 배열 (-1, TIME_PERIODS, 3) 모양으로 변경합니다. labels를 넘파이 배열로 변경하고, segments와 labels를 반환합니다. data_segments() 함수로 훈련 데이터 (x_train, y_train)와 테스트 데이터(x_test, y_test)를 생성하고, y_train, y_test는 원-핫 인코딩합니다.

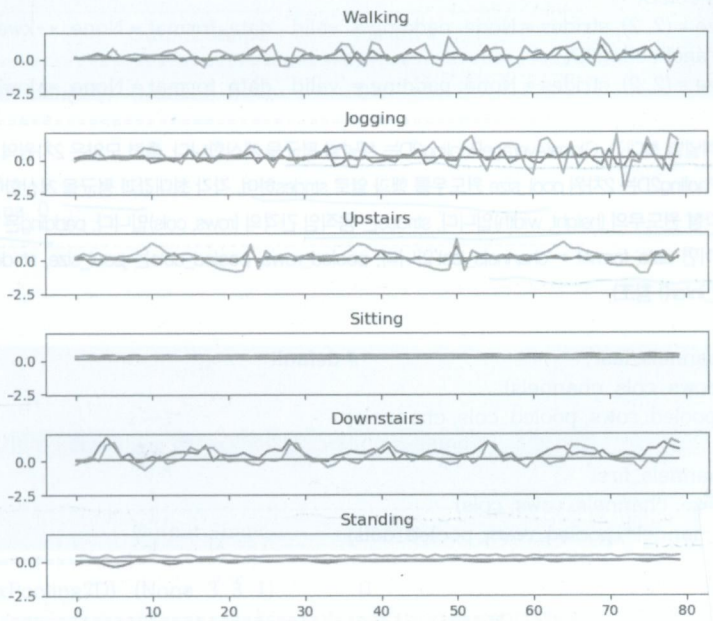
⑦ #5는 Input 층으로 shape = (80, 3) 모양의 입력을 받고, Conv1D, BatchNormalization, MaxPool1D, Dropout, Flatten, Dense 층 등으로 모델을 생성합니다([그림 39.1] 참조). Flatten 층으로 평탄화하고, activation = 'softmax' 활성화 함수, units = 6의 뉴런을 갖는 완전 연결 Dense 층을 추가합니다.

⑧ #6은 손실함수 loss = 'categorical_crossentropy'를 RMSprop(learning_rate = 0.001)로 epochs = 1000 반복 학습합니다. 훈련 데이터의 정확도(train_acc)는 99.96%입니다. 테스트 데이터의 정확도(test_acc)는 85.07%입니다. [그림 39.2]는 #6에 의한 WISDM 데이터의 정확도와 손실 그래프입니다.



▲ **그림 39.2** WISDM 데이터의 정확도와 손실

⑨ #8은 WISDM 센서 데이터의 일부를 그래프로 그립니다. train_label에 정수 레이블을 계산하고, np.where(train_label == i)[0][n]로 각 레이블에서 n번째 데이터를 plot_data 리스트에 추가합니다. ax[i]에 k = plot_data[i], x_train[k]의 활동을 그래프로 표시합니다. [그림 39.3]은 n = 1의 TIME_PERIODS=80 길이의 활동별 센서 데이터 그래프입니다. 'Walking'과 'Jogging'의 움직임이 크고, 'Sitting'과 'Standing'의 움직임이 작습니다.



▲ **그림 39.3** WISDM 센서 데이터 그래프