Laboratory # 7

Hand Gesture Recognition Using Leap Motion and Deep Learning

1 Introduction

This Lab uses a USB-connected sensor called Leap Motion. It detects a moving object in close proximity (1–20cm) by using the time-of-flight principle; in this sensor, near-infrared beams are sent and received to detect the distance to the object's surface, and, therefore, recover the object's shape and position (same as a Microsoft's Kinect camera). The purpose of this lab exercise is to apply the concepts of machine learning to **recognize gestures** performed by a hand, such as a circle and a swipe in the air, based on the coordinates of the hands' joints identified by the Leap Motion device.

Note that in the Lab project, you can do a variation, and recognize the identity of a subject performing one selected gesture, rather than identifying the type of gesture.

There are two options to get data for this Lab: you can collect your data using the Leap Motion provided in the lab, or use the data pre-recorded and available on D2L. To collect the data from the Leap Motion, you will need to set up an environment based on Python 2 version, explained in Section 1.2. If you prefer to use the data provided, you can skip this step.

The classification will be performed using Python 3 and Keras library[1]. The classifier used is the *Long Short-term Memory* (LSTM), a deep learning model for time-series analysis. Section 1.3 will explain how to create a new Python 3 environment for this purpose only.

1.1 Installation of Leap Motion driver

The UltraLeap (the company that developed the Leap Motion) created SDKs for various programming languages. Howe but each SDK (Software Development Kit) is provided by its version of the framework library[2]. The most up-to-date version for Python is *Leap Motion Orion 3.2.1*. By default, this version works only with Python 2.7

NOTE: you might be able to generate and compile your version of the "LeapPython" source code with SWIG interface file (it is a tool that connects programs written in C and $C++$ to Python) that is provided in the SDK. This compiling process is out of the scope of this Lab Project.

To set up the Leap Motion driver on Windows to collect data, you will need to follow the steps below:

1. Download the .zip file the *Leap Motion Orion 3.2.1* SDK from D2L.
2. The zip-file you downloaded contains the folder *LeapSDK*. You need to copy the following files to the same folder of your project's code (or you can find this file in D2L: LeapMotion-PythonFiles.zip):

```
LeapSDK/lib/Leap.py
LeapSDK/lib/x64/LeapPython.pyd
LeapSDK/lib/x64/Leap.dll
LeapSDK/lib/x64/Leap.pyc
```

[1] https://keras.io/
[2] https://developer-archive.leapmotion.com/documentation/python/index.html

In Section 1.2, we explain how to install Python 2 using your current Anaconda installation. After that, you will be able to collect the data.

1.2 Creating Python 2 environment

Anaconda offers the possibility of creating "environments"[3]. An environment plays the role of a container where you can install a different Python version with another set of libraries. By default, the Anaconda installation creates the "base" environment; it is shown in the first line when you open the *Anaconda Powershell Prompt*. You can create your environments regardless of the Python version.

For the data collection, you need to create a Python 2 environment. With the *Anaconda Powershell Prompt* opened, follow the instructions:

1. Create a Python 2 environment named "py2":

```
conda create —n py2 python=2.7
```

2. Activate the created environment:

```
conda activate py2
```

3. In this new environment (`py2`), install the Spyder editor along with some other libraries:

```
conda install spyder numpy pandas notebook
```

After these steps, start your Anaconda's entry on the Start menu. You shall see a new Spyder and Jupyter Notebook entries called "Spyder (py2)" and "Jupyter Notebook (py2)", respectively. This is the Spyder version that you need to run when Python 2 is required.

1.3 Installation of the Keras library

The machine learning technique used in this lab is a deep neural network with memory, called Long-Short-Term-Memory (LSTM). This is needed to recognize gesture performed in time. The LSTM network uses the Keras library (Python 3), which is not included in your default Anaconda installation. For the Keras installation, we recommend creating a new environment to avoid the conflict with other libraries, as explained below.

Open Anaconda Powershell Prompt and follow the steps below:

1. Create a new environment called `leapmotionNN`:

```
conda create —n leapmotionNN python=3.9
```

2. Activate the new environment:

```
conda activate leapmotionNN
```

[3] https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html?highlight=environment

3. Install the necessary libraries, including Keras and Tensorflow:

```
conda install —c conda—forge keras tensorflow jupyter opencv pandas scikit—learn
matplotlib notebook
```

After running these commands, a new Jupyter Notebook entry will appear on your start menu; it corresponds to the new environment: `Jupyter Notebook (leapmotionNN)`. You can run the LSTM in it.

2 The laboratory procedure

2.1 Data collection

With the Leap Motion already connected to your computer, open Spyder (for Python 2), then open the sample code `LeapMotion_Recorder.py`. When you run it, you'll be able to collect the data captured by the Leap Motion. When your finish performing a gesture, press `[ENTER]` in the Python terminal, to save your data in the file `data1.csv`.

Repeat this process 10 – 20 times to record several samples of the same gesture. For each gesture you will have a `.csv` file.

3 Procedure

All the steps described below are required to be done using Python 3. Note that Python 2 was used *only* to collect the data.

3.1 Data preparation and classification with LSTM

To train a classifier to distinguish between two gestures, you will need to collect data from the same subject performing both sets of gestures. Consider one gesture to draw a circle in the air, another one as drawing a line (a swipe). Once you have defined the gestures you want to collect, perform the same gesture 10 to 20 times.

All the data must to be saved in `.csv` files. We recommend that you saved one file per gesture. For example, the piece of code below shows the names of 20 files (10 drawing of a circle and 10 swipe movements) stored in two variables to be used in the training step:

```
circle = ['circle1.csv', 'circle2.csv', 'circle3.csv', 'circle4.csv', 'circle5.csv',
          'circle6.csv', 'circle7.csv', 'circle8.csv', 'circle9.csv', 'circle10.csv']

swipe = ['swipe1.csv', 'swipe2.csv', 'swipe3.csv', 'swipe4.csv', 'swipe5.csv',
         'swipe6.csv', 'swipe7.csv', 'swipe8.csv', 'swipe9.csv', 'swipe10.csv']
```

3.1.1 Preparing the data

Put all your `.csv` files in the folder `data/`. Instead of manually typing the name for each file, use the Python resource called *list comprehension* to recover all the file names:

```
# the directory where your data is
mypath = './data'

# creating a list with all the filenames
datafiles = [f for f in listdir('data') if isfile(join(mypath, f))]
```

All the data in `.csv` files are stored as Pandas' DataFrames[4] for easy recovery when necessary.

Before defining a DataFrame, we need to define the columns' names; those names correspond to the features captured by the Leap Motion. Check Leap Motion's web page to see which features can be recorded[5]. The following code defines all the columns' names necessary:

```
columns = ['handPalmPosition_X','handPalmPosition_Y','handPalmPosition_Z',
           'pitch', 'roll', 'yaw', 'GestureTypeCircle', 'GestureTypeSwipe',
           'wristPosition_X', 'wristPosition_Y','wristPosition_Z',
           'elbowPosition_X', 'elbowPosition_Y', 'elbowPosition_Z']

# finges used to perform the gesture
finger_names = ['Thumb', 'Index', 'Middle', 'Ring', 'Pinky']
# finges' bones identified by Leap Motion
bone_names = ['Metacarpal', 'Proximal', 'Intermediate', 'Distal']

for finger in finger_names:
    columns.append(finger + 'Length')
    columns.append(finger + 'Width')

# for each finger several features are collected
for finger in finger_names:
    for bone in bone_names:
        columns.append(finger + bone + 'Start_X')
        columns.append(finger + bone + 'Start_Y')
        columns.append(finger + bone + 'Start_Z')
        columns.append(finger + bone + 'End_X')
        columns.append(finger + bone + 'End_Y')
        columns.append(finger + bone + 'End_Z')
        columns.append(finger + bone + 'Direction_X')
        columns.append(finger + bone + 'Direction_Y')
        columns.append(finger + bone + 'Direction_Z')
```

Now load the data. The file names, previously defined when collecting the data, indicate which gesture is described in that specific `.csv` file. This gesture corresponds to the class we wish to classify.

3.1.2 Preprocessing for pre-recorded data

The code below has all the features as a list stored in the variable `x` (where each element is a DataFrame), and the corresponding class is contained in the variable `y`:

[4] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html
[5] https://developer-archive.leapmotion.com/documentation/python/index.html

```
# Features
x = []
# Labels
y = []

for sample in datafiles:
    relative_path = 'data\\' + sample
    tmp = pd.read_csv(relative_path, usecols=columns)

    # Normalize the sample size: LSTM requires all inputs to be the same size!
    print('{}\nsize raw = {}'.format(relative_path,tmp.shape))
    while tmp.shape[0] < NUMBER_TIMESTEPS:
        tmp = tmp.append(pd.Series(0, index=tmp.columns), ignore_index=True)

    if tmp.shape[0] > NUMBER_TIMESTEPS:
        tmp = tmp.head(100)
    print('size normalized = ',tmp.shape)

    tmp_x = tmp[[column for column in list(tmp.columns)
                 if column != 'GestureTypeCircle'
                 and column != 'GestureTypeSwipe']]
    tmp_y = tmp[['GestureTypeCircle', 'GestureTypeSwipe']]

    x.append(tmp_x)
    y.append(tmp_y)
```

3.1.3 Preprocessing for your own data

If you are working with your own dataset, first, you need to parse the data. The LeapMotion records the values of the coordinates X, Y and Z for each data record in one cell, in this order: (X, Y, Z). For our work, we need to divide this data into three separate columns.

```
def split_and_expand(df, column_name_prefix):
    columns = [f'{column_name_prefix}_{i}' for i in ['X', 'Y', 'Z']]
    df[columns] = df[column_name_prefix].str[1:-1].str.split(',', expand=True)
```

This operation shall be completed for all data points.

You also have to specify the ground truth label for each file. To do this, consider the name of the file. If the files start with "c" (circle), then GestureTypeCircle' = 1 and GestureTypeSwipe' = 0. If the file name starts with "s" (swipe), then GestureTypeSwipe' = 1 and GestureTypeCircle' = 0

```
for c in circle:
    relative_path = 'data\\' + c
    print(relative_path)
    tmp = pd.read_csv(relative_path)
    tmp['GestureTypeCircle'] = 1
    tmp['GestureTypeSwipe'] = 0
```

```
    tmp.to_csv(path_or_buf=relative_path, index=False)

for s in swipe:
    relative_path = 'data\\' + s
    print(relative_path)
    tmp = pd.read_csv(relative_path)
    tmp['GestureTypeCircle'] = 0
    tmp['GestureTypeSwipe'] = 1
    tmp.to_csv(path_or_buf=relative_path, index=False)
```

If you use your own dataset, you need to perform a data cleaning step: remove empty records and ensure that there are no artifacts (such as other than hands objects) during the recording.

Finally, we repeat the step (from the pre-recorded part) of all the features as a list stored in the input variable `x` (where each element of this vector is a DataFrame), and the corresponding class is contained in the output variable `y`.

3.1.4 Dividing the data into the training and testing sets

The last step when preparing your data is to divide it into two sets, one for training and one for testing. This is done using Scikit-Learn's function `train_test_split(...)` . Besides the input and output variables `x` and `y`, we have to define the proportion of the test set. In the example below, the test size corresponds to 30% of all data:

```
# Set a percentage of test set fraction
test_size = 0.3 # 30%

# Divide data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=test_size,
                                                    shuffle=True)

# Turn list(DataFrame) into numpy.ndarray
x_train = np.array(x_train)
x_test = np.array(x_test)
```

At this point, we have our data split into four variables: `x_train`, `x_test`, `y_train` and `y_test`. The variables `x_...` are the features, while the `y_...` are the corresponding labels. These input and output data are arranged into arrays (2D or multi-dimensional matrices).

3.2 Classification

To perform the classification of gestures, use the file `Lab07-LeapMotion-GestureClassifier.ipynb` as a template. Since the gesture is represented as a time-series, the hand joint coordinates change over time. Thus, the classifier should deal with such data. In this example, we will use the type of a neural network called *Long Short-term Memory* (LSTM) available in the Keras library.

The chosen LSTM has two input layers with 256 units each, three hidden fully connected layers with 512, 256, and 512 units, respectively, and a classification layer with 2 neurons (2 outputs, too).

3.2.1 Creating an LSTM in Keras

After arranging the data into arrays, two pieces of information are essential: the size of the input data and the size of the network output. Note that the neural networks usually use a one-hot encoding for the classes (a code that has one '1' and the rest are 0's). For example, the network with 2 units on the output layer would have Class 1 encoded as '10' 9'1' on the first unit's output, and '0'on the second unit's output), and Class 2 encoded as '01'. The number of outputs can be derived based on the other variables (the initially known number of classes), or defined manually as shown below:

```
# the number of features (from the data)
NUMBER_FEATURES = 202

# the number of classes/gestures
NUMBER_OUTPUTS = 2
```

The `Sequencial()` class will be used to create an empty model, which is a neural network architecture with certain parameters. Next, the LSTM and network layers (`Dense`) are added, followed by defining the *Rectified Linear Unit - ReLu* activation function.

The model is populated when you call the function `compile(...)`. This is where you also set the optimizer, and the metric to be evaluated during the training, among various other parameters[6]. The code below shows a function defined to facilitate the model creation. Note that number of input layers, LSTM layers and output layers, and the number of units in each as defined below:

```
def build_model():
    model = models.Sequential()

    # the input layer expects:
    # 1 or more samples, NUMBER_TIMESTEPS time steps and NUMBER_FEATURES features.

    # 1st LSTM layer with 256 units
    model.add(layers.LSTM(256, return_sequences=True,
                          input_shape=(NUMBER_TIMESTEPS, NUMBER_FEATURES)))
    # 2nd LSTM layer with 256 units
    model.add(layers.LSTM(256,
                          input_shape=(NUMBER_TIMESTEPS, NUMBER_FEATURES)))

    # Hidden fully connected layers of the neural network
    # 512 neurons
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dropout(0.5))
    # 256 neurons
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dropout(0.5))
    # 512 neurons
    model.add(layers.Dense(512, activation='relu'))

    # Classification layer of the neural network
```

[6] https://keras.io/api/models/model_training_apis/

```
    model.add(layers.Dense(NUMBER_OUTPUTS, activation='softmax'))

    opt = Adam(learning_rate=0.002)
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])

    # this shows the network structure
    model.summary()

    return model
```

To train the model, the `fit(...)` function is called along with the training set, the respective labels, the batch size (the number of samples used in each training step) and the number of epochs. Below, we train the model using 10 epochs, and the batch size equals to the training set size:

```
model.fit(X_train, y_train, epochs=10, batch_size=len_train)
```

When the model is trained, you can use the test set to evaluate the generalization performance. Function `evaluate(...)` is used to do so, with the arguments corresponding to the test set (samples and labels):

```
test_loss, test_acc = model.evaluate(X_test, y_test)
```

To recover the predicted class of each sample from the test set, you can use the `predict(...)` function:

```
# testing the classifier trained with the test set
y_pred = model.predict(X_test)

matches = (y_pred == y_test)
print('Total of matches: %d' % (matches.sum()))

match_rate = matches.sum() / float(len(matches))
print('Match rate: %.2f' % (match_rate))
```

Evaluation of the classifier after the training is an important step in your machine learning pipeline. To do so, use the code to create a confusion matrix and the classifier evaluation approach similar to the one used in Lab # 6.

4 Lab Report

Your report in the form of a Jupyter Notebook/Python (file extension `.ipynb`) shall include the following graded components (10 marks total):

- Introduction (a paragraph about the purpose of the lab, 0.5 marks).
- Description of the result on each exercise with illustrations/graphs and analysis of the results (9 marks are distributed as shown in the Exercise section).
- Conclusion (a paragraph on what is the main take-out of the lab, 0.5 marks).

Save your Notebook using menu "Download As" as `.ipynb`, and submit to D2L dropbox for Lab 7 by the following Thursday.

5 Lab Exercise in Jupyter Notebook with Python

For the following exercises, use the sample data available on D2L, or your own data.

- **Exercise 1** (3 marks): Consider 60% of samples per gesture for training and 40% for testing. Perform the classifier evaluation for this case. Next, use 80% of samples per gesture for training and 20% for testing. Perform the classifier evaluation and compare it against the first case (60 and 40%).
- **Exercise 2** (3 marks): Consider the parameters of the input and hidden LSTM layers. Consider a smaller number of units in the LSTM layers (for example, 128 units instead of 256 in the input layer, and 256, 128 and 256 units in the three hidden layers). Perform the classifier evaluation and compare the results for the original LSTM and the one with the reduced number of units.
- **Exercise 3** (3 marks): Consider changing the dropout probability for the LSTM trained on 60% of samples per gesture and tested on 40% of samples. In the code, the dropout value is set to 0.5. Change it to another value, for example, 0.6, 0.7 or 0.8. Perform the classifier evaluation and compare its performance against the network with the original dropout value of 0.5.

6 Acknowledgments

We would like to thank the TAs, Olha Shaposhnyk and Illia Yankovyi, for developing and testing the code used in this laboratory.

February 25, 2024