COMP704 – Machine Learning **Worksheet WK 2**

Data collection and Processing in Python

Introduction
In the lecture this week, we looked at machine learning and, in particular, data acquisition and processing as a step prior to training and testing ML algorithms. The goal of this workshop is to examine how we can instrument a simple Python application in order to capture data that could be used for machine learning and how we can start processing that data into collections and formats that can be used for training and testing ML-algorithms.

Breakout
This is a simple Pygame application that I found on the internet. We will use this as a test harness for data collection. The first step is to get the game into Pycharm and running. This will require the pygame module to be present and can be added to the existing breakout project as a module through the python settings.
It's likely that your install of Pycharm will not be set up with a valid interpreter, this should be in c:/program files x86/Python (version 3.something). **Please make sure you use Python 3.x rather than Python 2.7.**

If Pycharm has not been installed, it can be installed to your local drive. Likewise, if Python 3.x has not been installed, it can also be installed to your local drive. Don't attempt to install Pycharm or Python 3.x to the 'C' drive as this will fail.

File manipulation with Python
The project example-code/fileio contains code samples to read and write data from and to a file, as raw text, csv (comma separated variable) and Excel (xlsx) formats. All of these formats are useful, but often in different situations.

As a starting point to machine learning, we need to think about what data we need to capture from the game, how we should go about capturing it and what format it should take. From the lecture, we can see that this is often a highly iterative process, particularly when we are not sure precisely what data (and what features, in ML parlance) we are looking to capture.

Raw text can be a useful format to use when we can to create easily human-readable labels in data to see what is going on with the app that we are instrumenting for analysis. The downside with text format is that it often difficult to read and parse data back to a machine-readable form.

Raw-text file manipulation
To get a handle on this problem, use the raw text code from the sample to create a file when the player starts the game and capture the player's paddle position each frame. At the end of the game, close the file that you have saved the data in and write another Python program that will open this file and read all the data as pairs of numbers.

CSV file manipulation
Re-write the original program using the csv library to write data to a csv file, then write another application that can load and print out the csv data.

CSV files are a good data format to work with as scikit-learn uses them for training, however they often aren't that useful for us if we want to manage data from multiple runs.

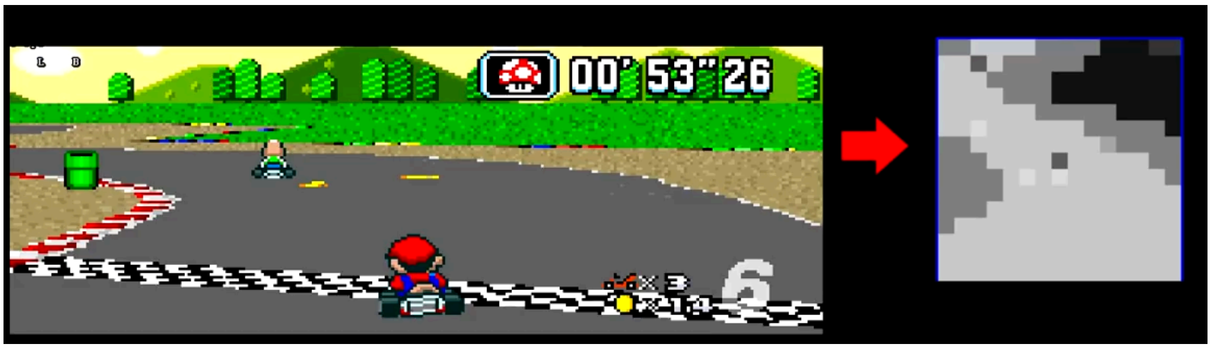
OpenPyXL file manipulation
Openpyxl is a library for working with xlsx spreadsheets with a key advantage over csv in that a worksbook can contain multiple worksheets (csv files). This makes xlsx very useful for storing multi-session data.
To demonstrate this, re-write the original program using the openpyxl library and store each game session (from when the player starts until 'game over') in a separate worksheet within the same workbook. When the player quits, save all the data to an xlsx file, then load the data.

Using the HTTP stack
The methods we have looked at so far are very good for storing a single user's data, but what if you want to capture from multiple users?
To do this, we can leverage the HTTP client/server architecture and implement a server that will store multiple sets of data, either in xlsx files or using a sqlite database (as it's part of Python).

The 2019-20-COMP704-02-fullstak-http-worksheet gives an introduction to fullstack Python development with HTTP, json and sqlite that you can use as a starting point to collect data from multiple players and multiple sessions.

Visualising / Processing data
As we saw from the Marioflow video, source game data is rarely in a perfect format for ML-algorithms, so processing needs to be undertaken. Generally, data is taken out of a game before it is processed to keep the source data intact. That way, if you need to re-process data, the source data has not been lost or converted.

With Marioflow, game video is captured and then converted to a smaller format of encoded data, as below:



For our crappy breakout, we need to think about what data we need to capture for training and how to present it to whatever ML-algorithm we decide to use. A good starting point may be to take a similar approach to Marioflow, in that the game frame is the input (along with the ball position) whilst the player’s bat is the output. As part of this conversion / compression process it is worth using the Pygame library to create a viewer of what the compressed data looks like.

In keeping with the manta of ‘not converting source data’, we can write a frame extractor that will capture the features of the breakout game and send them to our HTTP server every few game updates. This can be done by sending the brick, ball and player positions to the server and writing a compressor that will work through that data from the server. Marioflow uses a 200-node input, suggesting a 20x10 input matrix, we can look to use something similar. To determine if the compression makes sense, create a pygame application that can view this format.