**Introduction**

In 1983 Nintendo released its “Nintendo Entertainment System” (NES) games console (Wikipedia, 2019). A large variety of games were produced for the NES, and many a childhood has been spent playing games on the console.

The console is connected to a TV to display images, and the player controls the game using a handheld controller known as a “joypad”. The joypad has directional left/right/up/down type button (where two adjacent directions buttons can be pressed at once), and also an A and a B button. There is also a “start” and a “select” button, but these are typically not used in game play.

Even though the NES console itself has not been on sale for many years, it is still possible to play your favourite NES games today using one of a variety of software emulators. A NES software emulator accurately simulates the original NES console, with the original TV display rendered on a computer screen, allowing a player to play the games.

In the 36 years since the NES console was released, Artificial Intelligence (AI) has come a long way, particularly in the field of object recognition and deep neural networks. This project aims to build a stand-alone program to act as an artificial intelligence player for the well-known game Super Mario Brothers, by “seeing” (detecting) objects in the video stream produced by the emulator, and then responding by issuing commands to the emulator. The commands will be produced by a Neural Network that will take as input the current state of the world in the game.

**Background Research**

Talk about OpenAI, tensorflow and perhaps emulators in general.

Maybe mention the OpenAI projects here (and the Mario competition one), and why I want to make my own.

**Problem description**

In order to produce a full, standalone, AI player setup, several pieces of software will have to be built, training/test data generated, and Neural Networks trained.

The following high-level tasks have been identified:

* An existing open source NES emulator will be modified to send a video stream over a network connection, and also accept inputs over a network connection
* The TensorFlow (Google/Tensorflow, 2019) libraries will be leveraged for enemy object detection. This requires a training set to be generated for the object detection, The training set will be created by a program that will be developed as part of this project
* Training of an existing TensorFlow object detection model
* The AI player software itself, which will use TensorFlow’s object detection to build a view of the world. This view will be fed into a model that will produce the appropriate key combination to press to play the game. This software will also need to deal with the interaction with the modified emulator and displaying images to the screen.

The emulator and player will be completely separate, and they will share no state. All communication will happen over a network channel, in order to make the AI player appear as much as possible as a real human.

**Problem description - the emulator**

The job of the NES emulator is to run NES games, which is achieved by emulating the different hardware components that made up the original NES in software. There are a number of different Open Source NES emulators available, written in a range of programming languages.

The main requirements for the emulator used in this project are:

* Ease of modification, and written in a language that the author of this project feels comfortable using.
* Accurate emulation to ensure that the game runs as well as possible.

A player typically interacts with the NES emulator using the keyboard on the computer. The keyboard input from the user is translated to NES style status bits in the emulator (link to an example), and the game running in the emulator then reads the status and acts accordingly.

In the NES architecture, a game will write pixel values to an internal frame-buffer (link to the a frame buffer code example), which is then drawn on the TV. The frame-buffer can be thought of as an in-memory array of RGB values (Red, Green Blue colour values) for each of the pixels that form the screen. A physical NES console will periodically draw these pixels on the TV screen, but in an emulator the frame-buffer is simply drawn to the computer screen.

As the goal of this project is to create a software player, an interface for interaction has to be defined. As the AI player cannot directly press buttons, nor “see” the screen, the emulator will be modified to accept input over a network connection, and it will also send the screen pixel values to the AI player over the same network connection.

**Problem description – the training set**

The pre-trained TensorFlow object detection models cannot distinguish objects in NES games, as they have not been trained on them. The training set for object detection models typically consist of a large number of images, with ground truth boxes defined in XML, created by hand (insert link to some tensorflow training documentation).

As there are only a limited number of enemies and objects to detect in Super Mario Brothers, a stand-alone Python program to generate training data will be developed, thus avoiding time-intensive manual training set creation.

The basic algorithm for the proposed training set generator is:

* From a range of background images, select one
* Generate a new image by pasting a copy of an enemy or other object onto the selected image into a random position
* Create the ground truth XML file with the coordinates used in the previous step
* Save the newly created image.

**Problem description – Training the TensorFlow object detection model**

TensorFlow uses deep neural networks for object detection (insert link to TF Doc). There are a variety of object detection models for TensorFlow already developed (insert link to zoo), and pre-trained on existing data sets. They broadly fall into Regional Convolutional Neural Networks, or Single Shot Detection networks.

TensorFlow provide average inference times for the models in the zoo, and it appears that the SSD based models are mostly quicker for detection. This project will utilize one of them, as the goal is to be able to run the player on a low spec laptop without a GPU.

Training a TensorFlow model can take several days of GPU time, however as the project aims to only detect objects within a game frame, it is expected that the training phase will be somewhat simpler.

The standard configurations for the pre-trained models have configuration parameters that make TensorFlow modify the input data in ways that are unhelpful to the specific aim of detecting sprites in Super Mario Brothers.

A normal object detection model needs to be able to find instances of the same object held at different angles, where part of the object has been cropped, and of different sizes, but this is not true for the images in this project – the sprites are always the same size, and they do not appear rotated. Hence, we can likely gain accuracy and training speed by disabling these features.

The project will use a sample configuration file for an SSD Mobilenet V2 model as a starting point, for example the one used to train the SSD Mobilenet V2 model on the COCO dataset (Google, 2018). Figure 1 shows an example of parameters that will be removed or altered in order to speed up training.

Figure - superfluous training parameters (for a sprite based game)

anchor\_generator {

ssd\_anchor\_generator {

<… other parameters …>

aspect\_ratios: 1.0

aspect\_ratios: 2.0 # aspect ratios not equal to 1 are

aspect\_ratios: 0.5 # not required

aspect\_ratios: 3.0

aspect\_ratios: 0.3333

}

}

train\_config: {

<… other parameters …>

data\_augmentation\_options {

random\_horizontal\_flip { } # No need for flipping sprites

}

data\_augmentation\_options {

ssd\_random\_crop { } # Only ever consider full sprites

}

}

**Problem Description – The AI Player**

The AI player software will likely be the biggest piece of software for this project. In addition to running the models for object detection and game play, it will need to act as a bridge between the emulator and the models, and also to display images to an actual screen so that a human can see how well the player plays.

AI player to emulator network communication

The communication between the player software and the emulator will most likely be over a simple bi-directional TCP/IP based protocol. The following functionality will be implemented:

* Player to request screen update from emulator, and emulator responding with the current pixel values for the internal frame buffer used to represent the screen. The player will store the pixels in an identical frame-buffer, so that the data can be fed to the objet detection model.
* Player to send joypad input to the emulator. That is, simulate key presses. The NES, and the emulator, uses a one-byte bitmap to represent the current state of the joypad, and the player will send pre-formatted bitmaps with the correct key state.
* Player to request a reset the console and restart from the beginning. This particular functionality is not strictly required, but it is expected that there will be many training iterations when the AI model learns to play, and stopping and restarting the emulator, and re-establishing TCP/IP communication will take time, particularly if the model is trained in the cloud where stopping and starting an instance is not cheap in terms of time.

All the network operations will be initiated by the AI player. The reason for this decision is that some operations of the player, such as object detection and player logic is potentially quite CPU intensive. If the emulator were in control of sending screen updates to the player, the player might not be able to keep up with the amount of data, which is avoided if the player instead requests updates when it is ready.

AI player data flow

The AI player will request screen updates from the emulator, and use the response to fill its internal frame-buffer. The frame-buffer will then be fed into the TensorFlow object detection model, and the output from the model will be a set of bounding boxes indicating objects.

The location and size of the object bounding boxes will be fed into the neural network responsible for game-play decisions (The GPN – game play network). The GPN will output a decision of current desired key state. For example, should Mario be jumping now, or should he be running to the left, and so on.

Game play network

The actual game playing decisions will be taken by a Game Playing Netork (GPN). It will consist of a neural network with several layers. The network will have one output neuron per possible operation of the joypad. There are four directional buttons, of which two adjacent buttons can be pressed at once (for example, up and left), and also two relevant buttons (‘a’ and ‘b’), giving a total of 9 possible directional outputs (including ‘none’), and 4 different states of the buttons (for a total of 9 \* 4 = 36 states).

The input to the network will be a neuron for each of the following (all distances normalised to 0-1):

* Distance to next enemy relative to Mario (both left and right)
* Distance to next obstacle (left and right)
* Distance to next hole (where Mario can fall off the screen and hence die).

It is possible that as the solution gets implemented, more inputs will be required, but these are not known yet. For example, the game levels in Super Mario Brothers are time limited, and one potential input would be the time left until time is up (Perhaps to get Mario to run as time runs out).

Training of the GPN

The project will apply a reinforcement learning method, loosely combined with an evolutionary approach using an algorithm as described below:

1. The GPN will initially be set up with random weights.
2. From the current GPN, generate n additional GPNs by changing k random weight values by a small amount.
3. Play the n newly generated networks.
4. Using the reward function, select the GPN with the highest reward and make that the current one.
5. Loop from 2

In essence, to clear a level in Super Mario Brothers, the player has to move as far as possible to the right (until Mario reaches the flag pole placed at the very right of a level). A reward function that rewards moving to the right as quickly as possible will be developed to reflect this.

Other functions of the AI player software

In addition to the functions mentioned previously, the AI player software will also need to be able to detect when the game has finished – that is, when Mario runs out of time, is killed by a monster, or falls down a hole off the screen.

It will also need to be able to decode the textual information at the top of the screen to determine the amount of time left for the level. As can be seen from Figure 2, the information needed to identify the time is just the numbers 0-9, and with the numbers always being uniform and in the exact same locations, the software will use a simple byte pattern match to determine the number values.



Figure - typical game play scene

The AI player software will also be responsible for calculating the reward function. Inputs to the function will include the time left, and the number of pixels that Mario has moved to the right.

Super Mario Brothers involves elements of collecting coins for extra rewards, and the concept of in-game points that can be collected by killing enemies, or completing a level quickly. However, as these factors do not directly affect the objective of completing a level, they will be ignored.

**Tools and programming languages**

This project will utilize the LaiNES open source NES emulator. It is a multi-platform emulator, written in roughly 1000 lines of C/C++, and it can be extended to provide the networking functions required by the rest of the project (Orru, 2017).

All other software will be written in Python, and use TensorFlow (Google/Tensorflow, 2019) for object detection, and the open source PyGame library for graphical functionality (Shinners, 2000).

**Fall back position**

The stated aim of this project is to create a piece of software that can play Super Mario Brothers, and at least clear level one. As this is quite an ambitious undertaking, a suggested fall back position should the main aim fail is:

* Modify the emulator to send/receive data over the network
* Create a training set for object recognition using a Python program
* Train a TensorFlow model using the dataset
* Create the beginning of the AI player software, such that it can at least connect to the emulator, display the relevant graphics on screen, and use the object detection model to identify objects on the screen (and draw bounding boxes around them)

**Methodology and work plan**