**Project**

**Machine Learning in Gaming**

**ABSTRACT:**

This project aims at recreating Seth Bling’s MariFlow Neural Network and understanding its different settings and methodology that was used to train this model. The default configurations are then changed and the produced outcomes are discussed and analysed. This project also aims at understanding the usage of Tensor flow library to train a network.

**INTRODUCTION:**

This report briefly explains how to replicate Seth Bling’s MariFlow Neural network and all the software installations it needs. It also depicts different outcomes produced when the default configurations were changed. Based on the results obtained, the best configurations for this neural network are analysed.

**METHODOLOGY:**

A Neural Network is a type of Artificial Intelligence which can learn by example. In MariFlow project, the neural network is trained to mimic user’s own play style. Mariflow carefully examines my own game play in order to try and figure out exactly how I tend about using the controller to drive the kart so that it can drive just like me.

The network started out by knowing nothing about Super Mario Kart. Its only source of knowledge about the game was the game play footage. The goal is for its driving to be indistinguishable from the human. It does not know or care how well it’s playing. Its only goal is to mimic the recorded play style as close as possible.

The inputs to this network is a low resolution grey scale version of the screen. Each pixel is represented by a number between -1 and 1. After applying large sets of sigmoid calculations, the output is a set of 200 neurons that forms the 1st layer. Each neuron contains a value between -1 and 1. This is also represented by grey scale pixels. Finally, it outputs 8 values, one for each button that is capable of pressing. These are the predictions of what button the player is likely to press. All these computations happen 15 times per second.

**Steps to replicate MariFlow Neural Network:**

**Step 1: Install required software**

1. Python 3.5.0 or higher
2. CUDA Development Tools 8.0.61 (<https://developer.nvidia.com/cuda-80-ga2-download-archive>)
3. cuDNN v7.0 (<https://developer.nvidia.com/rdp/cudnn-download>): For this, you have to create an account and then download the software. Ensure that you add the directory where you installed the cuDNN DLL to your %PATH% environment variable.
4. Visual C++ 2015 Redistributable (<https://www.microsoft.com/en-us/download/details.aspx?id=48145>)
5. Tensorflow-gpu or Tensorflow 1.4: This can be installed using native pip command.

Command- pip3 install –upgrade tensorflow

To verify your installation:

* Invoke python from you shell
* Type in the following short program

>>> import tensorflow as tf

>>> hello = tf.constant(‘Hello, TensorFlow!’)

>>> sess = tf.Session()

>>> print(Sess.run(hello))

* If the output is as follows, then the Tensorflow is successfully installed.

Hello, TensorFlow!

Link: <https://www.tensorflow.org/install/install_windows>

1. Pygame: This can be installed by pip command

Command- pip install pygame

1. Bizhawk 1.12.2: <https://github.com/TASVideos/BizHawk/releases>
2. Super Mario Kart ROM: <http://www.completeroms.com/dl/super-nintendo/super-mario-kart-u-/12673>
3. Download dsp1b.rom from <https://caitsith2.com/snes/dsp/dsp1b.rom> and place it in “Firmware” folder of Bizhawk directory.
4. Finally, download the MariFlow zip folder of Seth Bling from <https://www.dropbox.com/s/m59prsodl2t8bec/MariFlow.zip?dl=1>. Copy the contents of the “Lua” folder of downloaded zip folder and copy them to the Bizhawk’s “Lua” folder.

**Step 2: Capturing Training Data**

The training data here is the recorded version of your game. To capture training data, you need to run a Bizhawk script called Capture.lua from Pizhawk Lua Console.

1. Open Bizhawk directory and run “EmuHawk” application. EmuHawk is an emulator that is used to run SNES games.
2. Go to File -> Open ROM -> and select the Super Mario Kart Rom file that you downloaded.
3. Go to Tools -> Lua Console
4. Go to Script in the Lua Console window and open “Capture.lua” script
5. Load up a course in any cc and pause the game.
6. Click “Start” on Cpature.lua window.
7. Unpause the game and play as many courses as you like.
8. At the end of each course the game will freeze momentarily as it writes the training data to a file. Click “Stop” to stop recording.

Your game will be recorded in a .txt file in <BizHawk>\Lua\Capture directory.

**Step 3: Training a Neural Network**

Training a neural network can be done in a lot of different ways. The architecture of the network and other different configurations have to be chosen. I have experimented with a various set of configurations whose results and observations are shown in this report under “Experiments” section.

Before experimenting with various configurations, we need to understand the attributes of a config file.

There are 2 main parts in a config file that is used to train the network- Data and Architecture.

“Data” contains all the information related to training data that we acquired and “Architecture” contains information about the neural network.

**Data** has following attributes:

Filename: The name of the training file that is used to train the network.

Sequence Length: The length or amount of input data that neural network can look at a time. The lesser the length, the more number of time steps the network will require to go through entire data.

BatchSize: The number of samples or sequences that it takes to make a prediction.

RecurButtons: This is a Boolean value. It clarifies whether to remember the previous step’s button press or not.

**Architecture** has following attributes:

Number of layers and number of neurons in each layer.

**Checkpoint**: The directory when the trained models should be saved is defined in this part of the config file.

**Train** has following important attributes:

DropoutKeep: The probability of output that is kept during the training process.

LossFunction: It defines the loss function that is used for training. It takes a value of either “Mean Squared Error” or “Cross Entropy”

**Command to train the network:** After defining your own config file, open a command prompt in the MariFlow folder and run:

Python Train.py <config\_file>

Where, <config\_file> is the name of your configuration file.

It may take several minutes for the training process. Pressing Ctrl+C during training, will manually save a checkpoint file and continue training whereas pressing Ctrl+Break will exit the training without saving. You can resume training from latest checkpoint file by running the same python command.

**Step 4: Running the Neural Network:**

Once the neural network is trained and it has written a checkpoint file, we can run the network as follows:

1. On the command prompt, from Mariflow folder, run

python RunLive.py <config\_file>

1. The above command will output a hostname that we will need to connect it with the emulator.
2. In the emulator, run “RemotePlay.lua” script in Bizhawk Lua Console.
3. This will bring up a small window.
4. Enter the hostname that was given by the python command and click “NN Start”.

The Emulator Client is now connected to the server and once we are in a level, it should start playing.

Once the neural network starts playing, a pygame window is displayed which shows the input sets, number of layers and the predicted output.

**EXPERIMENTS:**

Experiments with different configuration settings have been made.

**Experiment-1:**

For the first table, a batch size of 12 and sequence length of 50 is taken in common for a different set of layers and neurons.

Two layers with neuron sets 32-16, 50-25, 80-40, 100-50 and 3 layers with neurons 100-50-25 and 160-80-40 are taken and the observations are noted down in the below table. The changes in Finishing Time is observed for a particular set of Training configurations.

All the models were trained until they reached an approximate of 150000 number of batches.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Architecture** | | | **Running** | | | **Training** | | | |
| Layer 1 | Layer 2 | Layer 3 | Finishing Time | Time (sec) | Place | Batches | Cost | Batch Size | Sequence Length |
| 160 | 80 | 40 | 01:39.2 | 99.200 | 2 | 159734 | 0.0441 | 12 | 50 |
|  |  |  | 01:45.8 | 105.840 | 5 |  |  |  |  |
|  |  |  | 01:42.0 | 101.950 | 3 |  |  |  |  |
|  |  |  | 01:40.6 | 100.610 | 4 |  |  |  |  |
|  |  |  | 01:47.0 | 107.000 | 6 |  |  |  |  |
|  |  |  |  | **102.920** | **4.0** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 100 | 50 | 25 | 01:41.0 | 101.010 | 5 | 151847 | 0.0435 | 12 | 50 |
|  |  |  | 01:47.2 | 107.230 | 5 |  |  |  |  |
|  |  |  | 01:47.6 | 107.600 | 5 |  |  |  |  |
|  |  |  | 01:39.0 | 99.010 | 3 |  |  |  |  |
|  |  |  | 01:40.4 | 100.390 | 3 |  |  |  |  |
|  |  |  |  | **103.048** | **4.2** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 100 | 50 |  | 01:42.6 | 102.630 | 4 | 100272 | 0.0426 | 12 | 50 |
|  |  |  | 01:57.0 | 116.990 | 8 |  |  |  |  |
|  |  |  | 01:39.3 | 99.280 | 2 |  |  |  |  |
|  |  |  | 01:41.4 | 101.350 | 4 |  |  |  |  |
|  |  |  | 01:48.8 | 108.840 | 7 |  |  |  |  |
|  |  |  |  | **105.818** | **5** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 80 | 40 |  | 01:40.4 | 100.410 | 4 | 156970 | 0.045 | 12 | 50 |
|  |  |  | 01:41.7 | 101.650 | 3 |  |  |  |  |
|  |  |  | 01:39.5 | 99.500 | 2 |  |  |  |  |
|  |  |  | 01:42.5 | 102.520 | 4 |  |  |  |  |
|  |  |  | 01:50.0 | 110.000 | 7 |  |  |  |  |
|  |  |  | 01:44.4 | 104.440 | 5 |  |  |  |  |
|  |  |  |  | **101.704** | **3.6** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 50 | 25 |  | 01:38.8 | 98.830 | 2 | 151120 | 0.0459 | 12 | 50 |
|  |  |  | 01:45.3 | 105.260 | 5 |  |  |  |  |
|  |  |  | 01:41.5 | 101.450 | 2 |  |  |  |  |
|  |  |  | 01:50.6 | 110.630 | 7 |  |  |  |  |
|  |  |  | 01:41.3 | 101.340 | 2 |  |  |  |  |
|  |  |  |  | **103.502** | **3.6** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 32 | 16 |  | 02:50.2 | 170.240 | 8 | 183330 | 0.0428 | 12 | 50 |
|  |  |  | 01:42.8 | 102.750 | 3 |  |  |  |  |
|  |  |  | stopped |  | 8 |  |  |  |  |
|  |  |  | 02:19.8 | 139.800 | 8 |  |  |  |  |
|  |  |  | 03:00.8 | 180.800 | 8 |  |  |  |  |
|  |  |  |  | **148.398** | **7** |  |  |  |  |

**Table-1**

**Observation:** The network with two layers containing 80 neurons in the first layer and 40 neurons in the second layer was observed to be the best one. Mario in this architecture played intelligently by taking shortcuts. It was not only trying to finish the race but was also trying to win the race. It did not stop at any time during the play.

The network with 2 layers having neurons 32 and 16 in first and second layer respectively was observed to be the worst one even though it was trained for a longer time than other models. Mario kept stopping in middle of the game. It hits an obstacle and stops moving forward until someone pushed it from behind.

The network with 3 layers having neurons 160-80 and 40 was observed to be the second best architecture. The training time for this model to reach an approximate of 150000 batches was a little more when compared to other architectures but the game play was decent. Mario in this network was trying to escape from the obstacles thrown by other players.

**Experiment-2:**

For the second table, a batch size of 3 and sequence length of 12 is taken in common for a different set of layers and neurons.

Two layers with neuron sets 32-16, 50-25, 80-40, 100-50 and 3 layers with neurons 100-50-25 and 160-80-40 are taken and the observations are noted down in the below table. The changes in Finishing Time is observed for a particular set of Training configurations.

All the models were trained until they reached an approximate of 150000 number of batches.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Architecture** | | | **Running** | | | **Training** | | | |
| Layer 1 | Layer 2 | Layer 3 | Finishing Time | Time in sec | Place | Batches | Cost | Batch Size | Sequence Length |
| 160 | 80 | 40 | Stopped | 180.000 | 8 | 150111 | 0.0582 | 3 | 12 |
|  |  |  | 01:49.24 | 109.240 | 6 |  |  |  |  |
|  |  |  | Stopped | 180.000 | 8 |  |  |  |  |
|  |  | -1 | 01:47.36 | 107.360 | 5 |  |  |  |  |
|  |  |  | 01:40.56 | 100.560 | 2 |  |  |  |  |
|  |  |  |  | **135.432** | **5.8** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 100 | 50 | 25 | 01:41.77 | 101.770 | 4 | 279475 | 0.08 | 3 | 12 |
|  |  |  | Stopped | 180.000 | 8 |  |  |  |  |
|  |  |  | 01:41.29 | 101.290 | 5 |  |  |  |  |
|  |  |  | 01:45.75 | 105.750 | 5 |  |  |  |  |
|  |  | -1 | 02:13.67 | 133.670 | 8 |  |  |  |  |
|  |  |  |  | **124.496** | **6** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 100 | 50 | -1 | 02:20.73 | 140.730 | 8 | 162679 | 0.0552 | 3 | 12 |
|  |  |  | 01:40.61 | 100.610 | 4 |  |  |  |  |
|  |  |  | 01:40.79 | 100.790 | 4 |  |  |  |  |
|  |  |  | 01:45.84 | 105.840 | 5 |  |  |  |  |
|  |  |  | Stopped (Tried to move left & right but not forward | 180.000 | 8 |  |  |  |  |
|  |  |  | 01:37.95 | 97.950 | 4 |  |  |  |  |
|  |  |  |  | **120.987** | **5.5** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 80 | 40 |  | Stopped | 180.000 | 8 | 153893 | 0.0375 | 3 | 12 |
|  |  |  | 01:39.89 | 99.890 | 4 |  |  |  |  |
|  |  |  | 01:43.13 | 103.130 | 3 |  |  |  |  |
|  |  |  | Stopped | 180.000 | 8 |  |  |  |  |
|  |  |  | Stopped | 180.000 | 8 |  |  |  |  |
|  |  |  |  | **148.604** | **6.2** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 50 | 25 |  | 01:46.58 | 106.580 | 6 | 196208 | 0.0453 | 3 | 12 |
|  |  |  | 01:43.86 | 103.860 | 4 |  |  |  |  |
|  |  |  | 01:45.75 | 105.750 | 5 |  |  |  |  |
|  |  |  | 01:44.14 | 104.140 | 4 |  |  |  |  |
|  |  |  | 02:01.64 | 121.640 | 8 |  |  |  |  |
|  |  |  |  | **108.394** | **5.4** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 32 | 16 |  | Stopped in last lap | 180.000 | 8 | 264754 | 0.0475 | 3 | 12 |
|  |  |  | Stopped in last lap | 180.000 | 8 |  |  |  |  |
|  |  |  | 01:47.51 | 107.510 | 5 |  |  |  |  |
|  |  |  | Stopped in 4th lap | 180.000 | 8 |  |  |  |  |
|  |  |  | 02:00.51 | 120.510 | 8 |  |  |  |  |
|  |  |  |  | **153.604** | **7.4** |  |  |  |  |

**Table-2**

**Observation:** The network with two layers and neurons 50-25 in first and second layer respectively was observed to be the best one. This was the default configurations used by Seth Bling as well in his network. Mario of this architecture never stopped. It tried to take shortcuts to beat other players in the race and finished the trial intelligently.

Mario in network with neurons 80 and 40 in first and second layer was driving in a criss-cross manner. The average finish time in this architecture was 148 seconds.

The worst configurations was observed to be the one with 2 layers having 32 and 16 neurons. The average finish time in this architecture was 153 seconds and Mario always stood last in the race.

However the architecture with BatchSize 12 and Sequence length 50 was considered better than that of the one with BatchSize 3 and Sequence length 12 because with more sequence length, the amount of input data given to the network at a particular instant is more and hence it does not have to remember a move that was done a long time ago, instead it will just remember a move that was done few milliseconds ago and hence it is considered to perform more accurately.

**Experiment-3:**

The above two experiments were done with the Mean Square Error as their Loss function. In this experiment, we change the loss function to “Cross Entropy”. We perform the experiments on 2 layer architecture containing 80-40 and 50-25 neurons with BatchSizes 3 and 12 and sequence length 12 and 50.

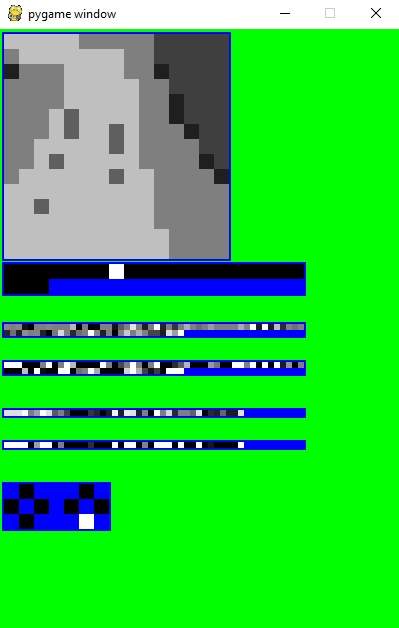
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Architecture** | | | **Running** | | | **Training** | | | |
| Layer 1 | Layer 2 | Layer 3 | Finishing Time | Time in sec | Place | Batches | Cost | Batch Size | Sequence Length |
| 80 | 40 |  | 01:40.3 | 100.330 | 4 | 236772 | 0.5226 | 3 | 12 |
|  |  |  | 01:40.3 | 100.290 | 2 |  |  |  |  |
|  |  |  | 01:42.8 | 102.770 | 3 |  |  |  |  |
|  |  |  | 02:23.0 | 142.950 | 8 |  |  |  |  |
|  |  |  | 03:00.0 | 180.000 | 8 |  |  |  |  |
|  |  |  | **02:05.3** | **125.268** | **5** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 50 | 25 |  | 01:40.8 | 100.810 | 2 | 260079 | 0.9174 | 3 | 12 |
|  |  |  | 01:44.9 | 104.890 | 5 |  |  |  |  |
|  |  |  | 01:40.0 | 100.010 | 2 |  |  |  |  |
|  |  |  | 01:39.9 | 99.890 | 2 |  |  |  |  |
|  |  |  | 01:42.4 | 102.380 | 4 |  |  |  |  |
|  |  |  | **01:41.6** | **101.596** | **3** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 80 | 40 |  | 01:44.0 | 103.960 | 3 | 89508 | 0.7837 | 12 | 50 |
|  |  |  | 01:40.2 | 100.230 | 3 |  |  |  |  |
|  |  |  | 01:41.6 | 101.550 | 4 |  |  |  |  |
|  |  |  | 01:43.2 | 103.200 | 3 |  |  |  |  |
|  |  |  | 01:38.7 | 98.680 | 2 |  |  |  |  |
|  |  |  | **01:41.5** | **101.524** | **3** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| 50 | 25 |  | 01:42.1 | 102.130 | 4 | 59290 | 0.6277 | 12 | 50 |
|  |  |  | 01:45.7 | 105.700 | 5 |  |  |  |  |
|  |  |  | 01:41.5 | 101.450 | 5 |  |  |  |  |
|  |  |  | 01:40.6 | 100.570 | 3 |  |  |  |  |
|  |  |  | 03:00.0 | 180.000 | 8 |  |  |  |  |
|  |  |  | **01:58.0** | **117.970** | **5** |  |  |  |  |

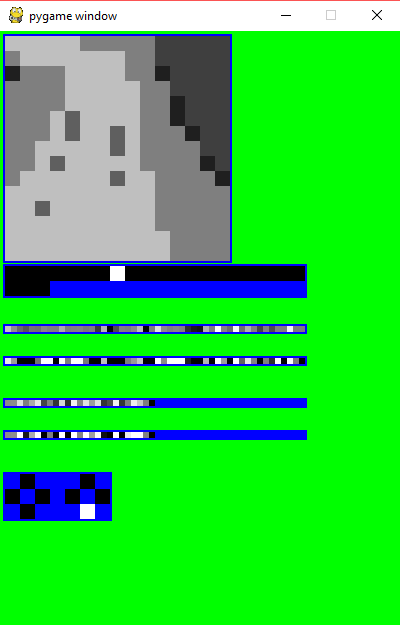
**Table-3**

**Observation:** The performance of architecture containing 80-40 neurons (batch size-3 and sequence length 12) was very bad when the loss function was Mean Squared Error. With Cross Entropy as its loss function, the behaviour was pretty good when compared to the previous one.

However, the best configuration was considered to be the one with neurons 50 and 25. It was approximately 10 seconds faster than the one with Mean Squared Error as its loss function.

**SCREENSHOTS:**

**2 Layer n/w with 50-25 neurons 2 Layer n/w with 80-40 neurons 3 Layer n/w with 160-80-40**

****

****

**Super Mario Kart Game**

**CONCLUSION:** After conducting various experiments with different configuration settings, the training architecture with 2 layers containing 80 and 40 neurons respectively and with batch size 12, sequence length of 50 was observed to be the best one. The average finish time of this architecture is 101 seconds and Mario stood an average of 3rd position in the race.

I think that the sequence length of this architecture is the main reason why this network performs better when compared to other architectures. Since it has a high sequence length, it can read more data in one look and can remember previous moves more accurately than the ones with less sequence length. Hence the chances to predict the next move are high in this architecture.

BizHawk is a multi-platform emulator with full recording support and Lua Scripting. This helps us run any SNES game with minimal installation requirements. This training process can be used to train any SNES game with similar process by adjusting the background architecture of each game with respect to its own game architecture.

The video recording of some of these architectures are given at this link:

**MariFlow 80-40:** <https://marist.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=098647e4-e089-4d0e-9c76-a8d30140c3f6>

**MariFlow 50-25:** <https://marist.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=64bb3d4d-4514-4a8d-894d-a8d30133aa44>

**REFERENCES:**

# Seth Bling’s Youtube Video- MariFlow - Self-Driving Mario Kart w/Recurrent Neural Network

# Bizhawk- http://tasvideos.org/BizHawk.html