MariFlow Manual

SethBling

Introduction

This manual explains, roughly, how to install, train and run MariFlow, a recurrent neural network for playing Super Mario Kart. If you want to try anything but the sample, I recommend learning about neural networks, there's a lot to know.

My video about MariFlow: https://www.youtube.com/watch?v=Ipi40cb_RsI

Installation

Requirements

The following software is required to run MariFlow. I've included the version numbers that I have installed on my computer. Some of the requirements are stricter than others. See here for more information on installing TensorFlow.

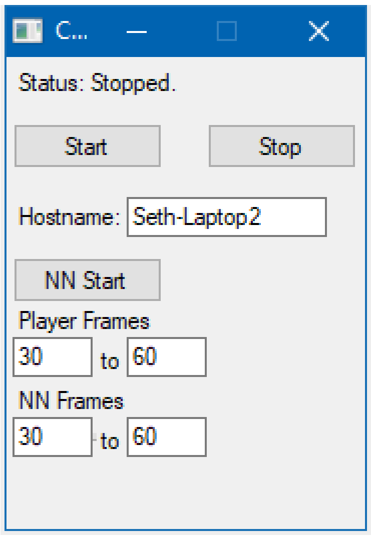
- Python 3.5.0 (64bit)
- CUDA Development Tools 8.0.61 (64bit)
- CuNN 6 (64bit)
    - For some reason downloading this requires creating an account.
    - With the release of Tensorflow 1.4, you may need CuNN 7 instead.
- Visual C++ 2015 Redistributable
- tensorflow-gpu/tensorflow 1.3 (can be installed with *pip* etc.)
- pygame (can be installed with *pip* etc.)
- Bizhawk 1.12.2
- Super Mario Kart ROM.

Placing Files

Download MariFlow.zip.

Place the contents of the zip file anywhere you'd like. Within your Bizhawk directory, there's a folder called Lua. Copy the contents of the Lua folder in the zip file into the Lua folder in the Bizhawk installation. When you start training models, make sure they aren't in a Dropbox folder, or the process will fail.

Capturing Training Data



Status: Stopped.

Hostname: Seth-Laptop2

Player Frames: 30 to 60

NN Frames: 30 to 60

To capture training data, you'll need to run a Bizhawk script called `Capture.lua` from the Bizhawk Lua Console (accessed via the *Tools* menu). This will create a small window that you can use to control the script. Load up a course in whatever cc you'd like (or even time trial) and pause the game. Click "Start" on the Capture.lua window. Now you may unpause the game. Run for as many courses as you'd like. You can load save states, or reset the emulator as much as you'd like. At the end of each course the game may freeze momentarily as it writes the training data to file. The data will be saved in a .txt file in the `<Bizhawk>\Lua\Capture` folder. Click "Stop" on the Capture.lua window to stop recording.

Modifying the Neural Network Inputs

Certain aspects of the neural network inputs can be modified within the source code of `Capture.lua`. You can change the screen size, tilt shift properties, and add extra network nodes. Anything that's part of the "cfg" dictionary in `Capture.lua` is modifiable. If you modify any of those constants, they will be recorded in a header of any captured data. This header will allow the data format to remain consistent for any networks that train or run using those constants, even if you further change the constants in the Lua file.

Additional Capture Features

L and R

L and R are not captured as part of the training data. Instead, pressing L or R will mark those frames as non-training frames. That is, they will still be seen during the training process, but no weight will be given to the controller inputs on those frames to improve the neural network. You can use L or R to intentionally drive the kart into a bad situation in order to create training data that shows the network how to recover from that bad situation.

+Screen

This will toggle a display of the neural network inputs as they'll be captured to file.

Capturing Mixed Data

This is a mode where the script hands control between the player and a neural network. This is covered below, and can only be used after you've trained a neural network.

Training a Neural Network

Training a Recurrent Neural Network can be done in a lot of different ways. You have to choose the architecture of the network, and various facets of how the training occurs. I've included some defaults that worked well for me, but to understand how changing the settings will affect things, you'll need to have some understanding of neural networks in general.

Config Files

All the settings for defining and training MariFlow are contained in a *.cfg file. `defaults.cfg` contains default values for every parameter, and `sample.cfg` has a sample configuration. You can create your own configuration files, too. Once you've created a configuration, open a command window in the MariFlow folder and run:

```
    python Train.py <config file>
```

Here are the parameters supported by the config file:

Section	**Parameter**	**Description**
Data	Filename	List of training file names, one per line (indent additional lines)
Data	SequenceLength	Number of time steps to train at once for backpropagation through time.
Data	BatchSize	Number of sequences to process per batch.
Data	RecurButtons	Whether to include the previous time step's button presses in the input set.
RNN	Layer<*number*>	The size of each layer of the RNN. You can have as many layers as you like, start numbering at 1.
Checkpoint	Dir	The directory to save checkpoints.
Train	DropoutKeep	The probability that each node's output will be kept during training.
Train	MaxGradNorm	The maximum gradient component value.
Train	VariationalRecurrent	A variation of dropout designed for RNNs.
Train	NumPasses	The number of passes through the training set (this keeps sequences from always containing the same training data).
Train	ValidationPeriod	The number of seconds between validation tests.
Train	LossFunction	Either "mean squared error" or "cross entropy".

Validation

It may take several minutes for the training data to be loaded and the training process to begin. The neural network will periodically be evaluated on a special batch that's set aside for validation (it has the same batch size and sequence length as any other batch). When a new best validation error is found, it will save the neural network to a checkpoint file in the checkpoint directory. If you press ctrl+C during training, it will manually save a checkpoint file and continue training. Press ctrl+Break to exit training without saving. You may resume training from the latest checkpoint file by simply running the same python command again.

Running a Neural Network

The Neural Network Server

Once you've trained a neural network and it's written a checkpoint file, you can run the neural network live by connecting it to Bizhawk. On the command line, from the MariFlow folder, run this command:

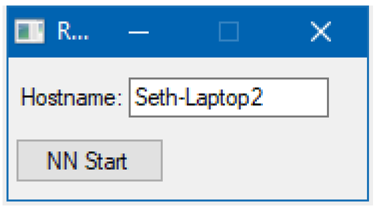
```
python RunLive.py <config file> [checkpoint number]
```

If you don't specify a checkpoint number, it will run the last checkpoint file saved. Otherwise it'll use the checkpoint saved at the batch number specified (this is the number at the end of the checkpoint file).

It may take several minutes to load the neural network, but once you do it'll tell you what hostname and port it's listening on for the emulator. Take note of the hostname, you'll need it to connect with the emulator.

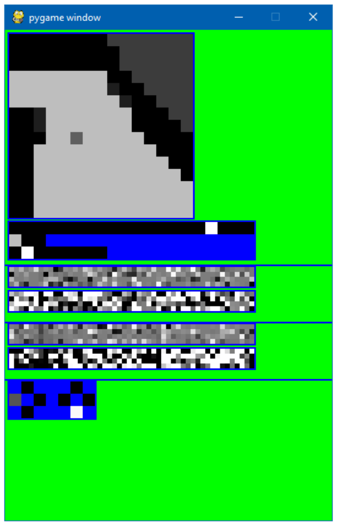
The Emulator Client

To connect the emulator, run the `RemotePlay.lua` script in the Bizhawk Lua Console. This will bring up a small window. Write the hostname you noted from the server into the textbox and click "NN Start". Once you're in a level, it should start playing. You can connect and disconnect from the server by clicking the button again.



Hostname: Seth-Laptop2

Network Display



Once the neural network is running, a window will pop up and display the neural network's inputs, states of the hidden layers, and predicted outputs. This window will freeze if you pause the emulator. Its background is green so that it can be overlaid on a livestream using a chroma key to make the background transparent.

Capturing Mixed Data

The `Capture.lua` script can be used to interactively create training data. In this mode, the script hands control back and forth between the player and the neural network. Once you've trained a neural network and run the server, you can operate in this mode to create training data that will cover a wider array of situations a self-driving kart might face. In the `Capture.lua` window, you can write the hostname to connect to the NN server. There are configurable minimum and maximum numbers of frames for handoff times. Remember that the game runs at 60fps.