Supervised Learning for Super Mario Bro's

Functional Specification

Computer Applications Final Year 2016

Eoin Murphy 11487358

Link to project blog:
http://blogs.computing.dcu.ie/wordpress/eoinmurphyfinalyear/

Table Of Contents

1. Introduction

1.1 Overview

We have seen over the past couple of year's intelligent machine's have become an every day part of our lives. Be it in Data Analysis, Product Recommendation, Facial & Speech Recognition, even to what we see on our social media there is a underlining algorithm or algorithms at the heart of these which has been fine tuned through usage of machine learning.

With this in mind my final year project entitled 'Supervised Learning For Super Mario Bro's' involves designing, building and training a neural network to learn how to play the Nintendo game Super Mario Bro's. The reason for this project is to get a greater understanding of the processes involved in designing machine learning solutions for problems. As with a lot of applications of artificial intelligence, there is the problem of how do we gauge if we have found the best solution? Since we as humans are forms of intelligence we can say if a machine can find a solution better then a human can then that machine can be said to have some form of intelligence. One of the best ways of showing this is by designing machines to play against humans in games (IBM's Deep Blue) [1]. That is the reasoning behind my project, the goal being to hopefully demonstrate that my network has the capability to play Super Mario better than a human. The value I will use to measure this is how fast the player completes a given level.

This project Involves designing the networks inputs and the encoding scheme for them, designing the networks architecture, Recording Exemplars to train the network and the intergartion with the BizHawk Emulator along with learning the programming language LUA to do such.

1.2 Glossary

- **Neural Network -** A Neural Network is a interconnected group of nodes. they consist of 3 layers of nodes the first being the input layer, secondly the hidden layer (some networks can consist of multiple hidden layers this is known as deep learning) [2] and finally the output layer. The input layer is simply put what the network sees and is passed to the nodes of hidden layer via weighted connections called synapses. Each node of the hidden layer calculates the sum of all inputs to it times the weight of the connection, this sum is passed through a activation function[3] to determine whether or not the node “fires” This result is then passed to the output layer and again by use of weighted connections between the hidden layer and output layer determines what is outputted by the network.

  In the case of this project I'm using a type of network type called a fully connected feed forwarding network. Simply put this means all nodes are directly connected to each other and data passed through the network goes in only one direction through it. So throughout this document when I refer to a neural network I'm referring specifically to this type (This is subject to change at the end of the project based on experimenting with the networks design but unlikely to be the case).

  For more information on the different types of networks

- **Supervised Learning –** Supervised Learning is the process of training a by neural network by showing the network a set exemplars. From this set the network adjusts the weights in its synapses so that when it receives a unseen input then that shown in the training set it can approximate the correct output. The algorithm I will be using to achieve this is back-propagation.
- **Exemplars –** Exemplars are a set of inputs mapped to their corresponding Outputs used in the training of the network.
- **BizHawk –** BizHawk is a Emulator for the Nintendo Entertainment System (NES) Hardware which allows the users to run ROM Files of the games developed for the NES and to play them as they would on such.

  For more information on BizHawk
- **LUA –** Lua is a scripting Language which is used with BizHawk to interact with the emulator and game running.
- **ROM File –** a ROM File is a digital copy of the data contained on a video game cartridge.

2. General Description

2.1 System Functions

The system will present users with a GUI within the BizHawk emulator to give users the options to record exemplars from their own game play which will be recorded to a DAT file. It will allow users to train the network using these file's containing the exemplars. Users are then able to exploit the trained network to play Super Mario Bro's.

Along side this core functionality users will have access to a settings interface this will allow users to alter certain variables within the system to give them more control on the network and how it is trained examples of these variables will be the number of nodes in the hidden layer, the rate of change with use of the back-propagation algorithm paths to the Exemplar file and the network values file.

The system will also provide a graphical representation of the network while it is running so that users can get a better understanding of what the network is doing and how it is doing it. This will show the inputs to the network in real time along with what neurons are firing and what buttons on the NES controller the network is pressing corresponding to the networks output.

2.2 User Characteristics & Objectives

It is very likely that users of this system will have very little expertise with this type of project. Due to this fact the system should be extremely simple to use. Where by it takes is a minimal amount of key strokes to go from recording exemplars to learning from them while at the same time hopefully giving the user a insight into what is going on and achieve a good understanding of the processes involved in neural networks. From a users perspective the system should be able to produce a neural network capable playing Mario after they have trained it and being able to do so satisfactory.

Desirable characteristics & objectives of the system are:

1. It should provide users with a good understanding of how the neural network is working
2. If the network is capable to learn in a relatively fast time frame it would be nice to demonstrate that at the core of back-propagation although it may seem intelligent it is not as it is simply copying how the user played. An example of this being if the user only presses the jump button then all the network will do is jump as it will never know to move forward as the user never showed it.
3. One objective is if I can get the network working with Supervised learning early on in development. I hope to work to getting a network trained using Reinforcement learning and allowing users to choose between which methods they would like to train the network.

2.3 Operational Scenarios

Operational Scenarios in this section will be illustrated with use-case specifications.

Use Case #1

1. Brief Description

This Use case Describes how a user uses the system to record Exemplars

2. Actors

2.1 User

2.2 BizHawk Emulator

3. Preconditions

The user has opened the Super Mario Bro's ROM in Bizhawk.

The Lua scripted has been loaded into Bizhawk and is active.

The user has the required emulator save state files saved in the correct path.

4. Basic Flow Of Events

1. The use case begins when the user presses the number 3 on the numpad.
2. The save state to the level for the user to play is loaded.
3. The user is prompted to press any key on the controller to begin.
4. The user presses a key on the controller.
5. The user plays through the level as they normally would.
6. The inputs and outputs for each frame is recorded to a DAT file.
7. The number of frames the player is allowed to play is reached.
8. The use case ends.

5. Alternative flows

5.1 Move by user results in Mario dying

If in step 5 the users moves cause Mario to die this can be as a result of being hit by an enemy or failing into a pit in the map then:

1. BizHawk stops recording frames
2. The save state to the start of the level is loaded
3. The use case resumes at step 3

6. Key Scenarios

6.1 Move by user results in Mario dying

7. Post Condition

7.1 Successful Completion

The Exemplar file name is loaded into the config property RECENT_EXEMPLAR_FILE

7.2 Failure Condition

The Exemplar file is not loaded into the config property as in 7.1

Use Case #2

1. Brief Description

This Use Case describes how a user trains the network.

2. Actors

2.1 User

2.2 BizHawk Emulator

3. Preconditions

The user has opened the Super Mario Bro's ROM in Bizhawk.

The Lua scripted has been loaded into Bizhawk and is active.

4. Basic Flow Of Events

1. The Use Case begins when the user presses number 4 on the numpad.
2. User is prompted to use either the most recent Exemplar file name or to use the file name save in the config as set in the settings interface.
3. User selects which file to use.
4. The lua Script in BizHawk calls the learn function.
5. The user is prompted on screen that the network is training from the exemplar file specified.
6. The learn function has completed.
7. The network values are saved to a DAT file
8. The user is prompted the network has finished training
9. The main GUI is shown back to the user
10. The use case ends

5. Alternative flows

5.1 No Exemplar files have been created

If in step 2 No Exemplar file names are found in the config file then:

1. The user is prompted that no exemplar files have been created.
2. The use case fails

6. Post Condition

6.1 Successful Completion

The network config file is loaded into the config property RECENT_NETWORK_CONFIG

6.2 Failure Condition

The network config file is not loaded into the config property as in 6.1 and the network values are not changed.

Use Case #3

1. Brief Description

This Use Case describes how a user exploits the network to play Super Mario.

2. Actors

2.1 User

2.2 BizHawk Emulator

3. Preconditions

The user has opened the Super Mario Bro's ROM in Bizhawk.

The Lua scripted has been loaded into Bizhawk and is active.

The user has the required emulator save state files saved in the correct path.

4. Basic Flow Of Events

1. The use case begins when the user presses the 5 on the numpad.
2. The save state to the start of the level to be played is loaded.
3. The lua script function Exploit is called.
4. The user is prompted to press 5 again to stop the network.
5. Either the user presses 5 or the number of lives for Mario reaches 0.
6. The network stops Exploiting.
7. The original GUI is loaded to the user.

8. The Use Case ends.

5. Alternative flows

there are no Alternative flows identified.

6. Post Condition

There are no post conditions as throughout the exploiting of the network all inputs and outputs are recorded and saved even if the net fails or succeeds.

Use Case #4

1. Brief Description

This Use Case describes how a user edits the systems settings in the config file.

2. Actors

2.1 User

2.2 BizHawk Emulator

3. Preconditions

The user has opened the Super Mario Bro's ROM in Bizhawk.

The Lua scripted has been loaded into Bizhawk and is active.

4. Basic Flow Of Events

1. The use case begins when the user presses 6 on the numpad
2. a interface with options for 'Change Exemplar file Location','Change Network config file location', 'Edit number of Neurons', 'Edit Learning Rate','Exit Settings' each selected via the number on the numpad respectively.
3. User selects an option.
4. If the user picks any option other then 'Exit Settings' the user is shown the current saved value for that option.
5. The user is prompted to edit the value
6. The user enters new value
7. User hits the enter key to save the new value
8. New value is validated (Spawns Use Case#5)
9. User is returned to the Settings interface
10. User selects 'Exit settings'
11. The Original GUI is displayed
12. The use case ends

5. Alternative flows

5.1 User selects a different option then exit

If in Step 9 the user selects a another option that is not exit then:

1. the Use case resumes at Step 4

5.2 Value is invalid

If the value entered is invalid Then:

1. value is not changed
2. User is notified of the value being invalid
3. Use case resumes at Step 5

6. Key Scenarios

6.1 Value is invalid

7. Post Condition

7.1 Successful Completion

The Config file value for the variable edited is updated to what the user entered

7.2 Failure Condition

The Config file is not updated.

Use Case #5

1. Brief Description

This use case describes the validation of a value entered for update in the settings.

2. Actors

2.1 User

2.2 BizHawk Emulator

3. Preconditions

User has reached Step 8 in Use Case #4

4. Basic Flow Of Events

1. If the value entered is a Path to file, Other wise jump to step 6
2. Try to load the file using io.open in Lua
3. If io.open returns a file not found error jump to step 15
4. return valid
5. Use case ends
6. If the value is for the learning rate Other wise jump to step 11
7. Check if value is in range 0 > c < 1
8. If not in range Jump to step 15
9. Return valid
10. Use case ends
11. Check value is greater then 1
12. if value is less than 1 jump to step 15
13. Return valid
14. Use Case ends
15. Return invalid
16. Use Case ends

5. Alternative Flows

No Alternative Flows identified

6. Post Condition

6.1 Successful Completion

Returns that the value entered is valid

6.2 Failure Condition

Returns that the value entered is invalid

2.4 Constraints

Key Constraints Identified are:

- Reading Ram Addresses – For this project all of the data being read from the Super Mario game comes from reading directly from the Games RAM. This was one of the key deciding factors on choosing Mario for the game to play as there is a large community behind programming/hacking Mario so nearly 80% of the Ram addresses have been mapped detailing their functionality.
- Designing Inputs – The design of the networks inputs can make or break this project and will require a lot of experimentation to find a input design that works.
- Recording Exemplars – Careful consideration most be taken when recording the exemplars to ensure that the values being read in correspond to the data required for learning.

3. Functional Requirements

The following is a lists of the Functional requirement's identified for this project:

- Build the neural network.
- Train the network.
- run the network.
- Read the games RAM adrresses.
- Designing / Encodign inputs.
- Get Inputs.
- Get Outputs.
- Allow users to generate exemplars through game play.
- Read controller values.
- Set controller values.
- Write exemplars to file.
- Write network values to file / import.
- Display user interface.
- Allow users to change system values.

The following describes each functional requirement.

Each Requirement is specified in the following format:

Name

Description

Provide's a description for the Requiment.

Criticality

Describes how essentail This requirement is to the overall system.

Technical Issues

Describes any of the design or implementation issues in meeting the requirement.

Dependencies With other Requirements

Describes the interaction with any other requirements.

Build the Neural Network

Description

The requirement of building the neural network is to write the data structure for the network so the system can store the network weights and the networks nodes as well as functions to pass inputs through the network and generate outputs via forward propagation.

Criticality

The neural network is the core component to the system.

Technical Issues

The networks data structure will need to be coded such that data can be easily propagated through the network and weights should be able to be easily modified during back propagation.

Dependencies with other Requirements

This requirement is dependant on the following:

- 'Get Inputs'
- 'Training the network'

Training the network

Description

Training the network invloves the process of forward propigating exemplar inputs throught the network and then using the Exemplar output for the input set back propigate the error through and adjust the weights for each synapsis.

Criticality

This requirement is as critical a compnent as the network itself as with out a way to train the network we have no way for the network to learn and thus generate appropriate outputs for inputs, After all the project title is supervised learning for super mario bros.

Technical Issues

Much as with designing the inputs this requirement will take a lot of experimentation to get right and will involve reading logs from the networks training to see how it is performing as well as exploiting the network.

Dependencies with other requirements

This Requirement depends on the following requirements

• 'Logging network'

• 'Read exemplar file'

Run the network

Description

The system must be able to forward propagate inputs through the network after training and then handle the corresponding outputs accordantly.

Criticality

Again as the system is to demonstrate a neural network with the capability's to learn and play Super Mario, running the network is a necessary requirement.

Technical issues

Running the network will require many aspects of the system to come toghter for it to work these being reading in inputs and enocding them correctly along with mapping the outout puts of the network to the controller while showing translating all this back into game play so the network plays the game.

Dependencies with other Requirements

This requiment is dependant with the following:

• Get inputs

• Get outputs

• Set controller values

Read Games RAM addresses.

Description

This Requirement is to read the data describing what is happing during the running of the game from the games RAM. Data identified in RAM that is important to the system is as follows:

- Mario's X and Y position in the games level.
- Each Enemys X and Y position in the games level.
- The current tiles displayed on the screen in relation to Mario's positon.
- Flag register as to whether Mario has died.

Criticality

This requirement is the most critical to the system as with out being able to read the games RAM we have no way of providing inputs to the network that describe what is happing during the games run.

Technical Issues

The technical issues attached to this requirement are that some pieces of data require data from one of more additional address. For example to get Marios actual x position you have to read two addresses. The reason for this is due to limitations of the NES hardware the maximum value that can be in an address is 255. As Mario's X position is going to be larger then 255 pixels the developers used a second address to count how many times the original x address reached 255 so to get Mario's actual x Position you multiply the value in the count address by 255 and then add it to the x address this gives you the true X value.

Due to this fact and that the source code for the game is not available to us we have to rely on a addresses map that details what each address is doing [4] to work out how to over come some of these issues.

Dependencies with other requirement's

This requirement does not have any other dependencies with other requirement's mainly due to the fact as it it one of the core elements to the system and as such many other requirements will be dependant on it.

Designing / Encoding inputs

Description

This Requirement involves Identifying and designing the inputs to the network these inputs will be live game data as gathered from the 'Read game RAM Addresses'. These inputs are then to be encoded appropriately for the network to be able to distinguish between different input sets.

Criticality

This Requirement is Critical to the system as incorect input sets along with if the network cannot separate the different sets will cause the network to fail and not show any progress.

Technical issues

This requirement will involve a lot of experimentation to find what input design and encoding works for the network.

Dependencies on other Requirements

This requirement is dependant on having meet the 'Read games RAM

addresses' requirement. This being as this requirement will require this data to begin working on it.

Get Inputs

Description

This requirement is to get the inputs from the game using the memory address values and have them encoded to then be able to pass to the network, or write to a file if used in generating exemplars.

Criticality

This Requirement is a again highly critical to the system as is a requirment for the network to be able to function.

Technical Issues

As inputs are taken from the games RAM which is in hex we may need to convert from hex depending on the encoding scheme.

Dependencies with other Requirements

For this requirement the following requirements must be meet first:

- Reading game RAM addresses
- Designing / encoding the inputs

Get Outputs

Description

This requiment is to read the outputs from the users gameplay when recording exemplars which are taken from reading the controller and are then encoded to a applicible format. This Requirment also is for reading the networks outputs which are then translated into controller presses via the set controller requirement.

Criticality

This requirement highly critical to the system as Incorrect outputs will cause the the network to fail

Technical Issues

No Significant Technical issues have been identifed to be asscoicated with this requirement.

Dependencies with other Requirements

This Requirement is dependant on 'Reading controler values' as the controllers values are the outputs for the network which need to be encoded.

Allow users to generate exemplars through game play.

Description

Users of the system should to be able to generate exemplars for the network to learn this is done by the user playing the game as normal during this the exemplars are to be written to a DAT file to be used later by the system.

Criticality

This requirement is highly critical to the system as without exemplars the network cannot be trained.

Thecnical issues

Technical issues for this requirement involve ensuring that exemplar inputs are correctly mapped to the corrisponding exemplar output as if the inputs are not mapped correctly to he outputs the network will adapt its self incorectly which can result in unexpected out by the network with unseen inputs which can cause the network to not perform as required.

Dependencies with other requirements

This requirement is dependant with following requiments:

- get inputs
- get outputs
- write exemplars to file

Reading Controller values

Description

The requirement is to be able to read what button the user is pressing on the controller in real time during game play.

Criticality

The networks output corresponds to buttons on the controller when recording exemplars the system must be able to read what button(s) the player is

pressing so these can be mapped in the exemplar file for the network to learn from.

Technical Issues

There are no significant Technical issues identified for this requirement.

Dependencies with other requirements

This requirement is not dependant with other requirements

Set Controller Values

Description

Outputs from the network are to be translated into key presses on the NES contoller which are to be set and executed during the networks runtime.

Criticality

With out this requirement being meet the network will not be able to move Mario through the level and thus recive new inputs to the network and accomplish its task of playing the game.

Technical issues

There are no significant Technical issues identified for this requirement.

Dependencies with other Requirements

This requirement is dependant upon the 'Get Outputs' requirement.

Write exemplars to file

Description

The system must be able to write exemplars generated to a DAT file so that they can be loaded and passed into the network for trainning. It also allows for the user to be able to record multiple sets of exemplars as some exemplars may not provide strong examples for the network to learn from and also allow for the network to be trained multiple times of the same exemplar sets if need being.

Criticality

This requiment is not highly critical to the system as the exemplars can be

stored localy in the system using arrays. However it will allow for the examination of exemplars to help in determinng pit falls in the networks learning stages.

Technical issues

The main issue is in determining a data structure for these DAT files to allow for them to be easily read by the system.

Dependencies with other Requirements

This Requirement is dependant on being able to generate Exemplars from game play as this is our source of exemplars.

Write network values to file / Import

Description

This Requiment allows for the system to write the networks values to a DAT file and then import from it so that the network can be rebuilt with out the need for training and can also be examined for finding problems with the network.

Criticality

Again simlar to 'Writing Exemplars to file' network values can be stored locally but with out this requirement it means ever time the system is re-run the network will need to be re-trained. If a working netwok has been generated and the values are not saved, re training the network will more then likely never result in a network with the same values.

Technical issues

The technical issues are again the same as for 'Writing exemplars to file' a suitable data structure must be implemented.

Dependencies with other requirements

This requirement is dependant on being able to 'train the network'.

Display User interface

Description

The system must be able to provide a sutible user interface to allow for the users to select and use the different systems functions. Along with this the User inteface should display give the user feed back on the systems running and provide a graphical display of the networks opertations during run time.

Criticality

As this requirement describes how the user will interact with the system it has a high criticality to the overall system.

Technical issues

Technical issues associated with this are primarly in design as the interface must be intuitive to use by the user but must also be as light weight as possible as adding graphical elemnts to the bizHawk emulator can have dramatic performance effects on the rendering of the game resulting in low FPS counts which can make the game close to unplayable.

Allow users to change system values

Description

The system should allow for users to have control over various function variables these variables include the location paths to Exemplar files, Network files along with variables used by the network during learning such as Learning rate, number of neurons.

Criticality

This functon is not critical to the systems overall performance but is to give users a feeling of control over the network while hopefully providing them with a better understanding of the inner workings of the system.

Technical issues

As this allows for users to change values critical to the system validation checks must be performed on any changes to ensure they wont cause underling problems in its running.

4. System Architecture