# 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:

- Train the network
- Run the network
- Read RAM addresses
- Get inputs
- Record exemplars
- Set controller values
- Display user interface
- Allow user to modify config settings
- Graphical display of network operations

The following describes each functional requirement.

Each Requirement is specified in the following format:

## Name

### *Description*

Provides a description for the Requirement.

### *Criticality*

Describes how essential 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.

## Training the network

### *Description*

Training the network involves the process of forward propagating exemplar inputs through the network and then using the Exemplar output for the input set to back propagate the error through and adjust the weights for each synapse.

### Criticality

This function is as critical a component 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 Function depends on the function record exemplars being fulfilled as it is the exemplars generated from this that are used to train the network

## Run the network

### *Description*

This function involves being able to read new inputs from the game and pass them through the network. Once these have been passed through and the network has generated outputs these outputs are mapped to key presses which are then executed resulting in mario's movement throughout the level.

### Criticality

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 function.

### Technical issues

Running the network will require many aspects of the system to come together for it to work these being reading in inputs and encoding them correctly along with mapping the output 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 requirement is dependant with the following:

- Get inputs
- Set controller values

## Read 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 Enemy's X and Y position in the games level.
- The current tiles displayed on the screen in relation to Mario's position.
- 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.

## 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.

### *Criticality*

This Requirement is a again highly critical to the system as is a requirement 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

## Record exemplars.

### *Description*

This function is to record exemplars from a users gameplay. This function works by allowing the user play through a level in super Mario for n amount of frames. The number of frames can be altered by the user in the settings config. As the user plays through the level the inputs from each frame are written to a DAT file along with the corresponding controller presses mapped as outputs. Each DAT file should have a unique name so that users can chose between different Exemplar files when training the network.

### *Criticality*

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

### *Technical issues*

Technical issues for this requirement involve ensuring that exemplar inputs are correctly mapped to the corresponding exemplar output as if the inputs are not mapped correctly to he outputs the network will adapt its self incorrectly 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 function does not have any dependencies on other system functions.

## Set Controller Values

### Description

Outputs from the network are to be translated into key presses on the NES controller 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 receive 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 function is dependant on the run network function as the outputs generated from this function are what translates into key presses.

## 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. These functions being train the network, record exemplars and run the network while also allowing users to access the function of modifying Config settings.

### Criticality

As this function 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 primarily 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 elements 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 modify config settings.

### 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 function 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.

## Graphical display of network operations

### Description

As the network is running a graphical display of the networks operations is to be shown to the user this display will show information on the current inputs being passed along with how strong each of the connections from the inputs to the hidden layer on this being how strong the weight is for that connection. It finally will show on the output layer what nodes have activated. This is to give users a better understanding of what the system is doing and how it is deciding what to do.

### Criticality

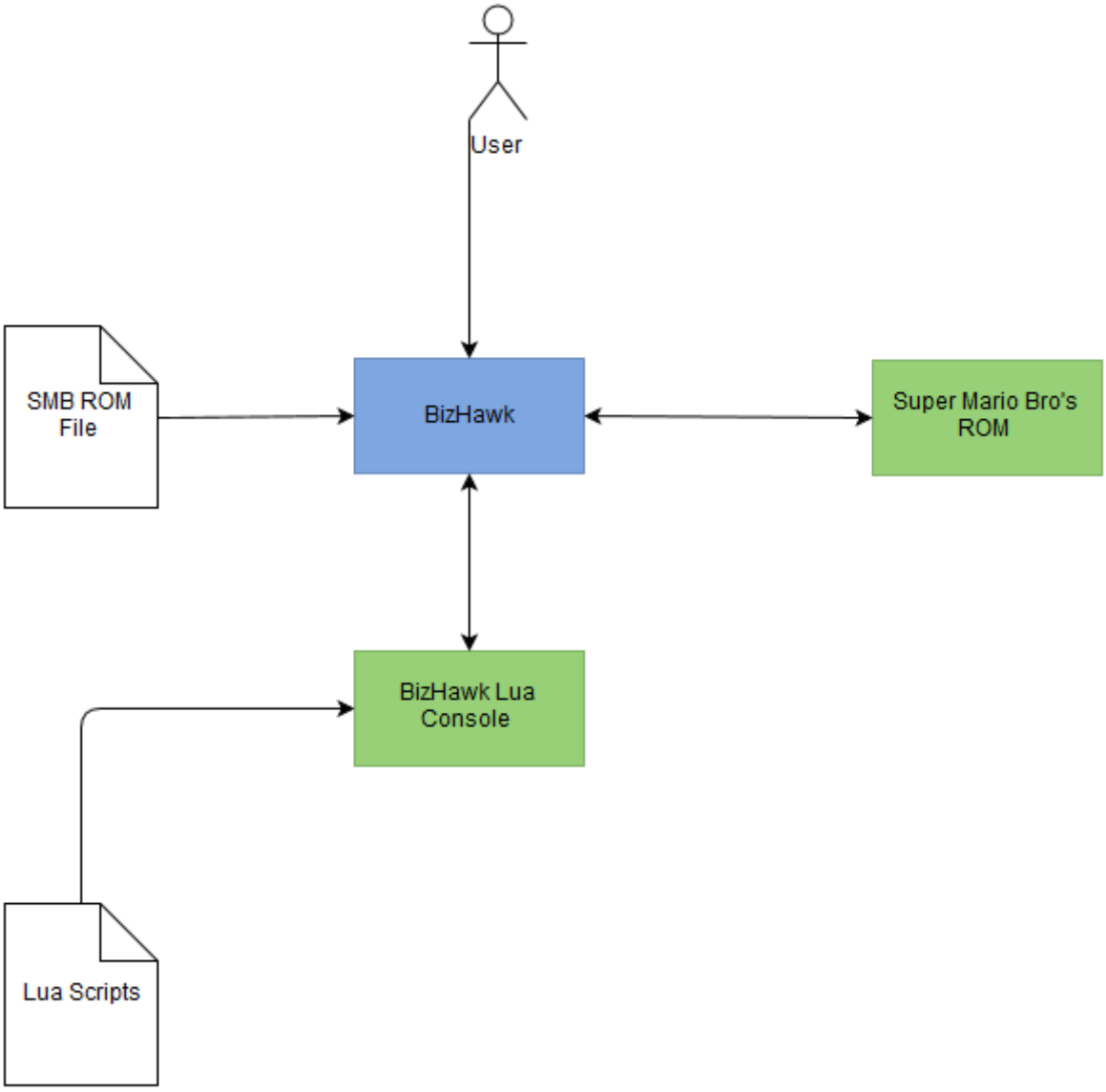
This function is not highly critical to the overall system however it will provide the user with strong insight into the networks operations along with being useful in debugging of the networks performance.

### Technical issues

The main technical issue with this function is on performance as if to much information is written to the BizHawk emulators screen it can have a detrimental effect on the games run time performance. Another issue is on how to display the data as if the network consists of say 100 neurons the problem is now on fitting all that data to the screen in such a way as it is not over whelming to the user.

# 4. System Architecture

The following section describes the systems architecture by describing a high-level overview of the system. Overall the system is not very architectural complex bellow is a diagram depicting the systems architecture along with a description describing the distribution of components.

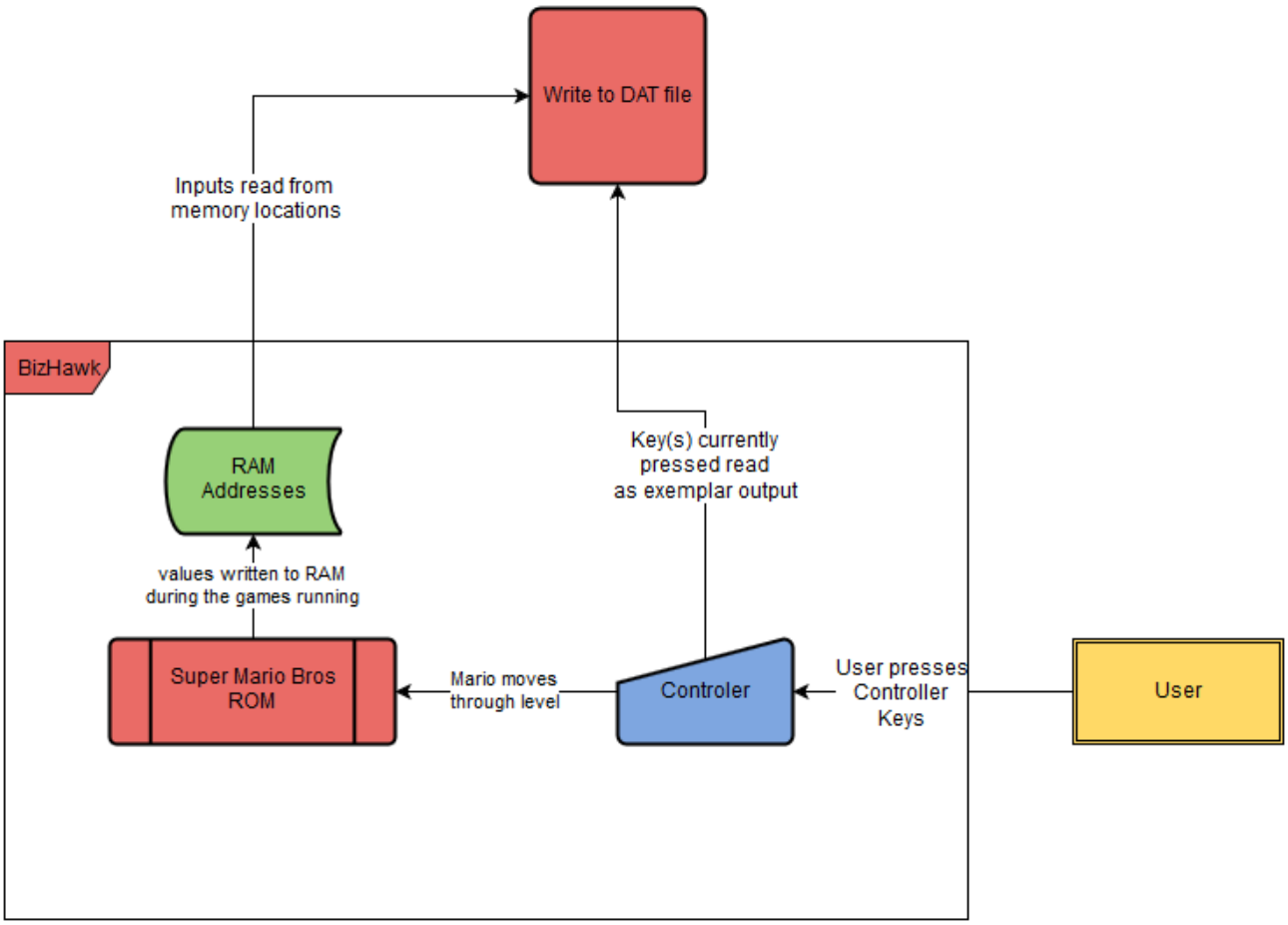



This diagrams shows there are five distinct components to the architecture; The BizHawk emulator, The Super Mario Bros. ROM File, the loaded Super Mario Bros. ROM, BizHawks Lua Console and finally the Lua Scripts which are the primary source of the system functionality.

# 5. High-level Design

The following section details and shows the high level design of each of the major systems processes

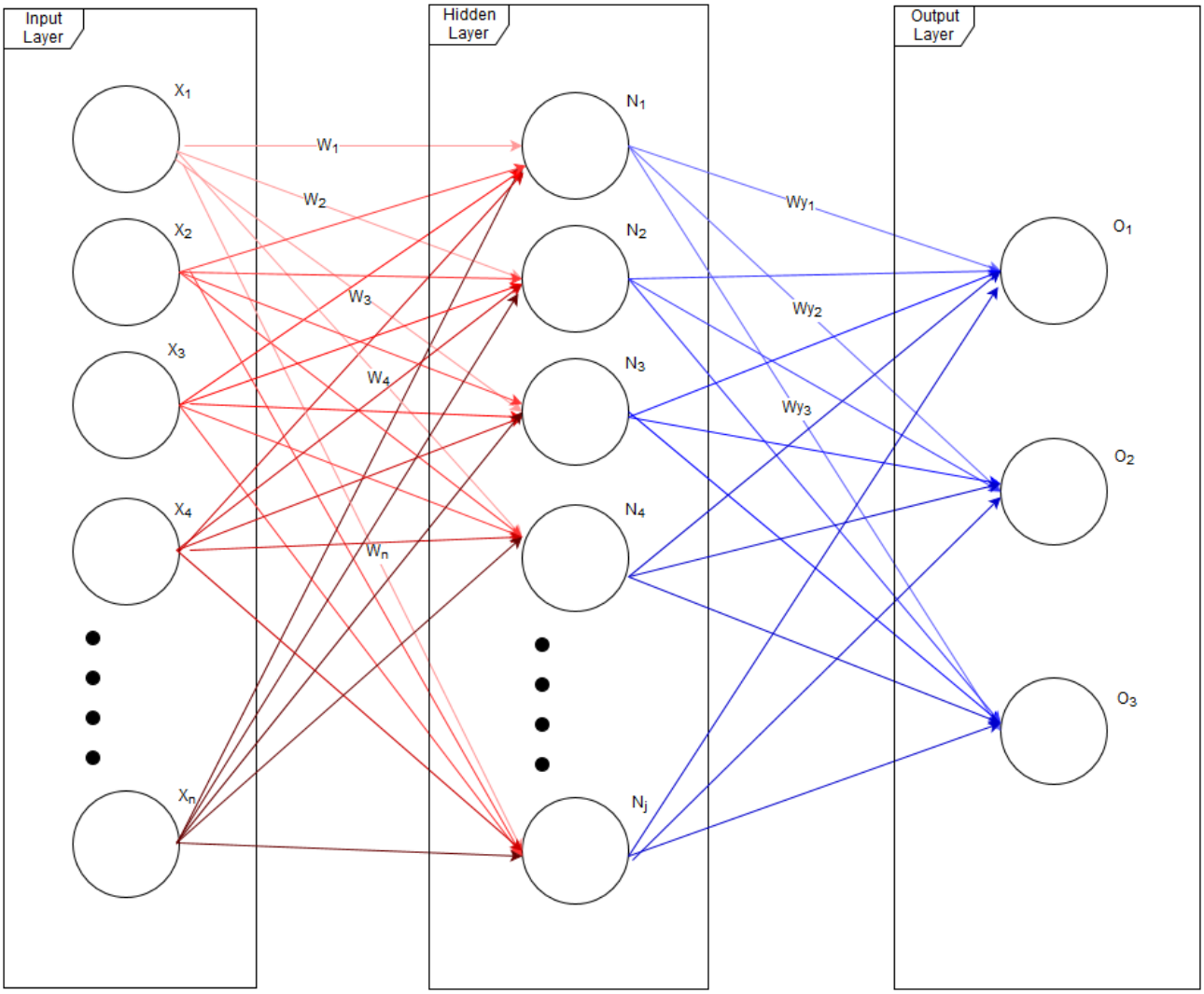
## 5.1 Record Exemplars




The following Data Flow Diagram(DFD) depicts how data is passed through the system when recording exemplars. From this diagram we can see that there are 5 major components to this process the first being the user who is simply playing the game as normal. Next is the input in the form of the controller where once the user presses a key its value is translated into a movement done by the Super Mario sprite in the game. The main predefined process is the Super Mario's ROM / game. During the games run time values in memory are constantly changing to reflect changes in the game, These can include either a enemy has been loaded onto the screen, what tiles to display or simply the players x and y position in the level. These values are updated at each frame render. For each frame the inputs are read from memory along with the controllers key values and are passed to the write process which simply writes the data to a DAT file in a specified format. The components 'Controller', 'Super Mario Bros. ROM' and 'RAM Addresses' are all components housed in the bizhawk emulator as identified by the frame surrounding them. The DFD ends when the number of allocated frames has been meet and the record Exemplar process finishes.

## 5.2 Neural Network structure

To provide a better under standing of the remaining processes the following depicts the structure of the type of neural network to be used and how data is passed through.




The above shows the workings of a Feed Forward neural network (FFN) more specifically a fully connected FFN, what this means is each node in each layer is connected via a synapse(The red and blue lines) where each synapse has a value attributed to them known as a synaptic weight which determines the strength of the connection and data can only pass in a forward direction.

In this diagram the network consists of n inputs and j neurons, the network then gives 3 outputs after passing data through it. Firstly data is passed in via the input layer each node corresponds to 1 input. The value in each of the node's is computed by the sum of the all inputs to the node times their synaptic weight.

$$N = \sum_{1}^{n} X_n \times W_n$$

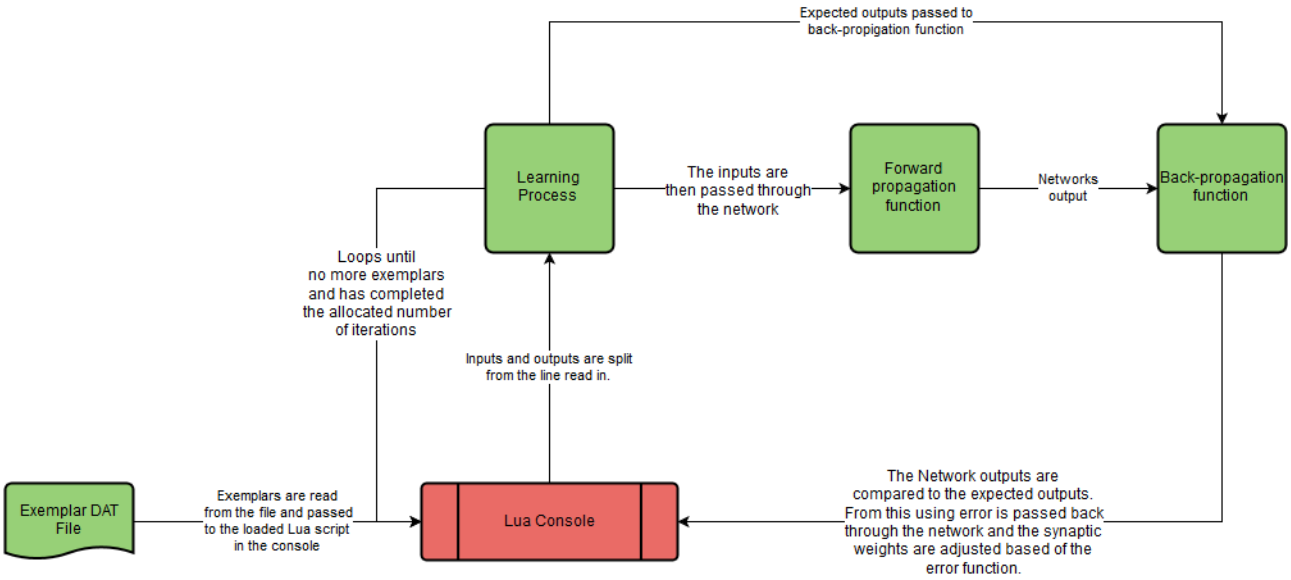
This value is then passed through a activation function to determine either the node fires or not. This activation function can be many different functions the most common are a step function, which corresponds to if the value computed in the node is greater then or equal 0 output a 1 or if the value is less then 0 output a 0. the other most common is a sigmoid function which is used in applications involving back-propagation and is the activation function our network uses.

$$sig(N) = \frac{1}{1 + e^{N}}$$

The above calculations are done for each layer and nodes in the network.

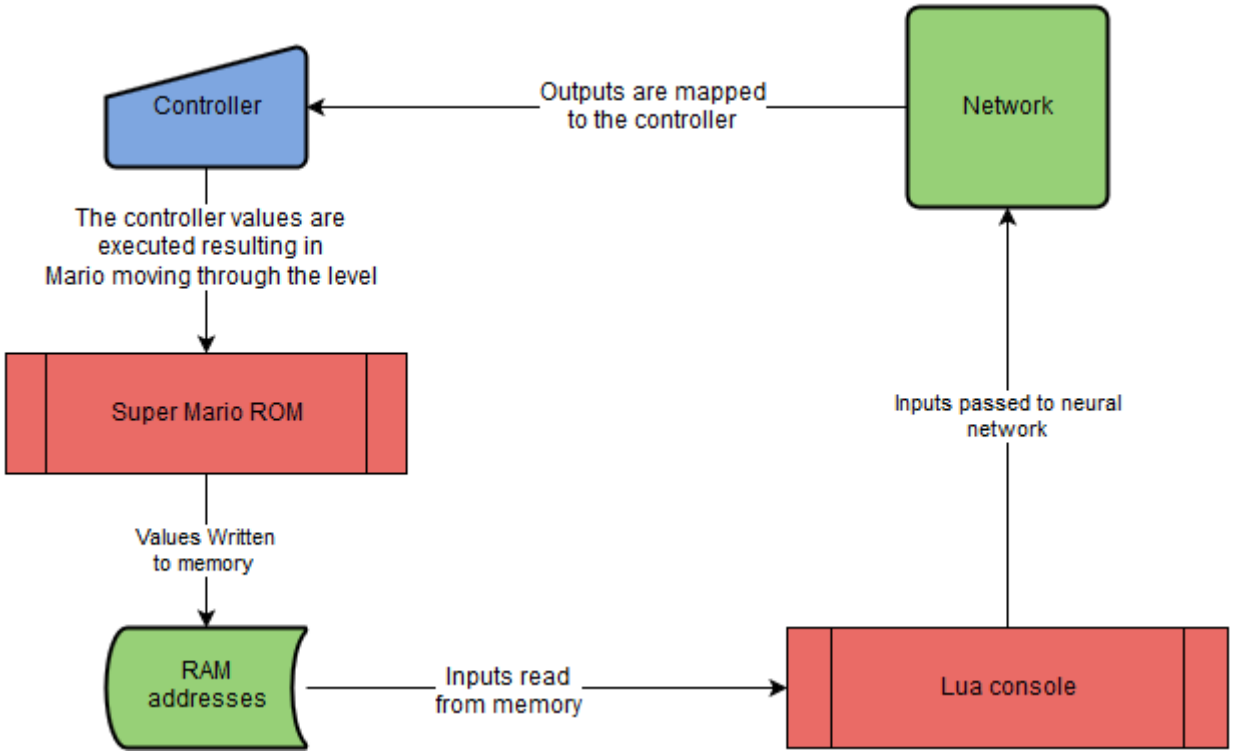
In our neural network application the values calculated in the output layer are translated into key presses so we can say using the above example if the output player were mapped to the keys A,B,C respectively if the value at node $O_1$ fired (its activation function outputs 1) and the other 2 didn't then our network's output is to press key A and not to press B and C.

## 5.3 Training the network




The following DFD describes how the process of training the network operates. It consists of 5 components, The first component is the DAT file that contains the exemplars. The Exemplars are read by the system via the BizHawks Lua console which has the system scripts loaded into it. Each line of the exemplar file is split and the inputs are passed through the network via forward propagation as described in 5.2. The outputs generated are then compared with the expected outputs as in the exemplar file and the back-propagation algorithm calculates the error rate and this value is passed back though the network to modify the weights so that on the next run the output of the same exemplar will result in a value equal to or closer to the expected. This processes continues over a set number of iterations and passes each exemplar through during all iterations.

## 5.4 Running the network




This DFD depicts the process of running the network and as having the network play the game. The process begins by reading the inputs in memory and then forward propagating them through the network. The resulting outputs are then mapped to their controller values which are then executed resulting in Mario moving through the level. As Mario moves the values in memory are updated and as such the process continues as above until a interrupt to stop is called.

## 6. Preliminary schedule

This section describes the schedule in which research and development(R&D) will take place for the system and is accompanied by a Gantt chart to visualise the schedule. R&D for the system has begun since the initial proposal in week 1 of the college year this is shown as such in the below Gantt chart.

The schedule is broken into 4 main phases:

- Research design phase
- Requirement design phase
- Development phase
- Network training phase

### *Research design phase*

This phase involves initial design of the project the primary task in this phase is to design the networks inputs it also serves to allow time to get familiar with the Lua programming language as it is a language I have not learned yet.

### *Requirement design phase*

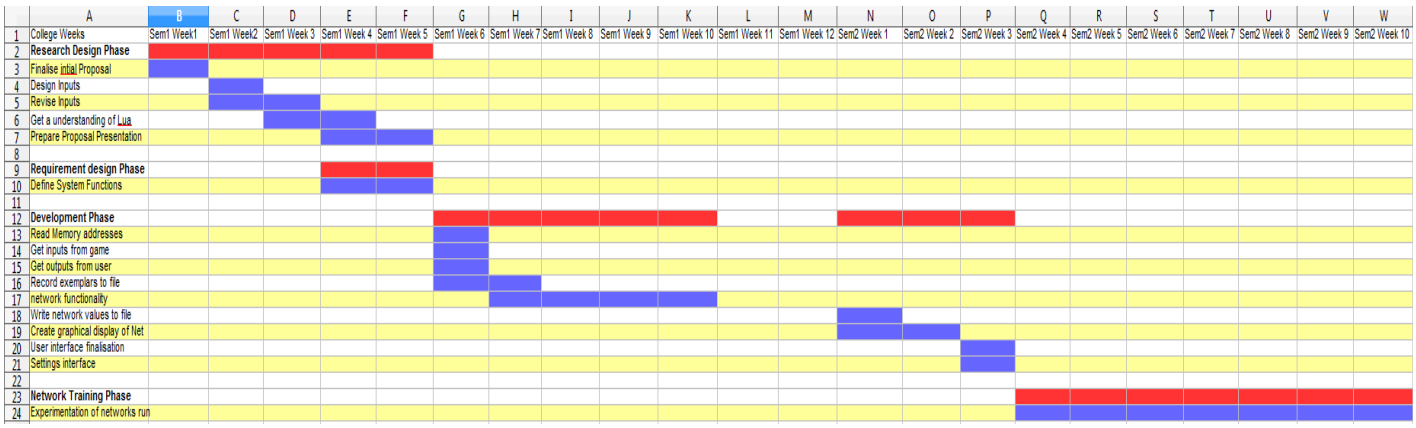
This phase involves working out the exact system requirements and functionality.

### Development phase

This phase involves the development of the functionality identified during the Requirement design phase along side the functionality in this document. As in the Gantt chart during the weeks of 11 and 12 for both semesters development is to slow as my time will be primarily focused on end of semester assignments. Development will still continue but not at its normal rate as I need to manage my time across multiple course modules.

### Network Training phase

This phase is the more crucial area of development which is why a large amount of time has been invested in it. This project involves a lot of experimentation with how the network is to operate to ensure it can successfully learn to play the game. It should be noted this project is not to design a system with the capability's of beating the game, This being it comes up with a unbeatable strategy and can play the entire game from start to finish flawlessly. Rather it is to have a strong attempt at such while learning the process of implementing neural network applications. The end goal I hope to achieve is that at the end of this phase the network has learnt how to complete a level where the unseen instances are greater then those of the seen and be able to do so faster then a average human can play showing the network has good generalisation capabilities!



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | College Weeks | Sem1 Week1 | Sem1 Week2 | Sem1 Week 3 | Sem1 Week 4 | Sem1 Week 5 | Sem1 Week 6 | Sem1 Week 7 | Sem1 Week 8 | Sem1 Week 9 | Sem1 Week 10 | Sem1 Week 11 | Sem1 Week 12 | Sem2 Week 1 | Sem2 Week 2 | Sem2 Week 3 | Sem2 Week 4 | Sem2 Week 5 | Sem2 Week 6 | Sem2 Week 7 | Sem2 Week 8 | Sem2 Week 9 | Sem2 Week 10 |
| 2 | **Research Design Phase** | | | | | | | | | | | | | | | | | | | | | | |
| 3 | Finalise intial Proposal | | | | | | | | | | | | | | | | | | | | | | |
| 4 | Design Inputs | | | | | | | | | | | | | | | | | | | | | | |
| 5 | Revise Inputs | | | | | | | | | | | | | | | | | | | | | | |
| 6 | Get a understanding of Lua | | | | | | | | | | | | | | | | | | | | | | |
| 7 | Prepare Proposal Presentation | | | | | | | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | **Requirement design Phase** | | | | | | | | | | | | | | | | | | | | | | |
| 10 | Define System Functions | | | | | | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | **Development Phase** | | | | | | | | | | | | | | | | | | | | | | |
| 13 | Read Memory addresses | | | | | | | | | | | | | | | | | | | | | | |
| 14 | Get inputs from game | | | | | | | | | | | | | | | | | | | | | | |
| 15 | Get outputs from user | | | | | | | | | | | | | | | | | | | | | | |
| 16 | Record exemplars to file | | | | | | | | | | | | | | | | | | | | | | |
| 17 | network functionality | | | | | | | | | | | | | | | | | | | | | | |
| 18 | Write network values to file | | | | | | | | | | | | | | | | | | | | | | |
| 19 | Create graphical display of Net | | | | | | | | | | | | | | | | | | | | | | |
| 20 | User interface finalisation | | | | | | | | | | | | | | | | | | | | | | |
| 21 | Settings interface | | | | | | | | | | | | | | | | | | | | | | |
| 22 | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | **Network Training Phase** | | | | | | | | | | | | | | | | | | | | | | |
| 24 | Experimentation of networks run | | | | | | | | | | | | | | | | | | | | | | |

*Gantt chart depicting each phase and their tasks.*[5]

# 7. Appendices

[1] https://en.wikipedia.org/wiki/Deep_Blue_%28chess_computer%29

[2] https://en.wikipedia.org/wiki/Deep_learning

[3] https://en.wikipedia.org/wiki/Activation_function

[4] http://datacrystal.romhacking.net/wiki/Super_Mario_Bros.:RAM_map

[5]