University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

# Solving Super Mario Bros. Using Behavioural-Based Reinforcement Learning

**Nathan Hannan-McCulloch**
April 4, 2022

# Abstract

Traditional Reinforcement Learning techniques seek to train an agent with an end-to-end approach, learning how to implement behaviours from scratch. This project seeks to explore an alternative method of training agents, instead using supervised training to learn individual behaviours that are then choreographed to accomplish a task, in this case solving a level of Super Mario Bros. The approach was found to be viable, though more time consuming and memory intensive than the standard approach whilst producing a weaker final result.

# Education Use Consent

# Contents

# 1 Introduction

## 1.1 Motivation

In 2009, a series of competitions launched entitled the Mario AI Championship (Karakovskiy and Togelius, 2012). The competition would recur annually for the next three years, before morphing into the more general Platformer AI Competition. The popularity of the game drew great publicity and interest in the premise of using artificial intelligence to solve the iconic game. In its initial form, the competition challenged contestants to submit controllers that would attempt to complete procedurally generated levels of Infinite Mario Bros., an open source game created based off the iconic 1985 classic, Super Mario Bros. Across the years of the competition, several attempts were made to solve the levels using learning based approaches, but they always proved to perform poorer than search-in-state-space techniques.

Using learning techniques such as neural networks or evolutionary computation is relatively common in creating agents to solve tasks in a video game environment. However, there are drawbacks to using these approaches. Training can take take an arduous amount of time, and the exploration required will explore many ineffective behaviours to accomplish the goal. This project seeks to explore whether implementing a behavioural-based approach to deep learning, common in robotics, could lead to a better solution. Using behavioural-based reinforcement learning can cut down on this wasted time and produce a more effective agent in a shorter training time. Taking a behavioural-based approach can also make the agent act in a manner closer to the expectations of a human, which could be beneficial if an agent is intended to act in an immersive or recognisable manner.

## 1.2 Aims and Goals

This project aims to explore how behaviour-based reinforcement learning can be implemented in an agent attempting to solve levels of Super Mario Bros. and how that agent performs in contrast to an agent trained in an end-to-end manner using a traditional reinforcement learning approach. In order to achieve this, several steps must be accomplished.

The agent must learn low-level behaviours in isolation. These behaviours should allow the agent to more effectively complete the level, whilst being composed by a high level-choreographer to be implemented effectively.

The high-level choreographer must be able to take the input of a level as it is being dynamically played, and be able to effective complete a level through implementation of the low-level behaviours.

A reasonable goal for the effectiveness of the agent would be for it to be capable of completing one level of the game. Ideally, the agent would be able to learn the isolated behaviours and train the choreographer faster than another agent can learn to solve the level using an end-to-end approach.

# 2 Background

In attempting to implement behavioural-based reinforcement learning to solve Super Mario Bros., two primary resources needed to be looked at. Behaviour-based reinforcement learning, and its implementation in other agents, and algorithms used in training agents to solve Super Mario Bros.

## 2.1 Behavioural-Based Reinforcement Learning

Behavioural-based reinforcement learning seeks to learn how to solve a task by first learning a set of low-level behaviours, then training a high-level choreographer how best to implement those behaviours. These behaviours are trained incrementally and individually to ensure that they are each effective in a well functioning environment.

### 2.1.1 Subsumption Architecture

The idea of behavioural-based reinforcement learning largely originates from Subsumption Architecture. Subsumption Architecture was proposed by Rodney Brooks, as presented in Brooks 1986, in an attempt to decompose the problem into a set of task-focused behaviours, which would each react to the input received by the robot, and each behaviour would then attempt to dictate the behaviour of the output. This differed from the conventional approach at the time, which saw the problem decomposed into functional units, where each layer would feed directly into the next. Brooks instead layered the behaviours to account for different levels of complexity in the actions of the robot. The foundational layer would attempt to accomplish simpler tasks, whilst each increasing layer would seek to achieve an additional degree of difficulty. The output of the foundational layer could be altered by the next layer to attempt to better achieve the second degree of complexity, which could in turn be altered by the next layer, and so on.

This subsumption architecture would give way to a new approach to robotics learning. Instead of tasks being required to be learned in an end-to-end manner, they instead could be accomplished by allowing behaviours to develop independently of each other and then combined to achieve a larger goal. This approach was popular largely in the field of robotics, but the same principles could be a applied to most any agent attempting to accomplish a task.

### 2.1.2 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning is a recently emerging field, that takes much inspiration from Brooks's subsumption architecture. Hierarchical reinforcement learning sees a traditional reinforcement learning approach broken down into low-level functions that learn a policy set to accomplish sub-tasks, and a high-level function that learns a policy set to accommodate for the larger task. This approach can combat the adverse effect of an environment with sparse rewards, where an end-to-end approach may struggle to learn robust policies, the low-level functions can instead learn to draw rewards from the immediate actions that are undertaken, whilst the high-level function accounts for the more sparse rewards of the long term goal.

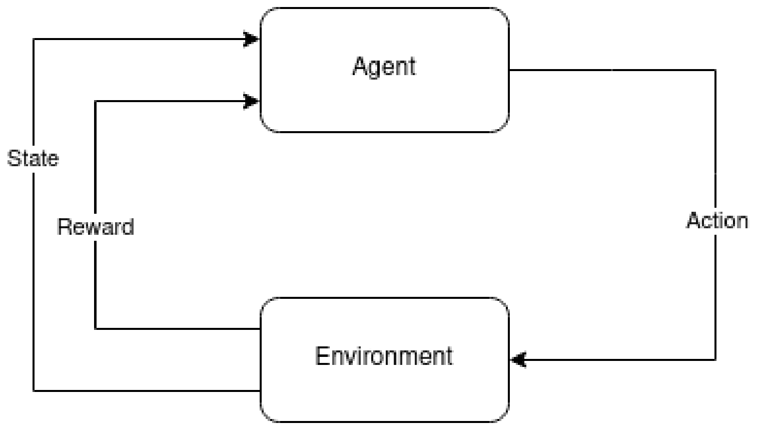


*Figure 2.1: A diagram of the learning cycle for reinforcement learning, where an agent takes an action based on a state, which is then input to an environment. The action is then evaluated by the environment outputting a reward value for the action taken, and a new state. These are then given back to the agent which will inform its next action.*

This approach has proven to be quite effective and similar to the approach used when implementing behaviour-based reinforcement learning. The primary difference between the two methods is that a hierarchical approach sees the low-level functions learnt simultaneously as the agent explores. A behaviour-based approach instead attempts to train each behaviour independently and in isolation.

## 2.2 Reinforcement Learning Algorithms

Reinforcement learning is an approach to training agents that focuses on action rewards. An agent will be capable of taking as input a state that is reflective of the current status of the environment the agent is in. The agent will then evaluate that state against a learned policy to make a judgement as to the best action to take under current circumstances. The action will then be carried out in the environment, leading to an updated state. The environment will also output a reward based on the effectiveness of the action in the environment. This reward will be appropriately scaled to encourage the agent to achieve the desired goals. In the case of Super Mario Bros., a positive reward would be gained for making progress towards the end of the level, a slightly greater reward for actions such as defeating an enemy or collecting a coin, and the greatest reward for completing the level. The reward would then be much lower if the actions are counter-productive, such as taking damage or dying. The effect of the chosen action based on the given state and the reward it produced would then be evaluated by the agent to better learn what actions are effective. Based on this updated policy, the agent will take the new state and evaluate the next action to take.

### 2.2.1 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) was proposed as an alternative to other leading reinforcement learning algorithms such as deep-Q learning and trust region / natural policy gradient methods (Schulman et al., 2017). PPO sought to provide a simpler algorithm that maintained good data efficiency and provided reliable performance.

PPO shares certain characteristics with some other popular policy optimization algorithms. It uses an actor/critic approach, where the actor suggests a policy from which the probability of actions is drawn, whilst the critic informs how positively or negatively the action affected the environment. PPO employs an "on-policy" approach, it seeks to train by taking a batch of values from an iteration and using them to update the policy then disregards the information it held before, meaning it only ever considers the current policy and training data, not previous versions. This runs the risk of leading the agent down the wrong, path however PPO seeks to avoid this by limiting how much a new policy can be updated from the current one.

# 3 | Design

For this project, a code base was provided by Gerardo Aragon-Camarasa, with much of the early design work already done. This covered the code for feature extraction and behaviour training. In addition, training data was supplied for the training of low-level behaviours.

## 3.1 Data Collection

To train the low-level behaviours, one of two approaches could have been taken. Either the behaviours could be given supervised training, where they are supplied with a large amount of data that would see them given training to try and learn how best to mimic the behaviours shown, or they could be given unsupervised training which would require some kind of reinforcement learning approach. To implement the reinforcement learning would have proven difficult. Each behaviour needed to be trained independently and incrementally, so for one to trained it would either have to be done in an environment where the other behaviours were not coming into consideration, or were already trained to behave at an optimum level.

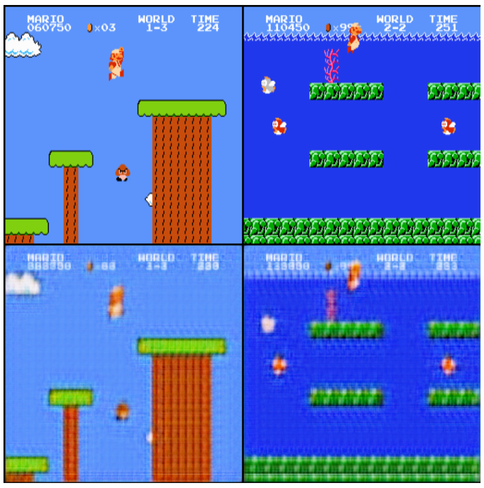
Since it would not be feasible to create a Super Mario Bros. environment wherein behaviours could be tested independently without creating a unique environment, it was instead more practical to provide expert data. This data was recorded using an emulator to complete play throughs of the game. Each frame was recorded, along with a corresponding set of data about the input of each frame, particularly the corresponding button presses.

## 3.2 Feature Extraction

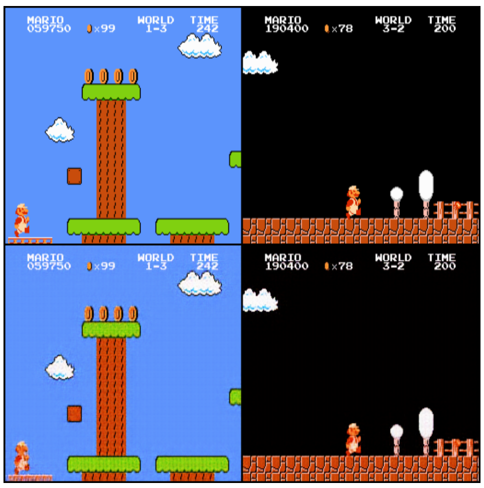
To train the behaviours, rather than providing a frame as input, it was instead decided to pass the images through a neural network that would extract features to act as the visual backbone for training. Feature extraction can be beneficial as it reduces the amount of redundant data being used to train the behaviours. By passing the image through a neural network then comparing it to the truth value of the original image, a loss can be derived from the Mean Square Error (MSE) between the two images. Training the feature extraction network on this data set would then lead to a compressed image with a minimal error between the that and the original, as in Figure 3.1. This eliminates some of the redundant data that from the original frames, which should produce a more effective result for the behaviour training.

## 3.3 Low-Level Behaviours

The first step in training the behaviours was to consider what those behaviours were. Consider the task of walking through a building, and what behaviours may be utilised in doing so. A person would obviously have to walk, so that would be one. Opening a door may be a behaviour, avoiding bumping into people and objects another, or climbing stairs perhaps. To apply this premise to playing the original Super Mario Bros. is easy enough in brief, but more complex in practicality. The game is a 2D platformer, so the behaviours seem simple. Move towards the end


MARIO 060750 x03 WORLD 1-3 TIME 224
MARIO 110450 x98 WORLD 2-2 TIME 251

*(a) Feature extraction after the first epoch.*


MARIO 059750 x99 WORLD 1-3 TIME 242
MARIO 190400 x78 WORLD 3-2 TIME 200

*(b) Feature extraction after the last epoch.*

***Figure 3.1:*** *Compression caused by the feature extraction. The top row on both shows the original images, the bottom row shows the compressed image. Though both images have undergone the same amount of compression, by the final epoch the image is noticeably closer to the originals.*

of the level, jump over threats and obstacles, hit blocks along the way. However, these behaviours are far from exhaustive. Different behaviours may need to be implemented for all different kinds of enemies that require different approaches. Some obstacles may need to be navigated rather than simply jumped over or ascended. Learning these specific behaviours would also be difficult as it would require a great deal of manually scrubbing the training data to sort and categorise it, and it may not always be clear which of these defined behaviours should be considered.

A simpler approach to the problem is to instead consider how the agent is controlled, and what data we have recorded. The agent is controlled through a set of 6 possible button pushes, up, down, left, right, A and B. These at the offset may not seem to be behaviours in the same manner described in the previous manner. For the human, the only behaviour they seem to need to learn is how to push a button. For the virtual agent, there isn't even a button to press. But consider how the behaviours described before for Mario can be decomposed. To approach the end of the level, the agent will typically have to be pushing the right button. To avoid from running into an enemy, they might have to stop pressing the right button. Thus by learning when and when not to press the right button, an agent can learn a decomposition of many conceivable behaviours. If all button presses were considered the same way, then the entire gameplay could be decomposed into six distinct behaviours, corresponding to each of the possible button inputs. And since each button press corresponding to each given frame was recorded in the data collection, this makes for easier supervised training, by simply looking at each frame and determining the likelihood of the given button being pressed for that frame.

### 3.3.1 Visual Motion

In addition to training the behaviour control for each button, training for the visual motion of the past few frames takes place for each button. Taking just the features as input for the behaviour model is an acceptable approach, but the agent can make a more informed call if it considers not only the frame from which it is making its move, but the motion it took to reach that point. This means looking at a range of the past few frames with feature extraction performed, and then passing that through a neural network to perform another type of feature extraction, to find the relevant features in the motion over the past few frames most relevant to implementing a behaviour. This visual motion can then be passed to a neural network representing the behaviour.

## 3.4 High-Level Choreographer

The purpose of the high-level choreographer is to determine when low-level behaviours should and should not be implemented. To train the choreographer however, a different approach must be taken from the other neural networks. The choreographer cannot be trained from the expert data that was supplied to it, else it would simply learn to replicate that method of solving the game under those same conditions, and each behaviour would be made in effect pointless as the choreographer would train simply to implement the behaviours from the same truth values the behaviours were trained on. Instead that means the behaviour must be trained in an unsupervised environment, where it can explore when and which behaviours it should implement. This means that a reinforcement learning algorithm must be implemented to select which behaviour leads to the best rewards from a given state.

Many algorithms would have been acceptable choices for this task. Asynchronous Advantage Actor Critic (A3C) provides a strong result. Deep-Q Learning (DQL) is another popular choice, but the decision was made to implement the program in PPO instead. Demonstrations have shown PPO to provide more consistent successes than A3C, and it is easier to implement and requires less parameter tuning than DQL.

Implementing a PPO algorithm for solving Super Mario Bros. differs very little for implementing the same algorithm for the high-level choreographer. The only core difference is that instead of choosing one of the possible inputs for the environment, it must instead choose which behaviours to implement. However, seeing as each behaviour correlates to pressing only one button, and it would be very difficult to solve Super Mario Bros. whilst only pressing one button at any given moment, it must instead choose a possible combination of behaviours to implement. In practice, this means that there are no less options for the high-level choreographer to choose from than their would be for a standard agent, just that the behaviours can intelligently implement themselves.

# 4 Implementation

## 4.1 Feature Extraction

For carrying out the project, as previously stated, a code base was provided covering feature extraction and low-level behaviour training, as was training data. The Training data was obtained through the use of an open source Nintendo emulator called FCEUX. FCEUX is capable of running Lua scripts, which allow for data recording to take place. The data was recorded at each frame, with the frame saved as a JPEG image, and the relevant data of control input being stored in a JSON file.

For feature extraction to be carried out, a neural network was constructed that would compress the image to a set of features by passing it through a set of convolutional layers of neurons. The features would then be passed back through a set of deconvolutional layers of neurons to restore the image, as seen in Figure 4.1. With the image encoded and then decoded again, the Mean Squared Error (MSE) between the two images was taken to calculate the loss. MSE is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{4.1}$$

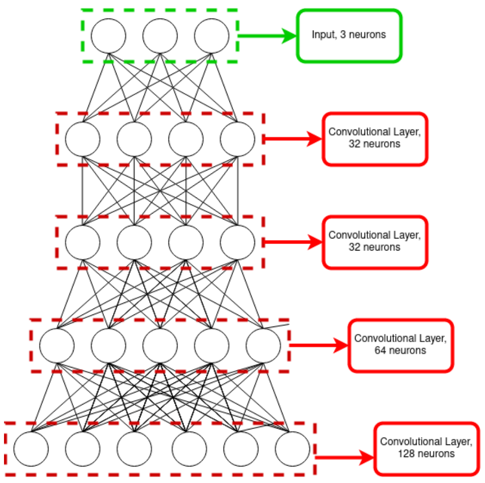
where N is the number of samples taken, y is a sample from the output and  is the corresponding sample from the truth values.

From this calculated loss, the networks can then do a backwards pass and optimize the weights for both the encoding and decoding neurons. The optimization algorithm chosen was Adam, which is memory efficient and straightforward to implement.

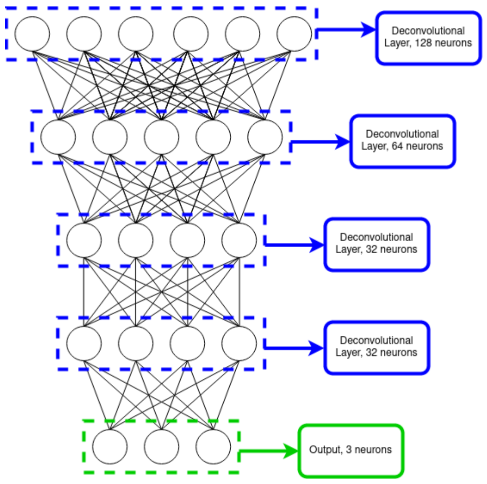
When training, the data was split 80/20 into training and validation sets respectively. At the end of each training epoch, the model was tested against the validation set to see if the model at the end of the epoch had been improved. If so, then the weights of the model were saved for access later.

The code was implemented in python using the packages Torch and CUDA. Torch is a package available for python providing a lot of functionality required for the project. Torch allows variables to be stored as Tensors, which allows for greatly increased computation power as they can be handled by a GPU instead of the CPU. Torch also provides the neural networks and optimizers used to build models. CUDA is a platform created by Nvidia that allows GPUs to be used for processing. Implementing this allowed for faster training. The training was carried out by four parallel workers. Though computing on a GPU provides a much faster process, the memory and speed available is dependent on the machine being used, if it even has a GPU. In this case, the limited memory proved to be something of a recurring problem. When training the feature extraction models, the batch size had to be lowered to 64 for the training set and 16 for the validation set. This did not interfere with the effectiveness of the model, just produced a slower result.

As can be observed in Figure 4.2, the validation set and the training set are largely in line, with both showing a sharp decrease in loss. This shows that the feature extraction training was largely


Input, 3 neurons
Convolutional Layer, 32 neurons
Convolutional Layer, 32 neurons
Convolutional Layer, 64 neurons
Convolutional Layer, 128 neurons

*(a) Neural network for encoding features from an image.*


Deconvolutional Layer, 128 neurons
Deconvolutional Layer, 64 neurons
Deconvolutional Layer, 32 neurons
Deconvolutional Layer, 32 neurons
Output, 3 neurons

*(b) Neural network for decoding an image from features.*

***Figure 4.1:*** *(a) shows the neural network used to compress images into a set of features, (b) shows the same design reversed so that the features may be restored into the image, so that the effects of compression can be seen.*

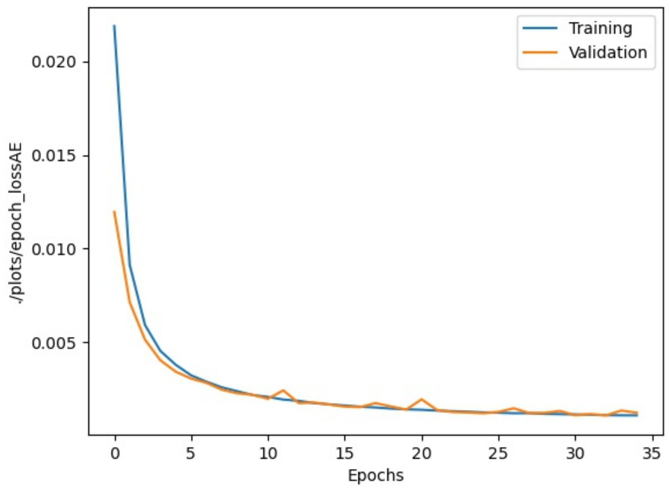


***Figure 4.2:*** *The total loss per iteration of the training set against the total loss per iteration of the validation set for feature extraction. It can be seen that the loss starts high but reduces quickly and starts to plateau around the thirtieth iteration.*

effective as it preforms exactly as desired against the validation set. This can also be seen by comparing the images having gone through encoding and decoding against the originals, as in 3.1. By comparing the reconstruction done on the first epoch against the last, it can be seen that there is a much greater consistency between the reconstructed image and the original after training has completed.

## 4.2 Low-Level Behaviours

For the low-level behaviour training, two models needed to be trained for each behaviour. One representing the visual motion across a set of frames, and one representing the behavioural activation. The model representing the visual motion needed to take a set of features as input and combine them into a reduced input for the behavioural network to examine. This meant that when creating the model, the number of frames that would be taken into consideration had to be declared and that then whenever attempting to extract features for the behavioural activation, a consistent number of frames needed to be used to ensure the weights of the model were still accurate. As seen in Figure 4.3, the model takes the 128 channels produced by the feature extraction as input, multiplied by the number of features which are concatenated together, and then is passed forward to a single 128 channel output. This acts as a feature representation for the frames together.

The feature representation of the motion is then passed directly into the model for the behavioural inhibition. As all the work of extracting features from the frames has already been done, the behavioural neural network simply passes the input through a linear layer of neurons into a single output channel, as seen in Figure 4.4. The desired output of this model is a single value representing the likely outcome of pressing the button at this moment. A high value would suggest that the button should be pressed at this junction, a low value suggests not. To train the model to output an accurate value, we train against the expert data. As the data on button presses were recorded for each corresponding frame in the expert data, it is possible to look at what buttons were pressed during the corresponding frames that are currently being analysed. The ratio of times the button that is being trained for was pressed across those frames is taken as the true probability of pressing that button. The loss function used in training behaviour activation and visual motion is MSE, and the optimizer algorithm used is again Adam.

Six behaviours were defined earlier, left, right, up, down, A and B. The training for each behaviour must be carried out independently from each other, and since it would be too taxing on the memory available to attempt to train multiple behaviours simultaneously, they are trained sequentially. When training and performing validation, feature extraction needs to be performed for every iteration, which can be a further strain on memory. The frozen weights for performing feature extraction are loaded in after having been trained. The training is performed over an epoch, doing a forward and backward pass each iteration, and then at the end the optimised model is tested against the validation data.

When looking at the loss across the training of the entire behaviour, it can be seen that the behaviours are being trained as desired, as in Figure 4.5 (e). For all behaviours there is a notable decrease in loss across the training as is desired, with some behaviours performing better than others, such as behaviour u as seen in Figure 4.5 (b).

## 4.3 High-Level Choreographer

For the implementation of the high-level choreographer, it was decided to use PPO as the reinforcement learning algorithm. A well-functioning version of this algorithm was implemented to solve Super Mario Bros. on GitHub (Nguyen, 2020), and this was used as a base from which

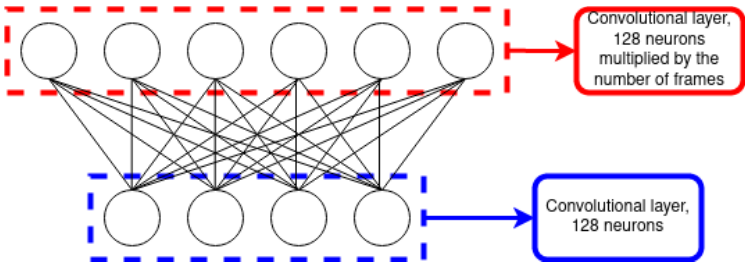


***Figure 4.3:*** *A neural network representing the visual motion across a series of frames. The network is declared with the number of frames that are being processed, and then takes in the feature representation of each of those frames together as input. These frames are then reduced to a single output for the behaviour network to take as input.*

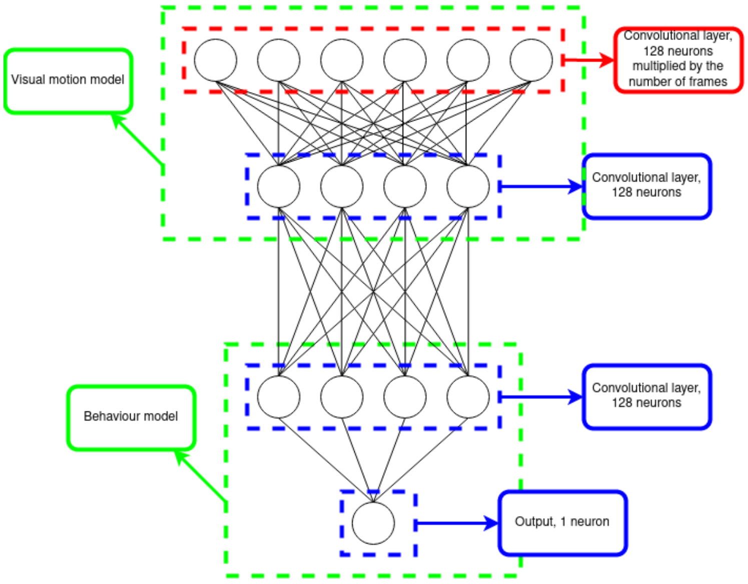
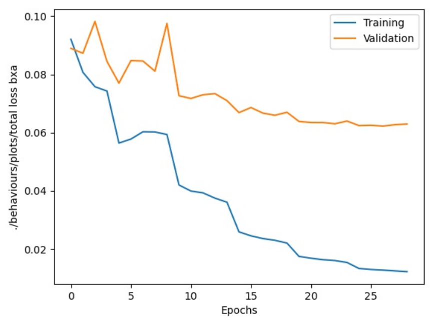
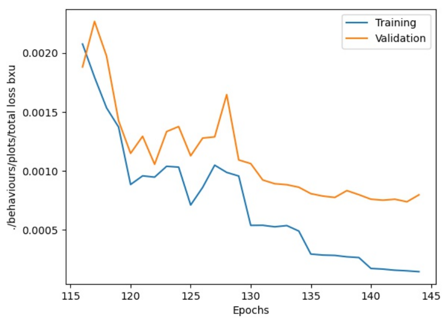


***Figure 4.4:*** *A neural network representing both the neural networks for visual motion and for behaviour implementation. The visual motion takes in a set of features as input and reduces it to a single 128 channel output.*



***(a)** Total loss per iteration of the training set against the total loss per iteration of the validation set for behaviour a training.*



***(b)** Total loss per iteration of the training set against the total loss per iteration of the validation set for behaviour u training.*

***Figure 4.5:** (e) shows the data for training behaviour a, (b) shows the same data for training behaviour u. Note that the Epochs listed on the x-axis were measured from the start of training of all the behaviours, rather than just the relevant behaviour.*

to build the choreographer. The algorithm was altered to implement Gym Retro by Gerardo Aragon-Camarasa.

The environment the agent is operating in is Gym Retro. The environment allows an agent to play a level of Super Mario Bros. on loop, earning pre-defined rewards for certain actions. The agent would net a slight reward for progressing towards the end of the level, a stronger reward for hitting blocks, gaining items, or defeating enemies. The biggest reward comes from completing the level.

The choreographer requires feature extraction and behaviour activation to be carried out at each iteration, so the trained weights from the past two steps must be imported for the choreographer to be trained and operate. When training the choreographer, the possible combinations of input actions must be considered. Certain combinations can immediately be disregarded as redundant, such as left and right, or up and down. This still leaves a large number of possible combinations though. Attempting to train with a higher number of possible inputs requires more time and more exploration on the part of the agent, and though attempts were made to account for the remaining possible combinations of behaviours, no agent was successfully trained. So, to further reduce the possible combinations, behaviours and combinations that are largely redundant in solving the game are also eliminated. The vast majority of Super Mario Bros. consists of jumping with the b button and travelling right. The up and down buttons are rarely if ever used, and moving left is typically counter intuitive. So, the up and down buttons are removed, and any combinations of left with other behaviours is disregarded. This leaves a more simple set of inputs for which an agent can successfully be trained.

With these possible behaviour combinations in mind, the neural network for the choreographer model is created, as seen in Figure 4.6. The choreographer takes a pre-processed frame from the environment as input. It would perhaps be more effective to take the features extracted from the past five frames as input, but it was decided against in an effort to maintain similarity to the original PPO implementation so that the effect of the low-level behaviours is more isolated and easier to evaluate.

To carry out feature extraction at the first frame, the environment must be reset a few times to gather enough states. For the rest of the training process, feature extraction is being carried out each iteration over the last set of frames. Each iteration runs for a defined number of steps, with multiple workers running in parallel. In initial attempts to deploy this algorithm, the agent was

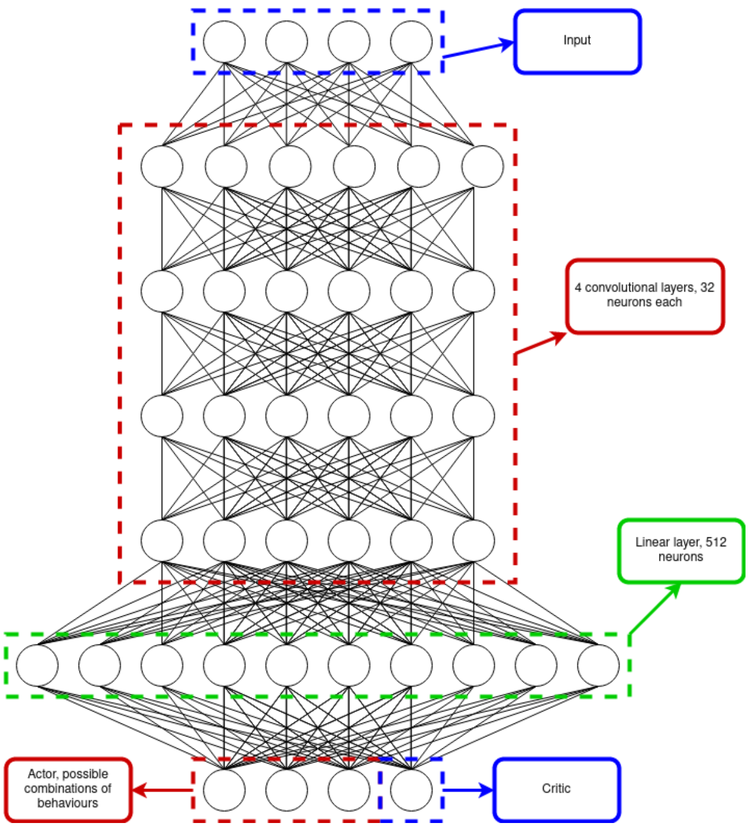

Input
4 convolutional layers, 32 neurons each
Linear layer, 512 neurons
Actor, possible combinations of behaviours
Critic

***Figure 4.6:*** *A neural network representing the high-level choreographer. The input is taken from however many channels are supplied by the input, then put through four convolutional layers, each consisting of 32 nodes. The output is then supplied to the linear layer, before being passed to the output, a critic and an actor, where the actor is composed of neurons each representing possible combinations of behaviours.*

only able to run for a very limited number of steps, about 32, due to all the data being stored as tensors, and all the models for feature extraction, behaviour activation, visual motion, and the choreographer all being stored in the GPU, quickly overwhelming its memory. This resulted in significant problems for the training because the agent would spend entire iterations training without moving from a spot it was currently stuck in. If the agent was split across two parallel workers, it could spend an entire iteration simply in the death animation before the level reset. A few techniques were implemented to lighten the load on memory. For all the models that were not being trained, the memory load could be greatly lessened by setting their parameters to not require gradients. This eliminates a large amount of data that the models were holding for backward passes to be performed, which do not need to occur here as only the choreographer is being trained. The other method of freeing up memory was to ensure that data was not being stored as tensors and in CUDA memory when not necessary. Only doing so when called upon allows for a greater amount of evidence to be accumulated by more iterations. In the end, training was able to successfully occur with 6 parallel workers taking 256 steps each.

Once a set of behaviours has been selected by the choreographer, it is over to the relevant behaviour model to implement it. The features extracted from the past set of frames is fed through the visual motion and behaviour activation models for the each behaviour, resulting in a set of activation probabilities. At the start of training, a control value is set at which, if the activation probability exceeds the value, the action will be taken in the environment. The behaviours that were selected by the choreographer and met the control value will be compared with a dictionary and turned into the corresponding integer for that set of actions for the environment.

As the agent is training, it has a test on the go function that allows for the current best version of the model to attempt to complete the level whilst the workers are exploring. This allows for the current ability of the choreographer to be studied whilst training is taking place. As there is no set limit to the amount of iterations that can occur this is the best method of judging when to finish training the agent.

As the agent is running through an iteration, the choreographer produces two outputs, one from the actor, one from the critic, as seen in Figure 4.6. From the actor, a policy is created to represent the distribution of probability for the behaviours using the Softmax function provided by Torch. This comes in the form of a vector of ratios ranging from 0 to 1 representing the probability of that behaviour. This policy is randomly sampled to select an action when exploration is taking place, and the highest probability is taken when testing for the best model. The policy and the value output by the critic are saved and stored for the training stage at the end of the iteration, along with the state that was input, the action that was chosen by the choreographer, the reward that was output, and whether or not the level was completed.

At the end of an iteration, the training stage is entered. First, the data collected throughout the iteration is analysed to calculate the General Advantage Estimation (GAE). Effectively, this shows how much better or worse an action will effect the current state. This is how the critic value is trained.

To calculate the GAE, first initialise it as 0, and declare a list to hold the return values, a representation of how positive or negative each action affected the game. Then loop through the lists of values output by the critic, rewards and status of whether or not it resulted in the level completing, going from the last values to the first. For each iteration, first update the GAE value,

$$GAE = GAE \cdot \gamma \cdot \tau, \tag{4.2}$$

where $\gamma$ is a discount factor to ensure that present rewards are more valuable than later rewards, and $\tau$ is a smoothing parameter to reduce variance. The GAE value is then calculated as,

$$GAE = GAE + r_i + \gamma \cdot v_{i+1} \cdot (1 - d_i) - v_i, \tag{4.3}$$

where r is the reward, v is the value, and d is the value of done. Looking at the value of done allows us to see if the level was completed, it is 0 if not, 1 if true. If the level was completed for any reason, then the next state was not a result of actions taken in this one so should be discounted. The GAE value is then added to the value and then appended to the list of return values, R.

Once the return values have been found, the training will occur over a set number of epochs, Each epoch will parse the data carrying out a training step over a batch selection of random samples from the data. From this batch a new policy will be created, and from that new policy the likelihood of choosing the actions that were chosen is considered, and compared to the old policies from the sample. The ratio of the two is given as

$$\theta = e^{log\pi_{new} - log\pi_{old}}, \tag{4.4}$$

where $\theta$ is the ratio, $\pi_{new}$ is the new policy and $\pi_{old}$ is the old policy.

Using this ratio and the return values calculated earlier we can calculate the loss value for the actor. The loss is calculated as

$$L_a = -\frac{\min(\theta \cdot (R_{batch} - V_{batch}))) + clip(\theta, 1-\epsilon, 1+\epsilon)}{2} \tag{4.5}$$

where $L_a$ is the loss value for the actor, min is a function returning the minimum value of a set, $R_{batch}$ is the return values of the set R at the randomly selected batch indices, $V_{batch}$ is the critic values recorded from training at the batch indices, $clip$ is a function that takes the set of $\theta$ as input and clips the ratio with a minimum of $1-\epsilon$ and a maximum of $1+\epsilon$, and $\epsilon$ is a clipping value to prevent the policy from deviating too far. The critic loss, $L_c$ is simpler to define. It can be taken as the MSE between the batch of return values, $R_{batch}$, and the value $V$ produced by the critic when creating the new policy. The total loss of the two is then measured as

$$L_T = L_a + L_c - \beta \cdot E \tag{4.6}$$

where $L_T$ is the total loss, $\beta$ is an entropy coefficient, and E is the entropy measured across the new policy.

From this total loss, a backwards pass can occur and the choreographer model can be optimised. This occurs for however many epochs were set at the start of training.

As the testing is being carried out as training occurs, the best performing models are saved for implementation. These models are saved automatically if they complete the level.

# 5 Evaluation

## 5.1 Feature Extraction and Behaviour Training

Whilst implementing the solution, it was necessary to evaluate the success of the feature extraction and behaviour activation whilst training to ensure that the next step of the solution would be viable. As can be seen in the data gathered on feature extraction in Figure 4.2, the model performed well. The data had a sharp decrease in loss over the first 10 iterations, and then began to plateau at a low value around 0.001. It is also easy to see the improvement and effectiveness of this model when looking at the difference between the reconstructions produced at the start and at the end of training, as in Figure 3.1.

Performing feature extraction does however put an additional strain on the limited memory available to the GPU. This is especially true if features are stored on the GPU as tensors when training an epoch, which can quickly take up space. The feature extraction was trained relatively quickly when run on a GPU, completing in a matter of hours.

When training the behaviours, the behaviours did train as desired as seen in 4.5, though perhaps not as well as possible. All behaviours saw a decrease in loss over the training period that roughly correlated between the training data and the validation data. The validation data did follow the pattern of the training data, but there was a notable deviation between the training data and the validation data. Both still produced a low enough loss to be effective though. The validation data skewed closer to the training data for some behaviours, specifically u and d. This was likely a result of the lower number of samples of those buttons being pressed, as they rarely come into play. The training data for all the behaviours can be seen in Appendix A.1.

## 5.2 High-Level Choreographer

To evaluate the success of the choreographer, a baseline was required for the results to be compared against. The baseline was measured by carrying out training on the original PPO code for solving Super Mario Bros. that was used as a base for the choreographer. When implementing the choreographer, an effort was made to ensure that the choreographer followed the same structure and used mostly the same resources to ensure that the difference produced between the two agents was as a result of the low-level behaviours' training. This meant using just a state as input for the choreographer instead of performing feature extraction as for the behaviours. They were also trained under the same conditions, with the same parameters and the same amount of workers running, both learning only the first stage of the game.

The total reward produced across an iteration was produced whilst training. By looking at the average values of these total rewards, we can see that the choreographer was successfully training. When we compare the rewards against those produced by the baseline agent over the same amount of training sessions, as in Figure 5.1, we can see that the behavioral-based approach provides a slightly better result. It is difficult to observe on the graph, but this improvement is especially noticeable in the very early stages of training, as the standard agent is learning not to implement behaviours that are counter-intuitive, the low-level behaviour activation is already preventing these counter-intuitive actions from being carried out. Throughout the rest of the

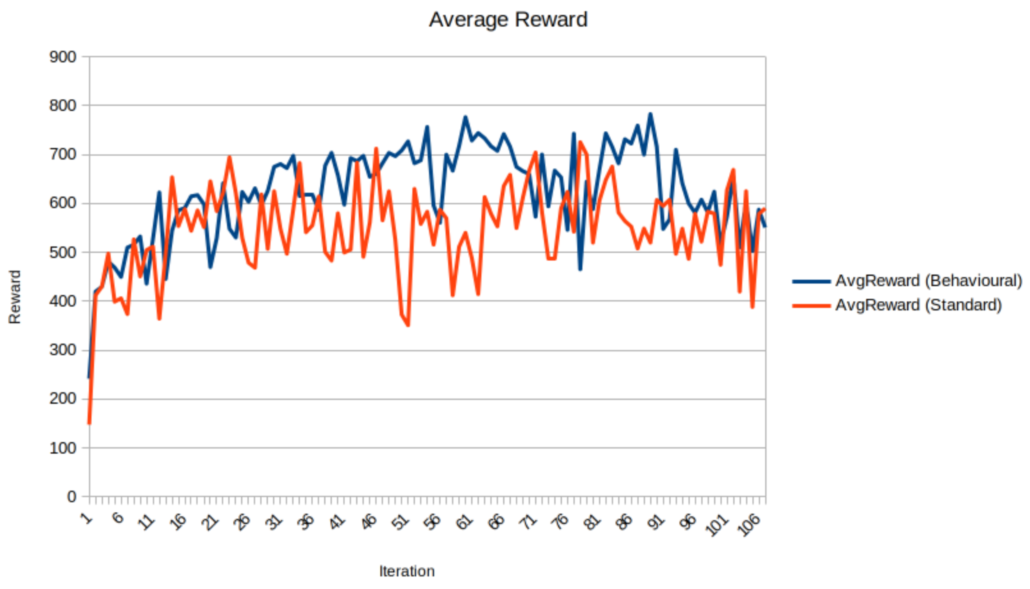


***Figure 5.1:*** *The average total reward over 20 training iterations of the choreographer and baseline plotted against each other.*

***Table 5.1:*** *The time taken over which testing was carried out.*

| | Iterations | Time (mins) | Time (hrs) |
|---|---|---|---|
| Baseline | 49095 | 4950 | 82 |
| Baseline | 28009 | 2678 | 44 |
| Choreographer | 28009 | 8102 | 135 |

training, the agents tend to receive similar rewards as they carry out exploration, however the choreographer suffers less pronounced dips in rewards than the standard agent. Again, this is as a result of the low-level behaviour training preventing the agent from carrying out the same mistakes.

### 5.2.1 Training Times

Agents were trained and tested simultaneously, with data being recorded and were stopped training when enough data had been gathered and the agent was performing reasonably well. It should be noted that Tables 5.1 and 5.2 show the recorded times taken for both the training and the testing respectively, however there is erroneous data. As the training and testing occurred simultaneously, it does not follow that they should have such different training times. Once this error was discovered, it was too late to find the cause of the problem and retake the data. However, though the times for training and testing are not consistent with each other, they are still consistent between the baseline and the choreographer, so conclusions can still be drawn from them. The time recorded for training was more accurate to the length of time training and testing was carried out for, so assume those values are true.

Just looking at the comparisons of the times shows some interesting data about how the two were comparatively executed. Table 5.1 shows the choreographer was tested for a considerably longer time than the baseline, however the agent was executing at a much slower rate, and thus was only able to achieve around two thirds of the attempts the baseline did. This would follow from

*Table 5.2: The time taken over which training was carried out.*

| | Iterations | Time (mins) | Time (hrs) |
|---|---|---|---|
| Baseline | 2132 | 1795 | 29 |
| Choreographer | 2132 | 1020 | 17 |
| Choreographer | 5645 | 2531 | 42 |

the fact that the choreographer has to carry out all the same processes that the standard agent does, in addition to performing feature extraction and finding the behavioural activation for all of the behaviours. This suggests that even if the approach to behavioural training does produce a result in less iterations than the standard end-to-end approach, it may take a longer time to do so. Confusingly however, the training data in Table 5.2 shows that the choreographer achieved more iterations in a much shorter time than the baseline model was capable of. The reason for this disparity is unclear. The training process also has to undergo the exact same conditions for both, with the choreographer taking on the additional weight of feature extraction and behaviour activation. This cannot be a result of any potential time recording issues, as there is invariably more data recorded for the choreographer training than for the baseline. A possible cause may have been varying levels of strain that was put on the GPU whilst training was being carried out for the various behaviours, but that seems unlikely to cause this level of discrepancy.

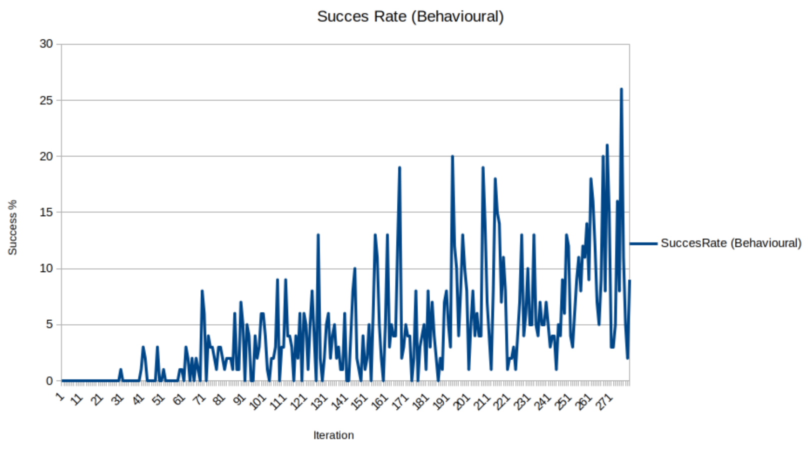
### 5.2.2 Success Rates

The agent may occasionally complete the level, but this does not mean that the agent is yet effective. This can happen simply due to luck and exploration, but may still represent a model that usually will fail to solve the level. Instead to measure the effectiveness of the agent, the number of times the level was successfully completed across 100 attempts was measured. As seen in Figure 5.2 (a), the agent did successfully learn to solve the level, and grew increasingly effective as it trained, reaching at its peak a little over 25%. However, when compared to the baseline, this initially seems an underwhelming result, which also showed an increasing level of effectiveness, peaking at over 35%. However, a few factors must first be taken into consideration. The behavioural agent performed considerably less testing iterations than the standard agent did, but the standard agent was trained and tested across a shorter time period, and it achieved less training in that time.

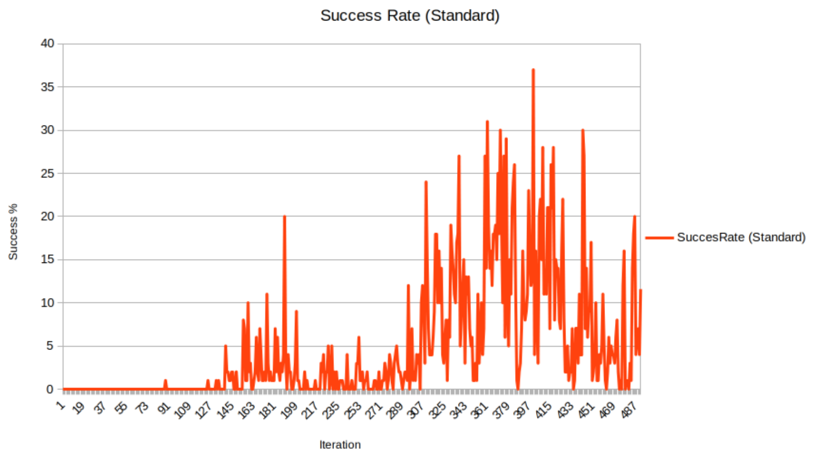
When the two success rates are plotted against each other, as in Figure 5.2 (c), it actually appears that the behavioural agent was increasing in its success rate much quicker than the standard agent, and that had it run for the same amount of iterations it would have produced a more successful agent. This is however countered by the fact that the behavioural agent had received more training than the standard agent, and was in fact training at a faster rate than the standard agent. This means that even though less testing iterations had been performed, the behavioural agent had received more training and should have been performing better than the standard agent at the equivalent point.

## 5.3 Summary

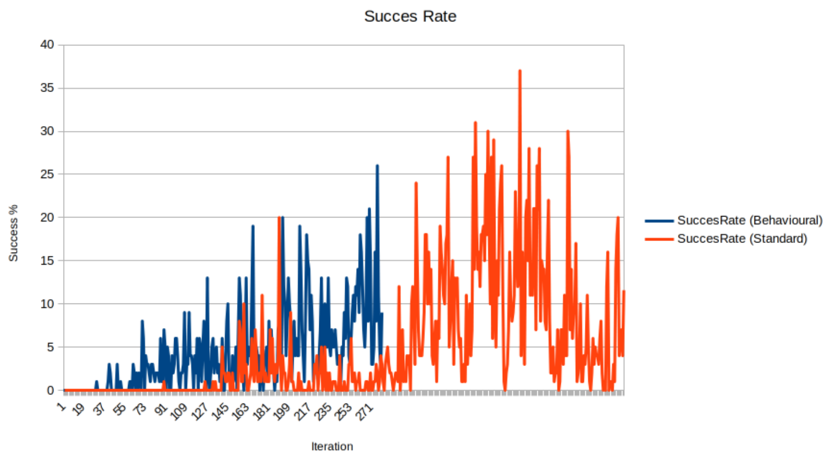
It can be seen from all the evidence that the behavioural approach to learning does work, producing a valid effective agent. The agent receives slightly better rewards when training than the end-to-end approach does, but in most other regards, it proves to be a lesser approach. The agent takes more training over a longer period but produces a lesser result. This may be a consequence of the fact that the number of valid inputs to the environment had to be reduced. The main effect of the low-level behaviours functioning simply as an activation value is that an agent learns when it should refuse to carry out a behaviour. This means it learns better how not



*(a) The success rate of the choreographer agent.*



*(b) The success rate of the baseline agent.*



*(c) The success rate of the choreographer agent charted against the success rate of the baseline.*

***Figure 5.2:*** *The success rates of the baseline and choreographer testing agents, calculated by the amount of times the level was completed across 100 attempts.*

to do bad behaviours than it does to implement the effective ones. Since the largely redundant inputs were already removed as viable actions manually, the standard agent had achieved largely the same effect as the behavioural-approach, without the need for extensive expert data, behaviour training and feature extraction.

# 6 Conclusion

## 6.1 Summary

Over the course of this project, a behavioural-based reinforcement learning approach has successfully been implemented to solve the NES classic game, Super Mario Bros. This approach sees a few crucial features implemented. Feature extraction, which reduces frames to a set of features so as to better extract the relevant information when training the low-level behaviours. Low-level behaviour training, which learns how best to activate the 6 valid input buttons for Super Mario Bros. by learning from recorded expert data. And a high-level choreographer, which learns to coordinate these low-level behaviours together to best play the game by implementing the Proximal Policy Optimization reinforcement learning algorithm.

Having evaluated the approach that was taken, it can be seen that whilst this behavioural approach does create a functioning agent that can successfully complete a level of the game. However, it does not appear to be a better approach to solving the game than the standard end-to-end learning approach. Passing data through all the required neural networks can be memory intensive, and produces a slower agent. Collecting the required expert data and training the low-level behaviours in addition to the high-level choreographer increases the end-to-end time even further when compared to the normal approach. And in spite of the greater training carried out and the additional time taken, the agent appears to perform less effectively than the agent trained with the standard approach.

## 6.2 Further Work

### 6.2.1 Further Experimentation

Much of the evaluating the effectiveness of the project was limited as a consequence of hardware and time restraints. There was little time for hyper-parameter tuning which could have resulted in a much more effective model. Had there been adequate time to tune and train a model using the standard approach that could solve the game with all the possible combinations of buttons as valid inputs, then the effect of the low-level behaviours may have been much more pronounced, and shown the value of this approach in conditions where the input actions are not manually restricted to ensure effectiveness. The choreographer was also only tested against the first stage of the game, where behaviours are relatively simple. Testing the agent against later stages where the behaviours are more varied and complex may have produced a more pronounced improvement.

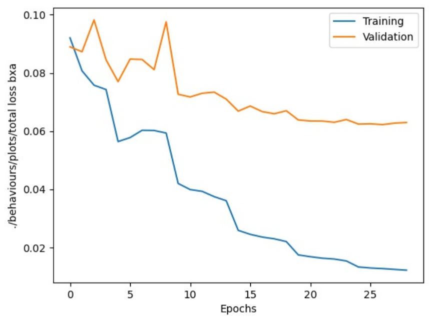
### 6.2.2 Alternate Behaviours

As mentioned in the design section, the behaviours were trained from recorded button inputs because it would be too complex to create a new environment where these behaviours could be effectively learned with a reinforcement learning approach. If an expanded effort was made to categorise the different kinds of obstacles and behaviours implemented through out the game, and an environment was created where the agent would be tested against randomly generated variations of these obstacles, it could be possible to train the behaviours without expert data and to replace the combination of button control behaviours with behaviours that are more distinctly identifiable, such as defeating an enemy, clearing an obstacle or collecting an item.

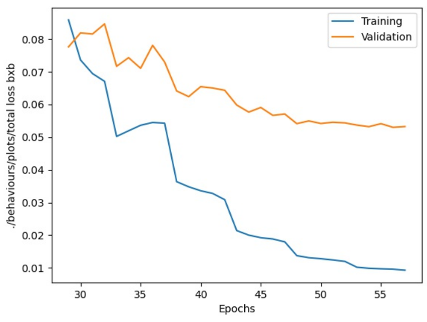
### 6.2.3 Expert Behaviours

Though the recorded data used throughout was described as expert data, it is largely just data of an average level of effectiveness by human standards. The importance of low-level behaviours might be more pronounced if trained on data and in an environment where a very rigorous and precise level of control is required. There is a sub-sect of fan-made Mario platformers known as Mario Kaizo, where highly complicated levels are presented that are only capable of being completed through very precise techniques used by high-level players. Using data recorded with these kinds of behaviours in an environment that requires that level of precision may be more feasible with a behaviour-based approach than with standard end-to-end learning reinforcement.

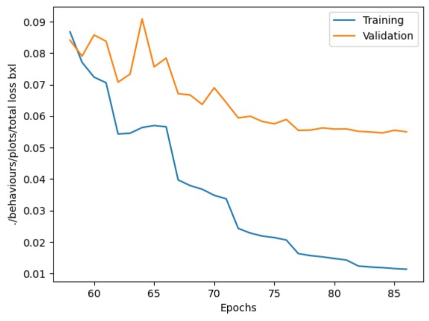
# A | Appendices



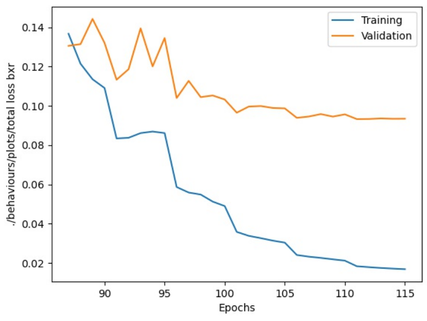
*(a) Total loss per iteration of the training set against the total loss per iteration of the validation set for behaviour a training.*



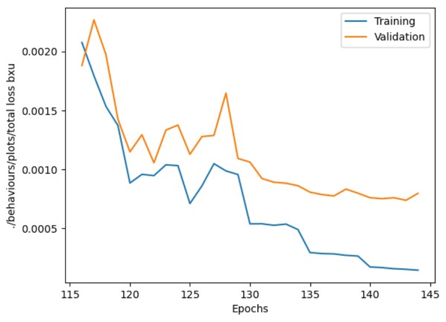
*(b) Total loss per iteration of the training set against the total loss per iteration of the validation set for behaviour b training.*



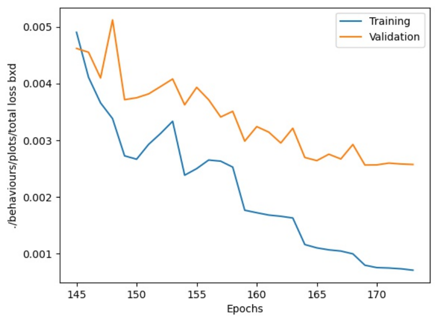
*(c) Total loss per iteration of the training set against the total loss per iteration of the validation set for behaviour l training.*



*(d) Total loss per iteration of the training set against the total loss per iteration of the validation set for behaviour r training.*



*(e) Total loss per iteration of the training set against the total loss per iteration of the validation set for behaviour u training.*



*(f) Total loss per iteration of the training set against the total loss per iteration of the validation set for behaviour d training.*

***Figure A.1:*** *The data for training all of behaviours.*

# 6 Bibliography

Nguyen, V., 2020. Proximal Policy Optimization (PPO) algorithm for Super Mario Bros. [online] GitHub. Available at: https://github.com/uvipen/Super-mario-bros-PPO-pytorch [Accessed 4 April 2022].

Brooks, R., 1986. A robust layered control system for a mobile robot. IEEE Journal on Robotics and Automation, 2(1), pp.14-23.

Karakovskiy, S. and Togelius, J., 2012. The Mario AI Benchmark and Competitions. IEEE Transactions on Computational Intelligence and AI in Games, 4(1), pp.55-67.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. 2017. Proximal Policy Optimization Algorithms. [online] Available at: https://arxiv.org/pdf/1707.06347.pdf [Accessed 4 Apr. 2022].

Pore, A. and Aragon-Camarasa, G. 2020. On Simple Reactive Neural Networks for Behaviour-Based Reinforcement Learning. [online] Available at: https://arxiv.org/pdf/2001.07973.pdf [Accessed 4 Apr. 2022].