Master in Computer Science

Guided Learning: Accelerated Learning Through Guidance

Keith Tunstead

A dissertation submitted to the University of Dublin, Trinity College, in partial fulfilment for the degree of

Master in Computer Science (MCS)

Supervisor: Prof. Joeran Beel

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science & Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland

April 10, 2019

Declaration

I, Keith Tunstead, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other university; and that the library may lend and copy it or any part thereof on request.

Signature:

Date: 10/4/19

Table of Contents

Abstract

This dissertation applies human intervention and guidance to modern reinforcement learning environments. Using Open AIs' Retro application, playing the classic Super Mario Bros, the concept of Guided Learning is presented where a reinforcement learning agent can effectively 'ask for help' as it encounters stagnation. At which point a human supervisor can effectively 'guide' the agent as to how to progress beyond the point of stagnation. This guidance is then encoded as a new, separately trained neural network known as a 'Taught Response Memory' that can be recalled when another 'similar' situation arrises in the future. This dissertation applies Guided Learning on top of an evolutionary algorithm but also shows how Guided Learning is algorithm independent and can be applied in any reinforcement learning context where human intervention is applicable. The results show that the initial Guided Learning implementation gives superior rate-of-progression in both the best case and worst case scenarios and yields, on average, an increase of 136% in the rate of progression of the most fit genome with minimal human intervention. This is due to the significant reduction at which the agent must explore the solution space by effectively giving the agent more information to exploit. The results obtained show good promise for Guided Learnings potential as such results were obtained with only a partial implementation and much future work still remains.

Keywords

Reinforcement Learning, Agent Teaching, Stagnation, Artificial Neural Networks, Evolutionary Algorithms, Supervised Learning, Human Skill Transfer

Acknowledgments

I'd like take this opportunity to thank my family for their exceptional support, not only through this year but over the entire duration of the last 5 years, I truly could not have done this without them. I'd also like to thank my friends and classmates for their continued support. Finally, thank you to my supervisor, Prof. Joeran Beel, for his exceptional patience and support throughout the duration of this project.

Chapter 1: **Introduction**

This dissertation will begin by detailing the problem of stagnation in reinforcement learning environments, followed by the goals for this dissertation. It will briefly explore some of Alan Turing's ideas of 'child machines' and his 'guiding principle', how they inspired the idea of 'teaching' (Lin, 1992) and the hypotheses that human teachers will play a crucial role in the teaching of initial general purpose AI systems. It will continue by then discussing our purposed solution to stagnation in reinforcement learning, Guided Learning, that uses human intervention and a novel method of encoding such intervention that effectively allows agents to 'ask for help' when they encounter stagnation and re-use such help in other similar scenarios. This dissertation will discuss how using such intervention can lower the degree that the agent must partake in exploration of the state space and, in doing so, increase the agents rate of progression towards the goal.

Finally, this dissertation will detail the performance improvement achieved by using Guided Learning in a gaming environment. Both a baseline approach and the Guided Learning algorithm will be compared. Notable improvements include over double the rate of progression on average for the most fit genome (136%) and superior rate of progression against both best case and worst case scenarios (558% and 47% increase respectively) with minimal human intervention. The approach will be tested specifically using Super Mario Bros for the Nintendo Entertainment System (NES).

1.1 Background

Typically, reinforcement learning agents work by balancing between exploration of the solution space and exploitation of known, previously explored states or state transitions. This balance is inevitably one sided as the agent starts off because it is forced to explore the state space initially. Not only this, but more generally, if the agent reaches a previously unseen situation it can also be forced to explore the options available to it in order to determine the outcomes of its actions. This can often lead to significant stagnation as the agent explores methods of progressing. Sometimes an agent can become stagnated while a human oversees its progress and oftentimes that human will have some idea as to how the agent could/should progress, particularly in the context of games.

The idea of human intervention in the learning process is not a new one and has its roots in some of Alan Turing's most foundational work (Turing, 2009)(Turing, 1996). Turing discusses subjecting the machine to a 'range of experience' and using 'indexes of experience' on top of a straightforward memory (see TRMs in section 2.4). Turing also discusses how the education process would be 'essential to the production of a reasonably intelligent machine' (Turing, 1996).

1.2 Research Problem

Current AI implementations place a large focus on the idea of fully automated learning, where an agent is given a task, with perhaps some samples of historical human input (Silver et al., 2016), and set to solving that task on its own. Although this is a perfectly valid approach and should be encouraged, I hypothesise that as we push ever closer to more general purpose AI, direct human intervention will be become increasingly important. Fully automated learning is very analogous to leaving a child to his/her own devices and expecting them to reason about the world on their own. There is only a finite amount of progress that that child would be able to make, primarily on a trial and error basis. We can observe similar situations in reinforcement learning environments with the phenomenon of stagnation (see figure 1.1).

One of the primary problems with training any kind of modern AI is stagnation. Stagnation occurs when the agent ceases to make progress in solving the current task. As an example, in a typical fully interconnected feed-forward Artificial Neural Network that employs Stochastic Gradient Descent (SGD) to search the solution space, typically referred to as back-propagation, this stagnation can be caused for a number of reasons. One such reason would be reaching a low gradient point in the solution plane while using a low learning rate, α, this causes very little progress to be made over the plane. A second reason for this would be using too high a value for α, such that the function jumps continually between the troughs of the local minima. These examples are typically addressed by using a variable learning rate that adapts over time (see section 4.2.2).

Such methods are not applicable to reinforcement learning environments as alternative algorithms are used to search the solution space. This is because reinforcement learning typically involves an agent that must make progress through an environment and the ideal input-output pairs are not known, as they are in Supervised Learning. As a result, reinforcement learning uses algorithms such as Q-Learning, where a policy of best actions from each state is generated, and Deep Q Learning, where a neural network is used to estimate the value function that is used to determine the best action to take. We will discuss such algorithms in more detail in section 3.1. As a result of these algorithm differences, other methods of reducing stagnation are required as we will discuss in chapter 4 .

While the concept of teaching already exists for reducing stagnation in reinforcement learning (see section 4.1.3), it primarily lacks the following:

1. Application in a modern context.
2. Standardisation of how intervention is encoded.
3. Details of when/how the system receives intervention.
4. Independence from the underlying algorithm.

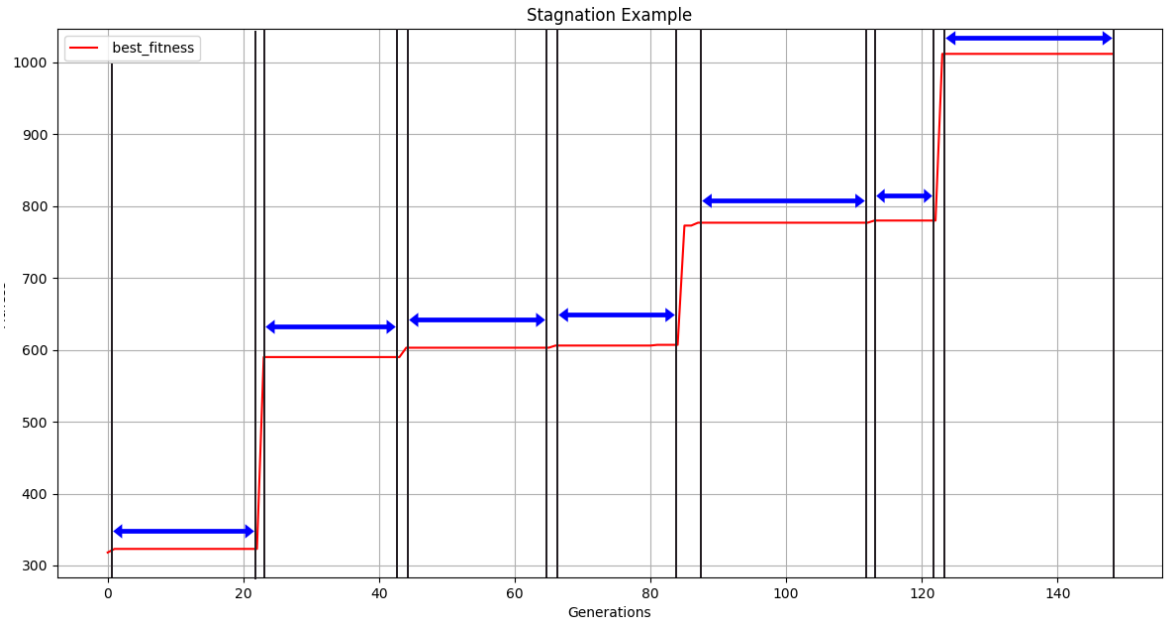

Figure 1.1: Stagnation Example. Blue arrows indicate the periods of stagnation.

1.3 Research Question

This project will attempt to answer the question: "To what extent can human intervention be used to reduce stagnation in the training of Artificial Neural Networks in modern Reinforcement Learning applications?". This dissertation will outline an algorithm that will allow specific types of agents to effectively 'ask for help' from a human supervisor when it has encountered stagnation, a novel method of encoding such intervention, if received, and the ability to re-use such interventions in other similar situations in the future. The primary focus of this dissertation will be the application of such a system in the context of games.

1.4 Research Goal

The primary goal in this dissertation is twofold, firstly, to create a system whereby when stagnation has encountered over a certain a period of time, the system will effectively ask a human supervisor for help in order to reduce the width of the stagnation periods, as displayed in figure 1.1. Secondly, to outline how such interventions should be encoded separately, and independently, from the main algorithm being used.

The secondary goal in this dissertation is to create a discussion on how human intervention may be advantageous to our pursuit towards modern day general purpose AI. We hope to support the argument made by Turing in a modern context, where he outlines that education experiences of the agent should be separated from standard experiences and that such an education process is essential in the creation of artificial general intelligence (Turing, 1996).

1.5 Contribution

To the best of my knowledge, Guided Learning is unique in that:

1. It allows the agent to 'ask for help' from a human supervisor as it encounters stagnation.
2. This intervention is optional and, as such, the agent does not require constant supervision in order for it to make progress.
3. The given human intervention is represented as separate 'memories', each of which are encoded as a separately trained neural network and its corresponding stimuli - 'Taught Response Memories' (TRMs).
4. It works independent of the underlying reinforcement learning algorithm employed. This means that human intervention can be taken advantage of in any reinforcement learning environment, provided such intervention is applicable.

When evaluated using an evolutionary algorithm and taking, on average, 10 interventions over a period of approximately eight hours, Guided Learning showed improvement both the rate of progression in training of both the mean best fit genome and the mean average genome by over double (136% and 112% respectively). This has a significant impact on overall training times over an extended period of time and could potentially result in the ability to train agents to achieve more complex tasks in more reasonable timeframes. TRMs also have the potential to be applied to previously unseen scenarios such as other levels in the game or even other games entirely. See chapter 6 for more detailed results and chapter 10 for more on future work.

Chapter 2: **Guided Learning**

In this chapter we will present our vision of the Guided Learning system. We will discuss the concept of using humans as teachers, explaining our rationale behind Guided Learning and introducing our concept of Taught Response Memories (TRMs) which is how we purpose human intervention should be modelled.

2.1 Vision

The ideal implementation of Guided Learning will allow an agent to effectively 'ask for help' when it encounters stagnation and has not made progress towards the goal in reasonable time. The system will then present the opportunity for a human to intervene and effectively show or guide the agent as to what action to take in order to overcome the present stagnation. Such intervention is optional and if the system receives no intervention within a reasonable time, then the agent continues as normal. This means that without any human intervention, the system should perform exactly the same as it did without Guided Learning present.

When human intervention is given, the intervention needs to be recorded in some way. Directly recording and storing the input is not deemed to be an option as it is not very flexible. Such a method does not allow for stochastic environments and while it may be possible to apply such methods to other scenarios, the lack of flexibility is an issue. For this reason it is necessary to encode the intervention in some other way. Guided Learning does so by training a new neural network via back-propagation. Where the networks expected input-output pairs are the stimulus and the human intervention respectively. The idea is that this 'memory' is plastic and can be altered over time in order to tend towards either a more optimal solution for that single scenario or towards its applicability, more generally, to other scenarios.

Such 'memories' should be completely independent of the underlying reinforcement learning algorithm used. This allows Guided Learning to be applied in any reinforcement learning environment where human intervention is applicable. For example, in this dissertation, an evolutionary algorithm is used as the main reinforcement learning algorithm but is not required to implement Guided Learning as any other algorithm that supports reinforcement learning could be used in its place.

Depending on how the stimulus is encoded (i.e. pixel data or some abstract representation), it should be possible to use such 'memories' in other, unseen scenarios. For example, in the game Super Mario Bros, when Mario gets stuck at the first green pipe and receives human intervention to jump over said pipe, such intervention should then be re-used in order to attempt to jump over the second green pipe, and the third and so on. There's also no particular reason as to why this 'memory' should not be recalled, not only in the first level but in other levels of the game also, where Mario encounters a green pipe. Taking this even further, ultimately, if one were to use an abstract representation of the environment, there is no reason why such a 'memory' could not be recalled in another similar platform or 'run and jump' based games where the agent is faced with some barrier in front of it.

2.2 Rationale

Having a human directly interact with an agent in order to help it progress is a potential way of reducing stagnation. This could particularly apply in scenarios such as games where it is trivial to allow human control. While fully automated learning plays a very important role today, as we strive to create ever more complex AI systems we should begin to question if this will be feasible for the complex environment of the real world. As mentioned previously, this is not a new idea and harks back to some of Alan Turing's original work, particularly his concept of 'child machines' (Turing, 2009) where he defined intelligence as the systems ability to learn, rather than current knowledge of the system. It is the systems ability to learn that will eventually lead to progression.

Taking the concept of 'child machines' further, and taking humans as a benchmark for general intelligence it is important to recognise the process involved for us as humans also. Whilst all humans have a distinct ability to learn, not all humans become astronauts. Equally, if you were to take a human born and raised in a remote environment he/she would likely not have the ability to read Shakespeare for example. It may seem obvious, but throughout our learning lives we are surrounded by teachers that help us and teach us how to progress. Not only do these teachers exist in educational institutions, but in the home also. This is an important part of our initial development and in the case of learning our first language for example, it is a necessity. That being said, we must also recognise the importance of self-learning and how failure can lead to progression. This is the classic rationale for exploration in reinforcement learning. If a child was to be left alone without any assistance, in the educational sense, while he/she might, for example, learn how to walk via trial and error, they would struggle to learn even the most basic of concepts of language on their own. It is this analogy that we can also apply to the first iterations of general purpose AI. I hypothesise that the first general purpose AI systems, while having the ability to self-learn through exploration, will also have human teachers and that these teachers will play a crucial and central role in the development of such a system. As we push closer to general purpose AI, it may be unreasonable to expect AI agents to effectively reason about the world without some form of direct human guidance, regardless of its underlying implementation.

2.2.1 Human Experts & Artificial Neural Networks

The idea of training Artificial Neural Networks based on human expert input has existed for a long time. Nechyba and Xu (1995), Guez and Selinsky (1988) and Asada and Liu (1991) are all examples of this. It is very common in games, as displayed with Deep Blue (Campbell et al., 2002) and AlphaGo (Silver et al., 2016), to use previous human vs human game records to train an AI. Such techniques are not too unlike some of the ideas we will purpose, however, rather than the system simply relying on the data that has been given, the system will actively seek out new information as it is needed, a process we refer to as 'asking for help'. Human expert samples will not be available in all cases or for all scenarios and therefore this may be an advantageous process. Guided Learning is by no means presented as a replacement for these current techniques, rather as an additional step in the learning process as a whole.

2.3 'Asking for Help'

In order for a AI agent to 'ask for help' in a productive manner we must define clearly the situation whereby it is appropriate for the agent to do so. There is no point in an agent requesting help in a situation where it can already make progress, and has previously shown to make progress in the past. The simplest way of defining such a situation is to determine if the system has become stagnant and is unable to make progress beyond a specific point. As an example, in World 1-1 of Super Mario Bros the first enemy encounter is a Goomba directly in Marios path, This can be seen in figure 5.1. In order for Mario to progress beyond this Goomba, it is necessary for him to jump over it. Using a standard reinforcement learning algorithm it is likely that the agent will struggle to pass this point for quite some time due to the amount of exploration the agent must partake in. In the case of using an Evolutionary Algorithm to search the solution space, depending on the diversity of the population, this can take many generations. For example with an initial population count of 10 genomes, it may take 40 generations but with a population count of 50 genomes, typically a genome will exist in the first generation that will pass this point. During our evaluation in chapter 6, this specific point was not much of an issue, however the following green pipe was and as a result, was quite often the first point where Mario will 'ask for help' and the opportunity for human intervention is presented.

In the context of reinforcement learning in general; judging stagnation, while subjective, is quite a simple matter thanks to the fitness function. To achieve this, we can simply take the derivative of the the fitness over time ($\frac{\delta f}{\delta t}$) and use it to determine if stagnation has occurred. If $\frac{\delta f}{\delta t}$ is close to zero or negative over a given portion of time, t, then we can say that stagnation has occurred. At this point we can trigger the system to ask for human intervention. Intervention is optional and the request need not be satisfied in order to continue the learning process.

2.4 Taught Response Memories (TRMs)

In this section I will discuss the purposed method of encoding human intervention. The idea is to facilitate the mapping of actions given during human intervention to known stimuli and to encapsulate the response using a neural network. Using a neural network in such a way, rather than just directly recording the human input, allows for plasticity of the network in the future. This allows us to either take sub-optimal human input and optimise it over time or to adapt the network so that it may be applied more effectively to other scenarios.

The main idea of applying these networks to reinforcement learning problems is the fact that it reduces the amount of exploration that the agent must partake in (see section 3.1.1 for further details on the explore-exploit tradeoff). Instead of the agent discovering the necessary actions to take via trial and error, the agent can simply 'ask for help' from a supervising human. Assuming the help is given, the agent can then 'remember' the actions when presented with a similar situation in the future.

2.4.1 Introduction

TRMs stem from the idea that humans have two kinds of responses to stimuli, learned and innate responses (Kimble, 2016). Certain stimuli such as touching a hot surface will trigger the innate response of removing your hand from the surface. These types of responses are instantaneous and do not require any conscious form of 'thinking'. Other stimuli will trigger learned responses such as knowing that $x = 4$ when presented with the equation $x = 2 + 2$. Whilst one may argue that logic itself is not learned, it simply exists, the meaning of the *symbols* '2' and '+' *are* in fact learned. It is not an innate response to know what '2' means, someone, when you were very young, taught you what it meant. In the context of this dissertation, the 'innate' response of the system can be thought of as the main neural network being evaluated. Whilst the 'learned' responses can be seen as a collection of separately trained TRMs.

A TRM is a memory of a series of actions that an agent has been taught in response to specific stimuli. To give a high level example of this in the context of Super Mario Bros; when Mario is confronted with an enemy/obstacle in front of him, he will more often than not be required to jump over it. The stimuli is the situation where Mario is confronted with an object and the series of actions involves jumping over the object. The initial main network may or may not already have this encoded in it's function, or some other output that achieves the same affect. If it does, then Mario will perform as expected without any assistance. However, if it does not, then it is highly likely that Mario will need to explore the solution space for quite some time before discovering an output that achieves the correct result, in this case, jumping over the obstacle.

2.4.2 Knowledge Transfer

Humans have numerous methods of knowledge transfer, all of which revolve around the four main forms of communication: written, verbal, visual and physical. Let us take an analogy of a human playing a previously unseen game. He/She may already be familiar with other similar games and so can make progress initially. Then, let's suppose the player reaches a point he/she is unfamiliar with. They will begin the process of exploration, searching for the correct sequence of actions to take in order to progress further. It would be very easy for the player to minimise this exploration, if he/she desired, simply by reading a guide for the level or asking a friend who has previously played the game. This friend could tell the player verbally how to continue or, if the friend happens to be in the same room as the player, they could physically and visually show/guide the player as to how he/she should proceed. However, it's important to note that the players friend may not give the ideal or optimal solution. While the solution given may lead to player progression, there may be a different option and the player can still discover this later on.

This analogy does not only work for games but for many tasks that we perform on a daily basis. For instance, when you start working in a new job, quite often a co-worker will assist you and show you how to complete tasks appropriately. We do this because it significantly reduces training time when compared to if you had to figure these things out on our own, not to mention reducing potential damage incurred by employees randomly exploring the tasks solution space.

2.4.3 TRM Implementation

A TRM is simply a neural network that is trained using human control data and its corresponding stimuli, a TRM may also contain other relevant information such as the duration of actions (see section 5.4.1). Specifically, fully connected feed-forward neural networks, with back-propagation, can be used due to all necessary input/output pairs being available. The inputs being the stimulus and the outputs being the human control. However, the underlying network implementation is not important and any other network architecture or gradient method could just as easily be used, but the speed of convergence will vary. It is however important that the TRM be a neural network. This is because such an implementation allows for plasticity. The idea being that each TRM can in fact vary over time. Just like the players friend not giving an optimal solution, we do not assume that the human intervention given is optimal either because it is unlikely to be. Training of the network itself is relatively simple because we do not care about overfitting. In fact, we purposely want to match the given input/output pairs initially and allow for some variation in the network later on.

The purpose of TRMs is to reduce the necessary exploration the agent must partake in by increasing the knowledge that it can exploit. Therefore making these TRMs plastic may seem counter intuitive because it reintroduces some exploration back into the system. However, the degree of exploration is limited to only minor changes in the network. These minor changes can vary but will typically involve a change of a single weight or the addition or removal of a single neuron in the TRM. Ultimately, the plasticity of TRMs is only necessary if we want to find the most optimal solution we can within a reasonable time frame. If we are searching for *any* solution then it can be advantageous to ignore this method, assuming that the TRM actually leads to progress being made.

TRM Selection & Activation

During normal evaluation of a network, the output actions will be mostly the outputs of that network. However, if the system has received human intervention and a TRM has been trained, the system will take the current stimulus and attempt to find a corresponding TRM which begins with similar stimulus. This TRM is identified if it is below a given 'similarity threshold' t for a given similarity function, $s(a, b)$, used to determine the similarity of two stimulus vectors. If a TRM is found (i.e. $s(a, b) \leq t$) then we will take the current inputs, feed them into the TRM and use the result as the output of the agent. If the appropriate value for t is set then we have found a situation that is compatible with a TRM and can, hopefully, take advantage of it. t is a configurable parameter and will need to be adjusted for various applications. The parameter must be tuned in order to minimise false positive and false negative TRM activation. See section 5.4.1 for more details.

TRM Replacement

Allowing TRMs to be replaced accounts for the fact that human input will likely be sub-optimal and may in fact lead to further stagnation or no progress at all. Therefore it is advantageous to allow the system to 'ask for help' in scenarios where it has previously asked but has failed to make progress. If this occurs, new human input will be trained and the previous TRM that occurs within a given fitness range from the current fitness will be replaced, assuming human intervention has been received. This means that the need for exploration via TRM plasticity is reduced when given more appropriate human input for the given scenario.

Single Action Taught Response Memories (SATRMs)

The issue with using neural networks as TRMs is that multiple human actions can conflict with each other. This can cause issues during back-propagation as the neural network may not converge and can make training of the network significantly longer. However, often times multiple actions are not required as a single action can be enough to make progress and overcome the source of stagnation. These special case TRMs can be trained much faster and can achieve errors that are orders or magnitude lower than multi-action TRMs. For simplicity, this dissertation will mainly focus on the use of SATRMs.

It's worth noting that multi-action TRMs can be split into multiple single action components, this would allow the use of multiple SATRMs in order to encode multiple actions. See chapter 10 for more details.

Chapter 3: **Background**

Since its inception, AI, and to a great extent reinforcement learning, has benchmarked itself via its ability to successfully play games. Particularly its ability to beat human players. From Arthur Lee Samuels first checkers playing program in 1959 (Samuel, 1988), to IBMs Deep Blue beating Garry Kasparov in 1997 (Campbell et al., 2002) and more recently Googles AlphaGo beating Lee Sedol at the notorious game of Go in 2016 (Silver et al., 2016). These events distinctly mark the ever increasing progression of AI over the years. While these systems, time and time again, prove their proficiency at single task problems, it is nonetheless a sign of our slow approach towards a truly general purpose AI.

3.1 Reinforcement Learning

Over the past number of years, Reinforcement Learning has been making significant progress. Rooted in some of Turing's original ideas of the use of 'favourable outcome' defined by the 'pleasure principle' with use alongside a 'random element' in order to achieve the perception of intelligence (Turing, 2009). Modern day reinforcement learning algorithms aim to maximise the idea of a cumulative reward. This typically manifests itself in striking a balance between 'exploration', Turing's 'random element', and 'exploitation', this is often referred to as the explore-exploit dilemma. At the core of many reinforcement learning algorithms are Markov Decision Processes. This relies on the Markov assumption that the probability of the next state s_{t+1} relies only on the current state s_t and the corresponding action performed a_t.

Reinforcement Learning algorithms typically involve an agent, a set of states S and a set of transitions or actions from each state T. Each transition is accompanied by a reward. The agent then typically has the goal of maximising its cumulative reward over time. As a result, the agent does not require any prior knowledge of the task. There is often a starting state and a terminal state, although a terminal state is not necessary. If a terminal state exists then the learning process is often episodic, otherwise it is continuous. Many different reinforcement learning algorithms exist. Some can construct simplified models of their world, others are model-free. The output of typical reinforcement learning algorithms is a policy of actions to take in each given state.

3.1.1 Multi-armed Bandit Problem

The multi-armed bandit problem encompasses the explore-exploit tradeoff in reinforcement learning environments.

> 'Consider a gambler who is presented with the opportunity to play any of n one-armed bandit machines. He wishes to allocate his successive plays amongst these machines to maximise his expected total-discounted reward. He does this one play at a time on the basis of prior information and observations to date.' (Weber et al., 1992)

In order to determine which machines give the best probability of reward, the bandit is required to try out each machine and explore its outcomes. The bandit then needs to decided as to whether or not it is advantageous to explore unknown outcomes or continue exploiting known outcomes.

3.1.2 Q-Learning

Q-Learning is a model-free reinforcement learning algorithm and as such, must learn the outcome of transitions via trial and error experience. It uses the concept of Q-value iteration in order to determine the actions with the highest expected outcome from each state using the form of the Bellman equation presented in equation 3.1, this is also known as the value function.

$$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')] \tag{3.1}$$

Use of this value function allows each state transition to be assigned an expected reward value. In stochastic environments this eventually allows the agent to chose the state transitions with the highest probability of maximising its cumulative reward.

3.1.3 Deep Reinforcement Learning

Deep Reinforcement Learning (LeCun et al., 2015), the use of deep neural networks alongside the concept of reinforcement learning, has lead to great advancements in the field of AI. Typically deep neural networks are used to approximate the reward function, however, this has some issues. The primary issue is the fact that using a network to estimate the value function can have negative effects such as divergence and instability. The reason for this is mainly because a small change in the network can lead to a significant change in the value estimation. Such issues do exist in the evolutionary algorithm used in this dissertation and we will discuss this further in chapter 7.

Deep Q-Learning (Mnih et al., 2015) offers a solution to such issues and is a particular example of successful integration of deep neural networks and reinforcement learning. Essentially, a neural network is used to approximate the value function and a technique known as Experience Replay is used to give the network more stability and reduce divergence (see section 4.1.1). Deep Q-Learning has since contributed to advancements in applications in finance (Deng et al., 2017), robotics (Levine et al., 2016) (Gandhi et al., 2017) (Pinto et al., 2017) and self-driving cars (Pan et al., 2017) amongst many others.

3.2 Artificial Neural Networks and Evolutionary Algorithms

Artificial Neural Networks (ANNs) and Evolutionary Algorithms (EAs) were fields that were first merged in the 1980's. While ANNs typically made use of the back-propagation algorithm, GAs offered a direct alternative. During this time the topology of the network was still predetermined by an engineer. It wasn't until the late 80's when 'Topology and Weight Evolving Artificial Neural Networks' (TWEANNs) first emerged. The merging of the fields of ANNs and EAs has since been named Neuroevolution (see section 3.2). Around the same time, in the late 80's and early 90's, many researchers were also toying with the idea of humans interacting with ANN's and directly training them (Guez and Selinsky, 1988)(Asada and Liu, 1991)(Nechyba and Xu, 1995). Some of these focused on control applications (Guez and Selinsky, 1988), with others leveraging ANNs for human-to-human knowledge transfer (Nechyba and Xu, 1995). These projects primarily focused on Supervised Learning tasks and as a result, used standard feed-forward networks with back-propagation, training the network directly using the input from the human. Since that time there has been very little research in how humans may interact with modern day deep neural networks. The reason for the modern day lack of interest in the field is the push for fully automated learning and thus, minimising the need for any form of human interaction.

Neuroevolution

Neuroevolution refers to a type of neural network architecture that employs evolutionary algorithms as a method of searching the solution space(Franken and Engelbrecht, 2003). Employing genetic algorithms means that the network can begin with a minimal structure of just inputs and outputs. The network can then 'evolve' to add, remove or change existing vertices and edges. This means that the network topology is not predetermined by the human programmer as is typical in full connected feed-forward networks. A disadvantage to this is the fact that the network must be given time to grow in order to reach a capacity whereby it can encapsulate a reasonable solution. This can be seen as a kind of 'cold-start' problem. Examples of neuroevolutionary algorithms include (Stanley and Miikkulainen, 2002)(Stanley et al., 2005)(Stanley et al., 2009).

3.2.1 Topology and Weight Evolving Neural Networks (TWEANNs)

TWEANNs are a specific version of Artificial Neural Networks that do not use the back-propagation algorithm. Instead, they learn by way of Evolutionary Algorithms in order to adjust their topology and weights and are primarily for use in reinforcement learning tasks. Theses algorithms are inspired by Darwinism and the process begins by generating a population of networks, referred to as genomes. Each member of this population is then evaluated based on the fitness function. After this, the most fit genome is declared the 'champion' of that generation and the top n genomes are selected for crossover and continue on to the next generation. Crossover is essentially the same concept as mating in the animal kingdom, where two genomes share some of their characteristics to produce an offspring. These genomes also have a chance of mutating, this can involve a random change to a single weight or the addition/removal of a new node/gene in the network. Allowing the topology of the network to change in such a way allows us to begin with a minimal network topology and slowly expand the network as is needed. This will often tend towards a minimal topological solution. The second advantage of this is that as the topology of the network grows, its capacity also grows, thus reducing the impact of catastrophic interference (see below). These

kinds of algorithms can be effective in gaming environments and in reinforcement learning, the genomes are essentially used to approximate the value function.

Catastrophic Interference

This phenomena occurs when the network does not have the capacity to fully model the given function and, as a result, will tend to 'forget' previously known responses as new responses are learned.

Chapter 4: **Related Work**

In this chapter we will discuss several methods of stagnation reduction. We will focus mainly on methods used in reinforcement learning applications such as Experience Replay, Action Models and finally Teaching which is where Guided Learning places itself. We will also touch on methods used in Supervised Learning, specifically when using back-propagation.

4.1 Reducing Stagnation in Reinforcement Learning

4.1.1 Experience Replay

Experience replay is widely regarded as an affective approach to reducing stagnation in reinforcement learning environments. The idea is instead of performing an action from a state, taking its reward, making the appropriate adjustments and moving on, the experiences of the agent are recorded such that they can be re-visited or 'replayed' again in the future without the need for actually taking those actions in any form of simulation.

> '[Some reinforcement learning] algorithms are inefficient in that experiences obtained by trial-and-error are utilised to adjust the networks only once and then thrown away. This is wasteful, since some experiences may be rare and some (such as those involving damages) costly to obtain. Experiences should be reused in an effective way.' (Lin, 1992)

A *lesson* is defined as a sequence of such experiences. Intermittently, these experiences are given to the agent in an attempt to reinforce such good/bad actions without the need for the agent to actually take those actions. This requires that the environment remain partially static as rapid change in the environment may lead to previous experiences being irrelevant. There is also evidence that presenting experience replays to the agent in backwards order can, in certain instances, be more effective (Lin, 1991).

Experience replay tends to only be advantageous if the replay is part of the current policy. This means that if the policy is constantly changing, experience replays essentially become stale. This is because the actions being taken when triggering experience replays no longer reflect the current policy and, as such, may have unfair negative effects on that policy.

> 'Consider an agent whose current policy chooses a very good action a in state x. If the agent changes to choose a very bad action b in the same state, the utility of state x, eval(x), will drop dramatically. Similarly, the utility of x will be underestimated if the bad action b are replayed a few times. For experience replay to be useful, the agent should only replay the experiences involving actions that still follow the current policy.' (Lin, 1992)

Experience Replay can not really changed much since its original inception and has most notably been used in Mnih et al. (2015) where it showed much better stability and a decrease

in divergence in Deep Q Learning, where an artificial neural network was used to approximate the value function.

4.1.2 Action Models

Action Models (Sutton, 1990) are typically applied in the field of robotics (Hester et al., 2010) and allow an agent to experience the consequences of actions without actually performing them in the real world. This requires building a simulation model of the environment and allowing the agent to take actions there instead. This is advantageous if such a procedure is of course faster than simply running the agent in a real world environment. However, Action Models require an accurate model of the environment to be available. For stochastic environments in particular, the probability distribution of the outcomes are needed in order for Action Models to be successful (Lin, 1992). In a way Action Models, like Experience Replay, can make use of prior experience by using previous outcomes, in the real world, to estimate the outcome probabilities for the model.

While Action Models are not specifically aimed at reducing stagnation as it is discussed in the context of this dissertation, such as where an agent ceases to make progress towards the current goal, they do, in some cases, allow training to be performed much faster overall.

4.1.3 Teaching

Teaching is described as using human knowledge to reduce stagnation in reinforcement learning environments (Lin, 1992). This is however a very broad definition. As such, there are instances where only a single network is used and the human input is used to alter that network as in Clouse and Utgoff (1992). Such approaches however lead to several issues including:

1. Catastrophic Interference due to insufficient network capacity.
2. Bias towards the human input if Experience Replay is not employed.
3. Non-convergence while back-propagating the human input into the network due to the presence of conflicting samples.

Point 1 above is of particular concern when evolving the underlying networks from a minimal solution, as we will be doing in this dissertation. This is because the network begins with a minimal capacity that grows over time, as such early interventions, where they are most needed, will likely have a negative effect on the performance overall.

In point 2, bias towards the human input will depend on how the network is trained with such input. As an example, if we attempt to train the network by minimising error with the human input it is likely that the such input will be used in the majority of circumstances, even in circumstances where it is not advantageous. Therefore, there will be a bias towards such input. Equally, if the network is not trained to minimise the human input enough, then it essentially fails to learn.

Point 3 will unfortunately be an issue in any circumstance that takes multiple actions from human intervention. Multiple valid actions are very likely to conflict with each other and as a result can cause non-convergence during training, this is why it is necessary to keep such intervention separated from the main network.

Guided Learning eliminates the issues in points 1) and 2) by separating these taught experiences from the main network and encoding them as TRMs. Point 3) is handled via the simplification of SATRMs and potentially allowing multiple actions to be split into several SATRMs, thus removing any potential conflicting actions.

There are examples of reinforcement learning agents 'teaching' other reinforcement learning agents, agent-to-agent teaching, such as in Torrey and Taylor (2013). In the paper, expert agents 'suggest' actions for learning agents to take. Such a technique has the advantage of removing the human element and allowing the system to learn in a fully automated fashion, however it also requires an expert agent to be available in the first place. We will discuss how such agent-to-agent teaching could be implemented in Guided Learning in chapter 10.

Modern methods of teaching usually involve agent-to-agent (Taylor et al., 2014) or agent-to-human (Zhan et al., 2014) teaching methods. There is an example of human-to-agent teaching that uses human interaction, referred to as 'demonstrations', to estimate the *reward* function in Deep Q Networks (Hussein et al., 2017). This paper focuses mainly on optimising the reward function rather than the policy and in scenarios where the reward function is optimal, such techniques are irrelevant.

4.2 Reducing Stagnation in Supervised Learning

Due to the use of Supervised Learning in TRMs it may be useful to touch on some of the techniques used in reducing stagnation in these situations also. We will primarily focus on reducing stagnation when back-propagation is employed.

4.2.1 Conditions of Stagnation

There are many conditions that can arise during the learning process that can lead to stagnation. Not only this but the data itself can be a considerable factor in stagnation as discussed in Gori and Tesi (1992) where linearly separable data is preferred. This essentially eliminates any conflicting samples in the data. Initial weights can also cause stagnation initially as explained in Lee et al. (1993). This is one of the primary advantages of randomly initialising the weights of the network. Other conditions that that can cause stagnation are outlined in Vitela and Reifman (1997) which describes the momentum factor as playing a significant role.

4.2.2 Adaptive Learning Rate & Momentum

The most common and effective method of reducing stagnation while using back-propagation is to use an adaptive learning rate, α, that varies over time depending on the rate of progress being made. This avoids the situation of using too low a value for α where the gradient is low and too high a value for α when the gradient is high, as described in section 1.2. The concept of momentum (Sutskever et al., 2013) can also be used in an attempt to overcome local minima with a relatively low peak.

The learning rate may also be modified by using independent, weight specific learning rates that vary based on momentum and a purposed 'delta-bar-delta' rule in Jacobs (1988). Or vary based on the signs of the partial derivatives of the error function as purposed in Riedmiller and Braun (1993).

Chapter 5: **Methodology**

In this chapter we will discuss the methodology and implementation of both the baseline and the Guided Learning algorithm. I will begin by outlining the OpenAI Retro application and the NEAT algorithm, and then detailing the baseline and Guided Learning implementations respectively. Guided Learning will simply act as an extension to the baseline implementation. The results of the baseline will be used as the main performance comparison for Guided Learning in chapter 6.

5.1 OpenAI Retro

OpenAI Retro is a game console emulation application that provides OpenAIs Gym toolkit (Brockman et al., 2016). This project uses OpenAI Retro in order to emulate the Nintendo Entertainment System (NES) and test both the baseline and Guided Learning approaches inside of the Super Mario Bros game. The use of Retro significantly simplifies the implementation of the system by abstracting away the details involved in, for example, writing a plugin for a specific emulator. This allows more time to be dedicated toward the actual implementation of the project and is the main reason why Retro was chosen.

Using Retro, we can pragmatically give control input to the game and hence control the agent. Retro also gives access to an Application Programming Interface (API) that can be used to pragmatically save and load game states, access raw pixel data that will be used as the stimulus and control the frame rate of the game. Retro also allows for the reading of in-game memory in order to extract the values of variables in real time, provided their locations are known. This allows common definitions for things such as the 'done' condition and the fitness function to be easily defined using a JSON format and passed directly to Retro. For example, the fitness function used to evaluate each genome in this project was defined, in plain English as: 'Each time the internal variable of xScroll is incremented, increment the fitness value', where the 'xScroll' variable is how much the level has scrolled right. This essentially encapsulates the distance that the agent has moved right. The goal in Mario is to reach a flag pole at the furthest right side of each level so this is a natural fitness function to select.

5.2 Neuroevolution of Augmenting Topologies (NEAT)

NEAT (Stanley and Miikkulainen, 2002) is an Evolutionary Algorithm and is a specific type of TWEANN (see section 3.2.1). NEAT outlines the algorithm for how genomes should crossover, as well as speciation of the population based on differences in genome topology. It also has support for many different types of mutations such as weight and activation function mutations, adding/removing both links and nodes and mutations that activate and de-activate vertexes. NEAT also has support for forward, backward and recurrent vertexes and is widely regarded as the de facto standard for TWEANNs today.

Speciation in NEAT allows genomes of similar topology to be grouped together. This not only allows NEAT to protect what's referred to as 'topological innovation' but also to have its own internal methods of dealing with stagnation. By monitoring the progression of individual species over many generations, if the species remains stagnant for the given threshold of generations then that species is removed from the population. The effectiveness of NEAT will not be evaluated in this dissertation, instead Guided Learning will be built on top of it.

5.2.1 NEAT-Python

For the actual NEAT implementation, the decision was made to use the NEAT-Python library (McIntyre et al.). This library is a faithful implementation of the original NEAT algorithm. The NEAT-Python library was choosen because Retro provides a Python interface and the library itself is relatively mature.

5.2.2 Why NEAT?

The NEAT algorithm was chosen for use due to its simplicity of implementation and because applications of NEAT primarily led to the inspiration of this project (SethBling, 2015). Due to the nature of Guided Learning however, the use of NEAT is not a requirement and any other algorithm with reinforcement learning capabilities could be used in its place.

5.3 Baseline Implementation

In this section some implementation details are specific to Super Mario Bros but these are simply optimisations in order to ease the process of evaluation. An example of such specific details can be seen in the tiling section below.

5.3.1 Configuration

There is a timer for each evaluation such that if no progress is made over the duration of the timer (i.e. the genome fitness does not increase) then the current genome is terminated and its current fitness is recorded. This avoids unnecessary time wasting and allows other genomes to be evaluated in due time.

Some of the specific configuration details can be seen in table 5.1. An initial population size of 50 was used in order to balance timely evaluation and diversity in the initial population. The elitism value of 5 means that the top 5 genomes of each generation will be preserved and a max stagnation value of 20 means that if a species is stagnant for 20 generations, it is removed. This is NEATs default method of handling stagnation as described in 5.2.

Table 5.1: NEAT Configuration

Parameter	**Value**
Initial Population Size	50
Activation Function	Sigmoid
Activation Mutation Rate	0
Initial Weight/Bias Distribution Mean	0
Initial Weight/Bias Distribution Std. Deviation	1
Weight & Bias Max Value	30
Weight & Bias Min Value	-30
Weight Mutation Rate	0.5
Bias Mutation Rate	0.1
Node Add Probability	0.2
Node Delete Probability	0.1
Connection Add Probability	0.3
Connection Delete Probability	0.1
Initial number of Hidden Nodes	6
Max Stagnation	20
Elitism	5

5.3.2 Game Mechanics

It's worth noting that there are certain game mechanics that apply specifically to Super Mario Bros that must be considered. Firstly the act of jumping is not just a single action, this is because the height at which Mario jumps is dependant on how long the player holds down the corresponding jump button. This effectively makes jumping time-series dependant and as such, difficult to 'record' using our simple fully connected TRM networks. As a result, the duration of actions is also recorded and stored in each TRM. This helped to get responses that were relatively consistent with the received intervention. For details on how this can be remedied see chapter 10.

Not only this, but if the jump button is held during play then Mario will only jump once. This is problematic because even if TRMs that have been trained to jump get activated, Mario himself may not jump if the genome has a tendency to hold down the jump button. This quite often occurs in practise. However, no steps have been taken to mitigate this problem because it should be something that the agent self-learns over time as it is simply a mechanic of the game. As a result of this, it is not guaranteed that even good quality TRMs will lead to immediate improvement.

5.3.3 Input Modeling

Whilst most modern approaches to reinforcement learning in games tend to solely use raw pixel data as inputs (Mnih et al., 2015), for the purposes of this dissertation the decision was made to simplify the inputs in order to reduce the dimensionality of the data. This leads to significant time savings when it comes to training test models as the number of inputs, and as a result, the number of neurons and vertices in the network, is reduced. The pixel data, or stimulus as it is referred to in this dissertation, is sampled every 10th frame during game play.

Grayscaling

The first and most simplistic technique is to grayscale the given frame. This leads to significant dimensionality reduction, even when using raw input values as each RGB value in the frame is converted to a single value giving a factor of 3 reduction in the number of inputs. For example, the NES uses a native resolution of 224x240 and Retro gives us each pixel as a triple of RGB values. Taking the entire frame as input in RGB space would therefore result in $224 \times 240 \times 3 = 161,280$ total inputs. By grayscaling we instead get $224 \times 240 \times 1 = 53,760$ inputs. The equation for grayscaling an image, as defined by the CIE 1931 colour space (Smith and Guild, 1931), is seen in equation 5.1. An example of a grayscaled frame can be seen in figure 5.1(b).

$$grayscale = 0.2126R + 0.7152G + 0.0722B \tag{5.1}$$

Tiling

In the game, the level itself is made up of a collection of 16x16 pixel tiles. These tiles are often reused heavily throughout the level. This introduces a large amount of redundancy when we are including each pixel of each tile. As a result, each tile can be further reduced to a single value by averaging the grayscale values in the tile. If performed correctly each distinct tile will have a distinct value that will essentially act as a form of identifier for that specific tile. See figure 5.1(c) for a visual example of how the game screen is tiled.

In order to implement this correctly we need to understand that the camera moves across the level in a smooth manner, as Mario progresses through the level. This means that we will have to partially crop the borders of each frame in order to align them correctly with the 16x16 tiles. See figure 5.1(c) for a visual representation of this cropping process.

This tiling process will yield a 13x14 grid and because each tile is reduced to a single average grayscale value we get a further input tile reduction to $13 \times 14 = 182$ inputs.

Radius Tiles

Because we are mostly interested in a small area surrounding Mario, we can get an even further reduction by only taking the tiles that surround him into account. A radius parameter r was added as a configuration parameter that can be adjusted. For simplicity, the output is not circular but rather a square such that the area of the square is $(r * 2 + 1)^2$. With a radius r of 3 this yields a total input tile count of just 49. See fig 5.1(d) for an illustration of this.

At this point we have successfully reduced our screen input data from $161,280$ to just 49. A reduction of almost 4 orders of magnitude. This significant reduction in complexity saves training time because it reduces the search space dramatically by only focusing on the necessary data and reducing the overall size of each network dramatically. It's worth noting again that these measures were taken for the sole purposes of reducing the input dimensionality and these steps are not necessary for implementing the Guided Learning algorithm.

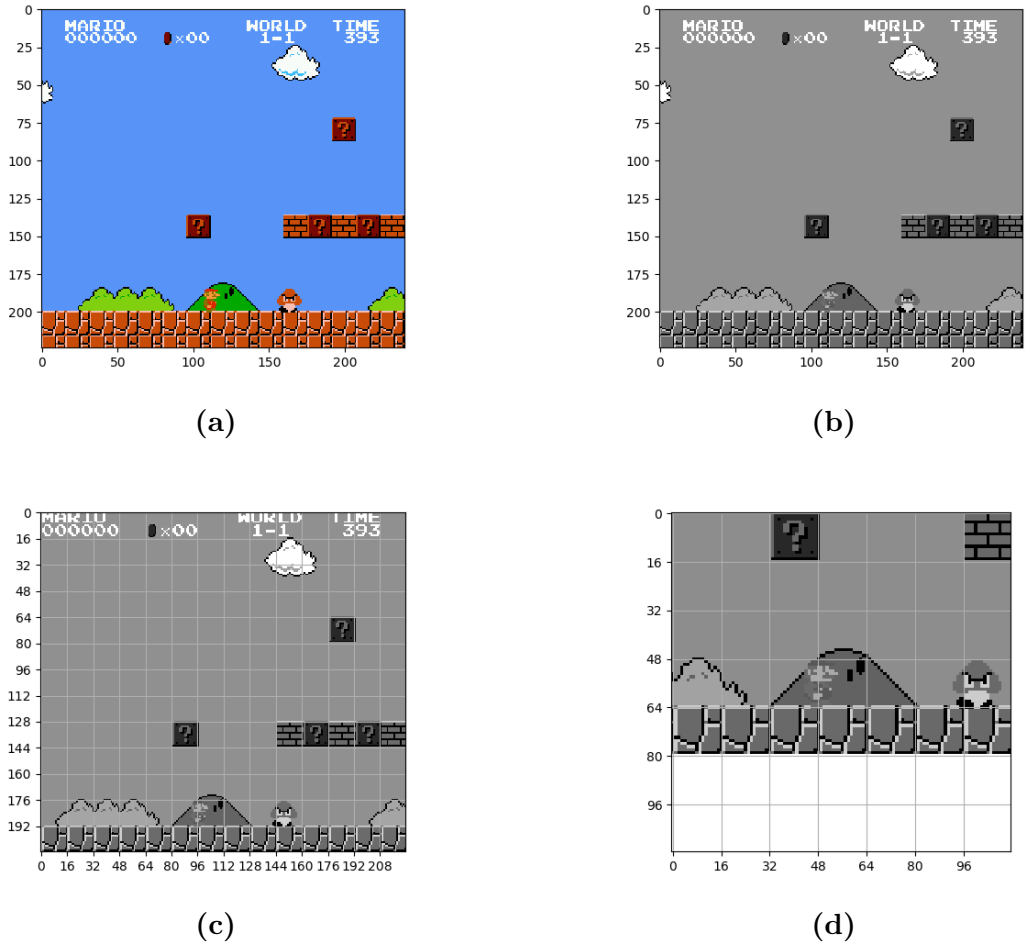

Figure 5.1: Input Reduction Pipeline Examples. **(a)** Raw RGB Frame **(b)** Grayscaled Frame **(c)** Aligned and Tiled Frame **(d)** Radius Tiles Surrounding Mario, $r = 3$

5.4 Guided Learning Implementation

After the baseline implementation has been built, we can start to work on implementing the actual Guided Learning algorithm. All the following improvements build upon the baseline implementation outlined previously. This section solely focuses on the specifics of implementing Guided Learning in this project, for a more general discussion and explanation of Guided Learning please see chapter 2

5.4.1 Human Intervention & TRMs

Stagnation

As described previously in section 2.3, we must have some form of definition for stagnation. For the purposes of this project, stagnation is defined as no progress being made (i.e. no increase in the highest/best fitness) over 3 generations. The current number of generations that have passed without progress are referred to as the 'stagnation count'. If this stagnation

count reaches 3 then the system will evaluate the current champion (i.e. the most fit) genome up to its max fitness minus a given parameter, as discussed below, and pause the emulator, waiting for keyboard input.

In the context of NEAT, the current fitness f of the system is taken from the fitness of the previous generations champion genome. Therefore we must evaluate n generations in order to determine stagnation. n is arbitrary and is determined at the discretion of the user but must be ≥ 2 due to determining stagnation based on the rate of change of fitness as described in section 2.3.

Taking Human Input

While the emulator is paused waiting for input, if no input is received within a given period of time, 8 seconds in our case, then a timeout will occur and the system will reset the stagnation count and begin evaluating the next generation. If however, keyboard input is received, then we start the emulator at a limited frame rate (30 fps). This is to allow the game to play smoothly and so that it is not excessively fast for the human giving the intervention. We then begin recording frames and their corresponding key presses. The user can then press a predefined key, 's', to signal that the correct input has been given to progress and stop the recording. If the user makes any errors, the process can be restarted at any point by pressing the 'r' key.

During the actual process of taking human input it is important to consider where the process should begin (i.e. what point the system 'asks for help'). For example we should not begin taking input at the maximum fitness value because at this point it will often be too late to change the agents outcome. So that leaves the question of how much of a buffer should be left before the maximum fitness is reached, before the system 'asks for help'? This value is referred to as the 'fitness buffer' and is very much dependant on context. For example, beginning human input at $max\_fitness - 50$ may work well for situations where Mario must jump over a green pipe, however, when applied to a situation where Mario must jump over a hole it may not leave sufficient platform space as seen in figure 5.2. This is because the genome can gain fitness as Mario falls through the hole. It's possible for this to be alleviated by adjusting the fitness function but adding such a fitness buffer gives extra flexibility. As a result, this value is left as a parameter to be tuned by the user.

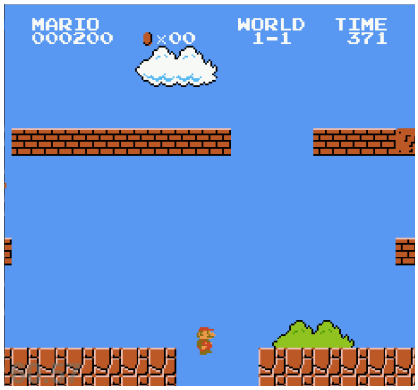

Figure 5.2: Insufficient platform space when 'asking for help' due to using too low a value for the fitness buffer.

TRM Similarity Function

The similarity function, $s(a, b)$, is used to compute the 'similarity' of two stimulus. The choice of similarity function can potentially depend on the application. For the purposes of

this dissertation, the similarity function used was the angle between the current input vector and each TRM initial input vector, the equation for which is shown in equation 5.2. Initially, the sum of the vector difference between the current input and each TRM initial input vector was used, however using the angle appears to give more reliable and reproducible results. The similarity threshold used for the angle difference was 0.015 radians. This figure was found via a process of trial-and-error that attempted to minimise false-positive TRM activation in favour of some false-negatives.

$$\theta = \arccos \frac{a.b}{|a||b|} \tag{5.2}$$

SATRMs

In the case of SATRMs, what we are mostly focused on testing in this project, we must insure that only a single action and its corresponding stimulus is recorded. This is easily accomplished by only taking the last key press of the interaction process. This allows Mario to begin moving forward at which point the user can initiate a jump, for example, at the correct moment, and press 's' to save the action and its corresponding stimulus and proceed to train the SATRM.

In the case of Super Mario Bros, more often than not, the action to take is a simple jump. The only issues are where the action should take place and how large the jump should be. This is one of the key game mechanics of Super Mario Bros as discussed in section 5.3.2 and makes the action of jumping time-series dependant. As a result, we add on an extra parameter d to the TRM that gives the duration of the action. This means that when a TRM is triggered, the inputs are forward propagated, and its output taken as the output of the current genome for d frames. This results in behaviour that is more consistent with the given human intervention. Usage of d could also be used in Multi-Action TRMS for more complex tasks.

Tensorflow & Keras

Keras, running a Tensorflow (Abadi et al., 2015) backend, was used to implement the TRM networks. A fully connected feed-forward network is trained using the previously recorded data. In order to attain the kind of plasticity of the network that was outlined in section 2.4.3, this Keras network is then converted to a NEAT genome prior to being stored as a TRM. This allows the TRM to be optimised in the reinforcement learning environment by using the standard NEAT mutation operations.

Due to the nature of the Keras network, its architecture needed to be designed. This process involved a lot of trial-and-error in order to achieve fast convergence. The use of SATRMS greatly eased this process because the use of a single sample eliminated any conflicting samples that would cause non-convergence. As a result, the used architecture consisted of a single hidden layer with 15 nodes. This architecture would likely need to be adjusted if one wanted to use multi action TRMs, or if Guided Learning was to be applied to a different game.

5.5 Metrics

Both the baseline and Guided Learning implementations will be evaluated using their mean average fitness and mean best fitness. These metrics are taken by running each algorithm a total of 50 times up to the 150th generation and taking the mean of the average fitness and top fitness for each generation across each run respectively. This allowed for moderately good averages to be taken within a timely fashion. Another key metric we will use for comparing the baseline with Guided Learning is the slope of the best-fit regression lines for multiple results. This gives a good indication of the overall rate of progression for both algorithms.

The individual genomes with the maximum and minimum regression slopes for their Best Fitness and Average Fitness results were found and analysed. These slopes give a good indication of particular genomes that performed well, in the case of highest slope, and genomes that performed poorly, in the case of lowest slope.

As our final metric we will use the mean number of generations it took to achieve various Best Fitness results. Due to the nature of Guided Learning and its focus on improving the most fit genome, we chose this metric to give a good overall summation of the effectiveness of the Guided Learning algorithm.

Chapter 6: **Results/Evaluation**

This chapter, will be compare the results achieved with the baseline implementation, explained in section 5.3, and the Guided Learning implementation, explained in section 5.4. In order to ensure consistency, the configuration and parameters outlined in section 5.3.1 will be used for both evaluations. For more in depth analysis and discussion of the results, see chapter 7.

6.1 Performance Comparison

Table 6.1 shows us the mean number of generations it took to achieve the given Best Fitness scores for both the baseline and Guided Learning. The table gives the corresponding improvement given by Guided Learning.

Table 6.1: Guided Learning vs. Baseline Mean Generations To Best Fitness

Best Fitness	**Baseline**	**Guided Learning**	**Improvement**
400	22	3	633%
600	48	24	100%
800	123	37	251%
1000	-	47	-
1200	-	65	-
1400	-	94	-

Table 6.2 gives the number of generations for the interval progression to be made. For example, it took the baseline algorithm 26 generations to move from a best fitness of 400 to 600 and it took Guided Learning 21 generations to do the same.

Table 6.2: Guided Learning vs. Baseline Mean Generations To Best Fitness Intervals

Best Fitness Interval	**Baseline**	**Guided Learning**	**Improvement**
0-400	22	3	633%
400-600	26	21	24%
600-800	75	13	477%

6.1.1 Mean Best Fitness & Mean Average Fitness

The results of the baseline and Guided Learning using Mean Average Fitness and Mean Best Fitness, can be seen in figure 6.1. This gives a good visual indication of Guided Learnings superior performance, for Best Fitness in particular, over the baseline.

The baseline shows a significant amount of stagnation initially. This stagnation occurs between between the 1st generation and the 20th generations for both the best fitness and

the average fitness scores. This is then followed by a significant improvement over about 7 generations followed by slower, but more consistent and continued improvement in top fitness.

For Guided Learning, while the initial stagnation seen in the best fitness for the baseline has been removed, there appears to be no real change in the stagnation occurring with the average fitness. This is due to the fact that Guided Learning inherently focuses mainly on guiding the champion genome. That being said there is improvement over the baseline in terms of rate of progression.

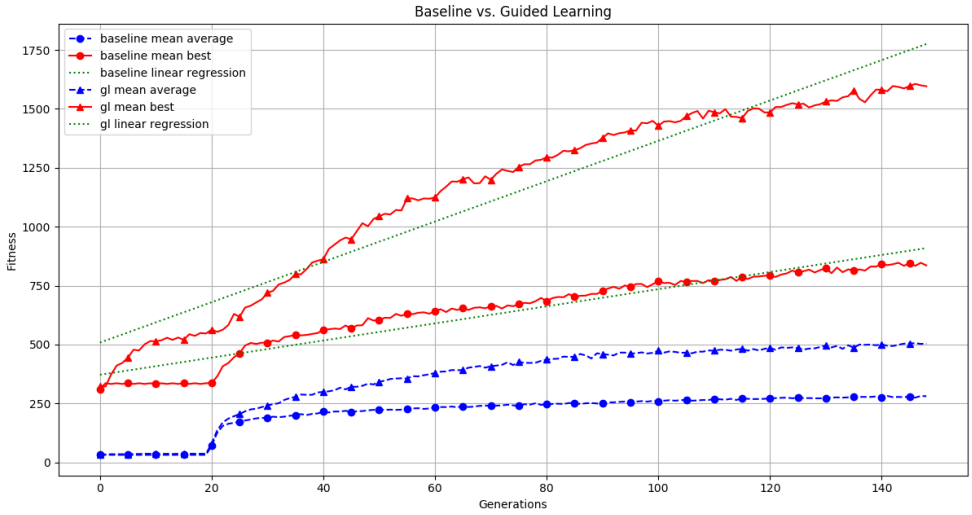

Figure 6.1: Baseline vs. Guided Learning Evaluation Results. Higher is better.

Analysing the slope of the regression lines, figure 6.2 and table 6.3 give the slope comparison for Mean Best Fitness and Mean Average Fitness.

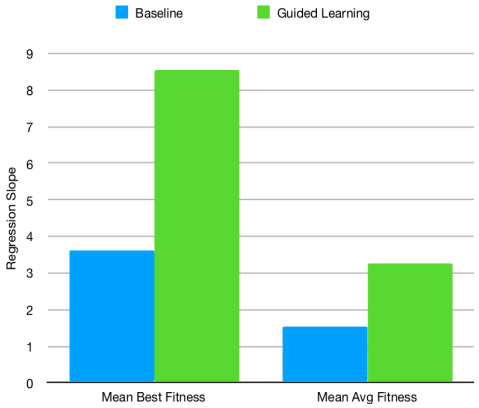

Figure 6.2: Baseline vs. Guided Learning Mean Slope Evaluation Results. Higher is better.

Table 6.3: Mean Best Fit Regression Slopes

	Baseline	**Guided Learning**	**Improvement**
Mean Best Fitness	3.63	8.56	136%
Mean Avg Fitness	1.53	3.25	112%

6.1.2 Best and Worst Performing Cases

Comparison of individual genomes with the highest and lowest regression slopes respectively can be seen in figure 6.3 and figure 6.4. The horizontal lines for the guided learning graphs indicate the points at which human intervention was given.

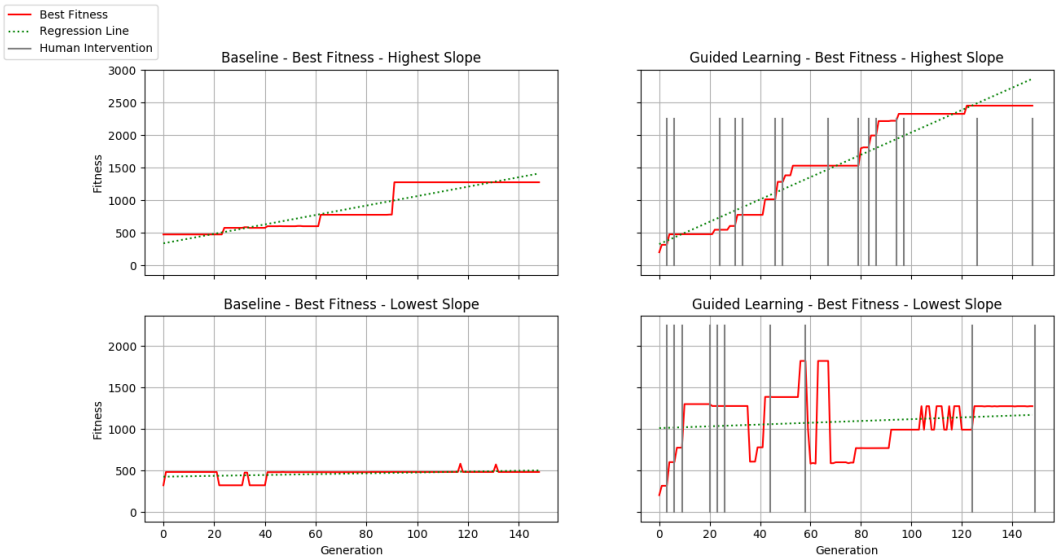

Figure 6.3: Baseline vs. Guided Learning Best Fitness Slope Results. Higher is better.

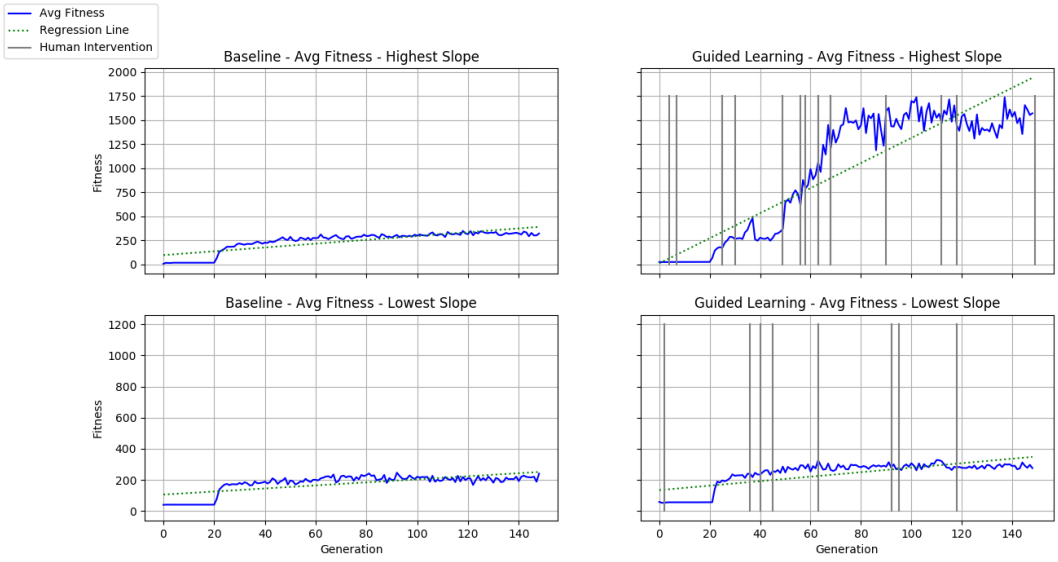

Figure 6.4: Baseline vs. Guided Learning Avg Fitness Slope Results. Higher is better.

As before with Mean Best Fitness and Mean Average Fitness, looking at the figures for these slopes yields figure 6.5 and table 6.4.

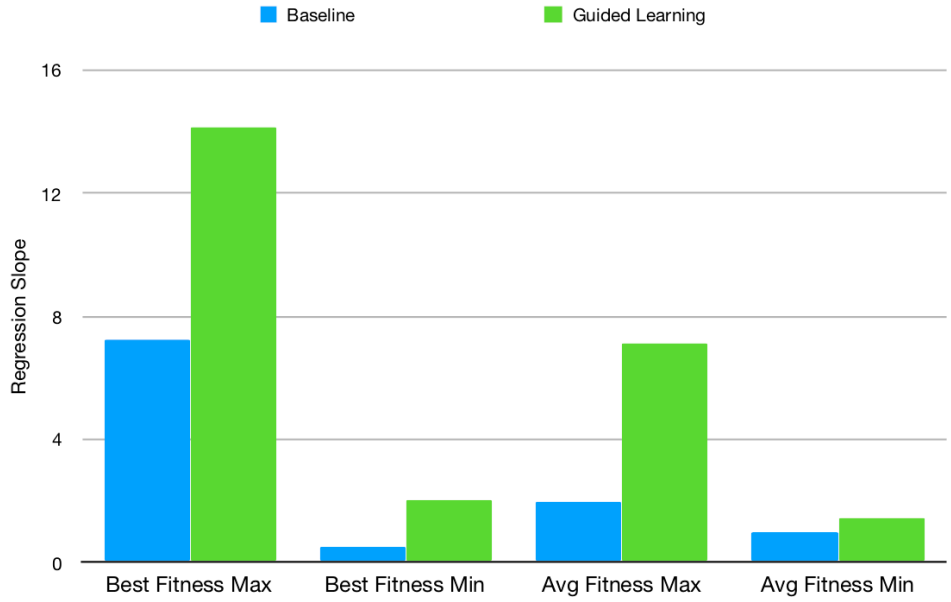

Figure 6.5: Best Fit Regression Slope Comparison

Table 6.4: Min-Max Best Fit Regression Slopes

	Baseline	**Guided Learning**	**Improvement**
Best Fitness (Max Slope)	7.25	17.16	137%
Best Fitness (Min Slope)	0.51	1.07	110%
Avg Fitness (Max Slope)	1.98	13.03	558%
Avg Fitness (Min Slope)	0.98	1.44	47%

Table 6.5 shows the max best fitness achieved at various generations. These results can be considered best-case results.

Table 6.5: Best Fitness Achieved

Generation	**Baseline**	**Guided Learning**	**Improvement**
50	783	1878	140%
100	1397	2330	67%
150	1287	2456	91%

Chapter 7: **Discussion/Interpretation**

Guided Learning has shown to outperform the baseline on every metric presented in chapter 6. The Mean Average and Mean Best results in section 6.1.1 gave a good overall representation of the extent to which Guided Learning outperformed the baseline in this particular application. It's important to remember however that Guided Learning requires human intervention in order to be effective. To put this into context, over 150 generations, an average of 10 interventions were given to each genome over a period of about 8 hours in total. Interventions were not given at each opportunity presented and were instead lazily applied. To put this into context, the average number of times an agent requested help was about 30, so human intervention was given about 1/3 of the time it was requested.

7.1 Run Time

Overall run time should also be considered. For the baseline, one generation was evaluated on average every two minutes. Whereas it took Guided Learning, on average, about four minutes to evaluate each generation. This is mainly due to the added overhead of checking each frame for a corresponding TRM. This could be reduced significantly by reducing the 'similarity threshold' when identifying compatible TRMs and allowing the similarity evaluation to occur every couple of frames as opposed to every frame, however, this could have potentially introduced false-positives that could have impacted on performance. Although statistics were not obtained for a false-negative:false-positive TRM activation ratio, very few of either were encountered/recognised during the evaluation process with the implementation outlined in 5.4.1. It was preferred to obtain an accurate representation of Guided Learnings potential and *then* focus on optimisation of the algorithm rather than obtaining results that were not representative. Unfortunately time did not allow for such an endeavour.

7.2 Fitness Decline

It is not uncommon for the Best Fitness score for an individual genome to decline during the evaluation process. This occurs for both the baseline and Guided Learning implementations. A particular example of such an occurrence can be seen in the lowest slope result for Guided Learning in figure 6.3. This can be particularly devastating when the top genome performs under the NEAT elitism threshold where the genome may be effectively removed from the population. Such an occurrence appears as though it may have happened in this particular case, on multiple occasions.

The reason for such occurrences is due to instability and divergence that occurs when using neural networks to estimate the value function in reinforcement learning. This is because a small change in the network can lead to significant changes in its responses to stimulus. The use of Experience Replay was successfully used to combat this issue for Deep Q-Learning in (Mnih et al., 2015). We will discuss this in further detail in chapter 10.

7.3 Algorithm Independence

One of the best advantages of Guided Learning is its algorithm independence. It does not require a specific algorithm to be used in order to be implemented. This is because TRMs are treated as completely independent entities. For example, we could model both our agent and TRMs using the Deep Q-Learning techniques outlined in Mnih et al. (2015) if we were so inclined. Equally, the agent and TRMs could be modelled using any other combination of a reinforcement learning algorithm alongside a supervised learning algorithm, including other neural networks such as LSTM networks or Neural Turing Machines (Collier and Beel, 2018), provided the instability and divergence issues, described above, are taken into consideration. Any combination of algorithms will work provided that the agent algorithm can be used in a reinforcement learning environment and the TRM algorithm can be used for supervised learning tasks.

Chapter 8: **Conclusion**

In this dissertation, a system has been created that effectively requests help from a human supervisor when it encounters stagnation over a given period of time. In the system, such intervention is optional and a novel method of encoding such interventions separately and independently has been explored and the results of which have been given in chapter 6.

The results achieved in this dissertation have shown that human intervention can successfully be used to reduce stagnation in the training of agents in reinforcement learning applications. We have also shown that minimal human intervention can, as a result of reduced stagnation, over double the rate of progression of a reinforcement learning agent. The extent of this acceleration and stagnation reduction has been shown in the results given in chapter 6 and their discussion in chapter 7.

In hindsight, given the issues with using neural networks in reinforcement learning, namely instability and divergence when estimating the value function, use of the NEAT algorithm is not in any way optimal. Instead it would have been a better decision to have used the Deep Q-Learning methods described in Mnih et al. (2015) and built Guided Learning on top of this instead. This way it would also have been possible to do a direct comparison of their results and Guided Learning.

Due to its algorithm independence, Guided Learning can be applied to any combination of algorithms, provided that the algorithm used to model the agent can be used in a reinforcement learning context and the TRM algorithm supports supervised learning. In this dissertation we also assume that the system is being used in a context where human guidance is possible.

Chapter 9: **Summary**

In summary, this dissertation has:

- Introduced the problem of stagnation in reinforcement learning environments and why it occurs, the goal of this dissertation as reducing stagnation by using human guidance and a novel approach of encoding such guidance.
- Discussed the vision of Guided Learning, its rationale, the concept of Taught Response Memories (TRMs) and the unique aspects that Guided Learning brings.
- Given background information pertaining to reinforcement learning algorithms, including Q-Learning and Deep Reinforcement Learning and the combination of artificial neural networks and evolutionary algorithms.
- Shown other related work in terms of techniques that currently exist for reducing stagnation in reinforcement learning and supervised learning including Experience Replay, Action Models and Teaching.
- Discussed the methodology used for the implementation of both the baseline algorithm and Guided Learning using OpenAI Retro and the NEAT algorithm.
- Presented the results of evaluation, where Guided Learning outperformed the baseline on all metrics presented, including Best Fitness, Average Fitness and the slope of the best-fit line, used to approximate the rate of progression, for both best and worst case scenarios.
- Presented a discussion and interpretation of the results where the issues of run time, instability and divergence are discussed including some of the advantages of Guided Learning such as algorithm independence.
- Concluded by discussing how the use of other algorithms, particularly Mnih et al. (2015) would have, in hindsight, been a better than NEAT for the purposes of a state of the art comparison.

Chapter 10: **Future Work & Limitations**

Building Guided Learning directly on top of more recent state of the art methods, such as those proposed in Mnih et al. (2015) and evaluating against those results would be an important priority as future work.

Guided Learning is also limited in the why that it handles human intervention that negatively affects progress. Currently, the agent will not consider if a TRM makes a positive or negative impact overall. As such, for particularly poor TRMs, it may advantageous to use them to show the agent what *not* to do under those scenarios. Such ideas are outlined in Lin (1992).

Currently, Guided Learning is limited in the way it can re-use TRMs in other scenarios than the scenario for which it was trained. In section 2.1 I outlined a scenario where the agent requested assistance to jump over the first green pipe and had envisioned that the subsequent TRM from successful human intervention could then be re-used in order to jump the second and third pipes, and so on. However, the necessary balance between false-negative and false-positive activation of TRMs made this unfeasible with the current method of input modelling. In order for such a situation to be feasible, we would require an abstract representation of the stimulus. An example of such a representation would be replacing input pixels with generic representations of barriers and enemies. Rather than a programmer specifying such details, it might be possible to classify and learn such objects from a combination raw pixel data and exploration. Not only this, but if one was to apply TRMs to other games the potential difference in input and output dimensionality would have to be considered.

In terms of the use of 'Single Action Taught Response Memories' (SATRMs), while this may appear initially as a disadvantage, it would be possible to encode a multi-action TRM as multiple SATRMs by taking their single action components. Thus maintaining all the advantages of SATRMs with none of the disadvantages of TRMs, particularly non-convergence due to conflicting actions. Unfortunately such techniques could not be explored and are left as future work.

Some of the implementation details discussed in this dissertation, particularly the recording of the duration of actions in TRMs, are only as a simplifications, and are not deemed to be ideal. The use of action duration could be completely removed by implementing TRMs using recurrent neural networks, however, such testing was out of scope for this project and as a result, is left as possible future work.

It is also important to note some of the optimisations that could be taken to help to improve Guided Learnings run time. As discussed in section 7.1, reducing the sample rate at which TRMs are compared to the current stimulus would be one way of doing so, at the risk of increasing the rate of false-negatives. Another way of optimising Guided Learning would be to look at the current similarity of a TRM and estimate how many frames, x, it may take between now and until that TRM becomes relevant and only evaluate that TRM when x frames have passed, once again, this would risk increasing the number of false-negative activations.

During this dissertation human-to-agent, agent-to-human and agent-to-agent teaching have been discussed, where Guided Learning has only been considered human-to-agent teaching. This is seen as a limitation as it requires a human supervisor to be present. However, it should be possible to expand Guided Learning to also encompass agent-to-agent based teaching that would allow expert agents to guide learning agents in a fully automated manner. Such a task is also left as future work.

Disclaimer: The ROM used during the creation of this work was created as an archival backup from a genuine NES cartridge and was NOT downloaded/distributed over the internet.

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

H. Asada and S. Liu. Transfer of human skills to neural net robot controllers. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2442–2448. IEEE, 1991.

H. Braun and J. Weisbrod. Evolving neural feedforward networks. In *Artificial Neural Nets and Genetic Algorithms*, pages 25–32. Springer, 1993.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. Deep blue. *Artificial intelligence*, 134(1-2): 57–83, 2002.

C.-C. Cheung, A. K. Lui, and S. S. Xu. Solving the local minimum and flat-spot problem by modifying wrong outputs for feed-forward neural networks. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2013a.

C.-C. Cheung, S.-C. Ng, A. K. Lui, and S. S. Xu. A new fast learning algorithm with promising global convergence capability for feed-forward neural networks. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2013b.

J. A. Clouse and P. E. Utgoff. A teaching method for reinforcement learning. In *Machine Learning Proceedings 1992*, pages 92–101. Elsevier, 1992.

M. Collier and J. Beel. Implementing neural turing machines. In *International Conference on Artificial Neural Networks*, pages 94–104. Springer, 2018.

Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2017.

V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4): 219–354, 2018.

N. Franken and A. P. Engelbrecht. Evolving intelligent game-playing agents. In *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 102–110. South African Institute for Computer Scientists and Information Technologists, 2003.

D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955. IEEE, 2017.

M. Gori and A. Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (1):76–86, 1992.

A. Guez and J. Selinsky. A neuromorphic controller with a human teacher. In *IEEE International Conference on Neural Networks*, volume 2, pages 595–602, 1988.

F. S. He, Y. Liu, A. G. Schwing, and J. Peng. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *arXiv preprint arXiv:1611.01606*, 2016.

T. Hester, M. Quinlan, and P. Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *2010 IEEE International Conference on Robotics and Automation*, pages 2369–2374. IEEE, 2010.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

A. Hussein, E. Elyan, M. M. Gaber, and C. Jayne. Deep reward shaping from demonstrations. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 510–517. IEEE, 2017.

R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

G. A. Kimble. Learning theory, 2016. URL `https://www.britannica.com/science/learning-theory`. [Online; accessed Mar 30, 2016].

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Y. Lee, S.-H. Oh, and M. W. Kim. An analysis of premature saturation in back propagation learning. *Neural networks*, 6(5):719–728, 1993.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Y. Li. Deep reinforcement learning. *arXiv preprint arXiv:1810.06339*, 2018.

Z. Li and D. Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

L. J. Lin. Programming robots using reinforcement learning and teaching. In *AAAI*, pages 781–786, 1991.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

T. Matiisen. Demystifying deep reinforcement learning, 2015. URL `https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/`. [Online; accessed Mar 12, 2019].

M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

A. McIntyre, M. Kallada, C. G. Miguel, and C. F. da Silva. neat-python. `https://github.com/CodeReclaimers/neat-python`.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

P. Moallem and S. A. Ayoughi. Removing potential flat spots on error surface of multilayer perceptron (mlp) neural networks. *International Journal of Computer Mathematics*, 88(1): 21–36, 2011.

M. C. Nechyba and Y. Xu. Human skill transfer: neural networks as learners and teachers. In *iros*, page 3314. IEEE, 1995.

X. Pan, Y. You, Z. Wang, and C. Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

A. P. Piotrowski. Differential evolution algorithms applied to neural network training suffer from stagnation. *Applied Soft Computing*, 21:382–406, 2014.

N. Prasad, R. Singh, and S. P. Lal. Comparison of back propagation and resilient propagation algorithm for spam classification. In *2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation*, pages 29–34. IEEE, 2013.

R. Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proceedings of the IEEE international conference on neural networks*, volume 1993, pages 586–591. San Francisco, 1993.

S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

A. L. Samuel. Some studies in machine learning using the game of checkers. iirecent progress. In *Computer Games I*, pages 366–400. Springer, 1988.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

C. Scheepers and A. P. Engelbrecht. Analysis of stagnation behaviour of competitive coevolutionary trained neuro-controllers. In *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, pages 1–8. IEEE, 2014.

SethBling. Mari/o - machine learning for video games, 2015. URL `https://www.youtube.com/watch?v=qv6UVOQ0F44`. [Online; accessed Sep 20, 2018].

F. M. Silva and L. B. Almeida. Acceleration techniques for the backpropagation algorithm. In *Neural Networks*, pages 110–119. Springer, 1990.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

T. Smith and J. Guild. The cie colorimetric standards and their use. *Transactions of the optical society*, 33(3):73, 1931.

K. O. Stanley. Neuroevolution: A different kind of deep learning, 2017. URL `https://www.oreilly.com/ideas/neuroevolution-a-different-kind-of-deep-learning`. [Online; accessed Nov 1, 2018].

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE transactions on evolutionary computation*, 9(6):653–668, 2005.

K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

S. G. Sterrett. Bringing up turings child-machine. In *Conference on Computability in Europe*, pages 703–713. Springer, 2012.

StuartReid. 10 misconceptions about neural networks, 2014. URL `http://www.turingfinance.com/wp-content/uploads/2014/05/Early-Stopping-300x247.png`. [Online; accessed Oct 30, 2018].

I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28(1139-1147):5, 2013.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. Elsevier, 1990.

M. E. Taylor, N. Carboni, A. Fachantidis, I. Vlahavas, and L. Torrey. Reinforcement learning agents providing advice in complex video games. *Connection Science*, 26(1):45–63, 2014.

L. Torrey and M. Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1053–1060. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

A. M. Turing. Intelligent machinery, a heretical theory. *Philosophia Mathematica*, 4(3): 256–260, 1996.

A. M. Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer, 2009.

A. Van Ooyen and B. Nienhuis. Improving the convergence of the back-propagation algorithm. *Neural networks*, 5(3):465–471, 1992.

J. E. Vitela and J. Reifman. Premature saturation in backpropagation networks: mechanism and necessary conditions. *Neural Networks*, 10(4):721–735, 1997.

C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

R. Weber et al. On the gittins index for multiarmed bandits. *The Annals of Applied Probability*, 2(4):1024–1033, 1992.

X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

C.-C. Yu and B.-D. Liu. A backpropagation algorithm with adaptive learning rate and momentum coefficient. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, volume 2, pages 1218–1223. IEEE, 2002.

Y. Zhan, A. Fachantidis, I. Vlahavas, and M. E. Taylor. Agents teaching humans in reinforcement learning tasks. In *Proceedings of the Adaptive and Learning Agents Workshop (AAMAS)*, 2014.

Y. Zhang and C. Liu. Accelerate deep q-network learning by n-step backup. In *Proceedings of the 2nd International Conference on Innovation in Artificial Intelligence*, pages 22–27. ACM, 2018.