# Combining Recent Advances in Reinforcement Learning in Super Mario Bros.

Student Name: Max Woolterton

Supervisor Name: Stefan Dantchev

Submitted as part of the degree of Msci Natural Sciences to the

Board of Examiners in the Department of Computer Sciences, Durham University

***Abstract* — Context:** A variety of, predominantly Reinforcement Learning (RL) based, approaches have been applied over the years to creating an intelligent agent able to play the platformer game Super Mario Bros. (SMB), however, the majority are heavily reliant upon expert feature-engineering and low-dimensional state-encodings to make learning tractable without the use of Neural Networks. This application has been largely untouched by any recent state-of-the-art RL contributions.

**Aims:** To investigate how best to apply and combine recent advances in deep RL so as to create a highly proficient agent capable of teaching itself to play SMB to human-level performance or above using raw-image-data without any feature-engineering.

**Method:** After surveying the field, a Recurrent Replay Deeper Denser Distributed DQN+ (R2D4+) (as named and defined within this paper) agent has been created by attempting to combine the constituent components of recent RL architectures, namely R2D2, Rainbow, D2RL and IQN, alongside some novel contributions, such as dropout, and an improved exploration scheme.

**Results:** The final agent is highly proficient, surpassing all previously surveyed agents, and converges to or surpasses the skill and speed of an expert on a variety of level difficulties and training-set sizes. The agent also generalizes impressively well to levels on which it has never been trained.

**Conclusions:** All surveyed contributions, apart from PER and IQN, are shown to be complementary in the context of training a RL agent to play SMB, which provides evidence that further investigation and combination of additional contributions would further advance the RL field. The novel use of dropout to boost generalization power and sample-efficiency is especially promising. Overall, deep RL in its current state can produce self-teaching intelligent agents capable of exceeding human-level performance in applications such as video games and motor control, and is still a field of much promise for the future.

***Keywords* —** Reinforcement Learning, Super Mario Bros., R2D4+, Deep Q-Learning, Machine Learning, Deep Learning, Video Games, Artificial Intelligence, Dropout, R2D2, Rainbow, D2RL, IQN.

## I INTRODUCTION

Game playing, whether it be board games, card games or video games, has always been a large application of Machine Learning (ML), wherein some form of Artificial Intelligence (AI) agent is designed to play a game of sorts. Traditionally, these agents, frequently called NPCs (non-player characters) or bots, are often rule-based and require extensive expert engineering and many hours of delicate design to create skilled and natural AI. Reinforcement Learning (RL), inspired by behaviorist psychology (behaviorism), is one of the three fundamental ML paradigms (Sutton and Barto, 1998), wherein agents are trained to behave optimally in an environment with respect to a given reward signal. In contrast to rule-based agents, which must be carefully engineered by their designers on a game-by-game or even level-by-level basis, RL agents, given a reward

function, are able to teach themselves to play any game by aiming to maximize their reward gained over time (Sutton and Barto, 1998). Therefore, it comes as no surprise that video games have been one of the biggest and also first successes of RL, and that the rapid development of the video game industry has caused a steadily increasing interest in the field of RL in recent years. The most popular algorithm in RL is called Q-Learning (Watkins and Dayan, 1992) and since its conception in 1992 it has been the subject of countless studies and extensions.

Super Mario Bros. (SMB) is a video game released by Nintendo in 1985 for the Nintendo Entertainment System (NES), wherein the main character is a plumber in red overalls named Mario. Specifically, SMB is a platformer, which is a genre of game where the aim is to navigate through and reach the end of a level consisting of obstacles, traps and enemies without dying. A player's skill in such a game is predominantly determined by their ability to complete difficult levels, along with how fast they can do so. See Tsay et al. (2011) for further description of SMB.

There have been many ML and RL approaches applied in the past to the creation of AI capable of playing SMB, a number of which are surveyed in §II. There are also many additional agents that were created for the yearly Mario AI competitions held between 2009 and 2012 (Togelius et al., 2013), which generated much interest in the field. However, what the vast majority of these approaches have in common is that they choose to represent the state using a custom handmade low-dimensional state-encoding to vastly reduce the size of the state-space, as it was viewed infeasible to learn directly from the raw-image-data. This is because many of these attempts were before the birth of deep RL, predominantly brought about by Mnih et al.'s (2013) DQN algorithm. This is what this paper approaches differently, and tries to apply the latest state-of-the-art (SOTA) deep RL to train an agent from raw-image-data.

The aim of this paper is to investigate the effects of combining some of the latest SOTA deep RL techniques and algorithms into a singular highly proficient SMB AI agent able to perform similarly to or out-perform human experts on a variety of tests. This is not as simple as designing and combining components, as not all algorithms are compatible, nor are their benefits necessarily orthogonal. To this end, a secondary goal of this project is to develop and display a deep understanding of the resultant agent and its ability to learn, predominantly demonstrated by breaking down its performance into that provided by each of its constituent components via ablation studies. The agent's ability to generalize to unseen problems is also of great interest, and therefore this will also be assessed, and broken down via ablation studies to offer an insight into the contributions of each component, while also ensuring that an apparent increase in the agent's ability is not simply an increase in over-fitting. The project's success and the final agent's ability will both be evaluated using a list of deliverables presented in §V. These deliverables assess the agent's skill, generalizability, memory and speed, alongside the understanding gained and demonstrated during the project itself.

At a higher level, this paper brings together the findings of R2D2 (Kapturowski et al., 2019), Rainbow (Hessel et al., 2017), D2RL (Sinha et al., 2020) and IQN (Dabney et al., 2018) (alongside some novel ideas), as many of the contributions this paper collates are included in either Rainbow or R2D2, with the most notable exceptions being the proposals of D2RL and IQN. This paper also clears up and/or improves many of the missing implementation details from R2D2, which sometimes took weeks to fix/determine. For additional clarification beyond the details given in this paper, please see the accompanying code-base.

The final agent produced, Recurrent Replay Deeper Denser Distributed DQN+ (R2D4+), is able to learn to complete almost any level, regardless of its difficulty, while also being able to

compete with and often beat a small sample of experts on speed-running times to complete a variety of levels. R2D4+ is also able to out-perform all surveyed RL agents, for example R2D2, Rainbow and IQN. Unfortunately, the agent is not able to learn to escape the infinite room loops in SMB's later levels. Extensive ablation studies for each component are also shown, allowing the reader to easily see how important each component is to R2D4+. Overall, a highly successful agent was produced, alongside valuable insights into the compatibility of various recent RL contributions. For example, IQN and Dueling (Wang et al., 2016) architectures are highly incompatible, and the now standard PER (Schaul et al., 2016) may not always be beneficial in cases of dense rewards. Additionally, some novel findings are presented, for example, the use of dropout may increase the sample-efficiency and generalization power of an agent, which, in contrast, is generally believed to make learning more difficult and thus is never incorporated in RL.

## II RELATED WORK

A wide variety of approaches have been applied to tackle the creation of AI agents in Mario over the years, from traditional Q-Learning to the Genetic Algorithm, with a number of hybrids in between, with fields such as Imitation and Apprenticeship Learning.

Traditional Q-Learning (Watkins and Dayan, 1992), where Q-values are stored in a Q-table with an entry for every possible state-action pair, is one of the most popular approaches, especially when combined with other techniques such as Hierarchical Reinforcement Learning or Deep Learning and a hand-crafted state representation in order to overcome some limitations of the original Q-Learning algorithm.

The notable exceptions are the genetic algorithms studied by Jørgensen and Sandberg (2009) and Du et al. (2009) and the fascinating experiments by Lee et al. (2014) into applying Apprenticeship Learning via Inverse Reinforcement Learning through the use of a reward function designed to provide a reward signal for behaving in a human-like fashion.

Liao et al. (2012) was the only surveyed work to apply traditional Q-Learning by itself, training an agent extensively on one low-difficulty level, resulting in a final win rate of 85%. Jørgensen and Sandberg (2009) made a very early attempt at a Deep Reinforcement Learning algorithm by approximating the Q-values using a Neural Network. Unfortunately, this agent struggled to learn in complex environments, but was shown to perform well in custom levels consisting only of 1 type of enemies/obstacle, such as only-Goombas or only-pitfalls, thus it would appear it was an issue of network capacity. Subsequently, a Genetic Algorithm rule-based agent was developed, and the strengths and weaknesses of these two solutions are compared. Du et al. (2009) took a very similar approach, except they compared their Genetic Algorithm with a Markov Decision Process (Puterman, 1994) based value iteration algorithm, quite similar to Q-Learning, except when using the Bellman equation, all other states' values are used rather than just the next visited state. Both papers arrive at very similar conclusions, namely that the Genetic Algorithm based agents struggle with generalization beyond the exact level-layout they were trained upon and the value iteration and Q-Learning algorithms are hindered greatly by the complexity and dimensionality of the state-action space.

In an attempt to reduce the size of the state-action space Mohan and Laird (2009) and Mohan and Laird (2010) apply Hierarchical Reinforcement Learning to transform the action space from individual button combinations into high-level actions, such as grab coin, tackle monster, and move right, each with their own sub-goal reward function (e.g. coins grabbed, monsters killed, distance moved...). Although this approach is almost identical to that of Tsay et al. (2011) (the

best surveyed agent, as described below), this agent performs considerably worse due to its poor relative quality of state representation chosen. Joshi et al. (2012) investigates restraining the state-action space further by applying an object-oriented state representation on top of Hierarchical Reinforcement Learning, and shows that the benefits of these two methods are orthogonal, while also investigating the learning and performance effects of constraining the state space via different levels and methods of abstraction.

In the majority of the surveyed work, and as is explicitly stated by Liao et al. (2012), one of the biggest challenges faced when designing an agent is defining the state, action and reward spaces. Designing the state space alone is the single largest factor in determining the relative success or failure of an agent's performance and ability to learn. This is due to the constant battle between the curse of dimensionality, introduced by the use of an information-rich state space, and the loss of important information caused by any attempt to counter this curse.

It therefore comes at no surprise that the best agent surveyed was the agent that used the smallest action-state space of 3072, Tsay et al. (2011). This agent used traditional Q-Learning on a state representation consisting of features such as "is there an enemy in front of Mario?" and "can Mario jump?". In addition, similarly to Mohan and Laird (2009, 2010) and Joshi et al. (2012), Hierarchical Reinforcement Learning was applied to transform the typical button-based actions into 4 higher level tasks: attack, upgrade, pass-through, avoid. This agent was able to complete 30 levels in the 2009 Mario AI competition (Togelius et al., 2010) benchmark and would have placed 4th, losing only to the winning A* algorithms, while performing substantially better than any other online-learning agent, the best of which completed only 3 levels.

The works of Tsay et al. (2011) were later expanded upon and combined with Imitation Learning by Lindberg (2014) to explore the possibility of creating a high performing agent that kept the tendencies and behavior patterns of the expert from which it was trained. Although there were some promising results, it is notable that much of the performance of the original agent was sacrificed, with their best agent only able to complete 4 levels in the aforementioned benchmark.

Furthermore, Tsay et al. (2011) later compared their original agent to an altered agent in which all enemies were no longer treated as the same, allowing the agent to, for example, differentiate between enemies that could be killed by stomping and those that doing so would result in death for the player. For example, this meant that when the agent encountered a "spiky" enemy it could learn to avoid it, rather than be encouraged to jump on it, expecting to receive a reward, like with any other enemy, and instead die, without being able to learn why. The resultant state space's size was 18432, a demonstration of how affected the size of the state space is by a small increase in information. Unfortunately, due to how susceptible traditional Q-Learning is to large state spaces, this increase in, what many would call vital, information overall hindered the agent's ability to learn as successfully as its smaller cousin; this is despite the state space still being many magnitudes smaller than some others, such as Liao et al.'s (2012) $2^{39}$ ($= 18432 \times 3{\times}10^7$), and Lee et al.'s (2014) $15^{23}$ ($= 18432 \times 6{\times}10^{22}$). However, it is of note that the state-action space is always sparse in the above works, as most states never get visited.

In contrast, the agent we propose learns in an action-state space of $14 \times 255^{84\times84\times4}$, which is a number that to write down in this paper would take approximately 17 pages, coincidentally the rest of this paper. This state space is the same as those used by the 2 most recent works surveyed in this paper (Klein, 2016; Heinz, 2019), as a stack of 4 downsampled $84{\times}84$ grayscale images has been standard in Deep Reinforcement Learning since Mnih et al. (2013). The choice of supplying the agent with an image, rather than a low-dimensional expertly human-designed state

representation, was made as it is in line with the aims and methodology/mindset of this paper, namely to design an agent that is capable of learning just as a human would, with no additional expert information or feature-engineering. Additionally, it has already been shown in various of the above literature that a highly proficient Mario controller can easily learn via traditional Q-Learning when supplied with a well-chosen state representation, however Super Mario Bros. has so far remained entirely untouched by the recent developments of the SOTA in Deep Q-Learning since the contributions of van Hasselt et al. (2015) (Deep Double Q-Learning).

Both Klein (2016) and Heinz (2019) directly mimic the works of van Hasselt et al. (2015) to create a single actor Deep Double Q-Learning agent using standard practices introduced by Mnih et al. (2013) such as frame-skipping, 4 frame-stacking, storing previous transitions in an Experience Replay buffer (Lin, 1993) and reward clipping. The only differences are that the standard max-pooling operation over the frame stack is excluded in Klein (2016), and that Klein (2016) uses the, now unavailable, 2009 Mario AI competition's (Togelius et al., 2010) benchmarking version of Infinite Mario (Super Mario Bros. with random level generation), whereas Heinz (2019) uses, like us, Kauten's (2018) implementation of Nintendo's original Super Mario Bros. NES (gym-super-mario-bros). Heinz's (2019) agent was only ever trained on one level, and performs excellently, with its speedrun time ranking above all but 1 human expert in evaluations, who used a secret shortcut to beat the AI. However, Klein's (2016) agent was trained on multiple levels sequentially (i.e. to reach level 1-2, Mario must first complete 1-1 each time etc.), but was unable to perform well past level 1-1, which it was able to complete with a mediocre time. The difference in performance between these two seemingly almost identical agents speaks of how difficult the convergence of a Q-Learning agent can be.

Our work differs from the above (Klein, 2016; Heinz, 2019), in that we try to bring together many contributions of other authors, much like Rainbow (Hessel et al., 2017), in order to try and bring forth a new SOTA to the field of Reinforcement Learning in Super Mario Bros, while attempting to train an agent that is capable of completing far more than a singular level in a rather more difficult setting than that of most of the surveyed work (using raw-image-data as state representations).

For an excellent timeline and in-depth description of the developments in the SOTA of Reinforcement Learning in the Atari57 benchmark over the past few years, see the current SOTA Deepmind's Agent57's (Badia et al., 2020*a*) accompanying blog post (Badia et al., 2020*b*).

## III SOLUTION

### *A Background Theory: Markov Decision Processes and Reinforcement Learning*

Markov decision processes (MDPs) (Puterman, 1994) provide a mathematical framework for modelling decision-making in an environment $\xi$, in our case SMB. An MDP is represented by a tuple $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ is the finite set of possible states of $\xi$, $\mathcal{A}$ is the finite set of legal actions that can be taken, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the (stochastic) one-step transition function to the next state $s'$ given the current state $s$ and a chosen action $a$ and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the (stochastic) reward function. In this paper, $\xi$ is modelled as a finite (as each episode's length is bounded by the in-game timer) Partially Observable Markov Decision Process (POMDP). POMDPs are an extension of the MDP framework, where the true state of the environment remains hidden and the decision process instead revolves around the information-deficient, partially-observable observa-

tions $o$ provided by $\xi$. The equivalent POMDP is represented by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, \mathcal{O})$, where $\Omega$ represents the finite set of possible observations $o$ and $\mathcal{O} : \mathcal{S} \to \Omega$ is the (stochastic) observation function mapping from true environment states to observations. However, for ease of reading and to conform to the notation found within most literature, outside of §III-A, an observation $o$ will be referred to as a state $s$.

Reinforcement Learning (RL), inspired by behaviorist psychology (behaviorism), is one of the three fundamental machine learning paradigms, wherein agents are trained to behave optimally with respect to a given reward signal. Our agent is specifically trained within the RL framework set out by Sutton (1988), whereby it is trained in a discretized environment with time-steps $t = 0, 1, 2 \ldots$, to maximize the future discounted return $G_t = \sum_{t'=t}^{\mathcal{T}} \gamma^{t'-t} r_{t'}$ at any given time-step $t$, where $\mathcal{T}$ is the time-step in which the episode in question terminates, $\gamma \in [0, 1)$ is the constant discount factor and $r_{t'} \sim R(s_{t'}, a_{t'})$ is the reward obtained from $\xi$ at time-step $t'$ having taken an action $a_{t'} \in \mathcal{A}$ from a state $s_{t'} \in \mathcal{S}$, given the observation $o_{t'} \in \Omega$. The actions $a_{t'}$ are chosen via some policy $\pi$, which, in some form, is being optimized (see §VII-A). Within this framework, when an action $a_t$ is taken, the environment transitions from state $s_t$ to state $s_{t+1} \sim T(\cdot|s_t, a_t)$ and the agent is provided with a new observation $o_{t+1} \sim \mathcal{O}(\cdot|s_{t+1})$ and reward $r_t \sim R(s_t, a_t)$ (NB in some literature you will see this reward defined as $r_{t+1}$). For details on the action space $\mathcal{A}$, reward function $R$ and observation space $\Omega$ see §III-E, §III-F and §III-B respectively.

Within this RL framework, our agent is trained to maximize $G_t$ using various extensions to an algorithm named Q-Learning (Watkins and Dayan, 1992). For sections of formal theory and explanations regarding Q-Learning, Deep Q-Learning, Double Q-Learning, Multi Step Returns and Value Function Rescaling see the respective sections within the Appendix, §VII.

### *B Architectural Overview and Design Choices*

The input to the Deep Q-Network (DQN) (Mnih et al., 2013) was chosen to consist of a stack of four $84 \times 84$ downsampled grayscale images, along with the latest action taken by the agent and most recent reward received from the environment. These images are first passed through a convolutional stack and then fed into an LSTM (long short-term memory) (Hochreiter and Schmidhuber, 1997) along with the previous action and reward to produce the "state-encoding", as per Kapturowski et al. (2019); see §III-C and Figure 1 for details. This combination is chosen to mimic and represent part of a human's (visual) sensory input (the images) and short-term memory (the recent actions and rewards, along with the LSTM's hidden state). Additional sensory input (auditory) is provided via the reward function signal; for full details on the design of the reward function see §III-F.

The observations comprise a stack of frames in order to counter the frame-wise partial-observability of games (Mnih et al., 2013), where one frame does not convey all the information of the true state of the game (e.g. velocities of entities), along with the perceptual aliasing of frames, where one frame could refer to many (vastly) differing true states; for example Mario falling into a fireball, a state of imminent death, or jumping over a fireball, no threat at all, could look exactly the same. Therefore, an observation contains four consecutive partially-observable "states" instead of one. However, the agent only "sees" a snapshot of the environment every 3 in-game frames, with its chosen action being repeated over these 3 frames (frame-skipping), as per Mnih et al. (2013). Thus, the images in question above were chosen to consist of a max-pool over these 3 frames, resulting in an observation containing information from 12 consecutive

frames. This pooling operation is performed to counter the "flickering" that can often occur in the NES environment and to not discard the additional information held in the other 2 frames, while at the same time not overwhelming the agent with it, and can be thought of as analogous to the visual motion-blur humans experience. In contrast to Mnih et al. (2013), as the SMB environment is $\sim$20 times more computationally expensive than the Atari environment, frame-skipping was performed not due to the imbalance of environment stepping and agent's action selection times, but due to the resultant linear increase in the agent's planning time horizon and reduction of the highly correlated nature of consecutive environment states and observations. In informal experimentation, the inclusion of frame-skipping vastly increased training stability and convergence of the agent, and was a key hyper-parameter (Braylan et al., 2015; Kalyanakrishnan et al., 2021). We hypothesize that frame-skipping is particularly vital to our agent due to its training via mini-batches of 80-transition-long sequences of experience (Kapturowski et al., 2019), rather than a mini-batches of de-correlated individual transitions sampled through the Experience Replay buffer (Lin, 1993). Thus, frame-skipping not only helps the mini-batches sampled remain independently and identically distributed (i.i.d.), as is required for Q-Learning, but helps reduce the correlated nature of the sequences themselves while ensuring the hidden gradients don't vanish too soon. Otherwise, the use of an LSTM would be in vain if no information could be passed along. Additionally, frame-skipping helps bring the response-time of the agent much closer to a human's, the designed audience of SMB, thus increasing the efficacy of actions taken by removing their warm-up/lag time (button presses are not meant to last for a singular frame, but for 5-20), features not intended to be felt or hinder one's performance, but to make the game feel smoother and more natural for a human, by taking into account their reaction/response times.

As mentioned above, a Recurrent Neural Network (RNN) is included to generate the state representation (Hausknecht and Stone, 2015; Mnih et al., 2016), specifically an LSTM (Hochreiter and Schmidhuber, 1997), with the agent being trained on mini-batches of sequences. As per R2D2 (Kapturowski et al., 2019), hidden-states are stored in and initialized from the Replay buffer during training and LSTM burn-in is performed for each sequence in the mini-batch (with all gradient flow stopped while the hidden-state is being "burned-in"). Both of the above are performed to alleviate the effects of recurrent state staleness and representational drift (Kapturowski et al., 2019) in the form of destructive updates to the RNN parameters near the beginning of sampled sequences. As per the findings of Kapturowski et al. (2019), the mini-batch size, sequence and burn-in lengths are chosen as 64, 80 and 40, respectively. However, as the ends of sequences are truncated in training due to the use of $n$-step returns, the sequences submitted to the Replay buffer are of length $40 + 80 + (n - 1)$ and these sequences overlap by $40 + (n - 1)$. In addition to what is described in Kapturowski et al. (2019), we chose to burn-in the start of episodes by prepending the initial state $s_0$ $(40 - 1)$ times. This was implemented as a fix for the severely unstable training experienced for weeks by following the exact details of the R2D2 algorithm described by Kapturowski et al. (2019).

Instead of sampling randomly from the Experience Replay buffer (Lin, 1993) like Mnih et al. (2013), we sample from a Prioritized Experience Replay (PER) buffer (Schaul et al., 2016) using the absolute TD-errors of a given sequence (see Kapturowski et al. (2019) for details on priorities used). This means that sequences that were more "surprising" to the agent are sampled with higher importance than those where the expected Q-values are already predicted well, as these sequences are the ones from which there is more to learn.

With the aim of increasing training stability and convergence, a variant of a DQN, using

Double Q-Learning (see §VII-C) (van Hasselt, 2010) is used, namely a DDQN (van Hasselt et al., 2015). This is achieved via the reduction of the (non-uniform) overestimation of Q-values present in DQNs. In and of itself overestimation is not an issue (if uniform), but the feedback-loop it creates can be, along with its limitation of future exploration into the so-far non-overestimated, but potentially optimal, areas of the state-action space.

To the same end of increasing convergence, a Dueling Network architecture (Wang et al., 2016) was employed where possible (when using a Dueling architecture in a distributional setting in conjunction with IQN, this caused the model to the rapidly diverge). The use of a Dueling architecture allows the model to decouple the estimation of the state values and the state-dependent action advantages. This is performed via temporarily splitting the DQN's single fully connected stream used to produce Q-values into two streams, the (state) value and the (action) advantage streams, before a clever merger to produce the final Q-values for each action (as shown in Figure 1). This allows the advantage stream to focus its capacity on key moments of gameplay, such as when there is an enemy or pitfall in front of Mario, over more trivial situations, while handing over the always-crucial estimation of the states themselves to the value stream. The gradient clipping employed by Wang et al. (2016) was also tested in this project, however this proved to be detrimental in informal experimentation.

In contrast to the one-step rewards used for bootstrapping targets in the Bellman equation update in DQN (see §VII-B), forward-looking discounted $n$-step returns (see §VII-D) (Sutton, 1988) are used to provide faster convergence via allowing the Q-Learning update to "see further ahead". Furthermore, in contrast to the typical clipping of rewards to a range such as $[-1, 1]$ for the sake of stability, which as a by-product removes the agent's ability to distinguish between large and small rewards, we employ an invertible value function rescaling $h(x) = \text{sign}(x)(\sqrt{|x| + 1} - 1) + \epsilon x$ (see §VII-E) (Pohlen et al., 2018).

Distributed RL is also applied in the form of a single-learner-multiple-actor scheme to allow more efficient data gathering by having 256 actors run in parallel in a multi-core set up. In order to ensure sufficient exploration of the state space both NoisyNets (Fortunato et al., 2017) and four $\epsilon$-greedy exploration schemes were experimented with, namely those of Rainbow (Hessel et al., 2017) (linear $\epsilon$-annealing), a custom multi-stage linear $\epsilon$-annealing of our own, and the two multi-actor $\epsilon$-spread schemes of Ape-X (Horgan et al., 2018) and Pohlen et al. (2018).

Sinha et al. (2020) (D2RL) proposed that recent advances in Deep Learning architectures could be applied to RL, resulting in an improved inductive bias. Specifically, they showed that using a deeper architecture in conjunction with dense-connections (skip-connections), as exemplified in Figure 1, in various pre-existing SOTA actor-critic RL algorithms, resulted in vastly improved sample-efficiency. This paper shows that the same applies to traditional RL.

Finally, a distributional RL variant of our algorithm was created, using the impressive framework of Dabney et al. (2018). Our transformed Implicit Quantile Network (Dabney et al., 2018) (IQN) instead models the return distribution, using quantile regression, rather than its scalar expectation, but unfortunately a further description of the approach and theory behind IQNs is beyond the scope of this paper.

### *C The Implementation*

All implementation was performed in Python 3.6.9 using PyTorch 1.7.1 alongside NVIDIA's CUDA 11.0 for implementation of all neural networks. Distributed learning was implemented natively in Python using its multiprocessing library via pipes, leading to 256 "actors" (SMB

environments) being run in parallel using 32 cores. The limitations of this set up and its differences with SOTA distributed RL algorithms, such as R2D2 (Kapturowski et al., 2019) and Ape-X (Horgan et al., 2018), are discussed in §V.

The final set up chosen for random exploration was a somewhat surprising combination of NoisyNets and an $\epsilon$-greedy policy, specifically where each of the $m$ actors $i$ for each level has $\epsilon_i = 1.5 \times 0.1^{\alpha_i + 3(1-\alpha_i)}$, where $\alpha_i = \frac{i}{m-1}$, adapted from the scheme used by Pohlen et al. (2018).

Stochastic gradient descent was performed using the Adam (Kingma and Ba, 2017) optimization algorithm to update the online network's parameters $\boldsymbol{\theta}$ (with $\boldsymbol{\theta}^-$ being held fixed via stopping all gradient flow through the target network) on gradients calculated using back propagation through time (Werbos, 1990) (BPTT), minimizing the loss function shown in (1). For more details on the layered construction of this loss function from the iterative updates used in the original Q-Learning algorithm, see the Appendix, §VII. This loss function incorporates $n$-step returns (and is thus truncated), value function rescaling, Double Q-Learning, the importance-sampling weights $w_i$ of a sequence $i$ from PER (Schaul et al., 2016), and is trained using mini-batches of 64 length-80 sequences $(s^i, a^i, r^i)$ (with a final appended $s$):

$$\frac{1}{64}\sum_{i=1}^{64}\frac{1}{2}w_i\sum_{t=0}^{80-(n-1)}\left(h\left(\sum_{k=0}^{n-1}\gamma^k r^i_{t+k} + \gamma^n h^{-1}\left(Q(s^i_{t+n}, \underset{a'\in\mathcal{A}}{\operatorname{argmax}}\, Q(s^i_{t+n}, a'; \boldsymbol{\theta}); \boldsymbol{\theta}^-)\right)\right) - Q(s^i_t, a^i_t; \boldsymbol{\theta})\right)^2 \tag{1}$$

Figure 1 shows the architecture of R2D4+'s final Q-Networks $Q(s, a; \boldsymbol{\theta})$ and $Q(s, a; \boldsymbol{\theta}^-)$. It is

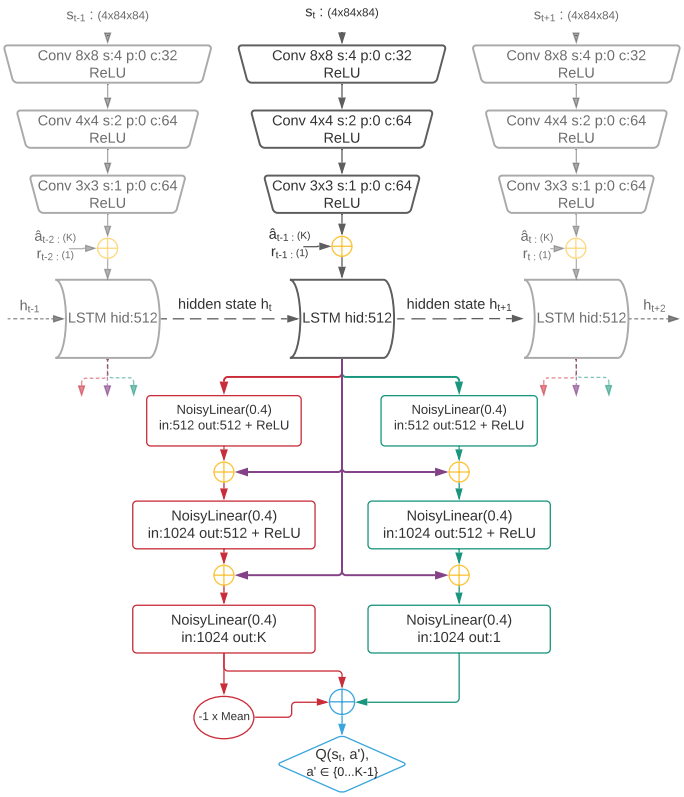
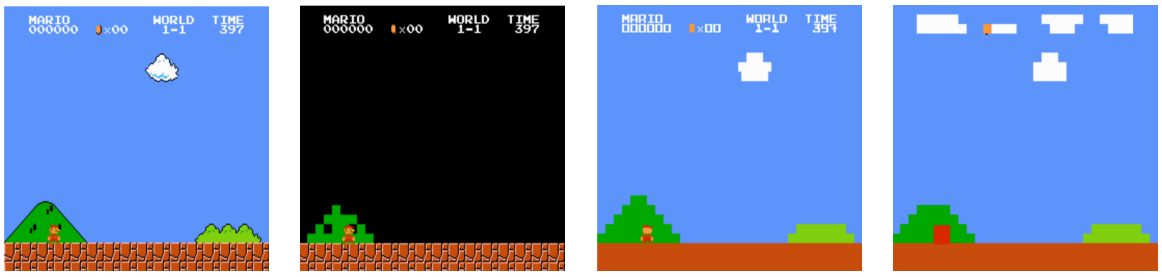



Figure 1: The structure of our Dueling LSTM-based Q-Network with $K$ actions, with value and advantage streams highlighted in green and red respectively, skip connections from D2RL shown in purple, tensor concatenation in yellow and an element-wise addition to produce the final Q-values in blue. The input sequences $s, a, r$ and $h_0$ are all sampled from a PER buffer, with $\hat{a}$ representing the one-hot-encodings of $a$.

of note, however, that, due to the inferior learning of the agent upon transformation into an IQN (Dabney et al., 2018), the resultant changes are not displayed in Figure 1 nor the loss in (1), both of which would otherwise look *vastly* different (and considerably more complex). However, the accompanying code-base's agent can be made distributional with the toggle of a switch, just like all other functionality and components, and the respective architecture inspected.

### *D The Environment*

The SMB OpenAI Gym environment (gym-super-mario-bros) was provided Kauten (2018), emulated in Python using nes-py. The environment runs at 30 frames-per-second (FPS), accepting an action and producing a $240\times256\times3$ full-colour image (from which an observation is produced, as per §III-B) at every step. With the use of 3-frame-skipping the agent perceives the environment running at 10 FPS, thus submitting 10 actions every second. Therefore, in this paper, a time-step $t$ refers to $0.1$ in-game-seconds. For comparison with works using the popular Atari-57 benchmark, the Atari environment runs at 60 FPS, and is approximately 20 times less computationally expensive per step. In Atari-57, the standard frame-skip parameter is 4, with a resultant perceived 15 FPS.




(a) SuperMarioBros-v0 (b) SuperMarioBros-v1 (c) SuperMarioBros-v2 (d) SuperMarioBros-v3

Figure 2: Example frame-renders from the different environments provided by Kauten (2018).

In this paper, two versions of the SMB environment are experimented with: "SuperMarioBros-v0" and "SuperMarioBros-v1". The only meaningful difference between these two is that the background is removed from the image in "SuperMarioBros-v1" (all sprites, e.g. enemies, are still shown in full detail), as shown in Figure 2. This is done to determine how hindered the agent is by the unnecessary information present in the background and test its ability to learn to discard this useless information, as a human does. "SuperMarioBros-v1" is used, solely, in the ablation (and generalization) studies with the aim to test the agent's unhindered performance, rather than its discerning ability (and memorization of specific levels). However, it is of note that, for approximately half of SMB levels, this change makes no difference, as there are already no backgrounds.

Additionally, we altered the environment so that if Mario gets stuck for 450 in-game frames, the game terminates. This *vastly* improved the training times of the agent, as it removed the need to render and store 100s of thousands of unnecessary identical frames, which would otherwise, hinder how i.i.d. the buffer is.

## *E The Action Space*

Super Mario Bros. (SMB) was a game developed by Nintendo for the Nintendo Entertainment System (NES), which had 6 buttons that could be pressed by the user: left (move left), right (move right), up (look up), down (crouch), A (jump/swim), B (accelerate / throw fireballs) (along with start (pause) and select). This resulted in a total of 20 ($5\times2\times2$) possible combinations, excluding combinations with the start and select buttons. However, many of these combinations either do not make any sense in SMB or are rarely used by the average user, for example any combination involving the up button, as this button generally has no use in SMB (unless standing under a secret vine), thus we give Mario access to only the most useful inputs. This results in our action space consisting of 7 inputs, identical to that used in most of the related works and often referred to as "simple movement": do nothing (NOOP), move right (right), jump right (right+A), run right (right+B), jump right while running (right+A+B), jump (A), move left (left). The left movement is kept particularly "simple", as the sole goal of SMB is to move right and reach the end of the level as fast as possible, alive. This reduction in action space is deemed suitable as it avoids an unnecessary inflation of the action space, allowing easier investigation into the unhindered performance of our proposed architecture. However, experiments were performed using "complex movement" to investigate the effects of an increased complexity of the action space (see Figure 4), wherein the agent was also allowed access to the following actions: jump left (left+A), run left (left+B), jump left while running (left+A+B), crouch (down), look up (up). Therefore, the "complex movement" scheme consists of 12 actions.

## *F The Reward Function*

A custom reward function, built on top of that engineered and offered by Kauten (2018) and comprising 5 components, was chosen to allow the agent to learn in a RL setting and provide the additional information (from the instruction manual, word of mouth and human conditioning) and the sensory input, which a human would have access to while playing SMB but which is missing from the pixel-only observations given to the agent. The goal of the game, moving to the right to complete each level without dying, in the quickest time possible (as described within the instruction manual or passed by word of mouth), is conveyed to the agent via the use of: the velocity term $v_t = x_t - x_{t-1} + (y_t - y_{t-1})/10$ (with $x_t$ & $y_t$ being the x and y coordinates at time-step $t$, and $v_t$ clipped such that $|v_t| < 15$ by Kauten (2018)); the time ticking penalty $\Delta clock_t = clock_{t-1} - clock_t$ (with $clock_t$ being the in-game clock reading at time-step $t$); and the death penalty applied upon loss of life $d_t = -15$ if $t$ is terminal state, else $d = 0$. In order to provide some of the auditory signal from the game, along with preconceived ideas of what is a "good" sound, it was observed that every time that a positive sound was played it was accompanied by an increase in the in-game score, thus the change in score $\Delta score_t = score_t - score_{t-1}$ between each time-step is included in the reward signal. For example, this provides incentive to kill enemies, collect coins and jump into item boxes. Finally, a boost/penalty for a change in Mario's state is included (encompassing information from the instruction manual, audio from the game and human conditioning) of $\Delta M_t = 10$ if Mario gets "stronger" (e.g. from tall to fireball-mode) or $\Delta M_t = -10$ if Mario gets hurt and becomes "weaker". This leads to the reward $r_t$ for a time-step $t$ being defined as:

$$r_t = v_t + \Delta clock_t + d_t + \frac{\Delta score_t}{200} + \Delta M_t \tag{2}$$

In experimentation, after observing that the agent never chose to pick up a mushroom or fire-flower to upgrade its state, various rebalancing and boosting of the encompassing terms of $r_t$ was attempted, however this generally harmed or sometimes de-stabilized learning, suggesting that the chosen $r_t$ lies near to some form of optimum.

### *G Testing, Verification and Validation*

TensorBoard was used to collect and analyze all data from tests and experiments in real-time. When tuning/testing hyper-parameters and model components for the final agent, experiments were run for at least 48 hours to test and compare the expected learning trajectories of the agents on multiple random seeds. The metrics used during testing were rolling averages of the episodic scores, distances travelled and completion rates during training, along with the equivalent metrics from the evaluation episodes conducted every 50 learner-steps on fixed sets of in and out of sample levels.

To verify the agent's learning, the convergence and gradient of the learning curve was observed, and the agent was compared to a random agent. To validate the agent's performance, its ability to complete difficult levels and its speed to complete various levels were compared against a small sample of expert players and online leader-boards.

## IV RESULTS

The agent was evaluated by training and testing on the various sets of levels of increasing difficulty and size described below, while plotting a rolling average of the episodic scores over all episodes concluded within the last 500 learner steps, alongside various other statistics, such as the loss, level-completion rates and breakdown of scores on individual levels using TensorBoard.

The data for each experiment was gathered using a sequential-sampling strategy derived from the insights of Decision Theory, wherein each experiment was first run on two fixed random seeds (742 and 101) for at least 48 hours each time, and then, if the noise/disparity between these two runs was sufficiently high, data from two additional seeds would be sampled (783 and 129). The majority of experiments showed very similar convergence across different seeds and thus were not rerun, overall showing a robustness of the model to the choice of random seed. At the end of this process the mean over all results across different seeds is taken and plotted using the Python module tensorboard-reducer.

All experiments were run on either an NVIDIA TITAN Xp or an NVIDIA GeForce RTX 2080 Ti, with all implementation performed in Python 3.6.9 using PyTorch 1.7.1 alongside NVIDIA's CUDA 11.0. Many thanks go to Durham University for the use of their NVIDIA CUDA Centre (NCC) GPU cluster for the testing, tuning and training of all models, clocking approximately 7500 GPU hours over a period of many months. Without the use of the same none of the research contained herein would have been possible.

Firstly, the agent's learning when trained on a singular level was evaluated for various levels of increasing difficulty taken from throughout the game. In this setting the agent is allowed to over fit and specialize as much as it likes, and we are testing its "maximum skill" and ability to learn in difficult situations. The results are shown in the top row of Figure 3.

Then, the agent's learning when instead trained and tested on an entire world (4 levels) was evaluated for worlds 1, 5 and 7 (an easy world and two difficult worlds respectively) the results of which are shown in the bottom row of Figure 3. For reference to other literature, in all plots, 10

Table 1: HYPER-PARAMETERS USED FOR ALL RUNS

| | | | |
|---|---|---|---|
| Discount rate $\gamma$ | 0.997 | Sequence length | $80+(n-1)$ $(+l)$ |
| Adam learning rate | $10^{-4}$ | Burn-in length $l$ | 40 |
| Adam $\epsilon$ | $10^{-3}$ | $n$-step returns $n$ | 5 |
| Value function rescaling $\epsilon$ | $10^{-3}$ | Sequence overlap length | $40 + (n-1)$ |
| Frame-skip | 3 | Noisy Nets $\sigma$ | 0.4 |
| Batch-size | 64 | Target Network update period | 300 learner steps |
| Prioritization exponent $\omega$ | 0.9 | Buffer size | 18000 sequences |
| Min sequences to train | 3000 | Prioritization importance sampling $\beta$ | $0.4 \rightarrow 1$ |

thousand (10k) learner steps are equivalent to 25 million environment frames. It is also of note that the data used in all plots, except the evaluation scores in Figure 4, is gathered from training episodes, and thus acts as a lower bound for the true ability of the agent. This is because during training, the agent is still performing random exploration using $\epsilon$-greedy and NoisyNets rather than acting optimally. Therefore, during evaluation episodes the agent performs approximately 5-10% better.

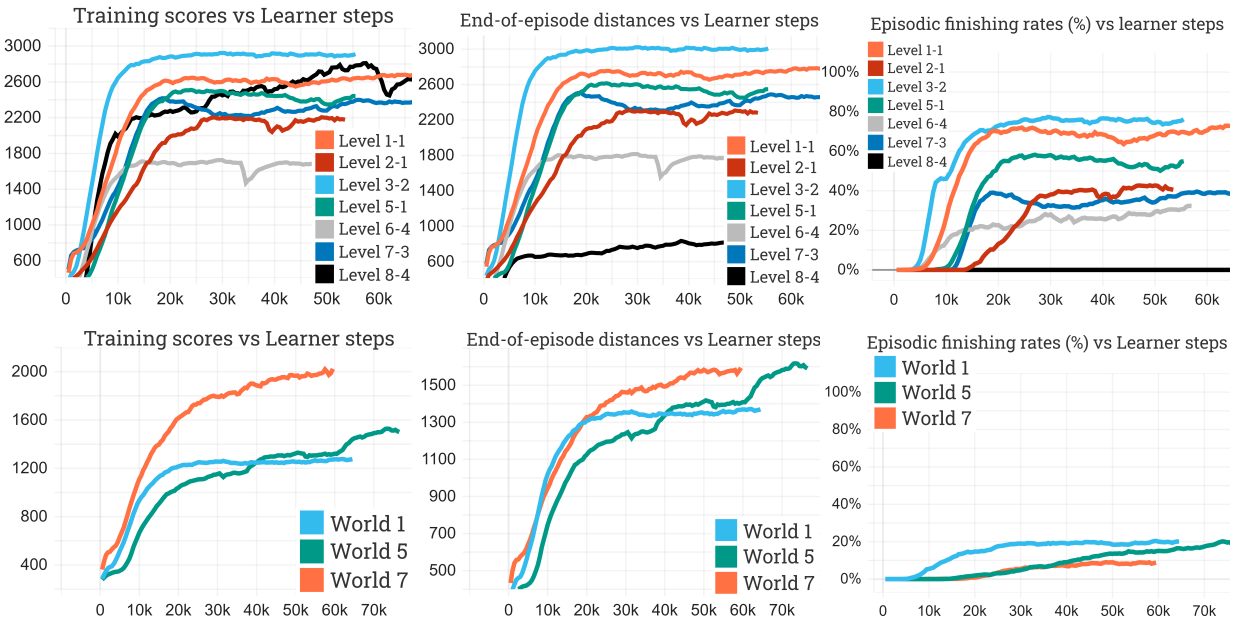



Figure 3: Left to right: plots of rolling averages of episodic training scores, end-of-episode distances and episodic finishing rates all vs learner steps, plotted throughout training; Top to bottom: singular level training sets to singular world training set.

Finally, the agent's performance when trained on the entire game, minus a small set of 4 evaluation levels (1-1, 2-1, 3-2 & 6-4) on which the agent is never trained, is evaluated. The episodic training scores along with a rolling average of the scores of the out-of-sample evaluation episodes (which are run every 50 learner steps) are plotted in Figure 4. The evaluation levels are excluded from the training set to test the generalization power of the agent and see how it performs on levels on which it has never been trained. This provides a measure of the "true skill" of the agent and determines whether it is simply over-fitting the training set.

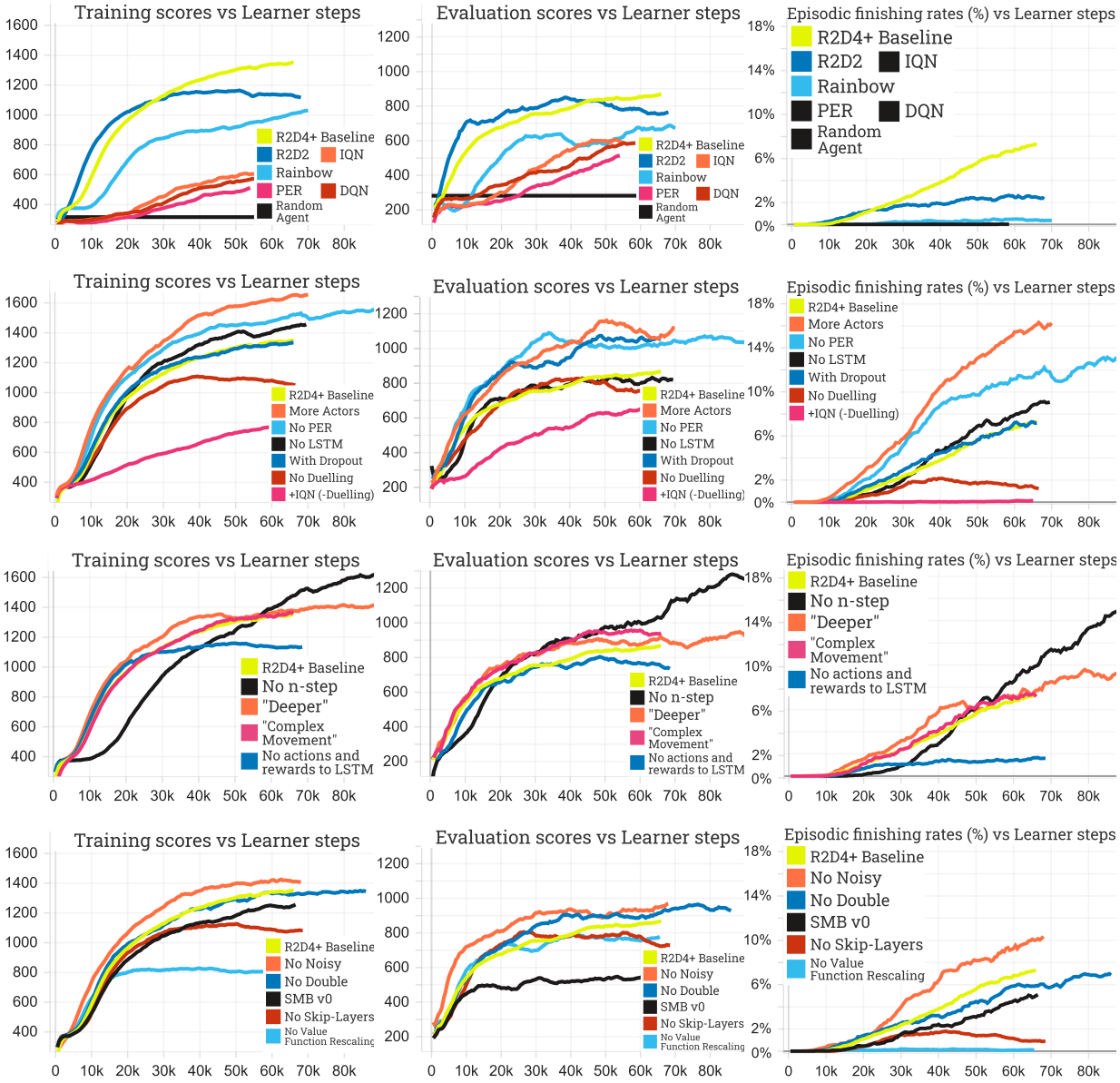



Figure 4: Left to right: plots of rolling averages of episodic training scores, out-of-sample evaluation scores and episodic finishing rates all vs learner steps, plotted throughout training.

In the above experiments, the size of the training set is increased from 1 to 4 to 28 in order to display the difference between an agent being able to expertly play and over-fit to one level vs an agent that truly understands the game's mechanics and rules and is able to play the entirety of the game at once, in addition to some levels it has never even seen. As justified in §III-D, these runs are performed using "SuperMarioBros-v1" rather than "SuperMarioBros-v0", on which all other runs are performed.

Simultaneously, the performance of the agent is compared to the predicted performance of the various literature's agents mentioned and referenced within this paper: DQN, DDQN, PER, Rainbow, R2D2 & IQN (in the top row of Figure 4). These agents are recreated using our code base by toggling components on and off, and all hyper-parameters are kept constant across runs to allow for a fairer comparison, apart from $\beta$-annealing, which is only enabled for R2D4+

and Rainbow (as per the literature). Furthermore, there are various additional differences to the literature, notably Rainbow is not distributional and all agents use the epsilon-greedy exploration scheme and distributed set-up described in §III-C. All these factors lead to the agents not being completely identical to their original versions. However, the majority of these changes should improve the algorithms and overall the resultant performance is deemed to be indicative of that of the original agents.

Figure 3 demonstrates that the agent is capable of teaching itself to consistently perform well and complete various levels and worlds of increasing difficulty. However, the agent is incapable of escaping the infinite-room-loops in 8-4 or 7-4, hence the large difference between scores and distances for the 8-4 and the World 7 runs. This is partially due to the inner workings of SMB, as the x-coordinate "inexplicably" jumps from a large value to 0 at some point, with Kauten's (2018) velocity function clipping this jump. However, for a RL agent, the efficient use of long-term memory in an almost a fully observable game is a great challenge. Figure 3 also shows that, in the absence of an infinite-room-loop, the episodic score provides a good estimate of the distance travelled, simply scaled by the speed of the agent. Furthermore, every level has a different length, and therefore the maximum score attainable on a given level varies greatly. Therefore, when comparing across training sets, observing the episodic finishing rates alongside the scores provides a useful indication of the ability of the trained agent. However, the best measure is to watch the videos of the agent playing, which are all provided as supplementary material, located at `https://github.com/shadowbourne/R2D4-RL`. Alongside these videos, higher definition interactive versions of all graphs are available, in case the reader has an inclination to inspect them in more detail.

Figure 4 shows that despite the agent not having ever seen or been trained on the evaluation set of levels, it still performs $\sim 70\%$ as well on them, and is able to consistently complete 1-1 and 3-2 ($3\%$ and $6\%$ finishing rates respectively) and occasionally 2-1 and 6-4 (see supplementary materials for plots and videos)! This suggests deep understanding of the game's mechanics rather than simple over-fitting. Furthermore, the top row of Figure 4 shows that R2D4+ considerably outperforms all previously surveyed RL algorithms, including ex-SOTA R2D2, alongside supporting the third row's evidence that PER actually hinders R2D4+'s ability to learn. The performance of a random agent is also shown for comparison. The second, third and fourth rows show the results of ablation studies in which the final agent was trained with all but one component enabled, so that the relative contributions of each component can be visualized and compared. In addition, some extra components were trialled during this stage, such as the novel contribution of applying dropout to RL, which is commonly regarded as destructive and useless, however, we have shown that dropout, whilst making little difference to the training scores, considerably boosts the out-of-sample evaluation scores and therefore the over-fitting of the agent. This drop-out was applied to all (noisy) linear and convolutional layers within the online network only, with $40\%$ and $10\%$ drop-out rates, respectively. It is also likely that in-sample evaluation scores would likewise be higher, as drop-out is enabled during acting. The other two trials were adding an additional linear layer alongside replacing the first skip-connection with one to the CNN output, which we name "Deeper", and a variant where R2D4+ uses 512 actors instead of 256, which we name "More actors".

Additionally, Table 2 shows the results of comparing the "speed-run" times on various levels of R2D4+, both single-level and whole-game (all 32 levels) editions, and a small sample of expert SMB players, who were given 50 attempts at each level after a small warm up. The best times

of the experts and R2D4+ were taken over the sample of 50 runs, all measured in seconds (lower is better). In contrast, 1024 actors are used in these shorter experiments to combat the effects of the resource constraints discussed in §V and demonstrated by the "More actors" experiment.

Table 2: SPEED-RUN COMPARISON OF R2D4+ AGAINST EXPERT PLAYERS

| Agent ↓ / Level → | 1-1 | 2-1 | 3-2 | 5-1 | 6-4 | 7-3 |
|---|---|---|---|---|---|---|
| R2D4+ Single-Level 1024 actors | 55 | 71 | **57** | **61** | **39** | 62 |
| R2D4+ Whole-Game 1024 actors | 57 | 75 | 59 | 65 | **39** | 63 |
| Expert 1 | 55 | 66 | 70 | 67 | 41 | **60** |
| Expert 2 | 80 | 79 | 68 | 66 | 61 | 64 |
| Expert 3 | **54** | **64** | 60 | 68 | 43 | 64 |
| Former World Record | 30 | 55 | 55 | 53 | 39 | 60 |

## V EVALUATION

The project will be evaluated using a list of deliverables of increasing difficulty, primarily assessing various aspects of the final agent, namely its skill, generalizability, memory and speed (in that order). In addition, there will be some deliverables assessing the knowledge and understanding gained and demonstrated during the project itself. The selected deliverables are as follows:

1. Implement an agent which is able to consistently complete (when trained solely on the equivalent level): **(a)** a singular easy level, specifically 1-1; **(b)** a singular difficult level, specifically 5-1 and 7-3. (Skill)
2. Implement an agent which is able to consistently perform well on all and complete most levels of (when trained solely on the equivalent 4 levels): **(a)** a singular easy world, specifically World 1; **(b)** a singular difficult world, specifically Worlds 5 and 7. (Skill)
3. Implement an agent which is able to consistently perform well on all of and often complete many of the levels on a training set consisting of the whole game (32 levels) minus a small set of 4 evaluation levels, specifically 1-1, 2-1, 3-2 and 6-4. (Skill)
4. Implement an agent which when trained on the whole game minus a small evaluation set of levels (which it is never trained on) is able to generalize and consistently perform well on and sometimes complete an out-of-sample: **(a)** easy level (1-1); **(b)** medium difficulty level (2-1); **(c)** difficult level (3-2 and 6-4). (Generalizability)
5. Implement an agent that is able to consistently perform well on a difficult level that requires the use of long-term memory, specifically to get out of infinite room loops for which Super Mario Bros' later levels are famous. (Memory)
6. Implement an agent which is able to perform similar to or out-perform a small sample of expert players on their best time to complete the following levels: 1-1, 2-1, 3-2, 5-1, 6-4 & 7-3. (Speed)
7. To develop a deep understanding of agent produced, and thereby the subset of RL surveyed, and to show evidence for or against the orthogonality of the various contributions incorporated into R2D4+; this understanding will be primarily demonstrated through ablation studies of the agent's abilities. (Knowledge and understanding)

Overall the project was extremely successful, with all but deliverable 5 being fully delivered. The success of deliverables 1 and 2 is demonstrated in Figure 3, deliverables 3, 4 and 7 in Figure 4, and deliverable 6 in Table 2. R2D4+ appears to be able to learn regardless of the difficulty of the level, with the effects of increasing the training set's size being an initial decrease in skill due to the inability to overfit (with this effect plateauing after the increase to 4 levels), as can be observed by comparing the scales of Figure 3 and Figure 4, followed by a minimal increase in time to converge, without any evident further decrease in skill. It is of note that the agent never decides that it is optimal to upgrade its state by picking up a mushroom or fire-flower, presumably as this costs time, however this could also be due to a lack of sufficient exploration. From a human perspective, this is very strange, as upgrading your state gives you an extra life, additional abilities and access to some new areas (which could potentially give rise to higher score).

The generalizability of the agent was beyond all expectations, with R2D4+ showing impressive convergence on all 4 evaluation levels (see trend of rolling average of evaluation scores in Figure 4), managing to reach the end of all of them at least once (2 consistently, 1-1 & 3-2). Generalizing to these difficult levels (3-2 & 6-4) is not an easy task, and therefore these results are very promising.

Considering the skill of a player in SMB is determined by their ability to complete difficult levels alongside how fast they are able to do so, deliverable 6 was key in determining the skill of the agent. R2D4+ was able to compete with and often beat the experts' best times. In all levels both variants of R2D4+ managed to beat at least 1 of the 3 experts, and on 6-4, a level with no shortcuts, even matched the world record. It is of note that the world record holder often uses various hidden shortcuts to achieve his superior times, and that the times of R2D4+ and the experts are near optimal without the use of the same.

Unfortunately, R2D4+ was unable to effectively utilize its long-term memory to escape any of the infinite room loops in the later levels of SMB, thus failing deliverable 5. However, although lacking any evidence to suggest the agent would have otherwise succeeded, we theorize that a large portion of the blame for this failure lies with the inner workings of the RAM (random-access memory) within SMB combined with the velocity-term reward clipping, as discussed previously. Therefore, a good next step and a test of this hypothesis would be to remove this clipping, which would potentially allow the agent to realize it was in a loop, as currently, the agent is rewarded for continuing within this loop. Furthermore, one could attempt to improve the memory component of the agent, namely the LSTM, potentially replacing it with a transformer by investigating the new advances in transformer-based RL (Upadhyay et al., 2019; Parisotto et al., 2020).

The ablation studies and preliminary experiments conducted provided sufficient data to gain considerable insight into the inner workings of R2D4+. They also provided evidence for the orthogonality of components, in other words which components worked well together. Some of the strongest results came from the investigations with IQN. Namely, an IQN is highly incompatible with a Dueling architecture, causing the resultant agent to rapidly diverge. Furthermore, even once having removed the Dueling architecture, transforming R2D4+ into an IQN still resulted in a detriment to performance, which, as theorized earlier, might be due to the lack of stochasticity within SMB. Most of the remaining contributions surveyed were found to be complementary, with the notable exceptions being PER, Noisy Nets and perhaps the use of an LSTM. PER harming the agent's learning is very surprising and thus could be the target of future investigation. Initially, it also appeared that the use of $n$-step returns was detrimental to learning. However, a

subsequent experiment run to investigate this phenomenon, provided in the supplementary materials, suggests that this is likely due to a badly tuned $n$, with $n = 3$ showing far better results than the original $n = 5$. The most significant insight gained from the ablation studies was from the novel experiments designed with the use of dropout, as discussed in §IV.

In addition to the robustness of the choice of random seed discussed in §IV, R2D4+ appears to also be robust to the choice of hyperparameters, with most changes and combinations tested in preliminary tuning making little difference to the performance of the agent, and none caused the model to diverge. However, the magnitude of each hyper parameter was determined by previous studies' chosen values.

The main limitations of the project were due to resource and time constraints. Firstly, it was thus not possible to run experiments for as long nor on as many random seeds as a typical academic study, although these effects were countered by the use of our sequential-sampling strategy. Furthermore, for statistical significance, all claims and results from this paper would have been verified on a variety of different games and applications. Secondly and most importantly, although distributed, the single GPU setup of the agent was extremely limiting and could have been *vastly* improved by the inclusion of additional GPUs, one acting as a learner and the rest acting as actors, much like is done in Horgan et al. (2018) and Kapturowski et al. (2019), allowing acting and learning to run in parallel at a much greater scale. A single GPU setup is especially limiting in our case due to the SMB environment taking $\sim$20 longer to render than the usual Atari suite. For evidence of this obstruction to performance, see the vast improvement shown by increasing the number of actors from 256 to 512 in Figure 4. The reason for not using an increased number of actors in all experiments is once again due to time restraints, as this increase almost doubles the clock-time per learner step due to the inefficiencies of a multi-core setup.

Additionally, due to memory restraints the chosen buffer size was below optimal, with preliminary experiments showing an increase in performance with larger buffer sizes. However, an alternative implementation could have saved the contents to storage, rather than retaining them in RAM. Unfortunately, this may result in a further increase in clock-time.

## VI CONCLUSIONS

This project successfully creates an intelligent agent capable of teaching itself to play SMB to an expert level from raw-image-data, given no feature-engineering nor additional information about the game's rules or internal workings. The project achieves this by investigating how best to apply and combine many recent advances in RL that have never been applied to SMB, to create a final agent, which we name R2D4+ (Recurrent Replay Deeper Denser Distributed DQN+). This agent is believed to surpass the performance of all previously surveyed agents within RL (when these agents are run in the same distributed set up as R2D4+). Additionally, the final agent is able to generalize well and sometimes complete levels it has never trained upon.

The main novel findings of this project are as follows, in order of significance: dropout may be used beneficially to decrease overfitting and increase generalization power of an agent without any additional changes to the architecture; PER actually hinders learning in SMB (thus potentially in other similar environments), perhaps due to the high reward density; the provision of actions and rewards for the LSTM's input is highly beneficial; an IQN is incompatible with a Dueling architecture, and the combination causes the agent's learning to rapidly diverge; an IQN is detrimental to learning in SMB, thus may only be beneficial in a stochastic environment; RL

agents can be hindered by useless information such as the game background, especially in terms of their generalization power. This paper also provides evidence for the benefits of using a deeper network architecture with skip connections in traditional RL, following the works of Sinha et al. (2020).

Future extensions to this project would include transforming the agent using a fully distributed framework over multiple GPUs to increase the efficiency of data gathering and reduce training times, while allowing the agent to converge to a higher level of performance by increasing the rate of flow of new experience while retaining training times at a feasible level. Additionally, in line with the positive results of using dropout to reduce overfitting, investigating and applying the recent and extremely promising findings in the use of data augmentation in RL in RAD (Laskin et al., 2020) and DrQ (Kostrikov et al., 2020) to vastly increase sample-efficiency and generalization power would be of great interest. Furthermore, to enhance R2D4+'s long-term memory transformer-based RL (Upadhyay et al., 2019; Parisotto et al., 2020) could be investigated. In order to better encourage exploration of the state-space and instill "curiosity" into the agent, a form of intrinsic reward motivation could be employed, similar to RND (Burda et al., 2018) and Agent57 (Badia et al., 2020*a*), or alternatively any of the other exploration improvements applied on top of R2D2 by Badia et al. (2020*a*). Overall, any additional contributions to the field of RL could be applied, whether they come from current SOTA Agent57, future SOTA algorithms or unique standalone papers.

## References


Badia, A. P. et al. (2020*a*), Agent57: Outperforming the atari human benchmark, *in* 'International Conference on Machine Learning', PMLR, pp. 507–517.

Badia, A. P. et al. (2020*b*), Agent57: Outperforming the atari human benchmark.
**URL:** *https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark*

Braylan, A., Hollenbeck, M., Meyerson, E. and Miikkulainen, R. (2015), Frame skip is a powerful parameter for learning to play atari, *in* 'Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence'.

Burda, Y., Edwards, H., Storkey, A. and Klimov, O. (2018), 'Exploration by random network distillation', *arXiv preprint arXiv:1810.12894* .

Dabney, W., Ostrovski, G., Silver, D. and Munos, R. (2018), Implicit quantile networks for distributional reinforcement learning, *in* 'International conference on machine learning', PMLR, pp. 1096–1105.

Du, Y., Cui, S. and Guo, S. (2009), 'Applying machine learning in game ai design'.

Fortunato, M. et al. (2017), 'Noisy networks for exploration', *arXiv preprint arXiv:1706.10295* .

Hausknecht, M. and Stone, P. (2015), 'Deep recurrent q-learning for partially observable mdps', *arXiv preprint arXiv:1507.06527* .

Heinz, S. (2019), 'Using reinforcement learning to play super mario bros on nes using tensorflow'.
**URL:** *https://towardsdatascience.com/using-reinforcement-learning-to-play-super-mario-bros-on-nes-using-tensorflow-31281e35825*

Hessel, M. et al. (2017), 'Rainbow: Combining improvements in deep reinforcement learning'.

Hochreiter, S. and Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.

Horgan, D. et al. (2018), 'Distributed prioritized experience replay'.

Jørgensen, L. D. and Sandberg, T. W. (2009), 'Playing mario using advanced ai techniques', *[online], IT University of Copenhagen, Available at: http://twsandberg.dk/media/4615/playing%20mario%20using%20advanced%20ai%20techniques.pdf* .

Joshi, M., Khobragade, R., Sarda, S., Deshpande, U. and Mohan, S. (2012), Object-oriented representation and hierarchical reinforcement learning in infinite mario, *in* '2012 IEEE 24th International Conference on Tools with Artificial Intelligence', Vol. 1, IEEE, pp. 1076–1081.

Kalyanakrishnan, S. et al. (2021), 'An analysis of frame-skipping in reinforcement learning', *arXiv preprint arXiv:2102.03718* .

Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J. and Munos, R. (2019), Recurrent experience replay in distributed reinforcement learning, *in* 'International Conference on Learning Representations'.
**URL:** *https://openreview.net/forum?id=r1lyTjAqYX*

Kauten, C. (2018), 'Super Mario Bros for OpenAI Gym', GitHub.
**URL:** *https://github.com/Kautenja/gym-super-mario-bros*

Kingma, D. P. and Ba, J. (2017), 'Adam: A method for stochastic optimization'.

Klein, S. (2016), 'Cs229 final report deep q-learning to play mario'.

Kostrikov, I., Yarats, D. and Fergus, R. (2020), 'Image augmentation is all you need: Regularizing deep reinforcement learning from pixels', *arXiv preprint arXiv:2004.13649* .

Laskin, M. et al. (2020), 'Reinforcement learning with augmented data', *arXiv preprint arXiv:2004.14990* .

Lee, G., Luo, M., Zambetta, F. and Li, X. (2014), Learning a super mario controller from examples of human play, *in* '2014 IEEE Congress on Evolutionary Computation (CEC)', IEEE, pp. 1–8.

Liao, Y., Yi, K. and Yang, Z. (2012), Cs229 final report reinforcement learning to play mario, Technical report, Technical report, Stanford University.

Lin, L.-J. (1993), Reinforcement learning for robots using neural networks, Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

Lindberg, M. (2014), 'An imitation-learning based agent playing super mario'.

Mnih, V. et al. (2013), 'Playing atari with deep reinforcement learning', *arXiv preprint arXiv:1312.5602* .

Mnih, V. et al. (2016), Asynchronous methods for deep reinforcement learning, *in* 'International conference on machine learning', PMLR, pp. 1928–1937.

Mohan, S. and Laird, J. (2010), Relational reinforcement learning in infinite mario, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 24.

Mohan, S. and Laird, J. E. (2009), 'Learning to play mario', *Tech. Rep. CCA-TR-2009-03* .

Parisotto, E. et al. (2020), Stabilizing transformers for reinforcement learning, *in* 'International Conference on Machine Learning', PMLR, pp. 7487–7498.

Pohlen, T. et al. (2018), 'Observe and look further: Achieving consistent performance on atari'.

Puterman, M. L. (1994), *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons.

Schaul, T., Quan, J., Antonoglou, I. and Silver, D. (2016), 'Prioritized experience replay'.

Sinha, S., Bharadhwaj, H., Srinivas, A. and Garg, A. (2020), 'D2rl: Deep dense architectures in reinforcement learning'.

Sutton, R. S. (1988), 'Learning to predict by the methods of temporal differences', *Machine learning* **3**(1), 9–44.

Sutton, R. S. and Barto, A. G. (1998), *Reinforcement learning: An introduction*, MIT press.

Togelius, J., Karakovskiy, S. and Baumgarten, R. (2010), The 2009 mario ai competition, *in* 'IEEE Congress on Evolutionary Computation', IEEE, pp. 1–8.

Togelius, J., Shaker, N., Karakovskiy, S. and Yannakakis, G. N. (2013), 'The mario ai championship 2009-2012', *AI Magazine* **34**(3), 89–92.

Tsay, J.-J., Chen, C.-C. and Hsu, J.-J. (2011), Evolving intelligent mario controller by reinforcement learning, *in* '2011 International Conference on Technologies and Applications of Artificial Intelligence', IEEE, pp. 266–272.

Upadhyay, U., Shah, N., Ravikanti, S. and Medhe, M. (2019), 'Transformer based reinforcement learning for games', *arXiv preprint arXiv:1912.03918* .

van Hasselt, H. (2010), 'Double q-learning', *Advances in neural information processing systems* **23**, 2613–2621.

van Hasselt, H., Guez, A. and Silver, D. (2015), 'Deep reinforcement learning with double q-learning'.

Wang, Z. et al. (2016), 'Dueling network architectures for deep reinforcement learning'.

Watkins, C. J. and Dayan, P. (1992), 'Q-learning', *Machine learning* **8**(3-4), 279–292.

Werbos, P. J. (1990), 'Backpropagation through time: what it does and how to do it', *Proceedings of the IEEE* **78**(10), 1550–1560.