Implementation and Analysis of Multiple Reinforcement Learning Algorithms to Play Mario

Aryan Singh
Computer and Information Science
University of Florida
Gainesville, FL 32611
aryansingh@ufl.edu

Vaibhav Kulkarni
Computer and Information Science
University of Florida
Gainesville, FL 32611
kulkarniv@ufl.edu

Abstract

The concept of implementing Reinforcement Learning algorithms to achieve human-level performance for game play has been around for many years. In 2013, Google Deepmind released the groundbreaking Deep Q-learning algorithm, which was successful across a variety of Atari games, meeting and occasionally exceeding human performance [12]. Followed by this DeepMind's artificially intelligent Go player defeated the world champion of Go. Even more recently, OpenAI's team of five neural networks, known as OpenAI Five, has gained a lot of attention by defeating a team of top experts in the very complicated video game Dota 2. In terms of both gameplay and observation space, Super Mario Bros. is far more complex than most of the aforementioned Atari games, making it a fascinating concept for implementing deep learning algorithms. We trained an agent that can exceed the human baseline on Super Mario Bros using state-of-the-art approaches such as Deep Q Network [12], Double Deep Q Network [11], DDQN with Prioratised Experience Replay [9] and Proximal Policy Optimization [5]. We perform experiments and analyse the performance of these algorithms. The code for this project is available at https://github.com/KulkarniVaibhav/Reinforcement-Learning-Algorithms-to-Play-Mario.

1 Introduction

1.1 Background

Super Mario Bros. is considered as one of the most popular video game that was released in 1985 for the Nintendo Entertainment System. In this game the player takes on the role of Mario, who must navigate through a difficult landscape and reach the finish line by performing multiple actions such as walking, running, and jumping in combination with the directional movements. Mario gathers coins while avoiding or destroying adversaries who can take his life if they come into contact with him. The game requires precise button inputs to effectively complete various complex platforming challenges over a broad collection of levels and stages.

Super Mario Bros. is a computationally challenging game and clearly falls into NP-Hard category [4], with the complexity arising from the arbitrary size of the playing field. Due to the fact that the player only sees a fraction of the total environment dependent on Mario's present position, obstacles and adversary arrive and exit the frame in a seamless side scrolling platform. The seemingly arbitrarily, long and unpredictable paths of each setting make implementing Reinforcement Learning to this problem, an intriguing and fascinating challenge to investigate.

1.2 Motivation

When reinforcement learning was initially introduced in the 1950s, it was divided into two themes: one focused on establishing learning methods through trial and error, while the other gave a more theoretical framework for solving optimal control issues. Reinforcement learning emerged as a more formalized field of study and development in the 1980s when these practical and theoretical methodologies merged. For example, Richard Sutton and Andrew Barto emphasized theories like optimal control and dynamic programming at the time and crucial component ideas like temporal difference learning, dynamic programming, and function approximation. Fast forward to the 2000s, when deep learning started revolutionizing reinforcement learning by removing the need to manually construct features and allowing the use of raw sensor data (such as the pixels of an image rather than a segmented image).

But before we go any further let us define what exactly is reinforcement learning?

In contrast to supervised learning (which utilizes labeled training data) and unsupervised learning (which draws inferences from input data without labeled answers), reinforcement learning entails a system making short-term judgments using trial and error to optimize towards a long-term objective. The reinforcement learning agent learns the actions needed to maximize rewards over a longer period, whereas deep learning is employed to construct mathematical representations of relevant variables.

In recent years, there have been a lot of breakthroughs in the field of reinforcement learning due to the influx of deep learning methods. Some of the success stories include the following:

- In late 2013, DeepMind achieved a breakthrough in the world of reinforcement learning: using deep reinforcement learning, they implemented a system that could learn to play many classic Atari games with human (and sometimes superhuman) performance. The computer program has never seen this game before and does not know the rules. It learned by using deep reinforcement learning to maximize its score given only the pixels and game score as the input. They had used Deep Q Network.
- Alpha Go is a computer system developed by Google DeepMind that can play the game Go. The game of Go starts with an empty board. Each player has an effectively unlimited supply of pieces (called stones), one taking the black stones, the other taking white. The main objective of the game is to use your stones to form territories by surrounding vacant areas of the board. Google DeepMind's Challenge Match was a five-game Go match between 18-time world champion Lee Sedol and AlphaGo played in Seoul, South Korea between 9 and 15 March 2016. AlphaGo won all but the fourth game.
- The next from Deep Mind was Alpha Zero. Alpha Zero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case. Previous versions of AlphaGo were initially trained on thousands of human amateur and professional games to learn how to play the game Go. Alpha Zero, slips the Supervised Learning step and learns to play simply by playing against itself, starting completely from random play.
- Dota 2 is a multiplayer online battle arena video game. It is played in matches between two teams of five players, with each team occupying and defending their own separate base on the map. Open AI Five is a team of 5 neural networks that have started defeating amateur human teams at Dota 2. The program defeated a human in 2017.

The complexity of the challenging problem and potential of Reinforcement Learning to address such problems makes it interesting to work with.

1.3 Related Work

Our research is fundamentally driven on the bases of the research work carried by multiple teams across past decade.

One of the best-known success story of implementing reinforcement learning is TD-gammon, a backgammonplaying program which learnt purely by reinforcement learning and self-play. [8]

The Q-network displayed a characteristics to diverge when model-free reinforcement learning algorithms (Q-learning) were combined with non-linear function approximators [10] or even off-policy learning. As a result, most reinforcement learning research til early first decade of 21st century has concentrated on linear function approximators aiming to achieve better convergence guarantees. Recent research shows that the divergence of Q-network has issue has been mitigated by incorporating new methodologies with Q-network [12] and Double Deep Q-network [3] such as Experienced Replay and Prioratized Experience Replay [9], Policy Gradient methods like Proximal Policy Optimization [5].

We studied the these findings and decided to put them to test through implementation as we implement and compare each of these methodologies to train our model to play Super Mario Bros.

- **Playing Atari with Deep Reinforcement Learning:** The first paper (published in 2013) from the Deep Mind team introduced how the Deep Learning model can successfully learn control policies from high dimensional sensory input. The paper used CNN which was trained with a variant of Q-learning to take in raw pixels and output a value function to estimate reward. The proposed model was used on seven Atari 2600 games and the model outperformed all the previous models in six games and surpassed human experts on three.

  https://arxiv.org/pdf/1312.5602.pdf

- **Deep Reinforcement Learning with Double Q-learning:** This paper (published in 2015) from the Deep Mind team affirmatively pointed out the overestimation problem that DQN networks suffer. This paper introduced a variation of the DQN algorithm and showed that this algorithm not only reduces the observed overestimation but also leads to better performance in several games.

  https://arxiv.org/pdf/1509.06461.pdf

- **Prioritized Experience Replay:** This paper (published in 2016) again from the Deep Mind team was again an improvement over the DQN model. The paper introduced a new framework for prioritizing experience so as to replay important transitions more frequently, and therefore learn more efficiently. In prior work, experience transitions were uniformly sampled from a replay memory which would simply replay transitions at the same frequency at which they were originally experienced, regardless of their significance. This new framework achieved a new state-of-the-art, outperforming DQN with a uniform replay on 41 out of 49 games.

  https://arxiv.org/pdf/1511.05952.pdf

- **Proximal Policy Optimization Algorithms:** This paper (published in 2017) from the Open AI team discussed a new technique compared to what we have seen so far. The paper introduced a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment and optimizing a "surrogate" objective function using stochastic gradient ascent. the advantage of this algorithm is that they are much simpler to implement, more general, and has a better sample complexity (empirically). The authors showed that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time on an experimental test of Atari game playing and simulated robotic locomotion tasks.

  https://arxiv.org/pdf/1707.06347.pdf

2 Problem Statement and Optimization problem

2.1 Q-Learning

Before moving to the deep q-network optimization problem lets first remind ourselves on Q-Learning optimization problem. In a finite Markov Decision Process, Q-learning is a model-free technique that directly samples from the environment to learn the ideal optimal policy. It is an off-policy approach that updates its Q-values by selecting the greedy action along the target policy while following a behavior policy that frequently encourages exploration via a ϵ-greedy strategy. Every time step from the present state to the value of the greedy action from the target policy, the Q-value is changed. So optimal action-value function $Q^*(s, a)$ is maximum expected return achievable by following any strategy, after seeing some sequence s and taking some action a, $Q^*(s, a) = max_\pi \, \mathbb{E}[R_t | s_t =$

$s, a_t = a, \pi]$ where π is policy mapping sequences to actions. If the optimal value $Q^*(s', a')$ of a sequence s' at the next time-step was known for all possible actions a', then the optimal strategy is to select the action a' to maximize the expected value [12].

$$Q^*(s, a) = \mathbb{E}[r + \gamma max_{a'} Q^*(s', a')|s, a] \tag{1}$$

Here γ is the discounted rate. A Q-network can be trained by minimising a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i

$$L_i(\theta_i) = \mathbb{E}[(\mathbb{E}[r + \gamma max_{a'} Q(s', a'; \theta_{i-1})|s, a] - Q(s, a; \theta_i))^2] \tag{2}$$

Now the gradient of of the loss function with respect to its weights we have.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}[(r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)] \tag{3}$$

We now take $Y_i^Q = r + \gamma max_{a'} Q(s', a'; \theta_i)$. We update the weights after taking action a in state s and observing an immediate reward of r and resulting state s'[11].

$$\theta_{i+1} = \theta_i + \alpha(Y_i^Q - Q(s_i, a_i; \theta_i))\nabla_{\theta_i} Q(s_i, a_i; \theta_i) \tag{4}$$

2.2 Deep Q-Network

In less intricate settings with small action spaces, Q-learning is fast and straightforward, but complicated games like Super Mario Bros. present the issue of a huge state-action environment, which would confine memory and make it difficult to sample enough state-actions. Deep Q-Network uses convolutional neural network to approximate the Q-value for every state. A neural network's value approximation can lead to instability, which Deep Q Networks addresses with two ways. To begin, goal Q-values are computed using a separate network whose weights are set and synced with the core Q-weights network's on a regular basis, stabilizing the training process.Second, experience replay saves and samples experience tuples (s, a, r, s') from an experience playback buffer at random, removing possible correlations between them. The target network, with parameters θ^-, is the same as the online network except that its parameters are copied every τ steps from the online network, so that then $\theta_i^- = \theta_i$, and kept fixed on all other steps[11]. The target used by DQN is then

$$Y_i^{DQN} = r + \gamma max_{a'} Q(s', a'; \theta_i^-) \tag{5}$$

2.3 Double Deep Q-Network

Standard DQNs fail to identify the optimal policy because they constantly choose actions with the highest values rather than seeking for a new highest value. DDQN separates action selection from target Q-value computation, lowering the risk of overestimation. In our research, we also look at Double Deep Q-learning, which prevents the agent's action value from being overestimated in a stochastic environment.In the original Double Q-learning algorithm, two value functions are learned by assigning each experience randomly to update one of the two value functions, such that there are two sets of weights, θ and θ'. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. For a clear comparison, we can first untangle the selection and evaluation in Q-learning and rewrite its target (4) as

$$Y_i^Q = r + \gamma Q(s'; argmax_{a'} Q(s', a'; \theta_i); \theta_i) \tag{6}$$

Notice that the selection of the action, in the argmax, is still due to the online weights θ_i. This means that, as in Q-learning, we are still estimating the value of the greedy policy according to the current values, as defined by θ_i. However, we use the second set of weights θ_i' to fairly evaluate the value of this policy. This second set of weights can be updated symmetrically by switching the roles of θ and θ' [11].

$$Y_i^{Double-Q} = r + \gamma Q(s'; argmax_{a'} Q(s', a'; \theta_i); \theta_i') \tag{7}$$

2.4 Double Deep Q-Network with Prioritized Experience Replay

Reusing experiences stabilizes training and improves sampling efficiency, which is why experience replay is so important in Deep Q-learning. Prioritized experience replay has been shown to increase training efficiency by biasing the sample distribution on the TD error and therefore improving the recycling of prior experiences. This is especially essential in games like Super Mario Bros., since the substantial negative reward that comes with dying is both scarce and valuable. We trained our agent using both strategies to confirm that PER produced superior outcomes than uniform experience replay. For a DDQN method with and without prioritized replay, the TD error, loss, and reward are calculated. In comparison to uniform experience playback, prioritized replay shows smaller variance in TD error and faster convergence to zero, better average reward, and a substantially faster drop in loss to zero, indicating higher training stability. To overcome uniform and greedy experience sampling issues, we introduce a stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling. We ensure that the probability of being sampled is monotonic in a transition's priority, while guaranteeing a non-zero probability even for the lowest-priority transition. Concretely, we define the probability of sampling transition i as

$$P(i) = \frac{p_i^\alpha}{\Sigma_k p_k^\alpha} \tag{8}$$

Here $p_i > 0$ is the priority of transition i. α describes the degree of prioritization where α=0 means a uniform case [9].

2.5 Proximal Policy Optimization

We wanted to train also train policy based methods to compare against value-based methods. The major limitation of a value based as what we have seen so is that these methods work better when action space is finite. However, in most of the real-world tasks, such as training an autonomous car to navigate, the action space is continuous which we cannot address using value-based methods. Policy-based methods are useful if the action space is continuous. Here we optimize a policy without using a value function. However, it needs a lot of training samples and thus leads to slow learning. Proximal Policy Optimization is one such popular policy based algorithm. PPO is easier to code and tune, sample efficient and performs comparably or better than other SOTAs. Moreover, unlike DQN, which learns from stored offline data, it can learn online without using a replay buffer that stores past experiences. That means, using PPO, the agent learns directly from the environment, and once it uses a batch of experiences, it discards that batch after doing a gradient update. The main idea of PPO is to avoid large policy update during training by computing the ratio between the old and the current policy. We define the policy gradient loss in a policy objective function as the expectation of the log of the policy times the Advantage function [5].

$$L^{PG}(\theta) = \mathbb{E}_i[log\pi_\theta(a_i|s_i)A_i] \tag{9}$$

Here π_θ is the policy, A_i estimates the relative value of action in the current state. The problem with policy gradient is that training using a single batch may destroy the policy since a new policy can be completely different from the older one, and thus it may have generalization problem. So, in order to make the new policy closer to the old policy, KL divergence constraint is added during the policy update.

$$\begin{aligned} maximize_\theta \, \mathbb{E}_i[\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)}A_i] \\ subjectto \, \mathbb{E}_i[KL[\pi_{\theta_{old}}(\cdot|s_i), \pi_\theta(\cdot|s_i)]] \leq \delta \end{aligned} \tag{10}$$

PPO used Clipped Surrogate Objective function for policy update. It uses ratio between the newly updated policy and old updated policy.

$$L^{CPI}(\theta) = \mathbb{E}_i[\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)}A_i] = \mathbb{E}_i[r_i(\theta)A_i] \tag{11}$$

Here CPI is conservative policy iteration. Now we clip above function so that large policies are avoided.

$$L^{CLIP}(\theta) = \mathbb{E}_i[min(r_i(\theta)A_i, clip(r_i(\theta), 1-\epsilon, 1+\epsilon)A_i] \tag{12}$$

Finally PPO Clipped Surrogate Objective Loss Function is as following. The second term is a squared error loss between the current and target estimate. The final term is an entropy term, that ensures the Agent to do enough exploration.

$$L_i^{CLIP+VF+S}(\theta) = \mathbb{E}_i[L_i^{CLIP}(\theta) - c_1 L_i^{VF}(\theta) + c_2 S[\pi_\theta](s_i)] \tag{13}$$

3 Methodology and Algorithms

3.1 Methodology

3.1.1 Model Architecture

We employed the usual Deep Q Network algorithm to train our agent, and we investigated the effects of Double DQN, dueling networks, and prioritized experience replay followed by Proximal Policy Optimization. The Stable Baselines library, which includes various implementations of common reinforcement learning algorithms [1], was used. Stable Baselines is based on Open AI's Baselines, which is a library for the Open AI Gym [7] RL framework.

3.1.2 Environment

We used gym-super-mario-bros, a custom environment created for Open AI Gym [6], to simulate the game. There are eight worlds in all, each with four levels, for a total of 32 different environments. The raw pixel data from each frame of the game serves as our state space, from which we make observations. The whole NES action space of 256 actions is condensed into action lists of 12 or less actions, with which we experimented with selecting one of the seven simple movement action sets. Moving left and right, as well as jumping, are among the options available.

3.1.3 Rewards

The agent is rewarded for moving to the right and progressing towards the goal. It is penalized for the passage of time and heavily penalized for death. Rewards are clipped between larger values to limit the scale of loss, thus capping the back-propagated gradient and providing more stable training. While clipping rewards between -1 and 1 (much smaller than what we took) is a common method used by papers such as Deepmind's 2015 Atari agent, we felt that the range of representation was not enough to differentiate between different rewards. For example, we needed death to produce a much more negative reward than passage of time. In designing the reward function, we tried not to use highly engineered features, instead using features that felt natural from a human perspective. For example, we would not reward jump height (which would greatly speed up training for stages with tall pipes), since an actual human learning the game for the first time would not naturally be incentivized to jump high and there may end up being no tall pipes at all in any stage. We chose to penalize passage of time only in the second half of training, because including this penalty for the entire training would cause the agent to devolve at times into learned helplessness, whereby it would choose to prematurely die in order to avoid suffering further penalties [2]. Because we valued time only under the context that the level has already been beaten (since we want Mario to beat the level as quickly as possible), we first trained without time penalty to ensure Mario beats the level. We later included time penalty to ensure Mario beats the level as quickly as possible. This is a minimally engineered decision since it is in the natural flow of a human learning the game to first attempt to beat the level before doing speed runs.

3.1.4 Data Set

For the experimentation purpose of this research we used Open AI Gym to create the game environment. We would use library gym-super-mario-bros to create the game environment and use nes-py to get the joypad controls to take actions in the environment.

3.2 Algorithms

3.2.1 DQN

Algorithm 1 Deep Deep Q-learning with Experience Replay

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights
for $episode = 1, M$ do
    Initialise sequence $s_1 = x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        Otherwise select $a_t = max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = (s_t, a_{t,t+1})$ and process $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ \text{r}_j + \gamma max_{a'} Q(\phi_{j+1}, a^{'}; \theta), & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

3.2.2 DDQN

Algorithm 2 Double Deep Q-learning (Hasselt et al., 2015)

```
Initialize primary network $Q_\Theta$, target network $Q_{\Theta'}$, replay buffer D, $\mathcal{T}$ «1;
for $each\ iteration$ do
    for $each\ environment\ step$ do
        Observe State $s_t$ and select $a_t \sim \pi(a_t, s_t)$;
        Execute $a_t$ and observe next state $s_{t+1}$ and reward $r_t = R(s_t, a_t)$;
        Store $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $D$
    end for
    for $each\ update\ step$ do
        Sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim D$;
        Compute target Q value: $Q^*(s_t, a_t) \approx r_t + \gamma Q_\Theta(s_{t+1}, argmax_{a'} Q_{\Theta'}(s_{t+1}, a^{'}))$;
        Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\Theta(s_t, a_t))^2$;
        Update target network parameters: $\Theta^{'} \leftarrow \mathcal{T} * \Theta + (1 - \mathcal{T}) * \Theta^{'}$;
    end for
end for
```

3.2.3 Prioritized Experience Replay

Algorithm 3 Double DQN with proportional prioritization

```
**Input:** minibatch $k$, step-size $\eta$, replay period $K$ and size $N$, exponents $\alpha$ and $\beta$, budget $T$
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ to $T$ do
    Observe $S_t, R_t, \gamma_t$
    Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = max_{i<t} p_i$
    if $t \equiv 0 \mod K$ then
        for $j = 1$ to $k$ do
            Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_j^\alpha$
            Compute importance-sampling weight $w_j = (N.P(j))^{-\beta} / max_i w_i$
            Compute TD-error $\delta_j = R_j + \gamma_j Q_{target}(S_j, argmax_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
            Update transition priority $p_j \leftarrow |\delta_j|$
            Accumulate weight-change $\Delta \leftarrow \Delta + w_j.\delta_j.\nabla\theta Q(S_{j-1}, A_{j-1})$
        end for
        Update weights $\theta \leftarrow \theta + \eta.\Delta$, reset $\Delta = 0$
        From time to time copy weights into target network $\theta_{target} \leftarrow \theta$
    end if
    Choose action $A_t \sim \pi_\theta(S_t)$
end for
```

3.2.4 Proximal Policy Optimization

Algorithm 4 PPO, Actor-Critic Style

```
for $iteration = 1, 2, ...$ do
    for $iteration = 1, 2, ..., N$ do
        Run policy $\pi_{\theta_{old}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{old} \leftarrow \theta$
end for
```

4 Experiments and Results

4.1 Experimental Setup

The experiment was done on the environment as stated above. Multiple environment wrappers were used which will be discussed below.

4.1.1 Warpframe

The observation area in Super Mario Bros. is huge. The resolution of the NES is 256×240 pixels.We can see that our observation space contains 1×10^{12} distinct states since each pixel has three channels ranging from 0 - 255. We downsample our observation space by converting the $256 \times 240 \times 3$ picture to an $84 \times 84 \times 1$ grayscale image to boost training efficiency. Due to the drastically decreased input size, we find that approach significantly increases training speed while keeping equivalent performance.

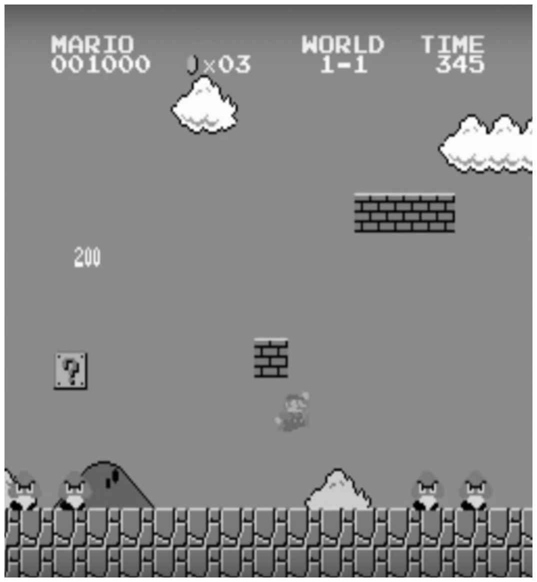
Figure 1: Resulting Frame After Applying Warpframe Wrapper

4.1.2 Framestack

At each timestep, the reinforcement learning agent sees just a single frame. It's tough to record movement in the scene and hence determine whether the player character is in danger using only a single frame. In Super Mario Bros., this is crucial since Mario's movement has momentum, which may force him to slide into barriers or foes. By concatenating the previous four frames as input into our DQN, we aim to capture velocity and speed information in the game. Intuitively, this allows our model to appropriately capture movement information and Mario's movement states, such as jumping, falling, and moving quickly or slowly to the left or right. This offers our model a significant performance gain, confirming our suspicion that velocity information is critical to playing Super Mario Bros. successfully.

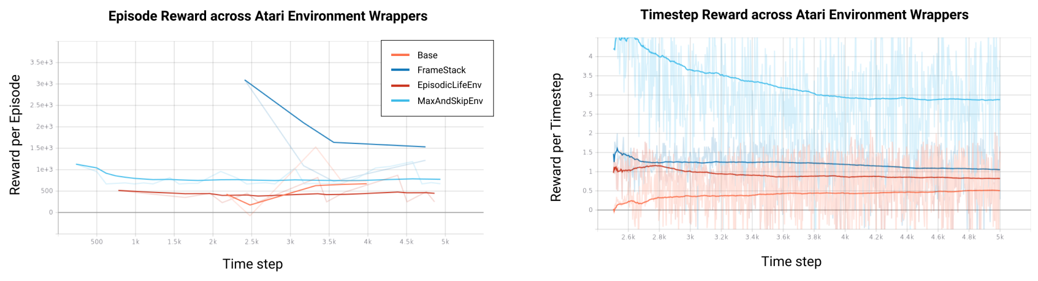

Figure 2: Reward per timestep and per episode for each environment wrapper used in isolation

4.1.3 Frame Skipping

The agent only observes and picks an action every kth frame using this wrapper, and the selected action is repeated for the frames in between each observation. This significantly speeds up training by k times since picking an action involves a model prediction and is considerably more computationally complex than producing a single frame. The value of k in our project is set to 8. Another advantage of skipping frames is that it allows an agent to continually attain a maximum height leap, which is the only method to go over tall pipes, which was a big hurdle before frame skipping was adopted.

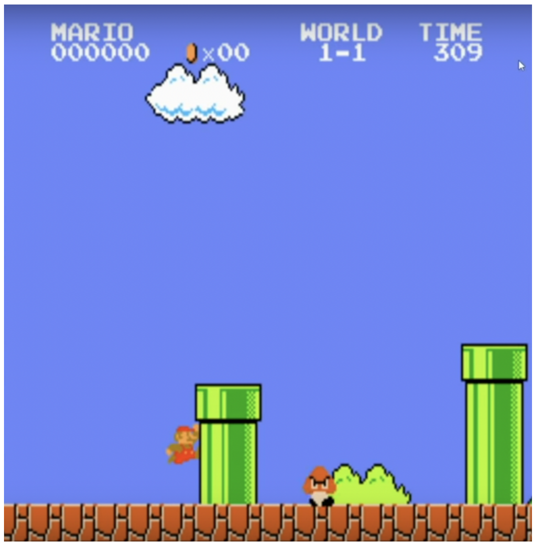

Figure 3: Example of a Tall Pipe

A maximum height leap necessitates 24 frames of jump actions in a row.Without frame skipping, the agent would have to learn to pick a jump action 24 times in a row, which is an extremely unlikely event that would take the agent a long time to master. We only need the agent to pick a jump action three times in a row if we skip 8 frames at a time, which is considerably easier for Mario to master. Mario's moves are less accurate, which is a disadvantage, but we discovered that this accuracy is not required for him to complete the level.

4.1.4 Episodic Life

The player in Super Mario Bros. has three attempts, or "lives" to complete the game. Mario loses a life when he comes into touch with an adversary or a fatal obstacle, falls down the bottom of the level, or runs out of time. Mario is put at the start of each life at the start of the level. If you lose all three lives, the game is finished. Our environment's default setup treats losing all three lives as a single episode. This can be troublesome since the agent will try three times to maximize the average payment. As a result, an agent that advances far in the level on the first try but fails early on future efforts will be judged worse than an agent who just advances a modest distance on all three attempts. Making each episode equal to the duration of a single in game life is a last change to our ecosystem. This aids convergence since performance is determined by the agent's performance on a single level try rather than the average of three efforts.

4.1.5 Model Training Details

As mentioned earlier Stable Baselines 2 was used for implementations of the various reinforcement algorithms. Stable Baselines is library developed by Open AI that contains the efficient implementation of famous RL algorithms. Below is the summarized table containing hyperparamter and other implementation details.

	DQN	DDQN	DDQN with PER	PPO
Cloud Provider	Google Colab	Google Colab	Google Colab	Google Colab
GPU	Tesla K4	Tesla K4	Tesla K4	Tesla K4
Data	Super Mario Bros-v0	Super Mario Bros-v0	Super Mario Bros-v0	Super Mario Bros-v0
Control Set	Simple Movement	Simple Movement	Simple Movement	Simple Movement
Grayscaled	Y	Y	Y	Y
Frame Stacked	4	4	4	4
Frame Skipped	8	8	8	8
Policy	LnCnnPolicy	LnCnnPolicy	LnCnnPolicy	CnnPolicy
Batch Size	512	512	512	512
Double_Q	N	Y	Y	N/A
Prioritized replay	N	N	Y	N/A
Prioritized alpha	N/A	N/A	0.6	N/A
Learning Rate	1e-4	1e-4	1e-4	1e-4
Exploration Fraction	0.1	0.1	0.1	N/A
Tensor board Use	Y	Y	Y	Y
Timesteps	50000	50000	50000	50000

4.2 Results

We trained one agent primarily on Stage 1 for 50000 time steps. After experimenting we noticed that Mario learned to reach to the end of the level by the end of 50000 time steps, after which it began optimizing its runs to avoid time penalties. Figures 5 and 6 show the reward and loss for these two agents as they progress through training; the continual increase in reward and decrease in loss suggests that more training is conceivable.

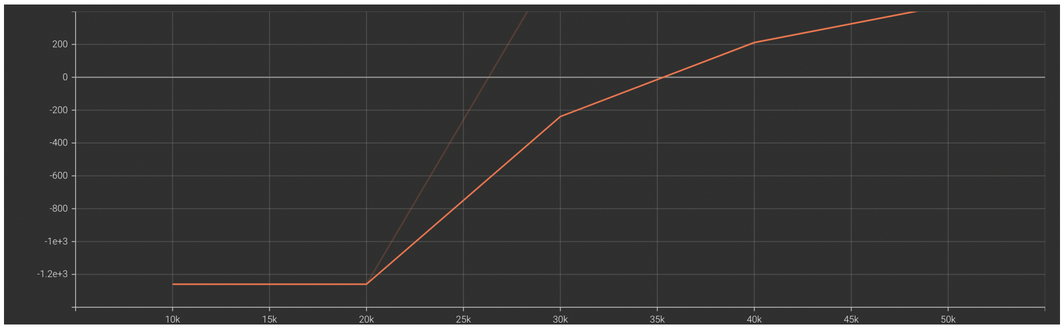

Figure 4: Average Reward Over 50K Time-Steps

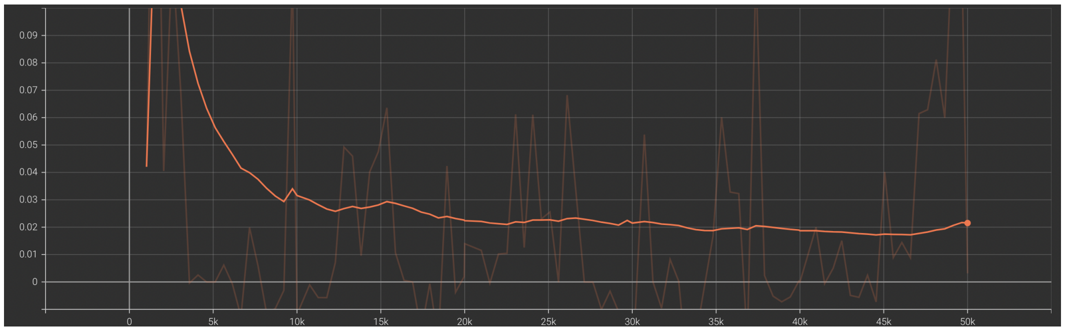

Figure 5: Gradient Loss Over 50K Time-Steps

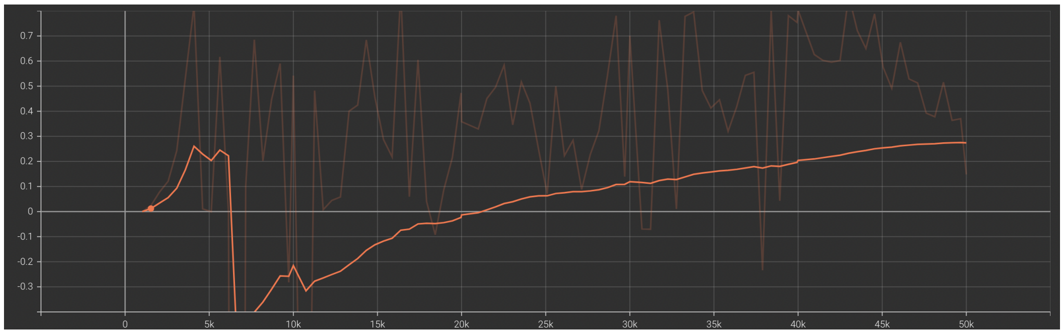

Figure 6: Explained Variance Over 50K Time-Steps

We let each of our agents play for 100 episodes to evaluate their performance. Stage 1 was played 100 times. We also examined an agent that chooses random actions and a beginner human player for 100 episodes each to compare our results. We have put the graphical results primarily for PPO, but the results we got in our study for DQN, DDQN and DDQN with PER. We have put other supplemental implementations and results in our Github repository.

5 Conclusion

After conducting our study we found that Double Deep Q-Learning with prioritised replay performs very well after applying the enhancements in terms of environment wrappers. The proximal policy optimisation also performs relatively well and takes lesser time and resources to train. We see that our agent is able to perform mechanically precise tasks. We only trained for one of the levels of Super Mario Bros., hence we saw that the agent was performing poorly compared to the state of the art available. We believe that if the agent is trained for more time steps and on random levels the agent might start recognising patterns intrinsic to the game, become level agnostic and even start surpass human level performance. Q-Learning and its improvements like DQN and DDQN and policy methods like PPO have been shown to achieve good results in Atari 2600 games. Our study analysing these methods show promise with significantly more complex environments.

6 Future Scope

We can extend our study to other reinforcement learning algorithms like A2C, DDPG, GAIL, TRPO etc. We analysed the most famous algorithms in the field but it may be possible that the less sought after algorithm can work better for our given situation. Another way to extend would be to train the agent for longer duration/time step and on random levels and worlds of Super Mario Bros. We can also explore the recent advancements in DQN algorithm such as distributed agents like ApeX and Gorila DQN and Recurrent Replay Distributed DQN which leverages artificial recurrent neural networks specifically Long Short Term Memory (LSTM). This is a more promising path since in our analysis we saw that DQN improvements improved agent performance.

Acknowledgment

We would like to thank Professor Kejun Huang for the course CAP 6610:Machine Learning and providing us the opportunity to apply our knowledge by building a project for the course. Each member contributed equally to the implementation, analysis and documentation of this study.

Checklist

1. For all authors...
    (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Refer section 4.2
    (b) Did you describe the limitations of your work? [Yes] Refer section 6 Future Scope

(c) Did you discuss any potential negative societal impacts of your work? [N/A]

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Refer section 3.1 Methodology

   (b) Did you include complete proofs of all theoretical results? [Yes] Refer section 4.2 Results

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code for this project is available at https://github.com/KulkarniVaibhav/Reinforcement-Learning-Algorithms-to-Play-Mario.

   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Refer section 4.1.5 Model Training Details

   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]

   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Refer section 4.1.5 Model Training Details

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [N/A]

   (b) Did you mention the license of the assets? [N/A]

   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Refer supplemental material and assets at https://github.com/KulkarniVaibhav.

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] The Data/Environment used for this research does not contain any personally identifiable information or offensive content

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

[1] Maximilian Ernestus Adam Gleave Anssi Kanervisto Rene Traore Prafulla Dhariwal Christopher Hesse Oleg Klimov Alex Nichol Matthias Plappert Alec Radford John Schulman Szymon Sidor Ashley Hill, Antonin Raffin and Yuhuai Wu. Stable-baselines. 2018. https://github.com/hill-a/stable-baselines.

[2] Vali Derhami and Zahra Youhannaei. Demonstration of learned helplessness with fuzzy reinforcement learning. *GitHub*, (12), 2008.

[3] Jack Zhang Edward Zhang, Stephanie Doan. Mari/o 2.0. *UCLA*, 2020.

[4] Erik D. Demaine Greg Aloupis and Alan Guo. Classic nintendo games are (np-)hard. *CoRR,abs/1203.1895*, 2012.

[5] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms. *OpenAI*, 2017.

[6] Christian Kauten. Super mario bros for openai gym. *GitHub*, 2018.

[7] Oleg Klimov Alex Nichol Matthias Plappert Alec Radford John Schulman Szymon Sidor Yuhuai Wu Prafulla Dhariwal, Christopher Hesse and Peter Zhokhov. Openai baselines. 2017. https://github.com/openai/baselines.

[8] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 1995.

[9] Ioannis Antonoglou Tom Schaul, John Quan and David Silver. Prioritized experience replay. *Google DeepMind*, 2016.

[10] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions*, 1997.

[11] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Google DeepMind*, 2015.

[12] David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Volodymyr Mnih, Koray Kavukcuoglu and Martin Riedmiller. Playing atari with deep reinforcement learning. *Google DeepMind*, 2013.