applied sciences

MDPI

Article

Performance Analysis of Reinforcement Learning Techniques for Augmented Experience Training Using Generative Adversarial Networks

Smita Mahajan [1,*], Shruti Patil [1], Moinuddin Bhavnagri [1], Rashmi Singh [1], Kshitiz Kalra [1], Bhumika Saini [1], Ketan Kotecha [2] and Jatinderkumar Saini [3]

[1] Department of Computer Science Engineering, Symbiosis International (Deemed) University, Pune 412115, Maharashtra, India
[2] Department of Symbiosis Centre for Applied AI (SCAAI), Symbiosis Institute of Technology, Symbiosis International (Deemed) University, Pune 412115, Maharashtra, India
[3] Symbiosis Institute of Computer Studies and Research, Symbiosis International (Deemed) University, Pune 412115, Maharashtra, India
* Correspondence: smita.mahajan@sitpune.edu.in

Abstract: This paper aims at analyzing the performance of reinforcement learning (RL) agents when trained in environments created by a generative adversarial network (GAN). This is a first step towards the greater goal of developing fast-learning and robust RL agents by leveraging the power of GANs for environment generation. The RL techniques that we tested were exact Q-learning, approximate Q-learning, approximate SARSA and a heuristic agent. The task for the agents was to learn how to play the game Super Mario Bros (SMB). This analysis will be helpful in suggesting which RL techniques are best suited for augmented experience training (with synthetic environments). This would further help in establishing a reinforcement learning framework using the agents that can learn faster by bringing a greater variety in environment exploration.

Keywords: reinforcement learning; generative models; generative adversarial networks; Q-learning; SARSA; Super Mario Bros

Citation: Mahajan, S.; Patil, S.; Bhavnagri, M.; Singh, R.; Kalra, K.; Saini, B.; Kotecha, K.; Saini, J. Performance Analysis of Reinforcement Learning Techniques for Augmented Experience Training Using Generative Adversarial Networks. *Appl. Sci.* **2022**, *12*, 12923. https://doi.org/10.3390/app122412923

Academic Editors: Maxim Mozgovoy, João M. F. Rodrigues and Paolo Burelli

Received: 15 September 2022
Accepted: 30 November 2022
Published: 16 December 2022

1. Introduction

Reinforcement learning techniques are usually experience-based rather than data-based. Hence, to train RL agents, they should be left to explore either in a sandboxed real environment or a simulation. While sandboxed real environments might not always be made available, simulations are an optimal solution [1]. However, this itself requires explicit effort to be made at times. It might not be possible to bring in a variety in the training environment for RL agents. These environments are only as good and as varied as the mind(s) behind the environment design can fathom. Generative models are useful for this, particularly generative adversarial networks. generative adversarial networks (GANs) were originally used as generative models for unsupervised learning. However, they have now found use cases in semi-supervised learning and fully supervised learning, as well as reinforcement learning. GANs have been conventionally used to generate new plausible examples for data sets and related tasks, majorly revolving around images and text. Generative adversarial networks, developed and launched in 2014, are a part of neural networks that do not require any supervision for learning. GANs have a system structure comprising two neural networks models, which are the generator and discriminator. The communication between these two models helps them to analyze the variations in the data set provided. The generator makes an attempt to mislead the discriminator by generating fake data samples (Figure 1). The discriminator tries to demarcate between the fake and real data samples. This is a repetitive process and hence improves the efficiency of the generator and discriminator. Some of the real-world applications of GANs include: creating

new image samples from existing samples, text or image translation, image editing and applications in the field of AI for generating video games scenes.

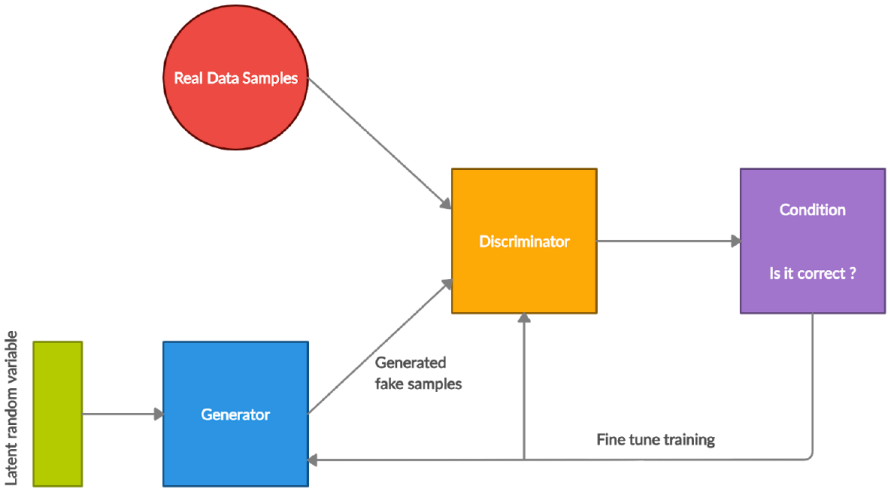

Figure 1. Working of GANs.

We recognize the use of GANs in reinforcement learning as an area of opportunity for training environment synthesis and augmentation. Augmented experience training refers to training RL agents by augmenting their environment exploration experiences. Technically, this translates to an RL agent being trained on variations in a domain environment so that the agent can converge with a good decision strategy sooner and can be more robust when variations and exceptions occur in real time in the actual environments. For example, imagine the case of autonomous driving. Some training environments would be:

1. A straight road with a starting point and ending point;
2. A curved road with a starting point and ending point.

It seems obvious to also train the agent for sharp turns as well as gentle turns. One way to achieve this could be by developing environments manually for each use case: sharp turns as well as gentle turns. However, for a more creative solution, GANs can be used to introduce variations in our environments. This is more scalable than a manual solution.

2. Motivation and Related Work

Our main motivation is to validate the possibilities of developing fast-learning and more robust RL agents using environment augmentation with GANs. While there has been work concerning improving the performance of RL agents and GANs, the feasibility of environment augmentation via GANs seems to be slightly unexplored. Vincent Huang et al. [2] proposed the enhanced training of RL agents by using EGANs to initialize the agents in order to make them learn faster. The EGAN approach could be used to speed up the early training phases for real-time systems with sparse and slow data sampling. Consequently, agents trained with EGAN and GAN achieved faster increases in rewards, and both could provide more modalities in data space. We identified that suitability is only mentioned for wireless communication. Exploration is required for applying the technique to other domains. Further work is needed to verify and fine-tune the system to achieve an optimal performance. Thang Doan et al. [3] proposed using the GAN environment for GAN-Q-learning to analyze its performance and provide a novel distributional RL method. This is an alternative to the traditional models. The end results state that Q-learning goes along with the benchmarked algorithms, such as GAN-Q-learning. A CNN was suggested in order to use rewards, and a high priority for the stability of future iterations should be carried out. Tim Salimans et al. [4] provided many training techniques that generate realistic human images. The methods provided can train RL agents, which were previously untrainable. Using the methods proposed, we can use them for training RL agents, reducing the training

time. Saldanha, J. et al. [5] presented their effort to synthesize pulmonary sounds of various categories utilizing variations in variational autoencoders, such as multi-layer perceptron VAE (MLP-VAE), convolutional VAE (CVAE) and conditional VAE. They compared the impact of supplementing the unbalanced dataset on the performance of different lung sound classifiers. To identify and categorize dementia into different groups based on its prominence and severity in the available MRI images, Jain V. et al. [6] suggested a unique DCGAN-based augmentation and classification (D-BAC) model technique. They suggested using a new GAN-augmented dataset to study the early detection of dementia, also known as mild cognitive impairment (MCI) and referred to as MCI. Machine learning, especially reinforcement learning, has recently found widespread use in wireless communications in the prediction of network traffic, identifying the quickest and shortest path while sending the data over the networks [7,8]. The model-free methodology known as Q-learning has been acknowledged as promising, mainly when used for complex decision-making processes. Excellent learning capabilities, successful outcomes and the ability to integrate with other models are just a few advantages of Q-learning [9]. According to Saeed Kaviani et al., DeepCQ+ routing [10], which innovatively integrates emerging multi-agent deep reinforcement learning (MADRL), achieves a consistently higher performance across various MANET architectures when training on only a small number of network parameters and conditions. Xinyu You et al. created a novel packet routing architecture based on multi-agent deep reinforcement learning in a completely decentralized setting. Each router in this system is equipped with a recurrent neural network (RNN) for training and decision making and an independent long short-term memory (LSTM) [11]. The Q-values associated with each potential course of action for a given input state serve as the targets of an input state in deep Q-learning. Q-values are the current estimates of the total future rewards obtained for each activity. These Q-values are determined via the temporal difference formula [12].

Ruiqi Wu et al. [13] trained a model to write "Chinese Calligraphy" via images from calligraphy books to eliminate the human involvement in the training and learning process of the robot. They were successful in creating a model that writes calligraphy strokes using images of calligraphy. The approach used to train the model had the following characteristics:

- An accurate reward algorithm was used to make the learning process efficient and take less time to train.
- In the algorithm proposed, the training actions used were simplified by making use of continuous values, which decreases the learning complexity, and the trained model produced actions directly without considering the time taken for writing the actions.

Guillermo Caminero et al. [14] proposed that we can detect intrusion by using adversarial reinforcement learning. In addition, according to them, the training time for adversarial reinforcement learning is less compared to other algorithms. Hence, adversarial reinforcement learning is a relative fit for performing predictions online, and is mainly suitable for data sets with unbalanced data and in situations where a high prediction accuracy is required. Chelsea Finn et al. [15] demonstrated equivalence between a maximum entropy inverse reinforcement learning (IRL) algorithm and a GAN model to evaluate the generator density. An equation was derived that used a form of discriminator that supports the values from the generator, leading to an unbiased estimation of the underlying energy equation. Justin Fu et al. [16] proposed adversarial inverse reinforcement learning (AIRL), which is a pragmatic and scalable IRL calculation, in view of an adversarial reward learning formulation. They showed that AIRL can regain reward works that are resilient to variations in the environment when training. AIRL is a functional and adaptable IRL calculation that can learn released rewards and significantly improves upon both prior imitation learning and IRL calculations. Karol Hausman et al. [17] proposed a multi-modal imitation learning framework that can segment and imitate skills from unlabelled and unstructured demonstrations. They came up with a method of imitation learning that learns a multimodal theoretical policy and successfully imitates automatically segmented

tasks. The imitation learning method performs this with the help of unstructured and undefined demonstrations. Ali Taleb Zadeh Kasgari et al. [18] proposed a GAN approach that helps in pretraining deep-RL with the use of synthetic and real data, thus developing an evolved deep-RL framework by exposing it to broad network conditions. The outcome was that, during extreme conditions, the proposed, experienced deep-RL agent can recover instantly, while a conventional deep-RL, under data rate restriction, considers epochs of high reliability and low latency. Michal Uricar et al. [19] proposed GANs to be applied in autonomous driving and advanced data augmentation. In the paper, the generator was trained in such a way that, after some cycles of epochs, it started to identify which part of the image was soiled. They trained a generator model that is capable of "de-soiling" the image as well as introducing some soiling to the image. Some gaps in the study were that it was not a diverse data set and that over-fitting was observed.

3. Problem Specification

For our analysis, we opted for Super Mario Bros (SMB) as our domain of concern. Super Mario Bros' aim is fairly straightforward: to start from one side of the stage and reach the target while making progress in the rewards earned. Our aim was a comparison of RL agents that are given the task of successfully navigating through an augmented environment. Mario usually needs to move right to achieve success. However, only moving right is not an ideal strategy. The goal of the Mario agent is to overcome each of the challenges in the form of obstructions in the path of its target while collecting coins. These kinds of issues make the job at hand considerably more complex. The attributes that must be necessarily specified for the formulation of Super Mario Bros as a reinforcement learning challenge are: state, action and reward spaces.

3.1. State Representation

A network of thirteen × sixteen tiles was used to represent the state of the agent. The numeric representation corresponding to the tile network is shown in Table 1.

Table 1. Representation of Mario's state.

Number in Tile Grid	Representation in SMB
0	Empty space
1	Entity (coin, field, pipe, etc.)
2	Enemy
3	Mario

3.2. Action Space

The NES game controller consists of six buttons that can either be hit or not. Though there can be various possible combinations of these buttons, only a few of the combinations are useful in the game. We can take the example of pressing the move forward and move backward together, which is as good as no action. The useful actions that we can perform via NES controller are represented in the form of a discrete action space with the mapping as shown in Table 2:

Table 2. Discrete action space with mapping.

Key	Action	Description
3	[0, 1, 0, 0, 0, 0]	Move Backward
4	[0, 1, 0, 0, 1, 0]	Jump Backward
5	[0, 1, 0, 0, 0, 1]	Sprint Backward
6	[0, 1, 0, 0, 1, 1]	Sprint + Jump Bk
7	[0, 0, 0, 1, 0, 0]	Move Forward
8	[0, 0, 0, 1, 1, 0]	Jump Forward
9	[0, 0, 0, 1, 0, 1]	Sprint Forward
10	[0, 0, 0, 1, 1, 1]	Sprint + Jump Fw
11	[0, 0, 0, 0, 1, 0]	Jump

3.3. Reward Function

Agents obtain a reward or a positive value when the agents perform certain actions correctly. In the Gym environment, their basic gain is based on the distance covered by the agent. The amount of distance covered by the agent determines the award for the agent. It all depends on distance (δ).

$$R(s) = \delta(s) - \delta(s-1) \tag{1}$$

The agent is rewarded a unit positive for moving in the direction of progress and unit negative for moving backward. No reward is given to the agent if no movement is detected.

$$R(s) = \delta(s) - \delta(s-1)\ if\ \delta(s) - \delta(s-1) > 0 \tag{2}$$

$$R(s) = \delta(s) - \delta(s-1) - 1\ if\ \delta(s) - \delta(s-1) \leq 0 \tag{3}$$

Along with the calculation of the distance rewards, customized rewards are also given based on the occurrence of certain events. The values of these rewards consist of random parameters, which are adjusted for further use.

- When the agent is killed, 100 is deducted.
- When the score of an agent increases by x as a result of gaining points or overcoming an obstruction, the scoring factor is multiplied to x and then added.
- If there is a large gap occurring mid-way through the level and Mario crosses it, then 500 is added.

4. Approach

4.1. GAN Model

A corpus of Super Mario Bros is used for the training of the GAN model [20]. The trained GAN model is then able to generate new SMB segments upon feeding the numeric input from the corpus. These segments together make up a SMB level that is given to the RL agent for exploration via Gym. This process is elaborated on as follows:

- With the help of some given reference, the GAN model generator will generate some environment.
- The generated environment is now fed into the discriminator.
- If the discriminator succeeds in recognizing the environmental fake-ness, then the cycle is complete and the environment is reconsidered, and the generator produces a new, possibly more realistic environment.
- If the discriminator is unable to differentiate between the real environment created and the synthetic environment, then the synthetic environment will be used more.
- In these synthetic environments, the reinforcement learning agent is trained.
- Different RL agents are trained on the environment, and a comparison between their performance is made.

GAN Implementation

GANs, being deep neural networks, fall into the category of unsupervised learning, which return phenomenal results in producing images from an existing set of images. In addition, by applying a covariance matrix adaptation evolutionary strategy, also known as CMAES, the levels generated by GANs are searched for, these levels having certain specific attributes. The flow of the process is shown in Figure 2.

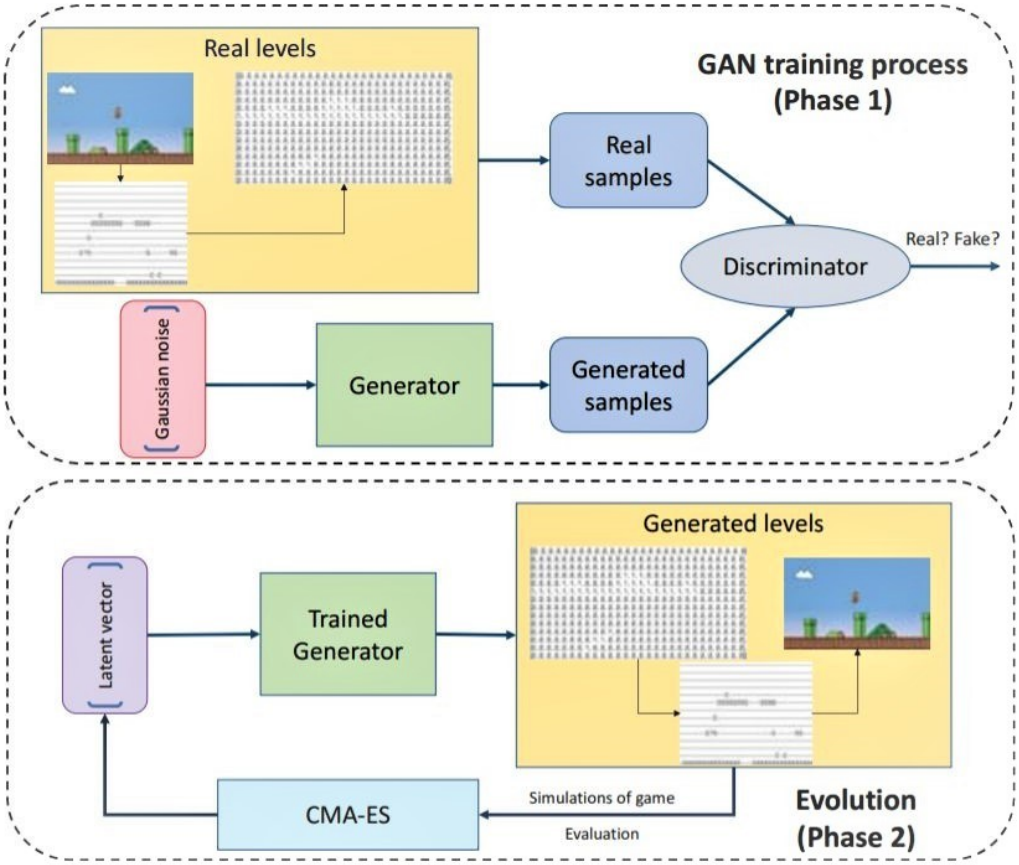

Figure 2. Proposed workflow for classification and validation of plant.

The approach is divided into two phases:

1. The GAN was trained on real levels of Mario that were converted from graphical representation into vector character representation, where each tile of the level is represented with a unique character.
2. In the second phase, we used a covariance matrix adaptation evolution strategy (CMAES) to find the latent vector suitable for us with certain attributes and properties, as certain vectors of the levels generated are unplayable.

This means that we trained our GANs model on the original Super Mario Bros, available as part of the Video Game Level Corpus (VGLC) and then used CMA-ES for finding ideal vectors that were again fed to the trained model that generates playable Mario levels.

Level representation. The levels of Mario are represented in different ways all over VGLC and AI frameworks, which have tile representations in common. The levels present in the Video Game Level Corpus are represented using a character symbol for every tile and aspect of the level, i.e, the bricks are represented with one character, and then enemies are represented with another character. By following this convention, VGLC processed the graphical level (Figure 3) into character representation, which can be further used in training RL algorithms or other usages.

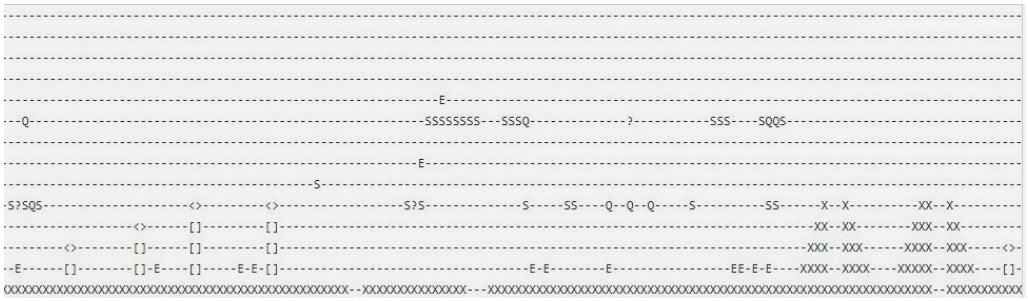

Figure 3. Level representation of Mario Bros in VGLC.

Furthermore, as we need to feed these levels into the GAN model for training, we converted this character representation into an integer representation, where the tiles are represented as an integer, and then converted this integer-level representation into a single encoded integer vector that was provided as an input to the discriminator of GAN. The output by the generator was also a single vector of integers in which, as discussed above, all of the tiles of the level generated were represented as an integer. Then, this obtained vector was given in AI Gym in order to render it into graphical representations.

As discussed earlier, the above Figure 4 shows the character-symbol-to-integer conversion representation. This conversion was performed in order to feed the levels into the GAN model, and was also required for the AI framework to produce the level's graphical representations.

Tile type	Symbol	Identity	Visualization
Solid/Ground	X	0	
Breakable	S	1	
Empty (passable)	-	2	
Full question block	?	3	
Empty question block	Q	4	
Enemy	E	5	
Top-left pipe	<	6	
Top-right pipe	>	7	
Left pipe	[	8	
Right pipe	]	9	

Figure 4. Tile encoding used for conversion into vector.

Custom OpenAI Gym Implementation. The official SMB Gym could not be used as we were feeding the synthetic environments from our GAN module to our RL agents. Hence, a custom Gym was required to be developed. Custom environments can be created in OpenAI Gym (Figure 5) by structuring the source for the custom Gym as a git repository for an installable pip package. The minimal required directory structure that enables Gym to recognize and provision custom environments is shown in Algorithm 1.

After the customization of the environment, the environment picked up a synthetic level from the GAN module and returned 16×13 integer arrays as state representations of the game states to the RL agent throughout the game play. Visualizing it would render something similar to the following pixelated level (Figure 6).

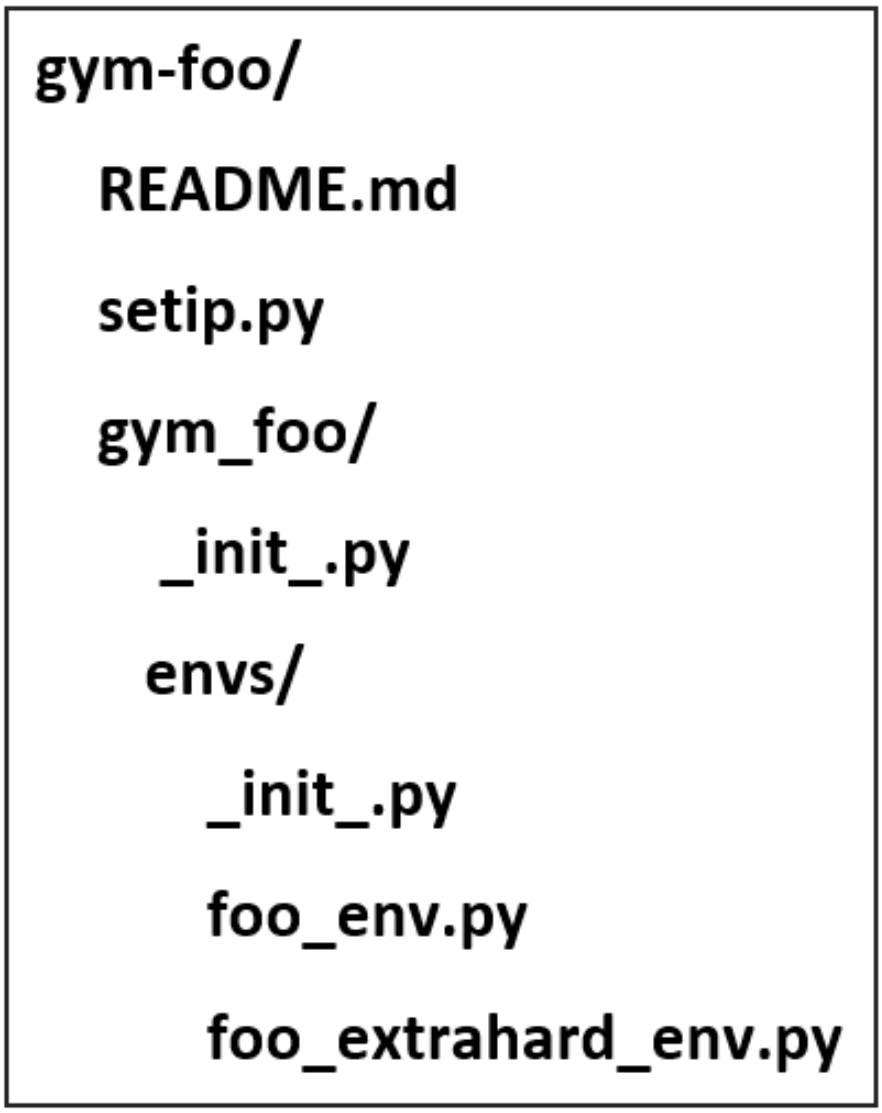

Figure 5. Custom OpenAI Gym implementation.

Figure 6. A visual of the initial state returned by the custom Gym.

4.2. RL Agents

We used three different techniques of reinforcement learning, namely Q-learning-exact, Q-learning-approx and SARSA-approx.

4.2.1. Q-Learning-Exact

In this technique, we know that, at one point in time, it will definitely stop making any improvement in the performance, but the sheer size of the state–action space makes it unsatisfactory for our problem. A back-of-the-envelope calculation shows that the number of state–action pairs is extraordinarily high. If we try to calculate the different state–action pairs, then it would be

$$number\ of\ state-actionpairs = 4^{16*13} * 9 \approx 1.52 * 10^{126} \quad (4)$$

However, practically, the agent comes across only a fraction of these pairs (Figure 7).

Algorithm 1: Exact Q-Learning algorithm

procedure EXACTQ(percept)
inputs: *percept*, a percept indicating the current states′ and reward signal r'
persistent: g, an exploration function
ϵ, the probability of a random move, decreasing from 1 to 0.05 based on iteration
Q, a table of action values indexed by state and action, initially zero
N_{sa}, a table of frequencies for state-action pairs, initially zero
s, a, r, the previous state, action, and reward, initially null

if $Terminal?(s')$ **then**
  $Q[s', None] \leftarrow r'$
if *s is not null* **then**
  increment$N_{sa}[s, a]$
  $W'[i] \leftarrow W'[i] +$
  $\alpha\,(r + \gamma \max_{a'} W \cdot F\,(s', a') - W \cdot F(s, a))\,F[i](s, a)$
$s, a, r \leftarrow$
$s', g\left(\epsilon_i, \arg\max a' Q\,[s', a'] + \frac{10}{Nsa[s^s, a']+1}\right), r'$
return a

Figure 7. The algorithm used for Exact Q Learning.

4.2.2. Q-Learning-Approx

In order to narrow down the number of different state–action pairs, we designed an attributes set, from which, various attributes can be combined to specify a state (Figure 8). The action value can then be expressed in the form of these attributes as follows:

$$Q(s, a) = w_1 f_1(s, a) + ... + w_n f_n(s, a) \quad (5)$$

Algorithm 2: Approximate Q-Learning algorithm

procedure APPROXQ(*percept*)
inputs: *percept*, a percept indicating the current state s' and reward signal r'
persistent: g, an exploration function
ϵ, the probability of a random move, decreasing from 1 to 0.05 based on iteration
F, a set of n feature functions that returns a feature value for a state action pair
W', a set of n weights for each feature function, initially all zero
N_{sa}, a table of frequencies for state-action pairs, initially zero
s, a, r, the previous state, action, and reward, initially null

if $Terminal?(s')$ **then**
  $\max_{a'} W \cdot F(s', a') \leftarrow r'$
if *s is not null* **then**
  increment$N_{sa}[s, a]$
  for $i = 1$ *to* n **do**
  $W'[i] \leftarrow W'[i] +$
  $\alpha\,(r + \gamma \max_{a'} W \cdot F(s', a') - W \cdot F(s, a))\, F[i](s, a)$

s, a, r, W $\leftarrow$
$s', g\left(\epsilon_i, \arg\max a' Q[s', a'] + \frac{10}{Nsa[s', a']+1}\right), r', W'$
return a

Figure 8. The algorithm used for approx Q-learning.

The attributes that we used are described below:

- **Speed:** This attribute is used for determining the distance and the direction, i.e., if the agent is making forward progress or not.
- **Action for Running:** This attribute returns if the action performed is for sprinting or not. Sometimes the agent sprints in order to overcome an obstruction.
- **Action for Jumping:** This attribute tells if the action performed by the agent is jumping. The knowledge of this can be of great help in picking up and overcoming obstacles and enemies in the way.
- **Forward/Backward:** This attribute tells if the action performed involves the pressing of forward or backward on the NES controller.
- **No Movement:** This attribute specifies if the agent is not able to make any movement in the forward direction. If this attribute is true, then the agent is forced to take certain actions.
- **Distances in all directions:** This attribute helps the agent to decide which is the best moment to cross an empty space.
- **Presence of adversary:** The agent might get killed because of an obstruction, or it might fetch rewards by overcoming an obstruction.
- **Opponent Destroyable:** This attribute tells the agent if the upcoming opponent can be destroyed or not, which basically depends on its closeness to the agent.
- **Opponent ahead/behind:** This feature helps the agent to decide the direction of the movement.

4.2.3. SARSA-Approx

Although we thought that approx Q-learning was an appropriate technique, we explored a minor variant. Except for one minor variation, approx SARSA is nearly similar

to approx Q-learning. The approx SARSA agent learns relative to the policy it follows, whereas the approx Q agent learns relative to the greedy policy (Figure 9).

Algorithm 3: Approximate SARSA algorithm

procedure APPROXSARSA(*percept*)
inputs: *percept*, a percept indicating the current state s' and reward signal r'
persistent: g, an exploration function
ϵ, the probability of a random move, decreasing from 1 to 0.05 based on iteration
F, a set of n feature functions that returns a feature value for a state action pair
W', a set of n weights for each feature function, initially all zero
N_{sa}, a table of frequencies for state-action pairs, initially zero
s, a, r, W, the previous state, action, reward, and weights, initially null

$$\mathrm{a}' \leftarrow g\left(\epsilon_i, \arg\max a' Q\left[s', a'\right] + \frac{10}{Nsa(s',a')+1}\right)$$

if $Terminal?(s')$ **then**
    $W \cdot F\left(s', a'\right) \leftarrow r'$
if *s is not null* **then**
    increment$N_{sa}[s, a]$
    for $i = 1$ *to* n **do**
    $W'[i] \leftarrow W'[i] +$
    $\alpha\left(r + \gamma(W \cdot F\left(s', a'\right)) - W \cdot F(s, a)\right) F[i](s, a)$
s, a, r, W $\leftarrow s', a', r', W'$
return a

Figure 9. The algorithm used for approx SARSA.

4.3. Function for Exploration

We used this function in the reinforcement learning algorithms with the objective of maintaining an equilibrium between taking a decision on the basis of policy and taking unexplored actions. In order to merge the action values of the agent and utilize the behaviour, we added some random decision acts in its gameplay. With a probability starting from 1 and decreasing until it reaches 0.05, the agent will consider random decisions on the basis of the following priorities, as shown in Table 3 with forward sprinting being the topmost priority.

Table 3. Distribution of priorities.

Distribution of Priorities	
Backward	1
Backward Jumping	1
Backward Sprinting	1
Backward Jump + Sprint	1
Move Ahead	3
Forward Jumping	3
Forward Sprinting	10
Forward Jump + Sprint	5
Jumping	1

Our derived function for exploration is:

$$g\left(\epsilon_i, argmax_{a'} Q[s', a'] + \frac{k}{N_{sa}[s', a'] + 1}\right) = \begin{cases} argmax_{a'} Q[s', a'] + \frac{k}{N_{sa}[s', a']+1} & \text{with probability} \\ 1 - \epsilon_i a' \in \text{PRIOR}, & \text{with probability } \epsilon_i \end{cases} \quad (6)$$

4.4. Attribute Updation

All of the attributes that we used for approximate algorithms are specified in Section 4.2.2. Now, we will understand how and when we updated the weights associated with each of those attributes. Whenever the agent gains any reward, the approx algorithm updates the importance (weights) associated with each of the attributes that are active at that particular moment. As we know that one state can only lead to a different state sequentially, credit must be given to both of the states, i.e., the present state as well as the state before that led us to this particular state. Thus, to sort this out, we included "eligibility traces" in our approx algorithms. Therefore, the approx Q-learning algorithm was converted to approx Q(λ), and the approx SARSA algorithm was converted to approx SARSA(λ). Therefore, now, the updating of weights associated with the attributes not only takes place on the basis of present state attributes, but the weights are also adjusted iteratively on the basis of the attributes associated with all of the states that our agent came across before the current state. λ was used for determining the magnitude corresponding to the update. Consider $W'[i_t]$ as the default update for the weight of feature i at time t. Then, our equation is:

$$W'[i_t] \leftarrow W'[i_t] \leftarrow \lambda W'[i_{t-1}] \leftarrow \lambda^2 W'[i_{t-2}] \leftarrow .. \leftarrow \lambda^n W'[i_{t-n}] \quad (7)$$

Here, n represents the number of latest timesteps.

5. Experiment

We wanted to analyze how an agent developed by a particular technique, in its nascent form, would perform in the synthetic environment. Therefore, we developed each algorithm—exact Q-learning, approx Q-learning and approx SARSA—from scratch. The agents of these techniques were deployed in our synthetic environment and observed for 100 training iterations. Notwithstanding our computational limitations, it was conclusive that the number of training phases chosen by us was enough for a comparative study of our agents. The agents would converge within 100 iterations. Running the agents for 1000 agents, as carried out, did not obtain any significant improvements in order to declare the performance of one agent better than the other. However, exact Q-learning performed otherwise. With more iterations, it ended up performing better. The model hyperparameters for each agent were selected qualitatively. Below in Table 4, we enlist the main model hyperparameters used, along with their initialization values. We also mention the reasoning behind selecting the specific values. Some more hyperparameters were used to capture information about instances such as scoring points by the collection of items and Mario dying. This was to be added in the reward function.

- α **(rate of learning):**

Table 4. α (rate of learning).

Agent	Value
Exact Q Learning	0.101
Approx Q	0.010
Approx SARSA	0.010

Avoid overlearning and divergence of weights by keeping the values low.

Actions should be chosen while also considering future states shown in Tables 5–7. A high value would be suitable, owing to the causal dependency between state sequences and 30 FPS in game runs.

- γ **(factor for discount):**

Table 5. γ (factor for discount).

Agent	Value
Exact Q -earning	0.950
Approx Q	0.950
Approx SARSA	0.950

- ϵ**-min (probability of minimum random moves):**

Table 6. ϵ-min (probability of minimum random moves).

Agent	Value
Exact Q-Learning	0.051
Approx Q	0.051
Approx SARSA	0.051

It is important to take a random action irrespective of the frequency of the exploration of a particular state. This also ensures that the Mario agent is not stuck at some point.

- λ **(updated attributes decay):**

Table 7. λ (attributes decay).

Agent	Value
Exact Q-Learning	N/A
Approx Q	0.800
Approx SARSA	0.800

We need to assign some credit to recent earlier states, owing to the dependency between game states. Hence, there is a high value for this parameter.

6. Results

We measured over 100 iterations of the mean distance and maximum distance reached by each agent. Following each iteration, progress through the level was reset. However, after the Mario agent reached a checkpoint, the next iterations, instead of a reset, started at the checkpoint. It should be noted that our agents reliably reached the checkpoint midway. We can see that, among all agents, only the heuristic agent was able to reach the end of the level (Figure 3).

After evaluating the results initially, our only concern was if the Q-learning agents had actually converged within the selected number of iterations for training into a good agent policy. We wanted to test for a much larger number of iterations for training, but, in terms of the computational resources available to us, it was the best we could carry out. Thus, we set out to quantify each agent's learning for evaluating the convergence (Figure 10). We can look at the number of new states acquired during 200 training iterations by the exact Q-learning agent in Figure 6. We find an approximately linear increase in this number for approx. 200 iterations. We understood that 100 iterations was indeed not enough for the exact Q-learning agent (Figure 11).

The weights for each feature usually converged for the approx Q-learning agent in under 100 iterations. This could possibly be due to Mario dying in between relatively frequently. It depicts notable fluctuations in the initial cycle of learning. A long stabilization then follows. From these findings, we concluded that additional training iterations for the approx Q-learning agent would not likely have provided a substantial improvement in the performance (Figure 12).

We ran the approx Q-learning agent for 1000 consecutive training iterations, as further proof of this theory. However, there was no significant enhancement in the performance. The weights converged, as predicted, after less than 100 iterations. Nine times out of ten, the gap after the checkpoint would still not be cleared by the agent.

∗*The approx SARSA learning agent's graph was almost identical to the approx Q-learning agent's plot above as shown in Figure 9.*

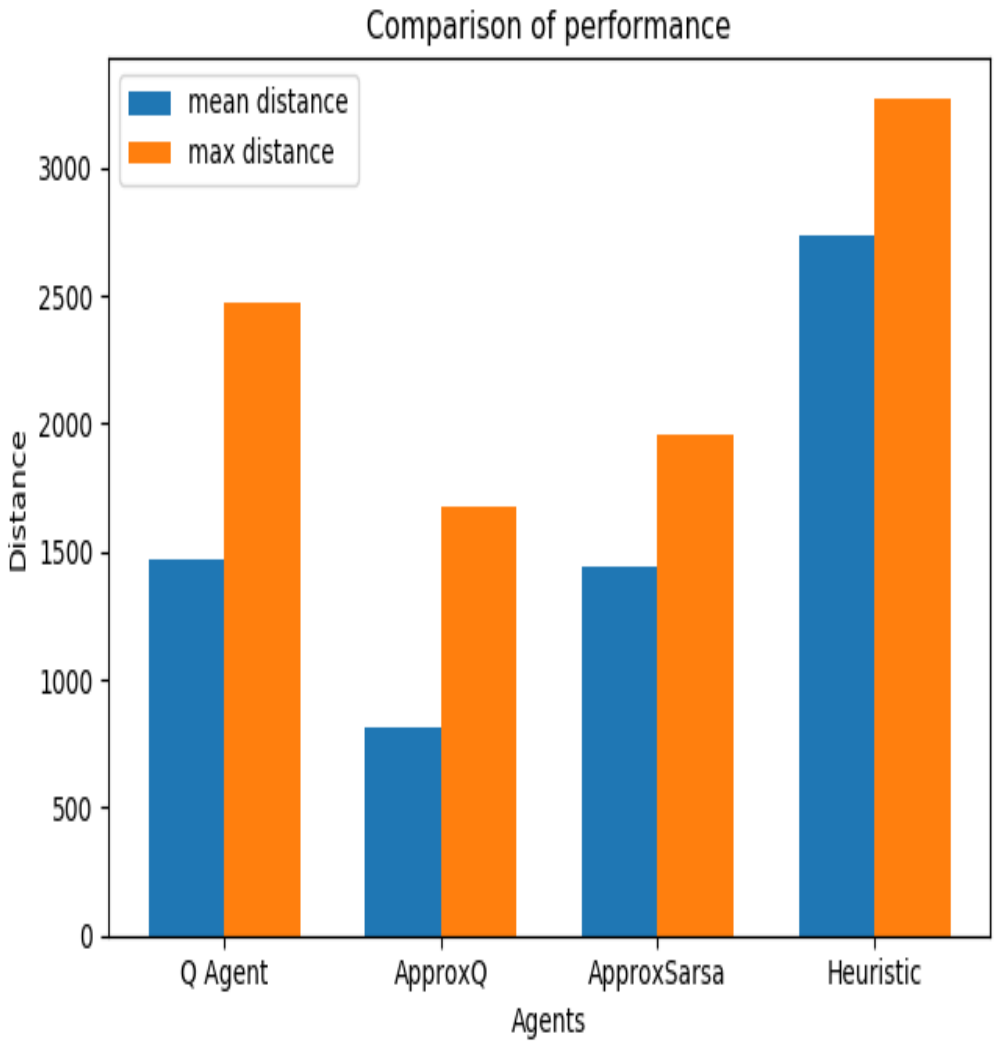

Figure 10. Performance comparison plot—depicts mean distance (blue colour) and maximum distance (orange colour) after 100 iterations.

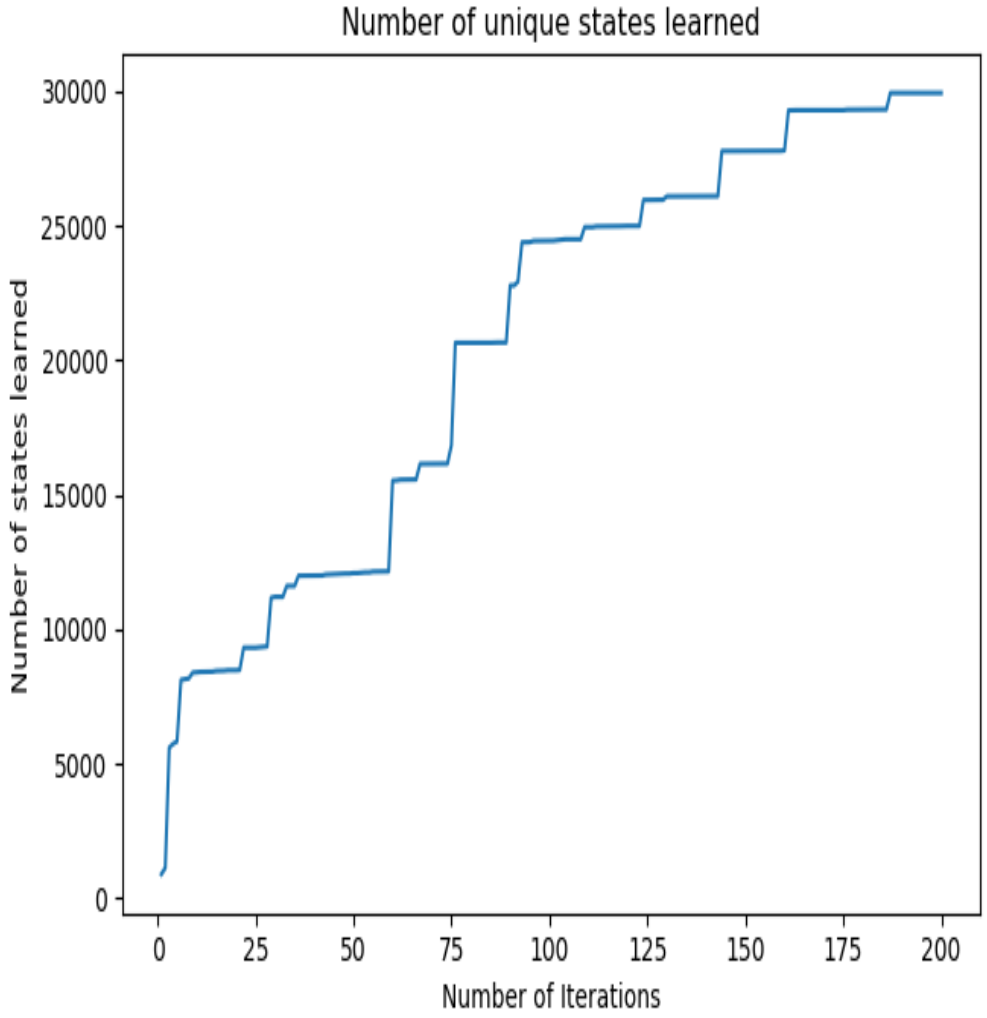

Figure 11. Exact Q-learning agent: Number of distinct states learnt vs. number of iterations. Model converges around 200 iterations.

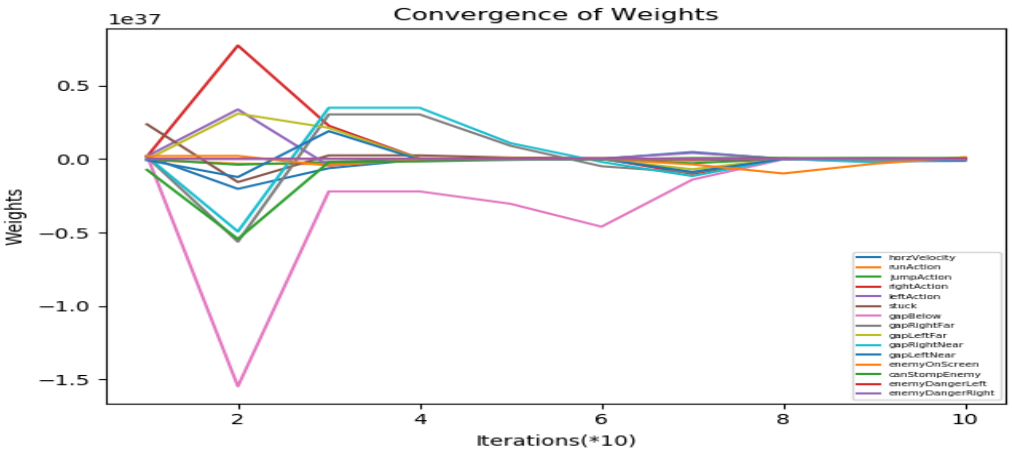

Figure 12. Approx Q-learning agent: Plot for convergence of weight. Notable fluctuations before convergence at around 80 iterations.

7. Future Work

A major drawback with approx Q-learning is the assumption of linearity wherein a linear weight mix is presumed for the values of Q. Correlating with the negative and

positive incentives while feature designing, relevant state facets were picked up. However, even after converging within 100 iterations, the approx Q-learning agent was not able to develop a better strategy, which is worth exploring. Whenever Mario took an action that helped him to progress, the right action function was activated. This almost always yielded a positive reward. However, it is not always correct to give a positive reward, and is in fact contextual. If Mario were to collide with an enemy, that should be a negative reward and not positive. However, if Mario would instead stomp the enemy, that is a positive reward. This, in general, is dependent on the state of the game. Therefore, optimum weights should be assigned to the functions. Making progress in the game often requires complex action sequences, such as start running to clear tall pipes, jump for nearly 25 sequential states along with right action and so on. Designing the reward mechanisms that could promote such action sequences is challenging. One way would be to provide higher rewards for clearing pipes. This might lead the agent to fiddle and explore around various actions sequences. However, the positioning of various obstacles should be considered within the state when carrying this out.

8. Conclusions

Of all of the agents, the heuristic agent slightly outperformed the rest. The other agents did not even reach the end of the level. Q-learning agents were unable to clear large gaps, especially the one after mid way checkpoint, with distances of around 1400. There is much variance that feature-based learning, such as approx Q-learning, can bring in. This can be an explanation as to why it was outperformed by exact Q-learning. In addition, since exact Q-learning does not use features, given enough time, it would perform even better. However, if a more consistent and comprehensive set of features can be designed, the exact Q-learning agent will be outperformed by the approx Q-learning agent by a notable margin. Based on our knowledge, ours is the first effort made to solve an augmented Super Mario environment. We hope that this proves to be helpful for others who approach this task in the future.

Author Contributions: Conceptualization, K.K. (Ketan Kotecha), J.S. and S.P.; conceptualization, S.M. and S.P.; methodology, R.S. and B.S.; software, K.K. (Kshitiz Kalra) and M.B.; validation, R.S., B.S. and M.B.; formal analysis, S.M.; investigation, S.M.; resources, B.S.; data curation, K.K. (Kshitiz Kalra); writing—original draft preparation, B.S.; writing—review and editing, S.M.; visualization, R.S.; supervision, S.M.; project administration, S.M.; funding acquisition, K.K. (Ketan Kotecha) and J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Research Support Fund (RSF) of Symbiosis International (Deemed University), Pune, India.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hu, J.; Cheng, Y.; Gan, Z.; Liu, J.; Gao, J.; Neubig, G. What makes a good story? designing composite rewards for visual storytelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*; Association for the Advancement of Artificial Intelligence: Menlo Park, CA, USA, 2020; Volume 34, No. 05.
2. Huang, V.; Ley, T.; Vlachou-Konchylaki, M.; Hu, W. Enhanced Experience Replay Generation for Efficient Reinforcement Learning. *arXiv* **2017**, arXiv:1705.08245.
3. Doan, T.; Mazoure, B.; Lyle, C. GAN Q-learning. *arXiv* **2018**, arXiv:1805.04874.
4. Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; Chen, X. Improved Techniques for Training GANs. *arXiv* **2016**, arXiv:1606.03498.
5. Saldanha, J.; Chakraborty, S.; Patil, S.; Kotecha, K.; Kumar, S.; Nayyar, A. Data augmentation using Variational Autoencoders for improvement of respiratory disease classification. *PLoS ONE* **2022**, *17*, e0266467. [CrossRef] [PubMed]

6. Jain, V.; Nankar, O.; Jerrish, D.J.; Gite, S.; Patil, S.; Kotecha, K. A Novel AI-Based System for Detection and Severity Prediction of Dementia Using MRI. *IEEE Access* **2022**, *9*, 154324–154346. [CrossRef]
7. Fu, Q.; Sun, E.; Meng, K.; Li, M.; Zhang, Y. Deep Q-Learning for Routing Schemes in SDN-Based Data Center Networks. *IEEE Access* **2020**, *8*, 103491–103499. [CrossRef]
8. Mahajan, S.; Harikrishnan, R.; Kotecha, K. Adaptive Routing in Wireless Mesh Networks Using Hybrid Reinforcement Learning Algorithm. *IEEE Access* **2022**, *10*, 107961–107979. [CrossRef]
9. Alavizadeh, Hooman, Hootan Alavizadeh, and Julian Jang-Jaccard. Deep Q-Learning based Reinforcement Learning Approach for Network Intrusion Detection. *Computers* **2022**, *11*, 41. [CrossRef]
10. Kaviani, S.; Ryu, B.; Ahmed, E.; Larson, K.A.; Le A.; Yahja, A.; Kim, J.H. Robust and Scalable Routing with Multi-Agent Deep Reinforcement Learning for MANETs. *arXiv* **2020**, arXiv:2101.03273.
11. You, X.; Li, X.; Xu, Y.; Feng, H.; Zhao, J.; Yan, H. Toward Packet Routing With Fully Distributed Multiagent Deep Reinforcement Learning. *IEEE Trans. Syst. Man Cybern. Syst.* **2022**, *52*, 855–868. [CrossRef]
12. Lingam, G.; Rout, R.R.; Somayajulu, D.V. Adaptive deep Q-learning model for detecting social bots and influential users in online social networks. *Appl. Intell.* **2019**, *49*, 3947–3964. [CrossRef]
13. Wu, R.; Zhou, C.; Chao, F.; Yang, L.; Lin, C.M.; Shang, C. Integration of an Actor-Critic Model and Generative Adversarial Networks for a Chinese Calligraphy Robot. *Neurocomputing* **2016**, *388*, 12–23. [CrossRef]
14. Caminero, G.; Lopez-Martin, M.; Carro, B. Adversarial environment reinforcement learning algorithm for intrusion detection. *Comput. Netw.* **2019**, *159*, 96–109. [CrossRef]
15. Finn, C.; Christiano, P.; Abbeel, P.; Levine, S. A Connection Between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models. *arXiv* **2016**, arXiv:1611.03852.
16. Fu, J.; Luo, K.; Levine, S. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. *arXiv* **2017**, arXiv:1710.11248.
17. Hausman, K.; Chebotar, Y.; Schaal, S.; Sukhatme, G.; Lim, J.J. Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets. *arXiv* **2017**, arXiv:1705.10479.
18. Taleb Zadeh Kasgari, A.; Saad, W.; Mozaffari, M.; Poor, H.V. Experienced Deep Reinforcement Learning with Generative Adversarial Networks (GANs) for Model-Free Ultra Reliable Low Latency Communication. *arXiv* **2019**, arXiv:1911.03264.
19. Uřičář, M.; Křížek, P.; Hurych, D.; Sobh, I.; Yogamani, S.; Denny, P. Yes, we GAN: Applying Adversarial Techniques for Autonomous Driving. *Electron. Imaging* **2019**, *2019*, 48. [CrossRef]
20. Summerville, A.J.; Snodgrass, S.; Mateas, M.; Ontanón, S. The VGLC: The Video Game Level Corpus. *arXiv* **2016**, arXiv:1606.07487.