robotics

MDPI

Article

Integrating the Generative Adversarial Network for Decision Making in Reinforcement Learning for Industrial Robot Agents

Neelabh Paul [1], Vaibhav Tasgaonkar [1], Rahee Walambe [1,2,*] and Ketan Kotecha [1,2,*]

[1] Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune 412115, India
[2] Symbiosis Centre of Applied Artificial Intelligence, Symbiosis International (Deemed University), Pune 412115, India
* Correspondence: rahee.walambe@sitpune.edu.in (R.W.); director@sitpune.edu.in (K.K.)

Abstract: Many robotics systems carrying certain payloads are employed in manufacturing industries for pick and place tasks. The system experiences inefficiency if more or less weight is introduced. If a different payload is introduced (either due to a change in the load or a change in the parameters of the robot system), the robot must be re-trained with the new weight/parameters and the new network must be trained. Parameters such as the robot weight, length of limbs, or new payload may vary for an agent depending on the circumstance. Parameter changes pose a problem to the agent in achieving the same goal it is expected to achieve with the original parameters. Hence, it becomes mandatory to re-train the agent with the new parameters in order for it to achieve its goal. This research proposes a novel framework for the adaption of varying conditions on a robot agent in a given simulated environment without any retraining. Utilizing the properties of Generative Adversarial Network (GAN), the agent is able to train only once with reinforcement learning and by tweaking the noise vector of the generator in the GAN network, the agent can adapt to new conditions accordingly and demonstrate similar performance as if it were trained with the new physical attributes using reinforcement learning. A simple CartPole environment is considered for the experimentation, and it is shown that with the propose approached the agent remains stable for more iterations. The approach can be extended to the real world in the future.

Keywords: reinforcement learning; GANs; Q-tables; noise vector; payload; industrial robots

Citation: Paul, N.; Tasgaonkar, V.; Walambe, R.; Kotecha, K. Integrating the Generative Adversarial Network for Decision Making in Reinforcement Learning for Industrial Robot Agents. *Robotics* **2022**, *11*, 150. https://doi.org/10.3390/robotics11060150

Academic Editors: Wilfried Lepuschitz, Markus Vincze, Thomas Meurer and Munir Merdan

Received: 14 October 2022
Accepted: 7 December 2022
Published: 9 December 2022

1. Introduction

There has been an immense need for robotic assistance in the industrial sector as witnessed in the last decade with the advent of various autonomous applications and their complexity. This is also observed in warehouse-based industries [1]. This is a recent development due to the increase in e-commerce and online shopping [2], including the premium services within online shopping such as one-day shipping [3]. Companies such as Amazon have already begun to introduce robots in their package-handling facilities [4]. Robots carry packages within the storage facilities and help connect the packages to their destinations. Another instance of robot intervention can be seen in Boston Dynamics with their robot dog [5]. This robot is capable of carrying payloads from one place to another and can autonomously prevent obstacles in its way. However, manual control is still required to maneuver the robot to its destination, making it somewhat human-dependent. Hence, there is a need to develop techniques that help the robot maneuver autonomously. One such technique is Reinforcement Learning (RL) training [6], RL is a training paradigm in artificial intelligence that comprises an environment and an agent. The goal is to train the agent through a reward-based trial-and-error system. Every positive step taken in the environment is rewarded and every negative step is penalized to the agent. The agent in this context is an industrial robot that is trying to carry a payload in its environment. Due to the fact that even though robots nowadays have physical capabilities to carry a

payload, they lack the intellectual prowess to handle their own motors in accordance with the wide range of loads that they carry [7]. Whenever the physical attributes of a reinforcement learning agent are modified, the agent's efficiency falls. This research tries to mitigate that by increasing robot adaptability while requiring one-time training. In this work Generative Adversarial Networks (GANs) [8] based approach is proposed to overcome this challenge and further enhance the capabilities of RL. The main goal here is to generate autonomous behaviors for the robot agent which can enable it to intellectually carry a wider range of loads. In the last few years, GAN and RL are integrated for a variety of problems. In [9], authors have proposed a methodology to employ RL for improving the GAN architecture search and also in reducing resource utilization. Here they used GAN as an MDP for smoother architecture sampling, providing a more effective RL-based search algorithm. This process reduced variance and noise from generated architectures while improving the computational speed. In [10], RL was used to determine necessary inputs to a GAN. The goal of the research was to generate complete point clouds from incomplete or noisy point cloud inputs. A pre-trained discriminator of GAN is used to decide the winner between the decoded output of the GAN and the output of the AE. The final decision of the completed output shape has a high resemblance to the original point cloud. On the other hand, utilizing GAN to improve or facilitate RL performance is also possible. In [11], generative models are used to generate synthetic samples to improve the exploration efficiency of the deep RL models. In [12] RL-CycleGAN is employed to demonstrate a solution to address the problem of simulations not being realistic enough to train the agent for real-world applications. Since building a simulation as close as possible to reality requires immense domain knowledge. They proposed the idea of using generative models to translate simulated visuals into realistic ones. Upon testing, RL-CycleGAN proved to be substantially better than conventional RL training for sim-to-real transfer. In [13], authors have demonstrated the effectiveness of pre-training the RL framework with a mix of synthetic data and real data from a generative model. Using a process such as this gave the RL framework a wider range of data to learn from. There is a resource allocation problem in OFDMA systems, and that is reducing power under the constraints of reliability, latency, and data rate. This problem was better resolved with the new pre-trained RL framework that helps determine data rates. In [14], human demonstrations to train reinforcement learning algorithms were emphasized. Instead of continuously providing human input to guide the agent, GANs were used. Here, a GAN network generated and provided the necessary guidance for the agent training alongside the reinforcement learning algorithms. This allowed the agent to depend much less on actual human inputs while keeping performance the same. Throughout the literature, it can be observed that generative models and reinforcement learning frameworks can benefit from each other in a number of ways. In this work, we have leveraged the power of GAN to diversify and improve the capabilities of an RL agent. Firstly, an RL agent is implemented and Q-tables are generated using the standard Q-learning algorithm [15] for a specific payload and environment condition for an agent. Further, these Q-tables are employed for training the GAN model. The input noise vector [16] to the Generator network is modified to generate new Q-tables corresponding to different payloads or environmental conditions of the agent. These tables can then be used to run the agent with the newer load. The primary advantage of this approach is that with a few rounds of trial and error with tweaking the noise vector, an optimal Q-table can be obtained without the need for any re-training. The conventional way of making a robot adapt to physical modifications is to have it re-train with reinforcement learning in its new form. This can be time and resource-consuming. The proposed approach is useful in saving tremendous computation costs as well as accelerating the process even though the payload is modified.

Contribution:

In summary, the contributions of this work are:

(a) Development and demonstration of an approach to make the robot agent adaptable to the increased payload via the use of GANs.

(b) Proposed an approach for mitigating the need for agent re-training with changing payload leading to saving of time and resources.

The paper is organized into the following sections: Section 2 discusses the problem formulation and assumptions. In Section 3, the method is discussed with a specific focus on the GANs and the Q-learning RL algorithm along with the environment chosen for the demonstration of this approach with the GANs training is discussed in detail. In Section 4, experimental analysis and results are presented followed by Section 5 of discussion that needs to be had in regards to this concept with a conclusion and discussion of future scope in Section 6.

2. Problem Formulation and Assumptions

It is important to address the incapability of a trained agent to successfully complete a task when it is put under different loads. Conventionally, the agent has to be re-trained with the newer load in order for it to be able to complete the same task successfully. In this work, an RL agent is combined with the GANs-based approach to avoid this retraining. Figure 1 shows this method via a simple schematic.

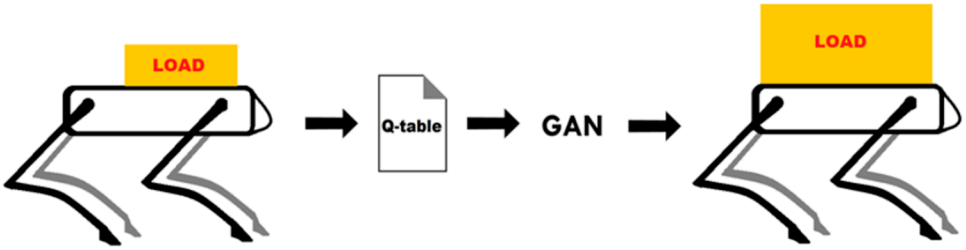

Figure 1. Demonstration of agent's adaptability to heavier weight.

Reinforcement learning is a framework for learning how to interact with the environment from experience (Refer Figure 2). This is a branch of artificial intelligence that has found applications in various domains, most notably in robot training. As the name suggests, this method tries to "reinforce" a particular behavior in the agent for it to perform in a certain way. A robot interacts with the world by measuring its state in that environment and then taking an appropriate action. Every such interaction yields a reward that the robot receives if the action taken by it was in the right direction towards accomplishing the goal. After taking any action, the robot enters a new state in the environment and the process repeats itself till the goal is accomplished or the robot fails. Goal accomplishment gives the robot a significant reward for all the actions it took so far to reach its destination which acts as a positive reinforcement. Contrary to that, failure yields no rewards and hence it acts as negative reinforcement. In this process, the robot builds a policy from all the positive experiences. The policy is used by the robot to maximize its chance of obtaining a future reward. In order to design a useful policy, it is required to understand what is the value of being in a certain state given that policy. So, after choosing a policy one can start to learn what is the value of each state based on what is the expected reward the robot will obtain in the future if the robot starts at that state and enacts that policy. Over time, the robot refines the value function and develops a better idea of what matters in the environment. Reinforcement learning has been implemented for robotics tasks such as visual control of robotic manipulators [17], obstacle avoidance [18], etc. In recent years, it has also been employed for various critical tasks such as the efficient scheduling of limited satellite-based radio resources to ensure enhanced transmission efficiency and meet the requested traffic with low complexity [19].

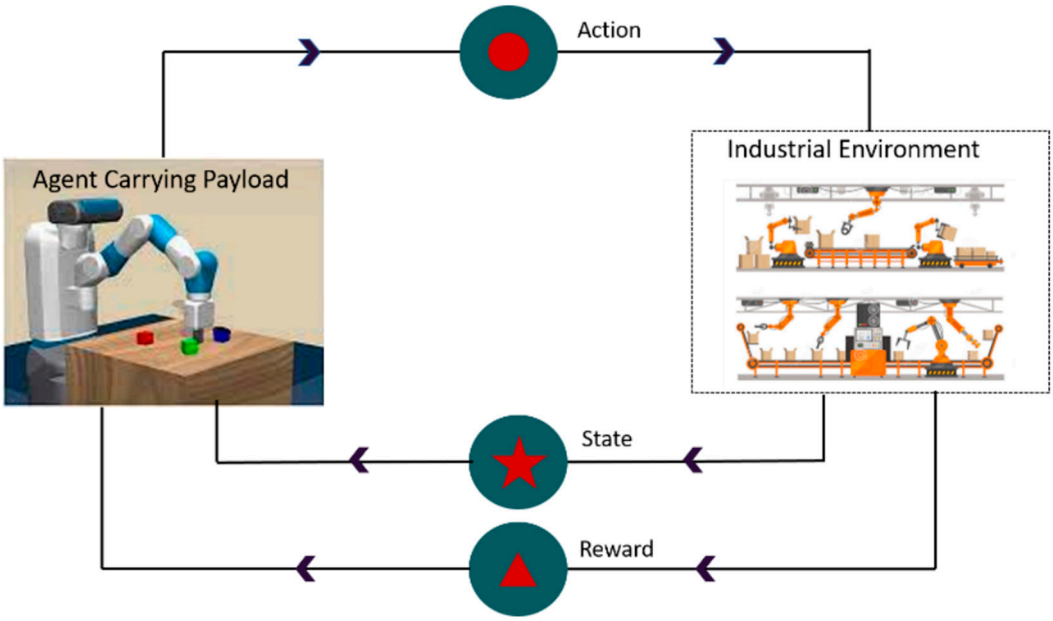

Figure 2. Reinforcement Learning.

Instead of just learning the policy and the value functions separately, in Q-learning, the robot can learn both of them at the same time. The Q function is not just a function of the state but is also a function of action. It represents the quality of being in a particular state and taking a particular action. The value update takes place by adding the old Q value to the product of the learning rate and probable maximum Q value in the future considering a particular action. As the robot goes through the learning process and updates the Q function, a table is populated with pairs of states and the corresponding actions yielding a high value. This table is called the Q-table and this process is called tabular Q-learning. The robot refers to the Q-table after training is completed and it is time to perform in a test scenario. This training methodology has a major drawback in that these tables are very environment-specific and robot-agent-specific. Hence, essentially, the robot populates the Q-table with the most optimal state-action pairs that it has learned from moving its own body in its own environment. As such, it cannot be generalized for every other similar functioning robot with different physical measurements. For example, consider a four-legged robot that has a load-carrying capability of a maximum of 10 kg. The robot is trained with Q-learning. The training can only be performed with a set physical dynamic. Either the robot trains with no load or with maximum load, as the training can only be performed with one set of physical attributes. If it is desired to make the robot efficient in carrying a wide range of weighted loads from 0 kg to 10 kg, multiple sessions of training will be required to achieve optimal results. To mitigate this limitation of multiple training sessions corresponding to the different payloads, in this research, we leverage the power of GANs.

Although GAN architecture is presented earlier, this research work tries to rather introduce the utilization of GAN network with Reinforcement Learning. Thus, producing improved results in simulations (as of now) of industrial robots. The novelty that this research work provides is the reduction in the training time of an RL.

3. Methodology

Each Q-table consists of features of a policy in a given environment and agent. Q-tables can be thought of as a collection of features representing a policy similar to how pixels in an image represent the features of an image. Conventionally Convolutional Neural Network (CNN) [20] are employed to learn the features of an image. In this work, instead of images, Q-tables are fed to CNN networks. CNN learns to recognize the patterns in the feature

space of the Q-tables and later on it is successfully able to classify and even generate new Q-tables as used in GANs.

GANs consist of a pair of neural networks that compete with each other. One is called the generator and the other is called the discriminator. The generator constantly tries to mimic a set output using something known as the noise vector. The discriminator's job is to crosscheck the generator output with what is real and the generator's job is to fool the discriminator as shown in Figure 3. As the generator's training progresses, it becomes better and better at producing real outputs as shown in Figure 4. As the networks start converging, the generator starts generating more and more realistic output which is highly indistinguishable from the actual real input. As such, the discriminator is no longer able to classify if the generated output is fake or not. Once this level of performance is achieved, the generator is disconnected from the GAN and is employed separately. There are several variants of GANs which can be used for various tasks. This paper employs Deep Convolutional GAN [21] (DCGAN) to generate new q-tables. DCGAN is one of the network designs for GAN. That comprises convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the down-sampling and the up-sampling.

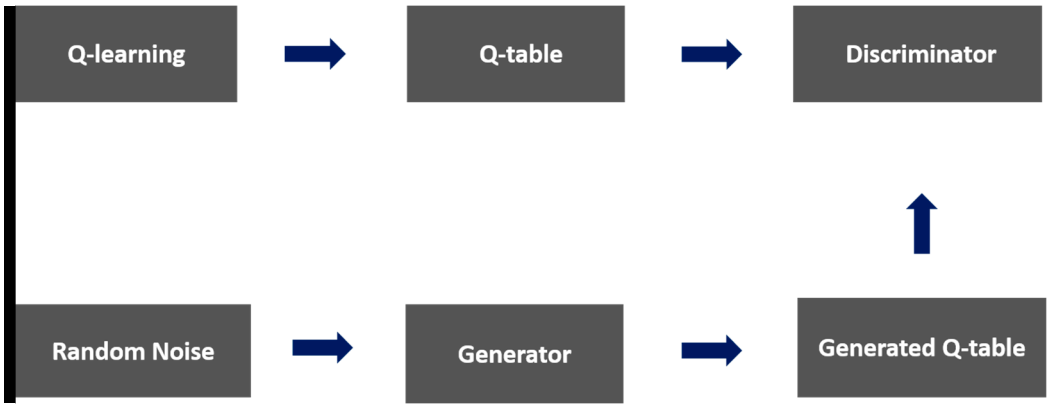

Figure 3. Network Training Process.

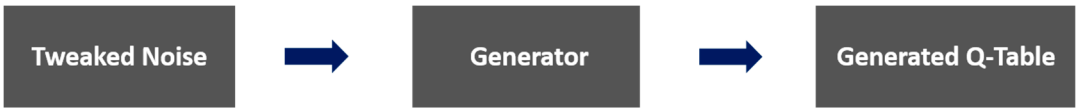

Figure 4. Generator Acting as Agent in the Environment.

In this research, first, an agent is trained in an environment using the Q-learning algorithm. Then after the training, a Q-table is generated. This table is passed to a DCGAN network and the generator is trained to produce similar tables. Since this table acts as a guide and calculates states and actions such that it can provide the maximum expected future rewards for action at every given state in an environment, they essentially carry information on how the agent should traverse through the environment successfully. Hence, the features of a Q-table can be learned, and even have a generative model produce similar tables for varying conditions. After the DCGAN has completed training, the generator can be disconnected and used exclusively. For the generator training, a noise vector is fed in as an input. The generator performs computations on the noise vector and produces output accordingly. Therefore, a noise vector can have a very significant influence on the generator output. By leveraging this feature of the generator multiple Q-tables with different capabilities to instruct an agent can be generated. The complete system pipeline is shown in Figure 5.

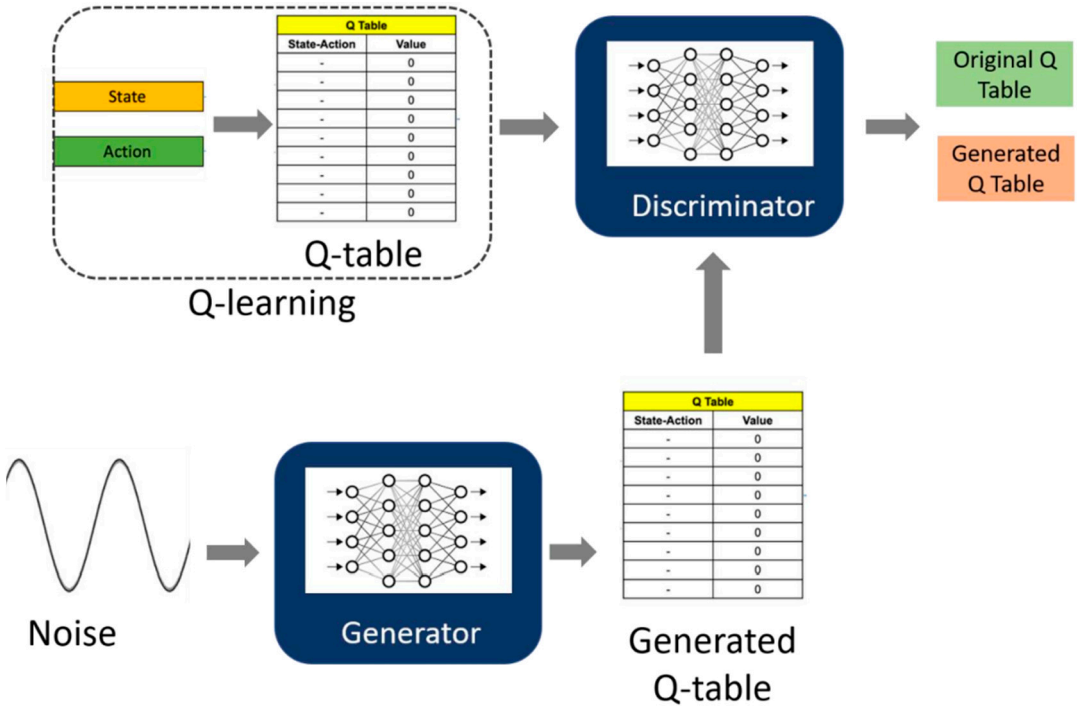

Figure 5. Complete Pipeline of the Proposed Method.

For demonstrating this approach, a simple CartpPole-v1 [22] environment is considered (shown in Figure 6).

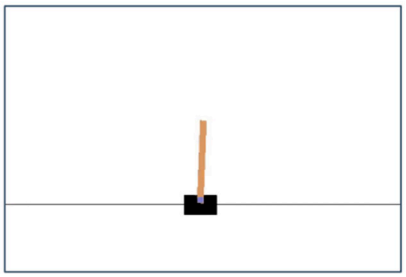

Figure 6. OpenAI Gym Cartpole Environment.

The first agent is trained in the OpenAI CartPole-v1 environment shown in Figure 6. Cartpole-v1 provides a 2D environment with a structure where a pole is attached to the cart and the cart is free to slide over a frictionless surface. The goal of this pole-carrying cart is to balance the pole in an upright position without letting it drop on either side. By sliding the cart left or right, the cart pole is balanced. Cartpole is controlled by applying force +1 and −1 to the cart.

State space includes cart position, cart velocity, pole angle, and pole velocity at the tip. These state values are continuous in nature, and later have to be discretized. The reason behind the conversion from continuous to discrete value is that Q-learning trains only on discrete state space and action space. Since action space is already discrete in nature having only right and left moves, no conversion is required. The agent is trained using the Q-learning approach for a specific set of parameters. These variations in parameters are illustrative of varying payloads to an industrial robot. This original q-table is passed through the GAN. The network learns the feature representation given by the q-table and tries to replicate them by re-shaping noise vectors. The main idea behind using DCGAN for this application is the use of convolutional layers. These layers are conventionally used to map features for images and video streams. Since the q-table represents the feature space of an agent, the CNN-based DCGAN can be employed for this specific task. The agent

learns its environment and learns the appropriate actions corresponding to the highest value. This information is stored in the q-table which can be imagined as a brain that can understand and envisage the result even before taking an action. The generator tries to produce q-tables as real as the original table. A noise vector of the exact same dimension as the original q-table is passed through 3 layers of transposed convolutional layers, each followed by a Batch normalization [23] and a ReLU [24] layer and then by the final 4th transpose convolution layer followed by a Sigmoid function. All the layers are configured to have the "same" padding such that the output dimension of each layer is unchanged. Output from here is passed to the Discriminator along with a normalized version of the original q-table. The discriminator learns the features and updates their weights along with the generator's weights. Discriminator uses two convolution layers each followed by a batch normalization layer and a leaky ReLU [25] layer, then it is another Convolution layer for the final 3rd layer with a linear and sigmoid layer at the end. Discriminator also has been configured to have "same" padding. The output is a probability of the realness of the input.

During GAN training, two inputs are required for the two components, the Generator and the Discriminator. Inputs to the Discriminator are always q-tables. It takes in two q-tables, one from the generator and one which is the real q-table i.e., from the first agent's q-learning. In a supervised learning fashion, the discriminator compares the two tables and essentially tries to classify which one is generated and which one is real.

Input to the generator is a noise vector. A noise vector is a collection of random values in the form of a matrix. The matrix obtained from the numbers is of the same dimensions as the original q-table. The nature of these randomly generated values falls on the Gaussian curve [26]. This also provides control over what type of output is generated by the generator. Figures 7 and 8 show the process for generator and discriminator training respectively.

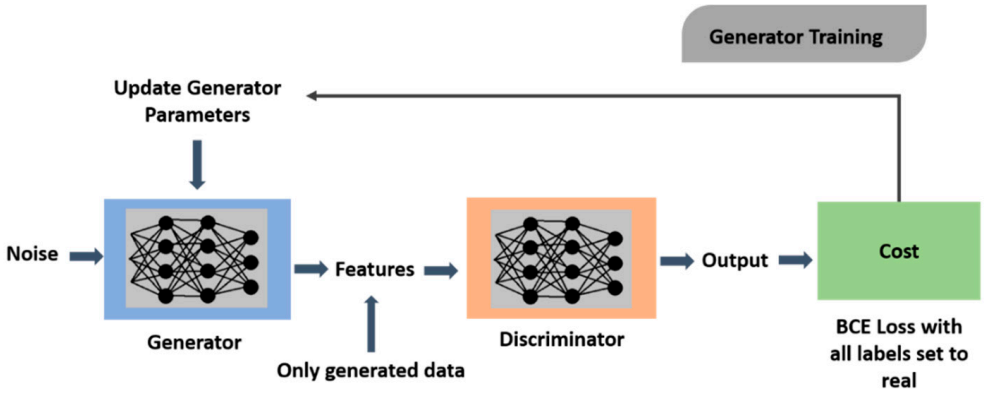

Figure 7. Generator Training.

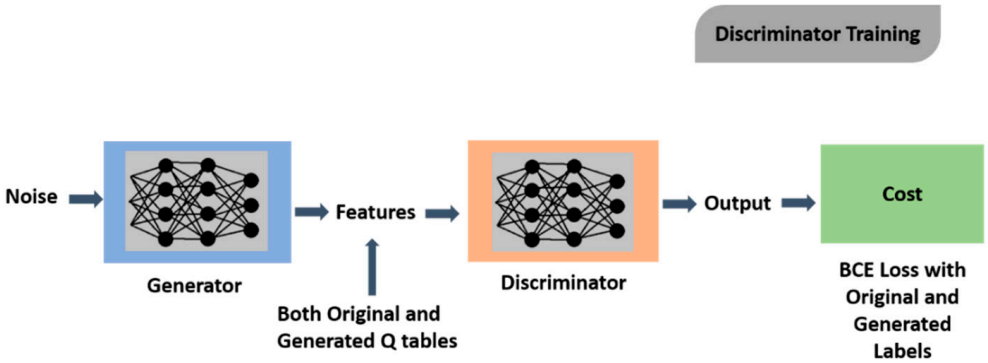

Figure 8. Discriminator Training.

As shown in Figure 9, the Gaussian curve has two parameters viz, standard deviation and mean. By manipulating these values, the output can be controlled. Taking control of the output is important here because that allows the robot to adapt to the new payload. Although in its current state the exact Gaussian parameters corresponding to a certain payload are ambiguous. The generator is attached to the environment to act as an agent. This research work uses the trial-and-error methodology to determine the appropriate noise vector for a given payload on the robot agent.

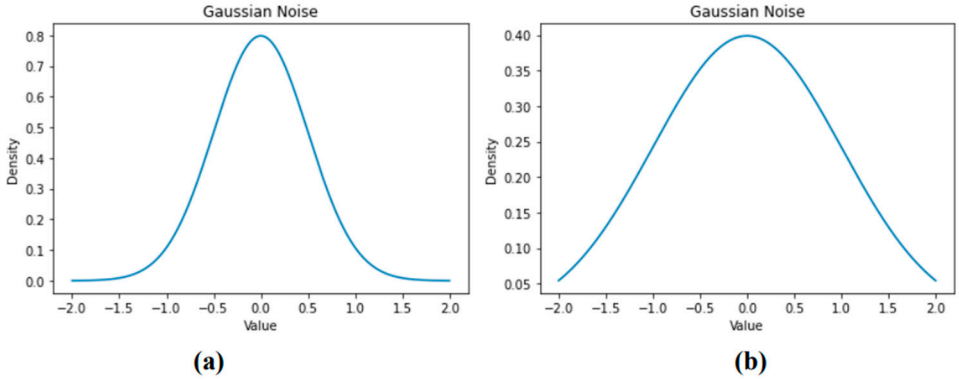

Figure 9. Gaussian noise with (**a**) mean 0.0 and standard deviation 0.5 (**b**) mean 0.0 and standard deviation 1.0.

The CartPole-V1 environment is chosen for the experimentation and demonstration of the proposed approach. This environment's states consist of four parameters: cart position, cart velocity, pole angle, and pole angular velocity. The action space consists of one discrete parameter, (0,1) i.e., left and right movement. That makes the q-table of 4 + 1 = 5 dimensions. Yet using all the state parameters to train the GAN network would require a lot of GPU memory. Hence, after testing the agent with an off-policy method it was found that cart position and cart velocity played a major role in obtaining the correct Q values. Therefore, taking only the two of the cart's state values the dimension of the q-table becomes three dimensional.

Experiments were performed on a system equipped with AMD Ryzen 9 4900HS CPU and Nvidia RTX 2060 max-q edition GPU. Reinforcement learning was performed on the CPU and GAN convolutions were performed on the GPU using CUDA cores. It is evident here that using the original q-table for updated parameters does not produce usable output. The entire movement of the agent changes and it struggles to perform with similar efficiency as it did before the update. Although not perfect, the GAN network does assist the agent in maintaining efficiency by providing generated q-tables. Currently, at this stage, the generated q-tables are picked by trial and error to check for the most efficient table. A looping mechanism is used to randomly pick mean and standard deviation values for the noise vector. For each loop cycle, a new noise vector is obtained from the mean and standard deviation values. The generator takes in these noise vectors and tries to act as an agent in the environment. The idea here is to reject the poorly performing tables for the high-performing ones and save the most efficient q-table for that particular set of parameters for future use when it may be desired to use it again.

4. Results

Experiments were performed on a system equipped with AMD Ryzen 9 4900HS CPU and Nvidia RTX 2060 max-q edition GPU. Reinforcement learning was performed on the CPU and GAN convolutions were performed on the GPU using CUDA cores. It is evident here that using the original q-table for updated parameters does not produce usable output. The entire movement of the agent changes and it struggles to perform with similar efficiency as it did before the update. Although not perfect, the GAN network does assist the agent in maintaining efficiency by providing generated q-tables. Currently, at this

stage, the generated q-tables are picked by trial and error to check for the most efficient table. A looping mechanism is used to randomly pick mean and standard deviation values for the noise vector. In each loop cycle, a new noise vector is obtained from the mean and standard deviation values. The generator takes in these noise vectors and tries to act as an agent in the environment. The idea here is to reject the poorly performing tables for the high-performing ones and save the most efficient q-table for that particular set of parameters for future use when it may be desired to use it again.

The graphs in Figures 10 and 11 are extensions of Table 1 and show the generator behavior when fed with a particular noise vector.

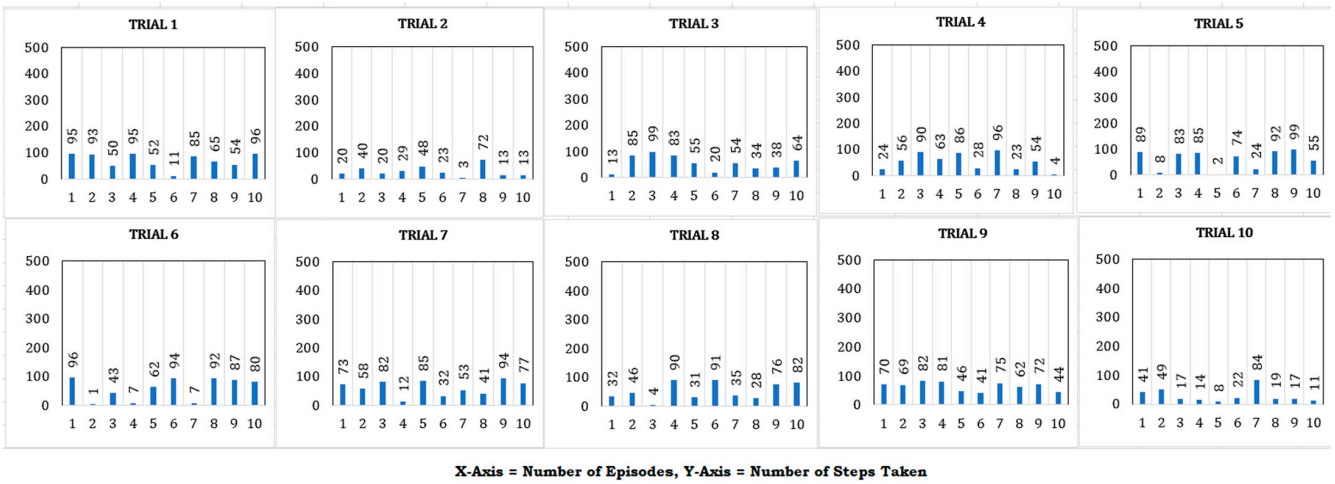

Figure 10. Before GAN, agent performance throughout the episodes average around 52 steps per episode, and the maximum of 500 steps per episode is never reached.

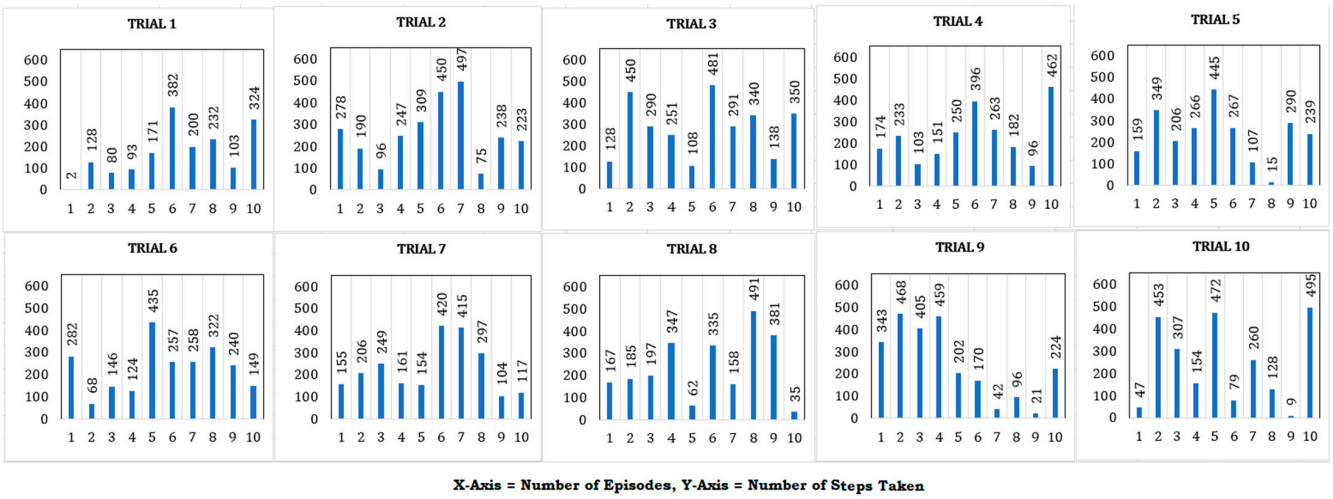

Figure 11. After GAN, agent performance throughout the episodes averages around 297 steps per episode, and a count of 497 steps per episode is also achieved which is very close to the maximum of 500 steps.

Table 1. Updated testing parameters (Mass of Cart and Mass of Pole) for the CartPole and the corresponding results of the RL agents with and without GANs.

	Training Parameters	**Testing Parameters**	**Before GAN**	**After GAN**
Gravity	9.8	9.8	Figure 10	Figure 11
Mass of Cart	1.0	**30.0**		
Mass of Pole	0.1	**2.1**		
Length	0.5	0.5		
Force Applied	10	10		
Tau	0.02	0.02		

Figure 10 presents plots corresponding to the episode-wise performance of the cart pole agent. On the *x*-axis, the number of episodes is marked against the number of steps taken on the *y*-axis. Agent performance throughout the episodes averages around 52 steps per episode and the maximum of 500 steps per episode is never reached. The performance in these graphs is an illustration of less than adequate performance given by the agent upon introduction to different environmental parameters. The agent is performing here without GAN interference.

In Figure 11, graphs illustrating agent performance with GAN interference are shown. Although not consistent, the agent now takes many more steps in forthcoming episodes as compared to Figure 10. Each graph demonstrates the performance variation of the agent with tweaks in the Input Noise Vector. After GAN, agent performance throughout the episodes averages around 297 steps per episode, and a count of 495 steps per episode is also achieved which is very close to the maximum of 500 steps.

5. Discussion

Though GAN and Reinforcement Learning have previously been used in literature, this paper is trying to bring the two methodologies together for a novel use case as we venture into industrial applications for robots. Further down the line, this concoction of GAN and RL will turn out to be fruitful as advances are innovated. Expected improvements over the years could be to have better integration, more variability tolerance in terms of load, etc.

6. Conclusions and Future Scope

In this paper, an RL and GANs-based methodology is proposed for handling the changing load-carrying requirements for an industrial robot agent. The Q learning-based algorithm is implemented for generating the RL agent's behavior. However, for a different payload condition, the RL algorithm needs to be retrained. To avoid this, GANs based model is proposed, which can generate different Q-tables for changing load conditions based on different input noise vectors. For the demonstration of the approach, a pilot environment of CartPole from OpenAI Gym is considered. From the results it can be observed that using this proposed methodology we can achieve impressive performance on increased load on the robot agent without requiring any additional training. Increased load in this context is the increased cart weight and increased pole weight in the CartPole-v1 environment. This pilot experiment can be extended to more complex real-world applications through customizable simulation platforms. Such research could result in being very beneficial in industrial scenarios wherein tasks include the transportation of payloads from one part of a factory to another.

Author Contributions: Conceptualization, R.W. and K.K.; methodology, N.P., V.T. and R.W.; software, validation, formal analysis, investigation, N.P. and V.T.; writing—original draft preparation, N.P. and V.T.; writing—review and editing, N.P., V.T. and R.W.; supervision, R.W. and K.K.; project administration, R.W. and K.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data can be found at: https://github.com/neelabhpaul/Variable_Payload_Agent.

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

References

1. Bogue, R. Growth in e-commerce boosts innovation in the warehouse robot market. *Ind. Robot. Int. J.* **2016**, *43*, 583–587. [CrossRef]
2. Schultz, D.E.; Block, M.P. U.S. online shopping: Facts, fiction, hopes and dreams. *J. Retail. Consum. Serv.* **2015**, *23*, 99–106. [CrossRef]
3. Amazon Prime and "Free" Shipping. Available online: https://escholarship.org/uc/item/0681j9rr (accessed on 5 November 2021).
4. Laber, J.; Thamma, R.; Kirby, E.D. The Impact of Warehouse Automation in Amazon's Success. *IJISET-Int. J. Innov. Sci. Eng. Technol.* **2020**, *7*, 63–70.
5. Bouman, A.; Ginting, M.F.; Alatur, N.; Palieri, M.; Fan, D.D.; Touma, T.; Pailevanian, T.; Kim, S.K.; Otsu, K.; Burdick, J.; et al. Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Las Vegas, NV, USA, 24 October 2020–24 January 2021; pp. 2518–2525. [CrossRef]
6. Van Hasselt, H.; Wiering, M.A. Reinforcement Learning in Continuous Action. In Proceedings of the 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, Honolulu, HI, USA, 1–5 April 2007; pp. 272–279. Available online: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4220844 (accessed on 17 June 2022).
7. Mahmood, A.R.; Korenkevych, D.; Vasan, G.; Ma, W.; Bergstra, J. Benchmarking Reinforcement Learning Algorithms on Real-World Robots. In Proceedings of the 2nd Conference on Robot Learning, Zurich, Switzerland, 29–31 October 2018; pp. 561–591. Available online: https://proceedings.mlr.press/v87/mahmood18a.html (accessed on 6 December 2022).
8. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
9. Tian, Y.; Wang, Q.; Huang, Z.; Li, W.; Dai, D.; Yang, M.; Wang, J.; Fink, O.; Zürich, E.; Europe, N. Off-Policy Reinforcement Learning for Efficient and Effective GAN Architecture Search (Supplementary Material). In Proceedings of the 16th European Conference on Computer Vision (ECCV 2020), Glasgow, UK, 23–28 August 2020.
10. Sarmad, M.; Korea, S.; Lee, H.J.; Kim, Y.M. RL-GAN-Net: A Reinforcement Learning Agent Controlled GAN Network for Real-Time Point Cloud Shape Completion. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 5898–5907.
11. Xu, D.; Zhu, F.; Liu, Q.; Zhao, P. Improving exploration efficiency of deep reinforcement learning through samples produced by generative model. *Expert Syst. Appl.* **2021**, *185*, 115680. [CrossRef]
12. Rao, K.; Harris, C.; Irpan, A.; Levine, S.; Ibarz, J.; Khansari, M. RL-CycleGAN: Reinforcement Learning Aware Simulation-To-Real. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020.
13. Kasgari, A.T.Z.; Saad, W.; Mozaffari, M.; Poor, H.V. Experienced Deep Reinforcement Learning with Generative Adversarial Networks (GANs) for Model-Free Ultra Reliable Low Latency Communication. *IEEE Trans. Commun.* **2021**, *69*, 884–899. [CrossRef]
14. Zhan, H.; Tao, F.; Cao, Y. Human-guided Robot Behavior Learning: A GAN-assisted Preference-based Reinforcement Learning Approach. *IEEE Robot. Autom. Lett.* **2020**, *6*, 3545–3552. [CrossRef]
15. Watkins, C.J.C.H.; Dayan, P. *Q-Learning*; Kluwer Academic Publisher: Boston, MA, USA, 1992; Volume 8.
16. Image Synthesis—Noise Generation. Available online: https://homepages.inf.ed.ac.uk/rbf/HIPR2/noise.htm (accessed on 31 October 2021).
17. Miljković, Z.; Mitić, M.; Lazarević, M.; Babić, B. Neural network Reinforcement Learning for visual control of robot manipulators. *Expert Syst. Appl.* **2013**, *40*, 1721–1736. [CrossRef]
18. Duguleana, M.; Mogan, G. Neural networks based reinforcement learning for mobile robots obstacle avoidance. *Expert Syst. Appl.* **2016**, *62*, 104–115. [CrossRef]
19. Hu, X.; Liao, X.; Liu, Z.; Liu, S.; Ding, X.; Helaoui, M.; Wang, W.; Ghannouchi, F.M. Multi-Agent Deep Reinforcement Learning-Based Flexible Satellite Payload for Mobile Terminals. *IEEE Trans. Veh. Technol.* **2020**, *69*, 9849–9865. [CrossRef]
20. Kim, P. Convolutional Neural Network. In *MATLAB Deep Learning*; Springer: Singapore, 2017; pp. 121–147. [CrossRef]
21. Radford, A.; Metz, L.; Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv* **2016**, arXiv:1511.06434.
22. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym. *arXiv* **2016**, arXiv:1606.01540.
23. Bjorck, J.; Gomes, C.; Selman, B.; Weinberger, K.Q. Understanding Batch Normalization. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; pp. 7694–7705.

24. Fred Agarap, A.M. Deep Learning using Rectified Linear Units (ReLU). Available online: https://github.com/AFAgarap/relu-classifier (accessed on 19 July 2021).
25. Dubey, A.K.; Jain, V. Comparative Study of Convolution Neural Network's Relu and Leaky-Relu Activation Functions. In *Applications of Computing, Automation and Wireless Systems in Electrical Engineering*; Lecture Notes in Electrical Engineering; Springer: Berlin/Heidelberg, Germany, 2019; Volume 553, pp. 873–880. [CrossRef]
26. Statistical Analysis Based on a Certain Multivariate Complex Gaussian Distribution (An Introduction) on JSTOR. Available online: https://www.jstor.org/stable/2991290?seq=1 (accessed on 30 May 2021).