Journal of Applied Data Sciences
Vol. 2, No. 2, May 2021, pp. 27-35

ISSN 2723-6471

Training Autonomous Vehicles in Carla model using Augmented Random Search Algorithm

Riyanto [1,*], Abdul Azis [2], Tarwonto [3], Wei Li Deng [4]

[1,2,3] Universitas Amikom Purwokerto, Indonesia
[4] Southeast University, Nanjing, China
[1] riyanto@amikompurwokerto.ac.id*; [2] abdazis9@amikompurwokerto.ac.id; [3] tarwoto@amikompurwokerto.ac.id; [4] dengli@scut.edu.cn
* Corresponding author

(Received July 1, 2019 Revised October 21, 2019 Accepted October 29, 2019, Available online October 29, 2019)

Abstract

Background, CARLA is an open source simulator for autonomous driving research. CARLA has been developed from scratch to support the development, training and validation of autonomous driving systems. In addition to open source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that are created for this purpose and can be used freely. *Methodology,* Augmented Random Search (ARS) is a method that trains a one-layer perceptron on input data by adding and subtracting a series of random noise repeatedly to the weight, recording the total rewards generated by this modification in separate episodes, then making weighted modifications to weights based on these rewards in a specified number of episodes, scaled at a predetermined learning rate. Test the ability of the Augmented Random Search (ARS) algorithm to train driverless cars on data collected from the front cameras per car. *Result,* in this study, a framework that can be used to train driverless car policy using ARS in Carla will be built using it as the main algorithm and will provide a more realistic representation of how humans apply these controls. *Novelty*, this makes ARS a very lightweight method for studying complex control tasks, and the study authors found that ARS offered at least 15x more computational efficiency than the fastest competing learning methods. More consistent results from Autonomous Vehicle Training have been demonstrated using CARLA, this research has succeeded in opening up most of the opportunities for further study on the same topic, considering how many unique situations the workers in this study experienced in the Carla simulator during the exercise.

Keywords: Self Driving; CARLA; ARS; BipedalWalker; Deep Q-Learning;

1. Introduction

Driverless car policy-making is approached from multiple directions. As Lex Fridman summarized in his 2020 Deep Learning State of Art lecture, leading research by Tesla and Waymo can be classified into two schools of thought: learning-based and map-based, respectively. For Waymo, sensory perception issues serve as an additional tool for making safe use of the underlying navigation system, whereas the Tesla Autopilot learning-based system constantly uses sensory data to improve their policy predictions by establishing edge associations. This study is most similar to an exploration of the basics of Tesla's learning-based method, in that the GPS and Lidar systems were not used, and instead the only data used to train policy came from camera sensors.

In terms of learning-based automation, Deep Q-Learning is one of the methods to be studied. Sentdex provides a series of tutorials on practicing self-driving car policy with Deep Q Learning and Carla. Deep Q Learning has drawbacks, especially the complexity of the computation. To apply the algorithm to this task, Sentdex trains a neural network on each frame received from the camera, and has two neural networks operating simultaneously. For someone without advanced computer hardware or without paid access to cloud development, this method is unapproachable. Carla itself presents computational challenges even for viable hardware, and this can be a barrier to travel researchers from using computationally complex learning algorithms with simulators. Further, Q-Learning is generally limited by the fact that the predicted Q values it generates correspond to discrete action spaces. When one considers multiple continuous value controls such as throttle, brakes and steering, partitioning these control values

into discrete actions with increased resolution creates an explosive size for the action space. Establishing policies that produce sustainable control values seems more appropriate for tasks such as driving.

A new option on the RL training algorithm menu was proposed in a 2018 paper by Mania, Guy, and Recht, and may not have received the full attention it deserves. Augmented Random Search (ARS) is a method that trains a one-layer perceptron on input data by adding and subtracting a series of random noise repeatedly to the weight, recording the total rewards generated by this modification in separate episodes, then making weighted modifications to weights based on these rewards in a specified number of episodes, scaled at a predetermined learning rate. The study authors used the MuJoCo task benchmark to make a case for their proposed algorithm, showing that it is capable of achieving competitive to superior results compared to top competitor model-free methods on continuous control tasks at significantly less computational costs. Since this uses only one perceptron layer, there is only one weight layer to train, and because the adjustments are random, there is no need to calculate the gradient of the loss function at each step. This makes ARS a very lightweight method for studying complex control tasks, and the study authors found that ARS offered at least 15x more computational efficiency than the fastest competing learning methods.

2. Augmented Random Search

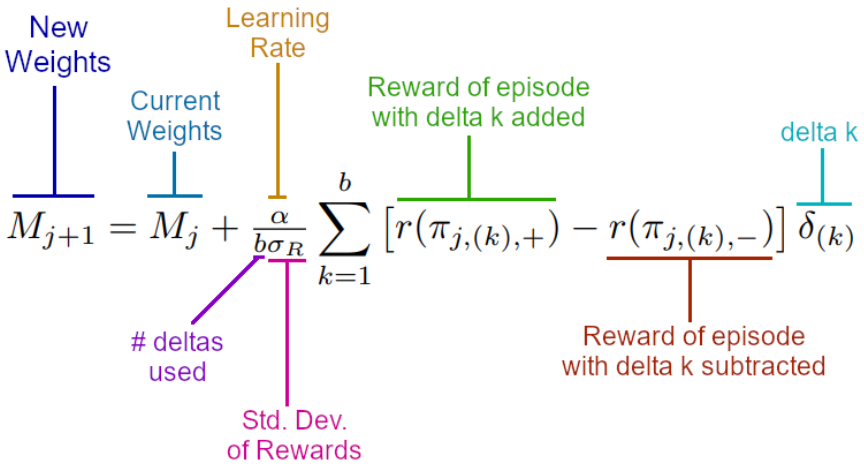

Fig. 1. ARS Formula

ARS was successfully demonstrated in a variety of robotic control tasks (including four-legged drives) with a linear policy [1]. This type of policy has also been successfully applied to four-legged Minitaur robots [2]. Specifically, [3] uses a policy modulation path generator (PMTG) which modulates the path and simultaneously modifies the output to obtain the final command for the motor. It is used successfully with a linear policy to achieve a gait of gait and run [4]. Tirumala et al [4] extended the PMTG framework to include a rotating gait in the Minitaur. However [5], the turns shown by the Minitaur were limited due to the absence of the kidnapping motor. In addition, the PMTG framework updates the corner commands after each time step [6-8]. Therefore, with a view to lowering computational overhead in hardware, and including abduction-based gait such as sideways, twisting in place, we make slight deviations from this approach, by directly obtaining a toe trajectory of trained policy [9]. The policy is updated every half step, and the motor is instructed to track the resulting trajectory [10]. We demonstrated the forward running, backward, side-step and turn gait library, in a custom-built four-legged Stoch2 robot [11-14]. Towards the end we demonstrated robustness by showing that linear policies are able to resist disturbances such as push and push [15].

3. Method

We were immediately prompted to consider the possible application of this approach to reinforcement learning in the field of autonomous navigation. Sentdex has provided a framework for collecting RGB camera sensor data from Carla to train the Deep Q Network (DQN), and we saw an opportunity to use its car environment design to test the ARS algorithm from a 2020 study on this same task [16].

We discovered ARS as a result of our forays into DQN, when we came across an informative video series on reinforcement learning by Skowster the Geek (Colin Skow), in which he decided to include a short video about ARS because of its relevance to the theme. Lured by the simplicity of the algorithm, we finally decided to take a closer look at it, as we began to realize that our hardware might not be able to achieve meaningful results for training DQN at Carla, after seeing less compelling results that Sentdex was able to do. Achieve far superior hardware usage over the course of several days. First, we tested this code in the BipedalWalker environment to see its efficacy for ourselves. Below we can see the episode's reward curve for the training steps for this test.

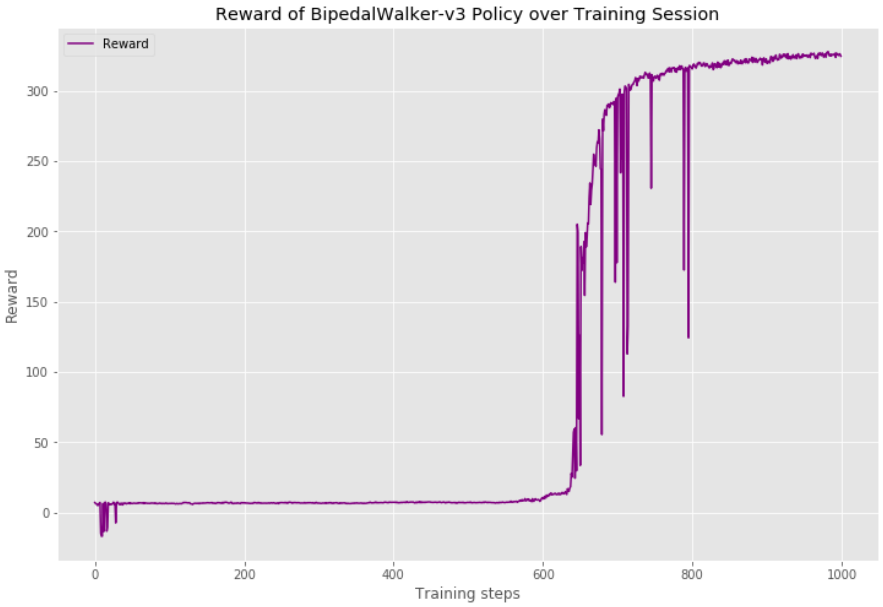

Fig. 2. Reward of BipedalWalker-v3 Policy over Training Session

We can see that for the first 600 training steps the reward curve is basically flat, but eventually the random deltas start building policy in all the right places, creating a sharp curve for increasing rewards eventually leveling up around the 800th episode, with some occasional mistakes along the way that will eventually also disappear at this point. Below, we can see the behavior that results from this trained policy in the BipedalWalker environment:

When we saw that the algorithm could achieve impressive results with computational ease over a reasonable number of episodes, we decided to see if we could incorporate the Carla Sentdex vehicle environment into this framework. To increase the input data on which the algorithm is trained, we convert raw RGB camera data into something that can represent general edge information by first passing the RGB camera frame through a pre-trained Convolutional Neural Network (CNN) called VGG19 (available in the TensorFlow / Keras package) on the way to the ARS algorithm. For this study, the 'imagenet' weight of VGG19 was used. This is an example of Transfer Learning, where we can take advantage of the lower layers of neural networks previously trained on large amounts of image data to apply generalized edge detection to different problems. Since we don't need to train all of these layers, we can simply use their prediction output as input into the single layer ARS perceptron.

The ARS input needs to be normalized, and usually this is done by taking the running statistics for each of the input components, then using these statistics to filter the average / standard deviation (normalization) of the future input. This allows the algorithm to create a suitable distribution to normalize the input over time as it experiences more states, without requiring any prior knowledge of this distribution. For this study, since the predicted output of VGG19 always occurs on a scale of 0 to 10, this method of normalizing the input by running statistics was replaced by simply dividing the input by 10 to normalize it [17]. Further studies testing standard filtration methods in this context may be needed, but appear to cause problems by treating invisible edge scenarios as extreme outliers when they occur after a large number of states have been observed [18]. The scale on which the input is located will affect the scale on which the weights are located, so adjustments to this part of the process will have an impact on the speed of learning and which standard delta deviation is most appropriate to facilitate learning. This is one of the reasons why input normalization is so important.

The Sentdex vehicle environment needs to be modified to use a sustainable action space with continuous control value, and the reward system needs to be adjusted to work more effectively. When self-driving cars are punished for collision, they tend to learn to drive in circles to avoid it, so attention is paid to finding ways to punish extreme or consistent directional driving, and rewarding movement in a straight line at speed [19]. This requires careful consideration. Since collisions are penalized, the rewards for speed and straight line must be high enough to compensate for the penalty for the many crashes that are bound to be encountered when the car is moving faster and turning fewer turns, so that this penalty won't be that high. to prevent the agent from achieving this goal. However, the collision penalty must be high enough if the hope is that the agent will eventually learn not to hit things. It can logically be expected that the edge scenarios which need to be studied for collision avoidance while driving at speed will only be experienced by agents when workers encounter them as a result of freer movement. In other words, the agent must first learn to move before he can find a pattern to avoid while moving. That being said, with no results comparing different reward / punishment systems, this takes a bit of guesswork.

The predictions for throttle, steering, and brakes coming from the perceptron are cut to fit Carla's appropriate control range (0 to +1, -1 to +1, and 0 to +1, respectively), but this means prediction of the value itself could fall outside this range. For throttle and brakes, this problem is ignored. However, for steering, which is generally better done with nuance, consideration is made by punishing each frame with the absolute value of the steering control value for that frame averaged by the absolute value of the average steering control value across all current frames. episodes, so the model will learn to prefer to keep the steering controls close to zero, because an extreme score would add a very negative score to an episode. Sitting still will be penalized to ensure that the car will not learn to avoid the penalty by not moving. The RGB camera has been adapted to shoot at a resolution of 224x224 pixels, as this is the image size used to train the CNN VGG19. Below, we can see an example of an image of Carla's world seen through one of these cameras.

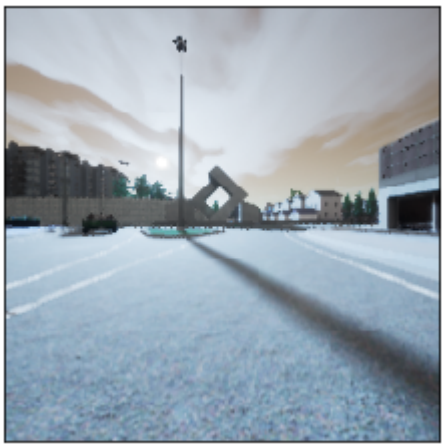

Fig. 3. Worker Overview

Now that we have an idea of what workers perceive as input, we can consider a simplified visual depiction of the data flow through each worker from camera, through policy, and to control:

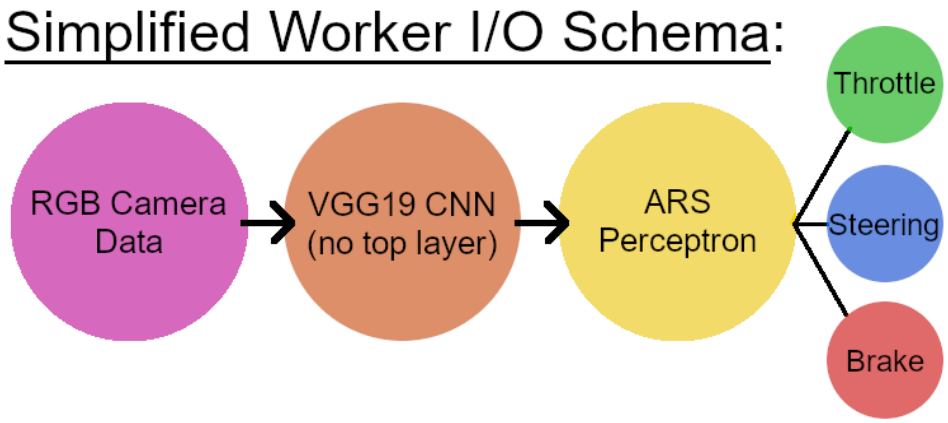

Fig. 4. Simplified Worker I/O Schema

After we successfully linked the Sentdex vehicle environment into the Skow ARS framework, we realized that the training would still be very time consuming as the ARS process had to perform a number of episodes (2x the number of deltas tested for each update step) before it could make any policy adjustments. , and since in the context of Carla's vehicle dealer training, each of those iterations will take a few (in this case 15) seconds, it would be best to try and do these iterations in parallel, and collect the results from multiple workers at each update step.

Now that we were able to train the same BipedalWalker environment using the code in the ARS repository, we knew it was time to set up linking in the Sentdex vehicle environment into this code. This takes some work, as the code in the ARS repo is quite complex, and it's engineered to work specifically with a gym environment. It is also not designed to retrieve a previously saved policy of retrieving training where it was left off, which we want to ensure is an option when working with tasks that inevitably require days of training, which people might want to interrupt periodically, or recover if it occurs. This gives us a great opportunity to dig into the core and core of coding with Ray,

and we look forward to training agents on future large clusters with that knowledge. On our gaming laptop we were able to run a local cluster that could handle 4 car workers running on Carla servers at once, which is much better than just running one at a time. The generated code that trains this Carla vehicle environment using ARS in parallel can be found in the ARS_Carla folder in the project repository. This is a Frankensteinesque combination of Sentdex's CarEnv for Carla, code from the ARS repository, and our own modifications / additions required for this task. In the next few paragraphs, we'll quickly summarize some of our modifications.

As discussed above, the ARS process usually normalizes inputs by creating running statistics of the mean and standard deviation of the observational space components, so as to normalize these inputs effectively as more states are observed, without requiring prior knowledge of the input distribution. This functionality is not required in the context of this task, as the CNN VGG19 output values are already scaled against each other by a known range between 0 and 10, so it is only necessary to divide the array by 10. However, we want to retain this functionality for later comparisons, so we break it and wrap it into a boolean parameter called 'state_filter' which can be passed as the code is executed.

We've also added optional functionality to pass existing policies via the utilization of the new 'policy_file' parameter, which can fetch the location of the .csv or .npz containing weights, and, in the case of the .npz file, possible information to initialize the observation distribution of the state filter which is used to normalize inputs if policies are trained with one of them. This allows a person to continue training where they left off at a later time. Next, we created two additional parameters 'lr_decay' and 'std_decay' which would reduce the learning speed and random noise size applied to the weights over time, to allow for earlier exploration, and ultimately support smaller learning steps once the agent has some training under his belt. Another parameter, 'show_cam', accepts an integer value specifying the number of worker cameras that will be made visible to the user during training. For long training episodes, we set it to zero to save CPU overhead, and enable it later to see the performance of the previously trained policy from a first-person perspective. It is always possible to view workers from an aerial viewpoint from the Carla server window, regardless of how many cameras are displayed.

ARS usually works using the reward amount across all steps in an episode (aka launch). Since each step for our car's environment is a frame that the camera sees, this creates challenges, such as when workers were asked to report how many frames they saw in each episode, it was found that any of the workers had frames per second. seen inconsistently over time, depending on how well the CPU is performing at all times. This means that adding up rewards per episode alone can give us an inaccurate representation of worker performance during launch, as some workers may see more frames per second than others due to fluctuating computer performance, and therefore have more opportunities to log reward or punishment. For this reason, it was decided that a more precise measurement in this context would be the average reward per step in an episode, calculated simply by taking the sum of the rewards for all steps of the episode and dividing by the number of saw steps of the worker. In this way, we can normalize the score to take into account the variable frame rate during training.

The episode ends as soon as a worker registers the collision. This creates an interesting problem, as Carla drops vehicles onto the map from a very small height (presumably to prevent them from getting stuck on the pavement), and sometimes this creates minor shocks to the vehicles listed as crashing in the first one. episode frames, effectively stopping the episode immediately with recorded penalties whenever this occurs. For this reason, we instruct workers to ignore collisions in the first frame of an episode, which fixes the problem and gives each delta a fighting chance to show its value.

4. Results and Discussion

A policy evaluation is carried out every 10 training steps by applying the current policy at 32 launches and recording descriptive statistics of the awards generated by the policy during this launch. Below, we can see a chart summarizing this evaluation during the 5 day training session conducted for this study.

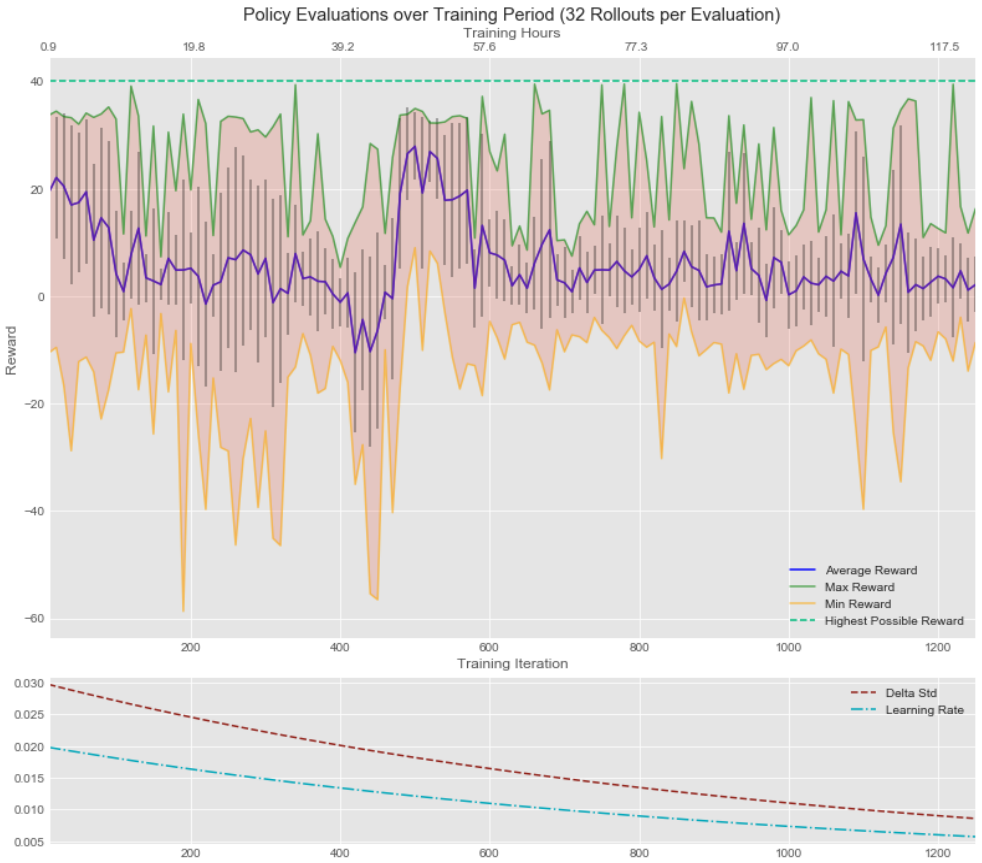

Fig. 5. Policy Evaluations over Training Period

We can see from the chart above that after 1250 training iterations, no substantial progress was made in the average reward over the 32 launches of each evaluation step. There does appear to be a reduction in the standard deviation of rewards over time, but without the desired increase in average reward.

There was a slow decrease in average reward in the initial training period, then a spike upward of about 500 training iterations, after which it decreased again. This may be an indication that the learning speed was set too high for this task, and more tests using different learning speeds were a good fit based on these results. This could also be an indication that delta std is also set too high, which also needs further testing.

Minimum rewards appear to increase over time, with much more extreme lows in the first part of the training period. The agent is penalized for a large steering control value, so this may represent a period before the weights are adjusted sufficiently to prevent extreme values for this control.

The maximum reward that can be achieved for each launch is 40, and we can see that individual launches periodically reach this level, even at the start of training. This is a good time to dive into the problems of the various circumstances that our workers experience once they have been put on the map. In this study, each worker was spawned at a random location on the map to start each episode, and therefore, the rewards that resulted from worker behavior were closely related to the location of the spawning grounds. A worker who is spawned with an open stretch of road can get close to the maximum score by swinging it and not driving it, while the same behavior at different spawning points will result in a collision, and hence a penalty. This makes it difficult for the algorithm to fairly evaluate the effectiveness of each adjustment against the weights, since pure luck plays a large role in how well the episode performs, and therefore in the contribution of each delta to the update step. This has to be tackled in the

future, and a good start is to spawn workers testing a particular delta at the same location for the positive and negative additions of that delta, so that the rewards compared in the renewal step are generated under identical conditions. This will result in a more meaningful contribution from each delta to the update step.

5. Conclusion

In this study, a framework that can be used to train driverless car policy using ARS in Carla has been built. Although effective policies were not achieved after the first round of training, many insights on how to improve these outcomes in the future have been obtained. The learning framework created provides the opportunity to easily apply these insights into the future.

The first problem that must be overcome is the variability of the circumstances in which workers are placed during the training steps. As noted above, a good first step in dealing with this problem is to have the positive and negative additions of each tested delta from the same spawning location on the map, to produce a more fair and meaningful reward comparison in the update step.

Another thing to consider is that BipedalWalker, which has the advantage of being placed in the exact same scenario for each episode, still takes about 600 update steps before the training process starts to have a noticeable effect on the rewards. Considering how many unique situations the workers in this study faced in the Carla simulator, it may come as no surprise that meaningful performance gains were not achieved after 1250 training steps, and it was possible that giving agents more time to train might ultimately pay off with an effective policy. In the future, training agents with more compute resources will help answer this question, as more workers can operate simultaneously and divide work in more ways, which will reduce the overall time requirements for each update step.

In this study, brake and throttle were kept as separate controls, but it may be more appropriate to combine them into a single continuous control value between -1 and +1, as doing so would prevent them from being applied at the same time, no matter how. the weights are adjusted, and will provide a more realistic representation of how humans apply these controls.

Changing the reward system can also result in better learning by agents, particularly measures of punishment for collisions. The agent will not be able to learn the correct behavior for edge information leading to a collision without experiencing the collision, so it may be appropriate to reduce the collision penalty so that the agent feels freer to explore this scenario, but not so much that it does not change policies to avoid similar collisions in the future.

The next thing to do is test more values for the hyperparameters and compare the results for each, which again could be much better done using larger hardware than the gaming laptop used in this study, as training time could be drastically reduced. by having more workers in parallel. A smaller value for learning speed should be explored, as we saw a decrease in the average reward in the initial period of this training session, which could be an indication that the learning speed was too large. Smaller values for delta standard deviation can also be explored. Furthermore, the number of deltas generated as well as the amount of deltas used in the update step can be changed to observe their effect on training.

References

[1] A. R. Conn, N. I. M. Gould, and P. L. Toint, “Globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds,” *SIAM J. Numer. Anal.*, vol. 28, no. 2, pp. 545–572, 1991, doi: 10.1137/0728030.

[2] M. Geng, K. Xu, B. Ding, H. Wang, and L. Zhang, “Learning data augmentation policies using augmented random search,” *arXiv*, 2018.

[3] A. K. Tiwari and S. V. Nadimpalli, “Augmented Random Search for Quadcopter Control: An alternative to Reinforcement Learning,” *arXiv*, 2019, doi: 10.5815/ijitcs.2019.11.03.

[4] S. Tirumala *et al.*, "Gait library synthesis for quadruped robots via augmented random search," *arXiv*, 2019.

[5] V. Kurenkov, H. Hamed, and S. Savin, "Learning Stabilizing Control Policies for a Tensegrity Hopper with Augmented Random Search," *arXiv*, no. 19, 2020.

[6] R. B. Gramacy *et al.*, "Modeling an augmented lagrangian for blackbox constrained optimization," *Technometrics*, vol. 58, no. 1, pp. 1–11, 2016, doi: 10.1080/00401706.2015.1014065.

[7] H. Teimourzadeh, F. Jabari, and B. Mohammadi-Ivatloo, "An augmented group search optimization algorithm for optimal cooling-load dispatch in multi-chiller plants," *Comput. Electr. Eng.*, vol. 85, no. xxxx, p. 106434, 2020, doi: 10.1016/j.compeleceng.2019.07.020.

[8] Y. Shang *et al.*, "Stochastic dispatch of energy storage in microgrids: A reinforcement learning approach incorporated with MCTS," *arXiv*, pp. 1–11, 2019.

[9] Maiti and Bidinger, "COMBINING RESULTS FROM AUGMENTED DESIGNS OVER SITES," *J. Chem. Inf. Model.*, vol. 53, no. 9, pp. 1689–1699, 1981.

[10] X. Chen, P. Wei, W. Ke, Q. Ye, and J. J. B, "Forecasting Events Using an Augmented Hidden Conditional Random Field," vol. 9006, no. November, pp. 354–365, 2015, doi: 10.1007/978-3-319-16817-3.

[11] B. I. Kim, H. Li, and A. L. Johnson, "An augmented large neighborhood search method for solving the team orienteering problem," *Expert Syst. Appl.*, vol. 40, no. 8, pp. 3065–3072, 2013, doi: 10.1016/j.eswa.2012.12.022.

[12] H. Li, K. M. Lam, and M. Wang, "Image super-resolution via feature-augmented random forest," *Signal Process. Image Commun.*, vol. 72, no. 2, pp. 25–34, 2019, doi: 10.1016/j.image.2018.12.001.

[13] Z. Liu *et al.*, "Subgraph-augmented path embedding for semantic user search on heterogeneous social network," *Web Conf. 2018 - Proc. World Wide Web Conf. WWW 2018*, pp. 1613–1622, 2018, doi: 10.1145/3178876.3186073.

[14] I. V. Tetko, P. Karpov, R. Van Deursen, and G. Godin, "State-of-the-art augmented NLP transformer models for direct and single-step retrosynthesis," *Nat. Commun.*, vol. 11, no. 1, pp. 1–24, 2020, doi: 10.1038/s41467-020-19266-y.

[15] H. Akeb, M. Hifi, and S. Negre, "An augmented beam search-based algorithm for the circular open dimension problem," *Comput. Ind. Eng.*, vol. 61, no. 2, pp. 373–381, 2011, doi: 10.1016/j.cie.2011.02.009.

[16] Y. Zhang, A. Albarghouthi, and L. D'Antoni, "Robustness to Programmable String Transformations via Augmented Abstract Training," *arXiv*, 2020.

[17] L. Costa, I. A. C. P. Espírito Santo, and E. M. G. P. Fernandes, "A hybrid genetic pattern search augmented Lagrangian method for constrained global optimization," *Appl. Math. Comput.*, vol. 218, no. 18, pp. 9415–9426, 2012, doi: 10.1016/j.amc.2012.03.025.

[18] Henderi and T. Wahyuningsih, "Comparison of Min-Max normalization and Z-Score Normalization in the K-nearest neighbor (kNN) Algorithm to Test the Accuracy of Types of Breast Cancer," *IJIIS Int. J. Informatics Inf. Syst.*, vol. 4, no. 1, pp. 13–20, 2021, doi: 10.47738/ijiis.v4i1.73.

[19] C. J. M. Liang *et al.*, "AutoSys: The design and operation of learning-augmented systems," *Proc. 2020 USENIX Annu. Tech. Conf. ATC 2020*, pp. 323–336, 2020.