Comparative Analysis of Tuning Hyperparameters in Policy-Based DRL Algorithm for Self-Driving Scenario

Shubhechchha Niraula[1]

[1]*Tribhuvan University, Institute of Engineering, Pulchowk Campus, Lalitpur, Nepal*

---***---

Abstract: *This technical article makes a comparative study on the outcome of policy-based deep reinforcement learning algorithm PPO (Proximal Policy Optimization) at different hyperparameter values in self driving scenario to analyze its effectiveness in navigation of vehicles. The agents are trained and evaluated in OpenAI Gym CarRacing-v0 simulation environment. The action space is continuous and the observation spaces are images of 96x96x3 dimension which is fed to the actor-critic network architecture. Three major hyperparameters discount factor, clip range and learning rate are altered to evaluate their performance in given environment. The behaviors of the agents while tuning the hyperparameters are observed and analyzed against evaluation metrics such as mean episode reward, policy loss, value loss, etc. The agents were found to perform optimally at the discount factor of 0.99, clip range of 0.2 and learning rate of 0.0003 which are also the baseline default values.*

Key Words: *PPO, DRL, Proximal Policy Optimization, Deep Reinforcement Learning, Self Driving*

1. INTRODUCTION

Deep reinforcement learning has prevailed in recent times as a popular algorithm to solve navigation problem for autonomous vehicles. It is seen as the possible tool to replace classical control mechanisms used in autonomous systems with numerous sensors and their compute hardware and software stack by simple output oriented programming logic.

Various types of learning agents have been developed over the years to accomplish multitude of tasks among which model-based algorithms, value-based algorithms and policy optimizing algorithms are popular among researchers. Model based algorithms represent the system in a dynamic model form and tries to optimize the solution. Value based algorithms prepare a table of value for each state-action pair which represents the possible outcome of the agent incorporating the reward value for upcoming outcomes. Policy optimization algorithms guides the agent directly depending upon whether the policy is deterministic or stochastic to maximize the reward based upon the probability distribution of outcomes developed and optimized previously during the training. In this article, we will compare and analyze the performance of Proximal Policy Optimization algorithm to evaluate its navigation efficiency in self driving scenario. The outcomes are weighed up in terms of evaluation metrics such as mean episode reward, policy loss, value loss, etc.

2. RELATED WORKS

The deep learning revolution started at around 2012 as function approximators in various domains which led researchers to investigate the implementation of these neural networks in pre-existing reinforcement learning algorithms. The new approach started using images as the state of the agent instead of position, velocity and distance to collision with other obstacles in the surrounding. Deep Q-Network (DQN) was introduced as a potential solution to solve high-dimensional state space environment with policies that worked efficiently in large array of problems with same algorithm, network architecture and hyperparameters [1]. The DQN agent succeeded in multiple Atari games with pixel and game score as inputs and performed better than majority of algorithms previously proposed, which were effective only in domains with fully defined, low dimensional state space. While DQN could solve problems with discrete action spaces quite effectively, a wide range of real physical environments are much complex and consists of tasks requiring continuous control. Deterministic policy gradient (DPG) algorithms were developed to encounter the problems with continuous action spaces [2]. REINFORCE was one of the early straight forward policy-based algorithms which operated by converging in the direction of performance gradient and hence, separate network for policy evaluation was not trained [3].

Deep Deterministic Policy Gradient (DDPG) was developed adapting to the underlying principles of DQN which could address the problems with continuous action domain [4]. Actor-critic architecture was incorporated in the algorithm which could optimize its own policy based on the state values obtained from critic network. Trust region policy optimization (TRPO) was introduced which prevented the policies from deviating excessively on a single cycle by adding a surrogate objective function to the algorithm [5]. Advantage Actor Critic (A2C) was put forward as synchronous version of Asynchronous Advantage Actor Critic (A3C) algorithm [6]. A2C has larger batch size which ensures the completion of training for each actor in that particular segment before the mean parameters are updated. Soft Actor Critic (SAC) is also a popular off-policy deep reinforcement learning algorithm based on maximizing the entropy by actor to optimize expected reward through randomness [7].

Among the algorithms proposed, Proximal Policy Optimization (PPO) is prominently used in testing self-driving features due to its effectiveness in the given scenario. The algorithm fine-tuned the outcome of previously existing policy-based reinforcement learning algorithms in which surrogate objective function is optimized using stochastic gradient ascent [8]. PPO allows multiple gradient updates over a data sample in mini-batches with repeated epochs. The algorithm, when it first launched, outperformed other online policy gradient methods in tasks including simulated robotic locomotion and Atari games.

Kiran et al. have outlined the development of deep reinforcement learning algorithms on autonomous driving scenario [9]. Chen et al. discussed and experimented with model free DRL algorithms for urban driving conditions [10]. Deep reinforcement learning has been proposed as a solution for mapless navigation, and ways to bridge the gap between virtual and real environment was also addressed [11]. Apart of ground vehicles, DRLs have also been researched and utilized in the domain of unmanned aerial vehicles (UAV) and unmanned underwater vehicles (UUV) [12].

3. SIMULATION SETUP

Despite of reinforcement learning algorithms performing quite well in controlled environments with finite state spaces and pre-defined quantifiable goals, autonomous driving is much complex scenario, especially in real world applications. The challenges are even higher given its safety issues. Hence, the algorithms are tested and evaluated in oversimplified gaming or simulated environments to verify its outcome.

3.1 Environment

The algorithms, in this article, are tested and evaluated on OpenAI Gym CarRacing-v0 environment which is much simplified version for autonomous driving scenario. The environment does not contain any static or dynamic obstacles which makes the task much easier than what would be expected in real world application. Stable baseline algorithm provided by the platform is compared against agents with varied hyperparameter values. Each agent was trained for an approximate of 200,000 timesteps to obtain the result.

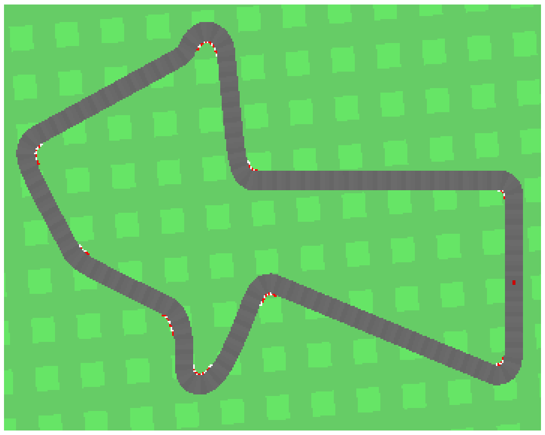

Fig - 1: OpenAI Gym sample racetrack

3.2 Action/Observation Spaces

The action space is continuous with steering value ranging from −1 to 1 for extreme left and right, and 0 for no movement. The gas amount can be increased or decreased within the range of 0 to 1 depending upon the amount of acceleration required. Similarly, braking range is within the value of 0 to 1 for no braking to full stop. The observation spaces are the images of 96x96x3 pixel size and color channels.

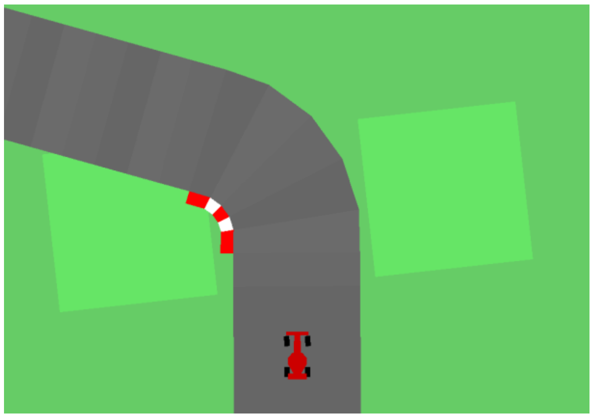

Fig - 2: Observation space sample

3.3 Reward Functions

The racecar environment is considered solved when the agent consistently scores 900+ points. Given that the number of tiles visited is N, reward of +1000/N is granted for every track tile visited. Also, 0.1 point is reduced on each frame encouraging the vehicle to reach the designated score point faster. An episode is completed when all the tiles of the track are visited. If the vehicle is too far away from the track, 100 point is reduced which eventually terminates the episode.

3.4 Network Architecture

Proximal Policy Optimization (PPO) adopts actor critic architecture, and in case of stable baselines provided by OpenAI, the network consists of two separate parts. The feature extractor class uses Convolutional Neural Network (CNN) to extract useful information from the image and is shared by both actor and critic to reduce computation time. The fully connected network maps the features to actions or state value depending upon whether it is being used by actor or critic. The loss function is given by:

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}\,[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

In above equation,

$$L_t^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$$

Also, the 2nd and 3rd term with co-efficient c1 and c2 are the mean squared error of value function and entropy loss respectively. The $r_t(\theta)$ term is the ratio of new policy to the old policy.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

PPO limits the update to policy network by restricting this ratio to certain values so that the sudden performance drop prevalent in actor-critic architecture is avoided. The term $\hat{A}_t$ is introduced to identify the good states and provide advantage to these sates. Also, the learning iterations during training is carried out over small fixed length trajectory of memories and multiple network update per sample is done.

3.5 Hyperparameters

The default hyperparameters and arguments used by the baseline algorithm are tabulated below.

Table - 1: Default hyperparameter values

Hyperparameters	Argument	Default value
Learning rate	learning_rate	0.0003
Number of steps	n_steps	2048
Mini-batch size	batch_size	64
Number of epochs	n_epochs	10
Discount factor	gamma	0.99
Trade-off factor (bias vs variance)	gae_lambda	0.95
Clipping parameter	clip_range	0.2
Entropy co-efficient	ent_coef	0.0
Value function co-efficient	vf_coef	0.5
Maximum gradient clipping value	max_grad_norm	0.5
Verbosity level	verbose	0

The code is written and executed on Jupyter Notebook using PyTorch framework with no GPU acceleration.

4. OBSERVATIONS AND DISCUSSIONS

The algorithms are compared and evaluated in terms of average reward per episode during training and evaluation cycles. Loss and time metrics are also analyzed to better understand the advantage and shortcomings of each algorithm. Three major hyperparameters i.e. discount factor (**γ**), clip range and learning rate (**α**) are altered and their behavior over 200,000 timesteps are observed to analyze their performance. The results are visualized using tensorboard log data and graphs provided by the platform.

4.1 Discount Factor

The discount factor is the measure of how much the policy is influenced by possible return from future steps compared to the immediate reward. The discount factor was altered within the range of 0.8 to 0.999 and the graphs were plotted in Tensorboard. In the figures below, orange, dark blue, light blue, red and pink plots represent the discount factor of 0.8, 0.9, 0.95, 0.99 and 0.999 respectively. The rewards and losses are plotted in y-axis against the timesteps in x-axis.

4.1.1 Graphs

The graphs obtained from the Tensorboad are shown below:

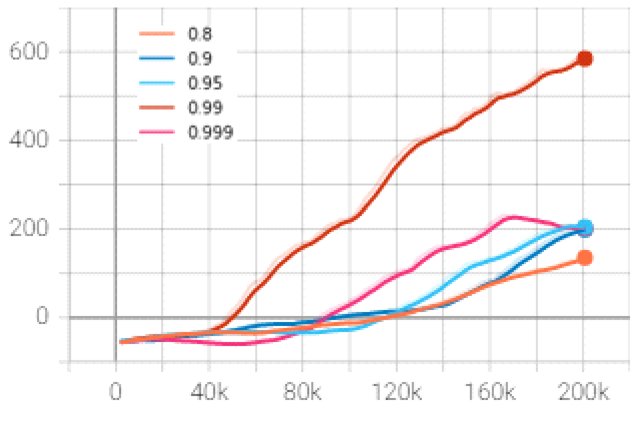

Chart – 1: Mean episode reward (y) vs. timesteps (x)

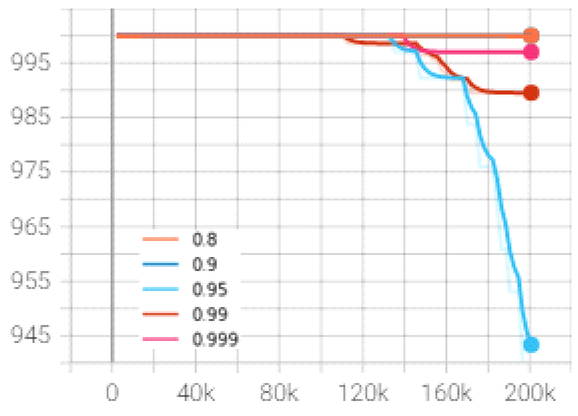

Chart – 2: Mean episode length (y) vs. timesteps (x)

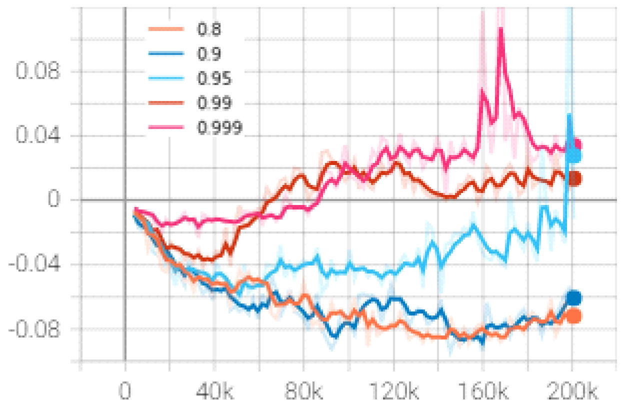

Chart – 3: Policy gradient loss (y) vs. timesteps (x)

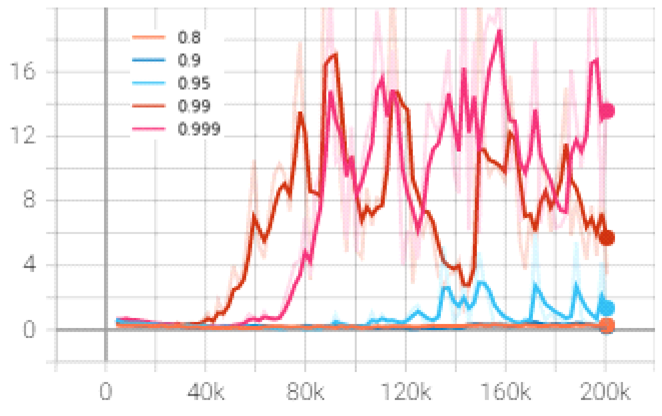

Chart – 4: Value loss (y) vs. timesteps (x)

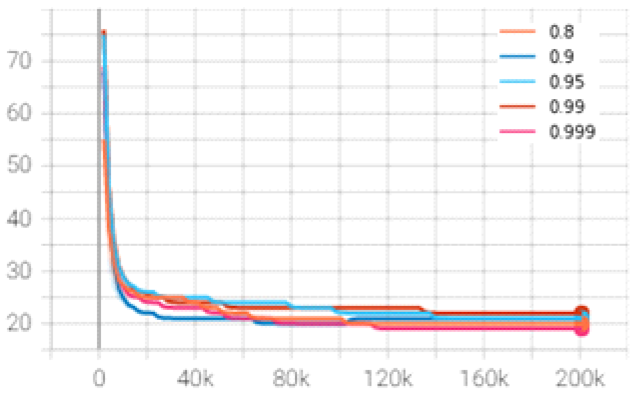

Chart – 5: Frame per second (y) vs. timesteps (x)

4.1.2 Inference

The mean episode reward was found to be maximum at discount factor of 0.99 as represented by red curve which is also the baseline default value. The discount factor was increased and decreased to 0.999 and 0.95 which gave the mean reward of 202 and 204 respectively at 200,000 timesteps, which is much lesser than 594 points obtained from 0.99. The discount factor was further decreased to 0.9 and 0.8 as represented by dark blue and orange curve which further lowered the outcome to 200 and 135 respectively. Hence, it can be inferred from the experiments above that the discount factor of 0.99 is best suited among the tested values for the given environment.

The policy gradient loss seemed to be gradually increasing initially in either direction of the axis and decreasing as the timestep progressed. It can be expected from the pattern that the curve will converge towards zero near the end if it is further trained to 1-2 million timesteps. The policy loss was higher initially because the agent had not learned as much in the beginning, hence large policy update occurred in each iteration. However, the more the agent learned, the policy started becoming more stable resulting in lower loss per update.

4.2 Clip Range

The restriction imposed on the ratio of new policy to old policy of the agent which determines the extent of update that a network can have per iteration is given by clip range. The algorithm was tested against three clip range values 0.1, 0.2 and 0.3 represented by blue, orange and red plots respectively. The progression of timesteps is plotted along x-axis and the rewards and losses are mapped in y-axis.

4.2.1 Graphs

The graphs obtained from the Tensorboad are shown below:

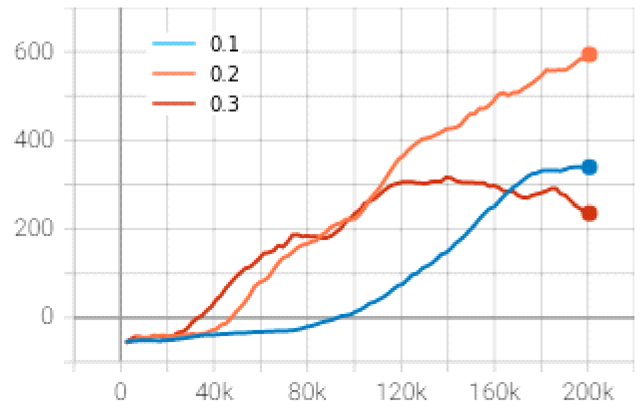

Chart – 6: Mean episode reward (y) vs. timesteps (x)

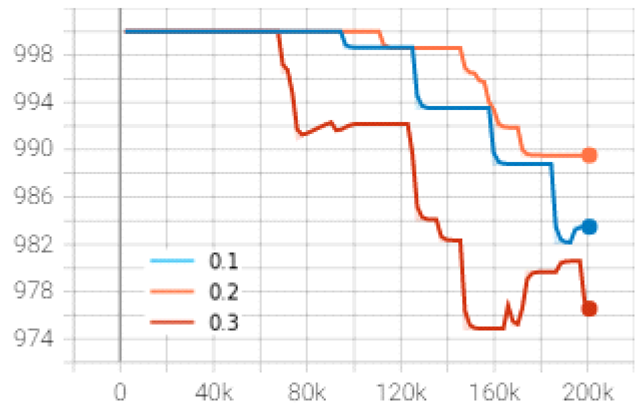

Chart – 7: Mean episode length (y) vs. timesteps (x)

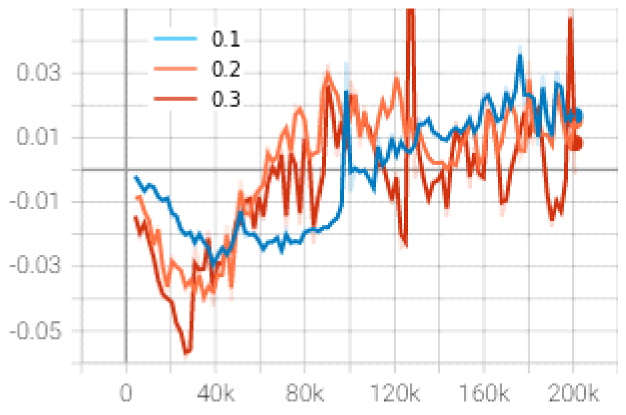

Chart – 8: Policy gradient loss (y) vs. timesteps (x)

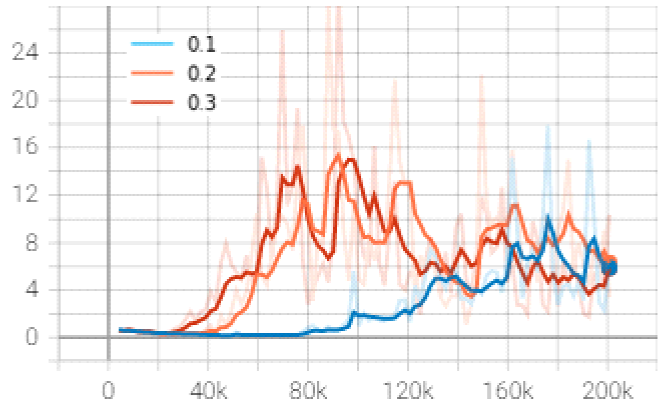

Chart – 9: Value loss (y) vs. timesteps (x)

4.2.2 Inference

The average episode reward started increasing faster in case of clip range 0.3 represented by red curve, however, the growth was not steady and stopped increasing after about 120,000 timesteps. The blue curve representing the clip range of 0.1 started sloping up later than other curves and yet succeeded in overtaking the clip range of 0.3. The average episode reward of default agent with clip range of 0.2 was still maximum than other agents at the landmark of 200,000 timesteps. The above experiment implies that tuning the clip range towards the lower value than the baseline default may produce better results than tuning it towards higher value if it is to be done at all.

The value loss curves were as expected for PPO learning cycles. The loss increased in its early cycles and came down to a steady range once the agent stabilized. The orange and red curve representing the clip range of 0.2 and 0.3 respectively started ascending at around 30k-40k timesteps and reached their peaks at about 100,000 timesteps. The curves started descending and stabilized around the value of 6. The blue curve with clip range of 0.1 started ascending much later and may give better results at higher timesteps if trained for longer. However, at 200,000 mark, the loss was found to be lowest for clip range of 0.2 with the approximate value of 3.5.

4.3 Learning Rate

PPO performs mini-batch stochastic gradient ascent for updating its network parameters, and learning rate controls the pace at which the updates take place. The baseline algorithm was tested against varying learning rate values ranging from 0.03 to 0.000003 to analyze its influence in the agent's overall performance. The learning rate values of 0.03, 0.003, 0.0003, 0.00003 and 0.000003 are depicted by dark blue, light blue, orange, red and pink respectively. As usual, x-axis represents timesteps and y-axis represents rewards and losses.

4.3.1 Graphs:

The graphs obtained from the Tensorboad are shown below:

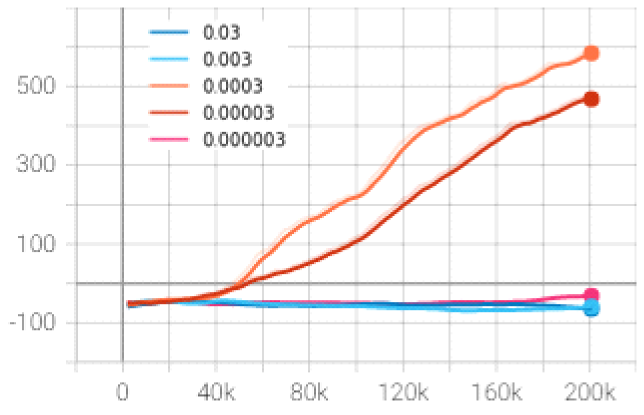

Chart – 10: Mean episode reward (y) vs. timesteps (x)

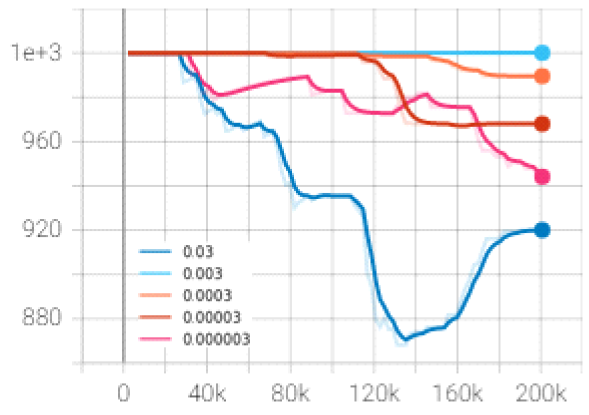

Chart – 11: Mean episode length (y) vs. timesteps (x)

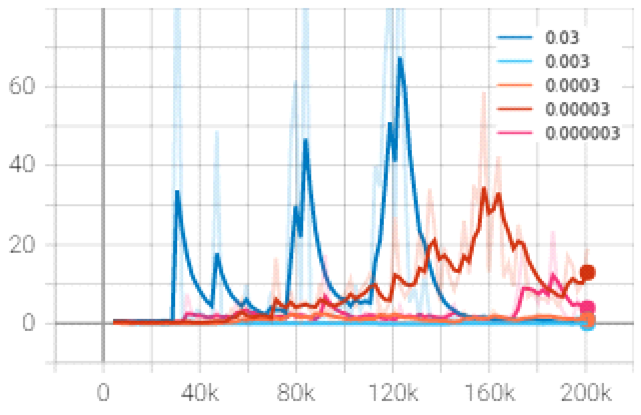

Chart – 12: Total loss (y) vs. timesteps (x)

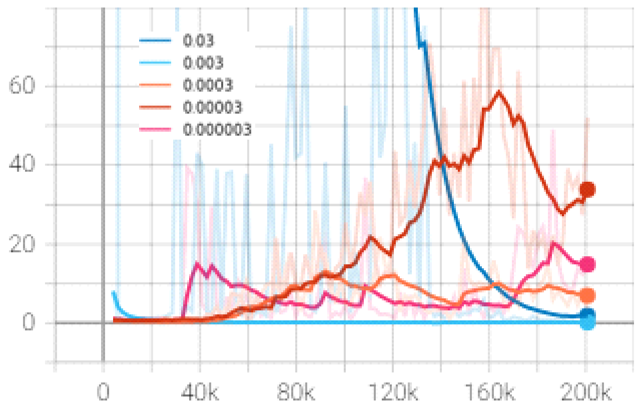

Chart – 13: Value loss (y) vs. timesteps (x)

4.3.2 Inference

The mean episode reward of baseline algorithm with learning rate 0.0003 was found to be at an approximate of 590 points (unsmoothed value) after 200,000 timesteps as depicted by the orange curve. However, while increasing the learning rate to 0.03, the mean episode reward came down to −65 points as shown by dark blue curve. Hence, the learning rate was further decreased to 0.00003 and the episode reward reached an average value of 470 points which is still lesser than that obtained from baseline default. The learning rate was then increased one unit higher than the baseline value to 0.003 which also gave similar result as that of 0.03. On the last experiment, the learning rate was decreased to 0.000003, however, the algorithm didn't converge at all leaving negative average reward. It can be inferred from the above experiments that the learning rate of 0.0003 is optimum among the tested values for the given environment.

In case of the loss plot, a consistent pattern was observed in the total loss of agent with learning rate of 0.03 represented by dark blue curve. The loss appeared to peak up abruptly at times taking much longer time to fall back to its previous range. This is possibly due to the performance falling of actor critic architecture which became prominent when the learning rate was not restricted to small value.

5. CONCLUSION

It can be concluded from the above observations that the hyperparameters used by baseline algorithm as shown in Table 1 are optimized to give the best possible outcome in majority of scenarios. However, the data and graphs obtained were based on training carried out for 200,000 timesteps which is much less to infer the agent's behavior in longer run. In majority of cases, the mean episode reward is within 400-500 which is only half the expected reward for the racetrack to be considered completed. Hence, training the agent for longer timesteps i.e. an approximate of 1-2 millions could yield better insight into the performance of algorithm at different hyperparameter values. For future work, the baseline algorithm can be tweaked to fit in other tools and techniques such as image processing, altering frame stacks and/or discretizing the action space to improve performance in terms of training time and accuracy.

REFERENCES

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529-533.

[2] Silver, David & Lever, Guy & Heess, Nicolas & Degris, Thomas & Wierstra, Daan & Riedmiller, Martin. (2014). Deterministic Policy Gradient Algorithms. 31st International Conference on Machine Learning, ICML 2014.

[3] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3), 229-256.

[4] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[5] Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In *International conference on machine learning* (pp. 1889-1897). PMLR.

[6] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928-1937). PMLR.

[7] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., ... & Levine, S. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

[8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[9] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., & Pérez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*.

[10] Chen, J., Yuan, B., & Tomizuka, M. (2019, October). Model-free deep reinforcement learning for urban autonomous driving. In *2019 IEEE intelligent transportation systems conference (ITSC)* (pp. 2765-2771). IEEE.

[11] Tai, L., Paolo, G., & Liu, M. (2017, September). Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 31-36). IEEE.

[12] Pham, H. X., La, H. M., Feil-Seifer, D., & Nguyen, L. V. (2018). Autonomous UAV navigation using reinforcement learning. *arXiv preprint arXiv:1801.05086*.