IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 10, No. 3, September 2021, pp. 649~656
ISSN: 2252-8938, DOI: 10.11591/ijai.v10.i3.pp649-656

Accelerating the training of deep reinforcement learning in autonomous driving

Emmanuel Ifeanyi Iroegbu, Madhavi Devaraj
School of Information Technology, Mapua Institute of Technology, Makati, Philippines

Article Info

Article history:

Received Sep 25, 2020
Revised Jun 23, 2021
Accepted Jun 25, 2021

Keywords:

Autonomous driving
Deep reinforcement learning
Latent state representation
Variational autoencoder

ABSTRACT

Deep reinforcement learning has been successful in solving common autonomous driving tasks such as lane-keeping by simply using pixel data from the front view camera as input. However, raw pixel data contains a very high-dimensional observation that affects the learning quality of the agent due to the complexity imposed by a 'realistic' urban environment. Ergo, we investigate how compressing the raw pixel data from high-dimensional state to low-dimensional latent space offline using a variational autoencoder can significantly improve the training of a deep reinforcement learning agent. We evaluated our method on a simulated autonomous vehicle in car learning to act and compared our results with many baselines including deep deterministic policy gradient, proximal policy optimization, and soft actor-critic. The result shows that the method greatly accelerates the training time and there was a remarkable improvement in the quality of the deep reinforcement learning agent.

Corresponding Author:

Emmanuel Ifeanyi Iroegbu
School of Information Technology
Mapua Institute of Technology
Makati, Philippines
Email: emmanueliroegbu@gmail.com, eiiroegbu@mymail.mapua.edu.ph

1. INTRODUCTION

Series of automation and machine learning development has progressed through the years in order to make everyday tasks effortless. Case and point, there has been a surge of interest in self-driving cars or autonomous vehicles (AV), the ability to drive safely from one point to another with little to no human intervention. With the recent study and advancement on AV, the attention of different high-tech companies and researchers are rapidly growing. They are highlighting the need for a level five (5) full automation [1], where vehicles can be driven safely without human intervention or supervision. There is still a long way for this to be achieved along with major challenges on how to design an accurate model of an autonomous vehicle in a real-world environment.

Existing decision-making methods for autonomous driving uses a rule-based approach which requires a manually designed driving policy [2], [3]. The problem with such an approach is they lead to inaccurate decision-making, as predefined models are biased in a highly stochastic environment and suffer heavily when tasked with different scenarios. As an alternative, many researchers have used an imitation learning technique that learns a driving policy based on data obtained from an expert driver [4], [5]. Though, imitation learning suffers from several weaknesses because a policy learned from an expert driver is limited to the knowledge of the said driver and most expert drivers only show good driving behavior, which the model may never learn how to deal with a risky situation. Imitating learning is also costly and time-consuming to implement in a real-world environment.

Another method is deep reinforcement learning (RL) it achieves human-level performance in playing games like Go [6] and Atari [7]. There have been some significant breakthroughs in continuous control of deep RL methods [8], [9]. As a result, deep RL has been used to learn a driving policy with no expert data. Despite these breakthroughs, training a deep RL in a highly stochastic environment is challenging due to the high demand of computational cost it requires i.e., to achieve reasonable performance in playing Atari games deep RL required hundreds of millions of training steps [10], resulting in months of consistent training. Similarly, AlphaGo trained data from expert Go players for over 30 million training steps. The sheer amount of computational cost imposed on training a deep RL algorithm undermines its application in autonomous driving.

In this paper, we proposed using state representation learning (SRL) to retrieve relevant information from raw pixel data to accelerate the training of a deep RL agent by learning a low-dimensional state representation of an autonomous driving environment rather than directly using a front view image as input to train a deep RL agent. We used variational autoencoder (VAE) [11], a type of generative neural network model that comprises an encoder and a decoder network. The idea is to utilize backpropagation to train the network to reconstruct a high-dimensional input signal after compressing it into a low-dimensional vector, as shown in Figure 1. The goal of a VAE is to minimize reconstruction loss. To achieve this, we collected raw image data by manually driving the vehicle in the simulated environment. These images are sent as input vectors into a VAE network architecture that comprises an encoder and a decoder. The encoder compresses the raw image data into the most essential components and attempts to reconstruct the original signal by decoding the compressed representation. The pre-trained VAE images are used to train our deep RL agent, which takes the encoded low-dimensional states-input and outputs continuous control commands.

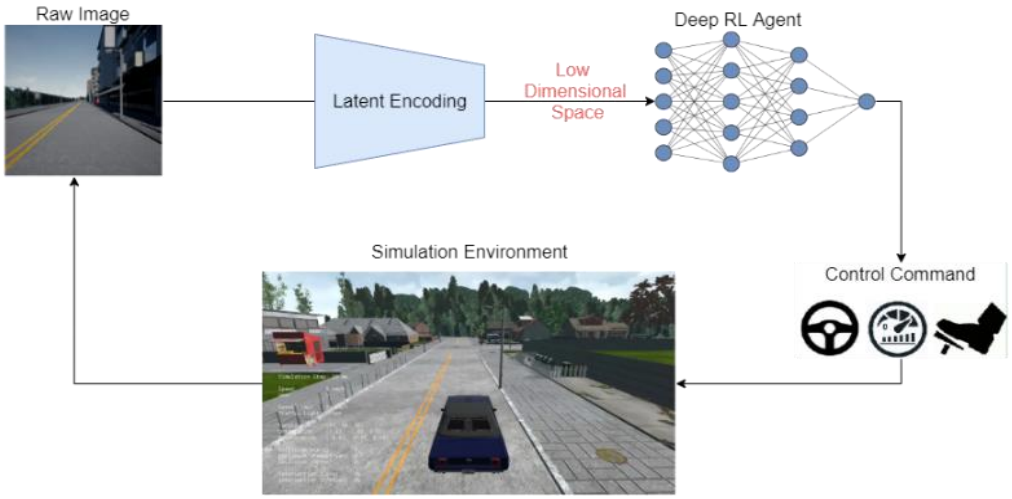

Figure 1. The proposed framework: first the raw captured image is encoded with a VAE and sent as input of a pre-trained low dimensional state space into our agent that learns a driving policy to output control commands such as throttling and steering

Ever since deep RL drew attention after its tremendous success in playing Atari games at a human level performance in high-dimensional state space, there has been some promising application in autonomous driving by many researchers [12]. Some of these researchers are Vitelli and Nayebi [8] which implemented a deep Q-network (DQN) agent on autonomous driving by simply using raw pixel data as input to train the policy network. They tested their method on a racing car simulator (TORCs) to evaluate the performance of the DQN agent. Pan *et al.* [13], acquired real-world images and fed them into an asynchronous advantage actor-critic model to train a suitable driving policy virtual environment. Sallab *et al.* [14] proposed a framework for an end-to-end autonomous vehicle based on a deep RL pipeline using recurrent neural networks to deal with partially observable Markov decision process scenarios in a TORCs simulator. Lillicrap *et al.* [12] tested their proposed continuous control deep reinforcement learning algorithm (deep deterministic policy gradient [DDPG]) on a lane following task in TORCs. Perhaps, there are many related works on autonomous driving with deep RL, however, most of the aforementioned approach is developed and tested in a racing environment or Donkey Car simulator [15] which is not practical in a real-world environment or used a raw pixel data directly from a front camera or manually designed feature

representation which can be computational intensity and time-consuming to train especially in a high-dimensional environment.

SRL has been gaining traction in deep RL because of its ability to extract low-dimensional state space from a high-dimensional environment by reducing the complexity of the environment. Raffin *et al.* [16], argued that SRL improves the quality of an agent's learning by giving the agent a state space that ignores irrelevant distractors, and is disentangled in its feature representation. Kendall *et al.* [17], was one of the first researchers to apply deep reinforcement learning with SRL in autonomous driving. They first proposed a method of policy learning by simply mounting a single monocular image as input to observe the environment and compared it with using an online VAE. Using VAE gave a minor advantage over raw image data. However, training a VAE online i.e., simultaneously with a deep RL agent causes instabilities in the policy training [18]. We addressed this issue in our paper by opting to train the VAE offline by using a pre-trained VAE with fixed feature extraction. We build on these by demonstrating how compressing the raw input image from high-dimensional observation to low-dimensional latent space before training our deep RL agent, not only reduces the training time but also improves the quality of the agent. On top of that, we trained and evaluated our agents using DDPG, proximal policy optimization (PPO), and soft actor-critic (SAC) and compared our results with the deep RL baselines.

2. ALGORITHM

Deep RL is simply the use of deep neural networks as a function approximator for policy and value functions in a reinforcement learning framework. The neural network is represented by a policy $\pi(a|s;\theta)$ and a value function $Q(s;\theta)$. We applied 3 state-of-the-art model-free deep RL algorithms to our proposed framework to learn a driving policy. This section gives a brief introduction to these algorithms.

2.1. Deep deterministic policy gradient

DDPG is a kind of actor-critic algorithm similar to an approximate deep Q network. It uses two separate neural networks for function approximators [12]: an actor and a critic. The actor is used to approximate optimal policy deterministically i.e., it always tries to output the best-believed action for any given state. The actor $\mu(s;\theta_\mu)$ learns an $argmax_a Q(s,a)$ as the best action. The critic $Q(s,\mu(s;\theta_\mu;\ \theta_Q)$ learns to evaluate the optimal action-value function by using the actor's best-believed action. The algorithm samples a batch of transitions from an experience replay memory then updates the critic network using a target of:

$$y_t = r(s_t,a_t) + \gamma.Q(s_{t+1},\mu(s_{t+1})) \quad (1)$$

Meanwhile, the actor-network is updated using gradients of the output from the critic network:

$$\nabla_{\theta^\mu} J \approx E_{s_t\ \sim\rho^\beta}[\nabla_a Q(s,a)|_{s=s_t,a=\mu(s_t)} \cdot \nabla_{\theta^\mu}\mu(s)|_{s=s_t}] \quad (2)$$

where $\nabla_a Q(s,a)|_{s=s_t,a=\mu(s_t)}$ is the action mean value of the critic network. To get the mean action, the actor-network uses the current state as input. Then it computes the actor gradient output for the actor weights given as $\nabla_a Q(s,a)$. The actor gradient is used to send control commands to the agent.

2.2. Proximal policy optimization

PPO is a policy gradient-based model-free reinforcement learning algorithm that employs first-order trust-region criteria to prevent divergence [19]. It combines trust region policy optimization (TRPO) [20] with gradient descent to stabilize training by creating a loss function that nearly guarantees a simultaneous improvement of the agent's policy. PPO attempts to improve the "surrogate" objective from TRPO by clipping it into a certain region and further solving it with the first-order optimization using stochastic gradient descent (SGD) [21]. The clipped "surrogate" objective is defined as:

$$J^{CLIP}(\pi\theta) = \hat{E}_t\left[\left(rt(\theta)\hat{A}_t\right), clip(rt(\theta),1-\varepsilon,1+\varepsilon)\hat{A}_t)\right] \quad (3)$$

where the probability ration $rt(\theta) = \pi\theta(s_t)/\pi\theta_{old}(a_t|s_t)$, the advantage estimate $\hat{A}$ is estimated by GAE. Without the hard constraint of Kullback–Leibler (KL) divergence common in TRPO, the update rule becomes:

$$\theta = \theta_{old} + \alpha\nabla_\theta J^{CLIP}(\pi_\theta) \quad (4)$$

where ∝ is the learning rate. To avoid greedy updates, PPO uses the change in probability ratio only when it makes the objective worse. Thereby providing an alternative clipped surrogate objective using a penalty on KL divergence with an adaptive coefficient.

$$J^{KLPEN}(\pi_\theta) = E_{t\theta_{old}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t - \beta KL\left[\pi_{\theta_{old}}(\cdot|s_t, \pi_\theta(\cdot|s_t))\right]\right], \quad (5)$$

where θ_{old} is the policy parameters before the update. Similar to (4), the KL penalty version is solved by applying an SGD leading to the update rule.

$$\theta = \theta_{old} + \alpha\nabla_\theta J^{KLPEN}(\pi_\theta) \quad (6)$$

The computation for optimizing both versions of the objective function is much less than that of a TRPO. Therefore, the experimental results when evaluated in a MuJoCO environment, RoboSchool, and Atria games task shows that PPO performs better than TRPO.

2.3. Soft actor-critic

SAC is an off-policy algorithm that employs the best sample complexity and convergence property without the need for too many hyperparameter tuning. This makes its performance significantly better than the DDPG and PPO methods in challenging decision-making tasks. SAC attempts to balance an expected return by learning a stochastic policy π^*that maximize both the rewards and the entropy as:

$$\pi^* = \sum_{t=0}^{T} E(s_t, a_t) \sim \tau_\pi[\gamma^t(r(s_t, a_t) + \alpha H(\pi(s_t))] \quad (7)$$

where the entropy is $H\big(\pi(s_t)\big) = E_{a\sim\pi(.|s)}[-log\ (\pi(s)]$. In the most actor-critic algorithm, we define the optimal policy in respect to the reward, however, SAC changes that traditional objective of the Markov decision process, by introducing a new value function called the soft value function defined as:

$$V(s_t) = E_{at\sim\pi}[Q(s_t, a_t) - \alpha\ log\ log\ (\pi(s_t))\ ] \quad (8)$$

where Q_θ is the soft Q-value function. To find an optimal policy, the critic has to convert into an optimal critic by using a bellman operator T^π such that:

$$T^\pi Q(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_t+1\sim p(s_t,a_t)}[V(s_{t+1})] \quad (9)$$

Given the established bellman operator, it tries to minimize the expectation over transitions of the specific temporal difference error. This process is called the policy evaluation step: where the critic parameters are learned by minimizing:

$$J(\theta) = E_{(st,at,st+1)\sim D}\left[\left(Q_\theta(s_t, a_t) + \gamma V_\theta(s_{t+1})\right)^2\right] \quad (10)$$

3. RESEARCH METHOD

3.1. Environment setup

The environment of the experiment is developed on car learning to act (CARLA) [22], an urban autonomous driving simulator built on an open-source protocol and implemented in an unreal engine 4 (UE4). It provides a variety of urban layouts, vehicle models, pedestrians, and road obstacles. making it possible for our agent to learn to drive in a 'realistic' urban traffic simulation environment. The simulation supports different sensors suits including cameras, raw images, depth maps, semantic segmentation images, and lidars. We concentrated on the raw pixel data from CARLA's built-in front-facing camera. And the VAE was trained for better state representation.

3.2. Continuous action space

Since model-free deep RL are continuous action algorithms, our deep RL agent acquired feedback during training from the environment by continuous action values such as steering, throttle, and brake. These are controlled by three different value action in CARLA:

– Steering [-1, 1]: controlled by a positive value that shows to steer right, while a negative value indicates to steer left.

– Throttle [-1, 1]: controlled by a positive value that shows forward acceleration, while a negative value indicates backward acceleration.
– Brake [0, 1]: a value of 1 means to apply the brake while a value of 0 means to prevent braking.

Similar to Kendall, we reduced the action space by removing the brake action. The vehicle will drive at a low speed with no other vehicle present and with no consideration of traffic rules eliminating the need for braking. That leaves us with a steering and throttle control vector.

3.3. Latent state space

As we intend to reduce the complexity of the environment from high dimensional state space to low dimensional state space, we gathered datasets needed to train the VAE by manually driving around the CARLA environment with a keyboard, collecting images from the front-facing camera. The vehicle does not follow any traffic rules of imitation learning. Also, there is no need for labeling such as the ground truth position of the vehicle in the training dataset with exception of the raw pixel data, which is essential. We collected 10,000 images (approximately 15 minutes of driving).

A VAE model to reconstruct the collected input image from the sampled gaussian latent vector and minimize the reconstruction loss was created. The model contains an encoder and a decoder with four 2D convolution layers (conv2d) with 32, 64, 128, and 256 separate channels. Each conv2d has two (2) strides sets and a ReLu activation function. As well as the latent space layer that is fully connected to the last conv2d with a size of 64, we also used Adam optimizer [21], with a learning rate of 10^{-4} and a batch size of 100. A KL-tolerance factor was applied to ensure the optimizer only applies a KL-loss once the KL-loss exceeds a tolerance factor greater than zero. In Figure 2, we explored how the encoder, q, takes raw image input x and outputs two vectors, z_μ and z_σ, through a parallel fully connected layer. Then we used sample z from $N(z_\mu, z_\sigma)$ and passed through a decoder p, producing a reconstructed signal $\hat{x}$.

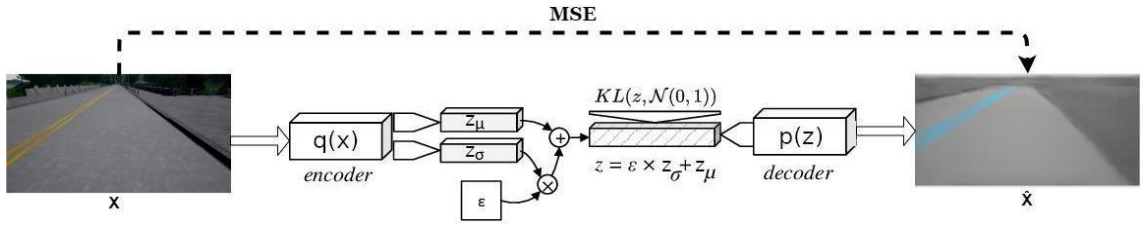

Figure 2. The architecture of our VAE

3.4. Network architecture

We set all networks with the same convolutional layers while the first dense layer is similar to our VAE model. To ensure the distribution of features is mobile, the policy was trained with a fixed feature extractor. The network architecture of the three deep RL algorithms (DDPG, PPO, and SAC) used in this work was adopted from implementing Stable Baseline by Hill *et al.* 2019 [23]. The implementation was altered by adding the optimization for the VAE in every episode. We found the following tuned hyper-parameters as the most effective in the experiment.

– DDPG: Defined a discount factor of 0.99, a gradient clipping of 0.005, the replay memory size of 10000, optimization steps of 300 between episodes with batch size 64, and an Ornstein-Uhlenbeck process [24] with $\theta = 0.15$ and $\sigma = 0.2$ inspired by the work of Lillicrap *et al.* [12].
– PPO: A multi-layer perceptron with a discount factor of 0.99, an entropy loss of 0.0, an optimization step of 256, and a mini-batch size of 4 with a learning rate of 3e-3 for 10 epochs.
– SAC: A buffer size of 30000 with an optimization step of 600 and batch size of 64. We set the learning rate to 3e-4 with gamma 0.99. SAC does not need an exploration policy; it is a component learned during training.

3.5. Reward function

As a baseline, we aimed to train our agent with a reward function similar to Kendall's. In their work, they simply defined a reward function that is proportional to the acceleration of the vehicle:

$$r(v) = \{0\ v\ \frac{v>v_{target}\ or\ on\ infraction}{otherwise} \quad (11)$$

where v is the current speed of the vehicle in km/h, v_{target} is the target speed (the recommended speed required for the agent to maximize reward), and an infraction when the agent gets off the track or collides with an obstacle. To prevent the agent from getting stuck in a bad state due to a well-known bug in CARAL, where collisions are not annotated when the vehicle is moving at a low speed of 1 km/h or less [25], we added a live penalty to punish the agent each time it gets stuck without moving or crashing at a low forward speed.

4. RESULTS AND DISCUSSION

To evaluate the effect better state representation has on policy learning, we trained a DDPG agent using raw pixel (baseline) and compared the result with our DDPG agent trained with a VAE (DDPG+ VAE). Similarly, we trained a SAC and PPO agent with pre-trained autoencoders and compared the result with our DDPG+ VAE agent. Finally, we evaluated our proposed method based on the cumulative reward obtained over 100 training episodes using the reward function described in section 3.5.

4.1. Comparing the baseline with our DDPG+ VAE agent

We established the baseline as a DDPG agent, which is trained directly from pixel data obtained from the vehicle's front camera. We selected the parameters of the baseline model similar to Kendall's implementation, whereas our DDPG+ VAE is the agent pre-trained with VAE as discussed in section 3.3.

Figure 3 shows how the baseline fell short in performance as compared with our DDPG+ VAE agent. The baseline agent could barely learn any decent lane following skills after 100 training episodes. It keeps turning right and then goes off lane at most episodes, while only managing an average cumulative reward of 128.31. Nonetheless, our pre-trained DDPG+ VAE agent was able to learn good lane following skills after only six (6) training episodes and having an average cumulative reward of 276.13. It even learned how to take turns after a couple more episodes. This proves that our DDPG+ VAE agent not only learns faster than the baseline but also improves the overall quality of the deep RL agent.

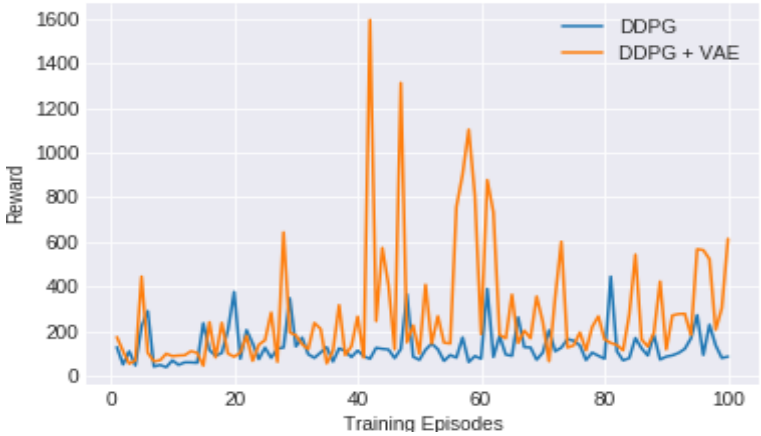

Figure 3. Comparing the results of the baseline agent with our DDPG agent with pre-trained VAE

4.2. Comparing our DDPG+ VAE agent with other model-free deep RL algorithm

To ensure the best algorithm for our method, we evaluated the effect of SRL on other deep RL algorithms in autonomous driving. We compared the performance of our proposed DDPG agent to other deep RL baselines with VAE. We trained both SAC and PPO agents with pre-trained VAE for 100 episodes and compared the results with our DDPG implementation. As observed from Figure 4, the SAC agent outperformed the other two agents by a large margin. It learned a good lane following skill on its first few episodes while being able to successfully drive around the town without any collisions accumulating an astonishing average cumulative reward of 631. Meanwhile, PPO and DDPG agents were only able to obtain an average cumulative reward of 276 and 246 responsively. We attribute the preeminence of the SAC algorithm to its ability to learn an exploration policy during training. As a result, we recommend SAC with pre-trained VAE as an alternative in speeding up deep RL training in autonomous driving. We have compiled videos showcasing the results of our experiments. This video can be found at https://youtu.be/kriYuE5z44M.

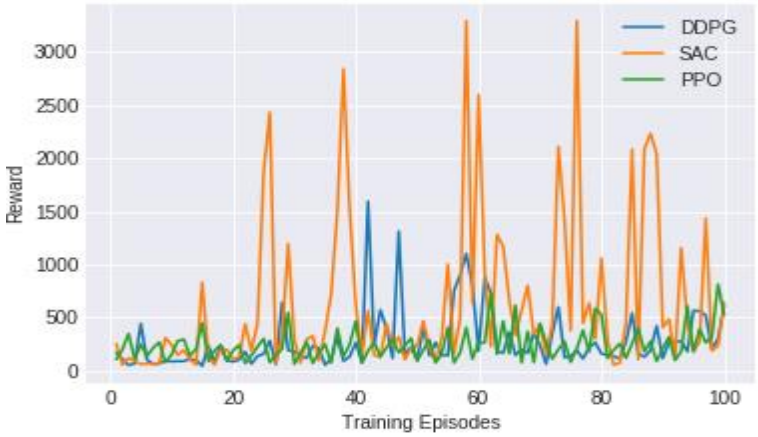

Figure 4. This shows that SAC has the best performance among all methods

5. CONCLUSION

In this paper, we explored how several aspects of deep reinforcement learning can solve a simple lane following autonomous driving tasks. From the methods used, we suggest that VAEs can accelerate the training of deep RL agents and how state representation affects the overall performance of the deep RL agent autonomous driving. This method exhibits that it is not only limited to 3D environments that are visually simple such as Prakash's Donkey Car simulator and Kendall's Custom in-house simulator but also efficient in a "realistic" urban driving simulator like CARLA. Although there were some challenges in setting up the final environment on CARLA, we have conclusively created a self-contained deep RL system that works alongside CARLA. We have shown that employing a pre-trained VAE helps improve training a deep RL agent as compared to training raw pixel data directly. It was also showcased how the VAE affects the overall result of the agent, and that a pre-trained VAE performs much better than optimizing the VAE the deep RL agent in alternate training phases. We also concur with the work of Raffin that demonstrates a more recent reinforcement learning algorithm, SAC, can work better than Kandell's DDPG in autonomous driving.

REFERENCES

[1] SAE J3016, "Surface Vehicle Recommended Practice Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," 2021.
[2] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, 2016, doi: 10.1109/TITS.2015.2498841.
[3] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, 2016, doi: 10.1109/TIV.2016.2578706.
[4] M. Bojarski, *et al.*, "End to End Learning for Self-Driving Cars," 2016, Accessed: Apr. 09, 2021. [Online]. Available: http://arxiv.org/abs/1604.07316.
[5] F. Codevilla, M. Miiller, A. Lopez, V. Koltun, and A. Dosovitskiy, "End-to-End Driving Via Conditional Imitation Learning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2018, pp. 4693–4700, doi: 10.1109/ICRA.2018.8460487.
[6] D. Silver, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, doi: 10.1038/nature16961.
[7] V. Mnih, *et al.*, "Playing Atari with Deep Reinforcement Learning," 2013, Accessed: Apr. 09, 2021. [Online]. Available: http://arxiv.org/abs/1312.5602.
[8] M. Vitelli, "CARMA : A Deep Reinforcement Learning Approach to Autonomous Driving," pp. 1–8, 2017.
[9] V. Mnih, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.
[10] V. Mnih, *et al.*, "Asynchronous methods for deep reinforcement learning," in *33rd International Conference on Machine Learning, ICML 2016*, 2016, vol. 4, pp. 2850–2869.
[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.
[12] T. P. Lillicrap, *et al.*, "Continuous control with deep reinforcement learning," 2016, Accessed: Jun. 23, 2021. [Online]. Available: https://goo.gl/J4PIAz.
[13] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to Real Reinforcement Learning for Autonomous Driving," 2017.
[14] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," in *IS and T International Symposium on Electronic Imaging Science and Technology*, 2017, pp. 70–76,

doi: 10.2352/ISSN.2470-1173.2017.19.AVM-023.
[15] B. Prakash, M. Horton, N. R. Waytowich, W. D. Hairston, T. Oates, and T. Mohsenin, "On the use of deep autoencoders for efficient embedded reinforcement learning," in *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI*, 2019, pp. 507–512, doi: 10.1145/3299874.3319493.
[16] A. Raffin, A. Hill, R. Traoré, T. Lesort, N. Díaz-Rodríguez, and D. Filliat, "Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics," 2019, Accessed: Jun. 23, 2021. [Online]. Available: http://flowers.inria.fr.
[17] A. Kendall, *et al.*, "Learning to drive in a day," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2019, vol. 2019-May, pp. 8248–8254, doi: 10.1109/ICRA.2019.8793742.
[18] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving Sample Efficiency in Model-Free Reinforcement Learning from Images," 2019, Accessed: Jun. 23, 2021. [Online]. Available: https://sites.google.com/view/sac-ae/home.
[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, "Proximal Policy Optimization Algorithms."
[20] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," 2015.
[21] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," 2015.
[22] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA: An open urban driving simulator," *arXiv*. 2017.
[23] A. Hill, "Stable Baselines," *github.com*, 2019. https://github.com/hill-a/stable-baselines (accessed Jun. 23, 2021).
[24] E. Bibbona, G. Panfilo, and P. Tavella, "The Ornstein-Uhlenbeck process as a model of a low pass filtered white noise," *Metrologia*, vol. 45, no. 6, 2008, doi: 10.1088/0026-1394/45/6/S17.
[25] CARLA, "Collisions are not annotated when vehicle's speed is low • Issue #13 • carla-simulator/Carla," *github.com*, 2017. https://github.com/carla-simulator/carla/issues/13.

BIOGRAPHIES OF AUTHORS

Emmanuel Ifeanyi Iroegbu is a graduate student in Computer Science at Mapua University, Manila. His research interest includes Deep Learning, Reinforcement Learning, Data Analysis, and Machine Learning.

Madhavi Devaraj has done a Ph.D. in Computer Science. She works as a distinguished professor at Mapua University, Manila. Her research interest includes Deep Learning, NLP, Scientometric Analysis, Reinforcement Learning, and Blockchain.