Study of the First Application of Deep Reinforcement Learning to Autonomous Driving

TEAM NUMBER - 8
GAO RUIFENG, A0213777X, TG2(CS5446)
HU WEILIN A0195438R , TG2(CS5446)
TAN HUI MIN A0225203X, TG2(CS5446)

November 22, 2021

1 Introduction

Wayve [1] is a pioneering artificial intelligence software company for self-driving cars. Their unique end-to-end machine learning approach learns to drive in new places more efficiently than competing technology. In this exposition project, we studied Wayve's first application of deep reinforcement learning onboard an autonomous car [2], which achieves the first real world run with Deep Q Networks (DQN) in a countryside road without traffic.

2 Autonomous Driving System

2.1 Introduction to Auto-driving System

The modern autonomous driving industry originated from DARPA Urban Challenge in 2007. This event required teams to build an autonomous vehicle capable of driving in traffic, performing complex maneuvers such as merging, passing, parking, and negotiating intersections. Six robotic vehicles eventually crossed the finish line which have proven to the world that autonomous urban driving could become a reality. This event was groundbreaking as the first time autonomous vehicles have interacted with both manned and unmanned vehicle traffic in an urban environment.

Since then, we have seen rapid growth of ADS research both in academic and industrial settings. Common practices in system architecture have been established over the years. Most of the ADSs divide the massive task of automated driving into subcategories and employ an array of sensors and algorithms on various modules. More recently, end-to-end driving started to emerge as an alternative to modular approaches. Deep learning models have become dominant in many of these tasks. Then, we will introduce two main algorithmic designs for the auto-driving system, namely the modular system and the end-to-end driving system[3], respectively.

2.2 Modular Auto-driving System

Modular systems, which are widely used in the real-world auto-driving systems, are structured as a pipeline of separate components linking sensory inputs to actuator outputs. Figure 1 gives an graph illustration of modular systems. Those components include localization and mapping, perception, assessment, planning and decision making, vehicle control, and human-machine interface etc. For each module, there are a lot of algorithms and methods developed in the research areas of robotics, computer vision and vehicle dynamics etc. However, the major disadvantages of modular systems are being prone to error propagation and over-complexity.

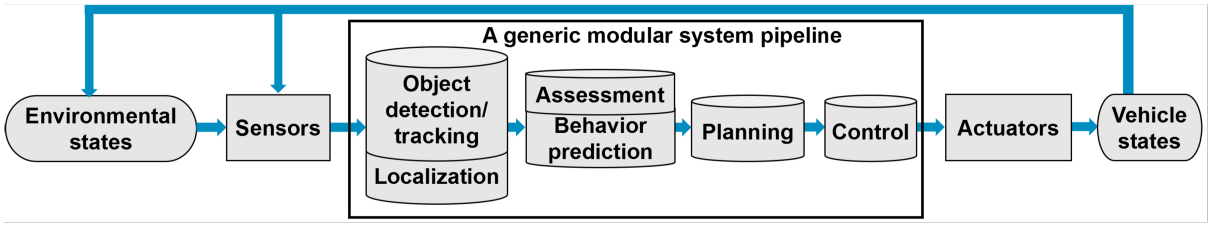

Figure 1: Modular Auto-driving System

2.3 End-to-end Auto-driving System

End-to-end driving system depicted in Figure 2, however, is a promising next-generation auto-driving approach, which will generate ego-motion like steering wheel and pedals directly from sensory inputs. Ego-motion can be either the continuous operation of steering wheel and pedals or a discrete set of actions, e.g, acceleration and turning left. There are three main approaches for end-to-end driving systems: direct supervised deep learning, neuroevolution and the more recent deep reinforcement learning. In this project, Wayve leverages a novel deep reinforcement learning model called Deep Q Networks (DQN) to build their autonomous driving system, which learns to lane follow from scratch with just 10 minutes of feedback.

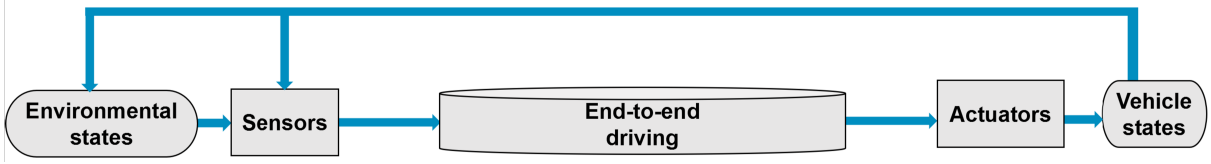

Figure 2: End-to-end Auto-driving System

3 Problem Formulation - Driving as a Markov Decision Process

3.1 MDP Formulation

To solve auto-driving problem with reinforcement learning algorihtms, the auto driving problem is formulated as a Markov Decision Process(MDP), which requires definitions for States s_t, Actions a_t, and Rewards $R(s_t, s_{t+1}, a_t)$.

State s_t: The state at time t is a combination of observation from monocular camera image, vehicle speed and steering angle. For the image, the paper trains VAE online from five random exploration episodes using a KL loss and a LE reconstructruction loss. Theoretically, state s_t is to be a Markov representation of all previous observations. While in this project, it is sufficient to use only the sensor observations from time t for the simple driving task.

Action a_t: This project applies a two-dimensional action space, with steering angle in the range of [-1,1] and speed setpoint in km/h.

Reward $R(s_t, s_{t+1}, a_t)$: The reward in the formulated MDP is defined as the average distance the vehicle has traveled before an infraction of traffic rules occurs. Intuitively, the longer the vehicle stays on track, the higher the reward value will be.

3.2 Approximating States - VAE

The project applies Variational Auto-Encoder(VAE) [4] to gain better approximation of the image inputs. As illustrated in Figure 3, the VAE consists of an encoder network and a decoder network. The encoder network learns a low-dimensional representation z of the original input image x, and the decoder reconstructs a "fake" input $\hat{x}$ from representation z. In the training process of the auto-driving system,

the VAE is also trained to minimize the loss between $\hat{x}$ and x so that the encoder can yield reliable representation of the original input.In this project, the VAE is trained to obtain state vector from input images from monocular camera on the vehicle.

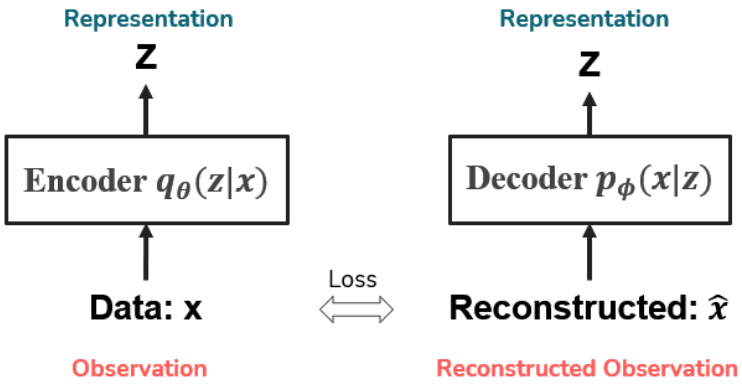

Figure 3: Variational Auto-Encoder illustration

3.3 Discrete VS Continuous

By definition of actions in Section 3.1, the action space in this formulation is continuous, where the steering angle is in the range of [-1,1] and speed setpoint in km/h.

Value-based reinforcement learning algorithms, such as the classical Deep Q-network, are not capable of handling problems with continuous action space due to the curse of dimensionality.

Although policy-based reinforcement learning methods is suitable for continuous action space, the training of these models can be very slow as it updates on episodes.

In this project, the Deep Deterministic Policy Gradient algorithm (DDPG) is applied to solve the auto-driving MDP. DDPG is a combination of both value-based and policy-based is thus a better choice, which can handle continuous action space and can be updated at each time step. DDPG will be further discussed in detail in Section 4.

4 Deep Deterministic Policy Gradient

4.1 Introduction to DDPG

DDPG[5] is a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces. It draws its roots from Deep Q Network (DQN) and Deterministic Policy Gradient (DPG).

In DQN, deep neural network function approximators were used to estimate the action-value function, the update equation is

$$\theta_i \leftarrow \theta_i + \alpha[R(s,a,s') + \gamma \max_a \hat{Q}_\theta(s',a') - \hat{Q}_\theta(s,a)] \frac{\partial \hat{Q}_\theta(s,a)}{\partial \theta_i}$$

where Q_θ is a neural network parameterized by θ. DQN can only handle discrete and low-dimensional action spaces, because it rely on finding the action that maximizes the action-value function, which in the continuous action space require an optimization of a_t every timestamp.

DPG, compared to traditional Policy Gradient methods, maintains a parameterized actor function which specifies the current policy by deterministically mapping states to a specific action rather than random action being selected based on certain action distribution in policy gradient. Actor is updated based on the policy gradient, i.e. the gradient of the policy's performance.

To learn policies in high-dimensional and continuous action spaces, DDPG introduce an actor-critic approach based on DPG algorithm with deep function approximators inspired by DQN.

4.2 Techniques Applied in DDPG

Replay buffer. Optimization algorithm for neural networks assume that the samples are independently and identically distributed (i.i.d), which doesn't hold when samples are generated from exploring sequentially in an environment. To overcome this, DDPG introduces a technique called replay buffer which is adopted from DQN. Actor and critic are updated by sampling a minibatch uniformly from the reply buffer.

"Soft" update target network. The Q network being updated is also used in calculating the target value, the Q update is prone to divergence. Inspired by DQN, DDPG applied "soft update" where the target network of both critic and actor was updated by having them slowly track the learned networks. This technique constrains the target values to change slowly, greatly improving the stability of learning.

Allow more exploration in action space. The trade-off of exploration and exploitation is always an important consideration in reinforcement learning algorithm. DDPG is off-policy algorithm which can treat the problem of exploration independently from the learning algorithm. To encourage more exploration, DDPG adds noise sampled from a noise process to the "best" action given by policy network and uses the policy with random disturbance as the exploration policy to be performed in environment.

4.3 Graph Illustration of DDPG

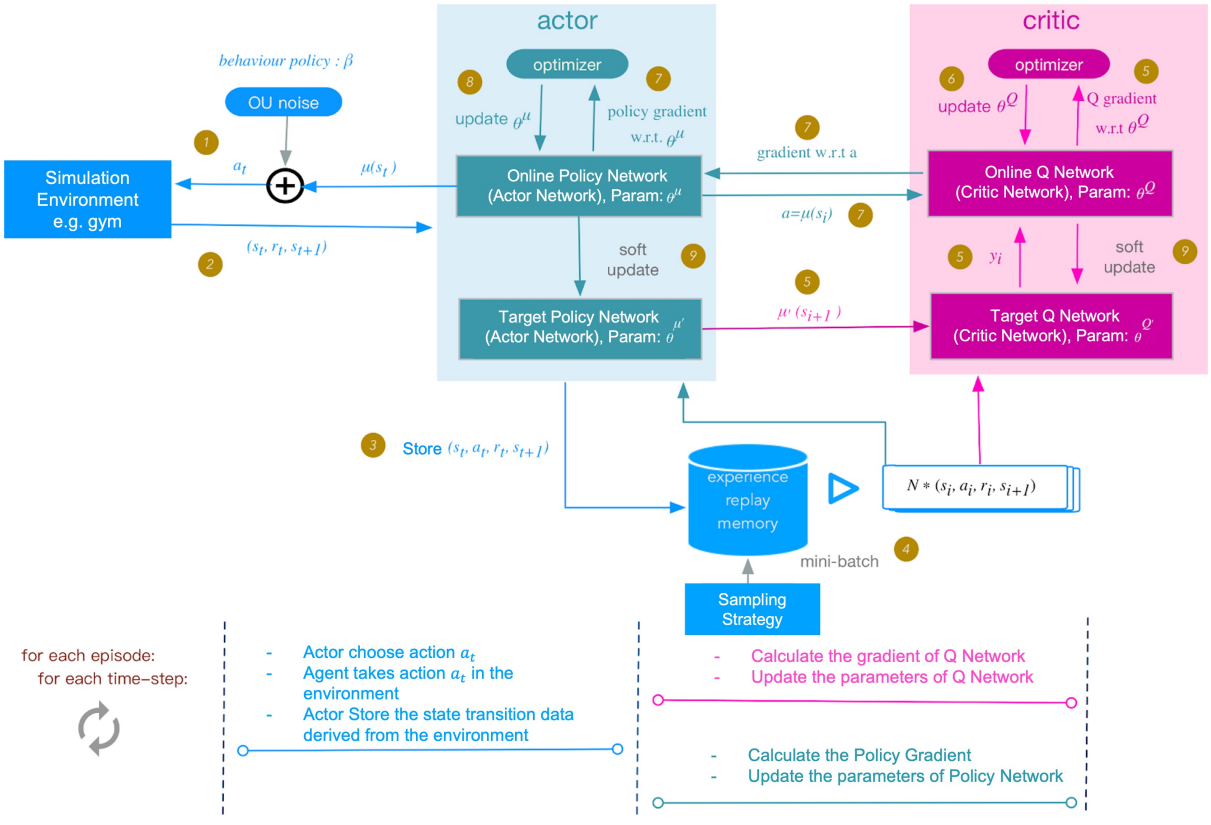

Figure 4: Graph illustration of DDPG

Figure 4 is the graph illustration of DDPG. Exploration policy a_t is generated by adding up the actor's "best" action $\mu(s_t)$ and the noise β from the Ornstein-Ulenbeck Noise Process. Agent executes the exploration policy a_t in the simulation environment, receives reward r_t and senses new state s_{t+1}. Then the tuple (s_t, a_t, r_t, s_{t+1}) will be stored in the experience replay buffer. A mini-batch is sampled with

some sampling strategies to update online critic network first by minimizing the loss between Q value given by target Q network and online Q network:

$$L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$$
$$\text{where } y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu\prime})|\theta^{Q\prime})$$

Then actor policy network is updated by using sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

After Online Q Network and Policy network has been updated, the target network of both will be "soft" updated by:

$$\theta^{Q\prime} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q\prime}$$
$$\theta^{\mu\prime} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu\prime}$$

4.4 DDPG in Auto-driving Systems

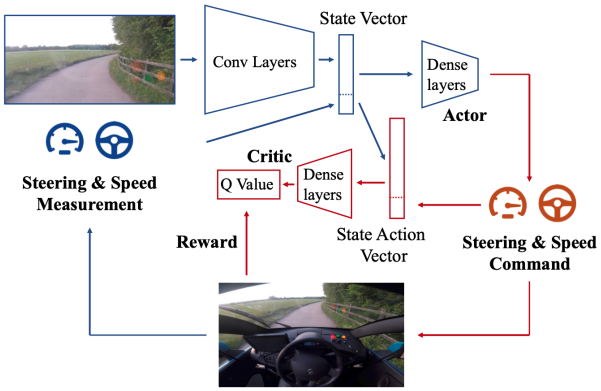

Figure 5: DDPG application in real auto-driving system

Figure 5 is a high-level illustration of the structure of the DDPG auto-driving system implementation in this project. The environment feeds real-time camera image, steering angle, and speed to the model as inputs, where the image is pre-processed by VAE convolutional layers and then combined with steering angle and speed value to construct a state vector. The state vector is then sent to both actor network and critic network. The actor network takes state as input and yield action accordingly to control the vehicle. The critic network evaluates this action based on current state and environment reward.After taking action and evaluation, the model takes another state vector from the environment and starts the next step.

5 Experiments and Demo

5.1 Experiment Setup

In order to verify the usefulness of the proposed solution mentioned in the paper, an experiment was conducted with reference to the DDPG algorithms with the parameters suggested by the author. However, instead of building the entire application from scratch, the experiment was conducted by making changes on top of a boilerplate git repository which already contains the base setup referencing the approach mentioned in the paper. However, as the boilerplate repository was created few years back, thus some dependencies and code changes were made in order to get the entire process to work.

The following prerequisite will be needed in order to have the application running in the machine:

- Linux OS
- Docker
- Compiled Donkey Car Simulator

After having all the prerequisite fulfill, the application can be build by running the docker command. The following are the libraries that are used in this project.

- Donkey gym
- Stable baselines
- Tensorflow
- Pandas
- NumPys

The command below is used to build the application:

```
docker build -t learning-to-drive-in-a-day .
```

Next, the application can then be started by running the following command for both training and testing phase.

```
sudo ./run-in-docker.sh
```

5.2 Algorithm Hyperparameters

Based on the research paper, the main hyperparameters that are concluded to be effective is to have future discount factor of 0.9, batch size of 64 and gradient clipping of 0.005 as shown in Table 1. Apart from these parameters, the noise in action is also added to encourage the agent to explore further instead of having pure exploitation to maximize the rewards. As this action space is continuous, instead of using epsilon greedy to encourage the agent via weightage, the Ornstein-Ulenbeck Process which uses the mean-reversion equation is used to generate the half life noise with θ of 0.6 and σ of 0.4. The parameters used for the noise can be found in Table 2.

Parameter	Value
Batch Size	64
Gradient clipping	0.005
Gamma	0.9
Memory limit	10000
Nb train step	3000

Table 1: DDPG Parameters

Parameters	Value
Theta	0.6
Sigma	0.4

Table 2: Ornstein-Ulenbeck Noise Parameters

5.3 Comparing Real Life Driving Result with Simulation

To evaluate whether the result mentioned in the paper is consistent based on the suggested parameters, few experiments were conducted to monitor the trends of the rewards as more episodes were run. On top of that, two approaches were implemented in this experiment as well to see whether the addition of VAE would be helpful in providing better state representation to the agent. Note that in this experiment, instead of the physical distance which could be measured in real life driving, a general reward function is used to compute the maximum rewards for each episode from the starting point till the donkey car is being detected to be out of the track.

Figure 6 is the result extracted from the research paper and from this figure, it has shown the result when using DDPG with VAE is much more effective than the one without VAE. Thus, the author

suggests that with the addition of VAE, it might be useful in getting quicker and more efficient result due to an improved state representation from the previous approach. After doing the training with the agent using the simulation, the result derived are as shown in Figure 7. From the figure, the rewards progression without VAE doesn't seem to have any improvement. But for the reward progression with VAE, there is an slight upward trends in the increasing rewards which is aligned to the finding mentioned in the paper.

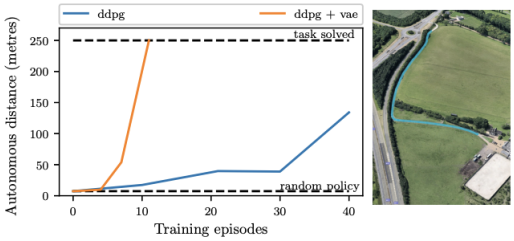

Figure 6: Result extracted from the research paper

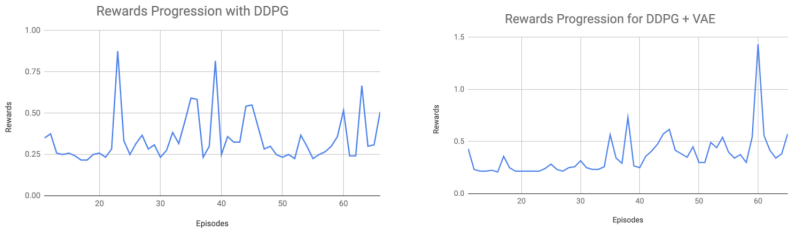

Figure 7: Rewards progression with DDPG and DDPG + VAE

Putting all the finding mentioned above into the following tables shown in Table 3 and Table 4 ,it does show some resemblance in the result, illustrating that the reinforcement learning does help to guide the donkey car agent to get better with the road. Moreover, with a better state representation given to the agent, it plays an important role in improving the agent's navigation decision as it slowly learns through trial and error.

For more information on the implementation, the code and the result can be found via this git repository.

	Training			Test	
Model	Episodes	Distance	Time	Meters per Disengagement	# Disengagements
Deep RL from Pixels	35	298.8m	37 min	143.2	1
Deep RL from Pixels	11	195.5m	15 min	-	0 height

Table 3: Extracted table from research paper

		Training			**Test**
Model		**Episodes**	**Average Rewards**	**Average Time**	**Average Rewards**
Deep RL from Pixels	Self trained model	60	0.403	41.638	0.511
	Provided model	-	-	-	0.526
Deep RL from VAE	Self trained model	83	0.298	24.8	0.758
	Provided model	-	-	-	2.746

Table 4: Tabulated result from the experiment

6 Challenges and Conclusion

When comes to autonomous driving, it involves several areas of research work to make it fully functional. In this project, the scope is narrowed down to focus only the navigation part of self-driving. Despite so, from the paper and the experiment conducted, we could see that the car agent does learn how to drive gradually using reinforcement learning. But there are still many areas of work needed to be addressed in this discussed topic such as improving the image interpretation or to have a specific reward function tag to a navigation goal instead of a general one.

In addition, DDPG has one major disadvantage over direct supervised learning is that it needs time to interact directly with the environment and have some fail attempts before it could learn the desired behavior. Therefore, other than the technical constraints such as needing the agent to learn it quickly in various context, there would be some moral concerns on whether we can tolerate the driving agent to make mistakes and how do we define the reward function in tricky situations illustrated in the moral machine experiments. After all, the mistakes it could make in real world scenarios might be disastrous and impossible to place a value to quantify nor justify for its navigation decision.

All in all, there are definitely many challenges ahead before autonomous driving can be implemented and accepted in the real world. While the example in this paper may seem to be too simplicity, it does provide a good start to advocate exploration of reinforcement learning in end-to-end driving which might open out to more alternatives to train the agent faster in a much shorter time frame.

References

[1] Wayve. Read about wayve's latest news and progress. https://wayve.ai/blog/research/, 2021. Last accessed 22 November 2021.

[2] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[3] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.