Machine Learning for Games: Car Parking Agent

Keerthana Nandanavanam
Viterbi School of Engineering
University of Southern California
Los Angeles, CA
nandanav@usc.edu

Krishna Manoj Maddipatla
Viterbi School of Engineering
University of Southern California
Los Angeles, CA
km69564@usc.edu

Nidhi Chaudhary
Viterbi School of Engineering
University of Southern California
Los Angeles, CA
nidhicha@usc.edu

Sumanth Mothkuri
Viterbi School of Engineering
University of Southern California
Los Angeles, CA
mothkuri@usc.edu

***Abstract*—Machine Learning has been actively used these days to make our lives easier. One such application lies in self-driving cars. But these cars have to be equipped with a robust parking system to identify the parking spot and seamlessly navigate to it overcoming the obstacles bestowed in its path. The purpose of this paper is to summarize the research we have done to mirror this use case. We will explore Proximal Policy Optimization algorithm, which is a powerful reinforcement learning algorithm as well as Imitation Learning to make our car agent park correctly at the highlighted spot and compare the results. We used an open source game in Unity as our simulation environment.**

***Index Terms*—Machine Learning, Reinforcement Learning, Unity, Proximal Policy Optimization, Imitation Learning, Generative Adversarial Imitation Learning**

I. Introduction

Self driving/autonomous cars are a big sensation in recent times. Tesla, Waymo, BMW and many other mega corporations are now actively investing in this trend. Out of the many design considerations for such an autonomous vehicle, having a good parking system that helps in navigating through obstacles, identifying the right spot and parking the vehicle is of paramount importance.

We drew inspiration from this exact problem and strived to develop a machine learning agent capable of doing the same using reinforcement learning. The fact that this technology can also be used for vacuum cleaners which can navigate through household items and clean the surface thoroughly has strengthened our motivation to work towards this project. Generally, it could be used for any obstacle avoidance and navigation system and hence it can have multiple applications from medicine to defense industry.

II. Related Works

A. Obstacle Avoidance and Navigation Systems

There has been significant research in the field of obstacle avoidance and navigation systems using reinforcement learning[1] and since this problem is the parent problem of our use case, we decided to experiment further on the research done in this domain. Proximal Policy Optimization (PPO) algorithm was found to be showing great results for the problem. So, we also drew inspiration from it, and incorporated PPO as our base algorithm.

B. Parking occupancy detection using CNN

The paper[2] describes parking occupancy detection systems using Convolutional Neural Networks (CNN) and Support Vector Machines (SVM). The classifier was trained and tested by the features learned by the deep CNN from public datasets (PKLot) having different illuminance and weather conditions.

C. Autonomous Vehicle Control using Reinforcement Learning

A lot of promising research has been done using reinforcement learning for strategic decision making[3]. The autonomous exploration of a parking lot is simulated and the controls of the vehicles are learned via deep reinforcement learning[4]. A neural network agent is trained to map its estimated state to acceleration and steering commands to reach a specific target navigating through the obstacle course. Training was performed by a proximal policy optimization method with the policy being defined as a neural network. This paper also motivated us to look at PPO as our base algorithm.

D. Policy Gradient based Reinforcement Learning

A policy gradient based reinforcement learning approaches for self driving cars in a simulated highway environment has been implemented and tested. The research showed that reinforcement learning is a strong tool for designing complex behavior on traffic situations, such as highway driving, where multiple objectives are present, such as lane keeping, keeping right, avoiding incidents while maintaining a target speed[5].

III. Environment

Unity is one of the most popular game development engines that provides built in features like physics, 3D rendering, collision detection without having to reinvent the wheel for developing a game. One of the main reasons for choosing this platform is its support for the "*mlagents*" package. *mlagents* package provides implementations (based on PyTorch) of state-of-the-art algorithms to enable game developers to easily train intelligent agents for 2D, 3D and VR/AR games[6]. Another important reason is that Unity is a cross platform engine meaning it can be used on a machine with any operating system (OS) like Microsoft Windows, Linux OS and Mac OS.

Picking the right game is of utmost importance. The game should be as close as possible to the real world scenario of parking a vehicle. Many factors have been taken into consideration before choosing the game like its complexity, hardware requirements of the machine where the agent is trained, installation requirements and the features that make the game closer to the real world setup. After looking at numerous options, the open source game [7] designed in the Unity environment was selected.

A. Game Description

The game has 2 levels of which level 1 consists of a bounded arena with a car starting at an arbitrary position and a parking spot appearing at another random position. The goal location or parking spot is highlighted in red color. The car has to first identify that highlighted spot and then navigate towards it through three obstacles that are placed in the center of the arena.

Level 2 of the game also consists of a bounded arena similar to level 1 but with moving obstacles and storeys. The car will start at an arbitrary position and a parking spot will appear either on the same storey or a path will be highlighted to another storey. The car has to identify the respective highlighted parking spot or storey entrance and navigate through moving obstacles in the arena to reach the parking spot.

Out of the rich set of algorithms provided by *mlagents* package, two algorithms, Proximal Policy Optimization and Imitation Learning using Generative Adversarial Imitation Learning (GAIL) were used in training the agent.

B. Game Modifications

On top of the open source game, a few modifications have been made before creating an agent. Firstly, a scoreboard has been added that displays the parking score, obstacle hit score, wall hit score and the cumulative reward. The parking score represents the number of times the agent parks the car in the designated spot. Obstacle hit score and wall hit score denotes the number of times the agent hits the obstacles and walls respectively and the cumulative reward represents the total reward accumulated for each episode of the game. Second, the boundaries and walls of the arena have been converted to collision objects. When an agent tries to park the car and hits any of the collision objects, the episode ends. After an episode ends, the agents starts again at the previous start location and a new episode begins. Third, the game was initially developed as a touch screen game. There were controls present on the screen and the user is expected to press the controls for changing the direction of the car. This behavior has been modified to use keyboard controls for navigation as it is easier to train an agent with keyboard controls rather than touch screen controls. Lastly, from a predefined set of parking spots, one random parking spot is chosen for each episode from the available set of parking spots and assigned for the agent to park.

IV. Proposed Methods

In this section, the machine learning algorithms used to train the agent to overcome obstacles and park at the highlighted spot have been put forward. To improve the inferences made by the agent, state-of-the-art methods were introduced where the agent mimics human behaviour and tries to learn the best policy through imitation learning. We further discuss the reward systems that worked best for each of the algorithms used for training.

A. Reinforcement learning

Neural Networks have shown a great potential for decision making in game playing agents. However, a simple Neural Network is a supervised ML. It takes an input which is propagated through the layers of the network and produces an output, which is then compared with the actual label and the errors are back-propagated till the network converges. Now, in supervised learning, it's difficult to get the training data. A human player will have to play for multiple hours and data frames will have to be generated from the games played to be fed to the system. Since this is a very tedious, time consuming and error prone task, we decided to move ahead with reinforcement learning.

Reinforcement learning[8] is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs and tries to maximize the total reward[9]. Typically, the RL agent takes an action at following a policy π based on the observation of the state s_t and reward r_t at time t. Since the action at is applied in the environment by the agent, the new state changes to s_{t+1} and a reward r_{t+1} is assigned to the agent.

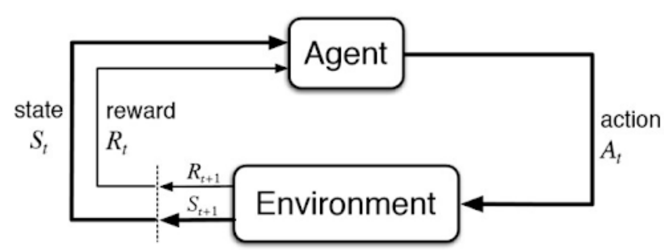

Fig. 1. Reinforcement Learning

One such class of reinforcement learning is *Proximal Policy Optimization* which learns online unlike experience replay by Deep Q-Networks. It strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small[10]. PPO tries to minimize the following objective function:

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

- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\hat{A}_t$ is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

B. Imitation Learning

Given a set of demonstrations or a demonstrator, the goal of imitation learning (IL) is to train a policy to mimic the demonstrations. Imitation Learning is usually the preferred algorithm when it is easy for an expert or a human to demonstrate the desired behavior expected from the agent rather than having the agent learn the desired behavior from a reward function. Instead of having to learn the entire policy from scratch, Imitation Learning tries to learn the decision policies based on the expert demonstrations.

The main component of IL is the environment, which is essentially a Markov Decision Process (MDP)[11]. This means that the environment has an S set of states, an A set of actions, a P(s'—s,a) transition model (which is the probability that an action a in the state s leads to state s') and an unknown R(s,a) reward function. The agent performs different actions in this environment based on its π policy. We also have the expert's demonstrations (which are also known as trajectories) τ = (s_0, a_0, s_1, a_1, ...) , where the actions are based on the expert's ("optimal") π^* policy[12].

There are two main approaches to learning a policy by mimicking an expert behavior: Behavioral cloning and Inverse reinforcement learning (IRL).

Behavioral cloning is a simple algorithm where it tries to learn a policy as a supervised learning problem by creating state-action pairs for a given set of demonstrations. The drawback for behavioral cloning is that it only tends to succeed with large amounts of data i.e. it needs a large number of expert trajectories and it is not efficient due to compounding error. Inverse reinforcement learning, on the other hand, does not suffer with this problem. IRL learns the reward function from the expert trajectories and then derives the optimal policy from it. However, they are extremely expensive to run[13].

Generative Adversarial Imitation Learning uses the formulation of Generative Adversarial Networks (GANs) i.e., a generator-discriminator framework, where a generator is trained to generate expert-like trajectories while a discriminator is trained to distinguish between generated and expert trajectories. GAIL directly learns the policy from the expert trajectories and not the reward function.

C. Reward System

The Reward System is the key to train an agent properly to learn a policy. Different reward functions have been shaped for both the algorithms by trial and error while training the agents. Table. I summarizes the rewards and penalties that were assigned to the agent for every action it takes.

TABLE I
REWARD SYSTEM

Condition	**Reward [PPO]**	**Reward [PPO with GAIL**
Within 2.5 units to the goal location	+0.00008	+0.00003
Best current distance to the goal location	+0.00002	+0.00002
Moving towards the goal but not the best distance to the goal in the current episode	-0.00004	+0.00001
Moving away from the goal	-0.00008	-0.00002
Within 2 units of distance to the wall	-0.005	-0.005
Within 2 units of distance to the obstacle	N/A	-0.005
Hit the wall [Episode Ends]	-0.5	-0.5
Hit an obstacle [Episode Ends]	-0.5	-0.5
Car Parked [Episode Ends]	+5	+5

V. RESULTS

The *mlagents* package saves statistics during the learning session. These statistics can be viewed on a utility called tensorboard. The hyperparameters used to train the agent are mentioned in Table. II.

TABLE II
HYPERPARAMETER TABLE

	PPO	**PPO with GAIL**
Batch size	512	256
Buffer size	10240	20480
Learning Rate	0.00001	0.00001
beta	0.001	0.03
epsilon	0.3	0.1
lambd	0.92	0.92
Hidden Layers	2	2
Neurons	64	64
Time horizon	128	256
GAIL strength	N/A	0.8

A. PPO

From Fig. 2 it is clear that the cumulative rewards keep on increasing with the number of steps. This means that our agent learnt a good policy and kept on accumulating more positive rewards over time. The model was run for 5M steps.

The policy loss as shown in Fig. 3 fluctuates throughout the training process, but it is less than 1. It means that the agent is trying to learn the optimal policy.

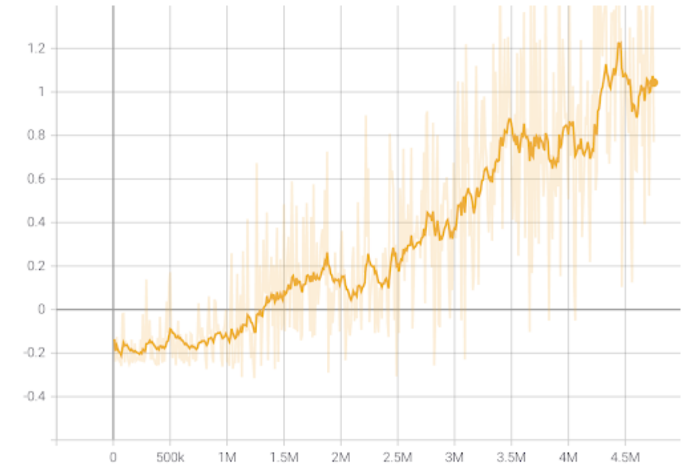

Fig. 2. PPO Cumulative Reward

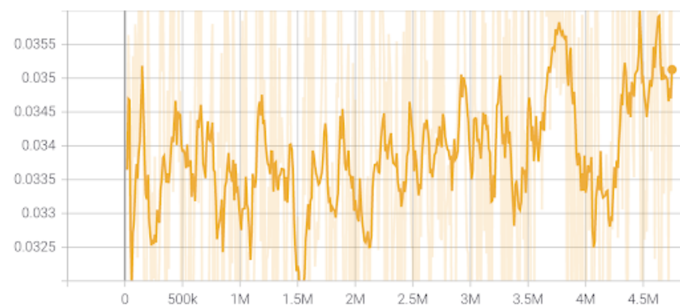

Fig. 3. Policy Loss

Fig. 4 shows that the entropy of the system is decreasing continuously over steps. Initially the decisions of the agent are random, but as it learns optimal policy, the decisions become more informed.

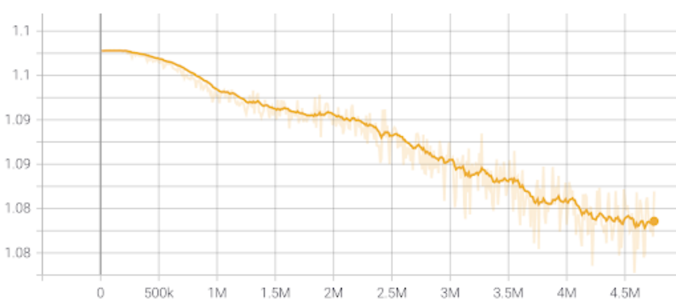

Fig. 4. Entropy

B. PPO with GAIL

Similar to the cumulative reward for the PPO model in Fig. 2, the cumulative reward for the agent trained using the GAIL model in Fig. 5 is also an increasing function. The more training steps, the better will be the policy learnt by the model. Also, the policy loss for GAIL model remained less than 1 throughout the training process as shown in the Fig. 6

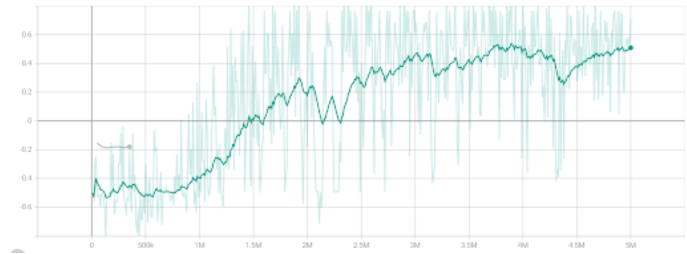

Fig. 5. Cumulative Reward

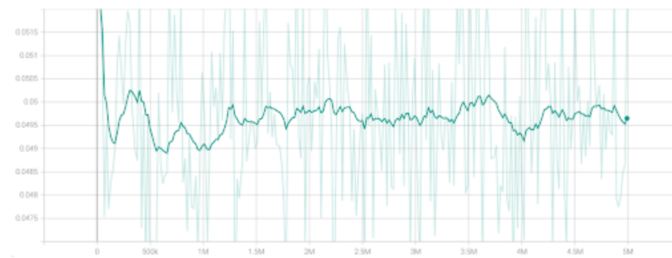

Fig. 6. Policy Loss

and the entropy is a decreasing function Fig. 7 indicating that the agent learned a better policy with time.

The mean loss of the value function update correlates to how well the model is able to predict the value of each state. At first, it increases since the agent is trying to learn. Once the reward stabilizes, it decreases[14].

The entropy of the system is decreasing continuously over steps. Initially the decisions of the agent are random, but as it learns optimal policy, the decisions become more informed.

The mean magnitude of the GAIL discriminator loss corresponds to how well the model imitates the demonstration

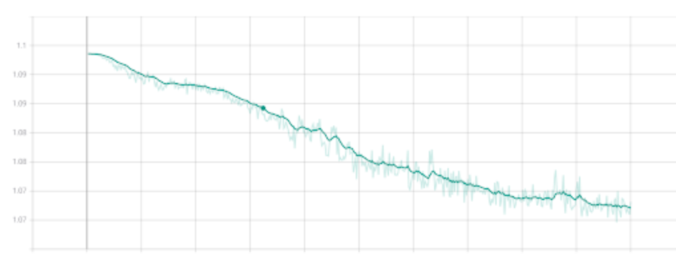

Fig. 7. Entropy

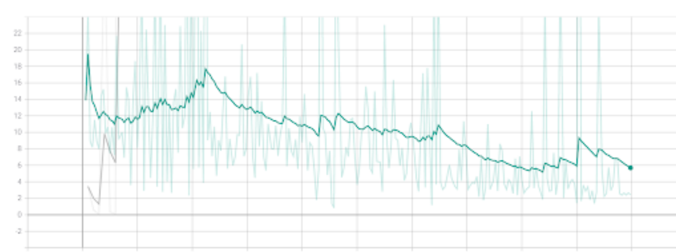

Fig. 8. Value Loss

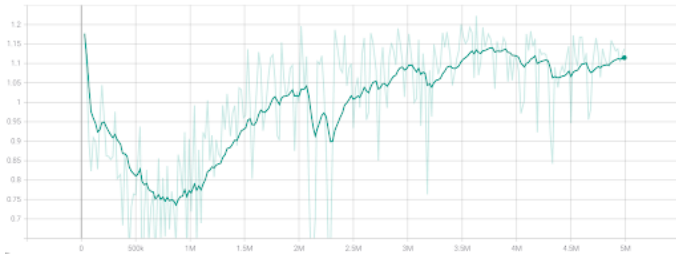

Fig. 9. Gail Loss

data[14].

C. Comparison Statistics

Table. III summarizes the results for the agents trained using the PPO algorithm and PPO with GAIL when the parking spots are same as the training spots. The test has been run for 150 episodes and the agent trained with PPO and GAIL is a clear winner having parked 121 times out of 150 and never hitting the wall compared to the 57 times parked by the PPO agent and 35 wall hits.

PPO with GAIL is also the winner for the second test case where parking spots are different from the training spots. The parking score for this agent is 138 with a massive accuracy of 92% when compared to 13% for the agent trained using PPO. These statistics are summarized in Table. IV.

TABLE III
TEST RESULTS WITH PARKING SPOTS SAME AS TRAINING SPOTS

	Number of Episodes	**Parking Score**	**Obstacle Hit Score**	**Wall Hit Score**
PPO	150	57	58	35
PPO with GAIL	150	121	29	0

TABLE IV
TEST RESULTS WITH PARKING SPOTS DIFFERENT AS TRAINING SPOTS

	Number of Episodes	**Parking Score**	**Obstacle Hit Score**	**Wall Hit Score**
PPO	150	20	108	22
PPO with GAIL	150	138	12	0

VI. LIMITATIONS, CONCLUSIONS AND FUTURE WORK

The main objective of the project was to develop a car parking agent that is capable of parking in the highlighted spot in a parking arena simulated using Unity while avoiding both static and moving obstacles and hitting the wall. The objective has been achieved and two agents have been developed for level 1 of the game where the agent had to avoid static obstacles.

Out of the two machine learning algorithms that were used, PPO and Imitation Learning using the GAIL, we noticed that the agent developed using the GAIL algorithm performed better than the PPO model. The GAIL agent was able to park in multiple parking spots while the agent trained using PPO was only able to park in one parking spot. The inferences made by the GAIL model were much better as the agent successfully parked the car in random test goal locations that are different from the training locations.

Even though the agent hit the walls a few times, with more training hours and different simulation environments, we are hopeful that the agent will be able to learn a much better policy for parking in any random spot with any arbitrary start location and in any environment.

One of the future targets that can be achieved as an enhancement to the current work is to make the acceleration and deceleration of the car as a training parameter. Currently, the car accelerates at a constant speed unlike the real world scenario where the car adjusts its speed based on its environment. Hence to make the model more realistic and closer to the actual world, we can make the car speed as a training parameter and the machine learning model would make inferences as to when to accelerate and decelerate based on the state of the game.

Secondly, apart from PPO and GAIL, there are numerous other algorithms that can be used for training an autonomous car parking agent both by the *mlagents* package in Unity and external resources. We have only scratched the surface of the problem for training an agent to park in a designated spot and there are many other ways that can be used to achieve the same result. A performance analysis into the different techniques to solve this problem and identifying the best algorithm that can be used to make inferences can be another future target for this project.

We hope that our efforts in developing an autonomous car parking system can be actually used in the real world scenario, thus solving one of the basic problems of machine learning - navigation and obstacle avoidance.

REFERENCES

[1] Daniel Zhang and Colleen P. Bailey. Obstacle avoidance and navigation utilizing reinforcement learning with reward shaping, 2020. URL https://arxiv.org/abs/2003.12863.

[2] Debaditya Acharya, Weilin Yan, and Kourosh Khoshelham. Real-time image-based parking occupancy detection using deep learning. *CEUR-WS*, 2087(5):33–40, 2018. doi: http://ceur-ws.org/Vol-2087/paper5.pdf.

[3] David Isele, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating intersections with autonomous vehicles using deep reinforcement learning. *CoRR*, abs/1705.01196, 2017. URL http://arxiv.org/abs/1705.01196.

[4] Andreas Folkers, Matthias Rick, and Christof Büskens. Controlling an autonomous vehicle with deep reinforcement learning. *CoRR*, abs/1909.12153, 2019. URL http://arxiv.org/abs/1909.12153.

[5] Szilárd Aradi, Tamás Bécsi, and Péter Gáspár. Policy gradient based reinforcement learning approach for au-

tonomous highway driving. pages 670–675, 08 2018. doi: 10.1109/CCTA.2018.8511514.

[6] Unity ml agents package for training game playing agent. URL https://github.com/Unity-Technologies/ml-agents.

[7] senevsemih. Unity–carparking. URL https://unitylist.com/p/134h/Unity-Car-Parking.

[8] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. URL https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[9] B. Thunyapoo, C. Ratchadakorntham, P. Siricharoen, and W. Susutti. Self-parking car simulation using reinforcement learning approach for moderate complexity parking scenario. In *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 576–579, 2020. doi: 10.1109/ECTI-CON49241.2020.9158298.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[11] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. *CoRR*, abs/1802.08757, 2018. URL http://arxiv.org/abs/1802.08757.

[12] Zoltán Lőrincz. A brief overview of imitation learning. URL https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c.

[13] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. URL http://arxiv.org/abs/1606.03476.

[14] Using tensorboard to observe training. URL https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Using-Tensorboard.md.