SUNY POLYTECHNIC INSTITUTE

PROJECT TITLE

AWS DEEP RACER

PROJECT REPORT

SUBMITTED BY

VISHAL KODAM(U00348357)
JONATHAN RAJ KATIKALA(U00348780)
ABHILASH REDDY DEVARAPALLI(U00348733)
SHASHIVARDHAN BATTULA(U00353739)
SAILESH SANGINENI(U00355440)

PREPARED FOR

MACHINE LEARNING – CS542

SUBMITTED TO

Prof. Yogesh Malhotra

AWS DEEPRACER

Vishal Kodam
Computer and Information Sciences
Suny Polytechnic Institute
Utica, New York, USA
kodamv@sunypoly.edu

Jonathan Raj
Computer and Information Sciences
Suny Polytechnic Institute
Utica, New York, USA
katikaj@sunypoly.edu

Abhilash Reddy Devarapalli
Computer and Information Sciences
Suny Polytechnic Institute
Utica, New York, USA
devaraa@sunypoly.edu

Shashivardhan Battula
Computer and Information Sciences
Suny Polytechnic Institute
Utica, New York, USA
battuls2@sunypoly.edu

Sailesh Sangineni
Computer and Information Sciences
Suny Polytechnic Institute
Utica, New York, USA
sangins@sunypoly.edu

ABSTRACT

The AWS DeepRacer project introduces a hands-on and engaging approach to reinforcement learning through autonomous racing. This initiative is designed to democratize machine learning, allowing individuals to experiment with complex concepts in a practical and enjoyable manner. Leveraging the capabilities of Amazon Web Services (AWS), DeepRacer provides both a virtual simulation environment and a physical racing car, creating a comprehensive platform for learning, experimentation, and competition.

General Terms

In the realm of AWS DeepRacer, several key concepts underpin the foundation of reinforcement learning. The concept of a **"state"** refers to the current situation or configuration of the racing environment, encapsulating variables like the car's position,
speed, and orientation. The **"reward"** is the feedback mechanism provided to the agent, reflecting the immediate outcome of its actions—positive for desirable behavior and negative for undesired actions.

The **"policy"** within the context of AWS DeepRacer is the strategy or set of rules that the agent employs to make decisions based on the current state. It defines the agent's behavior and guides its actions in the pursuit of maximizing cumulative rewards. The **"action"** is the move or decision made by the agent in response to the observed state, influencing the subsequent state and the overall learning process.

In the intricate dance between the **"agent"** and its surroundings, the agent is the learning entity—the DeepRacer model—tasked with navigating the racing environment. The **"environment"** encompasses the entire simulated or physical racing track, representing the dynamic space where the agent interacts, learns, and refines its decision-making processes.

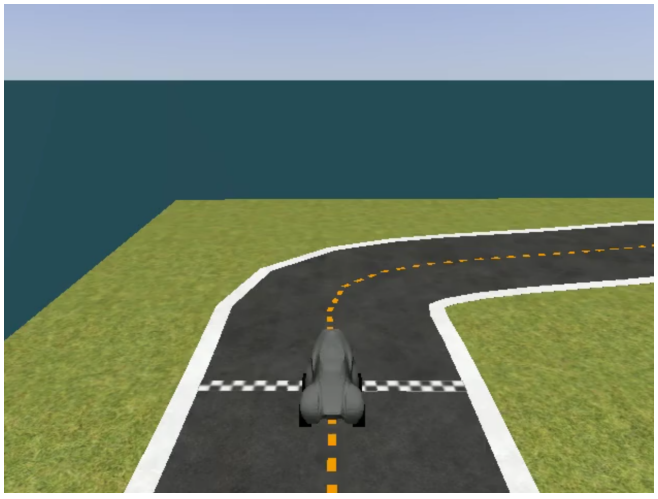
"Reinforcement Learning" is a pivotal concept wherein an agent, representing the DeepRacer model, learns decision-making by interacting with an environment, receiving feedback through rewards or penalties. The "Autonomous Racing" aspect refers to the vehicle's, or agent's, ability to navigate predefined tracks without direct human control, relying on learned models. These general terms underpin the core principles of DeepRacer, introducing users to the fundamentals of machine learning and autonomous systems. Understanding these terms is essential for individuals engaging with the project, providing a foundational grasp of the technologies at play in the exciting realm of autonomous racing and reinforcement learning.

AWS DeepRacer is a groundbreaking initiative designed to make machine learning accessible to a diverse audience, transcending traditional barriers of expertise. By harnessing the extensive capabilities of Amazon Web Services (AWS), this project encapsulates both a virtual simulation environment and a tangible racing car, inviting enthusiasts and developers alike to embark on an immersive journey of exploration and skill-building.

In a landscape where reinforcement learning often appears as a complex and abstract concept, AWS DeepRacer injects an element of excitement and hands-on engagement. It presents an opportunity for individuals, irrespective of their proficiency in machine learning, to interact with cutting-edge technology in a tangible and enjoyable manner. The amalgamation of virtual simulation and physical experimentation with a racing car empowers users to not only comprehend but actively participate in the nuances of autonomous vehicle development.

1. INTRODUCTION

In the context of AWS DeepRacer, the introduction to the documentation serves as a gateway to a transformative experience in the dynamic field of reinforcement learning and autonomous racing. At its core, AWS DeepRacer is a groundbreaking initiative designed to make machine learning accessible to a diverse audience, transcending traditional barriers of expertise. By harnessing the extensive capabilities of Amazon Web Services (AWS), this project encapsulates both a virtual simulation environment and a tangible racing car, inviting enthusiasts and developers alike to embark on an immersive journey of exploration and skill-building.



(Fig.1)

Key concepts underlying AWS DeepRacer, such as Markov Decision Processes (MDP) methods, form the backbone of the learning framework. MDP methods describe the interaction between the agent (DeepRacer model) and its environment, emphasizing the sequential decision-making process guided by states, actions, and rewards. These fundamental principles define the learning dynamics within the racing scenario.

The introduction also sheds light on sophisticated algorithms employed, with a notable mention of Proximal Policy Optimization (PPO). PPO plays a crucial role in optimizing the policy, or decision-making strategy, of the autonomous racing model. Its prominence signifies a focus on efficient and effective reinforcement learning, balancing exploration and exploitation in the ever-changing racing environment.

Further, the documentation delves into the intricacies of the action space type, elucidating the range of decisions the model can make during training and racing scenarios. Understanding the action space type is pivotal in comprehending the breadth of the agent's decision-making capabilities within the given environment.

Loss type is another critical facet highlighted, emphasizing the method by which the model learns from its experiences. This insight into the loss type provides a deeper understanding of the training process, where the model refines its decision-making strategy based on the evaluation of its actions and received rewards.

Reward functions, a cornerstone in reinforcement learning, are explored in the documentation. Crafting an effective reward function is crucial in shaping the learning objectives for the model. Examples include rewarding the model for staying on the track, following the racing line, and avoiding collisions. The intricacies of the reward function contribute significantly to the model's behavior and its ability to navigate the racing environment effectively.

Hyperparameter tuning, a critical aspect of model training, is outlined to underscore its role in optimizing the learning process. Experimenting with parameters such as learning rates and discount factors empowers developers to fine-tune

the model's behavior, enhancing its overall performance.

The training and evaluation processes are demystified, detailing the iterative refinement of the reward function and hyperparameter tuning to achieve optimal results. During training, the model learns by interacting with the environment, receiving rewards or penalties based on its actions. The evaluation phase assesses the model's performance, ensuring that it aligns with the desired behavior for autonomous racing scenarios.

In essence, the introduction to the AWS DeepRacer documentation sets the stage for a comprehensive exploration of reinforcement learning, showcasing the platform's commitment to democratizing machine learning and providing users with a transformative learning experience in the exciting domain of autonomous racing.

2. RELATED RESEARCH

Autonomous vehicles (AV) are the future of public transportation to reduce road congestion and accidents. However, fully self-driving cars are still a challenge for carmakers, and they must ensure the maximum driving security to avoid the ethical issue. To develop a mature model to AV, reinforcement learning becomes a solution which can explore many possibilities and choose the best possible action choice facing different road conditions. AWS DeepRacer is a comprehensive platform for researchers

Related research forms the intellectual backdrop of AWS DeepRacer, enriching the project by drawing from advancements in reinforcement learning and autonomous systems. Reinforcement learning, a cornerstone in the field, has seen significant contributions, such as Deep Q Networks (DQN), which enhance the capacity of agents to make decisions based on learned experiences. Research on policy optimization algorithms, like Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO), has influenced the development of effective learning strategies within DeepRacer.

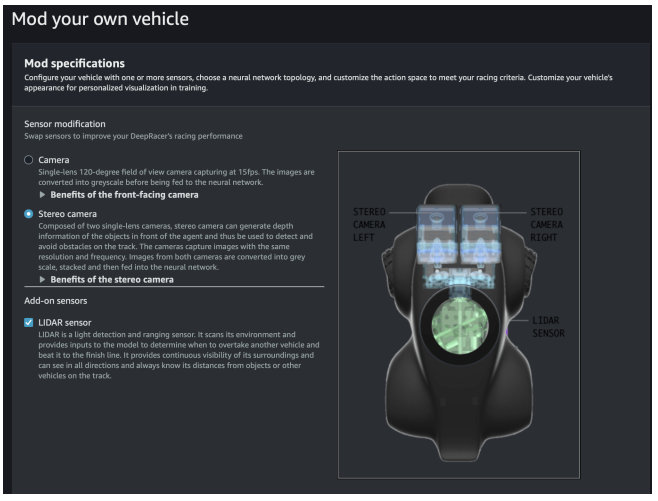
In the realm of autonomous systems, work on end-to-end learning for self-driving cars has offered insights into holistic model architectures. Transfer learning, a concept that allows models trained on one task to be adapted to another, has implications for improving DeepRacer's adaptability across diverse racing scenarios. The integration of these research findings enhances the robustness and versatility of AWS DeepRacer, providing users with a comprehensive and cutting-edge platform for exploration in the dynamic fields of reinforcement learning and autonomous racing. The continuous dialogue with related research ensures that AWS DeepRacer remains at the forefront of technological innovation and leverages the most effective strategies to advance the understanding and application of machine learning principles.

3. ALGORITHMS

In the context of AWS DeepRacer, algorithms play a pivotal role in orchestrating the learning process and refining the model's decision-making capabilities. The following sections provide insights into the fundamental algorithms utilized within the project.

Vehicle Modification

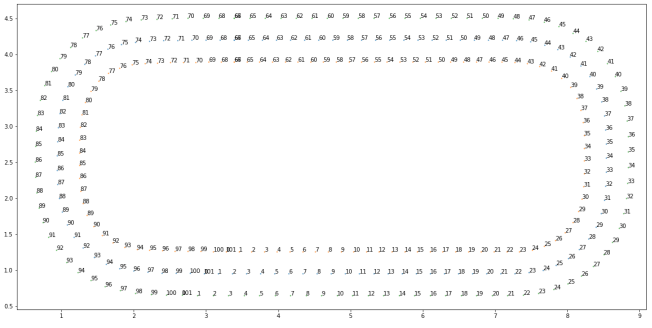
vehicle with stereo camera and lidar sensor.



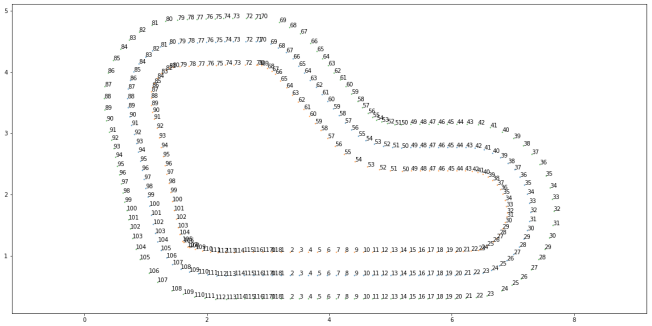
(Fig.2)

Simulation Environment Algorithms

AWS DeepRacer's virtual simulation environment incorporates algorithms that simulate real-world racing scenarios. Physics engines and collision detection algorithms enable realistic interactions, allowing the reinforcement learning model to adapt to various track conditions. Dynamic environmental factors, such as weather or surface conditions, are simulated through algorithms that introduce variability into the training process, enhancing the model's adaptability. Track what we considered is the way point track. So that we can make the agent satisfied with the rewards and reach the target.



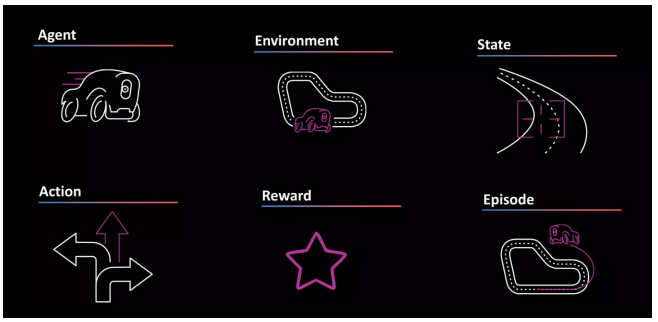
(Fig.3)



(Fig.4)

Reinforcement Learning Algorithms

DeepRacer employs state-of-the-art reinforcement learning algorithms to optimize the policy of the autonomous racing model. Model-free algorithms, including Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO), guide the agent's decision-making process. These algorithms strike a balance between exploration and exploitation, crucial for effective learning in dynamic racing environments.



(Fig.5)

Hyperparameter Tuning

Optimizing hyperparameters is a critical aspect of training machine learning models. AWS DeepRacer employs algorithms for hyperparameter tuning, allowing developers to experiment with parameters like learning rates and discount factors. Automated algorithms facilitate the search for optimal hyperparameter configurations, streamlining the model refinement process.

Evaluation Metrics Algorithms

The evaluation of model performance relies on algorithms that quantify key metrics. Huber and Mean Squared Error (MSE) functions are employed to measure the disparity between predicted and actual values during training and evaluation. These algorithms contribute to shaping the learning behavior of the model and assessing its proficiency in autonomous racing scenarios.

Transfer Learning Algorithms

Transfer learning algorithms enable knowledge gained in one racing scenario to be applied to others, promoting adaptability. The project leverages transfer learning to enhance the model's ability to generalize learning across diverse tracks, providing a more versatile and efficient autonomous racing agent.

The integration of these algorithms within AWS DeepRacer underscores the project's commitment to leveraging cutting-edge techniques, ensuring a robust and adaptive

platform for users to explore the intricacies of reinforcement learning and autonomous racing.

4. PROCEDURE

i. Define the Reward Function:

The provided reward function considers various parameters, including distance from the center, track width, and waypoints for different lanes and speeds.

It rewards staying on the track, following the racing line, and maintaining appropriate speeds based on waypoints.

```python
def reward_function(params):
    # ... (code)
    return float(reward)
```

ii. Set Up Waypoints:

- Define waypoints for the left, center, and right lanes, as well as waypoints for fast and slow speeds. These waypoints are used in the reward function to tailor the model's behavior.

```
#racing line
left_lane = [27,28,29,30,31,32,33,73,74,75,76,77,78,79,80,]#Fill in the waypoints

center_lane = [1,2,3,4,5,6,7,8,9,10,21,22,23,24,25,26,34,35,36,37,38,39,52,53,54,55,56,57,58,59
    ,60,61,62,63,64,65,66,67,68,69,70,71,72,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96
    ,97]#Fill in the waypoints

right_lane = [11,12,13,14,15,16,17,18,19,20,40,41,42,43,44,45,46,47,48,49,50,51,98,99,100
    ,101]#Fill in the waypoints

#Speed
fast = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,49,50,51,52,53,54,55,56,57,58,59,60
    ,61,62,63,64,65,66,67,68,69,70,71,72,96,97,98,99,100,101]#Fill in the waypoints, 2m/s
slow = [21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,73
    ,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,90,91,92,93,94,95]#Fill in the waypoints, 1m/s
```

(Fig.6)

iii. Reward Function:

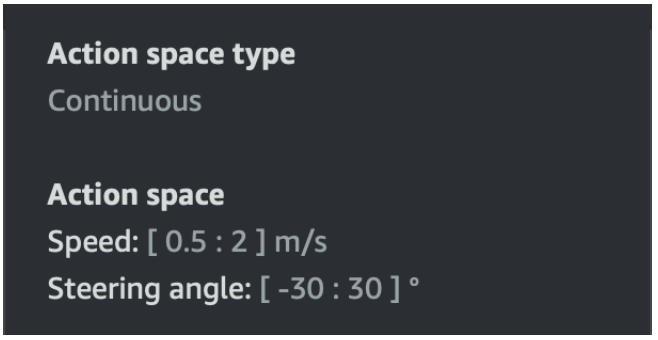
The reward function in AWS DeepRacer encompasses several key criteria for guiding the autonomous vehicle's behavior during training. Center Variance evaluates the car's positioning by measuring its distance from the center of the track. The Lane Check component incentivizes the model to stay in the correct lane, aligning its trajectory with predefined waypoints. Speed Check ensures that the model is appropriately maintaining its speed according to specified waypoints. Finally, Track Alignment serves as a crucial factor, providing rewards when all wheels of the vehicle remain on the track. These individual components collectively contribute to shaping the model's decision-making process and optimizing its performance in navigating the racing environment.

```
reward = 21

if params["all_wheels_on_track"]:
    reward += 10
else:
    reward -= 10

if params["closest_waypoints"][1] in left_lane and params["is_left_of_center"]:
    reward += 10
elif params["closest_waypoints"][1] in right_lane and not params["is_left_of_center"]:
    reward += 10
elif params["closest_waypoints"][1] in center_lane and center_variance < 0.4:
    reward += 10
else:
    reward -= 10
if params["closest_waypoints"][1] in fast:
    if params["speed"] == 2 :
        reward += 10
    else:
        reward -= 10
elif params["closest_waypoints"][1] in slow:
    if params["speed"] == 1 :
        reward += 10
    else:
        reward -= 10

return float(reward)
```

(Fig.7)

iv. Action Space Type

There are two different space states, continuous action space and discrete action space. We choose continuous action space for this model. speed and steering angle is set to the value.



(Fig.8)

v. Hyperparameter Tuning:

Hyperparameter	Value
Gradient descent batch size	64
Entropy	0.01
Discount factor	0.999
Loss type	Huber
Learning rate	0.00003
Number of experience episodes between each policy-updating iteration	20
Number of epochs	10

(Fig.9)

a.Gradient Descent Batch Size (64)

This parameter determines the number of experiences sampled from the replay buffer for each iteration of gradient descent. A batch size of 64 implies that 64 experiences are used to update the policy.

b. Entropy (0.01)

Entropy is a regularization term that encourages exploration. A value of 0.01 strikes a balance between exploration and exploitation in the learning process.

c. Discount Factor (0.999)

The discount factor determines the importance of future rewards. A value of 0.999 indicates a preference for long-term rewards in the decision-making process.

d. Loss Type (Huber)

Huber loss is a robust regression loss that mitigates the influence of outliers. It is chosen for its resilience to extreme values in the training data.

e. Learning Rate (0.00003)

Learning rate regulates the size of the step taken during optimization. A smaller learning rate (0.00003) ensures cautious updates to the model parameters, preventing overshooting.

f. Number of Experience Episodes Between Each Policy-Updating Iteration (20)

This parameter defines how often the policy is updated based on experiences collected. In this case, the policy is updated every 20 episodes, striking a balance between stability and adaptability.

g. Number of Epochs (10)

The number of epochs determines how many times the model iterates over the entire training dataset during each policy update. With 10 epochs, the model undergoes multiple passes over the data to refine its learning.

vi. Building the Model:

a. Define Reward Function:

Implement the provided reward function and adjust waypoints based on the oval track's characteristics.

b. Set Up Waypoints:

Clearly define waypoints for each lane and speed category, ensuring accurate representation of the racing scenario.

c. Configure Hyperparameters:

Use the provided hyperparameter values to configure the model. These parameters influence the learning process and overall model behavior.

d. Training and Evaluation:

Launch training simulations using the defined reward function, waypoints, and hyperparameters. Monitor training progress and

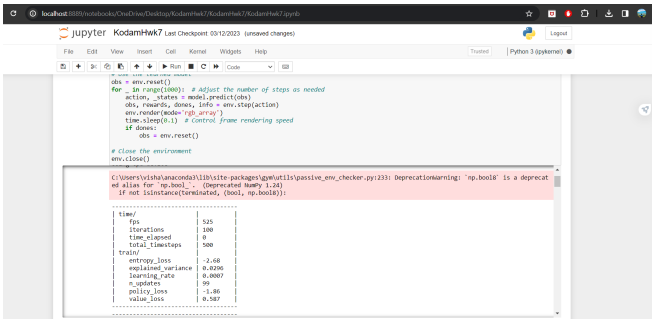
evaluate the model's performance using the specified reward and penalty criteria.

e. Iterative Refinement:

Fine-tune the reward function, adjust waypoints, or experiment with hyperparameters based on observed model behavior. Reiterate the training and evaluation process until the model demonstrates desired performance on the oval track.

AWS DeepRacer operates on the principles of reinforcement learning, where an autonomous racing model learns to navigate tracks by interacting with a simulation environment. The model's decision-making is guided by a reward function that incentivizes desirable behaviors, such as staying on the track and maintaining optimal speeds. Through a combination of waypoints and hyperparameter tuning, the model refines its policy to adapt to diverse racing scenarios. Once the model is trained and refined, it can be deployed to a physical DeepRacer car, enabling it to autonomously navigate real-world tracks. The deployment process involves transferring the knowledge gained in the simulation environment to the physical car, creating a seamless transition from virtual training to practical application. This mechanism encapsulates the end-to-end journey of building, training, and deploying a reinforcement learning model with AWS DeepRacer, making it a comprehensive platform for hands-on exploration and deployment of autonomous racing agents.

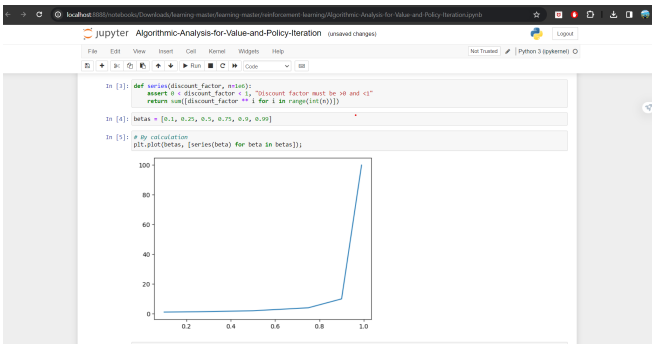
5. NOTEBOOKS

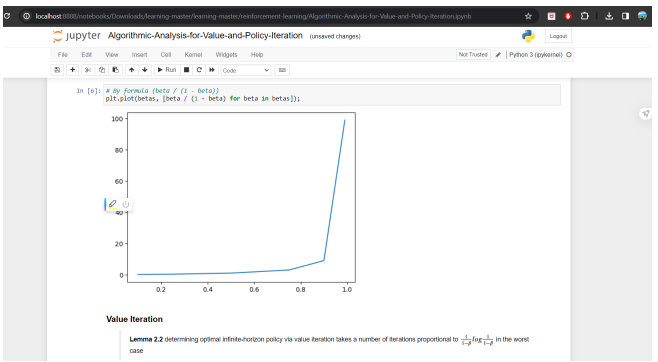


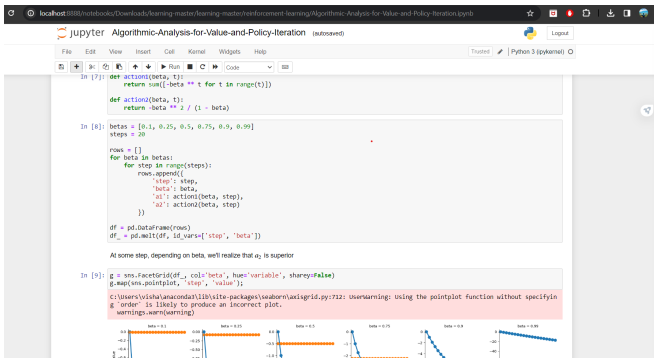

```
    print(f"Episode {episode + 1} reward: {episode_reward}")

average_reward = total_reward / num_episodes
print(f"Average reward over {num_episodes} episodes: {average_reward}")
env.close()
```

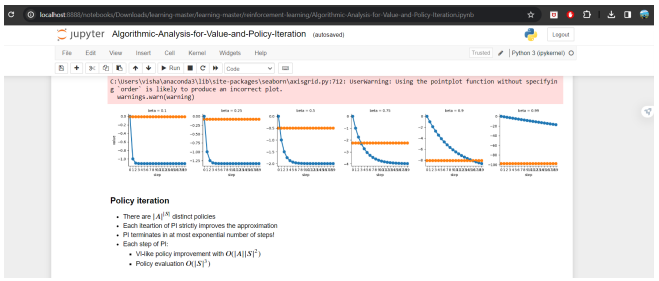
```
C:\Users\visha\anaconda3\lib\site-packages\gym\utils\passive_env_checker.py:233: DeprecationWarning: `np.bool8` is a deprecated alias for `np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
Episode 1 reward: 4.0
Episode 2 reward: 3.0
Episode 3 reward: -1.0
Episode 4 reward: 12.0
Episode 5 reward: -1.0
Episode 6 reward: 0.0
Episode 7 reward: -1.0
Episode 8 reward: -1.0
Episode 9 reward: 3.0
Episode 10 reward: -20.0
Episode 11 reward: -6.0
Episode 12 reward: 1.0
Episode 13 reward: 1.0
Episode 14 reward: -2.0
Episode 15 reward: 7.0
Episode 16 reward: 8.0
Episode 17 reward: 15.0
Episode 18 reward: 11.0
Average reward over 18 episodes: 1.8333333333333333
```

Basic RL implementation for the reward function to understand the workflow of the code in Deep Racer.

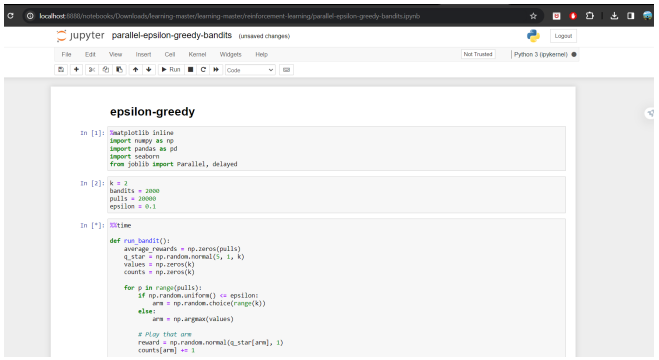


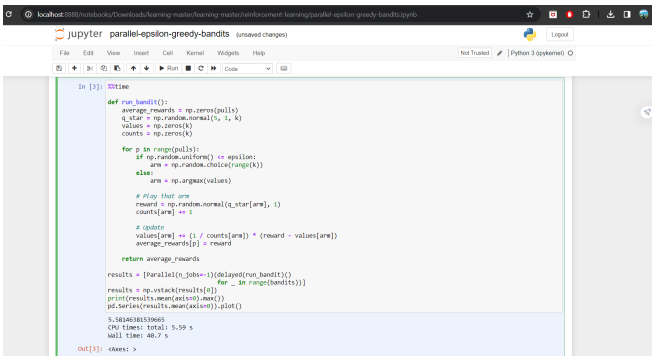


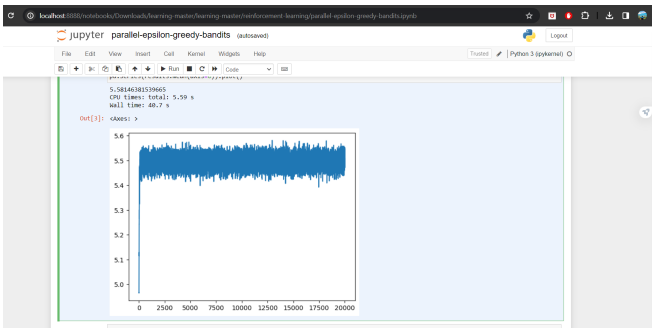




These snapshots show the value and policy iteration and discount factor. To understand how these affect the deep racer model in the hyperparameter.







Epsilon-Greedy is a simple method to balance exploration and exploitation by choosing between exploration and exploitation randomly. The epsilon-greedy, where epsilon refers to the probability of choosing to explore, exploits most of the time with a small chance of exploring.

6. FUTURE SCOPE

In the rapidly advancing field of autonomous systems, AWS DeepRacer stands out as a groundbreaking platform, offering a tangible bridge between theoretical reinforcement learning concepts and real-world implementation. One of the key strengths lies in the platform's sophisticated environment simulation, which serves as the training ground for reinforcement learning models. This virtual realm, manifested in a 3D racing simulator, enables developers to experiment with various track designs, replicating diverse real-world scenarios. Such flexibility is paramount in preparing models for the unpredictability of actual environments.

The versatility of AWS DeepRacer is exemplified in its support for multiple racing scenarios, each presenting unique challenges in track complexity and design. This adaptability empowers developers to tailor their training experience, ensuring that models are well-equipped to handle a wide range of real-world conditions. The

platform's race type feature aligns with the need for autonomous systems to navigate dynamic and unpredictable environments, making it a valuable asset for real-world implementation.

At the heart of AWS DeepRacer's learning process is the reward function, a pivotal component in reinforcement learning. Crafting an effective reward function is crucial for defining learning objectives. For instance, rewarding the model for staying on the track, following the racing line, and avoiding collisions shapes the behavior of the autonomous agent. This concept has profound implications in real-world scenarios, where reinforcing desired behaviors and penalizing undesired actions are crucial for the safety and efficiency of autonomous systems.

The inclusion of waypoints further enhances the platform's real-world applicability. Waypoints guide the model along a predefined path, enabling it to navigate the track efficiently. By strategically positioning waypoints, developers can simulate specific racing challenges, optimizing the learning process to address real-world complexities.

AWS DeepRacer's use of model-free reinforcement learning underscores its commitment to practical and adaptive algorithms. The model learns by interacting with the environment, receiving rewards or penalties based on its actions. The flexibility to fine-tune hyperparameters such as learning rate and discount factor allows developers to adapt the model's behavior, reflecting the need for adaptable systems in real-world applications.

The platform's reliance on the Huber and Mean Squared Error (MSE) functions for training and evaluation introduces robustness to outliers and ensures a balanced approach to error measurement. This aspect is vital in real-world scenarios where unexpected events or challenges may deviate from expected behavior.

Looking ahead, AWS DeepRacer's future work holds promise for further enriching the user experience and advancing the field of reinforcement learning. Enhanced virtual environments could introduce more sophisticated challenges, closely mirroring the intricacies of real-world racing scenarios. The extension of physical racing capabilities, possibly incorporating new sensors, brings the platform one step closer to seamless integration into real-world applications. Continued research into advanced algorithms and model architectures aims to enhance learning efficiency and adaptability, crucial for robust decision-making in diverse environments. The emphasis on expanded community engagement through forums, competitions, and workshops ensures a collaborative learning environment, accelerating the collective understanding of reinforcement learning principles for autonomous racing applications. As AWS DeepRacer evolves, it not only shapes the future of autonomous systems but also cultivates a community-driven ecosystem ready to tackle the challenges of real-world implementation.

(Fig.10)

7. CONCLUSION

AWS DeepRacer represents a pioneering initiative that democratically opens the doors to the captivating realms of reinforcement learning and autonomous racing. By leveraging the scalable infrastructure of Amazon Web Services, DeepRacer transcends traditional barriers, allowing enthusiasts and developers to delve into machine learning with hands-on, experiential learning.

The simulation environment, featuring realistic physics and dynamic racing conditions, serves as a crucial component in training reinforcement learning models. Algorithms governing the training process, such as reinforcement learning algorithms and

hyperparameter tuning, ensure an effective and efficient learning experience. Additionally, the employment of evaluation metric algorithms, including Huber and Mean Squared Error functions, guarantees a comprehensive assessment of model proficiency.

AWS DeepRacer's versatility extends into transfer learning algorithms, enabling the application of knowledge gained in one racing scenario to diverse tracks. This not only enhances the adaptability of the model but also positions DeepRacer as a flexible platform for exploring the complexities of autonomous systems.

Looking ahead, the project's commitment to continual improvement and innovation promises an even more immersive and impactful learning experience. AWS DeepRacer stands as a testament to the democratization of technology, making the intricate field of machine learning accessible to a broad audience and fostering a community-driven exploration of the fascinating intersection between artificial intelligence and autonomous racing.

8. CONTRIBUTION:

ShashivardhanBattula (team member) focuses on the foundational aspects of reinforcement learning (RL) within the context of the AWS DeepRacer project. They delve into the understanding of key RL concepts such as states, rewards, policies, actions, agents, and environments. Additionally, they explore the Markov Decision Process (MDP) as a mathematical framework and emphasize Proximal Policy Optimization (PPO) among various algorithms. The team member highlights why PPO is chosen for its suitability in the AWS DeepRacer project, considering its advantages in handling continuous action spaces and its stability in training.

Sailesh Sangineni (team member) delves into the value-based and policy-based methods in RL, considering algorithms like DQN, PPO, TRPO, and DDPG within the MDP framework. They justify the selection of PPO over other algorithms, aligning it with the project requirements. Additionally, the team member provides an overview of the essential elements required for the actual implementation of the AWS DeepRacer project, ensuring a clear understanding of the project's algorithmic foundations.

Abhilash Reddy Devarapalli (team member) focuses on defining the action space type (continuous and discrete) within the AWS DeepRacer project. They explore the significance of waypoints and how points are described on the racing track. Additionally, the team member defines the reward function, a critical component in guiding the agent's learning process throughout the project.

Vishal Kodam (team member) takes charge of learning and explaining the reward function code implementation. They discuss the utilization of huber loss and hyperparameters, emphasizing how these values impact the model's performance. The team member provides insights into the coding aspects of the reward function, highlighting its role in shaping the behavior of the AWS DeepRacer model.

Jonathan Raj Katikala (team member) focuses on the training and evaluation phases of the model within the AWS DeepRacer project. They discuss the reward graph as a metric for assessing the model's performance during training. The team member concludes the documentation by summarizing the key findings and outcomes of the AWS DeepRacer project, providing a comprehensive understanding of the training and evaluation processes.

Everyone contributed more than 20% of the project work each. Few tasks were done by 2 people like implementing notebooks, figuring out loss function by implementing

multiple functions and observing the output for our model.

Though in the start there was a problem in coordinating among ourselves, we eventually pulled through by conducting regular online meet sessions to know each person's task update.

Since everyone's contribution was significant we can't rank among ourselves. Since the AWS DeepRacer platform is not completely free, everyone used our temporary free access accounts, we were able to learn the working of it and implement our project models into it.

Overview

The outlined Artificial Intelligence-Machine Learning course is structured around two central themes: the core theoretical technical algorithms related to Deep Neural Networks (DNNs) and Deep Learning (DL) technologies, and the technical methods and measures for enhancing the performance of these algorithms. In the context of AWS DeepRacer, we can draw connections between these course themes and the aspects covered in the project documentation:

Core Theoretical Algorithms:

Regression, Classification, and DNNs: In the AWS DeepRacer project, the fundamental algorithms for training the autonomous racing model are rooted in deep neural networks (DNNs). These networks enable the model to understand complex relationships within the racing environment, allowing for both regression (predicting continuous actions like steering and acceleration) and classification (making decisions about actions at each time step).

CNNs: Convolutional Neural Networks (CNNs) may come into play when processing visual input data, such as the racing track images captured by the DeepRacer car's cameras. CNNs are effective in image recognition tasks and can be applied to extract features from the track images.

RL (Reinforcement Learning): AWS DeepRacer leverages reinforcement learning, an area covered in the course, to enable the model to learn optimal behavior by interacting with its environment. Concepts like states, rewards, and policies, as well as algorithms like Proximal Policy Optimization (PPO), are relevant here.

Ensemble Learning: The use of ensemble models (DTEs-Ensembles) may find application in combining projections from multiple models, potentially enhancing the robustness and accuracy of the racing model.

LLMs (Likelihood Models): Likelihood models could be integrated into the training process to understand the probability distribution of certain events or states, influencing decision-making in the model.

Technical Methods and Measures for Performance Improvement:

Hyperparameter Tuning: The course's focus on hyperparameter tuning is directly applicable in the AWS DeepRacer project. Fine-tuning hyperparameters, such as learning rates, can significantly impact the model's training efficiency and overall performance.

Regularization: Techniques for regularization, which prevent overfitting, can be crucial in ensuring that the trained model generalizes well to new, unseen racing scenarios. This aligns with the course's emphasis on avoiding model overfitting.

Optimization: The course covers optimization techniques, and in the context of AWS DeepRacer, this could involve optimizing the neural network's architecture, training process, or other aspects to improve the model's speed and efficiency in decision-making.

Bias-Variance Tradeoff: Understanding the bias-variance tradeoff, as covered in the course, is essential in balancing the model's ability to capture underlying patterns in the data without overfitting to noise, a critical consideration in real-world racing scenarios.

Learning Rate: The course likely explores the impact of learning rates on model training. In AWS DeepRacer, adjusting the learning rate is a practical implementation to optimize the convergence and stability of the reinforcement learning algorithm.

By considering these themes and related concepts from the course, the AWS DeepRacer project documentation can showcase the theoretical foundations and practical implementations of advanced AI-ML algorithms. The integration of these concepts ensures a comprehensive understanding of both the underlying technologies and the methodologies employed to enhance the performance of autonomous racing models.

References

[1]https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf

[2] DeepRacer Log Analysis GitHub Repository. (2023). "Oval Track Waypoints." Retrieved from https://github.com/cfghack/deepracer-log-analysis/blob/main/tracks/Oval_track.npy

[3] DeepRacer Log Analysis GitHub Repository. (2023). "Tracks Directory." Retrieved from https://github.com/cfghack/deepracer-log-analysis/tree/main/tracks

[4] DeepRacer Community Analysis GitHub Repository. (2023). "Tracks Directory." Retrieved from https://github.com/aws-deepracer-community/deepracer-analysis/tree/master/tracks

[5]Waypoint Visualization GitHub Repository. (2023). Retrieved from https://github.com/ARCC-RACE/waypoint-visualization

[6] Grand, Simon, et al. "Resource allocation beyond firm boundaries: A multi-level model for open source innovation." *Long Range Planning* 37.6 (2004): 591-610.

[7] Balaji, Bharathan, et al. "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning." *arXiv preprint arXiv:1911.01562* (2019).

[8]https://github.com/gord-peters/deep-racer-notebook

[9]https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_QuerySyntax-examples.html

[10]https://github.com/cfghack/deepracer-log-analysis

[11]https://github.com/jeremypedersen/deepracer-deepstats

[12]https://github.com/cfghack/deepracer-log-analysis

[13]https://github.com/aws-deepracer-community/deepracer-simapp

[14]https://codelikeamother.uk/using-jupyter-notebook-for-analysing-deepracer-s-logs

[15]https://docs.aws.amazon.com/sagemaker/latest/dg/reinforcement-learning.html

[16]https://gidutz.medium.com/turning-aws-deepracer-into-an-angry-bull-powered-by-tensorflow-9dbdd6b3e9cb

[17]https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_QuerySyntax-examples.html

[18]https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3

[19]https://nivlab.princeton.edu/sites/g/files/toruqf3851/files/bennett_langdon2021.pdf

[20]https://docs.aws.amazon.com/deepracer/latest/developerguide/educator.html