Portfolio Project - ML Agents Final Version

Karl Estes

Colorado State University Global

CSC 525: Principles of Machine Learning

Dr. Pubali Banerjee

February 13th, 2022

Portfolio Project - ML Agents Final Version

1. Introduction

The final version of my Autonomous Vehicle Agent is designed to navigate from a starting location to a goal location while avoiding obstacles in its path. The following paper discusses the construction of the Agent, choices around reward functions and hyperparameter choices, and some issues that were encountered along the way.

While explanations will appear in the following discussion, the final version of the Autonomous Vehicle Agent was simplified from the initial plan. No visual perception was utilized due to the significant slowdown of training CNNs on my local machine. Simplification of the training environment also occurred to accommodate the incorporation of basic curriculum learning. The final training environment does not include corners, but the layout of the obstacles for avoidance is random on each map generation.

2. Constructing The Agent

When designing the simulation environment in unity, criteria outlined by Huynh et al. (2021) were considered. They discussed four primary abilities an autonomous vehicle simulator must contain — control of the scenarios, sensing measurements, functions related to environment reset, time accelerations, etc., and ML integration. Since ML integration was provided by the Unity ML-Agents Toolkit (2021), the first three were the primary focus points.

Scenario control was handled via a map generation script. Each environment was broken down into tiles containing either a straight road, a parking lot, or a curved road. When running the simulation in the Unity engine, a height and width for a map space could be defined along with a total path length. Each time a map was generated, a new path was randomly generated,

and tiles were instantiated into the environment. The number of obstacles that appeared on the tiles was also controllable, though their placement was random for each tile.

Measurement sensing and control functionality were all implemented in the Agent's script. Control was handled by resetting the Agent to the start of the map in the event of a collision or falling off the map, and a new map was generated after each successful run. As for sensing, the Agent was provided data from two separate *Ray Perception Sensor 3D* components, which were able to detect tags denoting any of the objects that could spawn as well as the ground and the parking lot. One set of rays was established as long-distance while the other was used for more near-car and on-the-ground detection. The Agent was also provided information about its position, direction to the goal, and velocity. Initially, this information was provided as global coordinates/values, but better results were achieved when translated to local space coordinates/ values. Initially, the Agent was also provided with two *Camera Sensors*. It was found that training a convolutional neural network from two sensors was computationally expensive and significantly reduced training time. The final Agent design removed both *Camera Sensors* and did away with visual observations altogether due to training constraints.

As per the documentation in the Unity ML-Agents Toolkit (2021), including a *Camera Sensor* requires rendering a graphical instance of the environment during training. While the training process could be run at accelerated timescales, rendering with graphics limited the number of concurrent environments that could be utilized and reduced maximal training speed. For instance, a test of an Agent with and without visual sensors, with extrinsic reward only, and with default hyperparameter values yielded the following results: The Agent with visual observations took **1h 38m 48s** to reach **439k** steps and could manage to train in four

environments simultaneously. The Agent without visual observations took **10m 57s** to reach **435k** steps and managed to train eight environments simultaneously.

While operating in the simulated environment, the Agent controls three continuous values: thrust, steering, and braking. Since these three components can and do exist on a spectrum when driving a real car, the choice was made to keep the values continuous, though bounds were applied by clipping the signals between -1 & 1, -1 & 1, and 0 & 1, respectively. Each signal was then scaled by a *maxMotorTorque*, *maxSteeringAngle*, and *maxBrakingTorque*. Utilizing continuous values for autonomous vehicle simulations is not uncommon; previous experiments with OpenAI Gym have utilized continuous acceleration and steering for training a car to drive around a randomly generated racetrack (Holubar & Wiering, 2020). Thrust, steering, and braking are also easy to apply via a *Wheel Collider*, and the application of each force was accomplished by copying the example outlined in the **Unity Manual** (Unity Technologies, 2022).

3. Reward Function

Shaping the reward function was one of the more challenging tasks in designing the Autonomous Vehicle Agent. Initially, the Agent was provided with both continuous and sparse rewards. If the Agent collided with an object, tipped the car over, or fell out of the training environment, a negative reward was given, while a positive reward was provided upon reaching the goal. In order to encourage the Agent to reach the goal as quickly as possible, an existential time penalty was also provided.

Regardless of its magnitude, though, the existential time penalty seemed to introduce unwanted behavior in the trained agents. In all early configurations with a time penalty, the

Agent appeared to learn an optimal policy of "drive backward off the ledge as quickly as possible." A reasonable assumption is that this minimized the negative reward, and the exploration ability of the Agent was too minimal given the complexity of the environment.

To combat this, the second significant reward function change was to remove the existential time penalty and introduce imitation learning into the training process. While the specifics of the employed imitation learning schemes are discussed in section 4, it was inspired by Nair et al. (2018). They were able to use demonstrations in a reinforcement learning environment to overcome the problem of exploration for robotic tasks like stacking blocks. Based on their paper, only sparse rewards were used.

While this did seem to prevent the intentional cliff diving behavior, it was still challenging to get an Agent to converge on a reasonable result. While the final process breakdown of how this was accomplished is discussed in section 6, the reward function did undergo a third rework. A blog post by Bowers (2020) discussed a simple car agent driving around a fixed size stage looking for a blue target. In order to incentive the Agent to move towards the target, a reward was given based on the Agent's relative velocity towards the goal. Bowers (2020) also did not provide any penalties for "bad behavior," such as driving off the edge of the training environment. This reward scheme based on movement direction, no negative penalties, and a positive reward on reaching the goal is the reward structure adopted for final training.

4. GAIL, Curiosity, and Behavioral Cloning

Given the complexity of even a simple autonomous vehicle simulation, it was decided to include imitation learning in the training process. As described on the **Training with Imitation**

Learning page in the Unity ML-Agents Toolkit (2021) repository, two types of imitation formats are available: behavioral cloning (BC) and generative adversarial imitation learning (GAIL). Since the provided documentation stated that BC works best when nearly all states the Agent can be visited in demonstration files, GAIL was chosen as the only form of imitation learning. Ho & Ermon (2016) proposed GAIL as a new general framework for extracting a policy directly from data. GAIL provides a secondary reward signal that critiques and rewards an Agent for acting in a manner similar to a pre-recorded demonstration set.

Including a curiosity module was also explored for several Autonomous Vehicle Agent iterations but was ultimately left out of the final build. For an agent to succeed at learning, they must explore their environment while simultaneously exploiting any currently accrued knowledge to achieve the best possible goal (Still & Precup, 2012). While a curiosity module rewards an agent for exploration, the standard randomization of actions during training with reinforcement learning was chosen for the Autonomous Vehicle Agent.

5. Choosing Hyperparameters

Hyperparameter Tuning was done manually while testing the various agent configurations, and most advice for recommended hyperparameter values was taken from the **Training Configuration File** provided in the Unity ML-Agents Toolkit (2021) documentation. Since Unity ML-Agents was the framework being used for training the AI Agents and the implementation of the training algorithms are handled in that toolkit, it felt best to abide by the outlined recommendations for this project. Appendix A contains a copy of the hyperparameter file that was used to train the final Autonomous Vehicle Agent iteration.

The chosen hyperparameter values were based on using proximal policy optimization (PPO) as the training algorithm. PPO was proposed by Schulman et al. (2017), which showed better performance over other trust-region methods while being easier to implement. PPO is also built into the Unity ML-Agents toolkit, and there is a specific hyperparameter discussion in the documentation around PPO-specific configurations.

There were a few hyperparameters that, while still within recommended ranges, were tuned over a few testing sessions. For example, the *batch_size* and *buffer_size* were both increased from their default. Since all the actions the Agent could take are continuous, a larger batch size was necessary to capture a sufficient number of experiences with each gradient descent run. It is also possible to obtain a stable learning curve by increasing the batch size of different trainers without decaying the learning rate (Smith et al., 2017). While learning rate decay was still utilized, the research helped affirm the decision for the *batch_size* that was chosen. The total *buffer_size* was scaled to be multiple times larger than *batch_size* as per the instructions provided in the documentation.

Other notable hyperparameter values dialed in were *hidden_units, num_epoch,* and *strength* related to GAIL. *hidden_units* and *num_epoch* were increased since the autonomous vehicle tasks were complicated, especially when the Agent still had camera sensors. The *strength* of the GAIL signal was dialed down so the Agent would still consider its input but rely more on the extrinsic rewards it obtained. This was per a recommendation in the **Training Configuration File** regarding suboptimal demos. The final demonstration files utilized for training the GAIL network were not representative of every possible state the Agent could exist

in and were created by recording a human "drive" the car, thus they were deemed to potentially be suboptimal.

6. Final Training

After restricting the Agent's observation space and reward signal, the training proceeded in a curriculum learning-esque fashion. Curriculum learning is a form of neural network training that requires training examples to be sorted by difficulty and for the overall problem to be addressed in a series of mini-batches that exhibit increasing levels of difficulty (Hacohen & Weinshall, 2019). This meant breaking down the problem of reaching the goal into two distinct parts. First, the Agent would need to learn to reach the goal on the road without obstacles, and second, the Agent would need to learn to adapt to random placement of obstacles after each run. Two builds of the simulation were subsequently created — one without obstacles and one with.

The restructuring of the observation space and reward functions necessitated new demonstration files to employ GAIL, so the new Agent was trained with extrinsic rewards only in the no-obstacle environment. Since removing the CNN and the absence of the GAIL network yielded reduced complexity, eight environments were trained simultaneously to accelerate the initial training process. While the Agent was training, four demonstration files were created for use with GAIL in the second training environment. These demonstration files included two no-obstacle training sessions and two with-obstacle training sessions for a total of 57,767 steps comprising a total of 178 episodes.

7. Areas for Improvements

While there are several possible avenues to explore improving the created Autonomous Vehicle Agent, three primary areas stand out as good starting points. First, retraining the Agent

from scratch with a proper curriculum learning setup may yield better results. Bengio et al. (2009), who formalized curriculum learning, noted that an appropriately designed curriculum helped the training process of ML models across multiple domains converge faster and on better solutions. Since the manually implemented, and albeit somewhat crude, curriculum strategy was able to overcome some of the initial training problems with the Autonomous Vehicle Agent, it would be worth investigating a proper curriculum set up with any future experiments.

Secondly, introducing a hyperparameter optimization (HPO) algorithm could be beneficial. Tuning hyperparameters can be difficult by hand, and attempting full optimization without a rigid HPO algorithm is much more of an art than a science (Bergstra et al., 2011). An HPO algorithm could help narrow down the best HyperparameterSpec to achieve the fastest convergence and maximal training on subsequent runs.

Thirdly, restructuring the Agent's sensors might also yield better results. While no rigorous testing was done to determine which obstacles the Autonomous Vehicle Agent collided with the most, a cursory exploration (via watching various agent networks drive) indicated that smaller bushes on the ground were difficult for the Agent to detect because the long-range *Ray Perception Sensor 3D* component passed directly over the top of the object. By the time the short-range sensors picked up the obstacle, the Agent's speed was generally too great to avoid a collision, resulting in a restart of the episode. Rather than attempt another restructuring of the Agent's sensors, bushes were removed from the simulation environment. A restructuring of the sensors would most likely be necessary to ensure these objects were detected if placed back into the simulation.

References

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. *Proceedings of the 26th Annual International Conference on Machine Learning*, 41–48. https://doi.org/10.1145/1553374.1553380

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, *24*. https://papers.nips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html

Bowers, L. (2020, May 12). Learning to drive with Unity ML-Agents – A beginners guide to deep RL for autonomous vehicles – Auro. *Auro.Ai*. https://auro.ai/blog/2020/05/learning-to-drive/

Hacohen, G., & Weinshall, D. (2019). On the power of curriculum learning in training deep networks. *Proceedings of the 36th International Conference on Machine Learning*, 2535–2544. https://proceedings.mlr.press/v97/hacohen19a.html

Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. *Advances in Neural Information Processing Systems*, *29*. https://proceedings.neurips.cc/paper/2016/hash/cc7e2b878868cbae992d1fb743995d8f-Abstract.html

Holubar, M. S., & Wiering, M. A. (2020). *Continuous-action reinforcement learning for playing racing games: Comparing SPG to PPO*. https://arxiv.org/abs/2001.05270v1

Huynh, A., Nguyen, B.-T., Nguyen, H.-T., Vu, S., & Nguyen, H. (2021). A method of deep reinforcement learning for simulation of autonomous vehicle control: *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering*, 372–379. https://doi.org/10.5220/0010478903720379

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6292–6299. https://doi.org/10.1109/ICRA.2018.8463162

Probst, P., Boulesteix, A.-L., & Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, *20*, 1–32.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. https://arxiv.org/abs/1707.06347v2

Smith, S. L., Kindermans, P.-J., Ying, C., & Le, Q. V. (2017). *Don't Decay the Learning Rate, Increase the Batch Size*. https://arxiv.org/abs/1711.00489v2

Still, S., & Precup, D. (2012). An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, *131*(3), 139–148. https://doi.org/10.1007/s12064-011-0142-z

Unity ML-Agents Toolkit. (2021). [C#]. Unity Technologies. https://github.com/Unity-Technologies/ml-agents/blob/77d868c90aeb0964134329e4390b216c8e452a6a/docs/Getting-Started.md (Original work published 2017)

Unity Technologies. (2022, February 7). *Unity Manual: Wheel Collider Tutorial*. Unity Documentation. https://docs.unity3d.com/Manual/WheelColliderTutorial.html

Appendix A - Yaml Configuration File

```yaml
env_settings:
  # env_path: ../../Builds/Current_Build_NoGraphics
  num_envs: 8

engine_settings:
  no_graphics: true

behaviors:
  Car:
    trainer_type: ppo

    hyperparameters:
      batch_size: 3000 # Should be larger for continuous actions
      buffer_size: 30000 # larger buff size means more stables training updates
      learning_rate: 3.0e-4
      beta: 5.0e-3
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear

    network_settings:
      vis_encode_type: simple
      normalize: true # Normalization can be helpful in cases with complex continuous control problems so it was enabled
      hidden_units: 128
      num_layers: 2

    reward_signals:

      extrinsic:
        gamma: 0.99
        strength: 1.0

      gail:
        gamma: 0.99
        strength: 0.01 # Set lower since demos are suboptimal
        demo_path: /Users/karlestes/Documents/Grad School/CSC 525 - Principles of Machine Learning/Portfolio Project/AI_Vehicle/Assets/ML-Agents/Demonstrations/
        learning_rate: 1.0e-4
        use_actions: false
        use_vail: true # Enable if imitation learning has some trouble

        network_settings:
          hidden_units: 64
          num_layers: 2

    max_steps: 20000000
    time_horizon: 1500
    summary_freq: 15000
    checkpoint_interval: 500000
    keep_checkpoints: 15
    threaded: false
```