Autonomous Driving at Urban Intersections using Deep Q-learning

Christian Madali
M.S. student in the Khoury College of Computer Science at Northeastern University
madali.c@northeastern.edu
Code: https://github.com/christianmadali/cs5180$_final$

Abstract

In this paper, we propose a decision making algorithm for urban intersection navigation with autonomous vehicles. The algorithm is separated into two parts: a reinforcement learning algorithm and a path planning optimization algorithm. Traffic and the intersection are simulated through the OpenAI Gym highway-env and the problem is formulated as a partially observable markov decision process. Initial results show that the reinforcement learning algorithm has capability navigating the intersection though can be improved with additional iterations and usage of model predictive control.

Introduction

In order to make urban autonomous driving a viable option for urban intersections, a system must be able to deal with several complex factors. It should be able to handle multiple interacting agents, account for unpredictable driving behaviors, consider different road structures, and output commands that consider efficiency and safety.

The most important characteristic of the autonomous driving in urban intersections to consider is the complexity of other drivers. In general, it is difficult to model human behavior. The intentions of each driver are unknown and cannot simply be generalized to the rules of the road. To make matters more interesting, one does not simply consider one unpredictable agent. There are typically multiple cars at an intersection so this problems scales with complexity as the number of cars increase. Another notable point to consider is that not only should the decision making system make decisions that correctly get the agent across the intersection, but also one which makes smooth velocity changes and has a smooth trajectory.

Current solutions for decision-making technology include both classical methods (rule-based, optimization, and probabilistic) and learning-based methods (statistical, deep learning, and reinforcement learning) (Liu et al. 2021). Classical methods though have a notable downside in that many of them have difficulty handling complexities in the driving conditions or environment. Thus, the learning based methods seem to contain a better set of viable solutions for the problem of autonomous driving in urban intersections.

Reinforcement learning is an area of machine learning concerned with how intelligent agents should take actions in an environment in order to maximize a cumulative reward. In reinforcement learning, there is no need for labelled input/output pairs and instead focuses on exploration through trial and error and exploitation of current knowledge. If an agent is properly trained, the learned policy should accomplish complex tasks in relatively short amounts of time.

Using this as motivation, we will apply a set of reinforcement learning algorithms to autonomous driving at urban intersections. Our aim is to implement deep Q-learning as our reinforcement learning method while optimizing the trajectory using model predictive control (MPC) as implemented in previous literature (Tram et al. 2019).

Background

Problem Formulation The purpose of reinforcement learning is for an agent to learn an optimal policy that maximizes some cumulative reward. The agent will update some state-action pairs by interacting with the environment and receiving a penalty or reward. This process is conducted in a loop to iteratively update the state-action pairs as shown in Fig. 1.

Partially Observable Markov Decision Process A partially observable markov decision process, or POMDP, models an agent decision process similar to an MDP, but considers when the agent cannot directly observe the underlying state. Instead it maintains a probability distribution of different observations given the underlying state. Additionally, rather than mapping states to actions as mentioned above, it maps a history of observations or belief states to the actions. Formally a POMPDP is a 7-tuple $(S, A, T, R, \Omega, O, \gamma)$, where

- S is a set of states
- A is a set of actions
- T is a set of transition probabilities between states
- R is the reward function
- Ω is a set of observations
- O is a set of conditional observation probabilities
- γ is the discount factor

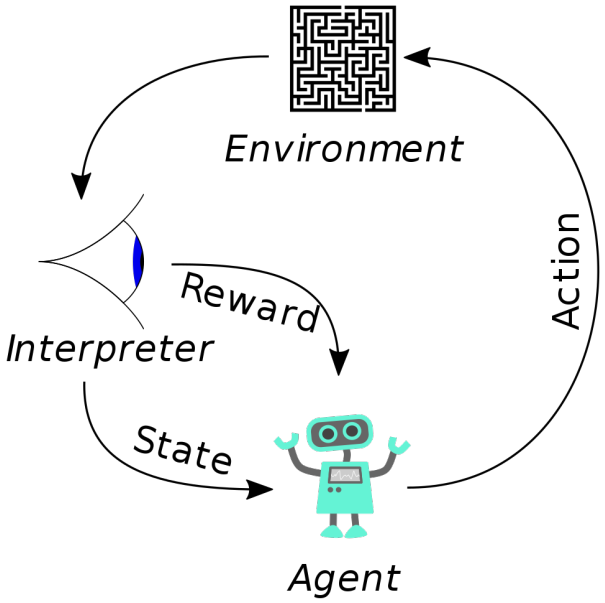

Figure 1: Typical reinforcement learning scenario.

At each time step, the environment in state $s \in S$ has an agent which will take an action $a \in A$ which transitions the environment to $s' \in S$ with probability T. The agent receives an observation $o \in \Omega$ based off the new state s' and action taken with probability O. The agent then receives reward r from the reward function R(s,a).

Q-Learning Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It can handle problems with stochastic transitions and rewards without requiring adaptations. It finals the optimal policy by maximizing the expected value of the total reward over successive steps. The core of this algorithm is a Bellman equation. An example formulation of a simple value iteration update is as follows: With each subsequent

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \Big)}^{\text{temporal difference}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

iteration, the policy should get closer to an optimal action-selection policy. With infinite exploration time and a partly random policy, Q-learning can identify that optimal policy.

Deep Q-learning Deep Q-learning (DQN) builds upon Q-learning as mentioned previously. Though in deep Q-learning, we use a neural network to approximate the Q-value functions. It takes in a state as an input and outputs an approximation of the Q-value for each action based on that state. This is especially useful in larger state spaces where a classic Q-table is not scalable. In the current problem, this is relevant because our state space is continuous.

Experience Replay

One fundamental requirement for stochastic gradient descent (SGD) optimization is that the training data is independent and identically distributed. When the agent interacts with the environment, the sequence of experience tuples can be highly correlated thus a naive Q-learning algorithm can be affected by this correlation. The act of sampling a batch of tuples from the replay buffer to learn is known as experience replay. Experience relay allows us to learn from individual experiences multiple times and breaks the harmful correlation.

Double DQN Double DQN uses two identical neural network models. One learns during the experience replay just like DQN does and the other is a copy of the last episode of the first model. The Q-value is actually computed with the second model.

Model Predictive Control Model predictive control (MPC) is an advanced method of process control that is used to control a process while satisfying a set of constraints. MPC is a multivariable control algorithm that uses an internal dynamic model of the process, a cost function over the receding horizon, and an optimization algorithm to minimize the cost function J using the control input u. In our problem formulation, we use a model free learning algorithm so we do not need an interal dynamic model of the process.

Related Work

As mentioned previously, the solution in this paper is based upon previous work on autonomous driving in urban environments (Tram et al. 2019). Similar work in a similar environment was also done with a higher focus on visual encoding to capture low dimensional states (Chen, Yuan, and Tomizuka 2019). In that literature, other techniques were used such as Twin Delayed Deep Deterministic Policy Gradient (TD3). Though these solutions yielded promising results, it did not account for smoothness of trajectory and were not implemented in this paper.

Problem Description

We describe the problem formulation for teaching an agent to autonomously drive in an urban intersection.

Partially Observable Markov Decision Process We define our problem as a POMDP represented by a 7-tuple $(S, A, T, R, \Omega, O, \gamma)$. We define our state as

$$S = [p^e, v^e, a^e] \tag{1}$$

where p = position, v = velocity, and a = acceleration for each car e. Additionally we define our control input for MPC as

$$u = j^e \tag{2}$$

where j = jerk for car e. We define our action space as

$$A = [0, 4.5, 9] \tag{3}$$

where each number correspond to a speed (e.g. idle, slower, faster). It is worth noting that we operate only in the longitudinal direction and assume there is some other stabilizer for

the lateral direction. For our rewards, we receive a reward of -5 when there is a collision, 1 when successfully crossing the intersection, and 1 when at high speed. We define our observation space as

$$\Omega = [Presence^e, Position_x^e, Position_y^e, Velocity_x^e, Velocity_x^e, cos(h)^e, sin(h)^e] \quad (4)$$

where h is the angle between the controlled car and an approaching car for car e. We defined $\gamma = 0.98$. While it was our intention to apply MPC to this problem formulation, it was difficult to correctly apply to our specific observation and action space and yielded unintelligible results. Thus the results shown in this paper follow the defined problem formulation without the usage of MPC.

Experiments

Our experiments consisted of training on 2000 iterations using DDQN. Originally our algorithm was made using just DQN with experience replay but that proved to be insufficient so additional modifications were made to use DDQN.

Environment Our environment consisted of a simple intersection with crossing cars as shown in Fig. 2. The intersection was allowed to have a maximum of 15 cars with a spawning probability of 0.6. As mentioned above, we trained on 2000 iterations. Each epoch was constructed as a fixed-horizon epoch.

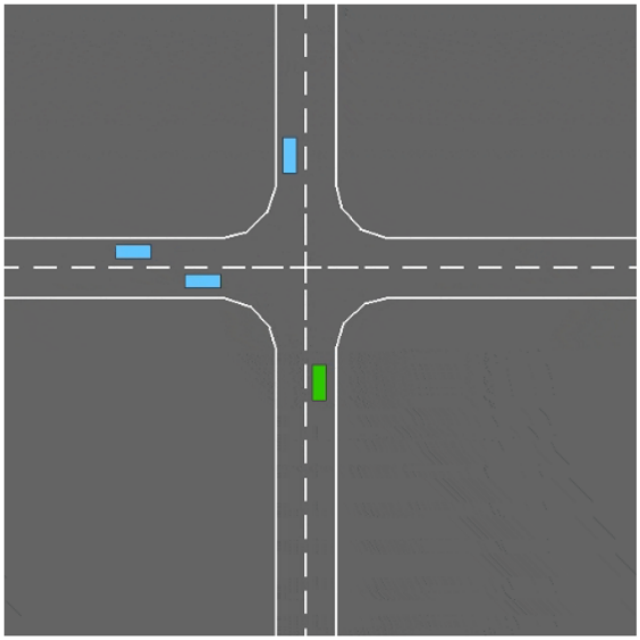

Figure 2: Example of rendered simulation environment.

Simulation The simulation environment used was OpenAI Gym. Specifically the highway environment as created by (Leurent 2018).

Results and Conclusions

We found that over our sweep of hyperparameters and network layers the most promising results are shown in Fig. 3. Initial results were very poor when using a simple DQN algorithm and thus the switch to DDQN was made. The rewards do appear to degrade though we theorize that the rewards would again begin to increase with increased iterations. It is also worth noting that an epoch is defined as a forward and backward pass of all training examples. Additionally we theorize that our results would greatly improve with the use of MPC.

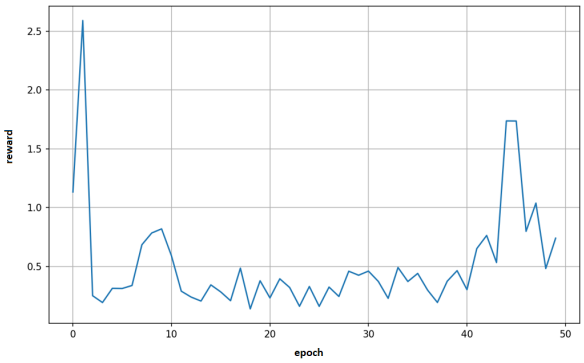

Figure 3: Double DQN reward.

References

Chen, J.; Yuan, B.; and Tomizuka, M. 2019. Model-free deep reinforcement learning for urban autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2765–2771.

Leurent, E. 2018. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env.

Liu, Q.; Li, X.; Yuan, S.; and Li, Z. 2021. Decision-making technology for autonomous vehicles learning-based methods, applications and future outlook. *CoRR* abs/2107.01110.

Tram, T.; Batkovic, I.; Ali, M.; and Sjöberg, J. 2019. Learning when to drive in intersections by combining reinforcement learning and model predictive control. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 3263–3268.