AI Driving Project Memo

Ravi Teja Gadde
Center for Data Science
rtg267@nyu.edu

Rong Zhao
Center for Data Science
rz729@nyu.edu

1 Introduction

In this project, we focus on the end-to-end learning of the lane following task for autonomous driving in a simulator using deep reinforcement learning. Current systems for autonomous driving rely on separately engineered perception, planning and control components which are trained by optimizing intermediate human designed objectives which need not be the ideal way to solve the final objective. We optimize the autonomous driving objective in an end-to-end fashion by taking the raw image from the front camera of the car as input and directly predicting control commands such as velocity and steering direction. We have used the Duckietown simulator to train our agent. The simulator use domain randomization that can help the deep neural network policy to generalize to the real word setting.

2 Reinforcement Learning

Reinforcement learning tries to maximize the expected life-time reward of an agent following a discrete time Markov Decision Process (MDP) defined by (S, A, P, r, γ) where S denote the set of states, A the set of actions, $P(s, a, s^{'})$ the probability of reaching the state $s^{'}$ by taking an action a at state s, $r(s, a)$ the reward obtained by taking an action a at the state s, γ the discount factor emphasizing how much we care about the future. Reinforcement Learning tries to maximize the expected cumulative reward or return $R_t = \sum_{k=0}^{\inf} \gamma^k r(s_{t+k}, a_{t+k})$ of an agent starting from any state s_t by learning a policy $\pi_\theta(a_t|s_t)$, parameterized by θ, denoting the distribution of the possible actions given the current state s_t.

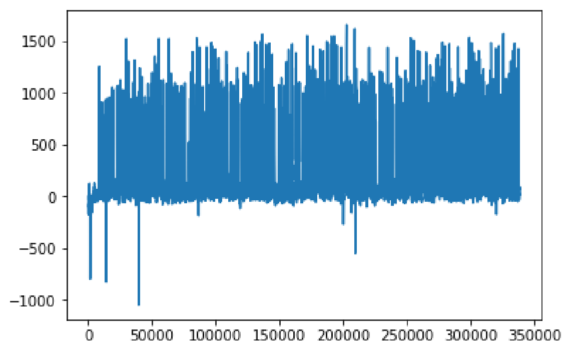
Policy gradient model-free methods update the policy parameters using gradient descent $\theta \leftarrow \theta + \alpha \Delta_\theta J(\theta)$ where,

$$\Delta_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \Delta_\theta log(\pi_\theta(a_t^i|s_t^i)(\sum_{t^{'}=t}^{T} \gamma^{t^{'}-t} r(s_{t^{'}}^i, a_{t^{'}}^i) - b) \tag{1}$$

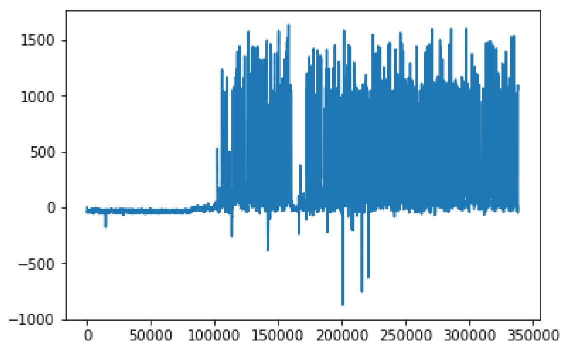
which tries to increase the likelihood of the actions at the current state that gives returns that are better than a baseline b. Actor-Critic methods try to decrease the variance in the policy gradient by learning a critic function $V_\phi^\pi(s_t)$ that tells the actor the direction in which it should update its parameters.

$$\Delta_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \Delta_\theta log(\pi_\theta(a_t^i|s_t^i)(\sum_{t^{'}=t}^{t+n} \gamma^{t^{'}-t} r(s_{t^{'}}^i, a_{t^{'}}^i) + \gamma^n V_\phi^\pi(s_{t+n}) - V_\phi^\pi(s_t)) \tag{2}$$

Preprint. Work in progress.



a) Rewards with the slack parameter



b) Rewards with the original function

Figure 1: Figure showing improvement in training time as result of adding a slack parameter to the reward

3 Simulator

The simulator for the duckietown environment is built on top of the OpenAI Gym environment. The observations received by the agent are single camera RGB images of size $120 \times 160 \times 3$. The action space consists of two continuous actions from -1 to 1 denoting the velocity and the steering angle of the agent.

4 Experiments

We use A2C on a single GPU (effective batch size 1) to train our model. Our action space consist of three discrete set of actions 1. Go straight, 2. Turn left and 3. Turn right. Our rewards take values from [-2, 0.4] and initially, we have clipped all the rewards r to be positive by setting $r = 0$ if $r < 0$, thereby equally penalizing all the actions that give negative rewards by being very strict during training. The agent made less progress as it is hard to explore and reach states that gives positive rewards initially during training since most rewards are 0. However, if we leave the reward function unchanged the agent can spend too much time on exploring states with bad rewards. In order to make the agent learn the policy with out spending too much time on unwanted states and also not being too strict during learning we have added a slack parameter b to the reward function where we clip $(r + b)$ to be positive. This has given significant improvement in the training time of the agent 1. After this we have found that our agent is shaking a lot and we increased the episode length which made our agent stable with out too many changes in the direction.

Our current agent is doing reasonably well in straight road, curved road and intersections. We would like to improve the smoothness of the driving more and also work on lane following with static obstacles.

5 Next Steps

- Predicting continuous actions instead of discrete actions.
- The current agent is late in identifying turns and taking actions. By the time the agent decides to make a right turn its too late and the car cannot make such sharp turns. We would want to add state history, for e.g last n images as input to the agent.
- Extend the existing model to lane following with obstacles on the road.
- Incorporate planning in to the agent for e.g. the agent predicts a sequence of actions rather than the current action.