Learning to Drive using Waypoints

Tanmay Agarwal*, **Hitesh Arora***, **Tanvir Parhar***
Shubhankar Deshpande, Jeff Schneider
*Equal contribution
Carnegie Mellon University
{tanmaya, hiteshar, ptanvir, shubhand, schneide}@cs.cmu.edu

Abstract

Traditional autonomous vehicle pipelines are highly modularized with different subsystems for localization, perception, actor prediction, planning, and control. Though this approach provides ease of interpretation, its generalizability to unseen environments is limited and hand-engineering of numerous parameters is required, especially in the prediction and planning systems. Recently, Deep Reinforcement Learning (DRL) has been shown to learn complex strategic games and perform challenging robotic tasks, which provides an appealing framework for learning to drive. In this paper, we propose an architecture that learns directly from semantically segmented images along with waypoint features to drive within CARLA simulator using the Proximal Policy Optimization (PPO) algorithm. We report significant improvement in performance on the benchmark tasks of driving straight, one turn and navigation with and without dynamic actors.

1 Introduction & Related Work

There has been substantive recent progress towards solving the long standing goal of autonomous driving [20, 16, 10, 7, 4, 1]. This problem ranges in complexity from learning to navigate in constrained industrial settings, to learning to drive on highways, to navigation in dense urban environments. Navigation in dense urban environments requires understanding complex multi agent dynamics including tracking multiple actors across scenes, predicting intent, and adjusting agent behavior conditioned on past history. These factors provide a strong impetus for the need of general learning paradigms that are 'complex' enough to take these factors into account.

Current state of the art systems generally use a variant of supervised learning over large datasets of collected logs to learn driving behavior [4, 1]. These systems typically consists of a modular pipeline with different components responsible for perception, mapping, localization, actor prediction, motion planning, and control [18, 6, 19, 11, 9]. The advantage they offer is the ease of interpretation and ability to optimize subsystem parameters in an understandable way. However in practice, it is extremely hard to tune these subsystems and replicate the intended behavior leading to poor performance in new environments.

Another approach that has recently become popular is exploiting imitation learning where we aim to learn a control policy for driving behavior by observing expert demonstrations [14, 1, 2, 12, 15, 21, 13]. The advantage of these methods are that the agent can be optimized using end to end deep learning to learn the desired control behavior which significantly reduces the effort of tuning each component that is common to more modular systems. The drawback however is that these systems are challenging to scale and generalize to novel situations, since they can never outperform the expert agent and it is impractical to obtain expert demonstrations for the all the scenarios that we care about.

Machine Learning for Autonomous Driving Workshop at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Deep reinforcement learning (DRL) has recently made large strides towards solving sequential decision making problems, including learning to play Atari games and completing complex robotic manipulation tasks. The superhuman performance attained using this learning paradigm motivates the question of whether it could be leveraged to solve the long standing goal of creating autonomous vehicles. This approach of using reinforcement learning has inspired few recent works [3, 7, 10, 8] that learn control policies for navigation task using high dimensional observations like images. The previous approach using DRL [3] reports poor performance in navigation tasks, while the imitation learning based approach [10] achieves better performance but suffers from poor generalizability. Another work [6] only learns the lane following task for a constrained environment and use a sparse reward structure that makes it challenging to learn. Moreover, learning policies from high dimensional state spaces still remains challenging due to the poor sample complexity of most model-free reinforcement learning algorithms.

In this paper, we propose an architecture where we combine the readily available low dimensional trajectory way-points and vehicle pose with higher dimensional latent representation of semantically segmented images that can be used to plan agents' trajectories and control. We discuss our problem formulation and DRL experimental setup, followed by our experimentation methodology in Section 2. We then discuss our results and conclude our work in Section 3.

2 Problem Formulation & Methodology

We formulate our problem of learning to drive using reinforcement learning for which we define our own reward function and environment within CARLA simulator (0.9.6) and train our learning agents using this setup on 4 different driving tasks [3], defined in Section 2.2. We use PPO, an on-policy model-free algorithm, [17, 5] to train our driving agents.

2.1 Reinforcement Learning setup

Our environment consists of our learning agent with other dynamic actors whose control actions are defined by (s, t, b) where s is the steer, t is the throttle and b is the brake action. The s action ranges between [-0.5, 0.5] whereas the t and b actions range between [0.0, 1.0]. We choose top-down semantically segmented (SS) image as one of our state input that is easily obtained from the CARLA's semantic segmentation camera sensor. Given the current state-of-the-art architectures in perception, we believe segmentation as a task can be trained in isolation and hence we focus on learning control agents using DRL directly from SS images. We use convolutional neural network based Auto-Encoder (AE) to reduce dimensionality of our SS image and use the bottleneck embedding as one of the inputs to the agent policy network (Figure 1). We refer to the AE bottleneck embedding as $\tilde{\mathbf{h}}$, and define it in Equation (1) with g being the encoder function of our AE.

$$\tilde{\mathbf{h}} = g(\mathbf{SS}_{\text{image}}) \tag{1}$$

Apart from our chosen sensors' input, the agent also requires an input to guide its navigation. Past approaches [3, 10] have used a higher level planner that directs the agent using high level commands on turning. Instead of this, we propose to use trajectory waypoints to guide navigation, which are readily available in real world autonomous vehicles. Given a source and destination location, waypoints are intermediate locations pre-computed at a fixed resolution (2m) using standard path finding algorithms and can be fetched easily from the CARLA simulator. We believe the features computed from waypoints can provide a richer signal to the learning agent for navigation. The waypoint features $\tilde{\mathbf{w}}$ are computed using some generic function f defined by the next n waypoints $(\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_n})$ and agent's current pose $\mathbf{p}$. These features $\tilde{\mathbf{w}}$ form the second input to our agent policy network as defined in Equation (2).

$$\tilde{\mathbf{w}} = f(\mathbf{p}, \mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_n}) \tag{2}$$

For simplicity, we define the function f as the average angle between the vehicle orientation and the next $n = 5$ waypoints but it can extended to work with any possible functional form. Our chosen function is explained in further detail in Appendix (Figure 2 and Equation (4)).

The input representation is then fed into our policy network $\pi(\hat{s}, \hat{v}|\tilde{\mathbf{h}}.\tilde{\mathbf{w}})$ (Figure 1) which consists of a multi-layer perceptron and outputs $(\hat{s}, \hat{v})$, where $\hat{s}$ is the predicted steer action and $\hat{v}$ is the predicted target velocity for that timestep. To ensure better stability, we utilize a PID controller that computes

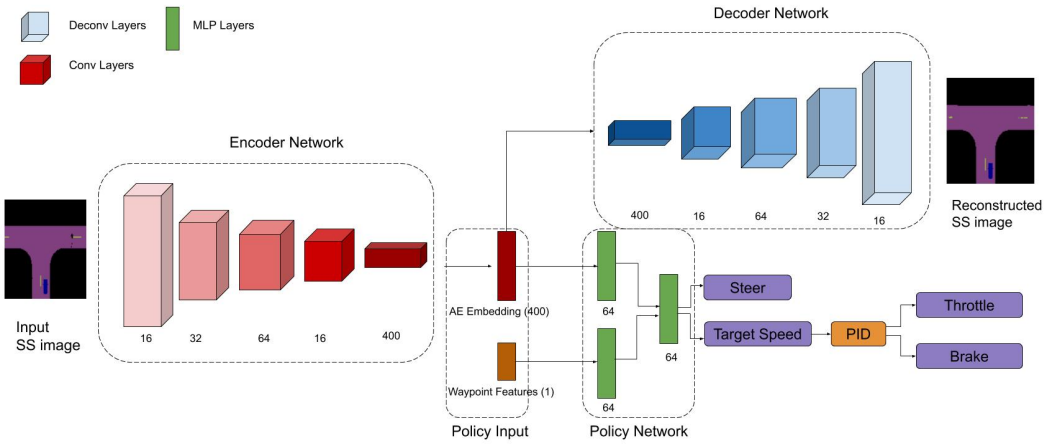

Figure 1: Our Proposed architecture: The inputs to our architecture are semantically segmented (SS) images and intermediate waypoints that we directly get from the CARLA simulator. The SS images are encoded using a pretrained auto-encoder whose bottleneck encoding alongwith waypoint features forms input to the policy network. The policy network outputs the control actions $(\hat{s}, \hat{v})$ where $\hat{s}$ is the predicted steer, $\hat{v}$ is the predicted target speed which is then mapped to predicted throttle and brake $(\hat{t}, \hat{b})$ using a PID controller.

the predicted throttle $\hat{t}$ and brake $\hat{b}$ actions. We also design a simple and dense reward function R that incentivizes our agent R_s based on its current speed u, penalizes R_d based on the perpendicular distance d from the nominal trajectory and incurs R_c if it collides with other actors or goes outside the road, denoted by indicator function $\mathbf{I}(c)$. Mathematically, our reward formulation can be described from Equation 3.

$$\begin{aligned} R &= R_s + R_d + \mathbf{I}(c) * R_c \\ R_s = \alpha * \mathrm{u}; R_d &= -\beta * \mathrm{d}; R_c = -\gamma * \mathrm{u} - \delta \end{aligned} \tag{3}$$

For each of the defined driving tasks, we set up each training episode as goal-directed navigational scenarios, where an agent is initialized at a source location in town and has to reach to a destination location. The episode is terminated as success case if the agent reaches within $10\,\mathrm{m}$ of the destination, while it is terminated as failure case if the agent faces a collision, or doesn't reach near destination within maximum number of timesteps (10,000).

2.2 Experimentation & Methodology

We train and evaluate our agent on four increasingly difficult driving tasks - (a) Straight, (b) One Turn, (c) Navigation and (d) Navigation with dynamic obstacles, which are part of the CARLA [1] benchmark [3]. In our setup, Town 1 is used for training and Town 2 for testing. Since the semantically segmented (SS) images contain a class label per pixel, the convolutional auto-encoder (AE) is trained to predict class label per pixel using reconstruction loss as the multi-class cross-entropy loss. The AE is pretrained on SS images collected using an autonomous oracle agent in the training town to speed up agent policy training. The AE's bottleneck embedding ($\tilde{\mathbf{h}}$) and waypoint features ($\tilde{\mathbf{w}}$) are then fed into the agent policy network which is trained using PPO algorithm.

We found that finetuning AE on the diverse SS images seen during training helps learn a better input representation, enabling the agent to learn a better policy. The agent policy network and AE are trained simultaneously and independently. For a fixed number of timesteps (n_{steps}), the AE is kept fixed and the agent policy is trained on transitions collected during those timesteps. Then the AE is fine-tuned by optimizing on the most recent n_{steps} SS images collected in a buffer, and the training continues. For each of the first three tasks, (a), (b) & (c), we train our agent on all the 25 scenarios respectively by randomly choosing a scenario in each training step while continually testing on all

[1] The existing benchmark suite is on CARLA version 0.8.4 and we ported the same benchmark scenarios for evaluation in CARLA 0.9.6.

Table 1: Quantitative comparison with other state-of-the-art deep reinforcement learning approaches on four goal-directed navigation tasks. The table reports the percentage (%) of successfully completed episodes in each condition. Higher is better. The tested methods are: CARLA RL baseline (CARLA) [3], CIRL [10], and our waypoint based DRL variants **WRL** and **WRL+**.

Task	Training Conditions				New Town				New Weather				New Town/New Weather			
	CARLA	CIRL	WRL	WRL+	CARLA	CIRL	WRL	WRL+	CARLA	CIRL	WRL	WRL+	CARLA	CIRL	WRL	WRL+
Straight	89	98	**100**	**100**	74	100	**100**	**100**	86	100	**100**	**100**	68	98	**100**	**100**
One Turn	34	97	**100**	99	12	71	**100**	99	16	94	**100**	99	20	82	**100**	99
Navigation	14	93	**99**	**99**	3	53	**97**	94	2	86	**99**	**99**	6	68	**97**	94
Navigation Dynamic	7	82	65	**79**	2	41	46	**60**	2	80	65	**79**	4	62	46	**60**

the 25 scenarios for the corresponding tasks. Since the task (d) of navigation with dynamic obstacles is same as the task (c) except the dynamic obstacles, we use the pre-trained agent from task (c) to initialize our agent for task (d). We finally report performance as the percentage of successfully completed episodes.

We experimented two variants of our above approach and setup of training the DRL agent policy. **WRL** refers to the variant in which our PID controller used only throttle to control speed with predicted break action $\hat{b}$ and collision based penalty R_c set to zero. We present another variant of our approach as **WRL+** in which the PID controller predicts both throttle $\hat{t}$ and brake $\hat{b}$ to control speed with the reward function defined in Equation 3. We observe this setup alone does not lead to performance improvement on task (d) for which we add frame-skip (using the same action for consecutive 10 frames) that helps in propagating the affect of each action further in time. Since learning to brake with dynamic actors is challenging, we also pre-train our **WRL+** agent on a simple scenario in which the agent learns to brake by driving on a straight road with two stationary cars blocking the road.

3 Results & Conclusion

To compare our proposed variants **WRL** & **WRL+**, we choose CARLA RL [3] and CIRL [10] as the baselines methods. We acknowledge that both the baseline methods use higher level navigation features and RGB images in contrast to richer low level waypoint features and simpler semantically segmented images used in our approach. The use of waypoint information is motivated from the fact that real-time waypoint and GPS information is readily available in real-world autonomous vehicles and hence can be combined with other visual features for training agents using DRL. Our evaluation benchmarks are more stringent and realistic when compared to the baseline methods as we terminate the episode on collision and count it as a failure case.

To report our reinforcement learning agent's results, we train our DRL agents on all the driving tasks for 3 random seeds and test it on all the 25 scenarios on both Town01 and Town02. For the task (d), we test across 5 random seeds of dynamic actors and report the mean performance in Table 1. Our results show that both of our model variants beat the current state-of-the-art methods using DRL on all the driving tasks when compared with our true reinforcement learning baseline [3]. We also observe similar performance to the CIRL baseline [10] which uses imitation learning as opposed to the reinforcement learning as the training paradigm. We observe that our **WRL+** variant gives more improvement in performance when compared to the **WRL** due to the frame-skip that assists in propagating the affect of each action further in time and also because of pre-traing on a simple scenarios with learning to stop. We also observe that our results give the same performance across different weathers as our input representation is invariant to change in weathers.

We demonstrate that our proposed architecture for learning to drive using the semantically segmented image and waypoint features gives a significant improvement in performance when compared to the existing DRL methods that learn from similar higher dimensional observation spaces. Our proposed approach readily exploits from richer and finer waypoint features and uses it to learn to drive within the CARLA simulator.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Junbo Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[2] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9, 2018.

[3] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio López, and Vladlen Koltun. Carla: An open urban driving simulator. In *CoRL*, 2017.

[4] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *I. J. Robotics Res.*, 32:1231–1237, 2013.

[5] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[6] Takeo Kanade, Charles E. Thorpe, and William Whittaker. Autonomous land vehicle project at cmu. In *ACM Conference on Computer Science*, 1986.

[7] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. *CoRR*, abs/1807.00412, 2018.

[8] Qadeer Khan, Torsten Schön, and Patrick Wenzel. Latent space reinforcement learning for steering angle prediction. *arXiv preprint arXiv:1902.03765*, 2019.

[9] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Sören Kammel, J. Zico Kolter, Dirk Langer, Oliver Pink, Vaughan R. Pratt, Michael Sokolsky, Ganymed Stanek, David Stavens, Alex Teichman, Moritz Werling, and Sebastian Thrun. Towards fully autonomous driving: Systems and algorithms. *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168, 2011.

[10] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric P. Xing. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *ECCV*, 2018.

[11] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Hähnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, Anna Petrovskaya, Mike Pflueger, Ganymed Stanek, David Stavens, Antone Vogt, and Sebastian Thrun. Junior: The stanford entry in the urban challenge. In *The DARPA Urban Challenge*, 2009.

[12] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L. Cun. Off-road obstacle avoidance through end-to-end learning. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 739–746. MIT Press, 2006.

[13] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*, 2, 2017.

[14] Dean Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *NIPS*, 1988.

[15] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep imitative models for flexible inference, planning, and control. *arXiv preprint arXiv:1810.06544*, 2018.

[16] Martin A. Riedmiller, Michael Montemerlo, and Hendrik Dahlkamp. Learning to drive a real car in 20 minutes. *2007 Frontiers in the Convergence of Bioscience and Information Technologies*, pages 645–650, 2007.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[18] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[19] Richard S. Wallace, Anthony Stentz, Charles E. Thorpe, Hans P. Moravec, William Whittaker, and Takeo Kanade. First results in robot road-following. In *IJCAI 1985*, 1985.

[20] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M. Rehg, Byron Boots, and Evangelos A. Theodorou. Information theoretic mpc for model-based reinforcement learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721, 2017.

[21] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

4 Appendix

4.1 Waypoint feature function

We first discuss the waypoint function that is used to compute the waypoint features. Given the next n waypoints $(\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_n})$ and agent's current pose $\mathbf{p}$, we define the function f as the mean waypoint orientation between the agent's current pose and the next n waypoints. Mathematically the function f can be defined by Equation 4 and from Figure 2.

$$\tilde{\mathbf{w}}_\theta = \frac{1}{n} \sum_{i=1}^{n} \left(\theta_{\mathbf{p}} - \theta_{\mathbf{w_i}} \right) \tag{4}$$

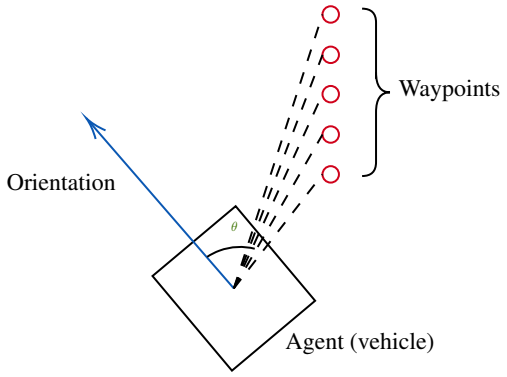

Figure 2: Waypoint orientation that forms our observation space.

4.2 Model Hyper-parameters

To compute the coefficients of the reward function as described in Equation 3, we performed a grid search over different values of α, β, γ & δ. For our experimental setup, we empirically found out $\alpha = \beta = 1$ and $\gamma = \delta = 250$ to be the most stable configuration.

We also present the best setting of hyper-parameters that we used in our entire experimental setup.

Parameter	Value
PPO Learning Rate	2e-4
Auto-encoder Learning Rate	5e-3
Auto-encoder Finetune Learning Rate	1e-4
Hidden Dimension AE	400
Entropy Coefficient	0.005
Frame-skip	10
N-steps	500
Max-steps	10000
No of random seeds	5