Navigating Intersections with Autonomous Vehicles using Deep Reinforcement Learning

David Isele[1], Akansel Cosgun[2], Kaushik Subramanian[3] and Kikuo Fujimura[2]

***Abstract*— Providing an efficient strategy to navigate safely through unsignaled intersections is a difficult task that requires determining the intent of other drivers. We explore the effectiveness of using Deep Reinforcement Learning to handle intersection problems. Combining several recent advances in Deep RL, were we able to learn policies that surpass the performance of a commonly-used heuristic approach in several metrics including task completion time and goal success rate. Our analysis, and the solutions learned by the network point out several short comings of current rule-based methods. The fact that Deep RL policies resulted in collisions, although rarely, combined with the limitations of the policy to generalize well to out-of-sample scenarios suggest a need for further research.**

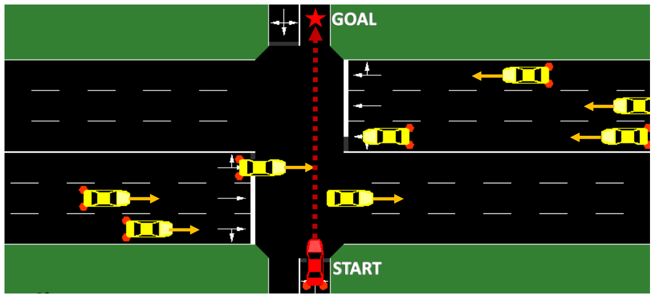


Fig. 1: Crossing a busy intersection. Red car is the autonomous vehicle and yellow cars are oncoming traffic. The objective is to determine the acceleration profile along the path.

I. INTRODUCTION

One of the most challenging problems for autonomous vehicles is to handle unsignaled intersections in urban environments. In order to successfully navigate through an intersection, it is necessary to understand vehicle dynamics, interpret the intent of other drivers, and behave predictably so that other drivers can appropriately respond. Learning this behavior requires optimizing multiple conflicting objectives including safety, efficiency, and minimizing the disruption of traffic. Balancing these trade-offs can be challenging even for human drivers: 20% of all accidents occur at intersections [1]. Acquiring the ability to perform optimally at traffic junctions can both extend the abilities of autonomous agents and increase safety through driver assistance when a human driver is in control.

A number of rule-based strategies have already been applied to intersection handling, including cooperative [2] and heuristic [3] approaches. Cooperative approaches require vehicle-to-vehicle communication and thus not scalable to general intersection handling. The current state of the art is a rule-based method based on time-to-collision (TTC) [4], [5], which is a widely used heuristic as a safety indicator in the automotive industry [6]. Variants of the TTC approach has been used for autonomous driving [7] and the DARPA urban challenge, where hand engineered hierarchical state machines were a popular approach to handle intersections [8], [9]. TTC is currently the method we employ on our autonomous vehicle [10].

While TTC has many benefits - it is relatively reliable, generates behavior that is easy to interpret, and can be tuned to reach a high level of safety - it also has limitations. First, the TTC models assume constant velocity, which ignores nearly all information concerning driver intent. This is problematic: in the DARPA Urban Challenge, one reason behind a collision between two autonomous cars was "failure to anticipate vehicle intent" [11]. Second, in public roads the often unpredictable behavior of human drivers complicates the use of rule-based algorithms. Third, in many cases an agent using TTC may be overly cautious, creating unnecessary delays. These reasons motivate our investigation of machine-learning based approaches for intersection handling in autonomous vehicles.

A number of machine learning based approaches have been used for the intersection handling, such as imitation learning, online planning and offline learning. In imitation learning, the policy is learned from a human drivers [12], however this policy does not offer a solution if the agent finds itself in a state that is not part of the training data. Online planners compute the best action to take by simulating the future states from the current time step. Online planners based on partially observable Monte Carlo Planning (POMCP) have been shown to handle intersections [13], but rely on the existence of an accurate generative model. Offline learning tackles the intersection problem, often by using Markov Decision Processes (MDP) in the back-end [14], [15].

In this paper, we use an offline learning method based on Deep Reinforcement Learning (Deep RL), which to our knowledge hasn't previously been studied for the intersection problem. Given the recent success of Deep Learning on a variety of control problems, [16], [17], [18] we are interested in evaluating the effectiveness of Deep RL in the domain of intersection handling. As seen in Figure 6, the autonomous vehicle is at an unsignaled intersection where the traffic is flowing in both directions. The autonomous vehicle starts from an initially stopped position. The path is assumed to be given by a higher level process, and the proposed approach is tasked to determine the acceleration profile along the path.

[1]David Isele is with The University of Pennsylvania
[2]Akansel Cosgun and Kikuo Fujimura are with Honda Research Institute
[3]Kaushik Subramanian is with the Georgia Institute of Technology

The contributions of this paper are three-fold. The first contribution is the novel way we combine several recent deep learning techniques in order to boost learning speed and improve performance. We use dynamic frame skipping [19] to expedite the system learning repeated actions, and prioritized reply [20] to ensure the network balances learning both positive and negative cases. Additionally, we take advantage of the off-policy nature imposed by experience replay learning to calculate and train on the full n-step return [21] which we found greatly reduces the learning time of DQNs.

The second contribution is our analysis of how well the DQN performs as compared to TTC in five different intersection simulation scenarios, considering safety metrics such as collision rate, and efficiency metrics such as success rate, total time to complete the intersection, and disruption of traffic. We further offer insight on the types of scenarios where DQN could be preferable over TTC. This analysis is of interest beyond the specific choice of DQNs as a base learner as it identifies scenarios where TTC can be improved and suggests methods that could improve it.

The third contribution is the analysis on how well the trained DQN policies transfer to different scenarios. To ensure safety and reliability, any learning system must generalize well to out-of-sample data. To fully appreciate the strengths and failings of our learned networks we run each network on intersection scenarios it did not train on. This analysis allows us to identify when we can and cannot expect a network to succeed in novel situations.

The rest of the paper is organized as follows: In Chapter II we describe the DQN formulation. In Chapter III, we describe the experimental setup and simulation environment. In Chapter IV we discuss the TTC and DQN results, and investigate how DQN policies transfer to different scenarios. In Chapter V, we draw our conclusions pertaining this work.

II. APPROACH

We view intersection handling as a reinforcement learning problem, and use a Deep Q Network (DQN) to learn the state action value Q-function.

A. Reinforcement Learning

In the reinforcement learning framework, at time t an agent in state s_t takes an action a_t according to the policy π. The agent transitions to the state s_{t+1}, and receives a reward r_t. The sequence of states, actions, and rewards is given as a trajectory $\tau = \{(\mathbf{s}_1, a_1, r_1), \ldots, (\mathbf{s}_t, a_t, r_t)\}$ over a horizon T.

A reinforcement learning task is typically formulated as a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the set of states, and $\mathcal{A}$ is the set of actions that the agent may execute. MDPs follow the Markov assumption that the probability of transitioning to a new state given the current state and action is independent of all previous states and actions $p(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t, \ldots, \mathbf{s}_0, a_0) = p(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t)$. The state transition probability $P : \mathcal{S} \times \mathcal{S} \to [0,1]$ describes the systems dynamics, the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ gives the real valued reward for a given time step, and $\gamma \in (0,1]$ is a discount factor that adds preference for earlier rewards and provides stability in the case of infinite time horizons.

The goal of reinforcement learning is to maximize the expected return $R = \sum_{t=0}^{T} \gamma^t r_t$ over a sequence of actions. The expected return for a specific state can similarly be represented $R_t = \sum_{k=0}^{T} \gamma^t r_{t+k}$. We use Q-learning to perform this optimization.

B. Q-learning

In Q-learning [22], the action value function $Q^{\pi}(s,a)$ is the expected return $\mathbb{E}[R_t|s_t = s, a]$ for a state-action pair following a policy π. Given an optimal value function $Q^*(s,a)$ the optimal policy can be inferred by selecting the action with maximum value $\max_a Q^*(s,a)$ at every time step.

In Deep Q-learning [16], the optimal value function is approximated with a neural network $Q^*(s,a) \approx Q(s,a;\theta)$ with parameters θ. The action value function is learned by iteratively minimizing the error between the expected return and the state-action value predicted by the network.

$$\mathcal{L}(\theta) = \Big(\mathbb{E}[R_t|s_t = s, a] - Q(s,a;\theta)\Big)^2 \tag{1}$$

Since in practice the true return is not known, it is often approximated by the one step return

$$\mathbb{E}[R_t|s_t = s, a] \approx r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta) \tag{2}$$

Using the one step return can make learning slow since many updates are required to propagate the reward to the appropriate preceding states and actions. One way to make learning more efficient is to use n-step return[21] $\mathbb{E}[R_t|s_t = s, a] \approx r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_{a_{t+n}} Q(s_{t+n}, a_{t+n}; \theta)$.

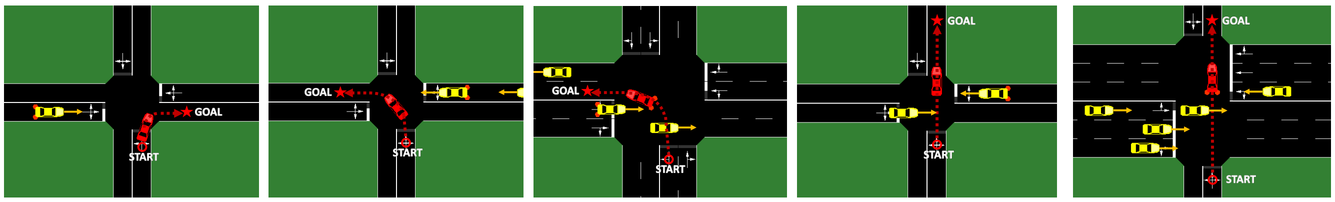
During learning, an ε-greedy policy is followed by selecting a random action with probability ε to promote exploration and otherwise greedily selecting the best action $max_a Q(s,a;\theta)$ according to the current network. In order to improve the effectiveness of the random exploration we make use of dynamic frame skipping.

C. Dynamic Frame Skipping

Frequently the same repeated actions is required over several time steps. It was recently shown that allowing an agent to select actions over extended time periods improves the learning time of an agent [19]. For example, rather than having to explore through trial and error and build up over a series of learning steps that eight time steps is the appropriate amount of time an agent should wait for a car to pass, the agent need only discover that a "wait eight steps" action is appropriate. Dynamic frame skipping can viewed as a simplified version of options [23] which is recently starting to be explored by the Deep RL community. [24], [25], [26].

D. Prioritized Experience Replay

In order to break correlations between sequential steps of the agent, experience replay is used [27]. An experience replay buffer stores previous trajectories which can be sampled during learning. A benefit of using experience replay is that important sequences which happen less frequently



(a) *Right* (b) *Left* (c) *Left2* (d) *Forward* (e) *Challenge*

Fig. 2: Visualizations of different intersection scenarios.

can be preferentially sampled [20]. We use the simplified approach proposed by Jaderberg et al. [24] which avoids the computation of a rank list and instead samples to balancing reward across trajectories.

E. State-Action Representations

Autonomous vehicles use a suite of sensors, and allow for planning at multiple levels of abstraction. This allows for a wide variety of state and action representations. An exploratory search of representations showed the selection of the representation had a significant impact on the agent's ability to learn. In this paper we present the two representations we found to work well.

Sequential Actions In sequential action representation the desired path is provided to the agent and the agent determines to accelerate, decelerate, or maintain constant velocity at every point in time along the desired path.

A birds eye view of space is discretized into a grid in Cartesian coordinates relative to the car's reference frame. This is a representation that could easily be constructed from a car's LiDAR scans. Every car in the space is represented by it's heading angle, velocity, and calculated time to collision. The heading angle, velocity, and calculated time to collision are all represented as real values.

Time-to-Go In the Time-to-Go representation the desired path is provided to the agent and the agent determines the timing of departure through a sequences of actions to wait or go. Every *wait* action is followed by another wait or go decision, meaning every trajectory is a series of wait decisions terminating in a go decision, and the agent is not allowed to wait after the go action has been selected.

A birds eye view of space is discretized into a grid in Cartesian coordinates. Every car in the space is represented by it's heading angle, velocity, and bias term.

The sequential scenario allows for more complex behaviors: the agent could potential slow down half way through the intersection and wait for on coming traffic to pass. The Time-to-Go representation more closely compares to TTC, placing importance on the time of departure. By analyzing the sequential action representation we are able to observe if their is a benefit to allowing more complex behaviors. The Time-to-Go representation focuses on the departure time, allowing us to specifically probe how changes in departure time can affect performance.

III. EXPERIMENTS

We train two different DQNs (Sequential Actions and Time-to-Go) on a variety of intersection scenarios and compare the performance against the heuristic Time-to-Collision (TTC) algorithm.

A. Time-To-Collision (TTC) Policy

The TTC policy serves as a baseline in our analysis. It uses a single threshold to decide when to cross. Below is an explanation of the algorithm. Consider an imaginary line emanating from the front of the ego vehicle, aligned with the longitudinal axis. We calculate the TTC with another vehicle as the time it takes for the vehicle to reach this imaginary line, assuming it will travel with constant speed. Among all the vehicles in the scene, we consider the one with the minimum TTC value. If this value exceeds the TTC threshold, then the ego vehicle starts the crossing phase and follows the Intelligent Driver Model (IDM) [28] until the goal is reached. If it doesn't exceed the threshold, the ego car continues to *wait*.

B. Experimental setup

Experiments were run using the Sumo simulator [29], which is an open source traffic simulation package. This package allows users to model road networks, road signs, traffic lights, a variety of vehicles (including public transportation), pedestrians to simulate traffic conditions in different types of scenarios. Importantly for the purpose of testing and evaluation of autonomous vehicle systems, Sumo provides tools that facilitate online interaction and vehicle control. For any traffic scenario, users can have control over a vehicle's position, velocity, acceleration, steering direction and can simulate motion using basic kinematics models. Traffic scenarios like multi-lane intersections can be setup by defining the road network (lanes and intersections) along with specifications that control traffic conditions. To simulate traffic, users have control over the types of vehicles, road paths, vehicle density, departure times and so on. Traffic cars follow IDM to control their motion. In Sumo, randomness is simulated by varying the speed distribution of the vehicles and by using parameters that control driver imperfection (based on the Krauss stochastic driving model [30]). The simulator runs based on a predefined time interval which controls the length of every step.

We ran experiments using five different intersection scenarios: *Right*, *Left*, *Left2*, *Forward* and a *Challenge*. Each of these scenarios is depicted in Figure 2. The *Right* scenario involves making a right turn, the *Forward* scenario involves crossing the intersection, the *Left* scenario involves making a left turn, the *Left2* scenario involves making a left turn across two lanes, and the *Challenge* scenario involves crossing a six lane intersection with increased traffic density.

Each lane has a 45 miles per hour (20 m/s) max speed. The car begins from a stopped position. Each time step is equal to 0.2 seconds. The max number of step per trial is capped 100 steps which is equivalent to 20 seconds. The traffic density is set by the probability that a vehicle will be emitted randomly per second. We use 0.2 for all scenarios except the challenge scenario where it is set to 0.7.

We evaluate each method according to four metrics and run 10,000 trials of each scenario in order to collect our statistics. The metrics are as follows:

- **Percentage of successes:** the percentage of the runs the car successfully reached the goal. This metric takes into both collisions and time-outs.
- **Percentage of collisions:** a measure of the safety of the method.
- **Average time:** how long it takes a successful trial to run to completion.
- **Average braking time:** the amount of time other cars in the simulator are braking, this can be seen as a measure of how disruptive the autonomous car is to traffic.

For TTC and the Time-to-Go DQN, after the algorithm has decided the path is clear it follows the Intelligent Driver Model. All state representations ignores occlusion, assuming all cars are always visible.

For the sequential action DQN, space is represented as a 5×11 grid discretizing 0 to 20 meters in front of the car and ± 90 meters to the left and right of the car. Each spatial pixel, if occupied, contains the normalized real valued heading angles, velocity, and calculated time to collision. The $5 \times 11 \times 3$ representation results in a 165 dimensional space. Exploratory studies found that this spatial representation outperformed higher dimensional representations with finer granularity. It was also found that real representations of the heading angle, velocity, and time to collision outperformed discretized versions. We hypothesize that this is because the real values allow for greater generalization.

For the Time-to-Go DQN, space is represented as a 18×26 grid in global coordinates. Unlike the sequential action DQN, this representation does not use the calculated time to collision for each car.

The sequential action network is a fully connected networks with leaky ReLU [31] activation functions. The network consists of 3 hidden layers each of 100 nodes each and a final linear layer with 12 outputs. The 12 outputs correspond to three actions (accelerate, decelerate, maintain velocity) at four time scales (1, 2, 4, and 8 time steps).

The Time-to-Go DQN network uses a convolutional neural network with two convolution layers, and one fully connected layer. The first convolutional layer has 32 6×6 filters with stride two, the second convolution layer has 64 3×3 filters with stride 2. The fully connected layer has 100 nodes. All layers use leaky ReLU activation functions. The final linear output layer has five outputs: a single *go* action, and a *wait* action at four time scales (1, 2, 4, and 8 time steps). Both networks are optimized using the RMSProp algorithm [32].

Our experience replay buffers store $100,000$ time steps. We have two buffers, one for collisions and one for both successes and timeouts. At each learning iteration we samples 25 steps from each buffer for a total batch size of 50.

Since the experience replay buffer imposes off-policy learning, we are able to calculate the return for each state-action pair in the trajectory prior to adding each step into the replay buffer. This allows us to train directly on the n-step return and forgo the added complexity of using target networks [33].

Each sequential action scenario was trained on one million simulations. Each Time-to-go scenario was trained on 250 thousand simulations. This difference was in order to roughly balance the runtime of the experiments.

The epsilon governing random exploration was decayed from 1.0 to 0.05 linearly over half the number of iterations.

For the reward we used $+1$ for successfully navigating the intersection, -10 for a collision, and -0.01 step cost.

TABLE I: Comparison of Different Algorithms

Scenario	Metric	*Random*	*TTC*	*DQN-Sequential*	*DQN-Time-to-Go*
Right	% Success	66.06	99.61	99.5	**99.96**
	% Collisions	9.96	**0.0**	0.47	0.04
	Avg. Time	13.2	6.46s	5.47s	**4.63s**
	Avg. Brake	6.0	**0.31s**	0.88s	0.45s
Left	% Success	45.9	99.7	**99.99**	**99.99**
	% Collisions	35.9	**0.0**	**0.00**	0.01
	Avg. Time	13.82s	6.97s	5.26s	**5.24s**
	Avg. Brake	4.51s	0.52s	**0.38s**	0.46s
Left2	% Success	45.45	99.42	99.79	**99.99**
	% Collisions	26.15	**0.0**	0.11	0.01
	Avg. Time	14.48s	7.59s	7.13s	**5.40s**
	Avg. Brake	1.47s	0.21s	0.22s	**0.20s**
Forward	% Success	66.20	**99.91**	99.76	99.78
	% Collisions	17.9	**0.0**	0.14	0.01
	Avg. Time	12.88s	6.19s	**4.40s**	4.63s
	Avg. Brake	4.65s	0.57s	0.61s	**0.48s**
Challenge	% Success	29.99	39.2	82.97	**98.46**
	% Collisions	41.45	**0.0**	1.37	0.84
	Avg. Time	15.7s	12.55s	9.94s	**7.94s**
	Avg. Brake	9.47s	**1.65s**	1.94s	1.98s

IV. RESULTS

Table I shows the results from our experiments. To yield the best results for TTC, TTC threshold for each scenario is chosen as the lowest that achieve zero collisions.

TTC method didn't have a collision in any of the scenarios, given that a large enough threshold is chosen. All other methods had non-zero collision rates for all scenarios, except DQN-Sequential for the *Left* scenario, which also had zero collisions. Among DQN methods, DQN Time-to-Go had substantially lower collision rate than DQN-sequential. For scenarios except *Challenge*, DQN Time-to-Go had only 7 collisions in a total of 40,000 simulations. We think it is

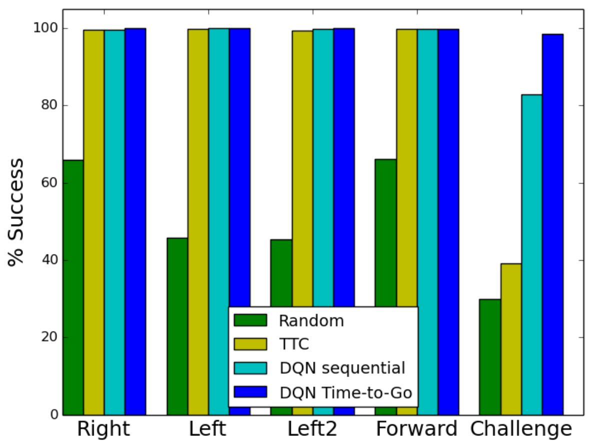


Fig. 3: Comparison of results for all methods and scenarios.

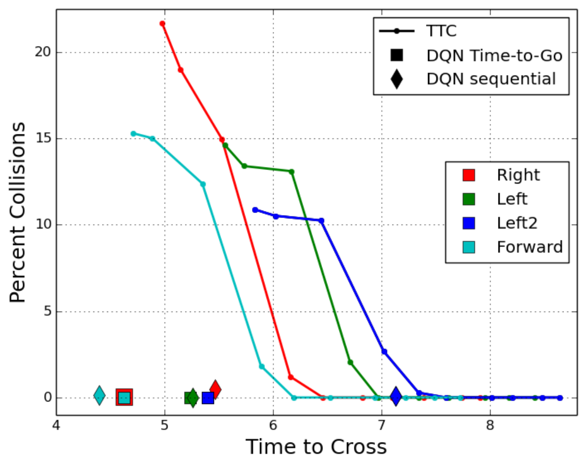


Fig. 4: Trade-off between the time to cross and collision rate as the TTC threshold is varied. Note that performance of the DQN dominates in every case. The challenge scenario is excluded for scale reasons, but the results are similar.

possible that the collision rate would converge to zero for these scenarios if the network is trained for long enough.

We see that both DQN methods are substantially more efficient reaching the goal than TTC. DQN Time-to-Go has the best task completion time in all scenarios, except *Forward*, where DQN-Sequential is faster. On average, DQN Time-to-Go was %28 faster in reaching to goal than TTC, whereas DQN Sequential was %19 faster than TTC. Therefore, the DQN methods have potential to reduce traffic jams due to their efficiency navigating intersection.

Braking time, a measure of how disruptive the car is to other traffic, is comparable for all methods. Because of this, we didn't find conclusive evidence of DQN methods being more aggressive than TTC, despite DQN methods posting non-zero collision rates.

DQN Time-to-Go has the highest success rates in all experiments, except one, where its success rate is only marginally lower than TTC. Both DQN methods and TTC were able to produce sound policies, evidenced by the ego car reaching the goal at least %99.5 of the time for all scenarios except *Challenge*. Success rates are shown in Figure 3.

An interesting result was that TTC did not reach the goal the majority of the time in the *Challenge* scenario, reaching the goal only %39.2 of the time, and posting a success rate only slightly more than Random (%29.9). For the same scenario, DQN Time-to-Go reached the goal %98.46 of the time, significantly outperforming other methods. We offer more insight on these results in Section IV-B.

While the DQNs are substantially more efficient, they are seldom able to minimize the number of collisions as successfully as TTC. This is due to the fact that TTC has a tunable parameter that adjusts the safety margin and we tune TTC to the lowest threshold that gives zero collisions.

Comparing the DQN's performance against the TTC curve as we trade off speed vs. safety (Figure 4), we see that in every instance the DQN's performance dominates the performance of TTC. This suggests that it is possible to design an algorithm that has zero collision rate, but with better performance metrics than TTC.

A. *Generalization and Transfer*

While the DQN does outperform TTC, being able to achieve a zero percent collision rate is very important.

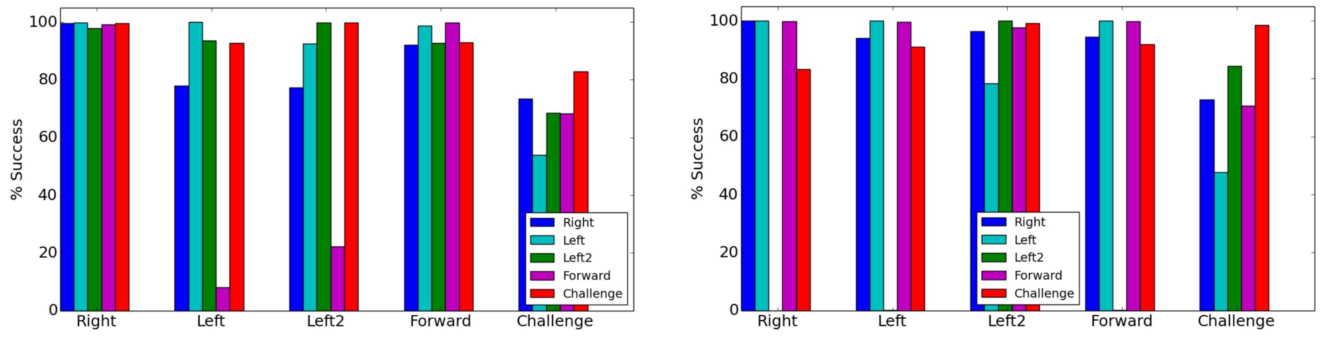
We believe that the few collisions that occur are a symptom of the system's difficulty generalizing. Rather than spending increased effort shaping the reward to drive down the number of collisions, we feel it is more in the service of building a robust system to understand *how* the system generalizes. In order to do this, we run the network trained from one scenario on every other scenario.

We suspect that training on multiple scenarios will improve the performance of each individual task. This is one of the core principles of multi-task learning [34], it has recently been demonstrated specifically on robots learning CNN representations from physical actions [35], and it is the intended direction of our future research. But with this initial study the focus was to get an understanding of how well a deep net system can generalize to out-of-sample data.

Figure 5 shows the transfer performance for both the sequential and Time-to-Go DQNs. We see that the networks trained on the *Left* scenario tend to transfer well to the other single lane scenarios *Right* and *Forward*. The networks trained on the *Challenge* scenario transfer well to the other multi-lane setting *Left2*. Generally speaking, the more challenging scenarios (based on the performance of the *Random* baseline) transfer well to easier domains, but changing the number of lanes creates interference. The Time-to-Go network trained on the *Left2* scenario appears to be an example of over fitting, where the system refuses to move in any of the single lane scenarios. Notably, no method transfers well to all tasks.

B. *Qualitative Analysis*

Comparing trials of the learned DQN networks and TTC, the DQN strategies take into account predictive behavior of the traffic. The DQNs can accurately predict that traffic in distant lanes will have passed by the time the ego car arrives



(a) Transfer Sequential DQN

(b) Transfer Time-to-Go DQN

Fig. 5: Transfer Performance. Networks trained in one scenario are run in different scenarios with to evaluate the generalization of each method.

TABLE II: Transfer Performance for DQN Sequential

Scenario	Metric	Training Method				
		Right	*Left*	*Left2*	*Forward*	*Challenge*
Right	% Success	99.5	**99.7**	97.8	99.2	99.6
	% Collision	0.47	0.21	1.68	0.78	**0.16**
	Avg. Time	5.47s	5.56s	5.12	**4.59s**	5.78s
	Avg. Brake	0.88s	0.72s	0.52	0.56s	**0.39s**
Left	% Success	77.9	**99.9**	93.5	8.19	92.6
	% Collision	16.0	**0.0**	6.30	51.4	7.33
	Avg. Time	10.6s	**5.26s**	7.31s	10.0s	5.28s
	Avg. Brake	3.83s	**0.38s**	1.38s	5.13s	0.53
Left2	% Success	77.3	92.6	**99.7**	22.2	**99.7**
	% Collision	12.1	7.31	**0.11**	16.7	0.27
	Avg. Time	11.3s	**5.41s**	7.13s	12.2s	5.61s
	Avg. Brake	0.66s	0.29s	0.22s	3.15s	**0.14s**
Forward	% Success	92.1	98.8	92.8	**99.7**	92.9
	% Collision	7.86	0.63	6.32	**0.14**	6.81
	Avg. Time	5.17s	5.29s	6.25s	**4.40s**	4.84s
	Avg. Brake	0.98s	0.94s	1.50s	**0.61s**	0.68s
Challenge	% Success	73.5	54.0	68.6	68.3	**82.9**
	% Collision	25.8	30.7	27.5	20.9	**1.37**
	Avg. Time	10.1s	13.0s	9.85s	**8.65s**	9.94s
	Avg. Brake	5.26s	6.62s	4.38s	3.81s	**1.94s**

TABLE III: Transfer Performance for DQN Time-to-Go

Scenario	Metric	Training Method				
		Right	*Left*	*Left2*	*Forward*	*Challenge*
Right	% Success	**99.9**	**99.9**	0.05	99.7	83.3
	% Collision	0.04	0.04	**0.0**	0.22	16.2
	Avg. Time	4.63s	4.87s	6.68s	**4.62s**	6.15s
	Avg. Brake	0.45s	0.34s	1.08s	0.48s	**0.23s**
Left	% Success	94.0	**99.9**	0.06	99.6	91.0
	% Collision	5.89	0.01	**0.0**	0.37	8.82
	Avg. Time	6.07s	5.24s	13.1s	**5.14s**	6.46s
	Avg. Brake	0.51s	0.47s	1.4s	0.52s	**0.37s**
Left2	% Success	96.3	78.3	**99.9**	97.7	99.1
	% Collision	3.70	2.98	**0.01**	1.78	0.87
	Avg. Time	6.33s	10.1s	**5.40s**	6.99s	5.65s
	Avg. Brake	0.29s	0.40s	0.20s	0.29s	**0.18s**
Forward	% Success	94.3	**99.9**	0.06	99.7	91.8
	% Collision	5.54	0.02	**0.0**	0.01	8.0
	Avg. Time	5.46s	4.67s	11.0s	**4.63s**	5.88s
	Avg. Brake	**0.29s**	0.46s	0.46s	0.48s	0.39s
Challenge	% Success	72.8	47.8	84.3	70.7	**98.4**
	% Collision	21.3	13.0	14.3	16.8	**0.84**
	Avg. Time	9.37s	11.4s	8.49s	10.2s	**7.94s**
	Avg. Brake	2.23s	2.16s	**0.46s**	2.22s	1.98s

at the lane. Also the DQN driver is able to anticipate whether on coming traffic will have sufficient time to brake or not. The few collisions seem to relate to discretization effects, where the car nearly misses the on coming traffic.

In contrast, TTC does not leave until all cars have cleared its path. In addition, TTC leaves a sufficient safety margin for on coming cars in distant lanes, since the same safety margin is used the gap gets exaggerated in close lanes. As a result TTC often waits until the road is completely clear, missing many opportunities to cross.

We see that selecting the departure time offers sufficient opportunity to improve over TTC without the need to incorporate the added complexity of allowing for dynamic acceleration and deceleration behavior. This holds even for the *Challenge* scenario. The Time-to-Go DQN often chooses departures when other cars are either in or approaching the intersection, correctly predicting that they will be clear by the time the car reaches that position.

V. CONCLUSIONS

Unsignaled intersection handling remains a hard task for autonomous vehicles, mainly because of unpredictable agent behavior. Rule-based intersection handling methods offer reliable and easy-to-interpret solutions, however result in sub-optimal behavior and task performance.

We showed a first system that uses Deep Q-Networks for the specific problem of intersection handling. By making use of the latest Deep RL techniques, we were able to build networks that, in some metrics, outperform a commonly used rule-based algorithm based on the Time-to-Collision (TTC) heuristic. While TTC achieved zero collision rate for all cases, DQN performed better on task efficiency and success rate. Although rarely, DQN methods caused collisions and thus is unsuitable for real-world implementation in its current form. Therefore, more investigation on DQN's is necessary to reduce the collision rate to zero.

We saw that determining the time to go is the most im-

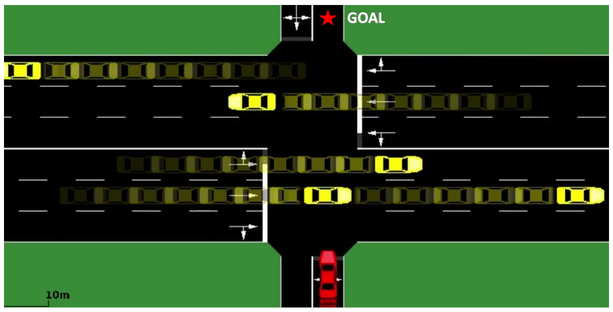


Fig. 6: DQN Time-to-Go predicts the opening and begins accelerating in anticipation of the clear path. TTC would have waited until all cars were clear, missing the opportunity.

portant part of the task, even in more complex environments when the ability to change speeds through the intersection might be beneficial. This can greatly reduce the complexity of the decision task. We also identified specific instances when TTC has difficulty.

While the networks demonstrate some ability to generalize to novel domains and out-of-sample data, more research is required to increase the robustness. Training one network for one scenario took almost a day on a modern multi-GPU computer, which made it impractical for network parameter tuning. As future work, we will investigate how the policy learned from one type of intersection scenario could be used as an initial policy for another scenario, building up a system that performs well in wide variety of domains.

REFERENCES

[1] National Highway Traffic Safety Administration, "Traffic Safety Facts, Tech. Rep. DOT HS 812 261, 2014. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/Publication/812261

[2] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio, "Cooperative collision avoidance at intersections: Algorithms and experiments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1162–1175, 2013.

[3] J. Alonso, V. Milanés, J. Pérez, E. Onieva, C. González, and T. De Pedro, "Autonomous vehicle control systems for safe crossroads," *Transportation research part C: emerging technologies*, vol. 19, no. 6, pp. 1095–1110, 2011.

[4] M. M. Minderhoud and P. H. Bovy, "Extended time-to-collision measures for road traffic safety assessment," *Accident Analysis & Prevention*, vol. 33, no. 1, pp. 89–97, 2001.

[5] R. van der Horst and J. Hogema, *Time-to-collision and collision avoidance systems*. na, 1993.

[6] K. Vogel, "A comparison of headway and time to collision as safety indicators," *Accident analysis & prevention*, vol. 35, no. 3, pp. 427–433, 2003.

[7] D. Ferguson, C. Baker, M. Likhachev, and J. Dolan, "A reasoning framework for autonomous urban driving," in *Intelligent Vehicles Symposium, 2008 IEEE*. IEEE, 2008, pp. 775–780.

[8] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schöder, M. Thuy, M. Goebl, F. von Hundelshausen, *et al.*, "Team AnnieWays autonomous system for the DARPA urban challenge 2007," in *The DARPA Urban Challenge*. Springer, 2009, pp. 359–391.

[9] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[10] A. Cosgun, L. Ma, J. Chiu, J. Huang, M. Demir, A. M. Anon, T. Lian, H. Tafish, and S. Al-Stouhi, "Towards full automated drive in urban environments: A demonstration in gomentum station, california," *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[11] L. Fletcher, S. Teller, E. Olson, D. Moore, Y. Kuwata, J. How, J. Leonard, I. Miller, M. Campbell, D. Huttenlocher, *et al.*, "The MIT–Cornell collision and why it happened," *Journal of Field Robotics*, vol. 25, no. 10, pp. 775–807, 2008.

[12] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[13] M. Bouton, A. Cosgun, and M. J. Kochenderfer, "Belief state planning for navigating urban intersections," *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[14] W. Song, G. Xiong, and H. Chen, "Intention-aware autonomous driving decision-making in an uncontrolled intersection," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[15] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps," in *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*. IEEE, 2014, pp. 392–399.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[17] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *arXiv preprint arXiv:1603.02199*, 2016.

[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[19] A. Srinivas, S. Sharma, and B. Ravindran, "Dynamic action repetition for deep reinforcement learning," *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[21] J. Peng and R. J. Williams, "Incremental multi-step q-learning," *Machine learning*, vol. 22, no. 1-3, pp. 283–290, 1996.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[24] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[25] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," *arXiv preprint arXiv:1604.07255*, 2016.

[26] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in Neural Information Processing Systems*, 2016, pp. 3675–3683.

[27] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[28] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, no. 2, p. 1805, 2000.

[29] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO–simulation of urban mobility," *International Journal on Advances in Systems and Measurements (IARIA)*, vol. 5, no. 3–4, 2012.

[30] S. Krauss, "Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics," Ph.D. dissertation, Deutsches Zentrum fuer Luft-und Raumfahrt, 1998.

[31] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, no. 1, 2013.

[32] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop, coursera: Neural networks for machine learning," *University of Toronto, Tech. Rep*, 2012.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[34] R. Caruana, "Multitask Learning," *Machine Learning*, vol. 28, pp. 41–75, 1997.

[35] L. Pinto and A. Gupta, "Learning to push by grasping: Using multiple tasks for effective learning," *arXiv preprint arXiv:1609.09025*, 2016.