Towards Learning Multi-agent Negotiations via Self-Play

Yichuan Charlie Tang
Apple Inc.
yichuan_tang@apple.com

Abstract

Making sophisticated, robust, and safe sequential decisions is at the heart of intelligent systems. This is especially critical for planning in complex multi-agent environments, where agents need to anticipate other agents' intentions and possible future actions. Traditional methods formulate the problem as a Markov Decision Process, but the solutions often rely on various assumptions and become brittle when presented with corner cases. In contrast, deep reinforcement learning (Deep RL) has been very effective at finding policies by simultaneously exploring, interacting, and learning from environments. Leveraging the powerful Deep RL paradigm, we demonstrate that an iterative procedure of self-play can create progressively more diverse environments, leading to the learning of sophisticated and robust multi-agent policies. We demonstrate this in a challenging multi-agent simulation of merging traffic, where agents must interact and negotiate with others in order to successfully merge on or off the road. While the environment starts off simple, we increase its complexity by iteratively adding an increasingly diverse set of agents to the agent "zoo" as training progresses. Qualitatively, we find that through self-play, our policies automatically learn interesting behaviors such as defensive driving, overtaking, yielding, and the use of signal lights to communicate intentions to other agents. In addition, quantitatively, we show a dramatic improvement of the success rate of merging maneuvers from 63% to over 98%.

1. Introduction

One of the key challenges for building intelligent systems is learning to make safe and robust sequential decisions in complex environments. In a multi-agent setting, we must learn sophisticated policies with negotiation skills in order to accomplish our goals. Fig. 1 provides an example of this in the domain of autonomous driving, where we wish to learn a policy to safely merge onto the highway in the presence of other agents. Merges are considered complex [33, 5, 26], where behavior planning must accurately predict each other's intentions and try to act rationally in a general sum game [2]. Traditional solutions often tackle these Markov Decision Processes (MDPs) by making many assumptions and engineering hard-coded behaviors, often leading to constrained and brittle policies [13, 21, 9]. For example, in lane changes and merges, a typical approach would first check for a sufficient gap in the adjacent lane (high-level behavior planning), followed by solving for the optimal future trajectory (low-level motion planning). While this is a sensible approach for the majority of situations, it quickly becomes hard to handle various corner cases: e.g. multiple vehicles are simultaneously trying to merge to the same lane. Moreover, it is difficult to reason about how others will react to our own actions and how we might react to their reactions and so forth.

Figure 1: *A merge scenario: the green and red vehicles want to find a gap and merge on, while some of the blue vehicles want to merge off. For robust decision making, it is critical to negotiate with other vehicles on the road.*

In a drastically different approach to solving the problem, reinforcement learning (RL) can directly learn policies through repeated interactions with its environment. Ideally, the simulated environment should be diverse and re-

alistic to facilitate real world transfers. The policy function is usually a general purpose deep neural network, making no *a priori* assumptions about the possible solutions. Recently, with the increase in computation power and methodological improvements, deep reinforcement learning (deep RL) has been applied to a variety of MDP problems and has achieved striking breakthroughs: superhuman performances in video games [19], Go [29], and most recently in StarCraft [32]. However, it is unclear whether or not these techniques can be easily transferred to driving applications. Unlike in games, driving is not a zero-sum game and involves partially observed stochastic environments, where safety and following traffic laws are paramount.

In this paper, we leverage deep RL and propose an iterative *self-play* training scheme to learn robust and capable policies for a multi-agent merge scenario in a simulated traffic environment. We first build a simulation of traffic based on real road geometry, where the road network is annotated by aligning with a real "zipper" merge from satellite imagery, see Fig. 1. We populate the world densely with rule-based agents that are capable of lane-following and safe lane changes. However, it quickly becomes apparent that simple rule-based policies are insufficient to deal with all of the complexities of the environment. We can do better by using RL to train policies in the presence of the basic rule-based agents. However, the RL policy easily overfits to the distribution of behaviors of the basic rule-based agents.

To overcome this, we devise an iterative self-play algorithm where previously trained RL policies are mixed with rule-based agents to create a diverse population of agents. In self-play, the policies being learned are also simultaneously used to control agents in the simulation. As training progresses, our agents' capabilities also evolve, increasing the diversity and complexity of the environment. In such an complex environment, a capable policy must learn when to slow down, when to accelerate and pick the gap to merge into. It must also learn to communicate intentions to other agents via turn signals or through its observable behaviors. Lastly, it needs to estimate the latent goals and beliefs of the other agents. Towards learning such policies, we demonstrate the qualitative and quantitative behaviors or our self-play trained policies in Sec. 6.

While we start with only a particular merge scenario, our underlying motivation is that once we learn good policies for all of the challenging scenarios, driving from any location A to any other location B can be 'stitched' together from these scenario specific policies, much like how options are used in hierarchical reinforcement learning. Our main contributions are as follows:

- A fast 2D simulation environment with realistic topology and a kinematics Bicycle motion model.
- Rule-based agents with basic driving abilities: avoiding collisions, making safe lane changes, using turn signal lights.
- Model-free RL policy learning from a combination of rasterized state encodings as well as low-dimensional state variables.
- Quantitatively, we demonstrate that we can improve the rate of successful merges from 63% to 98% via our proposed self-play strategy.
- Qualitatively, our self-play trained agents learned human like behaviors such as defensive driving, yielding, overtaking, and using turn signals.

2. Related Works

Using reinforcement learning algorithms to solve multi-agent systems is useful in a wide variety of domains, including robotics, computational economics, operations research, and autonomous driving. A comprehensive overview and survey on existing multi-agent reinforcement learning (MARL) algorithms is provided by [2]. More recently, an increase of interest in MARL has come from the perspective of achieving general intelligence, where agents must learn to interact and communicate with each other in a shared environment [16, 8]. Game theoretic approaches for MARL include fictitious play [10] and fictitious self-play, the latter of which has been recently proposed with deep neural networks for imperfect information poker games and shown to converge to approximate Nash equilibria [11].

Policy learning for the driving domain can be formulated as a MARL problem where each agent is a vehicle and the environment is the road scene. The most challenging aspect of this application domain lies in the ability to be robust and safe [27]. To this end, a method was proposed to improve functional safety by decomposing the policy function into two components, a learnable function which is gated by a hard constraint, a non-learnable trajectory planner [26].

There have also been previous work directly applying deep reinforcement learning to driving simulations. Deterministic actor critic [17] has been previously applied to control the steering and acceleration in a racing game [35], where the policy mapped pixels to actions. As part of the CARLA simulator, the authors also released a set of deep RL baselines results [6]. Deep RL was utilized for an end-to-end lane keep assist system [23], and also used for navigating a roundabout intersection [3]. For related but non-RL approaches, conditional imitation learning agents have also been learned via behavior cloning from expert data [4].

3. Preliminary

Before diving into the details of our algorithms and framework, we first provide a background on deep reinforcement learning and its approach to learning optimal policies. We will use the *Markov Decision Process* (MDP) framework to model our problem [22]. Reinforcement

learning consists of a type of algorithm for solving MDPs in which the agent repeatedly interacts with a stochastic environment by executing different actions. The MDP consists of a state space $\mathcal{S}$, an action space A, and a reward function $r(s,a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The model of the environment is: $p(s'|s,a)$ which specifies the probability of transitioning to state s' from state s and executing action a. The policy function $\pi_\theta(a|s)$, parameterized by θ, specifies the distribution over actions a given a state s. $\rho^\pi(s)$ denotes the stationary distribution over the state space given that policy π is followed. We denote the total discounted reward as $r_t^\gamma = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)$, where $\gamma \in [0.0, 1.0]$ is the discount factor. The value function is $V^\pi(s) = \mathbb{E}[r_1^\gamma | S_1 = s, \pi]$ and the state-action value function is $Q^\pi(s,a) = \mathbb{E}[r_1^\gamma | S_1 = s, A_1 = a, \pi]$. Reinforcement learning (RL) consists of class of algorithms which can be used to find the optimal policy for MDPs [30]. RL algorithms seek to find the policy (via θ) which maximizes the average expected total discount reward.

3.1. Policy Gradients

Policy gradient methods directly optimize the policy parameters θ and are a popular type of algorithm for continuous control. They directly optimize the expected average reward function by finding the gradient of the policy function parameters. The objective function can be written as:

$$J(\pi_\theta) = \int_S \rho^\pi(s) \int_A \pi_\theta(a|s) Q^\pi(a,s) da ds \qquad (1)$$

[24] showed that the gradient can also be in the form of: $\nabla_\theta J = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)]$, where A is the advantage function. A can also be replaced by other terms such as the total return, which leads to the REINFORCE algorithm [34]. In addition, based on the Policy Gradient Theorem, a widely used architecture known as the *actor-critic* replaces A with a critic function, which can also be learned via temporal difference and guides the training of the actor, or the policy function [31].

3.2. Proximal Policy Optimization

Proximal Policy Optimization (PPO) [25] is an online policy gradient method which minimizes a new surrogate objective function using stochastic gradient descent. Compared to traditional policy gradient algorithms where one typically performs one gradient update per data sample, PPO is able to achieve more stable results at lower sample complexity. The surrogate objective is maximized while penalizing large changes to the policy. Let $r_t(\theta)$ be the ratio of probability of the new policy and the old policy: $r_t(\theta) = \frac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)}$, then PPO optimizes the objective:

$$L(\theta) = \hat{E}_t \Big[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \Big] \qquad (2)$$

Figure 2: *Multi-agent zipper-merge simulation environment. Agents are randomly spawned at lane segments (A, B, or C) with a goal of arriving at a randomly chosen goal location (D, E, or F). Agents must also obey traffic laws and stay on the road.*

where A_t is the estimated advantage function and ϵ is a hyperparameter (e.g. $\epsilon = 0.2$). The algorithm alternates between sampling multiple trajectories from the policy and performing several epochs of stochastic gradient descent (SGD) on the sampled dataset to optimize this surrogate objective. Since the state value function is also simultaneously approximated, the error for the value function approximation is also added to the surrogate objective to form the overall objective.

Online model-free algorithms such as PPO do not make use of the experience replay buffer and it is beneficial to collect experiences in parallel to reduce the variance of the gradient updates. This can be easily done by simultaneously launching N agents in parallel and collect the experiences into a minibatch and update using the average gradient. While we found that PPO fairly stable and efficient, there are certainly various other alternative algorithms that one could use instead [17, 12, 7]. We leave experimenting with different underlying deep RL learning algorithms to future work.

4. Environment

We begin by first describing our simulation environment, as all of the subsequent discussions can be better understood in the context of the environmental details. Fig. 2 shows our multi-agent zipper merge RL environment, where agents

Scale (m)	Init vel.	IDM desired vel.	# Other agents	Spawn pr.	$R_{success}$	$R_{collision}$	$R_{out-of-bounds}$	$R_{velocity}$
340	[0, 5](m/s)	[10, 20] (m/s)	[0, 10]	1%	100.0	−500.0	−250.0	0.1

Table 1: *Zipper merge environment details. Bracketed values are the [min, max] bounds from which we randomly sample.*

start at one of (A, B, C) locations with the goal of getting to one of (D, E, F) destinations. Both starting and destination locations are chosen at random at the start of each episode. The action space of the environment consists of acceleration, steering, and turn signals.

Zipper merges, also known as *double merges*, are commonly recognized as a very challenging scenario for autonomous agents [26]. It is challenging as the some of the agents on the left lane intend to merge right while most of the right lane agents need to merge left. Signals and subtle cues are used to negotiate who goes first and which gap is filled. The planning also has to be done in a short amount of time and and within a short distance.

To capture the geometry of real roads, we used a satellite image as a reference and annotated polyline-based lanes to match approximately the different lanes and curvature of the road (white lines in Fig. 2). Our simulation is in 2D, which is a reasonable assumption to make as we are more interested in high-level negotiations as opposed to low-level dynamic vehicle control. The discretization of our simulation is at 100 milliseconds and the frame rate is 10 Hz. The state of every agent is updated in a synchronous manner. Tab. 1 details the parameters of our learning environment.

4.1. Dynamics

We use the discrete time Kinematics Bicycle model [15] to model vehicle dynamics in our environments. The nonlinear continuous time equations that describe the dynamics in an inertial frame are given by:

$$\dot{x} = v\cos(\psi + \beta), \quad \dot{y} = v\sin(\psi + \beta) \tag{3}$$

$$\dot{\psi} = \frac{v}{l_r}\sin(\beta), \quad \dot{v} = a \tag{4}$$

where x, y are the coordinate of the center-of-mass of the vehicle, ψ is the inertial heading and v is the velocity of the vehicle. l_r and l_f are the distance of the center of the mass to the front and the rear axles. β is the angle of the current velocity relative to longitudinal axis of the vehicle. In an effort to be realistic, we bound the acceleration to be between $[-6 \text{ to } 4]$ m/s^2.

4.2. Road Network

The road network of our environment consists of a 2D graph of straight, curved, and polyline lanes. Straight lanes are modeled by a left boundary line and a right boundary line. Curved lanes are modeled using Clothoids [18] with a specified lane width. Finally, arbitrary shaped lanes are modeled by separate left and right lane boundaries. In our 2D graph, each lane is a node is this graph and connected by incoming and outgoing edges. The direction of the edge indicates the direction of travel.

4.3. IDM Agents

As part of the environment, we have rule-based agents of differing "intelligence". Also known as intelligent driving models (IDMs) [14], the simplest agents perform lane-keeping starting from a specified lane using adaptive cruise control (ACC): slowing down and speeding up accordingly with respect to the vehicle in front. Building on top of these simple ACC agents, we add lane change functionality by using gap acceptance methods [13], so that merge can be done in a safer manner. We also vary the desired and starting velocities and accelerations to create a population of different IDM agents (they will drive a little differently from each other). To differentiate between the IDM agents, we will refer to the agent/vehicle that we are learning to control as the *ego* vehicle.

5. Model Architecture

In this section, we describe in detail our deep RL framework for self-play, which is used to learn sensible negotiation policies in the aforementioned zipper merge environment. We will explain the architecture of the policy network, the observation space and the cost function used for learning. At the core, our learning is a distributed form of the PPO policy gradient algorithm, where multiple environments are simulated in parallel to collect experiences. After first training a single deep RL agent with an environment populated with other rule-based IDM agents, we initiate self-play by replacing a portion of the agents with previously learned RL agents. As self-play training progresses, we iteratively add more RL agents with updated parameters to the training agent population.

5.1. Observations and State Encodings

The road geometry, state of other agents, and the goal/destination of the ego vehicle are all important for decision making. The observations from our RL agent's perspective consist of a combination of two components. The first is a top down rasterization view in ego-centric frame: see examples of this rasterization in Fig. 3 (top left). This rasterization captures the lane geometry, the route information, and the poses of all vehicles, essentially capturing the context of the scene. The rasterization is an RGB image of $128 \times 128 \times 3$. Rasterization is a simple encoding scheme

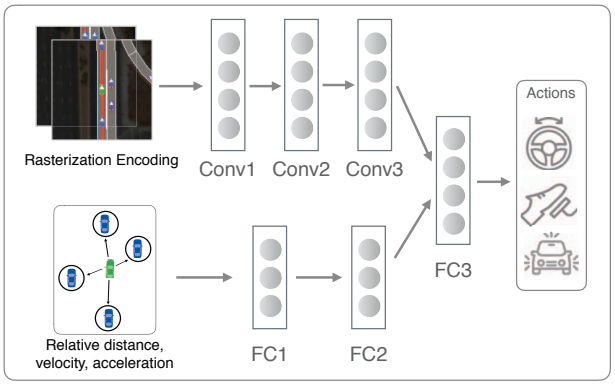


Figure 3: *Architecture diagram of the policy function. Inputs consist of two types. The first is a rasterized top-down image of the scene, with the ego vehicle in the center. The second is the low-dimensional precise measurements of relative distances, velocities and accelerations of other vehicles with respect to the ego vehicle. These inputs are passed through convolutional and fully connected layers to generate three actions: steering, acceleration and turn signals.*

that renders the scene into a sequences of image frames. The rasterized input contains all of the geometric relationships in metric space. As we will discuss later, state encoding will then be extracted by a convolutional neural network (CNN). We use OpenGL for efficient rasterization. Besides rasterization, we also use a 76 dimensional vector which is composed of the position, velocity, acceleration, orientation, and turn signal states of the 8 closest neighboring vehicles to ego. In the real world, these inputs can be given by the output of object tracking. We also encode ego's ideal reference route and provide it as a part of the 76 dimensional vector. Temporal history of two frames (200 ms) is used to provide temporal contextual information.

5.2. Policy Network

The policy function (Fig. 3) is represented by a deep neural network (CNN) which takes the state encoding of the environment (vehicle poses, lanes, etc.) as input and outputs steering, acceleration and turn signal commands for the ego vehicle. Our policy function is "end-to-end" in that it bypasses traditional trajectory planning and feedback control. Our policy has two streams of inputs, which are combined via late fusion at the third fully-connected (FC3) layer. The two streams are complementary to each other as rasterization provides spatial context but it is discretized and lossy. The low dimensional relative measurements are highly precise but do not capture road information. A 3-layer convolution neural network is used to process the rasterization stream. The layers of the CNN have a receptive field size of 4 with a stride of 2 to process the input channels. After 3 convolution layers, a fully connected layer embeds the scene into a 128 dimensional vector. In the other stream, two fully connected layers of dimension 64 encode the 76 dimensional input. The output of the two streams are concatenated into a single layer and an additional fully connected hidden layer is used to predict the actions. The action space is 3 dimensional: steering, acceleration, and turn signals.

5.3. Reward Function

The reward function, or cost function, is critical for guiding the search for the desired optimal policy. We define collision as the intersection of the bounding boxes of two vehicles, with a -500.0 reward. A vehicle goes out-of-bounds when its centroid is more than $0.75\times$ the lane width from the center-line of any lane. The cost incurred for out-of-bounds is -250.0. Successfully completing the zipper merge gains a $+100.0$ reward. There are also penalties of -0.1 whenever the turn signal is turned on to prevent the policy from constantly having the turn signal lights on. Vehicle velocity up to the speed of 15 m/s is rewarded with a scaling of 0.1. In order to reward a vehicle for remaining in the center of its lane, a lane center offset penalty of -0.1 is applied for every meter off the center line. Finally, we also penalize for non-smooth motion by a penalty of -2.0 multiplied with the change in steering angle.

It is also critical to introduce reward shaping [20] to make learning easier. Specifically, we linearly anneal the crash and out-of-bounds loss from -100.0 to their final value of -500.0 and -250.0, respectively, as training progress from 0 to 1000 updates. These hyperparameters are selected manually to ensure sensible performance. A better approach would be to find these values from real data, we leave this to future work.

5.4. Training

We began by training a single RL policy using PPO with a batch size of 32, learning rate of 0.0025, entropy coefficient of 0.001, and the number of environment steps between parameter updates is 1024. Distributional learning uses 32 separate processes to step through 32 environments in parallel. For each episode, roughly up to 10 agents are launched, each having their own random destination location. An episode is terminated when either one of several situations occurs: destination is reached, collision, ego going out of bounds, or after 1000 timesteps.

5.5. Multi-agent Self-play

Reinforcement learned policies are well known for their ability to exploit the training environment. In the case of our multi-agent environment, it is prone to overfit to the specific behaviors of the "sparring" IDM agents. For example, if the IDM agents have a tendency to brake suddenly and always yield to another merging vehicle, then the RL policy will learn to exploit this by being ultra aggressive. However, this

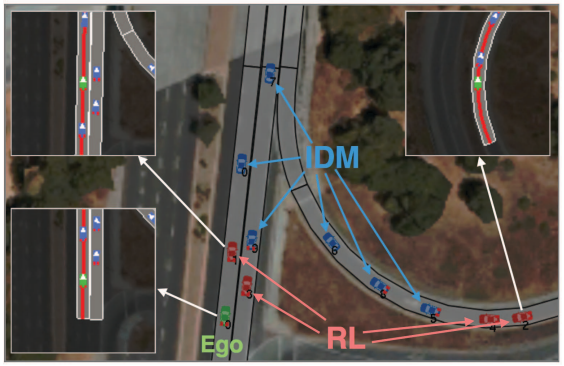


Figure 4: *Training during self-play. For ego (green), the sparring agent population consists of two types of agents, the IDM agents (blue) and other RL agents (red).*

could be dangerous in environments with agents that do not always yield. Additionally, it is suboptimal to train only with IDM agents as it is hard to manually tune IDM agents' parameters. For example, we noticed that IDM agents are to blame in most collisions involving IDM and RL agents.

Self-play is a very broad and general technique for increasing the complexity of the environment and learning better policies [1, 28, 11]. At a high level, self-play iteratively adds the latest of the learned policies back to the environment, increasing the diversity and perhaps realism of the environment. For our application, starting from a simple road network and a handful of rule-based IDM agents, self-play allows us to learn complex policies capable of a diverse set of behaviors.

Fig. 4 shows a scene of self-play training with a mixed set of agents. Rule-based IDM agents are in blue while the RL agents are in red. The green ego agent is also an RL agent. The ego and RL agents share the same policy, but their input representation is normalized to be the center of the world (see the insets pointed to by the white arrows). For each episode, there is a mix of IDM and RL agents with different policy parameters.

We divide our self-play training in 3 stages. In the 1st stage, the RL policy is trained in the *sole* presence of rule-based IDM agents. In stage 2, self-play is trained in the presence of 30% IDM agents, 30% RL agents from stage 1, and the other 40% are controlled by the current learning policy. In stage 3, we additionally train with the agents from stage 2. See Tab. 2 for the percentage[1] of agents in different populations. Alg. 1 further details the learning algorithm.

6. Experimental Results

We perform experiments with the aforementioned learning environment and deep reinforcement learning using a distributed learning system which simultaneously stepped through environments in parallel to collect experiences. An NVIDIA Titan X GPU is used to accelerate the learning of the policy function. Training is performed over roughly $10M$ environment update steps, which corresponds to about 278 hours of real time experience.

[1] These numbers are not exact as vehicles are spawned continuously and would-be vehicles occupying the same regions are not spawned.

	Population Agent Types			
Population	IDM	RL	SP1	SP2
Popul. 1	100%	0%	0%	0%
Popul. 2	50%	50%	0%	0%
Popul. 3	30%	30%	40%	0%
Popul. 4	10%	20%	30%	40%

Table 2: *Training agent population distribution of agent types for different self-play training and testing stages.*

Algorithm 1: Self-play multi-agent RL training

Input: Environment $\mathcal{E}$, S stages, N_{idm} IDM agents, N_{rl} self-play agents.
Initialize: Randomly initialize policy π.
for *stage* $s = 1$ *to* S **do**
  Choose the % mix of agents to create population, (see Tab. 2).
  Reset all envs with the new agent population.
  set I updates for this stage.
  for $i = 0$ *to* I **do**
    Launch K parallel threads to gather experiences $\{s_t, a_t, r_t, s_{t+1}, \ldots, s_\tau\}_k$ from all K envs $\mathcal{E}_k$.
    Batch $\mathcal{B} \leftarrow \{s_t, a_t, r_t, s_{t+1}, \ldots, s_\tau\}_k$ for all K.
    Update policy network parameters by training on data $\mathcal{B}$ (using Eq. 2).
  end
end

6.1. Quantitative Results

We plot performance measure in terms of the percentage of success[2], collision rates, and out-of-bounds rates during training. Fig. 5 shows these statistics with respect to accumulated parameter updates across all three stages. Results are averaged over 4 random trials. Performance dips at the start of every stage as new agents are introduced. By the end of stage 3, performance peaks against a diverse set of sparring agents.

In Tab. 3, we quantify various testing success and collision rates for policies learned at different stages. The evaluation, with standard errors, is over 250 random trials without adding exploration noise. As we can see from Tab. 3, the success rate of the IDM agents is very poor at 63%. This is mainly due to the fact that they are governed by rules and

[2] A successful merge is when an agent arrives at its desired destination location, randomly chosen at the start of the episode.

Stage	Agents	Training: Train w/ Population	Testing Success rate % (Collision rate %): Test w/ Popul. 1	Popul. 2	Popul. 3	Popul. 4
0	IDM	N/A	62.8±3.1 (0.8±0.6)	-	-	-
1	RL	(Popul. 1) IDM	94.8±1.4 (4.0±1.2)	77.2±2.7 (12.4±2.1)	-	-
2	SP1	(Popul. 3) IDM+RL+SP1	96.0±1.2 (3.6±1.2)	91.2±1.8 (4.4±1.3)	95.2±1.4 (3.2±1.1)	-
3	SP2	(Popul. 4) IDM+RL+SP1+SP2	93.2±1.6 (6.0±1.5)	94.8±1.4 (4.8±1.4)	95.2±1.4 (4.4±1.3)	98.2±0.8 (1.4±0.7)

Table 3: *Quantitative performance (successful completion and collisions) of various agents against different population of agents. Reported numbers are in success percentage (collision percentage). SP1/2 denote the self-play trained agents.*

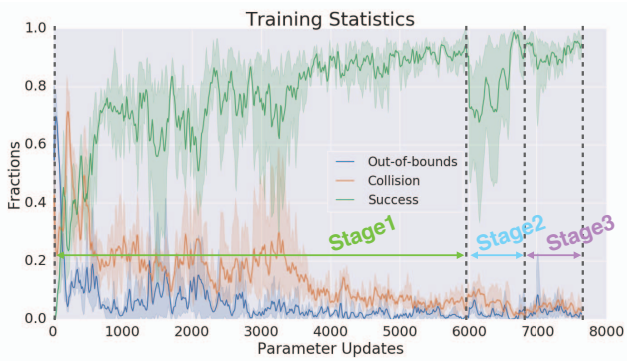


Figure 5: *Different statistics as a function of parameter updates: success rate, collision rate, and out-of-bounds rate. There are 1024 environment steps between weight updates. Arrows denote self-play stages (Sec. 5.5).*

gap acceptance theory [13], which is too simplistic. Next, in row two, we can see that RL trained agents can dramatically improves success rates (95%) against the population of IDM agents. However, performance decreases to (77%) when tested against a mix of IDM and RL agents, demonstrating severe overfitting. Using three stages of self-play, we can improve the success rate up to **98%** against a diverse population of other agents: IDMs, RL, Self-Play1, and Self-Play2 agents[3].

6.2. Qualitative Results

Our policies are able to learn a variety of interesting driving behaviors with self-play. These behaviors include overtaking to merge, emergency braking, using turning signals, and defensive yielding. They showcase various multi-agent interactions and diverse behaviors exhibited by the agents during merges. The resulting policies also learned to be flexible within the confines of the road, deviating from the center of the lane when necessary. Figs. 6, 7, 8, 9 show temporal snapshots of testing episodes. The ego agent is in green, RL and self-play agents are red, and the rule-based IDM agents are blue.

[3] See Tab. 2 for different training and testing population agent types.

(a) t=0.0s (b) t=4.8s (c) t=6.0s

Figure 10: *A failure case (rear-end). Ego (green) successfully brakes, however the vehicle behind it also brakes but not quickly enough, resulting in a rear ended collision.*

(a) t=0.0 s (b) t=1.0 s (c) t=3.4 s

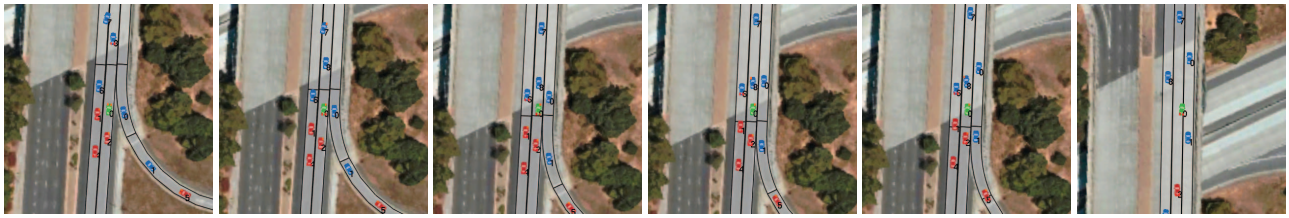
Figure 11: *Emergency stop example. Ego vehicle (green) brakes hard due to the sudden deceleration of the vehicle in front. Instead of only braking, the policy learned to turn its wheels to the side as well, reducing chances of collisions. Note that this is done without violating the right lane.*

6.3. Emergency and Failure Cases

Our algorithm is not perfect, whether it is due to unobservable intentions of other agents, information loss due to temporal discretization, or an inadequately scaled reward function, we still observe collisions in simulation. We analyzed some of these failure cases here. The most common cause of collisions is due to a sudden stop of the vehicle in front of ego. While the RL policy will try to brake, it is sometimes inadequate and results in a collision. In Fig. 10, while ego stops in time, unfortunately, the vehicle behind ego was unable to stop in time and caused a collision.

Another interesting example is an emergency braking situation shown in Fig. 11. Here, the IDM vehicle in front of ego abruptly stops. As ego decelerates, it also steers towards the right side[4]. This is interesting as it loosely correlates

[4] We hypothesize and speculate that it was useful to increase the dis-



(a) t=2.0 s (b) t=2.7 s (c) t=3.5 s (d) t=4.0 s (e) t=4.5 s (f) t=9.0 s

Figure 6: ***Turn Signal + Wait and Merge****: Ego learned to use its turn signal and waited until the adjacent vehicle in the entrance lane accelerated before merging.*

(a) t=0.0 s (b) t=4.8 s (c) t=6.0 s (d) t=6.4 s (e) t=6.8 s (f) t=7.2 s

Figure 7: ***Overtaking and finding a gap****: Ego cannot safely merge due to another vehicle merging to the right from the left-most lane at the same time. Instead, it accelerates to overtake and then merges.*

(a) t=0.0 s (b) t=4.8 s (c) t=6.0 s (d) t=6.4 s (e) t=6.8 s (f) t=7.2 s

Figure 8: ***Yielding****: Ego braked suddenly to yield due to the extremely aggressive behavior of the blue IDM agent.*

(a) t=0.0 s (b) t=4.8 s (c) t=6.0 s (d) t=6.4 s (e) t=6.8 s (f) t=7.2 s

Figure 9: ***Defensive****: Ego cautiously decides to sway back to the right portion of the lane at around the 6.0 secs mark.*

with what is observed in some human behaviors in similar emergency breaking situations.

7. Discussions

In this paper we proposed an algorithm towards tackling the challenging problem of multi-agent robust decision making in a simulated traffic merge environment. Starting with a geometrically realistic environment but populated with only simple rule-based agents, we iteratively increased the diversity of the agent population by using self-play to add more and more capable RL-based agents. We empirically showed that three stages of self-play can dramatically increase the success rate while also keeping failure rates low. In addition, qualitatively, the learned policies exhibit some interesting and human-like behaviors. For future work, it is critical to drive the collision rates to zero. Reward shaping and modeling the future uncertainties are also two possible avenues for the future.

tance traveled the wheel until the front made contact with the vehicle in the front. Interestingly, it does not go across the right ego lane boundary.

Acknowledgements We thank Barry Theobald, Hanlin Goh, Ruslan Salakhutdinov, Jian Zhang, Nitish Srivastava, Alex Druinsky, and the anonymous reviewers for making this a better manuscript.

References

[1] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch. Emergent Complexity via Multi-Agent Competition. *ArXiv e-prints*, Oct. 2017. 6

[2] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer, 2010. 1, 2

[3] J. Chen, B. Yuan, and M. Tomizuka. Model-free deep reinforcement learning for urban autonomous driving. *arXiv preprint arXiv:1904.09503*, 2019. 2

[4] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018. 2

[5] C. Dong, J. M. Dolan, and B. Litkouhi. Intention estimation for ramp merging control in autonomous driving. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1584–1589. IEEE, 2017. 1

[6] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 2

[7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. 3

[8] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016. 2

[9] D. González, J. Pérez, V. Milanés, and F. Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Trans. Intelligent Transportation Systems*, 17(4):1135–1145, 2016. 1

[10] J. Heinrich, M. Lanctot, and D. Silver. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning*, pages 805–813, 2015. 2

[11] J. Heinrich and D. Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016. 2, 6

[12] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed prioritized experience replay. *CoRR*, abs/1803.00933, 2018. 3

[13] S. Y. Hwang and C. H. Park. Modeling of the gap acceptance behavior at a merging section of urban freeway. In *Proceedings of the Eastern Asia Society for Transportation Studies*, volume 5, page e1656. Tokyo: Eastern Asia Society for Transportation (EASTS), 2005. 1, 4, 7

[14] A. Kesting, M. Treiber, and D. Helbing. Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4585–4605, 2010. 4

[15] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, 2015. 4

[16] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017. 2

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. 2, 3

[18] H. Marzbani, M. Simic, M. Fard, and R. N. Jazar. *Better Road Design for Autonomous Vehicles Using Clothoids*, pages 265–278. Springer International Publishing, Cham, 2015. 4

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. 2

[20] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999. 5

[21] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1, 04 2016. 1

[22] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. 2

[23] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. End-to-end deep reinforcement learning for lane keeping assist. *arXiv preprint arXiv:1612.04340*, 2016. 2

[24] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015. 3

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3

[26] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016. 1, 2, 4

[27] S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017. 2

[28] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv e-prints*, Dec. 2017. 6

[29] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017. 2

[30] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. 3

[31] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *In Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000. 3

[32] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, Y. Wu, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://bit.ly/2TeLiqh, 2019. 2

[33] J. Wei, J. M. Dolan, and B. Litkouhi. Autonomous vehicle social behavior for highway entrance ramp management. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 201–207. IEEE, 2013. 1

[34] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. 3

[35] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, C. Guionneau, and R. Coulom. TORCS, the open racing car simulator, v1.3.5. http://www.torcs.org, 2013. 2