Skills to Drive: Successor Features for Autonomous Highway Pilot

Laszlo Szoke, Szilard Aradi, *Member, IEEE*, Tamas Becsi, *Member, IEEE*, Peter Gaspar, *Member, IEEE*

***Abstract*—Reinforcement learning applications are spreading among different domains, including autonomous vehicle control. The diverse situations that can happen during, for instance, at a highway commute are infinite, and with labeled data, the perfect coverage of all use-cases sounds ambitious. However, with the complex tasks and complicated scenarios faced during an autonomous vehicle system design, the credit assignment problem arises. How to construct appropriate objectives for the artificial intelligence to learn and the preferences between the different goals also matter of the designer's choice. This work attempts to tackle the problem by utilizing successor features and providing a possible decomposition of the reward functions, guiding the agent's actions. This method makes the training easier for the agent and enables immediate, profound performance on new combined tasks. Furthermore, with the optimal composition, the desired behavior can be fine-tuned, and as an auxiliary gain, the decomposition empowers different driving styles and makes driving preferences rapidly changeable.**

We introduce the adaptation of FastRL algorithm to autonomous vehicle domain, meanwhile developing a stabilizing way of using Successor Features, namely DoubleFastRL. We compare our solution for a highway driving scenario with basic agents such as Q-learning having multi-objective training.

***Index Terms*—Autonomous Vehicles, Intelligent agents, Machine learning, Highway Assist, Reinforcement Learning, Successor Features**

I. Introduction

RECENTLY, many research areas pay increasing attention to artificial intelligence (AI). However, today, "powered by AI" mostly means using machine learning, deep learning, or artificial neural networks in the products or during their production. From computer vision, through speech recognition, to camera applications, we can find AI-based code whose goal is to make our lives easier and better. In this aspect, another rapidly developing and industrially backed research area is autonomous vehicles. Among others, the main driving force to apply AI in commercial vehicles lies within safer driving and comfortable travels. Many companies invest in developing autonomous vehicles, and an immense amount of research focuses on designing appropriate algorithms for diverse and challenging tasks. Supervised learning is popularly applied in different domains. Still, for autonomous vehicles (AV) other fields of machine learning, like Reinforcement Learning (RL) is also being considered because it supports online learning and interaction with the surroundings.

Furthermore, RL applications develop rapidly due to their recent success in different fields of life. Google DeepMind's AlphaGo Zero showed impressive performance in the game of Go several years ago. Since then, other successful implementations and agents have been published in even more complex domains, see [1]–[5]. Despite the success of RL methods, there are some disadvantages, must be solved problems that wait for solutions, one being the credit assignment. Finding the appropriate rewarding in case of a desired self-driving behavior is even more challenging. Since, in the conventional sense, the RL agents learn from scalar feedback, also known as a reward, the function encompassing the different aspects of behavior could become tangled. Hence, designing the appropriate reward function for a suitable credit assignment is key to RL-based systems' successful application. [6] further elaborates on the topic, pointing out design errors in the latest publications and formulating the problem with reward-shaping, reward tuning, and unintentionally included loopholes. Additionally, the authors also suggest sanity tests, a guide, how to find reward misdesign. Others, e.g., [7] show alternatives or possible solutions in their recently published work that can solve the more complex rewarding systems (such AVs have) by decomposition of the reward functions. Albeit the work focuses on a scavenger setting, using the so-called Successor Features, they provide tools for the agents to decompose and better understand their tasks.

Our choice of using highway scenarios of all possible situations is based on that it may seem simple at first glance, but there are many soft rules, e.g., keeping right or changing lanes, which creates the possibility of different driving styles based on the skills or the attitude of the driver. During highway control, there can be several styles induced by using SFs, which are not present in the case of other traffic scenarios. In our method, we wish to emphasize the advantages of using SFs, for which this highway scenario is essential. Also, a multilane highway provides several conflict situations because of the need for maneuvering. The reason is that rapidly changing weights result in different driving styles and behaviour, which are not only visible, but show meaningful information.

The research was supported by the Ministry of Innovation and Technology NRDI Office within the framework of the Autonomous Systems National Laboratory Program.

The research reported in this paper is part of project no. BME-NVA-02, implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021 funding scheme.

This paper was also supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences

Laszlo Szoke, Szilard Aradi and Tamas Becsi are with the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, Budapest, Hungary (e-mail: szoke.laszlo;aradi.szilard;becsi.tamas;@kjk.bme.hu).

Laszlo Szoke is also a researcher at Robert Bosch Kft., Budapest Hungary, laszlo.szoke@hu.bosch.com

Peter Gaspar is with the Systems and Control Laboratory, Institute for Computer Science and Control, Budapest, Hungary (e-mail: gaspar.peter@sztaki.hu).

In this work, after a mathematical background review, we look at the existing literature on simulation-based RL in the autonomous vehicle control domain, and set up a suitable environment for our perspectives. Moreover, we look at the literature available on successor features and the recent achievements of its usage. We concisely clarify its methodology and show how it works. As our contribution, we propose a way to define tasks for an agent in highway scenarios using the successor features by adapting them to the autonomous vehicle control case. We adopt the original FastRL learning algorithm [7] and with doing so we propose an agent that understands its tasks on a highway, meanwhile it is enabled to decompose the rewards and create meaningful features or "skills" that is beneficial in unknown situations. Basically, a different implementation and application of the method is given.

Furthermore, we stabilize the learning process of the adopted algorithm for our case, and thus introduce Double-FastRL (DFRL). As a consequence: upon changes in our preferences just by adjusting the policy, we can alter the behavior of a trained agent and induce different driving styles without any need of retrain. By decomposition during training and composition in inference time, the skill set of the RL agent becomes expandable, resulting in much less training and better performance on new tasks.

The results imply that the different combinations of learned skills can induce different driving styles and various situational understanding and actions. It is an efficient way of addressing compact problems such as autonomous driving and effectively tackling new function implementation. Our original intention was not to outperform the standard algorithms, like Q-networks, though the results show that the evaluation performance of the proposed algorithms surpass the baseline, with the additional benefit of applying the trained model on new tasks with instant performance.

II. Related work

a) Reinforcement Learning in vehicle control: The recent improvements of computer science and hardware resources made it possible for artificial intelligence and machine learning to have their renaissance. Fields like reinforcement learning also got their fair share of the renewals by the enabling of deep learning. In 1998, Sutton and Barto revisited the concept of RL [8]. Since then, there are thousands of articles and papers using the knowledge they laid down as building blocks. The research tries to make RL handier and more competent to solve the real problems in life. Thus, every now-and-then new methods, algorithms present themselves to making reinforcement learning more capable and powerful.

As [9] investigated, autonomous vehicles (AV) pose a challenging domain not only for the AI researchers but transport, vehicle, and network security engineers also. The interaction with the surroundings of a self-driving car must be seamless. Thus communication and its role in the traffic must be well defined. During operation, it can not malfunction, make bad choices because lives are at stake. Supervised Learning (SL) applications provide significant results in the topic. However, some think with proper rewarding, reinforcement learning could constitute a solution to artificial general intelligence [10]. Simultaneously, this allows us to consider it as a possible solution to the field of vehicle control. A further advantage of RL compared to SL is that the online interaction allows us to explore any exciting situation and thus extend our expectations and beliefs about how the world works. One must add that SL-based methods are often utilized for various tasks in RL settings, such as sensor data processing, classification, and object recognition. [11], [12]

Today, there are a vast amount of examples of RL in the vehicle domain. [13] provides an overview on the hierarchical motion planning problems and the basics of Deep Reinforcement Learning. The key elements of designing RL systems are drawn, and the paper also provides insight into vehicle models, simulation possibilities, and computational requirements of such tasks. Furthermore, the paper surveys the state-of-the-art solutions systematized by the different tasks and levels of autonomous driving.

Other works on the topic show different algorithms and approaches to apply in the field. E.g., complex urban scenarios with Double Deep Q-Network (DDQN), Soft Actor-Critic (SAC) and Twin Delayed Deep Deterministic Policy Gradient (TD3) [14], changing highway scenarios with modified Deep-Q-Networks (DQN) [12], deep deterministic policy gradient (DDPG) for lane change on highway [15]. However, no matter the applied algorithm, the environment, and the target task, none of the mentioned papers guarantee 100% performance. Our previous works also show examples of RL agents built with simple neural networks performing in highway scenarios [16], [17], but the perfect behavior can not be assured in every situation. Further examples of autonomous driving functions solved by RL, such as car-following, lane-keeping, trajectory following, merging, or driving in dense traffic, can be found in the following collection: [18]–[23].

There are examples of how an agent can be trained in an environment with a defined reward and goal, but further investigation and improvements are needed to achieve flawless behavior. Mostly, the definition of the environment contains anomalies, or the reward formulation is not perfect. It is a common mistake to use reward shaping, a detailed description of the problem found in [6], which can tweak the training but alters actual performance. Moreover, as the rewards change or the environment alters, the trained agents' performance tends to worsen due to the specific training circumstances.

The problem seems to initiate from that for every task, the RL agents - in general - learn their behavior from scratch, which highly differs from the way humans do.

Leveraging prior knowledge can offer a solution. Thus, in theory, the range of problems our agents can tackle can be significantly extended if they are endowed with the appropriate mechanisms, so-called "skills". The "skills" can be defined as solutions to distinct sub-tasks or elementary moves, which contribute to the solution of a bigger, more complex problem. For instance, to walk, one must first learn how to bend the knee or balance one foot. Such skills are building on top of each other, and combining them integrates to more intricate dilemmas. In recent literature, there are plenty of articles

promoting this idea of skills, called successor features (SF) [7], [24]–[27].

The key idea behind SFs is to decompose the state changes into features, which coincide with the skills in solving a problem. If we accept the features as different skills, we are one step from defining tasks with different reward functions and the combination of skills. This results in outstanding inference performance on unseen tasks, with the combination of previously learned skills [25]. The following subsection presents some of the significant papers on the topic.

b) Successor Features: The basic concept of successor features was set by Dayan [27]. He uses the fact that the focus of temporal difference algorithms is on the good estimation of returns over time. In conclusion, he states that the similarity of successors determines how appropriate the generalization between states is. By defining the term of successor representation, he bedded ground for future works. An example, [28] deals with end-to-end training from raw sensory data and presents deep successor reinforcement learning. The authors decompose value functions into two components, one being the reward prediction and a map of successors. Then they use the inner product of them to compute the value functions of the states. Another example is an application for navigation of robots in single maze-like environments, where the authors use the successor features [29]. [30] also leverages the power of Dayan's successor representation by demonstrating the efficiency of their approach on a collection of grid-worlds, and on Fetch, a high-dimensional robotic control environment. [31] presented at ICML[1] argues that optimal value function composition can be achieved in entropy-regularised reinforcement learning (RL). They show the composition of value functions in a high-dimensional video game and extend their result to the standard RL settings.

After Dayan, the first related article on Successor Features is [24]. This paper recycles Dayan's idea and extends it with General Policy Improvement (GPI), a generalization of policy improvement operation of dynamic programming that considers a set of policies rather than a single one. The seamless integration into RL is derived, and the authors provide performance guarantees for the transferred policy without any additional learning. Moreover, the authors do not stop at the application of GPI, but in a later published paper, they extend the idea with General Policy Evaluation (GPE). The next subsection collects some related work on the matter.

c) GPE & GPI: General Policy Evaluation (GPE) and General Policy Improvement (GPI) or as usually referred to as General Policy Update, are extensions of policy evaluation and improvement, respectively. However, instead of point-based operations, we extend the terms to set-based ones. [25] takes a further step in the application of successor features and introduces an algorithm to be used with deep reinforcement learning and successor features. In this way, the solution of previously solved tasks can be reused. Beyond theoretical proof of applicability, the authors reveal an agent trained on different tasks and then can combine its "skills" to act on new ones. Related to this development, [26] describes how to combine skills in reinforcement learning best. The possibilities of exploiting SFs seem plenty. Others, for instance, [32] combine the advantages of the scalability of Universal Value Function Approximators, the instant inference of Successor Features, and the strong generalization of GPI. [33] shows another powerful application of SFs with GPI and GPE. The authors use Variational Intrinsic Successor Features to create VISR that can outperform state-of-the-art RL models. First, by unsupervised pre-training, they define the successor features purely from state transformations, then they add rewards with reinforcement learning. It is definitely a step towards general reinforcement learning.

A good example for the full utilization of SF + GPE + GPI is described in [7]. FastRL shows how to apply the divide-and-conquer approach to RL, and the theory is supported by a vast amount of experiments.

In this work, we concentrate on the usability of SF+GPE+GPI in the domain of AVs and RL. For further articles on the topic considering transfer learning, hierarchical RL, GPI, GPE, and related applications of RL, see [34]–[36].

[1]International Conference on Machine Learning

III. Methodology

RL itself is based on online action-reaction scenarios. There is an actor, also called an agent, who gets information about the surroundings (usually referred to as the environment). In the case of an AV, this information might be a camera image, LIDAR, or other sensory input. Still, in general, any useful information about the environment that helps the agent in its decisions is acceptable. This meaningful information, which is the input of the agent, is referred to as a state. After observing the state of the environment, each RL agent tries to output an action that will maximize its rewards. Rewards, which are normally scalar signals, results of the state transitions caused by the agent's actions, a type of goodness indicator of the last decision. This enables the domain to solve tasks that can not be exactly defined, e.g., optimal highway driving. The result of reward-based training is similar to what humans experience when they learn based on the interaction and its consequences. Deciding the goodness of a complete process is much easier than evaluating each step and assessing exactly, which was a good or bad move. Contrary to Supervised Learning, RL does not require labelled training data, or exact definition of a proper action, instead it figures out the relations between interactions by trial and error and thus generates only the data it needs. However, one must note here the problem of credit assignment, which can hinder the successful trainings, but this will be detailed later.

A. Markov Decision Processes

The underlying mathematical formulation of RL problems is done through Markov Decision Processes, a mathematical framework for decision making in a situation where the system's output (environment) is partially randomized but depends on the actual current action. Suppose an MDP system has the Markov property. In that case, it means that the conditional probability distribution of the following state of a random process depends only on the current state, given all the

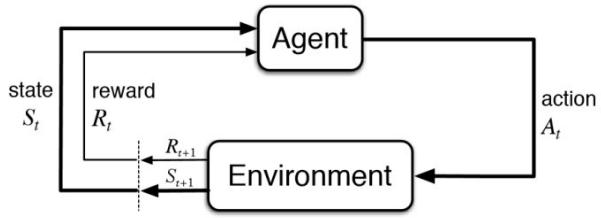


Fig. 1: RL loop of a time step

previous states [8]. This implies that the state should contain all information needed to draw this probability and also helps us to make decisions based on the current observable state.

An MDP can be defined with a 4 element tuple, $M \equiv< \mathscr{S}, \mathscr{A}, P_{\mathbf{S},\mathbf{A}}, \mathscr{R} >$, where at every given discrete time step of an episode, RL agents interact with the environment. As a result, they observe the changes at every $t \in \{1, 2, 3, 4, ...\}$ as state $s \in \mathscr{S}$, then choose action $a \in \mathscr{A}$ to maximize the expected reward $r \in \mathscr{R}$. During this process the environment transforms from state s to s'. This process is depicted in Figure 1. This is one step, and in the long run, the agent tries to find the maximum of the reward function.

[24] thinks of the reward functions as different tasks and thus, the goal of RL is to find a policy $\pi : \mathscr{S} \mapsto \mathscr{A}$ that maximizes the value of every state-action pair:

$$Q_r^\pi(s,a) \equiv \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r\left(S_{t+i}, A_{t+i}, S_{t+i+1}\right) | S_t = s, A_t = a\right], \tag{1}$$

where $\mathbb{E}^\pi[\cdot]$ denotes expectation over the trajectories induced by π, and $\gamma \in [0, 1)$ is the discount factor, used for discounting future rewards. Using (1) we can proceed to the formulation of GPE and GPI with SFs.

B. SFs, GPE and GPI

The utilization of these concepts results in "the capability of an agent to learn about complex reward functions at the same time has the potential benefits of the decomposition of complex tasks into simpler ones, the exchange of information between tasks, and the reuse of skills" [7]. Without the fully detailed deduction of the equations, based on [24] the following terms are used in our work:

First, we choose $\phi : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \mapsto \mathbb{R}^d$ as our arbitrarily selected feature function vector, where each element is a "feature". Then according to [24] any arbitrary preference vector $\mathbf{w} \in \mathbb{R}^d$, we can define the reward function as:

$$r_{\mathbf{w}}(s,a,s') = \phi(s,a,s')^\top \mathbf{w}. \tag{2}$$

Examples for such features are shown in Section IV-A2 , which induce the traditional scalar reward function when multiplied with $\mathbf{w}$. In general RL, this multiplication is included inside the environment in case of a multi-objective reward function, where the different components of the reward are "hard-coded", and thus cannot be changed rapidly. However, using (2) , we can alter the reward function values instantly by changing $\mathbf{w}$.

Following [7] , the successor features are defined as the expected value of our discounted feature functions along a trajectory when acting on policy π. This yields to equation (3).

$$\psi^\pi(s,a) \equiv \mathbb{E}^\pi\left[\sum_{i=0}^{\infty} \gamma^i \phi\left(S_{t+i}, A_{t+i}, S_{t+i+1}\right) | S_t = s, A_t = a\right], \tag{3}$$

If we multiply (3) with $\mathbf{w}$ on both sides, we can deduce the following:

$$\begin{aligned} \psi^\pi(s,a)^\top \mathbf{w} &= \mathbb{E}^\pi\left[\sum_{i=0}^{\infty} \gamma^i \phi\left(S_{t+i}, A_{t+i}, S_{t+i+1}\right)^\top \mathbf{w} | S_t = s, A_t = a\right] \\ &= \mathbb{E}^\pi\left[\sum_{i=0}^{\infty} \gamma^i \mathbf{r}_{\mathbf{w}}\left(S_{t+i}, A_{t+i}, S_{t+i+1}\right) | S_t = s, A_t = a\right] \\ &= Q_{\mathbf{r}_{\mathbf{w}}}^\pi(s,a) \equiv Q_{\mathbf{w}}^\pi(s,a) \end{aligned} \tag{4}$$

Consequently, $Q_{\mathbf{w}}^\pi(s,a)$ is the same Q-function as in (1), but now, it is parametrized by $\mathbf{w}$, and most importantly is simplified to the inner product of $\psi^\pi(s,a)^\top \mathbf{w}$. If we have n policies, we generate an $n \times d$ dimensional ψ and the dimensionality of $\mathbf{w}$ is $d \times k, \quad k \in \mathbb{R}$. This results in a very efficient form of GPE, where for all policies we can calculate the value function. This property we will use to our advantage. For more details on the deduction and proof of (4) see [7].

The last missing piece of the method is GPI. In this context, after Barreto et al. the improved policy yields to:

$$\pi'(s) \in \underset{a \in \mathscr{A}}{\operatorname{argmax}}(\max_{\pi \in \Pi} Q_r^\pi(s,a)) \tag{5}$$

Basically, we evaluate the value function of the state-action pair over all policies (GPE) and based on (5)[2] we choose the actions with the highest values.

After discussing the main prerequisites to the investigated method, we quickly review the environment and provide the details of the applied neural networks and training settings in the next section.

IV. Used Agents

A. Environment

There are a lot of available simulators for vehicle control purposes, depending on the dynamics models required, the traffic scenarios, and the purpose. Popular ones include Carla [37], PreSCAN [38] and CarSim [39]. However, they are mostly suitable for problems in need of high accuracy vehicle dynamics model and simulations with few vehicles. Our choice to simulate the highway scenarios is Simulation of Urban MObility (SUMO), a free open-source program designed for micro and macroscopic traffic simulation [40]. Refer to documents [41], [42] for an in-depth introduction to SUMO, in which the authors detail and explore the available operations in the software. TraCI is a submodule of SUMO, which establishes TCP/IP connection and thus can control the software in real-time. This supports C++. Python and MATLAB coding, and provides an interface to enhanced control of the software. To ease the usage of our environment, we use the OpenAI Gym interface and structure.

[2] In [24] theorem 1 shows (5) is a legitimate form of GPI.

Vehicle type	δ	$p_v[\mu, \sigma]$	$p(exists)$	$p(kr)$	$p(coop)$
car1	0.5	(1,0.5)	0.2	0.5	0.2
car2	0.4	(1, 0.3)	0.1	0	0.3
car3	0	(1.5, 0,45)	0.3	0	0
car4	1	(1.2, 0.2)	0.3	1	1
car5	0.8	(1, 0.2)	0.1	0.7	0.5
EGO	0	(1, 0)	1	0	0

TABLE I: Surrounding vehicle parameters

Fig. 2: The structured state space

1) Setup: Our environment setting is based on a highway scenario, which is justified by allowing recognizable changes during driving, thus helps visualizing the usage of SFs. We use episodic RL, where the termination events are collision, too slow commuting $v < 60$ $[km/h]$, or leaving the highway. A 1000 $[m]$ straight highway was designed with three lanes. Other traffic participants are generated randomly with randomized behavior and speed profile. Table I shows how the vehicles are generated. $p_v[\mu, \sigma]$ gives the speed attribute distribution of the vehicle with μ mean and σ variance relative to the allowed lane speed. $p(exists)$ is the probability of generating each car in the flow, $p(kr)$ indicates the probability of keeping right for that vehicle type, $p(coop)$ shows the percentage of cooperation: e.g willingness of yielding, lane-change or respect for others. The maximum speed is 50 $[m/s]$, and the desired speed randomly regenerates after 50 steps between $28 - 43$ $[m/s]$. Meanwhile, the average speed of the surrounding vehicles is around 25 $[m/s]$. This ensures that if the EGO wants to keep the desired speed, it has to overtake and change lanes. The simulation time and interaction frequency influences the performance of the system. With more frequent decisions the system gets slower due to the high number of inference of the agents, meanwhile with a sparser intervention period we could face possible danger due to latency in situation recognition. Considering general practice and the reasons mentioned, we selected 100 [ms] as our interaction and simulation step size, where the agent can select both lateral and longitudinal control. Thus, the action space stands from 3 lateral (left, keep, right) and 3 longitudinal control (slow down, keep speed, speed up), and the combination results in 9 actions. The lateral control hence entails only lane-change commands to both sides, which actualize instantly. It is considered a higher-level command, and our action space is similar to [43], [44]. The state-space includes information about the front and rear vehicles in both neighboring lanes, in a structured manner as Figure 2 depicts. The relative distance and speed are given, except for the side vehicles of the EGO. There only the presence is signaled with the values 1 or 0. To make the MDP fully defined, additional details include the EGO speed, heading, desired speed, and lane id. Thus, the state space can be written as an 18 element vector. All state vector values are normalized and scaled between $[-1, 1]$ element-wise. For other works using similar observation representation, see [29], [45].

2) Rewards: For SFs, the environment supports vectored rewards instead of scalar feedback. We described 6 different reward functions.

- r_1 signals the terminating events and results in -1 if any occurs, 0 otherwise.
- r_2 considers the divergence from the desired speed, with a maximum value of 0 and the minimum of -1,
- r_3 is an immediate reward for each successful lane-change 1 or 0.
- r_4 is designed to urge keeping right, and is 1 if the EGO is at the available most right lane, 0 otherwise.
- r_5 intends to enforce the following distance of the front vehicle. Its value is $max(-\frac{dv}{v_{ego}}, -1)$, where dv and v_{ego} is always positive, where dv is the speed difference between the EGO and the front vehicle and v_{ego} denotes the speed of the EGO.
- r_6 mirrors r_5 but for the vehicle behind the ego. It is designed to reduce cut-in when changing lanes.

As one can see, our environment contains terminal states due to the formulation of the problem. Thus, we have one SF for terminal events and others for immediate rewards.

B. Experiments

Parameters	
Transitions	20 000 000
Learning policies	6
Model hidden size / feature	128, 64
Replay memory size	20 000
Initial learning rate α	0.001 (AdamW)
Discount factor γ	0.9
Exploration factor ϵ	0.5 - 0.1
EGO reference speed change interval	50 steps

TABLE II: Training parameters

Our experiments with the proposed methods and explained environment contain several different training settings. Table II lists the parameters used by the most optimal training. As agents, first we adapt the FastRL algorithm proposed by [7] to the highway driving problem, and extend it with replay memory. FastRL is a modified Q-learning algorithm prepared for the usage of SFs. Our implementation is somewhat different from that of [7] , and can be leveraged from Algorithm 1 and 2 by choosing $\theta^{target} \equiv \theta^{acting}$. Furthermore, as a stabilizing factor, we propose an updated FastRL algorithm, DoubleFastRL (DFRL). The pseudocode for the proposed algorithms is given in Algorithm 1 and 2. Our DFRL draws ideas from DQN [46] and DDQN [47], and applies a second neural network (θ^{target}), that serves as a target successor features predictor. Similar to what DDQN added to DQN, our modified FastRL training method is more stable, and during training it has lower variance of loss of the predicted SF values. Both agents were developed in Pytorch implementing the SFs for RL trainings.

First, the agent acts based on a pre-selected policy for each episode. We observe the next states, rewards and save each interaction to the replay memory. Then, in the update function, the weights of the acting neural network θ^{acting} are updated by the loss back-propagation, which is the following. First,

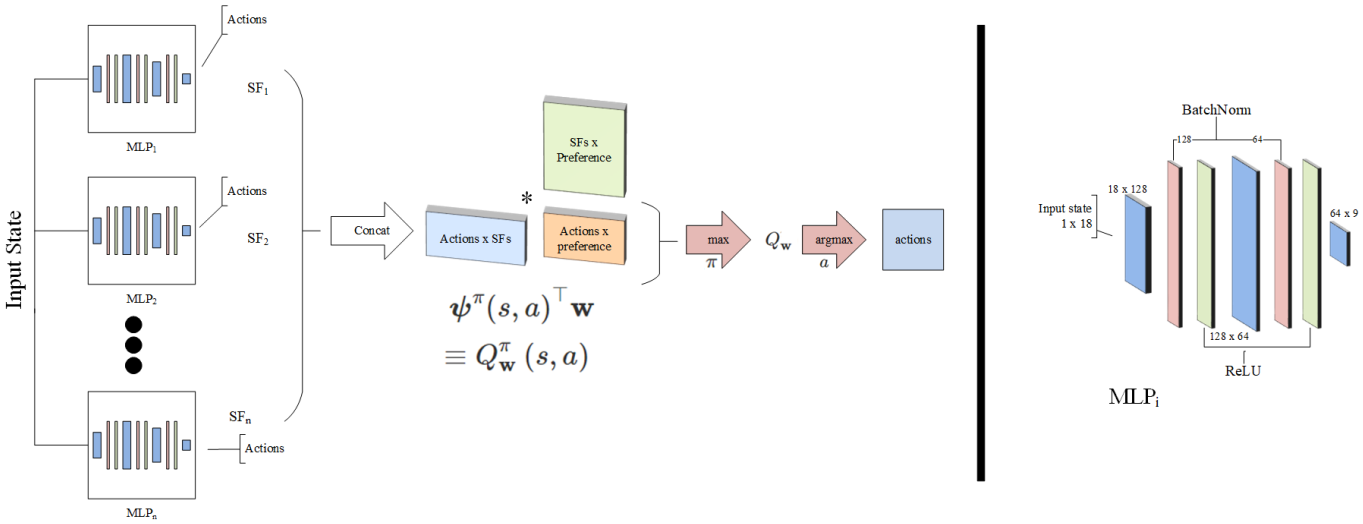


Fig. 3: The structure of the model

Algorithm 1 Proposed DoubleFastRL (DFRL) algorithm

procedure TRAIN
Initialize weights:
$\theta_i^{acting}, \theta_i^{target}, \quad i \in [0, d]$
Set $\mathbf{w} \in \mathbb{R}^{j \times d}$, j is arbitrary, $\mathbf{w}_j$ induces π_j
for each episode **do**
$s_t \leftarrow$ reset environment state ▷ $t = 0$
$j \leftarrow$ Uniform(w)
while not done **do**
if Bernoulli(ϵ) = 1 **then**
$a_t \leftarrow Uniform(\mathscr{A})$
else
$\psi^{\pi_\mathbf{j}} \leftarrow \theta_j^{acting}(s_t, b)$ ▷ $\forall b \in \mathscr{A}$
$a_t \leftarrow \underset{b \in \mathscr{A}}{\text{argmax}} \, (\psi^{\pi_\mathbf{j}\top} \mathbf{w_j})$ ▷ $\forall b \in \mathscr{A}$
$s_{t+1}, \phi_t, done \leftarrow Env(a_t)$ ▷ step with a_t
$memory \leftarrow Push(s_t, a_t, s_{t+1}, \phi_t, done, j)$
$s_t \leftarrow s_{t+1}$
UPDATE_NETWORKS()

Algorithm 2 Network updates with Replay memory

function UPDATE_NETWORKS()
if $memory > batch$ **then**
$(s_t, a_t, s_{t+1}, \phi_t, j_t) \leftarrow Sample(memory)$
$\psi_{pred}^{\pi_j} \leftarrow \theta_j^{acting}(s_t, a_t)$
$\psi_{temp}^{\pi_j} \leftarrow \theta_j^{acting}(s_{t+1}, b)$ ▷ $\forall b \in \mathscr{A}$
$a_{t+1} \leftarrow \underset{b \in \mathscr{A}}{\text{argmax}}(\psi_{temp}^{\pi_j\top} \mathbf{w_j})$ ▷ $\forall b \in \mathscr{A}$
$\psi_{target}^{\pi_j} \leftarrow \theta^{target}(s_{t+1}, a_{t+1})$
$\delta \leftarrow \phi_t + \gamma \psi_{target}^{\pi_j} - \psi_{pred}^{\pi_j}$
$\theta^{acting} \leftarrow \theta^{acting} + \alpha \delta \nabla_{\theta^{acting}} \psi_{pred}^{\pi_j}$
if update target **then**
$\theta^{target} \leftarrow \theta^{target} \cdot (1 - \tau) + \theta^{acting} \cdot \tau$
return δ

we predict the SF values $\psi_{pred}^{\pi_j}$ for the current states (sampled from the memory). Just to clarify the dimensionality of $\psi_{pred}^{\pi_j}$ is $[batch \times policies \times actions \times SFs]$. Then, we get optimal next actions a_{t+1} based on the next states. Inferencing our target network θ^{target} with the next action a_{t+1} and next state s_{t+1}, we get the target SFs $\psi_{target}^{\pi_j}$. Our proposed method (DFRL) naturally falls back to FastRL, if we inference $\psi_{target}^{\pi_j}$ with θ^{acting} instead of θ^{target}.

All this is based on a batch operation sampled from the $memory$. In the algorithms denote ϕ_t the feature function, which provides the element-wise rewards for the different features when multiplied with $\mathbf{w}$. Further parameters are the learning rate α, and the discount factor γ.

We designed $d = 6$ features correlating with the rewards mentioned above. Thus $\mathbf{w} \in \mathbb{R}^{n \times d}$ induces a set of policies $\{\pi_1, \pi_2, \ldots \pi_n\} \in \Pi$. We chose $n = d$, and linearly independent w_js, thus $\mathbf{w}$ is a diagonal matrix.

Based on the rewards, the policies give the following behavior:

- π_1 concentrates on avoiding the terminating events,
- π_2 acts to keep the reference speed,
- π_3 changes lanes when possible,
- π_4 wants to be in the most right lane,
- π_5 keeps a safe following distance when possible,
- π_6 prevents cut-ins to rear vehicles

Although π_3 is not optimal behavior in the case of highway driving, it can be used to get a drastically different behavior and thus show how the SFs work with the different preferences $\mathbf{w_j}$. Figure 3 shows the inference process of the agent. During training, we select the acting policy randomly as mentioned above rather than taking the maximum over policies, but in inference time, the maximum Q_w^π is chosen. The network structure is also depicted on the right side of Figure 3, where one can see that each SF has its own MLP network with 2 linear layers. The numbers represent the input and output dimensions of the layers. The hidden space has 128

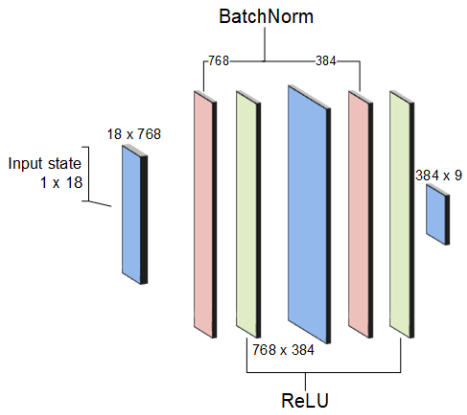


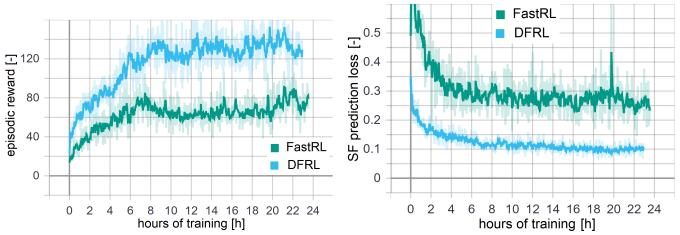
Fig. 4: Q-learning agent represented with MLP layers

and 64 neurons and is connected with $BatchNorm$ and $ReLU$ activation layers. We use a replay memory with a batch size of 4096. One should also note that the immediate rewards are independent of each other. For instance, π_3 got no punishment, even if the lane-change resulted in the departure of the highway (a terminating event), or π_2 got its step reward even if it collided. With this, all the policies are concentrating on their tasks, and try to maximize the rewards

To be able to compare both DFRL and FastRL algorithms in the case of highway driving, we trained a simple Q-learning agent as well, which task was to maximize a composed complex reward function r_{w_Q} expressed with ϕ:

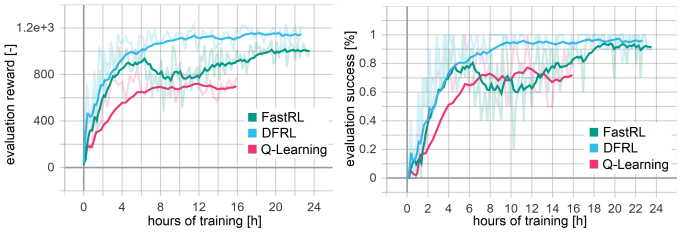
$$r_{w_Q} = \phi^\top \mathbf{w_Q}, \tag{6}$$

where $\mathbf{w_Q} = [1, 1, -0.5, 0.5, 0.5, 0.5]^\top$. Hence, a behavior is expected, where the agent keeps the desired speed if possible, it avoids the terminating events while trying to keep right, avoid unnecessary lane change and have decent distances both from front and back vehicles. Note that the agent is in a more difficult situation because it has to decompose and learn the different feature rewards by itself. Also, to make an even chance for the Q-agent, the number of parameters are $6 * 128$, and $6 * 64$, and the resulting dimensions of the network are shown in Figure 4. After the Q values are provided for the actions, we take the action with the maximum value as usual. As before, the numbers represent the input and output dimensions of each layer. All other objectives of the trainings are the same, to provide a fair comparison. Theoretically, the inference time of the SF-based agents scales linearly with the applied number of SFs compared to the Q-Learning baseline. In our work, we compared three agents with approximately the same number of layers and parameters. To make it even for Q-Learning, the SF models have one-sixth of the parameters per feature, resulting in a forward pass, which has approximately the same time demand in all three cases due to the technical and implementational considerations. The only additional calculation time is the multiplication by the preference vector. However, since our simulation runs with 0.1 [s] simulation steps, and the mentioned forward pass and dot product are faster, the additional latency is neglectable, and our agents run in real-time. Due to this, we do not conduct further performance optimization or comparison.



(a) training reward (b) training loss

Fig. 5: Training results of the SF-based agents



(a) evaluation reward (b) evaluation success

Fig. 6: Quantitative comparison of evaluation performances

V. Results

This section evaluates both the FastRL and DFRL agent against a Q-learning baseline and shows, how the different features have learned to behave according to our intentions.

In Figure 5 we plot the measured parameters throughout the training of the SF-based agents. The x-axis is in training hours, because each agent experienced 20 million transitions, as stated in II, and based on the episodic performance of the agents some may finish with less episodes. Figure 5 gives insight of the training processes. That is no need to investigate the Q-agent in this manner, because it has a different objective, it learns on different tasks and with a different loss function. It is clear that DFRL outperforms the adopted FastRL algorithm both in terms of reward and success rate. DFRL converges faster to the episodic reward maximum it can reach during the constantly changing tasks. Also interesting to see is that the DFRL agent learns to succeed during training approximately 30% of the time, however, both SF-based agents have the preference w_1 selected only one-sixth of the episodes. This expectation is exactly what performance the FastRL agent presents. This can be explained with the implicit effect of the following distance feature, which promotes safer distance, thus causing less collisions. It seems that DFRL can deplete this intrinsic information better. It also has lower estimation error for the successor feature values. 5b disguises how the loss of the SF-based agents changed over time, which is calculated based on $\delta = \phi_t + \gamma\psi^{\pi_j}_{target} - \psi^{\pi_j}_{pred}$ as seen in Algorithm 2. The plot indicates well that our proposed DFRL algorithm has a smaller loss value and deviation compared to FastRL, which helps the training as supported by Figure 5a; promotes better reward exploitation and more successful episodes. It can approximate the SF values more precisely, entailing a more meaningful SF representation, due to the additional stability provided by the double network structure. On the other hand, Figure 6a and 6b reflect on the evaluation performance of the agents during

the training processes, because evaluating on the task of the Q-agent, which learns to master this task during training as well, can emphasise why our method is advantageous. After every 100 episodes of training, we choose a preference vector $w_{eval} \equiv w_Q = [1.0, 1.0, -0.5, 0.5, 0.5, 0.5]^\top$. This gives us a fair comparison on the FastRL and DFRL agents' performance with regard to the baseline. Figure 6a and 6b depict that DFRL has a slight superior performance over the others, meanwhile both SF-based agents exploit the rewards better than the Q-agent. Our Q-agent sticks to a safe policy and thus settles with a smaller but secure reward. Note: these agents have been trained on the same machine and thus we can see that for the same amount of experience Q-learning needed 16 hours, and the SF-based agents required approximately 22 hours training. This differences come mostly from the episodic reload of the simulations, because Q-learning had the 20 million transitions in approximately $74k$ episodes, meanwhile, SF-based agents had $125k$ episodes to achieve the same. The plot length embodies that the SF-based methods, and most importantly our suggested one; do not have a huge overhead on the training compared to the baseline. Moreover, they can perform just effectively if not better on new tasks as well.

The plots of Figure 5 do not show the possible behavior changes of the SF-based agents with different preferences. Therefore, to fully emphasize the advantages of SF-based learning (and the usage of our proposed algorithm) in the case of the highway control domain, we have logged several essential data during the evaluation simulations and created box-plot comparisons of their diversity caused by the preferences. Figure 7 visualizes 100 runs of DFRL with one-hot vector preferences where the features are tested one-by-one. *Safe* means $w = [1, 0, 0, 0, 0, 0]$ resulting in a $Q_w(s, a) = \phi_1$. The other rows are interpreted accordingly based on the feature preference vectors described in IV-B. During the evaluation process, only the preferences change, the trained model is the same. The boxplots show the distribution of the episodic values throughout the 100 runs, and compare how the parameters vary. The yellow line represents the median of the values. The lane changes and "keeping right" subplots show the distribution of the percentage values over the steps. Meanwhile, the others present the summed episodic average values also evaluated during the same episodes.

The different preferences cause clearly distinguishable distributions of the measured data, which are visualized by the boxplots. One can recognize how the *Safe* feature results in slower commute, relatively low lane changes, bigger front and rear TIV[3]. Definitely interesting is the behavior of the *Speed keeper* controlled by ϕ_2. It concentrates on the reference speed and thus has a slight speed error and higher average value. The existing difference can be originated form the different traffic situations and the constantly changing desired speed. The *Lane changer* also behaves as expected. It changes lanes whenever possible, almost 100% of all steps. This, however, affects the other parameters like the TIV values. As mentioned above, the constant lane-change is not desired behavior, but it perfectly shows how well the successor feature-based learning works.

[3]*TIV* means time-in-between vehicles

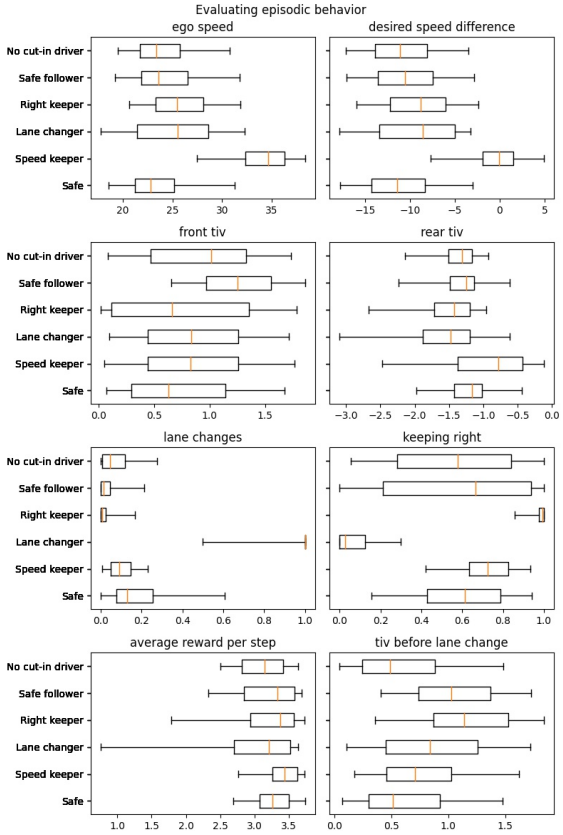


Fig. 7: Behavior difference of features in 100 runs

The *Right keeper* similarly satisfies our belief and, when possible, stays on the most-right lane. The deviation again is induced by the random starting lane of the EGO vehicle and the traffic scenes, which do not always make it feasible to occupy the ideal lane instantly. Finally, the *Safe follower* and *No cut-in driver* make exciting numbers in the front and rear TIV boxplots. It shows that limited by the simulation scenarios, the corresponding features (ϕ_5, ϕ_6) attempt to keep a safe time-in-between vehicles.

After investigating the successor features and their distinct way of acting, we conduct the same 100 episodes with several different preferences. Here, the SFs are combined and thus the $Q_w(s, a)$ is the dot-product of $\mathbf{w}$ and ψ_i as shown in (4). Figure 8 gives some examples for different preference behavior comparisons in a more realistic desired behavior setting. The bars show the successfully completed episodes from 100 runs compared between the Q-learning baseline, and both of the SF-based agents. The plot further explains the cause of the early termination achieved by the agents. To better evaluate the achieved performance, we compare not only our DFRL model with the basic Q-learning agent, but our implementation of FastRL as well. All of the baselines use $\mathbf{w} = [1, 1, -0.5, 0.5, 0.5, 0.5]$ as preference and hence, a reward function of $R(t) = r_1 + r_2 - 0.5r_3 + 0.5r_4 + 0.5r_5 + 0.5r_6$. In the case of the Q agent, this means a complex reward function, and it is equal with the one it experiences during training. Meanwhile, for the *FastRL and DFRL baselines* only the evaluation is done with this preference vector. Our

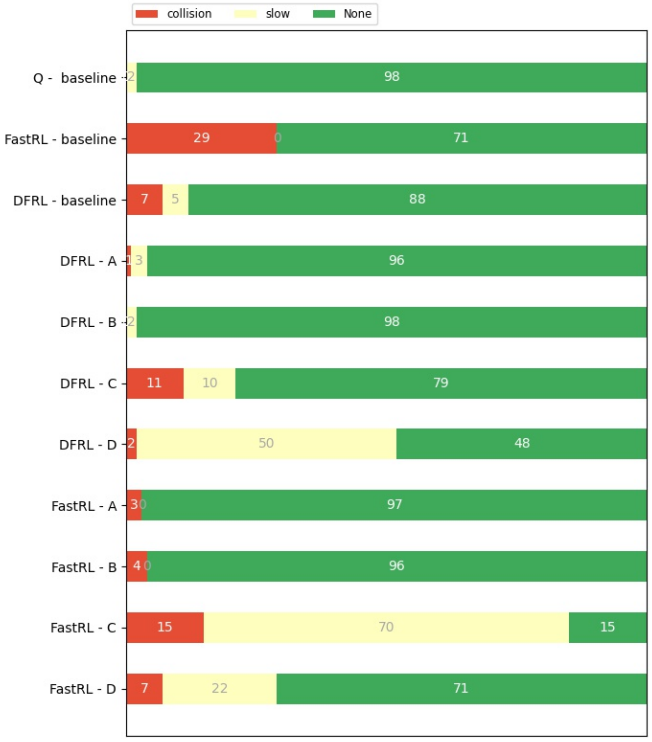


Fig. 8: Success rate of different preference runs

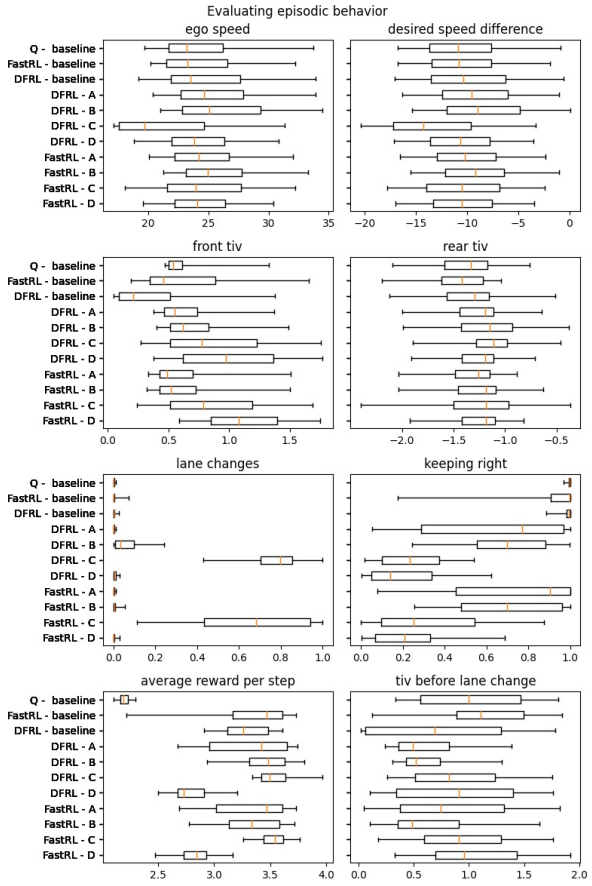


Fig. 9: Behavior difference of different preferences in 100 runs

evaluation in Figure 8 depicts that from 100 episodes, the Q-baseline performs the episodes 98% successfully, and fails 2 times due to slow speed. The FastRL baseline collides 29 times. Parallel, during the same simulations, our DFRL model improves the previous performance and gets only 7 collisions and 5 slow speed termination. The slow speed terminations are usually due to traffic reasons, where all the lines are blocked with slowly overtaking vehicles. In the video review, it is clear that the Q version chose the safer way and stayed mainly in the most-right lane, thus maximizing the overall reward. Meanwhile, the FastRL and DFRL baselines, change lanes and dare to take a bit riskier actions before the too slow other environmental vehicles hinder the reasonable commute. It must be stated, that although our SF-based agents perform worse on the scenarios, they were not trained with these tasks, thus it is new to them. Moreover, this 88 and 71 % performance gives a great insight of how task composition can be rapidly created by using SFs. The rest of Figure 8 compares the performance of the FastRL and DFRL agents with different preferences, where episodes without termination (None) are considered successful. The A - D decodes the following preference vectors:

$$\mathbf{w}_A = [1.0, 1.0, -0.5, 0.0, 1.0, 1.0],$$
$$\mathbf{w}_B = [1.0, 1.0, 0.0, 0.0, 1.0, 1.0],$$
$$\mathbf{w}_C = [1.0, 1.0, 0.5, 0.0, 1.0, 1.0],$$
$$\mathbf{w}_D = [1.0, 0.0, -0.5, -0.5, 1.0, 1.0]$$

For instance, *Preference A* is similar to w_Q, however it does not care about keeping right, and has a higher interest in the safe following and cut-in distance. As seen on the plot, this preference change results in only 4 terminations, which in case of our DFRL is mostly slow speed. Compared to *Preference B*, where the lane-change is not punished, we achieve only 2 terminations, comparable to the Q-baseline performance and outperforms FastRL. A further observation comes from *Preference C*. We reward lane-changes and thus induce more collisions and termination. Perhaps, the most significant difference can be seen here between the two SF-based agents, DFRL being on top. In case of *Preference D* we discourage keeping right and lane-change, meanwhile do not care about the desired speed. The encountered results show high termination due to slow commute, which is mostly caused by the agent following a vehicle that are designed to commute with low speed. We can state that our improved algorithm DFRL outperforms in every case the implementation adaptation of FastRL. This means it learns more meaningful SF representations, and generalizes better to new tasks. Furthermore it has smaller termination rate due to collision. This finding coincides with the expectations derived from the training/evaluation figures, and lower loss function.

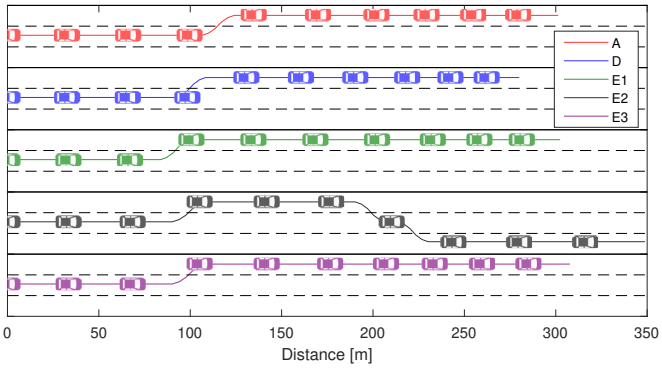
To be thorough regarding the full evaluation of the agents, Figure 9 characterizes the episodic parameters seen above for feature comparison. Looking at the values, one can gain more intuition on the induced behaviors of the features. The same values are monitored and depicted with boxplots. Medians are marked with yellow, and the 2σ deviation is shown by the lines outside the boxes. First, let us have a look at the *baselines*. It

is clear from the boxplots that all agents commute with similar speed and desired speed difference. The front TIV shows some difference between them, where DFRL has the lower values. Observing the average rewards one can conclude that the Q-baseline plays it safe, just as expected, meanwhile the SF-based agents reach multiple times more rewards per step. The most important result is that from feature-based learning and then composing an abstract behavior, our agents could commute with reaching more rewards than the one trained on the behavior. Despite this and the unseen task, as seen in Figure 8 the performance of DFRL is not significantly worse. It is insightful to evaluate the *Preferences A - D* as well. With the combined preferences, and learnt policies the agents do not follow the desired speed that accurately.

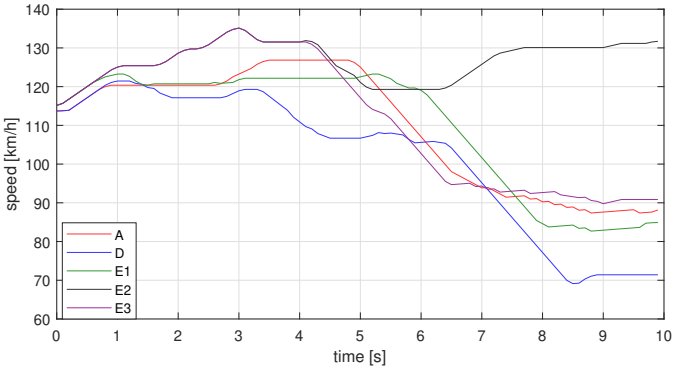
The reason is that all policies promote a behavior whilst learning from the mistakes of the other policies. When we inference the agents, we select the policy with the highest Q value, thus it can happen, that the safe policy π_1 discourages high speeds and present the highest Q value. The most evident influence on the parameters is caused by the lane-change preference value. When we have negative preference, the agents minimize the lane-changes, however, by *Preference B* a frequent lane-change is observable. Similarly the sign of the feature of keeping right can be estimated based on the plots. Positive preferences cause mostly steps spent in the most-right lane. Last, but not least, the front and back TIV is similar for the SF-based agents, nevertheless, a higher weight results in bigger TIV values. In terms of rewards, we can see that both FastRL and DFRL has high values, DFRL having smaller variance. The average reward however can only be interpreted with the preference pairs, due to addition of the reward components are not equal among the preferences A - D.

To visualize the different behaviors delivered by the agents depending on various weights, Figure 10a provides an overtake scenario, where the red car is the ego vehicle, which should overtake the car in front. For this demonstration we selected Preference A and D ($\mathbf{w_A}$ and $\mathbf{w_D}$ and added 3 extra weight vectors $\mathbf{w_{E1}} = [1, 1, -0.5, -0.5, 1, 1]$, $\mathbf{w_{E2}} = [1, 2, -0.3, 0, 0.2, 1]$ and $\mathbf{w_{E3}} = [1, 2, -0.5, 0, 0.5, 1]$. In Figure 10b the trajectories of these agents are visualized from the starting position during the next 10 seconds, which meet the expectations. *A* and *E3* are not expected to keep right, *D and E1* are discouraged to do so. Although neither is *E2*, due to the minor punishment for a lane change, it proceeds to the rightmost lane because it can keep the speed and safe distances better. Figure 10c depicts the speed profile of the cars during the monitored time window. *D* has no interest in keeping the desired speed, so it adjusts its speed to the traffic. Meanwhile, *E2* gets to the farthest due to keeping a higher speed and considers the lane change worthwhile to exhaust other rewards later. We think this highlights the essence of the whole method. We achieve new behavior by changing the preferences rapidly, without further training. A short video of this scenario is presented in https://www.youtube.com/watch?v=8NiWP_xNQBI.

(a) Initial scenario



(b) Trajectories of the agents



(c) Speed profiles of the agents

Fig. 10: Comparing different preferences of the same agent

VI. Conclusion

Our work reviewed the literature of successor features, general policy evaluation, and improvement, introduced the FastRL methodology adaptation, and designed a possible way to apply it in highway scenario training. A potential decomposition of the reward function is proposed and evaluated for the application in vehicle control realm. Furthermore, an improvement of the FastRL model is presented (DFRL), where the learning is stabilized by using a double architecture, similar to DDQN. We compared the achieved behavior of the agents with a general Q-Learning baseline, and outline the advantages of using the former solutions in case of multi-objective learning. Additionally, the environmental consideration as the basis of the whole domain transfer is detailed in our paper. In the results section, we point out how much better the credit assignment of the different tasks could be if the tasks are learned one by one, distinctly and later composed together for new challenges. DFRL outperforms the modified and adopted FastRL in this domain on many levels, which we emphasised throughout the paper. Besides, we provided experiments with different preferences of the same agent and

demonstrated how the learned behavior could quickly change and be changed simply by modifying the preference vector. With feature-based training, the agent gets more flexible, and its behavior can be altered rapidly, without additional learning, saving us hundreds of hours of training time. This can be a desirable property when designing reward composition, since the search through the possible reward functions does not require retraining. The discussed results make feature-based training of AI-controlled AVs attainable, where the appropriate skills and their expeditious combination can have considerable advantages in personalization, driving style management and adaptive reflection of changing conditions. Code for the training can be found open-sourced on GitHub [4].

REFERENCES

[1] D. Silver and et al., "A general reinforcement learning algorithm that masters chess, shogi and Go through self-play," 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

[2] O. Vinyals, I. Babuschkin, and et al., "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[3] OpenAI, :, C. Berner, and et al., "Dota 2 with Large Scale Deep Reinforcement Learning," *arXiv*, dec 2019. [Online]. Available: http://arxiv.org/abs/1912.06680

[4] J. Schrittwieser and et al., "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model," *arXiv*, nov 2019. [Online]. Available: http://arxiv.org/abs/1911.08265

[5] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the Atari Human Benchmark," *arXiv*, mar 2020. [Online]. Available: http://arxiv.org/abs/2003.13350

[6] W. B. Knox, A. Allievi, H. Banzhaf, F. Schmitt, and P. Stone, "Reward (Mis)design for Autonomous Driving," apr 2021. [Online]. Available: http://arxiv.org/abs/2104.13906

[7] A. Barreto, S. Hou, D. Borsa, D. Silver, and D. Precup, "Fast reinforcement learning with generalized policy updates," *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30 079–30 087, dec 2020.

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

[9] T. Tettamanti, I. Varga, and Z. Szalay, "Impacts of autonomous cars from a traffic engineering perspective," *Periodica Polytechnica Transportation Engineering*, vol. 44, no. 4, pp. 244–250, 2016. [Online]. Available: https://pp.bme.hu/tr/article/view/9464

[10] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, p. 103535, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370221000862

[11] M. Henaff, A. Canziani, and Y. LeCun, "Model-Predictive Policy Learning with Uncertainty Regularization for Driving in Dense Traffic," *7th International Conference on Learning Representations, ICLR 2019*, jan 2019. [Online]. Available: http://arxiv.org/abs/1901.02705

[12] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 7151–7160, nov 2019. [Online]. Available: http://arxiv.org/abs/1911.10868

[13] S. Aradi, "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles," 2020. [Online]. Available: http://arxiv.org/abs/2001.11231

[14] J. Chen, B. Yuan, and M. Tomizuka, "Model-free Deep Reinforcement Learning for Urban Autonomous Driving," *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pp. 2765–2771, apr 2019. [Online]. Available: http://arxiv.org/abs/1904.09503

[15] P. Wang, H. Li, and C.-Y. Chan, "Continuous Control for Automated Lane Change Behavior Based on Deep Deterministic Policy Gradient Algorithm," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2019-June, pp. 1454–1460, jun 2019. [Online]. Available: http://arxiv.org/abs/1906.02275

[4] https://github.com/szkLaszlo/fastrl_training

[16] L. Szőke, S. Aradi, T. Bécsi, and P. Gáspár, "Driving on highway by using reinforcement learning with cnn and lstm networks," in *2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*, 2020, pp. 121–126.

[17] L. Szoke, S. Aradi, T. Becsi, and P. Gaspar, "Vehicle control in highway traffic by using reinforcement learning and microscopic traffic simulation," in *2020 IEEE 18th International Symposium on Intelligent Systems and Informatics (SISY)*, 2020, pp. 21–26.

[18] 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.

[19] S. S. Gu and et al., "Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp. 3846–3855. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/a1d7311f2a312426d710e1c617fcbc8c-Paper.pdf

[20] P. Wang and C. Chan, "Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6.

[21] M. Zhu, X. Wang, and Y. Wang, "Human-like autonomous car-following model with deep reinforcement learning," *Transportation Research Part C: Emerging Technologies*, vol. 97, pp. 348–368, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X1830055X

[22] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Cooperation-aware reinforcement learning for merging in dense traffic," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 3441–3447.

[23] B. Kővári, F. Hegedüs, and T. Bécsi, "Design of a Reinforcement Learning-Based Lane Keeping Planning Agent for Automated Vehicles," *Applied Sciences*, vol. 10, no. 20, p. 7171, oct 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/20/7171

[24] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, and T. Schaul, "Successor features for transfer in reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[25] A. Barreto, D. Borsa, J. Quan, and T. Schaul, "Transfer in deep reinforcement learning using successor features and generalised policy improvement," in *International Conference on Machine Learning*. PMLR, 2018, pp. 501–510.

[26] A. Barreto, D. Borsa, and S. Hou, "The option keyboard: Combining skills in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[27] P. Dayan, "Improving Generalization for Temporal Difference Learning: The Successor Representation," *Neural Computation*, vol. 5, no. 4, pp. 613–624, jul 1993. [Online]. Available: https://www.mitpressjournals.org/doix/abs/10.1162/neco.1993.5.4.613

[28] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman, "Deep Successor Reinforcement Learning," jun 2016. [Online]. Available: http://arxiv.org/abs/1606.02396

[29] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-September. Institute of Electrical and Electronics Engineers Inc., dec 2017, pp. 2371–2378.

[30] R. Ramesh, M. Tomar, and B. Ravindran, "Successor options: An option discovery framework for reinforcement learning," *arXiv preprint arXiv:1905.05731*, 2019.

[31] B. Van Niekerk, S. James, A. Earle, and B. Rosman, "Composing value functions in reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6401–6409.

[32] D. Borsa, A. Barreto, J. Quan, D. Mankowitz, R. Munos, H. Van Hasselt, D. Silver, and T. Schaul, "Universal successor features approximators," *arXiv preprint arXiv:1812.07626*, 2018.

[33] S. Hansen, W. Dabney, A. Barreto, T. Van de Wiele, D. Warde-Farley, and V. Mnih, "Fast task inference with variational intrinsic successor features," *arXiv preprint arXiv:1906.05030*, 2019.

[34] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning," nov 2016. [Online]. Available: http://arxiv.org/abs/1611.02779

[35] A. S. Vezhnevets and et al., "FeUdal Networks for Hierarchical Reinforcement Learning," *34th International Conference on Machine Learning, ICML 2017*, vol. 7, pp. 5409–5418, mar 2017. [Online]. Available: http://arxiv.org/abs/1703.01161

[36] T. Zahavy, A. Hasidim, H. Kaplan, and M. Com, "Planning in Hierarchical Reinforcement Learning: Guarantees for Using Local Policies Yishay Mansour," Tech. Rep., jan 2020. [Online]. Available: http://proceedings.mlr.press/v117/zahavy20a.html
[37] 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*, 2017, pp. 1–16.
[38] "PreScan — TASS International." [Online]. Available: https://tass.plm.automation.siemens.com/prescan
[39] "Mechanical Simulation." [Online]. Available: https://www.carsim.com/
[40] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/
[41] D. Krajzewicz, G. Hertkorn, P. Wagner, and C. Rössel, "SUMO (Simulation of Urban MObility)," 2002. [Online]. Available: https://elib.dlr.de/6661/2/dkrajzew{_}MESM2002.pdf
[42] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO – Simulation of Urban MObility An Overview," 2011. [Online]. Available: https://elib.dlr.de/71460/1/SUMO{_}survey{_}SIMUL2011.pdf
[43] A. Alizadeh and et al., "Automated lane change decision making using deep reinforcement learning in dynamic and uncertain highway environment," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 1399–1404.
[44] S. Nageshrao, H. E. Tseng, and D. Filev, "Autonomous highway driving using deep reinforcement learning," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 2326–2331.
[45] Z. Bai, W. Shangguan, B. Cai, and L. Chai, "Deep reinforcement learning based high-level driving behavior decision-making model in heterogeneous traffic," in *2019 Chinese Control Conference (CCC)*, 2019, pp. 8600–8605.
[46] V. Mnih and et al., "Playing Atari with Deep Reinforcement Learning," dec 2013. [Online]. Available: https://arxiv.org/abs/1312.5602v1
[47] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 2094–2100, sep 2015. [Online]. Available: https://arxiv.org/abs/1509.06461v3

Laszlo Szoke received the B.Sc. degree in mechatronics in 2018, the M.Sc in autonomous vehicle control engineering in 2020, both from Budapest University of Technology and Economics, Budapest, Hungary.

He is currently pursuing the Ph.D. at the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, meanwhile simultaneously being a Ph.D researcher at Robert Bosch Kft., Budapest, Hungary. His research interest includes artificial intelligence, machine learning, reinforcement learning, mechatronics and vehicle control.

Szilárd Aradi (M'14) received the M.Sc. degree in 2005 and Ph.D. in 2015 from the Budapest University of Technology and Economics, Budapest, Hungary, where he is currently working with the Department of Control for Transportation and Vehicle Systems.

Since 2021, he has been an Associate Professor at the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics. His research interests include embedded systems, communication networks, vehicle mechatronics, and reinforcement learning. His research and industrial works have involved railway information systems, vehicle on-board networks, and vehicle control.

Tamas Becsi (M'14) Tamás Bécsi received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics, Budapest, Hungary, in 2002 and 2008, respectively.

Since 2005, he has been an Assistant Lecturer and since 2014, he is an Associate Professor, at the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics. His research interests include linear systems, embedded systems, traffic modeling, and simulation. His research and industrial works have involved railway information systems and vehicle control.

Péter Gáspár received the M.Sc. and Ph.D.degrees from the Budapest University of Technology and Economics (BME), Faculty of Transportation Engineering and Vehicle Engineering (KJK), in 1985 and 1997, respectively, and the D.Sc. degree in control from the Hungarian Academy of Sciences (MTA), in 2007. Since 1990, he has been a Senior Research Fellow with the Institute for Computer Science and Control (SZTAKI). Since 2016, he has also been a Research Professor. In 2004, he became the Head of the Vehicle Dynamics and Control Research Group and then in 2017, he became the Head of the Systems and Control Laboratory, SZTAKI. He was habilitated at the BME, in 2008, and he was appointed as the University Professor. Since 2013, he has also been the Head with the Department of Control for Transportation and Vehicle Systems (KJIT), BME KJK. His research interests include linear and nonlinear systems, robust control, multi-objective control, system identification, and identification for control and artificial methods. His research and industrial works have involved mechanical systems, vehicle structures, and vehicle dynamics and control. Since 2016, he has also been a Corresponding member of MTA. He is also a member of the IFAC Automotive Control and Transportation Systems Technical Committee, and the Chair of the International Federation of Automatic Control (IFAC) Hungary National Member Organization.