Automatically Learning Fallback Strategies with Model-Free Reinforcement Learning in Safety-Critical Driving Scenarios

Ugo Lecerf[1,2], Christelle Yemdji Tchassi[1], Sébastien Aubert[1], Pietro Michiardi[2]

***Abstract*— When learning to behave in a stochastic environment where safety is critical, such as driving a vehicle in traffic, it is natural for human drivers to plan fallback strategies as a backup to use if ever there is an unexpected change in the environment. Knowing to expect the unexpected, and planning for such outcomes, increases our capability for being robust to unseen scenarios and may help prevent catastrophic failures. Control of Autonomous Vehicles (AVs) has a particular interest in knowing when and how to use fallback strategies in the interest of safety. Due to imperfect information available to an AV about its environment, it is important to have alternate strategies at the ready which might not have been deduced from the original training data distribution.**

In this paper we present a principled approach for a model-free Reinforcement Learning (RL) agent to capture multiple modes of behaviour in an environment. We introduce an extra pseudo-reward term to the reward model, to encourage exploration to areas of state-space different from areas privileged by the optimal policy. We base this reward term on a distance metric between the trajectories of agents, in order to force policies to focus on different areas of state-space than the initial exploring agent. Throughout the paper, we refer to this particular training paradigm as learning fallback strategies.

We apply this method to an autonomous driving scenario, and show that we are able to learn useful policies that would have otherwise been missed out on during training, and unavailable to use when executing the control algorithm.

I. INTRODUCTION

Implementing a controller for AVs in a driving scenario is met with many challenges: both from the point of view of perception and control [1]. As in most applications of real-world RL, the uncertainty linked to the perception of the agent's environment must be considered for an effective controller to be developed. Even with the best possible road maps and sensors, it is impossible to eliminate all sources of uncertainty from a driving scenario, be they epistemic from imperfections in the vehicle's sensors, or aleatoric from the unpredictable interactions with other drivers [2].

Autonomous driving requires a strong notion of safety, and notably robustness with respect to unexpected changes in the agent's environment. For example, sensor perception quality can be heavily susceptible to adverse weather conditions [3]. Because of this the optimal behaviour is likely to change dynamically according to the vehicle's inputs, and a satisfactory control algorithm must be able to adapt on the fly. Safety criteria in autonomous driving applications are traditionally based on perceiving when a situation is no longer able to be handled by the acting controller, and then handing over the controls to either the driver, or a special-case controller. For example, [4] implement a deep neural network to detect the probability of a catastrophic outcome when following recommended actions, whereas other approaches look to estimate the confidence of a policy by estimating quantiles of the return distribution [5], or using an ensemble of networks to gauge whether an input was included in the policy's training data distribution [6].

RL techniques have been shown to be able to tackle the task of control in progressively more complex environments [7]. RL algorithms learn by optimizing their expectation of performance in an environment. In most cases, such as board games [8] or video games [9], the environment in which we seek to obtain the optimal behaviour can be modeled as a Markov Decision Process (MDP) with no loss of generality in the solution found by the RL agent. Through advancements in target updates [10], as well as agent architectures [11], RL agents have become increasingly efficient at finding the optimal solution to MDPs, even when requiring high degrees of exploration, where the optimal sequence of actions is hard to find [12].

Stochastic control environments such as driving scenarios are more difficult to optimize, given the probabilistic nature of both the observation and transition dynamics. Stochastic environments may be modeled as Partially Observable MDPs (POMDPs) [13]. Solving POMDPs is possible with methods combining learning and planning, such as [14]. However, a change in the values of the stochastic model parameters, for example a change in a vehicle's sensor accuracy, scene obstruction, or simply unplanned behaviour from another vehicle, may induce a sharp drop in the agent's performance due to its inability to generalize well to new environment parameters. Having access to a model of the environment dynamics allows us to use planning algorithms, such as MCTS [15], alongside learning to both increase sample efficiency, and have access to a better representation of the environment's state-space structure (model-based RL). In cases where planning is possible, it is much easier to find alternative strategies for an agent to solve its environment and hence be able to better adapt to eventual changes in the state-space [16].

In the model-free setting we consider in this paper however, efficient learning is notoriously hard, one of the factors being the increased variance of target updates from of the

[1]Renault Software Labs, Valbonne, France {ugo.lecerf, christelle.yemdji-tchassi, sebastien.s.aubert}@renault.com

[2]Eurecom, Biot, France. {ugo.lecerf, pietro.michiardi}@eurecom.fr

Published in Proceedings of 7^{th} edition of International Conference on Machine Learning Technologies (ICMLT) 2022.

lack of a planner able to average out return values from multiple simulated runs. A lack of an environment model also reduces the ability for an agent to adapt to changes to the environment after training, since we are unable to use a planner to explore the new dynamics before acting. Works such as [17] and [18] use deep recurrent Q-networks in order to build up knowledge of the MDP's state over time, and use a latent representation of the history of states and actions in order to inform an agent about its current state. In the case of stochastic parameter change, we are unable to predict how the returns of a policy will be affected.

This work is focused on the problem of training an agent to be robust to changes in the uncertainty affecting local areas of state-space affecting either the transition or observation models in the MDP-defined environment. When uncertainty arises in a local area of an MDP's state-space, we can try to learn policies exploiting different areas of the state-space, such that we maximize the probability that the return of at least one of the available policies remains unaffected.

Our main contribution is to implement a novel framework which aims to learn alternate policies, referred to as fallback strategies, which exploit different areas of state-space than the optimal policy. These fallback strategies serve as potential alternatives to safely navigate the environment, in the case of a change in the defining elements in the MDP model. We consider the model-free RL setting, and provide experimental validation for this framework by testing it in a driving scenario.

This paper is organized as follows: in section II we present the standard framework and notations used in RL. Section III presents existing methods for affecting agent behaviour in RL systems. Section IV presents our contribution of learning fallback strategies, the main topic of this paper, where we define multi-policy learning in the context of learning and retaining sub-optimal solutions. Experiments and results are presented in sections V and VI respectively, and section VII contains our conclusion along with future work.

II. NOTATION

A finite MDP is defined by the following elements:

- Finite set of states $s \in \mathcal{S}$. States are indexed by the timestep at which they are encountered: s_t.
- Finite set of actions $a \in \mathcal{A}$. Actions are also indexed by their respective timesteps: a_t.
- Transition model $\mathcal{T}(s,a,s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$, representing the probability of passing from s to s' after taking action a_t, $P(s_{t+1} = s' | s_t = s, a_t = a)$.
- Immediate reward function $R(s,a,s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.
- Discount factor $\gamma \in (0,1]$, controlling the weight in value of states further along the Markov chain.

A POMDP is further augmented by an observation model $\mathcal{O}$, when we no longer have access to the true state s_t, but an observation thereof $o_t = \mathcal{O}(s_t)$.

Actions in the MDP are taken by a (stationary) policy $\pi : \mathcal{S} \to \mathcal{A}$ mapping states to actions. The value of a state under a policy π, is given by the state-value function $V^\pi : \mathcal{S} \to \mathbb{R}$, which represents the expected future discounted sum of rewards, if policy π is followed from s:

$$V^\pi(s) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right], \quad s_0 = s, a_t = \pi(s_t), s_{t+1} \sim \mathcal{T}(s_t, a_t). \tag{1}$$

Actions are chosen by the policy π, so as to maximize the action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which assigns values to actions according to the value of the states that are reached:

$$Q^\pi(s_t, a_t) := \mathbb{E}_{s_{t+1}}\left[R(s_t, a_t, s_{t+1}) + \gamma \cdot V^\pi(s_{t+1})\right], \quad s_{t+1} \sim \mathcal{T}(s_t, a_t). \tag{2}$$

We use the Q-function in order to define the optimal policy, which we denote π^*, as the policy taking actions that maximize the action-value function: $\pi^*(s) := \arg\max_{a \in \mathcal{A}} Q^{\pi^*}(s,a)$. Equation (2) highlights that the state-action value function is a sort of one-step look-ahead of the value of the next possible state s_{t+1}, in order to determine the value of actions in the current state s_t. We denote the history of states visited by a policy π (trajectory through state-space) as:

$$\mathcal{H}_\pi := \{s_t\}_{t \in [0,T]}, \quad s_0 \in \mathcal{S}, a_t = \pi(s_t), s_{t+1} \sim \mathcal{T}(s_t, a_t). \tag{3}$$

III. RELATED WORK

The main element susceptible to engineering in MDPs is the reward function, which indirectly defines the agent's goal; learning about multiple goals can be translated into learning to solve MDPs with multiple reward functions. One of the most common uses for augmenting the reward function is to artificially boost the RL agent's degree of exploration [19], [20]. These methods dynamically change the value of the immediate rewards an agent gains, in order to encourage actions towards areas of state-space which are deemed more important. Our approach is similar to these in that we base an extra reward term on an external factor in order to affect the behaviour of exploring agents.

Although we base our approach on engineering the MDP's reward function, the problem we are tackling with our approach is distinct from the exploration problem in RL, and our objective is not to find an optimal exploration scheme. Exploration boosting methods are used within the context of a single environment, in order to avoid having the agent fall victim to being trapped within a local optima. Our goal however is to have the RL algorithm be able to retain sub-optimal solutions to the environment, once the training regime is ended. In a similar spirit to how TRPO [21] seeks to mitigate the concern that small changes in the parameter space may lead to sharp drops in performance, we seek to mitigate sharp drops in performance resulting from changes to stochastic environment parameters.

Moreover, the approach explored in this paper aims to learn policies with different behaviours, given the same initial environment conditions. Compared to methods which learn

only the environment's optimal policy, this allows us to have sub-optimal policies to use as fallback strategies which we can switch through using a hierarchical structure. This has an advantage over attempting to have a single policy learn to generalize over the space of MDPs, since this task (meta-learning) requires many more samples in order for an agent to achieve a good performance [22], our approach is more attractive for certain safety-critical applications.

Many previous works aim for agents to generalize to different goals within the same MDP [23], or even self-discover sub-goals that make learning more efficient [24]. In these cases, a difference in goals is translated through a change in the reward function for a goal-state g: $R(s,a,g) = R_g > 0$. These methods take advantage of the underlying structure in the goal-space in order to increase sample efficiency [25], as well as the generalization capabilities of agents to tasks with goals that were not present during training [26]. They provide a good way of finding different behaviours within a same environment, where the modification of the reward function is intended for a control algorithm to be robust in terms of changing objectives. Our problem statement focuses on dealing with scenarios when the objective (i.e. goal) of an environment remains the same, but the environment is expected to change in some unknown way.

IV. LEARNING FALLBACK STRATEGIES

In this section, we motivate the use of a new RL paradigm aimed at learning policies in the model-free setting which are robust to changes in the environment's stochastic parameters, affecting its dynamics locally. Note that because of the lack of access to environment models, we can only target the sampled transitions that are stored in the replay buffer used during training.

A. Agents' Behaviour

We can think of an agent's behaviour as the trajectory through state-action-space that results from following a policy π. Policies with similar behaviours (condition which we formally define later on) will have similar trajectories through state-action-space. During training while an agent's policy is improving, the agent's behaviour will evolve, until its optimal policy is reached.

When considering learning multiple policies in an environment, we must determine which ones are useful to us. We consider that any policy which sufficiently solves an MDP is of interest. *Sufficiently solved* is a criterion that may vary between different MDPs, hence we identify two ways this can be implemented into RL algorithms:

Definition (Sufficiently Solved). An agent's policy π sufficiently solves its respective MDP, if either of the following conditions are met, depending on the nature of the control task:

- Agent is able to reach a specific goal-state $g \in \mathcal{S}$.
- Agent is able to expect accumulated discounted rewards above a threshold score $V^{\pi}(s_0) > G_{min} \in \mathbb{R}$, for s_0 sampled from initial state distribution (Note that V^{π} is the true value function as in (1), not an approximation).

Based on this, we define what we mean by a valid strategy:

Definition (Valid Strategy). A valid strategy in an MDP, is the behaviour of a policy which sufficiently solves that MDP.

The use of either interpretation for *sufficiently solved* depends on the environment, and what it heuristically means to solve it. For example, in the case of an Atari game (e.g. Breakout), any behaviour from a policy which reaches above a threshold score, is typically considered to be a valid strategy. Another example would be an AV passing through an intersection, where any behaviour passing the intersection (reaching goal-state g) without collisions is a valid strategy. We use this definition to determine which policies are deemed useful during training.

Definition (Sub-optimal policy). A sub-optimal policy, denoted π_{sub}, is a policy whose expectation of return is within a margin ε, to that of the optimal policy π^*, at some given initial state s_0 sampled from the initial state distribution:

$$V^{\pi^*}(s_0) - V^{\pi_{\text{sub}}}(s_0) < \varepsilon. \tag{4}$$

Condition (4) is equivalent to saying that policy π_{sub} is a sub-optimal policy, whose behaviour is a valid strategy (in the threshold-score sense, $\varepsilon = V^{\pi^*}(s_0) - G_{min}$). For example in the case of an Atari game, any agent that achieves a score higher than the threshold, but less than the one obtained by the optimal policy π^*, verifies condition (4).

In order for the two policies to be considered as having different behaviours, they must be sufficiently different in the state-distributions that are encountered during execution. This implies the need for a metric $\mathcal{M}$ measuring the difference between agents' trajectories in state-space, which is not yet a part of standard reinforcement learning applications.

Definition (Sufficiently different behaviours). We can say that two policies, π_1, π_2 have $\mathcal{M}_d$-different behaviours, iff:

$$\mathcal{M}(\mathbb{E}[\mathcal{H}_{\pi_1}], \mathbb{E}[\mathcal{H}_{\pi_2}]) \geq d. \tag{5}$$

Condition (5) is equivalent to saying that the behaviours of π_1 and π_2 can be described as being heuristically different. In our approach, we base this heuristic on the similarity in terms of their respective trajectories through the MDP state-space. Setting a value $d \in \mathbb{R}$ is subjective: for instance, human experts may have arbitrary boundaries for when an agent's path through state-space is sufficiently different from a reference path, to consider both as having different behaviours. Condition (5) can be thought of as a non-parametric clustering with boundary d, where $\mathcal{M}(\cdot, \mathbb{E}[\mathcal{H}_{\pi_{ref}}])$ is the feature map in a policy's state-trajectory space, with respect to a reference policy π_{ref}. Once more, the correct segmentation of this space is subjective and may vary between experts based on experience.

We propose that in order to increase the robustness of an RL algorithm to changes in environment parameters, it should be able to learn sub-optimal strategies which have sufficiently different behaviours from each other policy in a training environment, satisfying both conditions (4) and

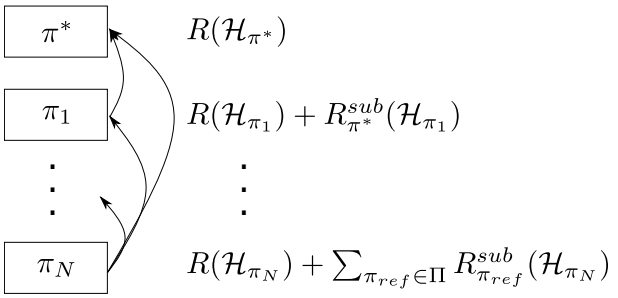


Fig. 1: Each subsequent pseudo-agent will have its relative pseudo-reward added onto its regular reward function, each extra term corresponding to another agent already present in that environment. This way, we verify that new agents have behaviours that are sufficiently different from those of all previous agents' policies.

(5). Changes in $\mathcal{T}$ or $\mathcal{O}$ may affect local areas of the environment's state-space differently, affecting some policies' expected returns more than others, depending on whether the introduced uncertainty affects their respective state-space paths. Condition (4) ensures that we only learn policies with satisfactory performance in the environment, whereas (5) aims to maximize the likelihood that at least one of the learned policies will have an expected return that is minimally affected by changes to either $\mathcal{T}$ or $\mathcal{O}$.

B. Fallback Strategies

Given an MDP, we wish to find the optimal policy along with a number *N* of sub-optimal ones. We consider situations where the agent is already able to find the optimal policy π^*, and demonstrate a method for finding policies π_{sub} that satisfy both (4) and (5).

The approach explored in this paper is to add an extra pseudo-reward term onto the regular reward for that environment, denoted $R^{sub}_{\pi_{ref}} \leq 0$, as a penalty to agents, for having similar state-trajectories to π_{ref}. We term the agents which are not seeking to learn the optimal policy, pseudo-agents. The pseudo-reward is based on the metric $\mathcal{M}$ between agents' state-space trajectories in order to satisfy (5). We use the following equation for the pseudo-reward to discourage pseudo-agents agents from copying other agents' expected path through state-space $\mathbb{E}\left[\mathcal{H}_{\pi_{ref}}\right]$:

$$R^{sub}_{\pi_{ref}}(\mathcal{H}_\pi) = -\frac{\alpha}{\mathcal{M}\left(\mathcal{H}_\pi, \mathbb{E}\left[\mathcal{H}_{\pi_{ref}}\right]\right) + \delta}, \qquad (6)$$

where $0 < \delta < 1$ avoids infinite penalties for exactly following $\mathbb{E}\left[\mathcal{H}_{\pi_{ref}}\right]$, giving the penalty an upper bound of $-\frac{\alpha}{\delta}$. π_{ref} is the reference policy, which may or may not be the optimal, according to the number of pseudo-agents. α is a scaling factor to adjust the amplitude of the pseudo-reward term, compared to the regular rewards. Pseudo-agents will be training concurrently to the optimal one, aiming to converge to distinct valid strategies within the same environment.

Fig. 1 illustrates the relationship between subsequent pseudo agents in the same environment. This approach can be extended to an arbitrary number *N* of pseudo-agents, by imposing condition (5) such that each subsequent agent has a sufficiently different behaviour to previous ones, hence every additional pseudo-agent will have one more reward term to compute. Although this adds complexity to the RL problem, the number *N* of total agents should remain reasonably limited: we should increase *N* according to the anticipated uncertainty on the environment parameters. $\pi_{ref} \in \Pi$ represents all previous agents (shown by the arrows in Fig. 1). Π is empty in the case of the optimal agent π^*, $\Pi = \{\pi^*\}$ for the 1st pseudo-agent, $\Pi = \{\pi^*, \pi_1\}$ for the 2nd pseudo-agent, and so on. For *N* pseudo-agents, this approach adds a computational cost of $o(N^2)$ in terms of pseudo-reward term computation. Increasing state-space size and dimensionality is susceptible to increase the number *N* of sub-optimal agents we wish to maintain as there are more opportunities for alternate valid strategies. However, *N* is limited by the number of expected changes in the MDP's state-space we wish the RL agent be robust to, hence the computational cost is expected to remain within the same order of magnitude as without the pseudo-reward implementation.

Implementing the additional pseudo-rewards will impact the new value-function estimate of pseudo-agents learning sub-optimal policies . So (4) should become:

$$V^{\pi^*}(s_0) - V^{\pi_{\text{sub}}}(s_0) < \varepsilon + \sum_{\pi_{ref}\in\Pi} R^{sub}_{\pi_{ref}}\left(\mathcal{H}_{\pi_{sub}}\right), \qquad (7)$$

such that π_{sub} would still be considered a valid sub-optimal policy.

Algorithm 1 gives a pseudo-code description of our implementation. $\pi_{ref} \in \Pi$ represents the same set of reference agents as in Fig. 1.

Algorithm 1 Learning Fallback Strategies with N pseudo-agents

```
Init π*, π_1, ..., π_N
while True do
    for π ∈ {π*, π_1, ..., π_N} do
        while episode not terminated do ▷ play episode
            a_t = π(s_t)
            s_{t+1} ~ T(s_t, a_t)
            r_t = R(s_t, a_t, s_{t+1})        ▷ regular step-reward
        end while
        r_pseudo = Σ_{π_ref∈Π} R^sub_{π_ref}(H_π)      ▷ pseudo-reward
        for t ∈ [0, T−2] do                    ▷ store in memory
            Memory(π) ← (s_t, a_t, r_t, s_{t+1})
        end for
        Memory(π) ← (s_{T−1}, a_{T−1}, r_{T−1} + r_pseudo, s_T)
    end for
end while
```

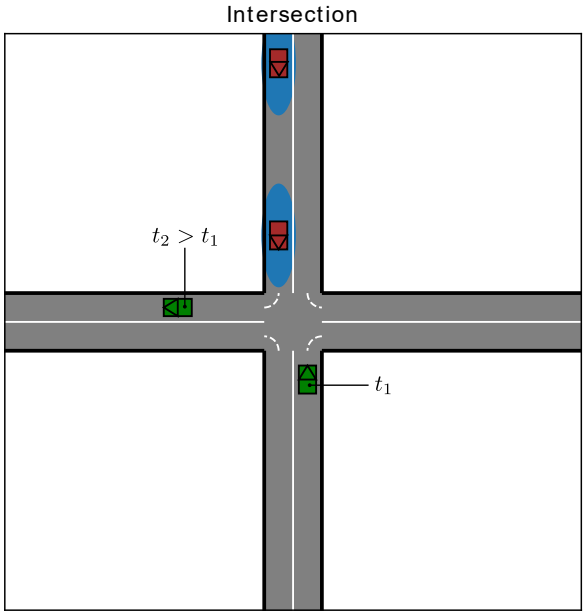


Fig. 2: Intersection environment. The ego vehicle (green) starts on the bottom-right lane, and makes a left turn. The ego vehicle can either speed up to pass in front of the oncoming target vehicles, or slow down to pass in-between them. The aim is to cross the intersection without crashing (t_2). Collision distance is given by the radius of the blue ellipses.

V. EXPERIMENTS

In this section we present a control task for an autonomous vehicle, to demonstrate the ability of our proposed method to discover and maintain valid sub-optimal policies. In this use-case, we limit ourselves to a single fallback strategy.

The environment consists of a 2-way intersection, where the agent's goal is to complete a left-hand turn, without crashing into any of the oncoming vehicles that cross straight through the intersection. Fig. 2 shows a frame of the environment with oncoming vehicles in the intersection. In driving scenarios, the controllable agent is usually referred to as the ego vehicle whereas the other uncontrollable vehicles are referred to as targets. In this scenario there are two main solutions for the ego to complete the task: the optimal policy in terms of performance is to speed up and pass before the first target vehicle, whereas the sub-optimal policy consists of slowing down and passing in-between the oncoming target vehicles. A change in scene detection may affect the variance in detected positions of the target vehicles, and cause the first strategy to be considered too risky to follow. In this case, learning a fallback strategy that may be less affected by a drop in target position confidence, may be considered safer and more useful.

In our use-case, the speed of target vehicles is constant (20 m/s). The ego's initial speed is also 20 m/s, and its action space corresponds to the following longitudinal acceleration values: $a \in \mathcal{A} = \{-4,-2,-1,0,1,2\}$ m/s^2. Since the path is already determined (making a left turn), the problem boils down to planning a speed profile for the ego which will complete the task in a minimum amount of time while avoiding catastrophic collisions. A collision is detected when the distance between the ego and target vehicles drops below a threshold value.

To penalize collisions and encourage faster episode termination, the reward is set-up as follows per time step t:

$$r_t = \begin{cases} -5 & \text{if collision} \\ -0.1 & \text{otherwise} \end{cases}. \tag{8}$$

We concurrently train an optimal agent π^*, along with one pseudo-agent π_1. We use deep Q-networks [8], using double Q-learning as well as non-prioritized experience replay [11]. The input state to the Q-network is the concatenation of the position x_{ego} and speed $\dot{x}_{ego}$ of the ego vehicle, along with the position and computed time-to-collision (ttc) of the 3 nearest targets:

$$s = \{x_{ego}, \dot{x}_{ego}, x_1, \text{ttc}_1, x_2, \text{ttc}_2, x_3, \text{ttc}_3\}.$$

Each of the components of s are normalized with respect to a maximum value.

We use the following path-metric $\mathcal{M}$ between the pseudo-agent, and the optimal agent's memory buffers (in practice we replace the expectation operator by the mean value over the last 100 samples of the agent's memory):

$$\mathcal{M}(\mathcal{H}_{\pi_1}, \mathbb{E}[\mathcal{H}_{\pi^*}]) = \int \left| \mu\left(\phi\left(\mathcal{H}_{\pi_1}\right)\right) - \mu\left(\phi\left(\mathbb{E}[\mathcal{H}_{\pi^*}]\right)\right) \right| d\phi(s), \tag{9}$$

where ϕ is a state-feature function, and μ is the density function over state features. In this example we use the speed of the ego vehicle as a state feature $\phi(s) = \dot{x}_{ego}$. The pseudo-reward received at the end of each episode by the corresponding pseudo-agent will be:

$$R^{sub}_{\pi^*}(\mathcal{H}_{\pi_1}) = -\frac{\alpha}{\mathcal{M}(\mathcal{H}_{\pi_1}, \mathbb{E}[\mathcal{H}_{\pi^*}]) + \delta} \tag{10}$$

with scaling factors:

$$\alpha = 1, \quad \delta = 0.1 \ .$$

These determine the relative weight of the pseudo-reward, with respect to the regular reward function R. A lower value for α will hardly penalize the pseudo-agent for having a similar state distribution to the reference agent, whereas higher weighting will make the pseudo-agent seek to have a highly different state-space trajectory, disregarding the original objective of the task given by the regular reward function. They are fixed by a rough initial sweep.

VI. RESULTS

Fig. 3 shows the training scores for both agents. We clearly see the second agent's convergence to its optimal performance 'lags' behind that of the optimal agent. This is most likely due to the fact that the pseudo-reward term $R^{sub}_{\pi^*}$ depends on the states present in the memory buffer for π^*.

Fig. 3: Training scores for both agents. Each is trained for 300k steps. π^* is the optimal agent, π_1 is the pseudo-agent.

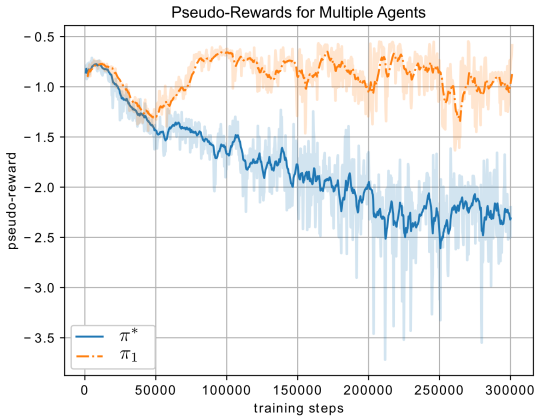


Fig. 4: Pseudo-reward, $R^{sub}_{\pi^*}$ calculated for both agents. The values for π^* are computed only for comparison to the values used by π_1.

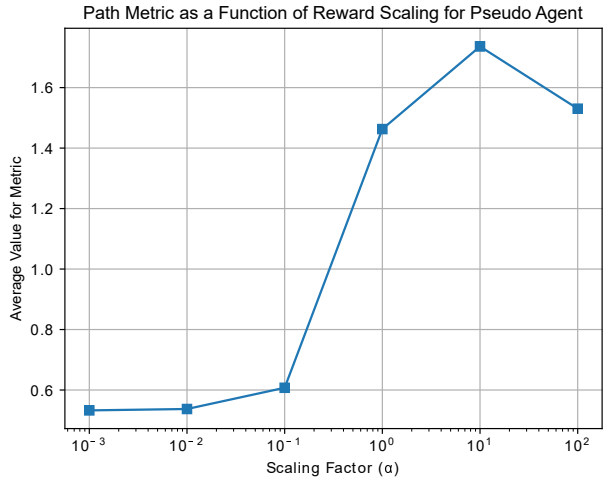


Fig. 5: Average values for $\mathcal{M}$ on the final 100 episodes of each pseudo-agent, for different pseudo-reward scaling α.

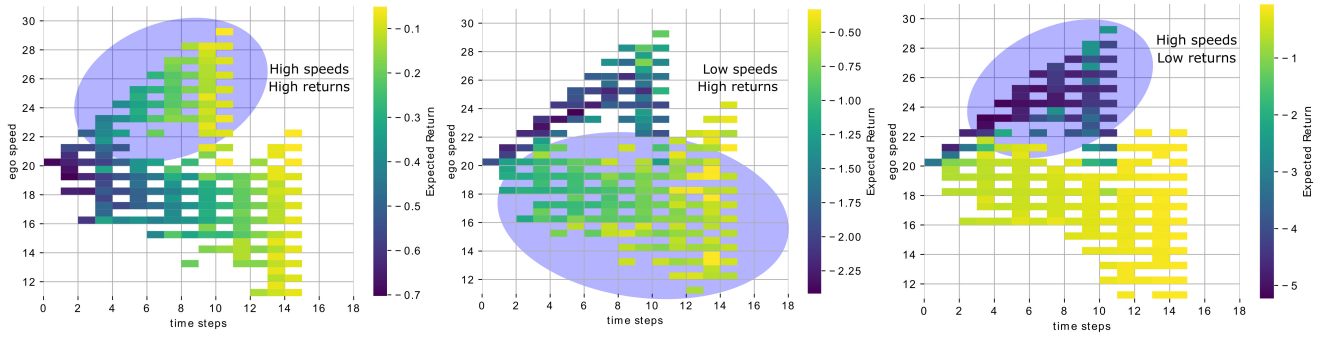
Hence $R^{sub}_{\pi^*}$ cannot be stable until π^* has converged, and there is little change in its memory's state distribution. This prevents the corresponding pseudo-agent, π_1, from converging earlier. Interestingly, π_1 reaches its best score before π^*: in our implementation, the optimal path (accelerating before the 1st target vehicle) is harder to find through exploration than the sub-optimal one (passing in-between the target vehicles). Once the pseudo-reward is stable enough to dissuade the pseudo-agent from copying the optimal agent's path, it is faster to converge to its new optimal policy (being the original sub-optimal policy).

Looking at Fig. 4, we see that both agents' path metrics are similar for approximately the first 50k steps, after which their policies start diverging. This means both had similar state distributions (mainly due to a high degree of random exploration) until that point. The value of $R^{sub}_{\pi^*}(\mathcal{H}_{\pi^*})$ keeps decreasing while the optimal agent is converging to its best policy, and levels out once it converges to its peak performance at approximately 200k steps. $R^{sub}_{\pi^*}(\mathcal{H}_{\pi_1})$ however levels out quite soon, closely corresponding to π_1 reaching its peak performance. Though it is still changing due to the changing state distribution in π^*'s memory buffer, this is hardly seen on the plotted values compared to the random oscillations.

Fig. 5 shows the effect that modifying the parameter α has on the resulting policy learned by the pseudo-agent π_1. As mentioned in section IV, the pseudo-reward must be scaled in such a way to fulfill both conditions (5) and (7). Learning with pseudo-agents can fail if it is not scaled properly. We see that there is a critical value for α, after which the pseudo-agent switches to a sufficiently different behaviour, according to condition (5). In this case, we can deduce that any value for d in the approximate interval $[0.8, 1.4]$ is suitable. Values < 0.8 will not steer the pseudo-agent towards a trajectory different to the optimal agent, whereas values > 1.4 would falsely rule out policies which we can consider as being heuristically different.

Figs. 6a and 6b show the ground truth for Q^{π^*} and Q^{π_1} respectively, in the ego trajectory feature space, represented as a 2D-tuple of ego vehicle speed, along with the corresponding time step of the episode $(t, \dot{x}_{ego})$. In our use-case, this representation is sufficient to see the difference between varying ego behaviours. We can see in Fig. 6a that with the unmodified reward, the optimal agent π^* prefers trajectories having higher speeds, as they correspond to a shorter episode duration which is optimal in the sense of the original reward structure. In Fig. 6b, adding an extra pseudo-reward changes the optimization landscape, and tends to steer the pseudo-agent towards areas of lower ego speeds. In all figures, we sampled 10 trajectories from different instances of both π^* and π_1, and plotted the mean Q-value for each $(t, \dot{x}_{ego})$ pair.

Fig. 6c shows the change in the expected returns in the case where there is an increase in uncertainty around the first target vehicle's position. In our experiments, we modelled a local increase in sensor uncertainty by increasing the effective collision radius of the first target vehicle by 50%. This modification leads to a sharp drop in the performance of



(a) (b) (c)

Fig. 6: Q-functions evaluated at different areas of feature space: (a) Q^{π^*} unaffected by the pseudo-reward, favors higher-speed trajectories. (b) Q^{π_1} using pseudo-rewards ($\alpha = 1$) favors lower-speed trajectories. (c) Q^{π^*} with increased uncertainty on target position, higher-speed trajectories result in a collision with first target vehicle.

π^*, whereas the state-subspace exploited by π_1 remains safe and unaffected. We can see that the new optimal policy in the case of Fig. 6c, is also reflected in Fig. 6b after adding the pseudo-reward term. This will allow us to use π_1 as a valid fallback strategy during execution, if ever there is a change in the environment that would not have been accounted for during the initial training phase.

VII. CONCLUSION

In this paper, we have introduced a new objective in an RL learning pipeline: keeping track of, and learning, sub-optimal policies encountered during the initial training phase. We have shown that through an intuitive modification of the reward model, that we are able to consistently learn these sub-optimal policies in the case of a driving scenario.

The context of this work is intended for methods to be applied to model-free problem statements. In the case where the model, even a partial model, or estimation thereof is available to the agent, we gain access to more powerful and data-efficient methods for dealing with introduction of local uncertainties to the MDP.

It is our goal to later combine this work with a hierarchical controller, to be able to quickly switch from optimal to fallback policies in the case of unexpected environment change during the execution phase. This will allow an autonomous vehicle agent to make use of its fallback strategies learned during training, according to its perception of the environment, much like a human would.

REFERENCES

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[2] S. Depeweg, J.-M. Hernandez-Lobato, F. Doshi-Velez, and S. Udluft, "Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, Jul 2018, pp. 1184–1193.

[3] S. Zang, M. Ding, D. Smith, P. Tyler, T. Rakotoarivelo, and M. A. Kaafar, "The impact of adverse weather conditions on autonomous vehicles: Examining how rain, snow, fog, and hail affect the performance of a self-driving car," *IEEE Vehicular Technology Magazine*, vol. PP, pp. 1–1, 03 2019.

[4] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Safe reinforcement learning with scene decomposition for navigating complex urban environments," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 1469–1476.

[5] W. R. Clements, B. Robaglia, B. V. Delft, R. B. Slaoui, and S. Toth, "Estimating risk and uncertainty in deep reinforcement learning," *arXiv Preprint*, 2019.

[6] C.-J. Hoel, K. Wolff, and L. Laine, "Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 1563–1569.

[7] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the Atari human benchmark," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, Jul 2020, pp. 507–517.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[9] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," 2019. [Online]. Available: http://arxiv.org/abs/1912.06680

[10] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *CoRR*, vol. abs/1710.02298, 2017. [Online]. Available: http://arxiv.org/abs/1710.02298

[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proceedings of the International Conference on Learning Representations*, 2016.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[13] M. J. Kochenderfer, C. Amato, G. Chowdhary, J. P. How, H. J. D. Reynolds, J. R. Thornton, P. A. Torres-Carrasquillo, N. K. Üre, and J. Vian, *Decision Making Under Uncertainty: Theory and Application*, 1st ed. The MIT Press, 2015.

[14] C.-J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 5, pp. 294–305, 2020.

[15] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez Liebana, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 1–43, 2012.

[16] R. McAllister and C. E. Rasmussen, "Data-efficient reinforcement learning in continuous state-action gaussian-pomdps," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[17] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *AAAI Fall Symposia*, 2015.

[18] P. Zhu, X. Li, and P. Poupart, "On improving deep

reinforcement learning for pomdps," 2017. [Online]. Available: http://arxiv.org/abs/1704.07978

[19] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt, and C. Blundell, "Never give up: Learning directed exploration strategies," in *International Conference on Learning Representations*, 2020.

[20] H. Eriksson and C. Dimitrakakis, "Epistemic risk-sensitive reinforcement learning," in *Proceedings of European Sympiosium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2020.

[21] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[22] L. Kirsch, S. van Steenkiste, and J. Schmidhuber, "Improving generalization in meta reinforcement learning using learned objectives," *arXiv preprint arXiv:1910.04098*, 2019.

[23] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, 2015, pp. 1312—-1320.

[24] M. C. Machado, M. G. Bellemare, and M. Bowling, "A Laplacian framework for option discovery in reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, pp. 2295–2304.

[25] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[26] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.