Reinforcement Learning for Autonomous Maneuvering in Highway Scenarios

Branka Mirchevska* Manuel Blum§ Lawrence Louis*
Joschka Boedecker§ Moritz Werling*

Abstract: In this work, we present a **Reinforcement Learning** (RL) based approach for autonomous driving in highway scenarios, including interaction with other vehicles. The method used is Fitted Q-iteration [1] with Extremely Randomized Trees [2] as a function approximator.
We demonstrate that Reinforcement Learning based concepts can be successfully applied and can be used to teach a RL agent to drive autonomously in an intelligent way, by following traffic rules and ensuring safety. By combining RL with the already established control concepts, we managed to build an agent that achieved promising results in the realistic simulated environment.

Keywords: Autonomous driving, Reinforcement Learning, Fitted Q-iteration, Extremely Randomized Trees

1 Introduction and Motivation

Autonomous driving in the past years, is the leading focus point in the automotive research field. Given the astonishing statistics indicating that the number of fatalities in traffic accidents in the last 10 years is 1.2 Million per year [3], it is expected to save millions of lives in the near future. Moreover, it is going to optimize traffic and significantly minimize travel times. Self-evidently, there is a strong benefit of building reliable autonomous vehicles. However, their implementation is a great challenge, both for the rule-based control and the machine learning research communities.
The **rule-based control** approaches rely mostly on hand-crafted preprogrammed implementations. Hand-crafting complex sets of rules, by incorporating expert knowledge for highly non-linear systems with large state and action spaces, is a cumbersome procedure. Since not all possible traffic scenarios can be predicted and preprogrammed, the performance under unseen circumstances may be questionable.
Machine Learning (ML) techniques on the other hand, provide the possibility for a driving agent to learn from data and improve in the future based on its gathered experience. They have been applied on autonomous driving related problems and have been

*The authors work at BMW Group, Parkring 19 85748 Garching
(branka.mirchevska, lawrence.louis, moritz.werling)@bmw.de
§The authors work at Freiburg University-Machine Learning Lab,
Georges-Koehler Allee 79 79110 Freiburg (mblum, jboedeck)@informatik.uni-freiburg.de

proven to be suitable means for solving them [4],[5], [6]. In this work, we focus on Reinforcement Learning [7], [8], [9], which allows the agent to learn the desired behavior based on feedback from the environment. This behavior can be learnt once and applied on the problem as is, or keep on adapting over time and eventually achieve a global optimum which is the ideal behavior that maximizes the reward. The goal of this work is to teach a RL agent to drive autonomously on highway scenarios. More precisely, to learn how to maintain its velocity as close as possible to the predefined desired velocity, by learning to perform the overtaking maneuver, driving smoothly and avoiding dangerous situations that would result in crashes. The performance of the RL agent was evaluated on realistic simulated environment. The results we achieved show that the RL agent learned to perform a discrete, higher-level set of actions in order to maximize its reward. These discrete actions are further converted into gas pedal, steering wheel and gear inputs by the lower-level underlying system, for the execution of the actual movement of the vehicle. The rest of the paper is structured as follows: In section 2 we give an overview of the theory behind RL and the algorithm implemented for solving the task. In section 3, we formalize the problem more thoroughly by introducing the state and action spaces, as well as the reward function. In section 4, the results achieved are discussed, and finally we conclude this work 5 and introduce prospective ideas for improvement.

2 Fitted Q-iteration with Extremely Randomized Trees

In the general RL setting, shown in figure 1 below, the agent learns by exploring the environment i.e. by taking actions and by getting a feedback signal from the environment for the taken actions.

Figure 1: RL Agent-Environment interaction

The RL method of choice for this work is the Fitted Q-iteration approach. It is a Q-learning [10] based dynamic programming approach with function approximation for the Q state-action value function. The Q state-action value indicates how good it is for the agent to perform a certain action in a certain state, in the long run. The need for a function approximation comes from the fact that we are dealing with an infinite horizon problem with continuous state space. Hence, the common lookup table approach, which assumes a table where the Q state-action value can be easily acquired for each state-

action pair, is not feasible. The algorithm 1, shown below comprises the procedure which assumes having a set of four-tuples in the form $< s_t, a_t, s_{t+1}, r_t >$ where s_t is the current state, a_t is the action taken from s_t, s_{t+1} is the state the agent moves into after performing a_t, and r_t is the received reward for performing a_t in s_t. In order to bridge over the usual lookup table approach for storing Q state-action values, we chose to rely on the regression Extremely Randomized Trees approach as a function approximator. Based on the set of four-tuples, the training batch is created where the input is of the form $< s_t, a_t >$ as in the equation on line 9 below, and the output is the approximation of the Q state-action value as shown in equation on line 10.

Algorithm 1 Fitted Q-iteration: adapted from Ernst et al. [1]

1: **Inputs:** a set of four-tuples F and a regression algorithm
2: **Initialization:**
3: Set N to 0
4: Let $\hat{Q}_N$ be a function equal to zero everywhere on $S \times A$
5: **Iterations:**
6: Repeat until stopping conditions are reached
7: $N \leftarrow N + 1$
8: Build the training set $TS = \{(i^l, o^l), l = 1, ..., \#F\}$ based on the function $\hat{Q}_{N-1}$ and on the full set of four-tuples F:
9: $$i^l = (s_t^l, a_t^l)$$
10: $$o^l = r_t^l + \gamma \max_{a \in A} \hat{Q}_{N-1}(s_{t+1}^l, a)$$
11: Use the regression algorithm to induce from TS the function $\hat{Q}_N(s, a)$

3 Problem Formalization

Below follows a description of the state representation observable by the RL agent, the set of possible actions that the RL agent can perform, as well as the cost functions used for training.

3.1 State representation

In this work, 20 features were used for state representation. State indicates what the RL agent is able to observe from the environment and act based on that information. The features are shown in figure 2 below. At each time-step, the agent is able to observe 20 features. More precisely, for the other vehicles surrounding the RL agent in the environment we consider:

- ***relative distance***, the distance from the surrounding vehicle to the RL agent,
- ***relative velocity***, the velocity of the surrounding vehicle relative to the RL agent,
- ***relative angle***, the orientation of the vehicle's vertical axis relative to the middle of the lane.

Whereas, for description of the RL agent, only the actual **velocity** and the **relative angle** explained above were taken into account.

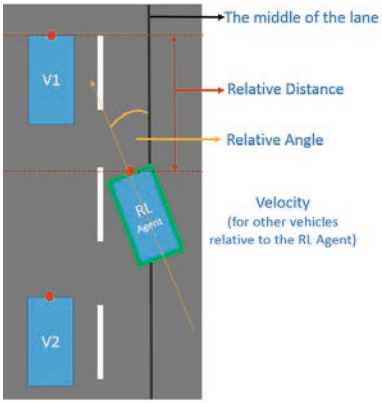

Figure 2: State representation

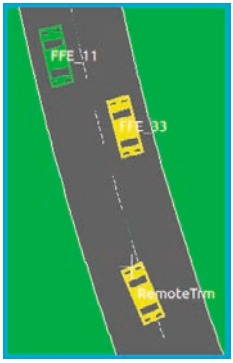

Figure 3: Simulation environment

3.2 Actions

The actions that the agent can choose from are going left, staying where it is (i.e. not going left or right) and going right. When the agent issues a lane-change action, the underlying traffic model will execute the corresponding movement by influencing the steering wheel, the gear and the gas pedal input.
Figure 3 above shows a screen-shot from the simulation environment. It shows an example time-step where 3 vehicles are depicting the RL environment. The vehicle marked with "RemoteTrm" is the RL agent which is performing an overtake maneuver at the moment.

3.3 Cost functions

Alternatively, instead of reward the notion of cost can be used. That means that instead of trying to maximize the accumulated reward over time (as was shown in the original algorithm), the agent will aim towards minimizing the accumulated cost. In this work, the term "cost" will be used.
The cost function is the core of the reinforcement learning task. It defines the task that the RL agent is expected to learn. It is a mapping from state-action pairs to a single number, indicating how good it was in that state to take that particular action in an immediate sense. Based on the cost function, the agent learns the optimal policy. The goal of the RL agent is to minimize the discounted cumulative cost it receives in a long run. The objective of our initial RL agent was to drive as fast as possible, while avoiding causing crashes by all means. In the experiments section, we present results for two distinct cost functions.

The first one is penalizing the RL agent for driving slower than its desired velocity. If the agent drives as fast as its desired velocity indicates, the cost is 0. If its velocity is lower than the desired velocity, than it is penalized proportionally: the slower it drives the higher the cost. The **initial cost function** is shown below:

if $Crash\_Happened == FALSE$ **then**
  $Cost = (RL\_Agent\_Velocity - RL\_Agent\_Desired\_Velocity)^2 * 0.0764$
else if $Crash\_Happened == TRUE$ **then**
  $Cost = 20000$
end if

When no crash occurs, the cost equals the difference between the agents velocity and its desired velocity, squared, then multiplied by a factor of 0.0764. The factor of 0.0764 was chosen empirically by experimenting with non-proportional cost function at the beginning of this work. When crash occurs, the cost equals 20,000. The value 20,000 was chosen empirically as well. It has to do with the duration of the simulations used for training. Namely, initially we considered 1,000 second episodes for training i.e. 2,000 time-steps. Taking into consideration the initial cost function, except crashing, the worst it can happen is the agent to not move i.e. to have velocity of $0m/s$. Due to the nature of the simulator that doesn't happen, i.e. the agent is always in motion, as well as the rest of the vehicles. We assumed that the agent will drive on average with approximately $14m/s$ which is 10 less than its desired velocity. We decided to penalize the agent with a cost of 10 in this case. That would mean if it drives with $10m/s$ for 2,000 time-steps it will collect total cost of 20,000. Because we want to teach the agent not to cause crashes under any circumstances, we decided to penalize the crash with a cost of 20,000, which indicates that crash should be strongly avoided, even if crash-prone behavior may increase the velocity. Afterwards the cost function was adjusted to be proportional to the desired velocity, and the factor of 0.0764 was introduced, to maintain the initial non-proportional ratio. Disregarding these specific cost numbers, it is important to mention, that it is crucial to choose the ratio between the cost for the crash and the non-crash situations in a way to ensure that the agent will learn the desired behavior. Otherwise, it might deduce for instance that it is more important to drive fast than to avoid crashes.
The second cost function, in addition to penalizing slow driving and causing crashes, penalizes slightly taking a lane change action and deviating from the action taken in the previous time-step. The **upgraded cost function** is shown below:

if $Crash\_Happened == FALSE$ **then**
  $Cost = Cost + (RL\_Agent\_Velocity - RL\_Agent\_Desired\_Velocity)^2 * 0.0764$
end if
// If the action is going left or right
if $a\_curr == -1$ **or** $a\_curr == 1$ **then**
  $Cost = Cost + 1$
end if
// If the current action is stay, but the previous one was left or right, or vice versa
if $(a\_prev == 0$ **and** $a\_curr! = 0)$ **or** $(a\_prev! = 0$ **and** $a\_curr == 0)$ **then**
  $Cost = Cost + 1$

end if
// If the current action is left, but the previous one was right, or vice versa
if ($a\_prev == 1$ **and** $a\_curr == -1$) **or** ($a\_prev == -1$ **and** $a\_curr == 1$) **then**
$Cost = Cost + 2$
end if
if $Crash\_Happened == TRUE$ **then**
$Cost = Cost + 20000$
end if

The pseudo-code above shows first, that the upgraded cost function also penalizes when the agent chooses to turn left or right. The reason behind that is the achievement of a smoother driving style. Second, it penalizes deviating from the previous action. If the previous action was "stay" and the current one is "left" or "right" or the other way around, a cost of just one is added to the cost already calculated based on the velocity. This means that it is more accepted to change from "stay" to "left" or "right" or the other way around, because that is expected to happen at some point while executing a lane-change maneuver. However, when the previous action was "left" and the current one is "right" or the other way around, then a value of 2 is added to the already calculated cost instead. This is done based on the fact that in one second, the agent shouldn't change the decision from wishing to go left to wishing to go right. (Note: an action is chosen every 500 ms).

4 Results

The RL model was trained on a 2-lane one-directional simulated highway including 34 other vehicles, controlled by the simulator, for 100 episodes. One episode comprises one simulation which lasts 200 seconds. Since, one time-step is $500ms$ long, each episode consists of 400 time-steps. Hence, the final model was trained on 40,000 training samples. The training data was collected online by letting the agent interact with the environment. The experiments were conducted on fixed 100 scenarios after every 10 episodes of training in order to track improvement. The evaluation parameters are:

- percentage of crash-free scenarios (CFS),
- average cumulated cost (ACC) and,
- average velocity (AV).

When it comes to the offline training in batch RL mode, first, we trained a model using the initial cost function. In table 1 below, can be seen that with the increase of the number of training samples (i.e. episodes), the performance increases. Moreover, the table shows that even though the velocity after training on 30 episodes doesn't significantly improve, 100% stable crash-free driving is achieved first after training for 90 episodes. The reason why in some cases the average velocity after training on less episodes is higher, than when trained on more, is that on the account of causing crashes, higher velocity is achieved.

#Episodes	CFS(%)	ACC	AV(m/s)	± Std. dev
10	99	16894.25	14.22	4057.97
20	97	12532.72	15.99	4395.10
30	95	11769.11	16.39	4433.92
40	97	11774.19	16.28	4224.98
50	99	11490.99	16.27	3576.07
60	99	10983.84	16.49	4026.28
70	100	11495.17	16.22	3601.45
80	99	10254.30	16.80	3555.54
90	100	10619.42	16.57	3409.90
100	100	11384.08	16.26	2776.10
TRM	**100**	**12275.67**	**15.76**	**3114.91**

Table 1: Initial cost function results

Additionally, on figures 4 and 5 below, plots of the progress over each 10^{th} episode, in terms of the **average cumulated cost** and the **average velocity** achieved are shown. The term *average* refers to an average over the 100 test simulations performed for the models trained on each 10^{th} episode, starting from 10 until 100 episodes. As the figures show, the average cumulated cost decreases, as the velocity increases until stability is reached after approximately 100 episodes. The green line, follows the performance of an agent which is a vehicle controlled by a hand-crafted rule-based approach. The shaded shape around the blue line depicts the ± one standard deviation from the average depicted with blue. The plots show that the RL agent after being sufficiently trained, achieves better average performance than the vehicle controlled by the simulator.

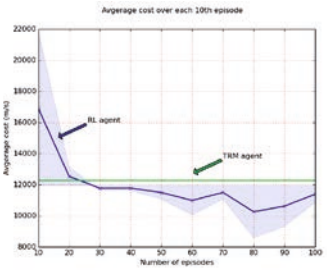

Figure 4: Initial cost function, avg. cumulated cost over each 10^{th} episode

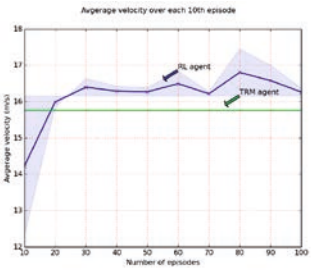

Figure 5: Initial cost function, avg. velocity over each 10^{th} episode

Furthermore, we performed the same testing procedure on the agent trained on the upgraded cost function. In figures 6 and 7, similar behavior as the one of the agent trained and tested on the initial cost function is shown. However, besides learning to avoid crashes and drive with a velocity as close as possible to its desired one, this model has learned how to drive smoothly without exhibiting unpredictable jerky movements. When we look

at the results in table 2, and compare them to the one from table 1, there is no visible difference. The reason behind that is that the cost function was only slightly adjusted to penalize mildly when the agent would choose to go left or right and when it changes the actions abruptly (for e.g. within a second). However, while observing a visualization of the two RL agents driving on the same simulation scenario, the difference is prominent. The RL agent taught only how to avoid crashes and to drive as fast as possible was not always following smooth trajectories. That behavior comes as a consequence of the fact that for the agent, the decision of going left or right, doesn't add additional cost. Additionally, the act of changing the action decision abruptly was not penalized. After the cost function was updated, the agent learnt to avoid choosing actions left or right when it was not sure that there is a need of a complete lane-change maneuver. That resulted in stability and smoothness in the driving style.

#Episodes	CFS(%)	ACC	AV(m/s)	± Std. dev
10	99	18040.35	13.86	3840.53
20	93	14329.94	15.54	4456.67
30	89	15019.16	15.60	5965.65
40	100	12346.63	16.01	3839.54
50	100	11893.21	16.23	3800.30
60	100	12104.52	16.15	3744.35
70	100	12056.58	16.17	3479.38
80	100	11752.23	16.29	3848.65
90	100	11530.79	16.35	3729.89
100	100	11156.60	16.51	3306.50
TRM	**100**	**12275.67**	**15.76**	**3114.91**

Table 2: Upgraded cost function results

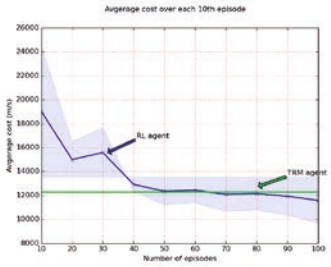

Figure 6: Upgraded cost function, avg. cumulated cost over each 10^{th} episode

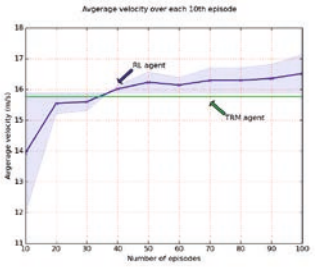

Figure 7: Upgraded cost function, avg. velocity over each 10^{th} episode

For better perspective, we conveyed a comparison of a part of a trajectory executed by the agent trained on the initial cost function, against the agent trained on the upgraded cost function. That is shown on figure 8 below. Both agents were let to drive on the same

highway scenario in order to be able to show a fair comparison. With blue, the trajectory of the agent that was trained on the upgraded cost function is shown. With red, the trajectory driven on the same path of the highway is depicted, for the agent trained on the initial cost function.

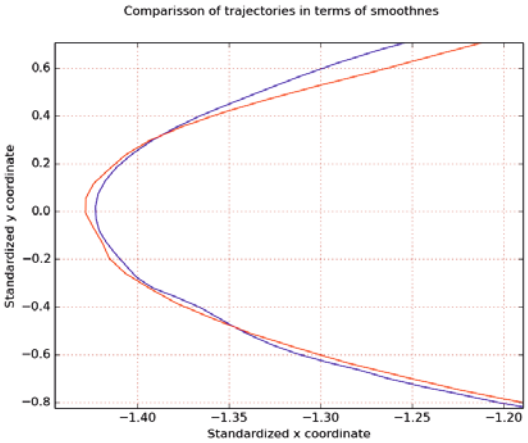

Figure 8: Comparison of part of a trajectory for the agent trained on the initial cost function (red), v.s. the agent trained on the upgraded cost function (blue)

As can be seen, shown in blue, the agent that was taught that performing turning actions is increasing the cost, and especially that the abrupt decision change in terms of actions is penalized, executes a smoother trajectory.

5 Conclusion and future work

Reinforcement Learning techniques, relying on function approximation can be successfully applied to solving realistic tasks, with infinite state space as the autonomous driving. We relied on Reinforcement Learning with function approximation to teach a RL agent how to drive efficiently and smoothly in a simulated highway environment. The model-free and offline method that we used, shows that the task can be learnt without the agent previously knowing the dynamics of the system including the reaction of other vehicles, and by learning offline, on already collected batch of training samples. However, online improvement is still possible, by allowing growing of the training set while learning. By considering a rule-based autonomous agent as a baseline, the results showed that the RL agent was able to achieve better performance in terms of average cumulated cost and average velocity.
We considered high-level decision making from the RL agent's side and low level execution

of the agent's decisions by the rule-based system. Thereby we showed that successful integration with the current control-based system is possible.
Furthermore, by additional upgrade of the cost function, different driving behaviors can be achieved which may lead to performance and safety improvement.
In order to speed up the learning, we could use expert data for training. This would provide a head start for the agent to learn from data with proven quality. Still, when applied in a growing batch mode, it will have the possibility to additionally improve, by using the already learned policy for more informed exploration.
Finally, it would be interesting to experiment with a different function approximator and compare the performance to the one of the regression trees.
All ideas for research mentioned above, as well as many more, leave space for potential improvement and optimization of the results and encourage further research in this field.

References

[1] Ernst Geurts Wehenkel. "Tree-Based Batch Mode Reinforcement Learning". In: *Journal of Machine Learning Research* (2005).

[2] Ernst Geurts Wehenkel. "Extremely randomized trees". In: *Springer Science + Business Media, Inc.* (2006).

[3] *World Health Organization*. `http://www.who.int/gho/road_safety/mortality/en/`.

[4] Dean Pomerleau. "ALVINN: An Autonomous Land Vehicle In a Neural Network". In: *Advances in Neural Information Processing Systems 1*. Ed. by D.S. Touretzky. Morgan Kaufmann, 1989.

[5] Sebastian Thrun et al. "Stanley: The Robot That Won the DARPA Grand Challenge: Research Articles". In: *J. Robot. Syst.* 23.9 (Sept. 2006).

[6] Mariusz Bojarski et al. "End to End Learning for Self-Driving Cars". In: *CoRR* abs/1604.07316 (2016).

[7] Martin A. Riedmiller, Michael Montemerlo, and Hendrik Dahlkamp. "Learning to Drive a Real Car in 20 Minutes". In: *Frontiers in the Convergence of Bioscience and Information Technologies 2007, FBIT 2007, Jeju Island, Korea, October 11-13, 2007*.

[8] Sascha Lange, Martin A. Riedmiller, and Arne Voigtländer. "Autonomous reinforcement learning on raw visual input data in a real world application". In: *The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia*.

[9] April Yu, Raphael Palefsky-Smith, and Rishi Bedi. "Deep Reinforcement Learning for Simulated Autonomous Vehicle Control". In: (2016).

[10] C. J. C. H. Watkins and P. Dayan. "Q-Learning". In: *Machine Learning* (1992).

[11] Riedmiller Gabel Lange. *Reinforcement Learning State-of-the-art*, pp. 45–71.

[12] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.