Reinforcement Learning with Hard Constraints for Autonomous Driving

Philippe Weingertner [*1] **Vaishali Kulkarni** [*1]
Ramtin Keramati (Project Mentor) [1]

Abstract

In this work we examine the problem of applying Reinforcement Learning (RL) to Decision Making for Autonomous Driving (AD). The human Driver Models of cars surrounding an AD vehicle are not known, they vary from one driver to the other and over time, and as such constitute the main source of uncertainty. A model free Reinforcement Learning agent can learn to make decisions without requiring an explicit model of the world. We investigate such a solution in this context. Enforcing safety and complying to constraints is of paramount importance for Autonomous Driving. An RL agent is maximizing an objective in expectation only. But safety can not be guaranteed in expectation. So we study how to combine Reinforcement Learning with constraints. We study how to deal with complex non differentiable constraints and explain why it may be required. We provide a new openai gym environment to study this type of problems, we benchmark state of the art RL algorithms, we investigate the influence of different states representation and how to cope with complex non differentiable constraints. We propose and experiment an algorithm to combine RL with non differentiable constraints and we provide literature references for alternative ways of approaching this type of problem.

1. Introduction

Reinforcement Learning (RL) has demonstrated its capability to learn efficient strategies on many different and complex tasks. In particular, in games like chess and go, the best human players have lost against RL algorithms (Silver et al. [14]). There is a growing traction for applying such RL algorithms to complex robotics tasks like Autonomous Driving. Nevertheless with Autonomous Driving we are dealing with additional challenges. We are in a partially observable environment where enforcing safety is of paramount importance. As a consequence, considering safety via a reward and the optimization of a statistical criteria is not sufficient. Hard Constraints have to be enforced all the time. We propose to study how the RL optimization criteria can be modified to deal with hard constraints; and more generally how an RL agent could be integrated in a Decision Making module for Autonomous Driving to provide efficient and scalable strategies while still providing safety guarantees. So we propose to address the following problem formulation:

$$\max_{\theta} \; E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi_\theta(s_t))\right]$$

$$\text{s.t. lower\_bound}(C_i(s_t, a_t)) \geq \text{Margin}_i \; \forall i \in [\![1, K]\!]$$

where the expectation corresponds to the statistical RL objective subject to a set of safety constraints.

We tackle the problem of safe control in physical systems where certain quantities have to be kept constrained. For an autonomous vehicle we must always maintain its distance from obstacles above some margin. But in fact the real state of the world is only partially observable and the Driving Models of surrounding cars are not known exactly: so we are dealing with uncertainty and our constraints are actually a set of random variables C_i which we want to lower bound. Note that in most of the references the constraints are only considered in expectation via a constraint of type $J^{\pi}_{C_i} = E_{\pi}\left[C_i(s, a)\right]$ whereas here we are interested in enforcing stronger constraints.

To handle this type of problem, 3 type of approaches are usually considered. From the most simple, conceptually, to the most complex, previous works either:

1. Propose a post RL safety check. Overriding the actions that are considered unsafe based on some predefined rules. This type of approach is used in a paper from BMW Group by Mirchevska et al. [11].

[*]Equal contribution [1]Department of Computer Science, Stanford University, California, USA. Correspondence to: Philippe Weingertner <pweinger@stanford.edu>, Vaishali Kulkarni <vaishali@stanford.edu>, Ramtin Keramati <keramati@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

2. Modify the exploration process so that only safe actions are explored. Similar to Bouton et al. [3] the idea is to derive an exploration strategy that constrains the agent to choose among actions that satisfy safety criteria. Hence the search space of policies is restricted to a "safe" or safer subspace of policies.

3. Replace the RL objective $\max_\theta \; E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi_\theta(s_t))\right]$ by an objective taking into account hard constraints

$$\max_\theta \; E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi_\theta(s_t))\right]$$

$$\text{s.t. lower\_bound}\left(C_i(s_t, a_t)\right) \geq \text{Margin}_i \; \forall i \in [\![1, K]\!]$$

and study how RL algorithms should be modified to account for this new objective. In a recent paper from DeepMind from Bohez et al. [2], this type of approach is applied to a realistic, energy-optimized robotic locomotion task, using the Minitaur quadruped developed by Ghost Robotics. But the constraints were considered only in expectation.

We will try to come up with a robust MDP objective taking into account model uncertainty (related to the uncertainty of the prediction of vehicles trajectories) to derive a safer RL algorithm, and use a constraint related to the minimum Time To Collision (TTC) that is computed between the ego vehicle and the predicted trajectories of the surrounding vehicles. This minimum TTC shall be greater or equal than a threshold margin.

2. Background/Related Work

In Mirchevska et al. [11] a DQN network is used for tactical decision making in an autonomous driving pipeline but the DQN algorithm itself is not modified to handle hard constraints and the safety is guaranteed by checking the output of the RL algorithm. Our objective here, in contrast, would be to have an RL algorithm that is directly dealing with hard constraints to avoid frequent and sub-optimal actions masking. A review of the different safe RL techniques has been done in García and Fernández [7]. Some techniques mainly deal with soft constraints by either reshaping the reward or trying to minimize the variance related to the risk of making unsafe decisions, while other try to handle hard constraints. Garcia et al. have analyzed and categorized safe RL techniques in two families of approaches: one consists in modifying the exploration process while the other consists in modifying the optimality criterion. In Leurent et al. [9] the RL objective is replaced by a surrogate objective which captures hard constraints and handles model uncertainty by defining a lower bound of the expectation objective. In Achiam et al. [1] constrained policy optimization is solved with a modified trust-region policy gradient. The algorithm's update rule projects the policy to a safe feasibility set in each iteration. But the policy is kept within constraints only in expectation. In Dalal et al. [5] they directly add to the policy a safety layer that analytically solves an action correction formulation per each state. This safety layer is learned beforehand but is approximated by a first order linear approximator. In Tessler, Mankowitz, and Mannor [16] and in Bohez et al. [2] the proposed approaches are completely in line with our objective here: modifying the RL objective such that it deals directly with hard constraints. But there is no closed form solution for such a problem and a Lagrangian relaxation technique is used for solving the constrained optimization problem. Given a Constrained Markov Decision Process (CMDP), the unconstrained problem is transformed to $\min_{\lambda \geq 0} \; \max_\theta \; L(\lambda, \theta) = \min_{\lambda \geq 0} \; \max_\theta \; [J_R^{\pi_\theta} - \lambda(J_C^{\pi_\theta} - \alpha)]$ where L is the Lagrangian and λ the Lagrange multiplier (a penalty coefficient). We propose to study how such techniques could be applied to the Decision Making process of an Autonomous Driving pipeline and we will benchmark different RL algorithms, modified to cope with hard constraints, in an Anti Collision Tests setting.

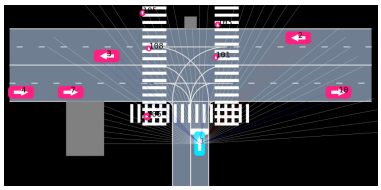
3. Approach

We consider the problem of decision making for an autonomous car. The Autonomous Driving pipeline consists of 3 main parts:

1. Perception: sensors, localization and sensors fusion provide a world model that will be used for scene understanding and decision making.
2. Planning: based on the above world model, decisions are taken and a trajectory is planned.
3. Control: actuators command are sent to control the vehicle.

We focus on the Planning module. This module is typically further decomposed into sub-modules in charge of:

1. Prediction: predicting the possible trajectories and intents of other drivers.
2. Decision Making: the decisions are typically abstract and higher level like change lane but can also be lower level like change longitudinal or lateral acceleration.
3. Motion Planning: based an above higher level decision, a trajectory will be planned.



The main challenge we would like to handle here is how to make good decisions in a situation for which we have:

- No established driving model for other vehicles. Every driver is different and may behave differently. So by construction we are in a model free setting. We do not know the driving model of other drivers. We can try to predict or estimate such a model, but there is no ground truth. And the way someone is driving may change suddenly. This is the main difficulty: we are faced with a **model free** setting, with **uncertain** and **non stationary behaviors**.
- **Safety requirements**: we have to make safe decisions to avoid collisions and not just 99% of the time. Nevertheless there are 2 type of collisions, responsible and non-responsible ones. There will never be a guarantee for 0% collisions, but we have to avoid responsible collisions and in the remaining cases, we have to make decisions that diminish the consequences of a collision.

This problem is of paramount importance and very challenging. The model free setting, where we do not know the driving model of others, is a good match for a RL model free formulation of the problem. But the safety requirements are handled with rules we want to enforce and check. With a RL solution we are defining an objective that is optimized in expectation. It can be very good in expectation. But if it has variance, we may occasionally fail the safety requirements. So in terms of problem formulation we would like to add hard constraints to an objective in expectation.

We want to avoid collisions. But this is a very late signal. We prefer to use a signal, that we can constrain and use ahead of time to prevent collisions or to diminish the consequences of collisions. A signal of interest is the Time To Collision: it acts as a proxy signal to prevent collisions. Typically the constraint we will use is that the minimum Time To Collision shall be above some margin. As we are dealing with uncertainties, we do not know the driving models of other, the TTC or minTTC is actually a Random Variable. So ultimately in the way we compute the TTC or minTTC and the way we set constraints, we have to account for uncertainties.

As a summary:

1. An AD pipeline typically predicts the Driver Model or trajectories of others. Let's assume it tries to best match an IDM or MOBIL driver model for every surrounding car. The IDM driver model enables to predict longitudinal acceleration, it depends on 5 parameters, whereas the MOBIL driver model enables to predict the lateral acceleration and depends on 3 parameters (one parameter being a level of politeness). These models are much more valuable than raw trajectories prediction, as they take into account contextual information: like relative speeds or distances. As a consequence if the context change, the prediction will change. Whereas simple models like CV (Constant Velocity) or CA (Constant Acceleration) do not take into account contextual information. So typically CV or CA models may be reliable for very short time predictions whereas IDM and MOBIL models are useful for longer time predictions or when a driver is doing more complex maneuvers like a lane change or a lane merge.
2. Once we have driving models available for surrounding cars, we can evaluate the minTTC. To simplify things, assume we are in a scene with 2 cars. The prediction module has best fitted 2 IDM models with 5 parameters each. And estimated the range of uncertainties for every parameter. Based on this we can define a conservative or robust TTC value: we sample possible trajectories and account for the min TTC obtained.

Ultimately we want to have a constraint on this minTTC: we would like to enforce that $minTTC \geq margin$

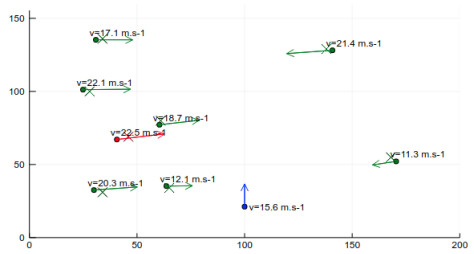
Now this is where the main challenge is when considering RL with Hard Constraints. The constraint that is of interest to us here, is very complex to compute. There is no simple differentiable graph that could be defined to express this constraint. With above IDM or MOBIL models: we would iterate over time steps for every cars, and once we know the new position of every other car, we could estimate the longitudinal acceleration of one car and so on ... Additionally, to account for uncertainties we typically have to sample IDM parameters. So the hard constraint computation is a non trivial piece of code that is not well suited for expression as a Tensorflow Graph. Most of the papers dealing with RL and hard constraints assume the constraint is expressed as a mathematical expression that is differentiable. So all the methods dealing with Lagrangian formulation or Lyapunov stability analysis are not applicable to our case. There tend to be lots of mathematical derivations in most of the papers, but ultimately these methods are applicable in restricted cases and usually tested on relatively toy constraints; like a torque constraint. In some cases, when the constraint is not differentiable, a differentiable model is learned, to best match the original constraint. The problem we are considering here, in its most generic formulation, is how to handle complex, non differentiable constraints, in an RL setting.

Another key point to consider, is that we have to handle unsafe states, being in an unsafe state (collision risk with

e.g. minTTC¡=10), identifying it as unsafe and trying to move as fast as we can to safer states. In Achiam et al. [1] a safe start region is considered and the agents tries to remain in safe regions while learning new parameters for the Policy Network. So our setting and objectives are different as well: we evaluate a safety cost and if it does not match our constraint, we try to improve to match it and then keep it above the hard constraint. So we try to deal with a recovery phase if the hard constraint is not matched.

4. Experimental Setup

We developed an openai gym-act test setup for experimentation. It is a test scene where a car, which we refer as the ego-car, has to drive from point A to point B as fast as possible while other cars are crossing its path. Other cars may be instantiated randomly at different positions, with different speeds. They may use different driver models: CV, Basic, IDM. It is a model free setting. We are trying to learn an agent that can drive efficiently (as fast as possible from point A to point B with realistic bounded accelerations $a \in [-2; 2] ms^{-2}$) and safely (while minimizing the percentage of collisions).



An animated example is provided here: ACT

The problem is setup as a Markov Decision Process (MDP) with:

- **States**: different variants will be tested, but as a starting point we consider $\{(x, y, v_x, v_y)_{ego}, (x, y, v_x, v_y)_{obj_{1..n}}\}$
- **Actions**: continuous longitudinal accelerations $a_{longi} \in [-2ms^{-2}; +2ms^{-2}]$
- **Rewards**: $-1$for every timestep, -1000 for a terminal collision state, $+1000$ for terminal target state
- Discount factor: 0.99. I could be 1 as we are dealing with a finite time horizon task or lower to reflect decreasing prediction accuracy over time.
- **Model-Free setting**: the ego-vehicle does not know the driver model of other cars, it could be CV, Basic or IDM.

The openai gym package developed supports all the standard openai gym API (step, reset, render, etc) and custom ones have been added to experiment with safety constraints and penalties.

5. Experimental Results

The source code of all developments and experiments can be found here: CS234 Project.

The source code of the developed openai gym agent can be retrieved and installed from here: gym-act

The source code of the modified vanilla policy gradient (out of CS234 programming assignment 3) with backtracking line search algorithm is available here: PG Backtrack

The implementations used to benchmark PPO, TRPO, DDPG, TD3, SAC are from openai, Author Josh Achiam

5.1. Experiments on Policy Gradients algorithm performances

The main reference papers for the bench-marked algorithms are: TRPO Schulman et al. [13], PPO Schulman et al. [12], DDPG Lillicrap et al. [10], TD3 Fujimoto, Hoof, and Meger [6], SAC Haarnoja et al. [8]

We decided to focus on policy gradients algorithms as we are dealing with continuous action spaces. We start the benchmark with a Vanilla Policy Gradient. We were initially interested in Trust Region Policy Optimization. TRPO updates the policy by taking the largest step possible to improve performance while satisfying a constraint expressed in terms of KL-divergence. With VPG it is dangerous to use large step sizes, as a single bad step, may collapse the policy performance. In RL we are relying on the data collected with our policy: so a bad update may have much more severe consequences than in Supervised Learning. TRPO avoids this risk of bad update by controlling the step size in a principled way. It is computing the step size with a complex second-order method. PPO is a family of first-order methods that tackles the same problem as TRPO in a simpler way and empirically performs similar to TRPO. So far we have considered on-policy algorithms (VPG, TRPO, PPO). DDPG and TD3 are off-policy algorithms. DDPG can be thought as Q-learning for continuous action spaces: it is motivated the same way: if you know the optimal action-value function $Q^*(s, a)$ then the optimal action can be found by solving $a^*(s) = argmax_a\ Q^*(s, a)$. To deal with continuous action spaces DDPG approximates $max_a\ Q(s, a) \approx Q(s, \mu(s))$. It uses Replay Buffer, and Target Networks like DQN. It also provides a deterministic policy which is ultimately what we are interested in for Autonomous Driving. Twin Delayed DDPG (TD3) is a set of improvements on top of DDPG. DDPG tends to learn a Q-function that potentially dramatically overestimate Q-values which leads to policy breaking. TD3 learns 2 Q-functions

instead of one and uses the smaller of the 2 to form the targets. TD3 also updates the policy less frequently than the Q-function and while training adds noise to the target action to make it more robust.

Soft Actor Critic (SAC) optimizes a stochastic policy in an off-policy way like TD3 and similarly incorporates the clipped double-Q trick. A specific feature of SAC is **entropy regularization**: increasing entropy results in more exploration which can accelerate learning and can prevent the policy from converging to a bad local optimum.

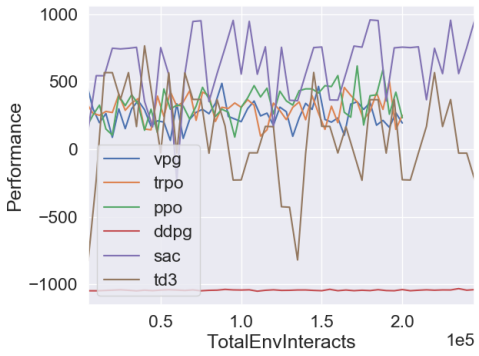
TD3 and SAC are two state of the art policy gradient algorithms which usually obtain very good results: HalfCheetah, Hopper, Walker, Ant.

In the results presented below, we are using a state space corresponding to relative coordinates. As we will later, this is the state space representation for which we obtain the best results. The training was done over 50 epochs which resulted in around $0.5e6$ environment interactions which is usually enough to classify the relative performance of different algorithms and is relatively fast in terms of experiments (based on MuJoCo benchmark reviews): it takes less than 1 minute per epoch with all algorithms. The neural networks used are typically with 1 layer of 300 neurons. To provide an intuitive interpretation of the below graph, a reward of 0 can be mapped to 50% collision rate, a reward of 500 to 25% collision rate: so we are really targeting an average reward of 1000. SAC is the only algorithm getting close to this target with a somewhat simple task setting: we are using only 2 cars crossing the way of the ego vehicle. Note that by default the scene is initialized in such a way that a random policy or a policy that systematically uses an acceleration of zero, will collide: so even in this simple setting, the network has to learn to get reasonable results. Now, considering an higher dimension problem, with 4 cars in the scene in addition to the ego vehicle, by moving to R^{16} state space, all algorithms fail to learn a policy better than a random one. The tests were run by using different driver models, CV, Basic or IDM, and the results are consistent.

The problem appears to be challenging to solve. The key challenges are:

- Model Free setting. The ego vehicle is not provided with information about the driver model of other.
- The obstacles are dynamical and they move at the same speed than the ego vehicle.
- The expected reaction time is relatively small. The agent has to find quickly a solution or it will collide and the task is over. Initially some agents were learning to drive backward, we penalized this possibility by a reward of -1000 and considered this as a terminal state.
- The state space is high dimensional $R^{n_{obj} \times 4}$
- Maybe the information contained in the state space representation is conflicting: with cars coming from the right and other cars coming from the left. We could try to specialize more the agent.
- The driving direction of the other cars is not completely orthogonal to the one of the ego vehicle. So it is actually a 2D (unknown) dynamics problem which has to be solved.
- Another point to consider is that by design we are starting in an unsafe state: typically the scene starts with a $minTTC < 10$ seconds which is our criteria for safety. The hard constraint we would like to enforce. So the problem is more challenging than starting in a safe state and trying to remain in a safe state.

So as a first conclusion, ACT openai gym module is providing a challenging task to solve for state of the art RL algorithms. More than what we expected initially.



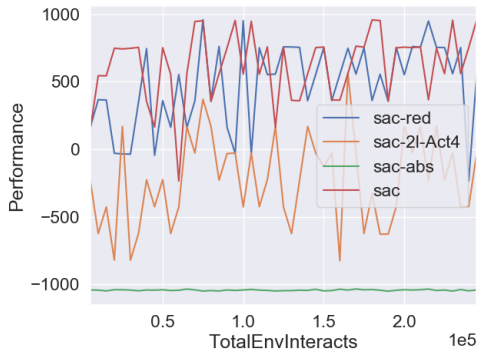
As an additional comment, in the above chart, the results for DDPG are always -1000. This result happened very rarely but sometimes it did; for other algorithms as well. Apparently depending on some random initial conditions we end up being trapped in a bad local minimum. But for DDPG type of algorithms, we consider the result with TD3: as it is an improved version of DDPG. But this also means for more consolidated results we should average the results over several tests as there is a dependency on the random initial values.

5.2. Experiments on state space representations

We experimented with different state spaces representations

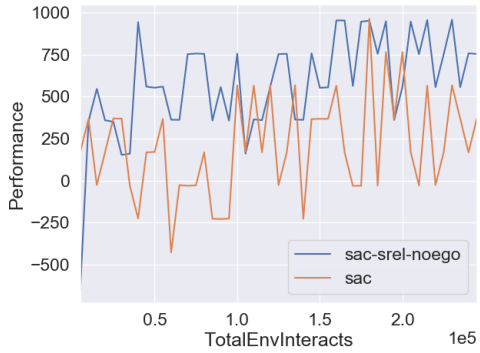
- **sac**: a R^8 relative state space representation, with 2 cars relative to ego with $[x, y, v_x, v_y]$ per car
- **sac-abs**: a R^9 absolute state space representation, with 1 ego car + 2 cars with $[x, y, v_x, v_y]$ per car
- **sac-2l-Act4**: a R^{16} relative state space representation, with 4 cars relative to ego with $[x, y, v_x, v_y]$ per car

- **sac-red**: a R^4 reduced state space representation, with 1 car with minTTC or lowest distance if TTC$= \infty$



We also experimented an image state space representation: this could be interesting as independently of the number of cars it will always be R^3 (with potentially stacked images like in the DeepMind DQN Nature paper). We run tests using $(250, 250, 3)$ state-images with cars represented by arrows matching their positions and speed vectors. The training was much slower and the reward stabilized around 250 for a task with ego+2cars; and remained at -1000 for a task with ego+10cars.

Small changes in the state space representation can have a large impact on the results. We tested SAC around two variations of the same type of representation. In both cases we use relative coordinates for the cars but in one case we have 1 extra information concerning the ego car with its state vector. In the later case, we get the worst results as can be checked on the orange curve below. Actually this information is not required and can be thought as redundant or confusing for the learning process.



Finally the best results are achieved with relative state vectors representation, but it does not scale well with the number of cars.

5.3. Experiments on non differentiable constraints

The task appears to be very challenging to solve. Here the agent has to learn everything from scratch: but even if we do not know exactly the driving behaviors of others, we have nevertheless some knowledge about how they could drive. In one way or the other, they have to comply to the law of physics relating the evolution of position to speed which is related to the evolution of acceleration. How could we break down the task so that we do not try to learn everything from scratch but leverage on some prior knowledge ? How could we integrate safety rules ?

Considering the typical Planning and Decision Making part of an AD pipeline, we usually have two distinct modules: the prediction module (trajectories and behavior predictions) and the Decision Making module. By construction the prediction module is providing some prior knowledge to the Decision Making module. In some ways the problem we have to solve probably best fit in between a pure model based approach and a pure model free approach.

How could we combine the 2 approaches:

- For safety reasons we have to define rules and to explicit models. In case of an accident, a car manufacturer will have to explain why a decision was taken, what was checked in terms of safety.
- Model Free setting: because indeed by construction we are in a Model Free setting. We do not know the driver models of others.

We are proposing to experiment with an algorithm that is interleaving a Model Free and a Model Based check and search refinement step: trying to reinforce the findings of each others.

We start with a Vanilla Policy Gradient algorithm where at each Policy Network parameters update (trying to improve over an objective in Expectation) we will use the prior knowledge provided by the Prediction module. Equipped with some driver model knowledge e.g. an estimated IDM driver model, depending on 5 parameters for which we have different level of confidence, we will estimate by how far off we are w.r.t. to our safety constraint, and perform a line search to improve over this safety criteria. As a reminder this safety criteria is potentially complex to compute and non differentiable. To account for uncertainty and to deal with the curse of dimensionality, we typically have to sample from these driver models to come up with a more robust safety evaluation.

In terms of coding, we used as a starting point the code from CS234 programming assignment 3 and modified it to incorporate a Backtracking line search method at every Policy Network parameters update. Note that in the case of these experiments, the sampling part is left for further

developments.

The Backtracking Line Search is executed over a batch of 50000 states tested over different policy networks π_θ. The batch of 50000 states provides a rather large set of experiences to test different Policy Networks.

The Policy Network parameters, $\theta \leftarrow \theta + \alpha_{bls}\nabla\pi_\theta$, are adjusted for different step sizes with $\alpha_{bls} < \alpha_{sgd}$

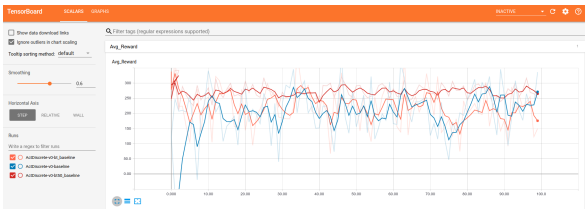
So we have a 2-stages Gradient Descent, with 2 interleaved steps:

- The first step deals with a differentiable loss function, trying to optimize $Q(s, a)$ in expectation
- While the 2nd stage is refining the step size α over a non differentiable safety penalty to search for safer policy networks.

The drawback of this method is that it slows down the training. Computing the safety penalty over 50000 states requires around 17 seconds. And we have to run this computation for every line search point. An log example is provided hereafter.

```
Time to compute penalty: 17.25 sec
old_penalty 326022.49
sgd_penalty 322133.54
Average reward: 251.22 +/- 10.82
Time to compute penalty: 17.11
old_penalty 316778.55
sgd_penalty 354335.09
Backtracking bt_penalty 348790.05
Backtracking: improvement at iter 0
Backtracking Time: 17.31 sec
Average reward: 273.23 +/- 9.44
```

The benchmark has been performed with Vanilla Policy Gradient algorithm developed in the context of the programming assignment 3. In the tensorboard plot below, we see in red the curve corresponding to the Line Search version with 50 points. During the training we see regularly the Line Search being able to find policies that decrease the safety penalty. Nevertheless the overall end result is not a huge improvement: it mainly stabilizes the results and reduces variance. In order to get this small improvement or stabilization we had to extend the line search to 50 points. So the results are somewhat disappointing.



There are a few points to take into consideration.

First of all, the problem appeared to be much more challenging to solve than anticipated. So far the main problem is not to improve safety over a reasonable baseline, but mainly to get a reasonable baseline for any scene with more than 4 cars in addition to the ego vehicle. We probably need to bring more structure to the problem by finding either a better space representation or breaking down the problem in a different way. For example instead of training an agent to deal with 10 cars we could use several agents dealing with less cars each and adopting a conservative action among the proposed actions.

Now concerning the algorithm proposed, interleaving a model Free RL learning step with a step of Model Based checks, we could experiment the other way round: here the trend is mainly given by the model free RL learning steps and we are looking for some safety improvements in a neighborhood of pre-determined points; these points corresponding to the weights of a policy network. In Bouton et al. [3] the idea was to derive an exploration strategy that constrains the agent to choose among actions that satisfy safety criteria: so the logic was the other way round.

Also even if the safety constraint or penalty is non differentiable, we could try to learn, beforehand or concurrently, a differentiable version (like a neural network) of these safety constraints or penalties. Such that we could ultimately derive a single graph leading to a single loss function that could be used to guide the learning process. This is an idea that was developed in Dalal et al. [5] where the authors directly add to the policy a safety layer that analytically solves an action correction formulation per each state. This safety layer was learned beforehand.

But in any cases a Backtracking Line Search could be considered as an additional safety check to control further the learning process. Also it may be useful as one classical problem in RL is that a single bad step can collapse the policy performance. As a conclusion it may be necessary but not sufficient.

There is probably no single solution and we would have to explore more the combination of different ideas.

6. Conclusion

We came up with a practical study setup, reviews, benchmarks, proposals and analysis related to Reinforcement Learning for Autonomous Driving and how to deal with complex non differentiable constraints. This may be a requirement for real products. But even in its simplified version, solving the task proposed by openai gym Act10-v0 environment, where an agent has to quickly find a solution to avoid collisions, appeared to be much more challenging

than anticipated. We had to switch back to a simplified version of the problem. We reduced the state space dimension to get reasonable results. Alternatively our tests with raw image representation and CNN networks have not provided good results. Additional work is required to study how we could cope with more complex scenes: this could be by decomposition of a complex scene or task into simpler ones or by the use of a more structured state space representation. Dealing with complex non differentiable constraints is a challenge. The proposed algorithm is useful: it reduces the variance of the results and provides some guarantee to avoid a single bad step during the learning process which could collapse the policy performance; this is a typical problem for Reinforcement Learning as opposed to Supervised Learning. But another approach may be to match or learn a differentiable version of the constraints so it could be more efficiently integrated into the learning process of the agent. Ultimately the two approaches could be combined.

References

[1] Joshua Achiam et al. "Constrained Policy Optimization". In: (2017). URL: http://arxiv.org/abs/1705.10528.

[2] Steven Bohez et al. *Success at any cost: value constrained model-free continuous control*. 2019. URL: https://openreview.net/forum?id=rJlJ-2CqtX.

[3] Maxime Bouton et al. "Reinforcement learning with probabilistic guarantees for autonomous driving". In: *Conference on Uncertainty in Artifical Intelligence (UAI)*. 2018. URL: https://drive.google.com/open?id=1d2tl4f6GQgH1SERveTmPAR42bMVwHZAZ.

[4] Yinlam Chow et al. "A Lyapunov-based Approach to Safe Reinforcement Learning". In: *CoRR* abs/1805.07708 (2018).

[5] Gal Dalal et al. "Safe Exploration in Continuous Action Spaces". In: (2018). URL: http://arxiv.org/abs/1801.08757.

[6] Scott Fujimoto, Herke van Hoof, and Dave Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *CoRR* abs/1802.09477 (2018).

[7] Javier García and Fernando Fernández. "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of Machine Learning Research* (2015). URL: http://jmlr.org/papers/v16/garcia15a.html.

[8] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018).

[9] Edouard Leurent et al. "Approximate Robust Control of Uncertain Dynamical Systems". In: 2018.

[10] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015).

[11] Branka Mirchevska et al. "High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning". In: 2018. DOI: 10.1109/ITSC.2018.8569448.

[12] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017).

[13] John Schulman et al. "Trust Region Policy Optimization". In: (2015).

[14] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* (2017). URL: http://dx.doi.org/10.1038/nature24270.

[15] Aviv Tamar, Shie Mannor, and Huan Xu. "Scaling Up Robust MDPs Using Function Approximation". In: *Conference on Machine Learning*. ICML'14. Beijing, China, 2014. URL: http://dl.acm.org/citation.cfm?id=3044805.3044913.

[16] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. "Reward Constrained Policy Optimization". In: abs/1805.11074 (2018). URL: http://arxiv.org/abs/1805.11074.