Reinforcement Learning with Hard Constraints for Autonomous Driving

Philippe Weingertner, Vaishali G Kulkarni
pweinger@stanford.edu, vaishali@stanford.edu

Project Mentor: Ramtin Keramati

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. [8]). 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; how algorithms like DQN could be modified to cope with such 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:

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

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

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.

2 Background/Related Work

In Mirchevska et al. [7] 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 [5]. 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. [6] 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. [4] 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 [9] 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

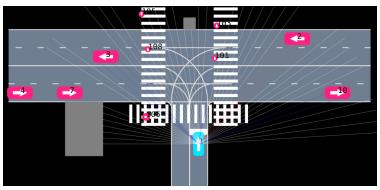
Why is it important? What are the key limitations of prior work? What are you proposing to do? What will that allow us to do now? (e.g. it addresses some prior key limitations, is computationally faster, etc etc)

We consider the problem of decision making for an autonomous car. The Autonoumous Driving pipeline consists of 3 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 like accelerate, slow down or change lane are taken and a trajectory is planned.
3. Control: the planned trajectory is followed as close as possible taking into account a detailed dynamical vehicle model and actuators command are sent to control the vehicle.

We focus on the Planning module. This module is typicaly further decomposed into 3 sub-modules:

1. A prediction module: predicting the trajectories and intents of other drivers. Typically a Driving Model can be inferred in real time to match the behavior of other drivers and to anticipate what they could do in the future.
2. A decision making module: the decisions are typically abstract and higher level like change lane but can also be low level like change longitudinal or lateral accelerations.
3. A motion planning module: based an above higher level decision, a trajectory will be planned



The difficulty in Decision Making for Autonomous Driving is due to the combination of different challenges but the main challenge we would like to adress 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 such a model, to estimate it, but in reality there is no absolute 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 **uncertainties** and **non stationnary 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, for non-responsible collisions, 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 occasionaly fail the safety requirements. So in terms of problem formulation we would like to add hard constraints to an objective in expectation.

Now let's consider further what these Safety Constraints could be. Ultimately we want to avoid collisions. But this is a very late signal and we want to use a signal, that we can constrain and use ahead of time to prevent collisions or to diminish the consequences of collisions (in case they can not be avoided). A signal of interest is the Time To Collision: it is a sort of a proxy signal for the collisions we can explicit. Once available we can define constraints on it. So 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 quick summary:

1. An AD pipeline typically predicts the Driver Model or trajectories of others. Let's assume it tries to best match an IDM/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, because as input they take into account contextual information: like relative speeds or distances. So they take into account what other cars do, to predict what the car of interest will do. And 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 horizons whereas IDM/MOBIL models are very usefull for longer time horizons or when a driver is doing something more complex like a lane change or lane merge.
2. Once we have driving models available for surronding cars, we can evaluate the minTTC. To simplify things, assume we are in a scene with 2 cars driving longitudinally. 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.

And 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 models: we would iterate over time steps for every cars, and once we know the new position of every other car we can estimate the longitudinal acceleration of one car and so on ... And on top of that to account for uncertainties we 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 usualy tested on relativelly 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 best handle complex, non differentiable constraints, in an RL setting. Anoter key point to consider, is that we have to handle unsafe states, being in an unsafe state (collision risk, minTTC<=10), identifying it as unsafe and trying to move as fast as we can to safer states. In some paper like CPO/TRPO they consider they start in a safe region and try 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, if it does not match our hard 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. And the main difficulty of the setting is that we are dealing with moving objects for which we have no ground truth models.

A DQN network will be used as a baseline (we may change later to Policy Gradients or Actor Critic. This is a topic for further refinement). We will consider 3 type of modifications. From the most simple, conceptually, to the most complex, we will:

1. Propose a post DQN safety check. While the DQN network will compute $Q(s, a_i)$ values for every possible actions, we want to exclude the actions that are unsafe, before deciding what action to take (before taking the $argmax_{a_i}\ Q(s, a_i)$). This type of approach is used in a paper from BMW Group by Mirchevska et al. [7].
2. Modify the DQN training algorithm and especially 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 $\max_\theta \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi_\theta(s_t))]$ by an objective taking into account hard constraints

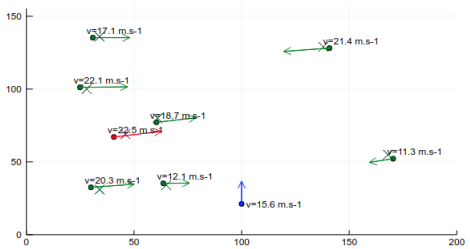
$$\max_\theta \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi_\theta(s_t))]$$
$$\text{s.t. } lower\_bound(C_i(s_t, a_t)) \geq \text{Margin}_i \ \forall i \in [\![1, K]\!]$$

   and study how RL algorithms like DQN 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 developped an openai gym, module that enable to experiment with some of the challenges.

4 Experimental Setup: Implementation of an openai gym module

We developped an openai gym test setup for experimentations. We used to have some early code in Julia used in a previous Stanford CS238 course but we migrated everything to Python and properly developped a generic easy to instal and run open ai gym module. It is a test scene where a car, which we call the ego-car, has to drive from point A to point B as fast as possible while othe 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).



The problem is setup as a MDP with:

- States: different variants will be tested, but as a starting point consider $\{(x, y, v_x, v_y)_{ego}, (x, y, v_x, v_y)_{obj_{1..n}}\}$
- Actions: longitudinal accelerations $a_{longi} \in \left[-2ms^{-2}; +2ms^{-2}\right]$
- Rewards: $-1$for every timestep,-1000 for a collision terminal state, $+1000$ when goal is reahced terminal state
- Discount factor: 0.99
- Model-Free setting: the ego-vehicle does not know the driver model of other cars, it could be CV, Basic or IDM.

An openai gym package has beeen developed and can be easilly used with any existing RL framework. It 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

5.1 Policy Gradients based algorithms benchmark: VPG, TRPO, PPO, DDPG, TD3, SAC

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, somewhat too large, 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 algorithm. 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 improvments on top of DDPG. DDPG tends to learn a Q-function that potentially dramaticaly 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.

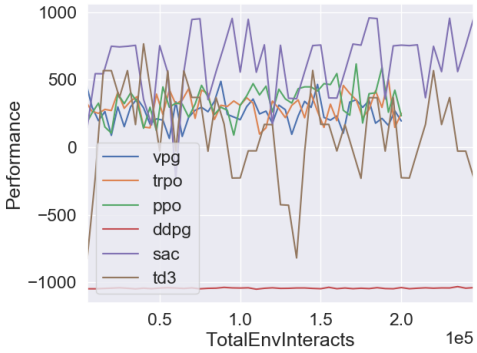
TD3 and SAC are two state of the art policy gradient algorithms which usually get the best results on robotics benchmarks: HalfCheetah, Hopper, Walker, Ant ...

In the results presented below, we are using a state space corresponding to relative coordinates: the position and speed of the cars are provided relative to the ones from the ego-vehicle. As we will see in the next section, this is the state space representation for which we got the best results. The training was done over 50 epochs which resulted in arround $0.5e6$ environment interactions which is usually enough to classify the relative performance of different algorithms and is relativelly 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 targetting for 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 $\mathbb{R}^{16}$ 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. To see more differences we think we would need to add more vehicles in the test scene. But so far the problem appears to be very challenging to solve. The key challenges are:

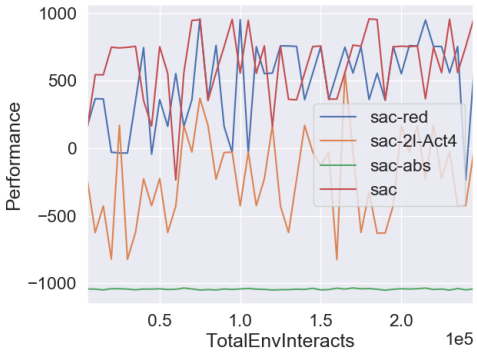
- Model Free setting. The ego vehicle is not provided with any information about the driver model of othe cars.
- The obstacles are dynamical and they move at the same speed than the ego vehicle
- The expected reaction time is relatively small. The agent has to find quickly a solution or it will collide and the task is over. Initially some agents were learning to drive backward, we penalized this possibility by a reward of -1000 and considering this as a terminal state
- The state space is big it is $\mathbb{R}^{n_{obj} \times 4}$
- Maybe the information contained in the state space representation is conflicting: with cars coming from the right and other cars coming from the left.
- 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.

In the next section we will study the impact of the state space representation.

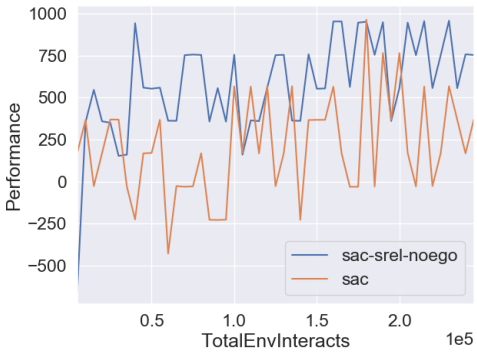


5.2 Experiments on different state space representations and how it scale

- With vectorized absolute state space representation: **sac-abs** with a $\mathbb{R}^9$ state space
  - 1 ego car + 2 cars with $[x, y, v_x, v_y]$ per car
- With vectorized relative state space representation: **sac** with a $\mathbb{R}^8$ state space
  - 2 cars realtive to ego with $[x, y, v_x, v_y]$ per car
- With vectorized reduced state space representation: **sac-red** with a $\mathbb{R}^4$ state space
  - 1 car with minTTC or lowest distance if TTC$= \infty$
- With vectorized relative state space representation: **sac-2l-Act4** with a $\mathbb{R}^{16}$ state space
  - 4 cars realtive to ego with $[x, y, v_x, v_y]$ per car



- With image state space representation: could be interesting as independently of the number of cars it will be $\mathbb{R}^3$ with e.g. 4 stacked images
  - Tests done with image erpresentation $(250, 250, 3)$; training is much slower ; reward around 250 for ego+2cars; reward at -1000 for ego+10 cars
- Now an interesting case: sac with a small change in the state space 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: cf orange curve below



5.3 Implementation and Experiments on how to handle complex non differentiable safety constraints and penalties

- Line search based on penalty constraint

Backtracking Line Search over a batch of 50000 states tested over different policy networks π_θ

Where $\theta \leftarrow \theta + \alpha_{bls} \nabla \pi_\theta$ is tested 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 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.

By definition a lot of states are unsafe. They correspond to a predicted $minTTC < 10$ seconds. we are trying to come up with a policy that pushes us towards safer states with bounded accelerations at every time step or that ensures we remain in safe states if this is already the case. But we can not move in a single step from an unsafe state to a safer state.

```
Time to compute penalty: 17.25
old_penalty 326022.49
sgd_penalty 322133.54
Average reward: −1881.22 +/− 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 bt_penalty=348790.05 sgd_penalty=354335.09
Backtracking Time: 17.31
Average reward: -1773.23 +/- 9.44
```

What experiments did you run, what did you find out. It's okay if your new idea wasn't better. But if so, it's important to have ideas about why it didn't work.

5.4 What are next steps / open questions that if you were to continue working, you would do?

- Curse of dimensinality: more efficient state space representation
- Line search is slow: learn where to search first ?

6 Conclusion

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: *Workshop on Safety Risk and Uncertainty in Reinforcement Learning, Conference on Uncertainty in Aritifical Intelligence (UAI)*. 2018. URL: https://drive.google.com/open?id=1d2tl4f6GQgH1SERveTmPAR42bMVwHZAZ.

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

[5] 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.

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

[7] 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.

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

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