Safe Online Exploration with Nonlinear Constraints

Eleanor Quint
equint@cse.unl.edu

Ian Howell
ihowell@cse.unl.edu

Garrett Wirka
gwirka@cse.unl.edu

Stephen Scott
sscott@cse.unl.edu

Hoang-Dung Tran
dtran30@unl.edu

School of Computing
University of Nebraska-Lincoln
Lincoln, NE 68588

Abstract

Safe exploration is critical to using reinforcement learning in complex, hazardous, real-world environments for which offline data aren't available. We propose a nonlinear safety layer that, unlike prior work, requires no restrictions on the policy or environment, and doesn't require offline training. We demonstrate that a nonlinear model has higher prediction accuracy than a similar linear model and that a linear safety layer fails to learn a non-conservative policy in Safety Gym environments where the nonlinear layer does not.

1 Introduction

Reinforcement learning (RL) relies on exploration to learn a safe, effective policy [1]. Exploring hazardous and complex environments can be important to employ RL effectively on safety-critical tasks, like driving autonomous vehicles [2], cooling data centers [3], and controlling cyber-physical systems [4]. These tasks are too complex to directly specify behavioural constraints, and some environments don't allow cost-effective or safe data collection. Ideally, learned safe exploration would be achievable online during training, starting from a policy initialized with little or no offline data. However, prior work in safe shielding of RL policies relies on offline training [5, 6], having a model of environment dynamics [6, 7], or requires significant assumptions about the policy or environment [5, 8]. By contrast, we would like to rapidly learn to avoid violating unknown constraints while maximizing an unknown reward function in an unknown environment.

Our proposed method uses a safety layer to learn with an unconstrained RL algorithm. Similar previous work has explored defining safety as bounding observable quantities from physical systems [5]. However, as we demonstrate with experiments in Section 4, the linearity assumption that can be made in some physical systems doesn't necessarily generalize. Like this previous work, we use the constrained Markov decision process (CMDP) framework to define constrained reinforcement learning in continuous environments [9]. This adds to the usual MDP formalism a cost function that should be minimized while simultaneously maximizing the return in each episode. In addition to minimizing the per-episode cost of the trained policy and maximizing per-episode return, we aim to promote safe exploration by minimizing the total cost accumulated over the training process [10].

Toward these goals, we propose a nonlinear safety layer, parameterized by a deep neural network that estimates cost given state and action, that censors the stochastic policy to ensure the final sampled

Workshop on Safe and Robust Control of Uncertain Systems at the 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Online.

action is safe. Efficiently sampling a safe action is made possible using two verification techniques: DeepPoly [11] and star sets [12], arranged into a novel data structure called a constellation.

This work makes the following contributions:

1. We introduce the **constellation**, a probabilistic data structure built on star sets, for piece-wise linearizing a ReLU-family deep neural network. It is independent of the choice of input data and facilitates a probabilistic analysis of the DNN.
2. We propose an algorithm for safely sampling from a stochastic policy subject to constraints defined by a deep neural network with no prior knowledge of the environment or agent, and no offline data.
3. We empirically demonstrate the relative effectiveness of a nonlinear safety layer over a similar linear model in predicting constraint violations and safe exploration.

2 Related Work

Safety Layers for Safe Exploration Dalal et al. [5] propose a safety layer that relies on the often linear nature of environment dynamics to estimate cost as a linear function of the action with coefficients from a state-fed DNN. A safe action is selected by sampling from the policy and solving a simple convex optimization to find an action that is both safe and minimizes Euclidean distance from the sampled action. It is assumed that the linear layer is pre-trained offline. We demonstrate in Section 4 that the linear safety layer doesn't function for online training in a similar environment to [5], in contrast to our proposed nonlinear model. Wagener et al. [6] use an advantage-based intervention rule to enable safe exploration with strong performance guarantees on the trained policy. However, this guarantee relies on a shielding method that requires significant prior knowledge of the environment to define both intervention criteria and a safe fallback policy. Shao et al. [7] use reachability-based trajectories to design a safety layer using a known model of environment dynamics to constrain a physical robot.

Verification Tran et al. [12] introduce star sets (hereafter **stars**) as a verification tool to represent neural network transformations of sets rather than of single inputs. Each star s is composed of two parts, an input set I_s and an affine transformation, that, when composed, give its output set. The input set is defined by a convex polytope—an intersection of half-spaces. [12] also introduces the `stepReLU` operation that linearizes the ReLU operation in one dimension by splitting the input set into subsets ≥ 0 and < 0. This allows an entire ReLU-activated DNN to be linearized exactly for some subset of the input in order to calculate its exact output.

Our sampling algorithm relies on another verification technique, DeepPoly [11] that inexpensively estimates overapproximate output bounds of a DNN given input bounds. Both DeepPoly and stars are reachability methods typically employed to guarantee that an output set doesn't intersect any unsafe subset of the output space [13], though there also exist optimization and search-based methods for verifying this type of property as well [14]. We repurpose these in our proposed algorithm.

Similar to our proposed constellation data structure, Bak [15] presents an "abstraction tree" for complete, deterministic DNN verification. Both methods arrange stars in a tree, and construct the tree in a similar manner. The key novelty in our proposal, however, is the choice to design the tree with probabilistic properties and instantiate it with a probabilistic algorithm. In other words, the constellation data structure specifically refers to a probabilistic tree used to answer probabilistic questions about a DNN, which is a fundamentally new use of reachability in deep neural networks.

3 Methods

Let stochastic policy π parameterize the multivariate Gaussian action distribution, $\mathcal{N}(\mu(s_t;\theta_\pi), \Sigma(s_t;\theta_\pi))$. Sampling from this distribution directly could result in an unsafe action. Similar to previous work [5, 6, 7], we use a safety layer to ensure that the chosen action is safe. Unlike previous work, we modify the distribution to censor unsafe actions rather than sampling a concrete action and updating it with a linear optimization. This smöl difference allows us to correctly estimate the likelihood of the action under the censored distribution and doesn't distort the underlying distribution unnecessarily.

Our nonlinear safety layer uses a ReLU-activated DNN $c_t = f(s_t, a_t; \theta_c)$ to directly estimate cost c_t given state s_t and action a_t. In this section, we first introduce the constellation data structure. Then, we use the constellation to reparameterize the policy's action distribution with respect to the DNN's structure and parameters. Finally, we describe an algorithm for censored sampling that modifies the action distribution using the safety layer DNN and efficiently samples using the constellation and DeepPoly.

Constellations As a first step towards efficiently sampling an action that is safe relative to the DNN cost estimate, we arrange stars into a probabilistic data structure termed a *constellation*. We define a constellation to be a lazily constructed directed tree where each node is a star and has children resulting from applying a DNN operation to the star. The tree root is a star with the full input set as its input polytope and the identity function as its affine transformation. To compute the effect of a DNN on the input set, we rewrite it as a sequence of affine and `stepReLU` operations. Then, each operation is applied to a star to generate its children and the next operation is applied to each of those children, and so on. Practically, in sampling, a random process selects only one child to be expanded, which leads to a DFS-like tree construction. Each `stepReLU` operation partitions the star's polytope and each dense layer modifies the star's affine transformation. The leaves of the constellation are reached when there are no operations remaining in the DNN, their input polytopes are a partition of the DNN input set, I, and their output sets are a partition of the DNN output set.

Sampling with Constellations To sample from the policy-parameterized Gaussian distribution over actions $\mathcal{N}(\mu, \Sigma)$, we reparameterize as the choice of one of the leaves $\ell \in L$ and a Gaussian truncated to that leaf's input set I_ℓ. Because the input polytopes of the leaves L are a partition of the input set I, we have the identity $\mathcal{N}(x; \mu, \Sigma) = \mathcal{N}(x; \mu, \Sigma, I_\ell)\text{Cat}(\ell; W_L)$, where the categorical probabilities W_L are equal to the likelihood that a vector sampled from $N(\mu, \Sigma)$ falls in a leaf's input polytope, I_ℓ. This reparameterization factors the input distribution in such a way that censoring it, conditional on the DNN parameters, is straightforward.

The number of leaves in the constellation is exponential in the number of neurons in the DNN n, so we reparameterize the categorical distribution over all 2^n leaves as a sequence of n Bernoulli distributions. Let $o_0, \ldots, o_{i-1}$ be the outcome of i sequential Bernoulli trials. Each Bernoulli outcome along the sequence determines whether the path from the root to star s should proceed with the upper or lower subset of a `stepReLU` operation, $s_{>0}$ and $s_{\leq 0}$ respectively. Thus, the sequence of Bernoulli outcomes $o_0, \ldots, o_i$ uniquely determines a star s in the constellation, and a sequence of length n determines a leaf. The probability each of the children $s_{>0}$ and $s_{\leq 0}$ is equal to the probability that a sample from truncated Gaussian $\mathcal{N}(\mu, \Sigma, I_s)$ falls into that child's input set. Rewriting, we get

$$\mathcal{N}(x; \mu, \Sigma) = \mathcal{N}(x; \mu, \Sigma, I_\ell)P(\ell), \text{ where } P(\ell) = \prod_{i=0}^{n} \text{Bernoulli}(o_i | o_0, \ldots, o_{i-1})$$

which admits an efficient procedure for censored sampling. The interested reader may refer to Section A in the appendices for more information.

Censored Sampling with Constellations Input sets represented in the safety layer's constellation can be labelled as safe or unsafe using the bounds of its reachable output set, which can be rapidly overapproximated by DeepPoly. In our application, this is a range of reachable cost values $[c_l, c_u]$, that can be checked against a pre-set safety upper limit C. Starting at the root and at each branch encountered in walking to a leaf through the constellation, we check the reachable set of each of the child stars and, if $c_l > C$, we zero the probability of selecting that child, which modifies the action distribution. If $c_u < C$, then the sampling procedure can stop early and the star's entire input polytope can be sampled from with a truncated Gaussian. If a leaf is reached before finding a totally safe star, i.e., a leaf with input set that contains both safe and unsafe actions, it can be efficiently split into a totally safe and a totally unsafe subset by intersecting the input set with a single hyperplane. Then, sampling can proceed using the safe subset. This sampling procedure is summarized in Algorithm 1 in the appendices. We can guarantee that this sampling procedure will halt in bounded time and is complete in the sense that it will find a safe action if there is one or, alternatively, correctly report that there is no safe action. In the case that no safe action exists some alternate strategy must be defined, typically a fallback policy or terminating the episode.

Table 1: Comparing error rates between the linear and nonlinear models on an unsafe PPO policy.

	Linear		Nonlinear		Relative Change	
Robot	TPR	TNR	TPR	TNR	TPR	TNR
`point`	73.71%	74.47%	**75.85%**	**79.21%**	2.90%	6.37%
`car`	68.88%	73.91%	**70.19%**	**78.51%**	1.89%	6.22%
`doggo`	73.77%	71.63%	**79.78%**	**74.95%**	8.14%	4.63%

Time Complexity The main concession that the proposed censored sampling algorithm makes relative to the linear safety layer approach [5] is speed. The proposed algorithm is a Las Vegas algorithm, i.e., it is guaranteed to be complete, but requires a random, bounded amount of time to run. In the worst case, the upper bound on runtime is:

$$\mathcal{O}(2^n \cdot \texttt{DeepPoly} \; + \; 2^{n-1} \cdot \texttt{SelectChild}),$$

where n is the number of neurons in the DNN, `DeepPoly` is the time required to run the DeepPoly algorithm on a suffix of the DNN, and `SelectChild` is the time required to estimate the parameter of, and sample from, a Bernoulli weighted to the relative likelihood of the children.

DeepPoly is designed to be a very fast, approximate reachability algorithm that runs in linear time with respect to the number of DNN layers, and the `SelectChild` subroutine requires estimation of the CDF of the truncated Gaussian corresponding to each child polytope. This can be accomplished in linear time with respect to the number of DNN neurons. In practice, however, we find that much less work needs to be done than the pessimistic bound, both due to the branch-and-bound DFS search of the constellation and the following simple modification to the algorithm. At each iteration, the current star's input set is sampled and the sample is run forward through the DNN. If the output is safe, then the sample can be returned and the algorithm terminated early.

In many realtime settings, a variable runtime can be disqualifying. To address this, the sampling procedure can be converted into a Monte Carlo algorithm by adding a time limit and caching the intermediate unsafe samples. Then, if the time limit is reached before the algorithm has completed, the unsafe sample that is estimated to be the least unsafe can be returned as the algorithm output. This is the scheme we employ in the experiments in Section 4, with a per-action limit of 200 milliseconds.

4 Experiments

For the sake of comparison to previous work, we restrict our experiments to Safety Gym environments with only the `Point` robot. `Point` is a simple robot constrained to the 2D plane, with one actuator for turning and another for moving forward or backward [10], and resembles the spaceship robot in experiments by Dalal et al. [5]. In the `Goal` environments a robot must navigate to a goal while avoiding hazard zones using a lidar-like sensor, in the `Button` environments a robot must navigate both still and moving hazards to reach touch a series of buttons [10]. The number appended to the environment indicates the relative challenge of navigating the hazards present. We use an efficient minimax tilting method to compute the CDF of and sample from truncated Gaussian distributions [16] and proximal policy optimization (PPO) as our underlying training algorithm [17].

Our first experiments are designed to assess the modelling gap between the linear model introduced in the safety layer of Dalal et al. [5] and the nonlinear model of our safety layer. In this implementation, the only required modification to the linear safety layer is to make its cost prediction absolute rather than an additive update to a base value because the Safety Gym environments cost values per time step are binary rather than real valued. The nonlinear safety layer uses a neural network with identical architecture to the linear version, except that it outputs the predicted cost value directly. The true positive and true negative rates of each approach are recorded in Table 1. Generally, false negatives have a much higher risk than false positives, and so we emphasize the increase in the true positive rate, though an increase in the true negative rate implies that the nonlinear model has the potential to be less conservative.

Our second set of experiments directly tests the effect of censoring the policy with each of the linear and nonlinear safety layers in an online setting with no pre-training. Results from the `Goal1`, `Goal2`, and `Button1` environments are presented in Figure 1. The episodic returns achieved by the nonlinear method were very similar to the unconstrained agent in the `Goal1` and `Button1` environments. Although the episodic cost is difficult to interpret because large standard deviations

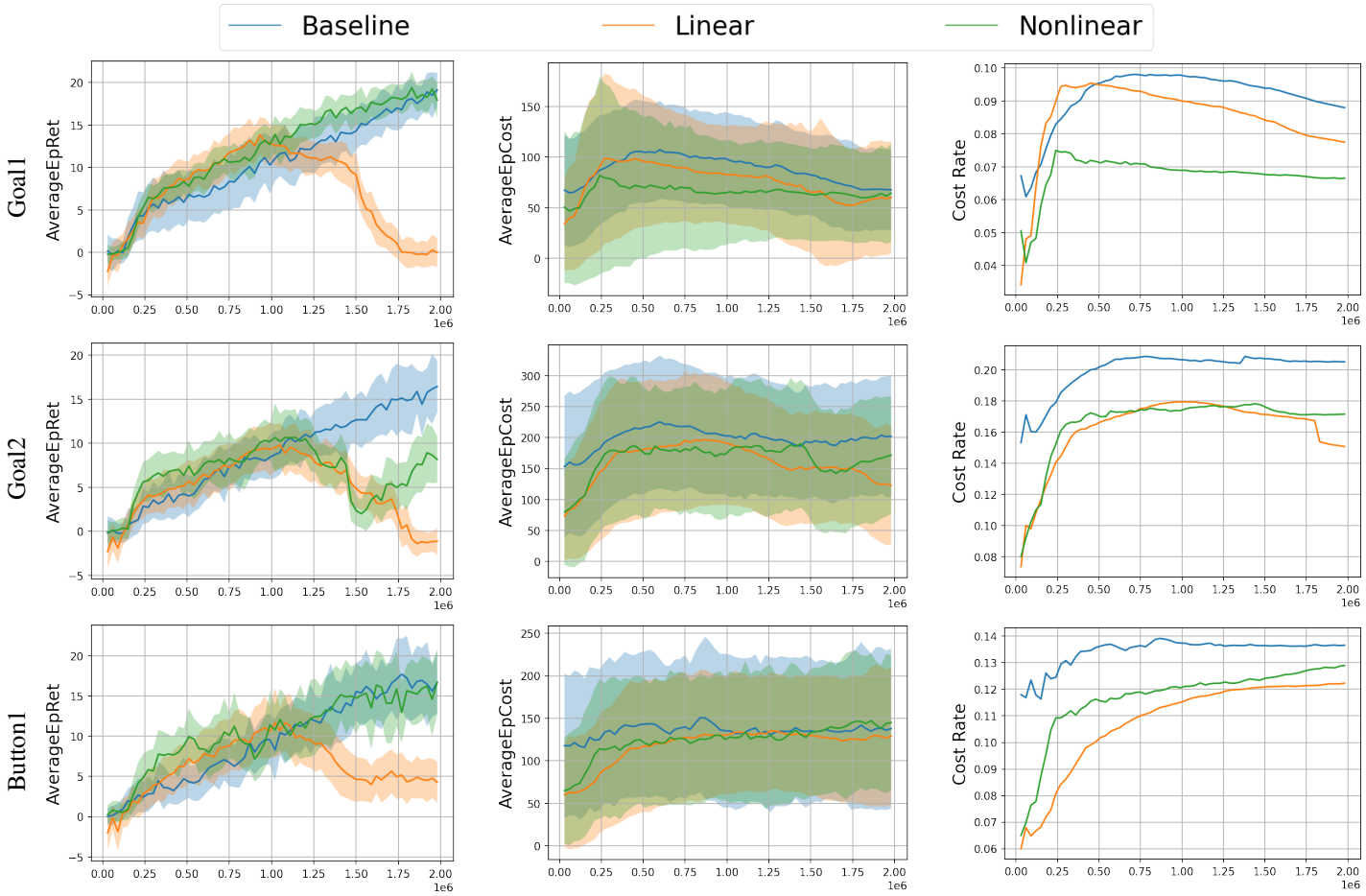

Figure 1: Performance profiles comparing the effect of linear and nonlinear safety layers on training to a baseline PPO model without a safety layer.

make the difference between the mean values seem small, the cost rate, is more interpretable. Cumulatively, the nonlinear method improves the cost rate by about 10% or more over the baseline.

We find that the behavior of the linear safety layer is very different in that the learned policy tends to collapse to be overly conservative part way through training, occasionally to the point of constraining the policy from taking any useful actions. In the `Goal2` environment the performance of the nonlinear model began to collapse, though the agent recovered and finished training with significantly higher returns than the linear method. We hypothesize that these collapses are due to the safety layer choosing very low-likelihood actions in the course of training, leading to instability in the policy gradient updates. For the linear agent, we believe selection of low likelihood actions resulted from difficulty in linearizing the constraint function and, thus, overly conservative updates to the action. We have also observed similar behavior in the nonlinear agent, in which it's due to numerical instability in our implementation, rather than any fundamental feature of the proposed method. Another consequence of this collapse is that the linear method has much lower cost rate than the unconstrained baseline. This may be more due to inaction, as episodic cost is correlated with episodic return. Performance in these metrics have been found to trade off with one other meaningfully in previous work [10].

5 Discussion

We've introduced a method for efficiently sampling from a DNN-parameterized safety layer. In experiments, we find that it can learn a non-conservative policy where a linear safety layer fails. More hyperparameter tuning or combination with another safety method will be required as part of future work to more significantly reduce constraint violations. Further, experiments against other

methods [6, 7], with varying degrees of pre-training, and in more environments will clarify the trade-offs of the proposed technique.

Acknowledgments and Disclosure of Funding

This work was not directly funded, but was supported peripherally by the University of Nebraska-Lincoln.

References

[1] Georg Ostrovski, Pablo Samuel Castro, and Will Dabney. The difficulty of passive learning in deep reinforcement learning. Advances in Neural Information Processing Systems, 34, 2021.

[2] Sina Mohseni, Mandar Pitale, Vasu Singh, and Zhangyang Wang. Practical solutions for machine learning safety in autonomous vehicles. In Huáscar Espinoza, José Hernández-Orallo, Xin Cynthia Chen, Seán S. ÓhÉigeartaigh, Xiaowei Huang, Mauricio Castillo-Effen, Richard Mallah, and John McDermid, editors, Proceedings of the Workshop on Artificial Intelligence Safety, co-located with 34th AAAI Conference on Artificial Intelligence, SafeAI@AAAI 2020, New York City, NY, USA, February 7, 2020, volume 2560 of CEUR Workshop Proceedings, pages 162–169. CEUR-WS.org, 2020.

[3] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018.

[4] Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved problems in ML safety. 2021.

[5] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerík, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. CoRR, abs/1801.08757, 2018.

[6] Nolan Wagener, Byron Boots, and Ching-An Cheng. Safe reinforcement learning using advantage-based intervention. In Marina Meila and Tong Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, volume 139 of Proceedings of Machine Learning Research, pages 10630–10640. PMLR, 2021.

[7] Yifei Simon Shao, Chao Chen, Shreyas Kousik, and Ram Vasudevan. Reachability-based trajectory safeguard (rts): A safe and fast reinforcement learning safety layer for continuous control. IEEE Robotics and Automation Letters, 6(2):3663–3670, 2021.

[8] Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 908–918, 2017.

[9] Eitan Altman. Constrained Markov decision processes, volume 7. CRC Press, 1999.

[10] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. arXiv preprint arXiv:1910.01708, 7, 2019.

[11] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages, 3(POPL):1–30, 2019.

[12] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-based reachability analysis of deep neural networks. In International Symposium on Formal Methods, pages 670–686. Springer, 2019.

[13] Sayan Mitra. Verifying Cyber-Physical Systems: A Path to Safe Autonomy. MIT Press.

[14] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher A. Strong, Clark W. Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. Found. Trends Optim., 4(3-4):244–404, 2021.

[15] Stanley Bak. Execution-guided overapproximation (EGO) for improving scalability of neural network verification. In International Workshop on Verification of Neural Networks, 2020.

[16] Zdravko I Botev. The normal law under linear restrictions: simulation and estimation via minimax tilting. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79(1):125–148, 2017.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. CoRR, abs/1707.06347, 2017.