7-2023

Safe MDP planning by learning temporal patterns of undesirable trajectories and averting negative side effects

Siow Meng LOW

Akshat KUMAR
Singapore Management University, akshatkumar@smu.edu.sg

Scott SANNER

Safe MDP Planning by Learning Temporal Patterns of Undesirable Trajectories and Averting Negative Side Effects

Siow Meng Low[1], **Akshat Kumar**[1], **Scott Sanner**[2]
[1] Singapore Management University
[2] University of Toronto
smlow.2020@phdcs.smu.edu.sg, akshatkumar@smu.edu.sg, ssanner@mie.utoronto.ca

Abstract

In safe MDP planning, a cost function based on the current state and action is often used to specify safety aspects. In the real world, often the state representation used may lack sufficient fidelity to specify such safety constraints. Operating based on an incomplete model can often produce unintended negative side effects (NSEs). To address these challenges, first, we associate safety signals with state-action trajectories (rather than just an immediate state-action). This makes our safety model highly general. We also assume categorical safety labels are given for different trajectories, rather than a numerical cost function, which is harder to specify by the problem designer. We then employ a supervised learning model to learn such non-Markovian safety patterns. Second, we develop a Lagrange multiplier method, which incorporates the safety model and the underlying MDP model in a single computation graph to facilitate agent learning of safe behaviors. Finally, our empirical results on a variety of discrete and continuous domains show that this approach can satisfy complex non-Markovian safety constraints while optimizing an agent's total returns, is highly scalable, and is also better than the previous best approach for Markovian NSEs.

Introduction

In several environments, the agent must avoid both potential hazards while simultaneously optimizing its total accumulated reward. The existing literature in safe planning uses a constrained MDP formulation (Altman 2021) and represents safety specifications as safety constraints that are derived from the immediately observable state and/or action (Achiam et al. 2017; Tessler, Mankowitz, and Mannor 2018; Stooke, Achiam, and Abbeel 2020; Chow et al. 2019; Dalal et al. 2018; Simão, Jansen, and Spaan 2021). However, obtaining a perfect description of the target environment becomes practically infeasible as autonomous agents are increasingly deployed in the real world (Dietterich 2017). As a result, operating on such incomplete models may produce undesirable side effects, also called *negative side effects* (NSEs), which are often discovered after agent deployment (Amodei et al. 2016; Alizadeh Alamdari et al. 2022; Krakovna et al. 2019; Saisubramanian, Kamar, and Zilberstein 2020, 2022). Therefore, addressing such NSEs has become a key challenge to increase the safety of deployed AI agents (Saisubramanian, Kamar, and Zilberstein 2022).

One popular method used to solve MDPs with safety constraints is the Lagrange multiplier method (Tessler, Mankowitz, and Mannor 2018; Stooke, Achiam, and Abbeel 2020). In this approach, the Lagrange multiplier (which acts as a penalty for constraint violation) is adapted slowly as the learning proceeds, converging to constraint-satisfying policy while also optimizing the primary reward objective. Other approaches include Lyapunov functions (Chow et al. 2019), Trust Region methods (Achiam et al. 2017), and constraint safety layer (Dalal et al. 2018) methods. Notably, all these methods model safety requirements as functions of safety cost functions which are assumed to be known and Markovian and hence functions of immediate state and action. Our work focuses on settings without such modelling assumptions, where we learn a classifier trained on trajectories labeled with different categories of safety labels for NSEs, hence modeling non-Markovian safety side effects. We then integrate this classifier with our safe planning approach.

A closely related line of work for addressing NSEs is presented in (Saisubramanian, Kamar, and Zilberstein 2020, 2022; Shah and Krasheninnikov 2019). Saisubramanian, Kamar, and Zilberstein (2022) define NSE as *undesired, unmodeled effects* due to incomplete MDP model specifications. Since not all undesirable effects can be foretold in advance, the model specification may lack sufficient fidelity to represent different types of NSEs. Their work proposed a supervised learning model to learn about NSEs through various types of human feedback data about NSEs, including human demonstration. Other research works propose different ways to infer NSEs, for instance through initial state configuration (Shah and Krasheninnikov 2019), reachability of other states (Krakovna et al. 2018) or attainable utility (Turner, Hadfield-Menell, and Tadepalli 2020) after performing an action, ability to perform future task different from the current task (Krakovna et al. 2020). Bayes reasoning was also used in (Hadfield-Menell et al. 2017) to infer the true reward specification from a number of candidate reward functions. One common theme behind all these works is that not all NSEs can be anticipated precisely at design time and they need to be dynamically learned through human feedback or inferred. Furthermore, all these works focus on Markovian NSEs, rather than a more general model

where NSEs are associated with trajectories. It is a critical gap in previous approaches, such as (Saisubramanian, Kamar, and Zilberstein 2020), as they decompose the penalty associated with an NSE into additive penalties associated with each state-action pair, which may not always be the case in a real world setting. Alternatively, one can expand the state space definition to make NSEs Markovian, however this may make the agent's primary task computationally challenging due to complex state space, and knowledge about different types of NSEs is also not known apriori.

Another line of work (Junges et al. 2016; Alshiekh et al. 2018; Jansen et al. 2020; Jothimurugan et al. 2021) has explored using logic specifications, such as temporal logic, to specify safety constraints which can be non-Markovian. In these methods, safety criteria need to be *pre-specified*. As NSEs are side effects *unmodeled* at the design stage, it is not feasible to pre-specify the NSE criteria and these methods are not suitable for NSE setting. In contrast, our method only requires trajectory labels for NSE trajectories. In addition, temporal logic methods typically assume a safety-relevant abstraction of the original MDP (i.e. safety-relevant MDP) (Alshiekh et al. 2018; Jansen et al. 2020) which requires significant efforts from domain experts to construct, as opposed to just using the NSE labels. Lastly, our method learns to optimize reward and satisfy constraint jointly, whereas temporal logic methods synthesize a safety shield in-silos (Alshiekh et al. 2018; Jansen et al. 2020). The shield either pre-specifies allowable actions or post-corrects the actions selected by the agent. This may lead to a sub-optimal strategy with reduced action space.

Our main contributions are the following. *First*, we formulate safe MDP planning problem as constrained MDP (Altman 2021). Unlike previous methods, we do not assume a Markovian safety cost function is given. We utilize a supervised learning model to learn safety characteristics of a trajectory from the NSE data. *Second*, we integrate the learned safety model and the MDP model in a single computation graph, and develop a Lagrange multiplier (Bertsekas 1999) method to optimize the policy while respecting trajectory-based safety constraints. We (a) proposed a method applicable to both Markovian and non-Markovian NSEs, and discrete and continuous domains, with much higher scalability than the previous method (Saisubramanian, Kamar, and Zilberstein 2020); (b) developed a model-free approach and show that our model-based method is significantly better. *Finally*, our empirical results on a variety of discrete and continuous domains show that our highly scalable approach can satisfy complex non-Markovian safety constraints, while optimizing agent's total returns, and outperform previous best approach for Markovian NSEs.

Problem Formulation

Sequential decision making under uncertainty is typically represented using Markov Decision Processes (MDPs) (Sutton, Barto et al. 1998). Constrained Markov Decision Processes (CMDPs) (Altman 1998) allow incorporation of constraints in the problem formulation, which for example can model safety aspects in decision making. A CMDP can be defined using tuple $(S, A, \mathcal{T}, R, \gamma, b_0, \mathcal{C}, \mathcal{D})$. We assume a general setting with continuous state and action spaces ($S \subseteq \mathbb{R}^n$, $A \subseteq \mathbb{R}^m$). The environment state transition is characterized by the function $p(s_{t+1}|s_t, a_t) = \mathcal{T}(s_t, a_t, s_{t+1})$. A reward function $R : S \times A \to \mathbb{R}$ maps a state, action to a scalar reward value; $\gamma \in [0, 1)$ is the reward discount factor, and b_0 is the initial state distribution. We assume a model-based setting where different model components are known (e.g., reward, transition function).

Constraints are represented using cost functions $c_i \in \mathcal{C}$, and cost limits $d_i \in \mathcal{D}$ on the total expected costs. Each function $c_i(s, a)$ maps a state-action tuple to a real valued cost. Often, cost functions and constraints are used to represent safety aspects in a problem domain.

Policy: We consider a probabilistic policy parameterized by a rich function approximator, such as a deep neural network. A probabilistic parameterized policy is denoted as $\pi_\theta(s, a)$ which refers to the probability of executing action a in state s; θ refers to the parameters (to-be-optimized) of the policy function approximator.

The CMDP problem: The *primary objective* of a CMDP is to maximize the expected sum of discounted rewards over trajectories $\tau = (s_0, a_0, s_1, a_1, ...)$ by following a parameterized policy π_θ. The objective is (Achiam et al. 2017):

$$J_R(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$$

where $\tau \sim \pi_\theta$ indicates that trajectories are sampled from the distribution induced by the policy π_θ, initial state distribution b_0, and the state transition function.

Similar to J_R, we can define the expected value of a cost function $c_i \in \mathcal{C}$ over the induced trajectories:

$$J_{c_i}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t c_i(s_t, a_t)] \tag{1}$$

The CMDP problem is expressed as:

$$\max_\theta J_R(\pi_\theta) \tag{2}$$

$$\text{s.t.} \quad J_{c_i}(\pi_\theta) \leq d_i \quad \forall c_i \in \mathcal{C} \tag{3}$$

where $d_i \in \mathcal{D}$ is the corresponding threshold for J_{c_i}.

There are some key implicit assumptions in such a CMDP formulation. First, it is assumed that the cost functions c_i, which for example can define the safety aspects, are known. This may not always be the case in a real world setting as obtaining the exact numerical specification of safety cost functions is challenging. Second, it is assumed that the expected safety cost in (1) is decomposed and additive per time step. As noted in section , this may not be always feasible if the model specification is incomplete, which may make safety as a function of the entire trajectory τ, rather than just the immediate state and action. To address these issues, next we discuss a formulation in which data about safety aspects is collected in the form of negative side effects (NSEs), and a safety model is learned from such a dataset.

Negative Side Effects (NSEs)

Our NSE definition is similar to Saisubramanian, Kamar, and Zilberstein (2022) where we consider episodic tasks and

each complete trajectory $\tau = \langle s_0, a_0, s_1, \ldots, a_T, s_T \rangle$ can be of arbitrary (but finite) length, terminating in state s_T.

Definition 1. *Let Ψ denote a* partition *of the space of all possible complete trajectories for the given MDP. The set Ψ can be further divided into K mutually exclusive sets of NSE* categories*: $\Psi = \psi_1 \cup \psi_2 \cup \ldots \cup \psi_K$.*

Each ψ_i defines a category of NSEs (e.g., mild, moderate, severe, or no-NSE). Without loss of generality, we assume that a trajectory only belongs to a *single category* of NSEs (e.g., the most severe form of NSE occurring in the trajectory). When a trajectory $\tau \in \psi_i$ is observed during execution, NSE i is said to have occurred. We consider *non-Markovian* NSEs as NSE severity can depend on the entire trajectory, in contrast to Markovian NSEs which depend on a single state-action pair (Saisubramanian, Kamar, and Zilberstein 2020). Non-Markovian NSEs are more general than Markovian NSEs, and can model complex NSEs without the need to expand the state representation of the MDP. Expanding the state representation to ensure Markovian NSEs may introduce new risks and require extensive data collection/evaluation, which is undesirable.

We also note several modeling benefits of our method over prior works (Saisubramanian, Kamar, and Zilberstein 2020, 2022). Although prior works introduce non-Markovian NSEs, they do not present an approach to solve it. We use powerful neural network based function approximators that can scale well to large state spaces, unlike tabular policies used in prior works, which are not scalable to large state spaces. Previous work is also limited to discrete state and action spaces, whereas our method can handle both discrete and continuous problems. In prior works, one needs to chose an arbitrary numerical penalty score for different types of NSEs, and it is unclear how to choose it, whereas in our work, we directly exploit different categories of NSEs (e.g., mild, moderate, no-NSE), which is relatively easy to specify (e.g., by elicitation from domain experts) in comparison.

Trajectory Classification with RNNs

In general, it is infeasible to exactly define each NSE partition ψ_i beforehand. As noted in (Saisubramanian, Kamar, and Zilberstein 2020, 2022), as agents are deployed in the real world, data about different NSEs can be collected. We assume that such a dataset is available that contains trajectories for different NSE categories $k = 1 : K$ (as noted in definition 1). A data entry in this set, (τ, k), implies NSE category k is associated with the state-action trajectory τ. In our empirical experiments, we optimized a policy without averting NSEs for a domain, and subsequently used the trajectory data during the learning process for NSE labeling. This approach resembles incremental learning process in practice, where an agent interacts with the environment without first being aware of the NSE. The historical trajectory data is collected and labeled with the corresponding NSE class (including no-NSE). The labeled data entries (τ, k) can then be used for classifier training.

As the trajectory τ can be of variable length for episodic tasks, we propose an RNN - Recurrent Neural Network (Rumelhart, Hinton, and Williams 1986) in classifying a trajectory τ into its respective NSE category k. We used Gated Recurrent Units (GRU) as our specific RNN architecture, which balances between accuracy and network parameter size (Cho et al. 2014). The GRU network used in our empirical experiment is depicted in Figure 1.

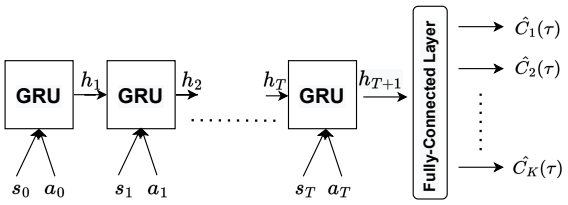

Figure 1: NSE Classifier Using Recurrent GRUs

We also note that the trajectory classification method we have presented above allows a modular separation between optimizing the primary task of the underlying MDP, and safety specifications. As more NSE data is gathered, we can update our NSE classifier with additional data, without the need to always change the underlying state space and reward function each time new NSEs are discovered. This enables modular and continual safety learning, which is critical for real world deployment of autonomous agents.

CMDP for Avoiding Non-Markovian NSEs

As noted in section , the CMDP objective is to maximize the expected sum of discounted rewards over trajectories τ. The additional NSE constraints imposed are to limit the occurrence probability of trajectories with undesirable NSEs. Note that we consider NSEs which are non-Markovian and depend on the entire trajectory. Thus, the non-Markovian NSE based CMDP problem can be expressed as:

$$\begin{aligned} &\max_{\theta} J_R(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \\ &\text{s.t. } J_{c_i}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[C_i(\tau)] \leq d_i \quad \forall i \in 1, 2, ..., K \end{aligned} \tag{4}$$

In this formulation, K refers to the number of NSE categories while $0 \leq d_i \leq 1$ specifies the tolerable proportion of trajectories having NSE of class i. The value d_i is application-dependent and typically configured by the practitioner. $C_i(\tau)$ is an indicator function to denote the presence of NSE of class i in the given trajectory τ. As NSEs are unmodeled side effects, they may not have a closed-form expression. Instead, it needs to be estimated using labelled trajectory data as noted in section . In this paper, we approximate the ground-truth $C_i(\tau)$ using RNN-based trajectory classifier output $\hat{C}_i(\tau)$, which can be interpreted as the estimated probability of NSE of class i occurring in trajectory τ. The approximate CMDP problem using trajectory classifier can be rewritten as follows:

$$\begin{aligned} &\max_{\theta} J_R(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \\ &\text{s.t. } J_{\hat{c}_i}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau)] \leq d_i \quad \forall i \in 1, 2, ..., K \end{aligned} \tag{5}$$

Proposed Method

We next present a Lagrangian relaxation based method (Bertsekas 1999), which has been popular recently to solve CMDPs. We will primarily focus on new techniques that enable us to handle trajectory based constraints $J_{\hat{C}_i}(\pi_\theta) \leq d_i$ in (5), which are complex to optimize over as they involve a RNN-based trajectory classifier (i.e., $\hat{C}_i$), and are non-additive and non-decomposable among time steps, unlike the case in standard CMDPs (see Eq. (1)).

Lagrangian Method

We apply the Lagrange multiplier method to solve the CMDP problem (5). The Lagrange multiplier method is shown to converge to the local optimum of CMDP problems and performs well empirically (Tessler, Mankowitz, and Mannor 2018; Stooke, Achiam, and Abbeel 2020). The Lagrangian dual problem of (5) is written as:

$$\min_{\boldsymbol{\lambda}\geq 0} \max_{\theta} L(\boldsymbol{\lambda}, \theta) = J_R(\pi_\theta) - \sum\nolimits_{i=1}^{K} \lambda_i [J_{\hat{C}_i}(\pi_\theta) - d_i] \tag{6}$$

We can also compute the gradient of Lagrangian function L w.r.t. different Lagrange multipliers λ_i and the policy parameters θ as:

$$-\frac{\partial L}{\partial \lambda_i} = J_{\hat{C}_i}(\pi_\theta) - d_i = \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau) - d_i] \tag{7}$$

$$\begin{aligned} \frac{\partial L}{\partial \theta} &= \nabla_\theta J_R(\pi_\theta) - \sum\nolimits_{i=1}^{K} \lambda_i \nabla_\theta J_{\hat{C}_i}(\pi_\theta) \\ &= \nabla_\theta J_R(\pi_\theta) - \sum\nolimits_{i=1}^{K} \lambda_i \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau)] \end{aligned} \tag{8}$$

Optimizing the Lagrangian problem (6) can provide a local optimum of the problem (5). The vector, $\boldsymbol{\lambda} = [\lambda_1 \ \lambda_2 \ ... \ \lambda_K] \geq \mathbf{0}$, contains the Lagrange multipliers for each of the K constraints. Each Lagrange multiplier serves as the tradeoff coefficient for the respective constraint. Learning to solve this Lagrangian problem, as shown in (Tessler, Mankowitz, and Mannor 2018), typically involves updating $\boldsymbol{\lambda}$ and θ using the gradient expressions derived in (7) and (8) (we refer the reader to Tessler, Mankowitz, and Mannor (2018) for details).

There are a number of methods estimating the first gradient term $\nabla_\theta J_R(\pi_\theta)$ in (8), including model-based Stochastic Value Gradients (Heess et al. 2015), iterative lower bound optimization (Low, Kumar, and Sanner 2022), and model-free Proximal Policy Optimization (PPO) (Schulman et al. 2017). The empirical study in this paper has been conducted using PPO as it provided better performance compared to others in terms of achieving good solution quality with higher sample efficiency. Next, we mainly focus on estimating the second gradient term $\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau)]$. This is much more challenging than the standard Lagrangian method for CMDPs (Tessler, Mankowitz, and Mannor 2018) as $\hat{C}_i$ is an RNN output, rather than an additive sum of costs as in a standard CMDP (see Eq. (1)).

Model-Free Gradient Estimation

The second term $\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau)]$ of (8) involves a gradient of an expectation taken over a random variable. The score function estimator (Fu 2006) makes use of the log derivative trick and estimates the gradient using the output score from classifier. The score function estimator is derived as:

$$\begin{aligned} &\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau)] \\ =&\nabla_\theta \Big[\mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau)] - d_i\Big] \\ =&\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau) - d_i] \\ =&\mathbb{E}_{\tau\sim\pi_\theta}[(\hat{C}_i(\tau) - d_i)\nabla_\theta \log P(\tau;\theta)] \\ =&\mathbb{E}_{\tau\sim\pi_\theta}[(\hat{C}_i(\tau) - d_i)\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t)] \end{aligned} \tag{9}$$

T in (9) is the maximum timestep encountered in a particular trajectory τ. The $-d_i$ term, introduced at the first step, serves as a baseline to reduce the variance of this estimator. This estimator provides a clever way to collect on-policy samples, estimate gradients wrt θ and adjust the policy parameters θ accordingly.

Before discussing the interpretation of the estimator in (9), we first highlight that the policy parameters are adjusted in the opposite direction of this gradient due to the negative sign in (8). Consider one particular trajectory τ from the minibatch samples collected in (9). When $\hat{C}_i(\tau) - d_i$ is positive, the policy update will move in the opposite direction of $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t)$, decreasing the likelihood of performing similar sets of actions in future. Conversely, when $\hat{C}_i(\tau) - d_i$ is negative, the incremental update will increase the likelihood of performing such actions.

Although the model-free method provides a neat way of estimating the gradient, it is rather difficult to apply in practice due to the following reasons. Firstly, score function estimators tend to have high variance (Schulman et al. 2015), in turn increasing the difficulty of moving towards the feasible constraint region in a consistent manner. Secondly, the output $\hat{C}_i(\tau)$ is an approximate value of the unknown true value $C_i(\tau)$. The classification error exacerbates the high variance issue. Lastly, the score function estimator sums up the gradients evaluated at all individual timesteps, i.e. $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t)$. This implies that the policy will be adjusted to decrease or increase (depending on the sign of $\hat{C}_i(\tau) - d_i$) the likelihood of performing the sampled actions at every single timestep. Even though the presence of an NSE is determined by inspecting the entire trajectory τ, the contributing factor could be a smaller subpart of the trajectory. In such cases, adjusting the actions at every single timestep can lead to over-correction.

In light of the difficulties in applying model-free gradient estimation methods, we now propose a model-based gradient estimation method which better exploits the differentiable function approximator learned in $\hat{C}_i(\tau)$.

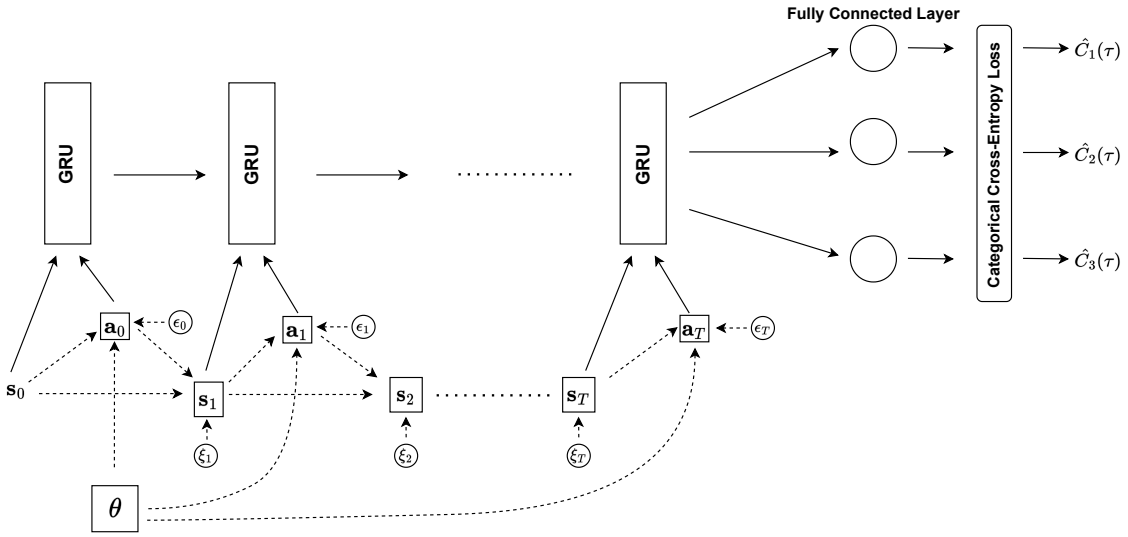

Figure 2: Stochastic Computation Graph of $\hat{C}_i(\tau)$ for trajectory τ where number of NSE classes is $K = 3$

Model-Based Gradient Estimation

The pathwise derivative (PD) estimator (Glasserman 2004; Schulman et al. 2015) has lower variance but it requires (a) $\hat{C}_i(\tau)$ be a continuous function of θ, (b) τ be a deterministic and differentiable function of θ. To fulfil these conditions, we apply the "reparameterization trick" on both the transition model $\mathcal{T}(s_t, a_t, s_{t+1})$ and the policy $\pi_\theta(s, a)$.

Figure 2 illustrates such a reparameterized model using a stochastic computation graph (Schulman et al. 2015). Figure's upper half depicts the GRU network used in classifying trajectory τ, which outputs $\hat{C}_i(\tau)$. The bottom half outlines the relationship between policy parameters θ and the state-action pairs (s_t, a_t) observed in trajectory τ. To express s_t, a_t as deterministic and differentiable function of θ, the reparameterization trick uses exogenous random samples ξ, ϵ to sample from the probability distributions $\mathcal{T}(s_t, a_t, s_{t+1})$ and $\pi_\theta(s, a)$ respectively. For location-scale probability distributions (e.g. Gaussian, Gamma), sampled state can be written (Kingma and Welling 2014) as $s_{t+1}(\theta, \xi) = \mu(s_t, a_t; \theta) + \sigma(s_t, a_t; \theta) \cdot \xi_{t+1}$ where μ, σ refer to the location and scale respectively while ξ_t is a random sample drawn from a normalized distribution with zero location and unit scale. For categorical distributions, Gumbel softmax (Jang, Gu, and Poole 2016) can be employed to rewrite sampled action $a_t(\theta, \epsilon) = one\_hot(\text{softmax}(\log \pi_\theta(s_t, a) + \epsilon_t))$ with ϵ_t drawn from Gumbel(0, 1). Sampled state can be rewritten similarly.

We call this method model-based gradient estimation as the transition model $\mathcal{T}$ is required for the reparameterization trick. The pathwise derivative estimator is written as:

$$
\begin{aligned}
&\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(\tau)] \\
=&\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\hat{C}_i(s_0, a_0, s_1, a_1, ..., s_T, a_T)] \\
=&\nabla_\theta \mathbb{E}_{\boldsymbol{\epsilon},\boldsymbol{\xi}\sim p(\cdot)}[\hat{C}_i(s_0, \theta, \boldsymbol{\epsilon}, \boldsymbol{\xi})] \qquad (10) \\
=&\mathbb{E}_{\boldsymbol{\epsilon},\boldsymbol{\xi}\sim p(\cdot)}[\nabla_\theta \hat{C}_i(s_0, \theta, \boldsymbol{\epsilon}, \boldsymbol{\xi})] \qquad (11)
\end{aligned}
$$

The reparameterization trick is performed in step (10) to rewrite $\hat{C}_i(\tau)$ as a function of the initial state s_0, policy parameters θ and the exogenous noise vectors $\boldsymbol{\epsilon}, \boldsymbol{\xi}$ (of size $T + 1$ and size T respectively) drawn from probability distribution independent of θ. Recall that classifier output is a function of trajectory $\tau = (s_0, a_0, s_1, a_1, ..., s_T, a_T)$. All intermediate s_k (except s_0) has a recursive relationship with the previous state-action pair (s_{k-1}, a_{k-1}) and can be rewritten as a function of θ and $\boldsymbol{\xi}$ using the reparamterization formulae described earlier. Similarly, a_k is dependent on s_k and θ. Since s_k can be written as a function of θ and $\boldsymbol{\xi}$, the observed a_k (sampled from probability parameterized policy $\pi_\theta(s, a)$) can also be written as a function of $\theta, \boldsymbol{\epsilon}$ and $\boldsymbol{\xi}$ in a similar manner. As the function approximator $\hat{C}_i(\tau)$ provides a continuous and differentiable function, the model-based gradient estimation method exploits this and averages the sample derivative $\nabla_\theta \hat{C}_i(s_0, \theta, \boldsymbol{\epsilon}, \boldsymbol{\xi})$ in the minibatch.

We also discuss the characteristics of this estimator. First, model-based gradient estimation utilizes the derivative $\nabla_\theta \hat{C}_i$ instead of the actual output $\hat{C}_i$. Using the derivative typically results in lower variance (Schulman et al. 2015; Heess et al. 2015), especially when the function $\hat{C}_i$ is smooth. Second, the proposed trajectory classifier uses the cross-entropy loss function, causing the gradient $\nabla_\theta \hat{C}_i$ to gradually diminish when predicted probability $\hat{C}_i$ approaches 1 or 0. The small gradient values encountered will impede the gradient update of the Lagrange Multiplier term. Regularization (e.g. Dropout) is recommended to prevent the classifier from outputting a value very close to 1 or 0 (e.g. ≥ 0.9999). In addition, we recommend a non-zero initial value for $\boldsymbol{\lambda}$ to prevent θ from quickly converging to an infeasible region where the evaluated gradient $\nabla_\theta \hat{C}_i$ is very small. Last, for the computation of $\hat{C}_i$, the trajectory classifier can have different weights for the state-action pair (s_t, a_t) at different timestep t. This permits the policy net-

	Boxpushing		Driving	
	Reward	NSE	Reward	NSE
Markov-HA-S	-30.43 ± 2.29	0.76 ± 3.40	-26.59 ± 0.46	8.24 ± 8.75
MF-Lagrange	-28.29 ± 1.92	59.00 ± 10.95	-28.10 ± 0.98	5.10 ± 7.46
MBGE	$\mathbf{-26.80 \pm 0.89}$	$\mathbf{0.30 \pm 0.72}$	$\mathbf{-25.78 \pm 1.01}$	$\mathbf{2.97 \pm 6.61}$

Table 1: Grid World Experiment Results - Boxpushing & Driving

work to target specific timesteps which greatly impacts the final score $\hat{C}_i$ and adjust the policy parameters θ using the gradients evaluated at these timesteps accordingly. This is beneficial and tackles the over-correction issue discussed in model-free gradient estimation.

Empirical Experiments

To test the effectiveness of our proposed methods[1], we present two sets of experiment results. The first set of experiments involves two different tasks in a grid-world environment where state and action spaces are both discrete. The objective of this experiment is to compare against the current state-of-the-art safe planning approach (Saisubramanian, Kamar, and Zilberstein 2022) which avoids Markovian NSEs. We show that our proposed model-based gradient estimation (MBGE) method produces better results in balancing between maximizing reward and averting NSEs. The second set of experiments evaluates the performance in performing two distinct tasks in continuous MDP planning domains. The detailed definitions of these continuous MDP planning domains can be found in (Bueno et al. 2019; Bueno 2020). Our proposed method successfully optimizes rewards while reducing the occurrence of non-Markovian NSEs below the specified thresholds.

To perform gradient-based updates, our proposed method collects a batch of trajectory samples in a learning epoch and each experiment run trains the agent for a number of learning epochs (1000 for Grid-Worlds, 5000 for continuous domain). During training, after a fixed number of learning epochs, 100 separate test trajectories (different from the training trajectories) are collected for performance evaluation. The rest of hyperparameter settings used in our experiments are provided in the supplement.

For performance reporting, we report the average and standard deviation values for 5 training runs. This is because our approach involves learning from samples collected from stochastic environments, while conventional planning approach like (Saisubramanian, Kamar, and Zilberstein 2022) does not collect trajectory samples. To have a fair estimate of the planning performance of our approach, we evaluate the average performance over the 5 runs.

Grid World Domains

Here we provide a brief description of the grid world domains used to test our approach against current state-of-the-art Markovian NSE planner which learns NSE from human-provided numerical score of a state-action pair. We call this method *Markov-HA-S* and refer the readers to (Saisubramanian, Kamar, and Zilberstein 2022) for more comprehensive description of the experiment setup. Experiments have been carried out for two separate tasks:

Boxpushing The agent is required to pick up a box at a specific location and move towards a goal position (Seuken and Zilberstein 2007). The agent will incur NSEs if it lands on certain surface types (e.g. surface with carpet, fragile surface) while pushing the box.

Autonomous Driving The autonomous agent is incentivized to move toward a goal position as quickly as position (Wray, Zilberstein, and Mouaddib 2015). However, it will incur NSEs if it tries to drive fast while nearing a pedestrian or a puddle.

Negative Side Effects For *Markov-HA-S*, NSEs are defined as Markovian NSEs which are immediately observable for a given state-action pair. To compare against their result, we adapt our classifier architecture and make it a Multi-Layer Perceptron (MLP) classifier (Goodfellow, Bengio, and Courville 2016) which classifies NSE for a given state-action pair, i.e. $\hat{C}_i(s, a)$. Similar to *Markov-HA-S*, experiments were conducted in fully avoidable NSE scenarios and thus our constraint can be adapted as $\sum_{t=0}^{T} \hat{C}_i(s_t, a_t) \leq 0$.

Classifier Training We use similar strategy as *Markov-HA-S* in training the classifier $\hat{C}_i(s_t, a_t)$, using random query data. Note that for our training data, $C_i(s_t, a_t)$ is an indicator function specifying whether NSE of class i is present while *Markov-HA-S* requires it to be a numerical score. For boxpushing domain, our classifier was trained with $\sim$1.7k samples as it was enough to achieve close to 100% classifier accuracy. We compare our method with the *Markov-HA-S* trained with larger learning budgets, i.e. {2k, 4k, 6k, 7k} samples, and we report its average performance over these training budgets. This ensures that variation in a single *Markov-HA-S* learning budget does not unfairly affect our comparisons. For driving domain, our classifier was trained with $\sim$3.5k samples (our classifier achieved close to 99% accuracy) and compared against the *Markov-HA-S* trained with learning budgets {4k, 6k, 7k}.

Results and Discussion As we are using Markovian NSEs for the Grid-World experiments, we report the NSE values using the same NSE penalty as *Markov-HA-S*: fixed numerical value 5 and 10 for mild and severe NSEs respectively. We adapt our MBGE method to reduce the Markovian NSEs. For our model-free variant, classifier's output $\hat{C}_i(s, a)$ is treated as a proxy for true costs $c(s, a)$ and the accumulated cost over the trajectory is to be minimized. The performance

[1]Code available on: https://github.com/siowmeng/Avert-NonMarkovNSE

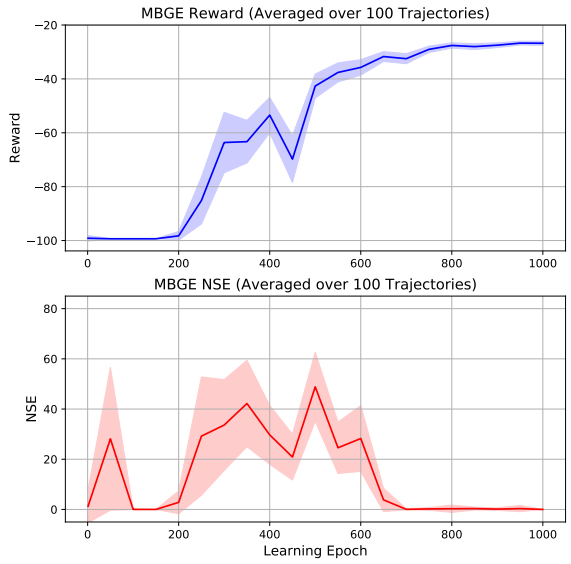

Figure 3: One MBGE Training Run for Boxpushing

(average $\pm$ one standard deviation) of this model-free variant is reported in Table 1 as *MF-Lagrange*, alongside MBGE and *Markov-HA-S*. The reported values are aggregated over five different problem instances for each task.

In both tasks, MBGE achieves the highest rewards and lowest NSEs. In boxpushing domain, MBGE's NSE is very close to zero with very low variance. In comparison, *Markov-HA-S* incurs significantly higher NSE with much higher variance and *MF-Lagrange* has serious difficulty in NSE reduction. This agrees with our analysis in Section , since an agent only incurs NSE when it is already pushing the box. *MF-Lagrange* treats classifier output as a blackbox and adjusts all the performed actions in an NSE-inducing trajectory. In contrast, MBGE utilizes the classifier gradient and can fine-adjust the actions only at timesteps when agent is pushing the box, resulting in better NSE performance. In autonomous driving domain, MBGE manages to reduce the average NSE to a very small value with lower variance. These results support our claim where our method is general enough to be applied to Markovian NSEs and yet achieve better performance than baselines.

To better understand how MBGE learns a safe policy, we plot a single MBGE training run in Figure 3. Every 50 epochs, the policy is executed in test environment and the reported reward and NSE are averaged over 100 test trajectories. The semi-transparent band indicates ± 0.5 standard deviation. In the first 200 epochs, the agent explores the environment and has not discovered that the box can be picked up to achieve higher reward, thus both the reward and NSE are low. After epoch 200, the agent learns to push the box to the goal and starts achieving better reward. At the same time, the amount of NSE increases because agent has not learned to avoid dangerous surfaces and this in turn increases the value of Lagrange Multiplier λ_i. This makes the effect of $\nabla_\theta J_{\hat{c}_i}(\pi_\theta)$ in (8) larger and causes the agent to adjust its policy based on both $\nabla_\theta J_R(\pi_\theta)$ and $\nabla_\theta J_{\hat{c}_i}(\pi_\theta)$. This can be observed toward the later part of training (after epoch 500), where the agent learns to jointly optimize both reward and NSE, eventually converges to a safe optimal policy.

Non-Markovian NSE Experiment We refer the readers to the supplement for additional experiments demonstrating that MBGE better satisfies non-Markovian NSE constraints than *MF-Lagrange* and *Markov-HA-S* baseline.

Continuous Domains

We experimented with two distinct tasks in continuous planning domains: Navigation (Faulwasser and Findeisen 2009) and Heating, Ventilation and Air Conditioning (HVAC) (Agarwal et al. 2010). We provide brief descriptions of these two tasks below and refer the readers to Bueno (2020) for complete specifications.

Navigation The agent is required to avoid a deceleration zone and move toward a goal position in a 2D continuous grid. Reward is defined as the Euclidean distance between the agent and goal position at every timestep. On top of the existing specifications, we defined an additional *dirty* zone $(2 \leq x \leq 4.5, 0 \leq y \leq 10)$, sitting between the agent's starting position and goal position. The agent incurs no NSE if it passes through the dirty zone fewer than two timesteps throughout the entire trajectory (mild NSE: $2-3$ timesteps, severe NSE: ≥ 4 timesteps in the dirty zone). Our target is to be NSE-free 95% of the time (i.e. $\mathbb{E}_{\tau\sim\pi_\theta}[C_0(\tau)] \geq 0.95$).

HVAC In HVAC control, the agent controls the amount of heated airflow into a set of inter-connected rooms. The original problems specifies room temperature to be between $20°C$ to $23.5°C$. It also incentivizes the agent to maintain the room temperatures at the midpoint of this temperature range (around $21.75°C$). Unbeknownst to the MDP designer, one of the rooms is actually a server room and there is a safety requirement to further keep the temperature of this room below $21°C$. As such, our non-Markovian NSEs are defined as follows: a trajectory is NSE-free if the server room temperature is higher than $21°C$ for no more than one *consecutive* timestep (mild NSE: $2-3$ consecutive timesteps, severe NSE: ≥ 4 consecutive timesteps above $21°C$). The NSE constraint is to have no-NSE 95% of the time (i.e. $\mathbb{E}_{\tau\sim\pi_\theta}[C_0(\tau)] \geq 0.95$).

Classifier Training The NSEs defined for the two tasks above are non-Markovian NSEs in nature and the whole trajectory needs to be inspected to determine the presence of an NSE, i.e. $C_i(\tau)$. As such, we use the GRU-based classifier $\hat{C}_i(\tau)$ illustrated in Figure 2 and report the percentages of test trajectories actually having no NSE, i.e. $\mathbb{E}_{\tau\sim\pi_\theta}[C_0(\tau)]$. For both navigation and HVAC tasks, we trained the classifier using close to 200k labelled trajectories, achieving around 99% classification accuracy.

Results and Discussion The boxplots in Figure 4 report the total reward and percentage of NSE-Free Trajectories

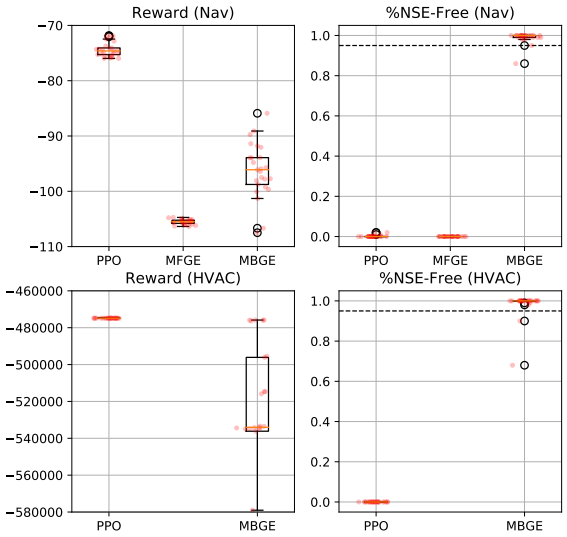

Figure 4: Continuous Domain Experiments (box plots)

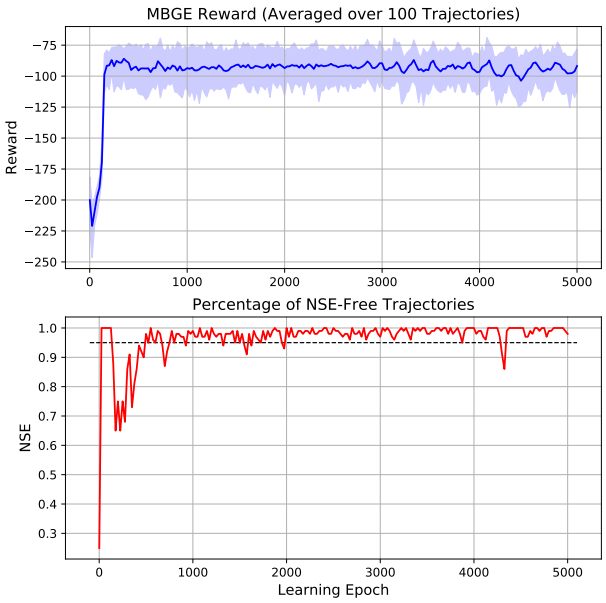

Figure 5: One MBGE Training Run for Navigation

achieved by policies trained using PPO, Model-Free Gradient Estimation (MFGE) and Model-Based Gradient Estimation methods (with the latter two being the proposed methods in this paper). The charts present the policy performance within their last 200 epochs in five independent training runs.

From the top half of Figure 4 (for navigation domain), PPO achieves the best reward while incurring NSE almost 100% of the time. This is expected since PPO simply optimizes reward, without being aware of the presence of NSEs. Our proposed MBGE method is able to strike good balance between reward optimization and NSE avoidance, achieving average reward around -95 while almost always satisfy the NSE constraint. The MFGE method significantly underperforms, collecting lower reward and never meets the NSE constraint. The high-variance nature of MFGE method creates difficulty in practice. Furthermore, the NSE we defined is typically observed over a smaller subpart of the trajectory. As discussed in Section , MFGE tends to overcorrect the entire trajectory and cannot perform targeted adjustment.

We do not include MFGE in HVAC performance reporting since it fails to converge to the constrained local optimum in all training runs. The bottom half of Figure 4 shows similar pattern, with PPO being able to maximize reward but violating constraint all the time. For MBGE, most runs converge to safe policy with average returns around -520k, supporting our claims that MBGE enables agent to optimize reward and NSEs jointly.

Figure 5 illustrates the MBGE learning process for the navigation domain (figure for HVAC can be found in the supplement). The solid blue line indicates the total reward averaged over 100 test trajectories and the semi-transparent band outlines the maximum and minimum reward among the 100 test trajectories. As the agent learns the move toward to goal, it starts to incur NSE (around epoch 150) since the shortest path would pass through the dirty zone. This increases λ_i and intensifies the effects of NSE term $\nabla_\theta J_{\hat{c}_i}(\pi_\theta)$ during policy update. Armed with the gradient information from the learned classifier, MBGE optimizes total return and NSE jointly and discovers a safe path bypassing the dirty zone (around epoch 500). The red curve demonstrates that the converged policy is NSE-free at least 95% of the time.

Conclusion

We have presented a method for safe MDP planning that avoids negative side effects (NSEs), which may arise during policy execution based on an incomplete model of a complex real world environment. Unlike previous works that require knowledge of numerical safety cost functions, our method learns a RNN-based classifier that learns to label state-action trajectories with different safety categories based on collected NSE dataset. Thus, our method can address a rich set of non-Markovian NSEs, unlike previous works which are limited to Markovian safety cost functions. Furthermore, we developed a model-based method that integrates the differentiable classifier with the MDP model to estimate the gradient of the classifier w.r.t. policy parameters, which is a better approach than an entirely model-free way of estimating gradients. Empirically, our method worked significantly better than a number of baselines.

Acknowledgments

This research/project is supported by the National Research Foundation Singapore and DSO National Laboratories un-

der the AI Singapore Programme (Award Number: AISG2-RP-2020-016).

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained policy optimization. In *International conference on machine learning*, 22–31. PMLR.

Agarwal, Y.; Balaji, B.; Gupta, R.; Lyles, J.; Wei, M.; and Weng, T. 2010. Occupancy-Driven Energy Management for Smart Building Automation. *Power*, 50(100): 150.

Alizadeh Alamdari, P.; Klassen, T. Q.; Toro Icarte, R.; and McIlraith, S. A. 2022. Be Considerate: Avoiding Negative Side Effects in Reinforcement Learning. In *AAMAS-22*, 18–26.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe Reinforcement Learning via Shielding. In *AAAI-18*, 2669–2678.

Altman, E. 1998. Constrained Markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Mathematical methods of operations research*, 48(3): 387–417.

Altman, E. 2021. *Constrained Markov Decision Processes*. CRC Press. ISBN 9781351458245.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *CoRR*, abs/1606.06565.

Bertsekas, D. 1999. *Nonlinear Programming*. Athena Scientific.

Bueno, T. P. 2020. rddlgym. https://github.com/thiagopbueno/rddlgym. Accessed: 2022-03-08.

Bueno, T. P.; de Barros, L. N.; Mauá, D. D.; and Sanner, S. 2019. Deep Reactive Policies for Planning in Stochastic Nonlinear Domains. *AAAI*, 33(01): 7530–7537.

Cho, K.; van Merriënboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. *SSST-8*, 103.

Chow, Y.; Nachum, O.; Faust, A.; Duenez-Guzman, E.; and Ghavamzadeh, M. 2019. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*.

Dalal, G.; Dvijotham, K.; Vecerik, M.; Hester, T.; Paduraru, C.; and Tassa, Y. 2018. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*.

Dietterich, T. G. 2017. Steps toward robust artificial intelligence. *AI Magazine*, 38(3): 3–24.

Faulwasser, T.; and Findeisen, R. 2009. Nonlinear model predictive path-following control. *Nonlinear model predictive control*, 384: 335–343.

Fu, M. C. 2006. Gradient estimation. *Handbooks in operations research and management science*, 13: 575–616.

Glasserman, P. 2004. *Monte Carlo methods in financial engineering*, volume 53. Springer.

Goodfellow, I. J.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. Cambridge, MA, USA: MIT Press. http://www.deeplearningbook.org.

Hadfield-Menell, D.; Milli, S.; Abbeel, P.; Russell, S. J.; and Dragan, A. 2017. Inverse reward design. *Advances in neural information processing systems*, 30.

Heess, N.; Wayne, G.; Silver, D.; Lillicrap, T.; Erez, T.; and Tassa, Y. 2015. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jansen, N.; Könighofer, B.; Junges, S.; Serban, A.; and Bloem, R. 2020. Safe Reinforcement Learning Using Probabilistic Shields. In *CONCUR 2020*.

Jothimurugan, K.; Bansal, S.; Bastani, O.; and Alur, R. 2021. Compositional reinforcement learning from logical specifications. *NeurIPS*, 34: 10026–10039.

Junges, S.; Jansen, N.; Dehnert, C.; and Topcu, U. 2016. Safety-Constrained Reinforcement Learning for MDPs. *TACAS*, 130.

Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.

Krakovna, V.; Orseau, L.; Kumar, R.; Martic, M.; and Legg, S. 2018. Penalizing side effects using stepwise relative reachability. *arXiv preprint arXiv:1806.01186*.

Krakovna, V.; Orseau, L.; Martic, M.; and Legg, S. 2019. Penalizing Side Effects using Stepwise Relative Reachability. In *AI Safety Workshop, IJCAI*.

Krakovna, V.; Orseau, L.; Ngo, R.; Martic, M.; and Legg, S. 2020. Avoiding side effects by considering future tasks. *Advances in Neural Information Processing Systems*, 33: 19064–19074.

Low, S. M.; Kumar, A.; and Sanner, S. 2022. Sample-Efficient Iterative Lower Bound Optimization of Deep Reactive Policies for Planning in Continuous MDPs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9): 9840–9848.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *nature*, 323(6088): 533–536.

Saisubramanian, S.; Kamar, E.; and Zilberstein, S. 2020. A Multi-Objective Approach to Mitigate Negative Side Effects. In *IJCAI-20*, 354–361.

Saisubramanian, S.; Kamar, E.; and Zilberstein, S. 2022. Avoiding Negative Side Effects of Autonomous Systems in the Open World. *Journal of Artificial Intelligence Research*, 74: 143–177.

Schulman, J.; Heess, N.; Weber, T.; and Abbeel, P. 2015. Gradient Estimation Using Stochastic Computation Graphs. In *NeurIPS*, 3528–3536.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Seuken, S.; and Zilberstein, S. 2007. Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs. In *UAI*, 344–351.

Shah, R.; and Krasheninnikov, D. 2019. Preferences Implicit in the State of the World. In *International Conference on Learning Representations (ICLR)*.

Simão, T. D.; Jansen, N.; and Spaan, M. T. J. 2021. AlwaysSafe: Reinforcement Learning Without Safety Constraint Violations During Training. In *AAMAS 2021*, 1226–1235.

Stooke, A.; Achiam, J.; and Abbeel, P. 2020. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, 9133–9143. PMLR.

Sutton, R. S.; Barto, A. G.; et al. 1998. Introduction to reinforcement learning. Vol. 135.

Tessler, C.; Mankowitz, D. J.; and Mannor, S. 2018. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*.

Turner, A. M.; Hadfield-Menell, D.; and Tadepalli, P. 2020. Conservative agency via attainable utility preservation. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 385–391.

Wray, K.; Zilberstein, S.; and Mouaddib, A.-I. 2015. Multi-Objective MDPs with Conditional Lexicographic Reward Preferences. *AAAI*, 29(1).

Supplemental Materials to Submission: Safe MDP Planning by Learning Temporal Patterns of Undesirable Trajectories and Averting Negative Side Effects

1 Hyperparameter Settings for Grid World Experiments

Neural Network Model	Hidden Units	Normalization	Learning Rate	Minibatch Size	Others
Classifier	[32, 32]	Batch Norm	0.0001	100	Dropout: 0.5
Policy Net	[32, 32]	Layer Norm	0.0003	100	PPO Clip: 0.2
Value Net	[32, 32]	Layer Norm	0.0003	100	-

Table 1: Hyperparameter Settings of Neural Network Models for Grid World Experiments

The initial value of Lagrange Multiplier ν_i used is 1.0. The learning rate for ν_i is 0.0003 for both boxpushing and driving tasks.

2 Hyperparameter Settings for Continuous Domain Experiments

Neural Network Model	Hidden Units	Normalization	Learning Rate	Minibatch Size	Others
GRU Classifier	[64, 64]	-	0.0001	100	Dropout: 0.5
Policy Net	[64, 64]	Layer Norm	0.0003	100	PPO Clip: 0.2
Value Net	[64, 64]	Layer Norm	0.0003	100	-

Table 2: Hyperparameter Settings of Neural Network Models for Continuous Domain Experiments

The initial value of Lagrange Multiplier ν_i used is 1.0. The learning rates for ν_i is 0.003 (Navigation task) and 0.005 (HVAC task).

3 Additional Experiment Results

Figure 1 illustrates the MBGE learning process for the HVAC task in continuous domain experiment.

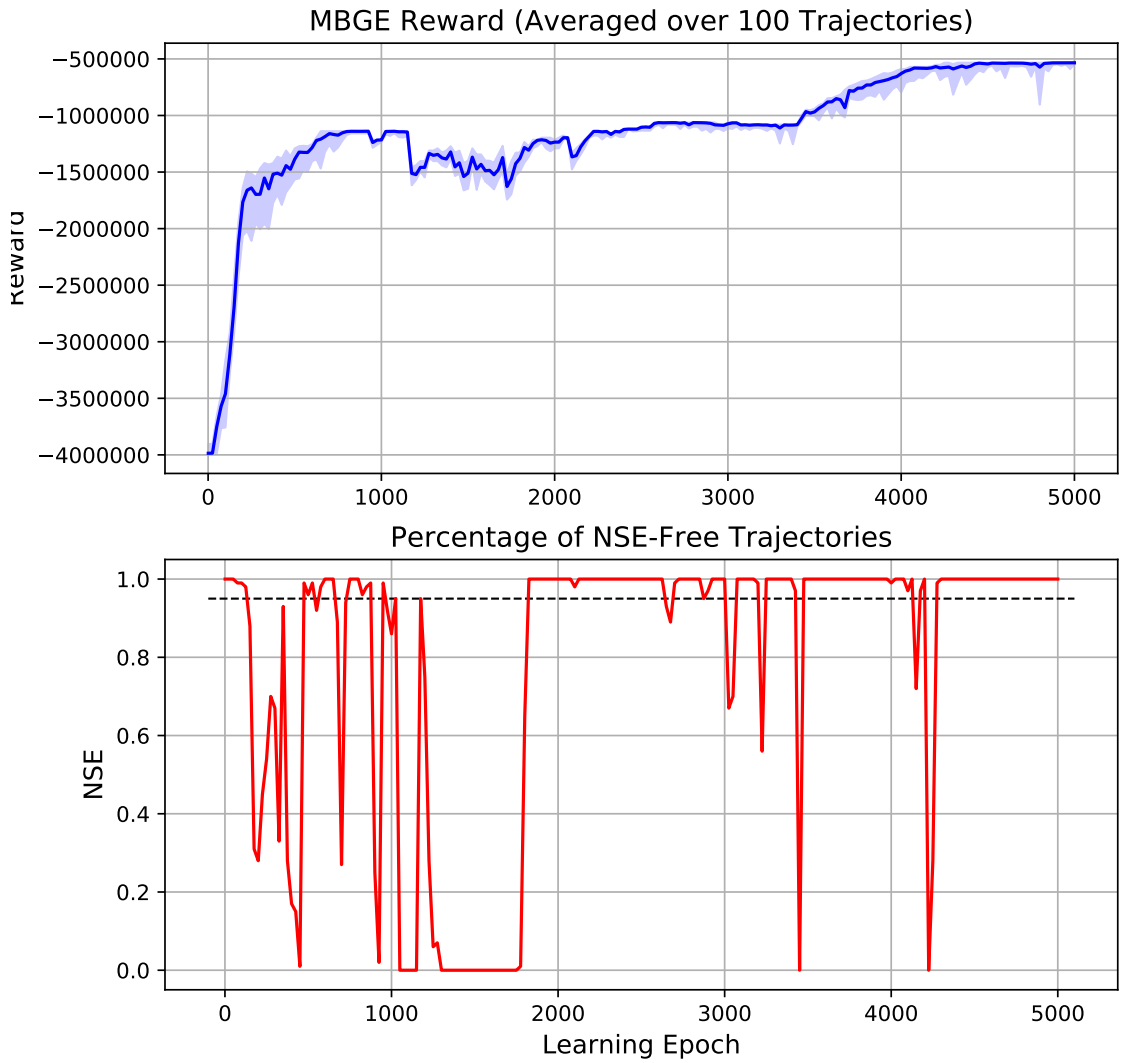

Figure 1: One MBGE Training Run for HVAC

4 Hardware & Software Specifications

All experiments were conducted on an AMD EPYC 7371 16-Core CPU machine with 512 GB memory, running Ubnutu 20.04 LTS. The Python package of PyTorch v1.11 was used to train the neural networks.

5 Code Appendix

Source code used in conducting the empirical experiments is available on https://github.com/siowmeng/Avert-NonMarkovNSE.