Probabilistic Inference for Safe Reinforcement Learning

Semester Project Report

Pablo Lahmann

July 14, 2024

Supervised by Dr. Lukas Meier, Jonas Hübotter and Yarden As

Department of Mathematics and ETH AI Center, ETH Zurich

Abstract

This semester project explores entropy-regularized algorithms in Safe Reinforcement Learning (Safe RL) through the lens of Probabilistic Inference. We present a derivation of a modification of the Soft Actor-Critic (SAC) algorithm for application in Safe RL, building on the work of Jonas Hübotter. The new algorithm is then implemented, based on an existing SAC codebase, provided by Yarden As. This report describes the implementation steps, challenges, and results obtained in several reinforcement learning environments.

Contents

1 Introduction

In this section, we outline the objectives and scope of our semester project, providing an overview of the underlying topics and motivations.

1.1 Goals

The main objective of this project is to introduce and implement a novel Safe RL algorithm, and test its performance in common Safe RL environments. In the future, this implementation might be used to compare the model to common baselines in the domain. In this work, we will delve into the theoretical foundations of the algorithm, outline the implementation steps, document encountered challenges, and distill key lessons learned.

Personally, this project presents a valuable opportunity to engage deeply with state-of-the-art algorithms in both RL and Safe RL. By implementing complex Machine Learning algorithms, I intend to strengthen my programming skills and gain practical insights into of algorithmic design and implementation.

1.2 Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning paradigm, in which sequential decisions are optimized to achieve a certain goal. It is commonly applied to areas such as robotics, healthcare, finance, autonomous systems, and more.

Key components of RL are the agent, environment, actions, and rewards. The agent describes an entity sequentially taking actions in an environment and obtaining a new reward value from every action. After each such step, the environment evolves into a new state from the point of view of the agent. A common example for an environment is balancing a pendulum that is attached to a cart moving along a 1-dim line. In this example, the actor is the cart, that can apply a force in either the positive or negative direction along its movement axis. The received reward could relate to how far away the pendulums angle is from standing upright.

Formally, we will regard the RL environment to be modeled by a Markov Decision Process (MDP), discrete in time. This is defined by the tuple $(\mathcal{X}, \mathcal{A}, p)$, where $\mathcal{X}$ is the continuous state space, and $\mathcal{A}$ the continuous space of actions that can be taken. We say that at any time step t, we are in the state $\mathbf{x}_t \in \mathcal{X}$. Taking the action $\mathbf{a}_t \in \mathcal{A}$ results in the agent arriving in state $\mathbf{x}_{t+1} \in \mathcal{X}$. Which value exactly $\mathbf{x}_{t+1}$ attains after taking an action is regarded to be probabilistic and governed by the so-called *dynamics model* $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t)$.

To guide the actor in its decision making, we introduce a reward $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$, which the actor tries to maximize. Note that both functions p and r are unknown to the agent.

The actor is represented by a policy $\pi(\mathbf{a} \mid \mathbf{x})$, which is a probability distribution over the actions $\mathcal{A}$ and decides the next action $\mathbf{a}$ of the agent, given a state $\mathbf{x}$. In this project we consider policies $\pi_{\boldsymbol{\theta}}$ characterized by parameters $\boldsymbol{\theta} \in \Theta$. During training, we try to find good

parameter values $\boldsymbol{\theta}$, which correspond to a policy that maximizes the expected cumulative rewards. For more information about basic theory in RL, we refer to Hübotter [2023].

A key difference to supervised learning, and thus challenge of RL, is that the observed data used for training is entirely dependent on the actions taken. This introduces a trade-off between exploration of the state space and maximization of rewards gained in already known states, which the actor needs to balance.

1.3 Safe Reinforcement Learning

The concept of safety in RL addresses a fundamental challenge encountered when deploying RL solutions in real-world applications. It ensures that agents do not engage in behaviors that could potentially pose risks or dangers in their operational environments. For instance, in autonomous driving scenarios, where RL agents represent cars navigating traffic, avoiding states that could lead to accidents or traffic violations is crucial. While RL models are typically trained in simulated environments where risky actions have no real consequences, real-world complexities like street traffic cannot be efficiently simulated. This requires agents to be trained in real-world scenarios. It is necessary to introduce strategies, which discourage the agent from visiting such *unsafe* states during inference, and sometimes even during training. This adds another layer of complexity to the agent, since next to reward maximization it must now carefully consider what states to avoid.

Formally, this is done by introducing an unknown cost function $c : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$, which is associated with an action $\mathbf{a}$ taken from a certain state $\mathbf{x}$. A common strategy for finding a *safe policy*, is to maximize the expected cumulative rewards, while fulfilling an upper bound constraint on the expected cumulative costs. This is a constrained optimization problem. For more basic theory in Safe RL see Hübotter [2023] (section on "Contstrained Exploration").

1.4 Probabilistic Inference

Probabilistic inference is a statistical framework for updating beliefs about events based on observed related information, and thereby quantifying the confidence in certain outcomes. For example, we could try to predict whether a new friend would enjoy a particular movie, without knowing anything about their preferences. Later, we might hear that their favorite movie is within the same genre. Upon hearing this, our confidence, that the friend will enjoy the movie, rises. Thereby updating our beliefs based on related information.

This idea is formalized using Bayes' rule for two random vectors $\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m$

$$p(\mathbf{x} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x}) \cdot p(\mathbf{x})}{p(\mathbf{y})}. \tag{1.1}$$

We formulate our initial belief through the *prior* $p(\mathbf{x})$. After observing $\mathbf{y}$, the updated belief about $\mathbf{x}$ is the *posterior* $p(\mathbf{x} \mid \mathbf{y})$, which can be computed from the *likelihood* $p(\mathbf{y} \mid \mathbf{x})$. In terms of the previous example, $\mathbf{x}$ describes whether our friend will like the new movie, and $\mathbf{y}$ that their favorite movie is of the same genre.

In the context of (Safe) RL, probabilistic inference gives a very natural way of quantifying uncertainty in model predictions. This allows for informed decision making under uncertainty about rewards or safety.

1.5 Safe Soft Actor Critic

RL can be viewed as a probabilistic inference problem within certain types of graphical models Levine [2018]. In this framework, the goal is to infer the optimal policy that maximizes expected cumulative rewards, given the observed states and dynamics of the environment.

The Soft Actor-Critic (SAC) algorithm Haarnoja et al. [2018] is based on the concept of entropy regularization. This means, the policy is designed to maximize expected cumulative rewards, while encouraging exploration through the simultaneous maximization of entropy. This can be interpreted as a probabilistic inference problem Levine [2018] with respect to Hidden Markov Models (HMMs). Within this semester project we explore the extension of the SAC algorithm to the Safe RL domain, by considering a modification of the HMM in Levine [2018], which incorporates safety into its structure. This eventually leads to a safe version of SAC. Within this report we refer to this off-policy and model-free algorithm as *SafeSAC*.

2 Theory

In this chapter we describe the process of deriving the SafeSAC model objective, and compare it to other popular algorithms in Safe RL. Further note, that we assume the reader is familiar with basic concepts in Probabilistic Artificial Intelligence, such as Surprise, Entropy or KL Divergence. For foundations, refer to Hübotter [2023].

2.1 SafeSAC Objective

Remember that the (Safe) RL environment can be described with a Markov Decision Process (MDP) $(\mathcal{X}, \mathcal{A}, p)$. Let us particularly consider a process of T steps, i.e. transitions from one state $\mathbf{x}_t$ to the next $\mathbf{x}_{t+1}$ (given an action $\mathbf{a}_t$), with $t \in \{1, ..., T-1\}$. This induces a trajectory $\tau = (\mathbf{x}_1, \mathbf{a}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T-1}, \mathbf{a}_{T-1}, \mathbf{x}_T)$. Naturally, in RL we are not particularly interested in maximizing the reward for one particular action, but rather the cumulative reward for the entire trajectory. In the spirit of probabilistic inference, we formulate the probability of a trajectory, given a followed policy $\pi_{\boldsymbol{\theta}}$ by

$$\Pi_{\boldsymbol{\theta}}(\tau) = p(\mathbf{x}_1) \prod_{t=1}^{T} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{x}_t) p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t). \tag{2.1}$$

As mentioned in 1.5, to extend this idea to the Safe RL domain, we formulate the RL problem as a modified version of a Hidden Markov Model (HMM). More precisely, at every time step we add hidden optimality and security variables $O_t, S_t \in \{0, 1\}$, indicating whether the played action $\mathbf{a}_t$ was resp. optimal or safe in state $\mathbf{x}_t$. For simplification we denote the events $O_t = 1$ and $S_t = 1$ (i.e. played action $\mathbf{a}_t$ was optimal and safe) as $\mathcal{O}_t$ and $\mathcal{S}_t$ respectively.

Clearly we are interested in trajectories which are conditioned on optimality and safety. Their distribution can be written in the following way

$$\Pi^s_\star(\tau) \doteq p(\tau \mid \mathcal{O}_{1:T}, \mathcal{S}_{1:T}) = \frac{p(\tau, \mathcal{O}_{1:T}, \mathcal{S}_{1:T})}{p(\mathcal{O}_{1:T}, \mathcal{S}_{1:T})} \tag{2.2}$$

$$\propto p(\tau, \mathcal{O}_{1:T}, \mathcal{S}_{1:T}) = p(\mathbf{x}_1) \prod_{t=1}^{T} p(\mathbf{a}_t, \mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t) p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t). \tag{2.3}$$

Note: We assume that the initial state distribution and dynamics are fixed, i.e. $p(\mathbf{x}_1 \mid \mathcal{O}_{1:T}, \mathcal{S}_{1:T}) = p(\mathbf{x}_1)$ and $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t, \mathcal{O}_{1:T}, \mathcal{S}_{1:T}) = p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t)$.

We define $\beta(\mathbf{a}_t \mid \mathbf{x}_t)$ to be the probability of action $\mathbf{a}_t$ that was optimal and safe. We write

$$\beta(\mathbf{a}_t \mid \mathbf{x}_t) \doteq p(\mathbf{a}_t, \mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t) = p(\mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) p(\mathbf{a}_t \mid \mathbf{x}_t) \tag{2.4}$$

Notice that in our description of the Hidden Markov Model, we have not yet discussed the distribution of the variables $\mathcal{O}_t$ and $\mathcal{S}_t$. In theory, this distribution can be chosen arbitrarily. But similarly to Levine [2018] we assume

$$p(\mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) = p(\mathcal{O}_t \mid \mathcal{S}_t, \mathbf{x}_t, \mathbf{a}_t) \cdot p(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) = \exp\left(\frac{1}{\lambda} r(\mathbf{x}_t, \mathbf{a}_t)\right) \cdot \mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t). \quad (2.5)$$

Where $r(\mathbf{x}_t, \mathbf{a}_t)$ is the reward function, and $\lambda > 0$ an adjustable parameter.

Note: For $\mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t)$ we will use an independently trained classifier to predict $S_t \doteq \mathbb{1}\{c(\mathbf{x}_t, \mathbf{a}_t) \leq 0\}$ during implementation of SafeSAC.

Note: Since the choice for the above distribution is arbitrary, in *eq.* (2.4) we can assume wlog that the prior policy $p(\mathbf{a}_t \mid \mathbf{x}_t)$ is uniform on the set of actions $\mathcal{A}$ for all t. This simplification does not loose generality because any non-uniform $p(\mathbf{a}_t \mid \mathbf{x}_t)$ can be attributed to $p(\mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t)$ via e.g. a modified reward function. Thus we can rewrite *eq.* (2.4)

$$\begin{aligned} \beta(\mathbf{a}_t \mid \mathbf{x}_t) &\propto p(\mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) && (2.6) \\ &\propto \exp\left(\frac{1}{\lambda} r(\mathbf{x}_t, \mathbf{a}_t)\right) \cdot \mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t). && (2.7) \end{aligned}$$

The goal of Safe RL is to find an optimal and safe policy $\pi^\star_s(\mathbf{a}_t \mid \mathbf{x}_t) \doteq p(\mathbf{a}_t \mid \mathbf{x}_t, \mathcal{O}_{t:T}, \mathcal{S}_{t:T})$, which induces the trajectory distribution $\Pi^s_\star(\tau)$ from *eq.* (2.2).

We approximate $\Pi^s_\star$ using a parametrization $\Pi_{\boldsymbol{\theta}} \approx \Pi^s_\star$, with $\boldsymbol{\theta} \in \Theta$. Since we are trying to approximate a distribution, it makes sense to try to minimize the reverse KL-Divergence, like in common approaches in Variational Inference. The KL-Divergence is given by

$$\mathrm{D}(q(x) \| p(x)) = \mathbb{E}_{\sim q(x)}\left[\log \frac{q(x)}{p(x)}\right] = \mathrm{H}[q \| p] - \mathrm{H}[q] \quad (2.8)$$

where $q(x)$ and $p(x)$ are probability distributions. Specifically, we are looking for $\boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta} \in \Theta} \mathrm{D}(\Pi_{\boldsymbol{\theta}} \| \Pi^s_\star)$, which can be expressed in terms of surprise and entropy

$$\begin{aligned} \arg\min_{\boldsymbol{\theta} \in \Theta} \mathrm{D}(\Pi_{\boldsymbol{\theta}} \| \Pi^s_\star) &= \arg\min_{\boldsymbol{\theta} \in \Theta} \mathrm{H}[\Pi_{\boldsymbol{\theta}} \| \Pi^s_\star] - \mathrm{H}[\Pi_{\boldsymbol{\theta}}] \\ &= \arg\min_{\boldsymbol{\theta} \in \Theta} \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\theta}}}[\mathrm{S}[\Pi^s_\star(\tau)] - \mathrm{S}[\Pi_{\boldsymbol{\theta}}(\tau)]] \\ &= \arg\min_{\boldsymbol{\theta} \in \Theta} \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\theta}}}\left[\sum_{t=1}^{T} \mathrm{S}[\beta(\mathbf{a}_t \mid \mathbf{x}_t)] - \mathrm{S}[\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{x}_t)]\right] && (2.9) \\ &= \arg\max_{\boldsymbol{\theta} \in \Theta} \sum_{t=1}^{T} \mathbb{E}_{(\mathbf{x}_t, \mathbf{a}_t) \sim \Pi_{\boldsymbol{\theta}}}[\log p(\mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) - \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{x}_t)]. && (2.10) \end{aligned}$$

Where we write $p(\mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t)$ instead of $\beta(\mathbf{a}_t \mid \mathbf{x}_t)$, because $p(\mathbf{a}_t \mid \mathbf{x}_t)$ is constant (since we assume it is uniform), so $\log(p(\mathbf{a}_t \mid \mathbf{x}_t))$ is a constant additive term to the whole objective, thus not affecting the optimization. In the objective *eq.* (2.10), $\log p(\mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t)$ introduces

a penalty for unsafe and suboptimal actions and states. We call this a logarithmic barrier. The logarithmic barrier discourages exploration close the suboptimal and unsafe boundary. Note that this could potentially be a reason for the model to struggle to expand its safe set.

Further note that we can write

$$\log p(\mathcal{O}_t, \mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) = \frac{1}{\lambda} r(\mathbf{x}_t, \mathbf{a}_t) + \log \mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t). \tag{2.11}$$

At a later point we introduce a safety critic, which learns the quantity $\mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t)$ in the same way that the critic in SAC learns the rewards.

The objective from *eq.* (2.10) simplifies to

$$\arg\max_{\boldsymbol{\theta} \in \Theta} \sum_{t=1}^{T} \mathbb{E}_{(\mathbf{x}_t, \mathbf{a}_t) \sim \Pi_{\boldsymbol{\theta}}} \Big[\frac{1}{\lambda} r(\mathbf{x}_t, \mathbf{a}_t) + \log \mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) + \mathrm{H}[\pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}_t)] \Big]. \tag{2.12}$$

This is quite similar to the entropy-regularized objective of the SAC algorithm, but here we have an additional term in the expectation, which addresses safety. The SafeSAC model tries optimizing this objective.

Note: It can be shown that the optimal policy to the objective in 2.12 behaves in some sense pessimistically with respect to downstream rewards and safety. But still less pessimistic than greedily disregarding downstream results altogether.

Note: Intuitively, minimizing the objective 2.12 can be seen as the agent hallucinating that it acts optimally and safely, and then acting to minimize the surprise from deviating from this hallucination. See section "Entropy Regularization as Probabilistic Inference" in Hübotter [2023] for a more detailed explanation.

2.2 SafeSAC

With SafeSAC we are essentially looking to apply an actor critic approach to solve the objective in *eq.* (2.10).

The regular SAC algorithm is based on a 'soft' version of the Q and V functions Haarnoja et al. [2018]. They can be derived from the general form of the objective in *eq.* (2.9). Their definition is generally given by

$$Q(\mathbf{x}_t, \mathbf{a}_t) \doteq \log \beta(\mathbf{a}_t \mid \mathbf{x}_t) + \mathbb{E}_{\mathbf{x}_{t+1} \sim p(\cdot \mid \mathbf{x}_t, \mathbf{a}_t)}[V(\mathbf{x}_{t+1})] \qquad \text{and} \tag{2.13}$$

$$V(\mathbf{x}_t) \doteq \log \int_{\mathcal{A}} \exp(Q(\mathbf{x}_t, \mathbf{a}_t))\, d\mathbf{a}_t \tag{2.14}$$

Remember that in our discussion, $\beta(\mathbf{a}_t \mid \mathbf{x}_t)$ accounts for optimality and safety simultaneously (see 2.4).

We can compare this to the SAC Q-function and V-function in the non-safe RL setting. Let us denote them by Q' and V'. In this case we regard the probability of an action only being optimal, written as $\beta'(\mathbf{a}_t \mid \mathbf{x}_t) = p(\mathbf{a}_t, \mathcal{O}_t \mid \mathbf{x}_t) = \beta(\mathbf{a}_t \mid \mathbf{x}_t)/\mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t)$. Note that *eq.* (3.7) and *eq.* (3.8) equivalently hold for Q', V', β'. From this, we can derive the following dependencies

$$Q(\mathbf{x}_t, \mathbf{a}_t) \doteq Q'(\mathbf{x}_t, \mathbf{a}_t) - \gamma(\mathbf{x}_t, \mathbf{a}_t) \qquad \text{and} \tag{2.15}$$

$$V(\mathbf{x}_t) \doteq V'(\mathbf{x}_t) - \mathbb{E}_{\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_t)}[\gamma(\mathbf{x}_t, \mathbf{a}_t)] \tag{2.16}$$

Where $\gamma(\mathbf{x}_t, \mathbf{a}_t)$ is the expected joint surprise about being safe along the trajectory. It is given by

$$\gamma(\mathbf{x}_t, \mathbf{a}_t) \doteq -\log \mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) + \mathbb{E}_{\mathbf{x}_{t+1}, \mathbf{a}_{t+1}}[\gamma(\mathbf{x}_{t+1}, \mathbf{a}_{t+1})] \tag{2.17}$$

The SafeSAC model optimizes these Q and V functions. For the final implementation, we could either learn Q', V' and γ separately, or directly learn Q and V.

2.3 Related Work

In this section we give a brief description of important and related algorithms in Safe RL. We describe their general approach, as well as compare them to SafeSAC.

Constrained Policy Optimization (CPO) A very common baseline in Safe RL is the CPO algorithm Achiam et al. [2017]. A special property of CPO is that it provides some guarantees for satisfying safety constraints on the expected cumulative cost throughout the training process. In contrast, many other safe RL algorithms only guarantee that safety constraints will be fulfilled eventually. At each step k for maximizing the objective, CPO fundamentally restricts the new policy π to be within a local neighborhood of the previous iterate π_{k-1}, and to be within a set of feasible distributions, which all fulfill the safety constraint. This is done by closely approximating both the objective and constraints with functions that are easy to estimate.

A key difference between SafeSAC and CPO is that CPO stays within a constrained optimization framework, while SafeSAC incorporates safety directly into the objective through a logarithmic barrier term.

Constrained Variational Policy Optimization (CVPO) CVPO Liu et al. [2022] follows the primal-dual approach. This type of approach simultaneously maximizes the expected reward and minimizes a safety penalty function, balancing the trade-off between performance and safety constraints. The primal-dual method is widely used in Safe RL due to its effectiveness in ensuring that policies adhere to safety constraints while optimizing the reward. Similarly to SafeSAC, CVPO also comes from viewing Safe RL through the lens of probabilistic inference. In particular, CVPO formulates safety as part of a variational inference problem, optimizing actions that satisfy constraints on the cost while maximizing reward. This makes CVPO a good example of how regarding Safe RL as a probabilistic inference problem can result in powerful new algorithms.

3 Implementation

In this chapter we explain how the SafeSAC algorithm is implemented, and discuss challenges that occurred during implementation and testing in different (Safe) RL environments.

3.1 Gradient Updates and Pseudocode

We describe the implementation of the previously described SafeSAC objective *eq.* (2.12). For this we will learn the functions Q',V' and γ independently.

Q' and V' Functions from SAC We use parameters $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ to approximate Q' and V', i.e. $Q'_{\boldsymbol{\phi}}$ and $V'_{\boldsymbol{\psi}}$. For training, the squared error objectives are optimized

$$J_{Q'}(\boldsymbol{\phi}) \doteq \mathbb{E}_{(\mathbf{x}_t,\mathbf{a}_t)\sim\mathcal{D}}\Big[\frac{1}{2}\Big(Q'_{\boldsymbol{\phi}}(\mathbf{x}_t,\mathbf{a}_t) - \log\beta'(\mathbf{a}_t \mid \mathbf{x}_t) - \mathbb{E}_{\mathbf{x}_{t+1}\sim p(\cdot|\mathbf{x}_t,\mathbf{a}_t)}\Big[V'_{\boldsymbol{\psi}}(\mathbf{x}_{t+1})\Big]\Big)^2\Big], \tag{3.1}$$

$$J_{V'}(\boldsymbol{\psi}) \doteq \mathbb{E}_{\mathbf{x}_t\sim\mathcal{D}}\Big[\frac{1}{2}\Big(V'_{\boldsymbol{\psi}}(\mathbf{x}_t) - \mathbb{E}_{\mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_t)}\Big[Q'_{\boldsymbol{\phi}}(\mathbf{x}_t,\mathbf{a}_t) - \log\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{x}_t)\Big]\Big)^2\Big] \tag{3.2}$$

Where $\mathcal{D}$ is a replay buffer.

Unbiased gradient estimators are given by

$$\widehat{\boldsymbol{\nabla}}_{\boldsymbol{\phi}} J_{Q'}(\boldsymbol{\phi}) \doteq \boldsymbol{\nabla}_{\boldsymbol{\phi}}\, Q'_{\boldsymbol{\phi}}(\mathbf{x}_t,\mathbf{a}_t)\Big(Q'_{\boldsymbol{\phi}}(\mathbf{x}_t,\mathbf{a}_t) - \log\beta'(\mathbf{a}_t \mid \mathbf{x}_t) - V'_{\boldsymbol{\psi}}(\mathbf{x}_{t+1})\Big), \tag{3.3}$$

$$\widehat{\boldsymbol{\nabla}}_{\boldsymbol{\psi}} J_{V'}(\boldsymbol{\psi}) \doteq \boldsymbol{\nabla}_{\boldsymbol{\psi}}\, V'_{\boldsymbol{\psi}}(\mathbf{x}_t)\Big(V'_{\boldsymbol{\psi}}(\mathbf{x}_t) - Q'_{\boldsymbol{\phi}}(\mathbf{x}_t,\mathbf{a}_t) + \log\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{x}_t)\Big). \tag{3.4}$$

Safety Q^c and V^c Functions Address safety by training any classifier (e.g. MLP) $\mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t,\mathbf{a}_t)$ to predict whether the cost $c(\mathbf{x}_t,\mathbf{a}_t)$ for $\mathbf{x}_t,\mathbf{a}_t$ given is negative, or not (i.e. to predict the probability of $\mathbb{1}\{c(\mathbf{x}_t,\mathbf{a}_t)\leq 0\}$).
For parameterizing γ we define the corresponding "safety" value functions

$$Q^s(\mathbf{x}_t,\mathbf{a}_t) \doteq -\log\mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t,\mathbf{a}_t) + \mathbb{E}_{\mathbf{x}_{t+1}\sim p(\cdot|\mathbf{x}_t,\mathbf{a}_t)}[V^s(\mathbf{x}_{t+1})] \tag{3.5}$$

$$V^s(\mathbf{x}_t) \doteq \mathbb{E}_{\mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_t)}[Q^s(\mathbf{x}_t,\mathbf{a}_t)] \tag{3.6}$$

Note that $Q^s(\mathbf{x}_t,\mathbf{a}_t) = \gamma(\mathbf{x}_t,\mathbf{a}_t)$ from *eq.* (2.17). This allows us to write the following new dependencies for the Q and V value functions corresponding to the Q and V of SafeSAC.

$$Q(\mathbf{x}_t,\mathbf{a}_t) \doteq Q'(\mathbf{x}_t,\mathbf{a}_t) - Q^s(\mathbf{x}_t,\mathbf{a}_t) \qquad \text{and} \tag{3.7}$$

$$V(\mathbf{x}_t) \doteq V'(\mathbf{x}_t) - V^s(\mathbf{x}_t) \tag{3.8}$$

Let us parameterize Q^s and V^s with the parameters $\boldsymbol{\varphi}$ and $\boldsymbol{\omega}$ respectively. During training, we can update $Q^s_{\boldsymbol{\varphi}}$ and $V^s_{\boldsymbol{\omega}}$ iteratively, following the the squared error objectives, analogously to *eq.* (3.1) and *eq.* (3.2)

$$J_{Q^s}(\boldsymbol{\varphi}) \doteq \mathbb{E}_{(\mathbf{x}_t,\mathbf{a}_t)\sim\mathcal{D}}\Big[\frac{1}{2}\Big(Q^s_{\boldsymbol{\varphi}}(\mathbf{x}_t,\mathbf{a}_t) - \log\mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t,\mathbf{a}_t) - \mathbb{E}_{\mathbf{x}_{t+1}\sim p(\cdot|\mathbf{x}_t,\mathbf{a}_t)}[V^s_{\boldsymbol{\omega}}(\mathbf{x}_{t+1})]\Big)^2\Big], \tag{3.9}$$

$$J_{V^s}(\boldsymbol{\omega}) \doteq \mathbb{E}_{\mathbf{x}_t\sim\mathcal{D}}\Big[\frac{1}{2}\Big(V^s_{\boldsymbol{\omega}}(\mathbf{x}_t) - \mathbb{E}_{\mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_t)}\Big[Q^s_{\boldsymbol{\varphi}}(\mathbf{x}_t,\mathbf{a}_t)\Big]\Big)^2\Big] \tag{3.10}$$

Analogously to before, this gives us the following gradient estimates

$$\widehat{\nabla}_{\boldsymbol{\varphi}} J_{Q^s}(\boldsymbol{\varphi}) \doteq \nabla_{\boldsymbol{\varphi}} Q^s_{\boldsymbol{\varphi}}(\mathbf{x}_t, \mathbf{a}_t)\Big(Q^s_{\boldsymbol{\varphi}}(\mathbf{x}_t, \mathbf{a}_t) - \log \mathbb{P}(\mathcal{S}_t \mid \mathbf{x}_t, \mathbf{a}_t) - V^s_{\boldsymbol{\omega}}(\mathbf{x}_{t+1})\Big), \tag{3.11}$$

$$\widehat{\nabla}_{\boldsymbol{\omega}} J_{V^s}(\boldsymbol{\omega}) \doteq \nabla_{\boldsymbol{\omega}} V^s_{\boldsymbol{\omega}}(\mathbf{x}_t)\Big(V^s_{\boldsymbol{\omega}}(\mathbf{x}_t) - Q^s_{\boldsymbol{\varphi}}(\mathbf{x}_t, \mathbf{a}_t)\Big), \tag{3.12}$$

Policy To apply the reparametrization trick for gradient estimation, we rewrite the policy using a neural network $\mathbf{a}_t \doteq f_{\boldsymbol{\theta}}(\epsilon_t; \mathbf{x}_t)$ with $\epsilon_t \sim \mathcal{N}(0, 1)$. For training, we optimize the KL-Divergence

$$J_\pi(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_t \sim \mathcal{D}}[\mathrm{D}(\pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}_t) \| \pi^\star(\cdot \mid \mathbf{x}_t))] \tag{3.13}$$

Where $\pi^\star$ is solution to the objective of SafeSAC *eq.* (2.12). It can be shown that the optimal policy is given by $\pi^\star(\mathbf{a}_t \mid \mathbf{x}_t) = \exp(Q(\mathbf{x}_t, \mathbf{a}_t))/\exp(V(\mathbf{x}_t))$. Implying that the objective function for optimizing $\pi_{\boldsymbol{\theta}}$ can be approximated by

$$\mathbb{E}_{\mathbf{x}_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}(0,1)}[\log \pi_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\epsilon_t; \mathbf{x}_t) \mid \mathbf{x}_t) - Q_{\boldsymbol{\phi},\boldsymbol{\varphi}}(\mathbf{x}_t, f_{\boldsymbol{\theta}}(\epsilon_t; \mathbf{x}_t)) + V_{\boldsymbol{\psi},\boldsymbol{\omega}}(\mathbf{x}_t)]. \tag{3.14}$$

Note that $\pi_{\boldsymbol{\theta}}$ is implicitly defined through $f_{\boldsymbol{\theta}}$. Further, by *eq.* (3.7) and *eq.* (3.8), Q and V are implicitly parameterized respectively by $(\boldsymbol{\phi}, \boldsymbol{\varphi})$ and $(\boldsymbol{\psi}, \boldsymbol{\omega})$.

The gradient can be approximated with

$$\widehat{\nabla}_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta}) \doteq \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{x}_t) + (\nabla_{\mathbf{a}_t} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{x}_t) - \nabla_{\mathbf{a}_t} Q_{\boldsymbol{\phi},\boldsymbol{\varphi}}(\mathbf{x}_t, \mathbf{a}_t))\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\epsilon_t; \mathbf{x}_t). \tag{3.15}$$

Algorithm A formulation of the update steps involved in training the SafeSAC algorithm can be found in Algorithm 1.

Algorithm 1: Safe Soft Actor-Critic

Initialize parameters $\boldsymbol{\varphi}, \boldsymbol{\omega}, \boldsymbol{\phi}, \boldsymbol{\psi}, \boldsymbol{\theta}$
for *each iteration* **do**
  for *each environment step* **do**
    $\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}_t)$
    $\mathbf{x}_{t+1} \sim p(\cdot \mid \mathbf{x}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_t, \mathbf{a}_t, r(\mathbf{x}_t, \mathbf{a}_t), \mathbf{x}_{t+1})\}$
  end
  Update classifier on samples from $\mathcal{D}$
  `// safety critic update`
  Sample mini-batch of transitions $(\mathbf{x}_t, \mathbf{a}_t, r_t, \mathbf{x}_{t+1})$ from $\mathcal{D}$
  $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} - \lambda_{Q^s} \widehat{\boldsymbol{\nabla}}_{\boldsymbol{\varphi}} J_{Q^s}(\boldsymbol{\varphi})$
  $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \lambda_{V^s} \widehat{\boldsymbol{\nabla}}_{\boldsymbol{\omega}} J_{V^s}(\boldsymbol{\omega})$
  `// critic update`
  $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \lambda_{Q'} \widehat{\boldsymbol{\nabla}}_{\boldsymbol{\phi}} J_{Q'}(\boldsymbol{\phi})$
  $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \lambda_{V'} \widehat{\boldsymbol{\nabla}}_{\boldsymbol{\psi}} J_{V'}(\boldsymbol{\psi})$
  `// Policy update`
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda_{\pi} \widehat{\boldsymbol{\nabla}}_{\boldsymbol{\theta}} J_{\pi}(\boldsymbol{\theta})$
  `// Target networks polyak update`
  $\boldsymbol{\psi}_{\text{target}} \leftarrow \tau \boldsymbol{\psi} + (1 - \tau) \boldsymbol{\psi}_{\text{target}}$
  $\boldsymbol{\omega}_{\text{target}} \leftarrow \tau \boldsymbol{\omega} + (1 - \tau) \boldsymbol{\omega}_{\text{target}}$
end

3.2 Coding

In this section we discuss the coding process of this semester project. The personal goal of this project was to understand and modify complex code for state-of-the-art RL models. We describe the general setup, what the provided codebase looked like, describe the environments which we worked on, and lastly, what challenges we encountered and how they were resolved.

3.2.1 Setup

An existing implementation in `Python` of the SAC algorithm was provided in As [2023]. Additionally, access to GPUs was granted, which could be used to add batches to a running queue for execution. The codebase is neatly organized, implementing all necessary functionalities for efficiently running a complex RL algorithm. Among others, this includes:

- Robust logging functionalities, allowing for detailed tracking of metrics and video rendering.
- Replay buffer.
- Configurations to clearly define hyperparameters, environment settings, and experimental setup.
- Acting logic of an agent.
- Framework to facilitate parallel training through asynchronously managing episodic interactions.
- Learning logic to update the agents' policies.
- Trainer managing the training loop, including initialization and training epochs.
- Implementations of some environments, defining specific tasks for the agent.
- Environment wrappers to modify or augment an environment's behavior.
- Implementation of the SAC algorithm and its gradient updates.

For specific information about the implementation, such as certain hyperparameters, refer to the implementation code on GitHub As [2023].

As part of this project, the code As [2023] was augmented to run SAC and SafeSAC (see branch `semester_project_lahmann` in the repository) on the following environments/tasks:

- `pendulum` (`Pendulum-v1`): is part of the open-source package `Gymnasium` Farama-Foundation [2024], which provides a standardized set of environments in which RL algorithms can be tested. This task describes a pendulum that is fixed at one end and free at the other. The pendulum starts at a random initial position, and the goal of the agent is to swing it into an upright position by applying a rotational force around the fixed point.

- `dm_cartpole`: is based on the `dm_control` package Tunyasuvunakool et al. [2020], developed by Google DeepMind to simulate physics-based RL environments. The `cartpole` environment simulates a cart moving along a straight line with a pole attached to one end. The other end of the pole is free to swing around, allowing it to move dynamically as the cart moves. Within this environment, we look at two different tasks
  - `dm_cartpole_balance`: The pole starts at a near-upright position, with primary goal to keep the pole balanced upright for as long as possible.
  - `dm_cartpole_swingup`: The pole starts in a downward position, with the goal to swing up the pole, and keep it balanced for as long as possible. Clearly, this task is a lot more complex than `dm_cartpole_balance`. We later augmented this environment with a cost, depending on the cart leaving a safe region on the movement line.
- `point_to_goal`: is an environment within the `safe-adaptation-gym` As [2024], implementing cost in its environment, to test Safe RL algorithms. In `point_to_goal`, the agent is a point mass, which can move in any direction on a 2D plane. The goal is to navigate to a target goal position, while trying to avoid different types of obstacles. A cost is incurred, if any obstacles are hit.

The provided codebase worked with environments from the `Gymnasium` package. Thus, both `dm_control` and `safe-adaptation-gym` had to be adapted to properly work within the code.

SafeSAC was implemented in an iterative fashion, continuously improving its performance. This proved to be more intricate than expected, even leaving some room for further experimentation after the end of this project. Nevertheless, the existing SafeSAC implementation has seem some considerable improvements throughout this process.

3.2.2 Iterations and Challenges

In this section we discuss the different stages of the environment and algorithm implementations. We discuss the motivation of each step, outline challenges, and describe how these challenges were tackled. The goal of this section is to give an insight into the thinking process, while underlining which implementation aspects proved to be particularly important for testing of the SafeSAC algorithm.

After working on the formal part of the project, the first task was to get acquainted with the code, and get it to run properly. The coding environment had to be set up correctly, and the existing code needed some minor corrections to run. Going through the various files of this repository and trying to understand how each file contributed to the whole framework took quite some time, but was a very valuable and insightful experience.

Unsuccessful Rendering During the early stages of development, persistent issues with enabling the rendering of video footage of the agents behavior were encountered. The code

would not compile, due to errors about not being able to initialize a headless EGL display. This issue would persist, even when disabling rendering in the configuration file. EGL (Embedded Graphics Library) is an interface that helps the used graphics rendering technology called OpenGL communicate effectively with the basic functions of the computer's operating system. MuJoCo (Multi-Joint dynamics with Contact) is the physics engine that is used by OpenGL, to render graphics of the RL environment, and interfaces with the operating system through EGL. Most importantly, EGL requires access to a GPU to run properly. As it turned out, the errors were happening, because the code was being tested in debugging mode, where the debugger did not have access to a GPU. Only at a later point did it become clear that this issue would not persist, when running the code on one of the provided GPU's. Still, it was necessary, to find a way to be able to use the debugger. Thus MuJoCo's OpenGL environment variable `MUJOCO_GL` was changed from `"egl"` to `"osmesa"`. OSMesa is an implementation of the OpenGL API designed specifically for off-screen rendering without relying on physical GPU hardware. The adjustment enabled the continued use of debugging in the coding environment despite its GPU limitations. This adaptation underscores the importance of environment variables in tailoring software configurations to specific hardware capabilities and debugging requirements.

Implementing `dm_control` Environments After running SAC on the `pendulum` environment with promising results, the next objective was to extend its application to more challenging environments such as `dm_control_balance` and `dm_control_swingup`. This aimed to evaluate SafeSAC's performance in more diverse scenarios. A significant hurdle was that the current codebase was designed exclusively for `Gymnasium` environments and not compatible with `dm_control`. Therefore, each `dm_control` environment needed conversion into the `Gymnasium` format. Fortunately, using an existing code snippet for this conversion helped with the process, although it required adjustments to integrate seamlessly with the existing code. Additionally, a custom cost was implemented using a wrapper. This cost would give the value 1, if the cart moves outside of a certain interval on the movement line, and 0 otherwise. SAC demonstrated robust performance across both new environments.

First Implementation of SafeSAC With the initial environments successfully running, the next objective was to implement a simplified version of SafeSAC. In this initial iteration, no trained classifier was used to assess the safety of a given state. Instead, a uniform classification of states was assumed, with a constant probability value of classification. This approach eliminated the need to integrate the logic for initializing and training a classifier, thereby simplifying the implementation, and *not* taking costs of the environment into account. Notice that this version of SafeSAC is equivalent to SAC, since all safety-related terms in the objective *eq.* (2.12) are constant and do not affect the optimization, effectively reducing it to the SAC objective. This simplification allowed the focus to be on implementing the safety critic first.

The safety critic was designed to mirror the idea of a critic, though bootstrapping a different quantity, see *eq.* (2.17). It was implemented in a straightforward manner, without addressing maximization bias, polyak averaging, or using a discount factor, as is done for the critic.

Still, this required several adaptations in the SafeSAC workflow, including changes to logging, instantiating trained networks, introducing new update logic for the safety critic, and modifying the actor's loss computation to incorporate the safety critic's values.

A detailed implementation logic can be found in Section 2.1 and the code repository As [2023]. Initial runs in the environments revealed that this simplistic implementation of the safety critic was inadequate, as SafeSAC significantly underperformed compared to SAC. This highlighted the need for further improvements to the implementation.

Improving the SafeSAC implementation A natural first step in trying to improve the performance of this simplified version of SafeSAC, was to keep the safety critic's logic as close as possible to the logic of the critic. This was done by addressing the maximization bias by training two safety critics in parallel and choosing their minimum value for inference, performing polyak averaging of the safety critic, and lastly implementing the discount factor from the expected cumulative cost. As expected, these changes significantly improved the performance of the simplified SafeSAC. However, these enhancements were implemented iteratively throughout the project, as it was initially unclear which aspects of the implementation were causing issues.

Considerable time was spent reviewing the initial safety critic implementation to identify potential errors. Various aspects were examined, including the correct formulation of the actor's update policy, or the order of gradient updates for the different trained networks. Ultimately, the safety critic was adapted as described, leading to better performance of SafeSAC, matching SAC in our environments. This is apparent from Figure 1. We suspect that this drastic improvement in performance was mostly due to the polyak averaging, and the discount factor.

Non-Uniform Classifier With a working implementation of SafeSAC with uniform classifier, the next step was to incorporate a more complex, non-trivial classifier into the algorithm. A fully connected MLP architecture was chosen for the classifier, matching the model type of the other trained networks in the code. Therefore, a classifier class was defined, and a dedicated function was created for instantiating the classifier. Additionally, the loss function and corresponding gradient updates were added to the training logic of the code, and the safety critic's loss computation was adapted to utilize the classifier's predictions.

The biggest challenge in implementing SafeSAC was determining the optimal training method for the classifier. Initially, the classifier was naively trained by performing gradient updates for every batch on which the other networks, such as the actor network, were trained. However, given that the classifier was solving a fundamentally different problem from the other networks, i.e. supervised learning, it was likely that a different updating strategy might be necessary. Unfortunately, this naive updating method did not yield satisfactory results. Both objective scores and rendered videos from the training process indicated that SafeSAC was not learning anything meaningful. This became even more apparent in the `point_to_goal` environment, as will be discussed later.

Training the classifier proved to be very challenging. Even by the end of this project, an

Figure 1: Comparison of SafeSAC with a uniform classifier and SAC on the `dm_control_swingup` environment shows both algorithms achieving significant improvements in the training objective. It's important to note that neither algorithm considers costs in this environment.
The x-axis represents the number of training steps. `train/cost_rate` indicates the average cost over the last 50,000 steps, while `train/objective` denotes the training objective score that the agent aims to maximize.
Note: This plot was generated *after* identifying incorrectly implemented costs in this environment.

effective training method for the classifier that enabled SafeSAC to achieve satisfactory results had not been found. Nonetheless, the following paragraphs will describe the attempts at solving the issue.

Falsely Implemented Costs in `dm_cartpole_swingup` Upon examining the logged training metrics, it was surprising to see the safety critic's loss increasing. Additionally, a preexisting metric tracker indicating the average cost of a trajectory consistently showed a value of 0. Initially, this metric was dismissed, assuming it might not be applicable to our custom environment implementation. This issue arose in the `dm_cartpole_swingup` environment, where a custom cost had been implemented. Suspecting an error in logging or the environment implementation, the first step was to confirm that logging was working correctly. It then became apparent that the cost wrapper contained a minor typo, causing the environment to always incur a cost of 0 and thus declare every state as safe. Although this was a crucial mistake which needed to be corrected for SafeSAC to work properly, resolving it did not solve the broader issue of SafeSAC failing to learn the objective correctly. In retrospect, more critically questioning all logged metrics could have prevented this problem much earlier.

Implementing `point_to_goal` At this point of the implementation process, SafeSAC was not functioning properly in environments with costs. To better understand the underlying issue, the environment `point_to_goal` was introduced where the optimal policy was not safe. Unlike the `dm_cartpole_swingup` environment, which had a safe optimal policy, this new environment would highlight the behavioral differences between SafeSAC and SAC more

clearly. Ideally, SafeSAC should prioritize safety more visibly than SAC, providing insights into why SafeSAC was not previously learning safety properly.

In practice, SAC learned to maximize the objective very well but completely ignored safety. In the video renderings it was apparent, that the agent would recklessly navigate through obstacles to reach the goal. The expectation was that SafeSAC would learn to avoid obstacles. Further testing in this environment revealed that the safety discount factor significantly influenced SafeSAC's behavior. Lower discount factors made SafeSAC overly cautious, preventing effective learning, while higher discount factors made it disregard safety altogether, behaving similarly to SAC. This behavior can be observed in Figure 2.

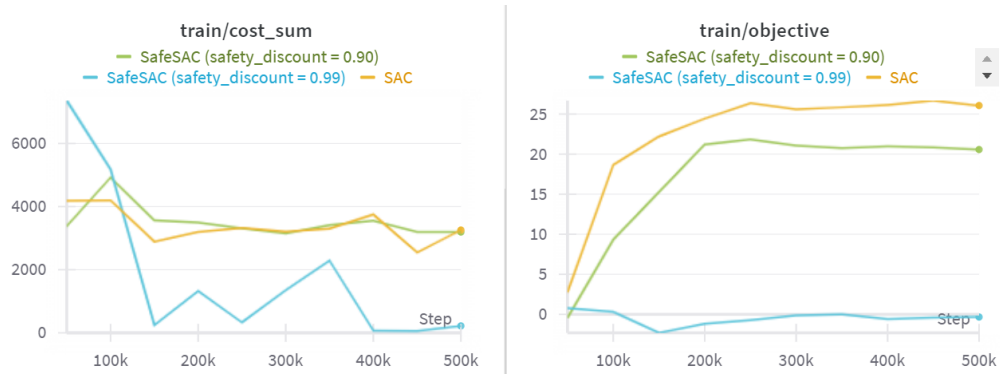

Figure 2: Comparison of SAC and SafeSAC when the safety discount factor is changed in the `point_to_goal` environment.

This suggested that the classifier was struggling to differentiate between safe and unsafe states, hindering the agent's ability to explore properly. The agent either ignored safety (like SAC) or became too cautious to move.

Training an Isolated Classifier on Offline Collected Data The current theory is, that the reason for SafeSAC not achieving good performance, while trying to stay safe, is that the classifier is not learning how to distinguish between safe and unsafe states. To investigate this, the classifier was trained in isolation using trajectories from the `point_to_goal` environment, sampled from a randomized policy.

Initially, the isolated classifier performed well on its test dataset, which shifted the focus to other potential reasons for SafeSAC's poor performance.At one point, it was even considered to implement the Q and V functions of SafeSAC directly, without going through intermediate Q', V' and γ as described in *eq.* (2.15). However, theoretically, this approach should not show different results from the existing implementation.

Eventually, it was discovered that the isolated training implementation had a flaw: the collected data was shuffled before performing the train/test split, corrupting the test data due to high correlations between transitions from the same trajectory. After resolving this corruption issue, training the classifier to distinguish between safe and unsafe states proved to be exceedingly difficult. Unfortunately, a successful training method for the classifier to accurately classify safety in isolation was not identified within the project's scope.

It is important to note that the trajectories from a randomized policy exhibited an extreme class imbalance, with unsafe states comprising only about 2% of the samples in the dataset. To address this issue, the classifier was trained using a modified loss function that more heavily weighted unsafe states, and at later problem solving iterations even disregarded batches containing only safe states to counter the bias towards always predicting safety. Various approaches were tested, including different MLP architectures, learning rates, and objectives, but none enabled the model to reliably differentiate between safe and unsafe states. The classifier kept showing results such as the one in Figure 3.

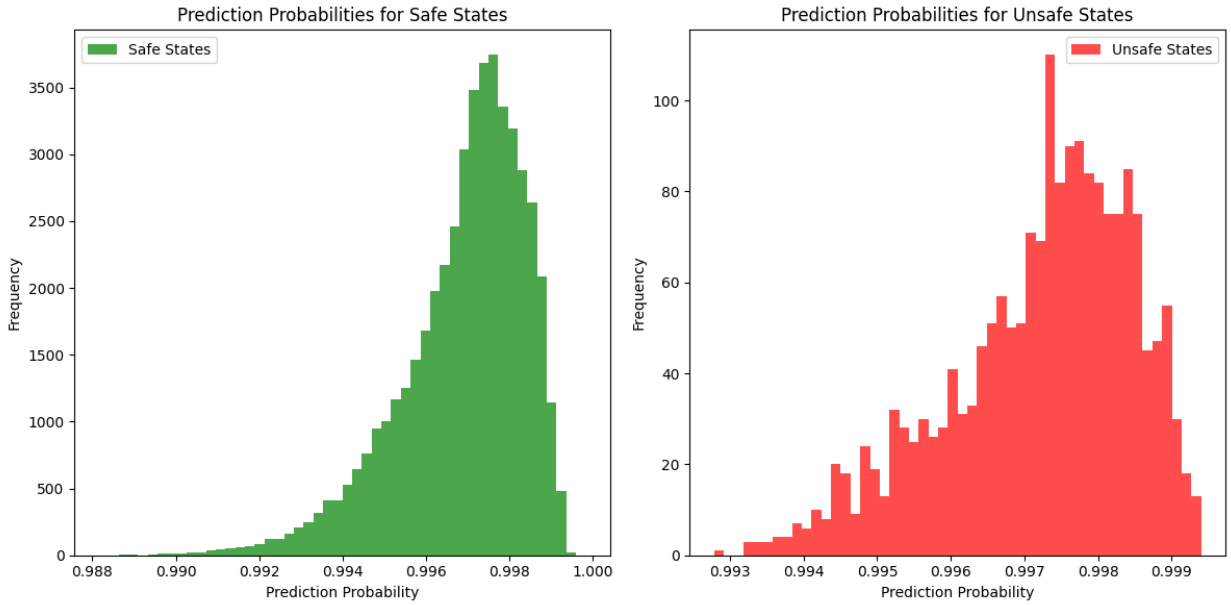

Figure 3: An overview of the offline classifier's predictions of safe and unsafe states on a test set generated from a randomized policy in the `point_to_goal` environment reveals an inability to distinguish between safe and unsafe states. This MLP classifier (2 hidden layers with 64 nodes each) was trained on approximately 200k transitions.

Furthermore, the collected trajectories used for training the classifier might not be representative of those encountered when training SafeSAC. The classifier's predictions influence future trajectories, potentially introducing bias and negatively impacting SafeSAC's performance. For instance, if the classifier is highly accurate, the trajectories will contain fewer unsafe states, possibly leading the agent to become overly confident about safety. This is a critical consideration for future work with SafeSAC.

Iteratively Re-Train Classifier Online Given that the data used to train the classifier differs in the online setting (i.e., while training SafeSAC), it might be beneficial to test classifier solutions in the online setting as well. Attempts to train an offline classifier and replace the online classifier with it did not yield good results. Different classifier architectures were also tested, but they did not lead to improvements.

The next step would have been to iteratively retrain the classifier from scratch. Specifically,

after a certain number of steps (e.g., every 1000 steps), the classifier would be retrained on all the data available up to that point. Unfortunately, there was not enough time to implement this approach within the project's scope.

4 Discussion

In this chapter, we describe the primary challenges encountered throughout the project and their impact on the work. We discuss to what extent the initial goals for the project were achieved and the lessons learned from the experience. Lastly, we explore possible future steps to further investigate the functionality of SafeSAC.

4.1 Main Challenges and Lessons

Several challenges emerged during the project, each contributing to the complexity and the learning process:

Familiarity with the Research Area Initially, my limited familiarity with the research area posed a challenge. It took time to become acquainted with relevant research papers and understand common practices in the domain of Safe RL. This foundational work was necessary to grasp the theoretical and practical aspects of SafeSAC.

Programming Experience with Complex ML Algorithms The lack of programming experience with complex Machine Learning (ML) algorithms was another challenge. Although it did not prevent me from progressing, I was not used to the typical problems encountered in debugging of such applications. However, this experience was invaluable as it allowed me to learn a much about implementing state-of-the-art ML algorithms, such as handling complex codebases, logging, parallel computation, or neatly structuring complex algorithms.

Classifier Implementation for SafeSAC The most significant implementation challenge was getting the classifier for SafeSAC to function correctly. Training the classifier to distinguish between safe and unsafe states turned out to be the most difficult part of the implementation and remained unresolved.

Best Practices in Implementation A key lesson learned was the value in implementing solutions as thoroughly as possible from the start. Simplifications should be avoided if there is no good reason to employ them. Ensuring that the general idea of the implementation is sound before testing is crucial. At every step, it is vital to confirm that the implementation behaves entirely as expected before proceeding. Logging as many metrics as possible and understanding each behavior in the logs is essential. If something looks odd, it might not cause an immediate problem, but it will likely become an issue later.

4.2 Future Work

Future work could focus on several key areas to further explore and improve SafeSAC:

Resolving the Classifier Training Issue The main focus should be on resolving the training issue with the classifier. Given that the classifier's inability to distinguish between safe and unsafe states is a significant hurdle, experimenting with different training strategies and architectures in an online setting may yield better results.

Assessing Environment Complexity It is possible that the current environment is too difficult for the SafeSAC algorithm to perform satisfyingly on. Testing SafeSAC in a variety of environments with varying levels of complexity could provide insights into its limitations and capabilities.

Comparative Evaluation In environments where SafeSAC performs well, it should be compared against common baselines such as Constrained Policy Optimization (CPO). This comparative evaluation would help in understanding the relative strengths and weaknesses of SafeSAC.

Modifications to SafeSAC Reflecting on the freely chosen distributions of the hidden safety and optimality variables in the Hidden Markov Model (HMM) during the derivation of the SafeSAC objective *eq.* (2.5), invites the thought that an alternative choice of distribution might yield better results in practice. Opting for a distribution that does not require training a classifier could potentially avoid the challenges encountered in classifier implementation.

By addressing these areas, future work can build on the current project to enhance the understanding and functionality of SafeSAC, potentially leading to more robust and reliable Safe RL algorithms.

5 Conclusion

This project was a deep dive into the domain of Reinforcement Learning (RL) with a specific focus on entropy regularization. We introduced a novel algorithm, SafeSAC, based on the Soft Actor Critic (SAC) algorithm, but adapted for the Safe RL domain. The central idea of SafeSAC is to approach the safe RL problem as a probabilistic inference problem.

While the conceptual framework of SafeSAC is compelling, the practical implementation proved to be highly complex. The project involved numerous steps, each presenting unique challenges and learning opportunities. Throughout this process, I gained valuable experience in handling complex RL algorithms. This experience has not only expanded my knowledge in the Safe RL domain but also boosted my confidence in handling applied machine learning projects in the future.

A significant challenge was the training of the classifier, which is crucial for distinguishing between safe and unsafe states. Despite extensive efforts, this key challenge remained unresolved.

However, the work provides valuable insights into the workflow of such tasks, highlighting how to approach and tackle the inherent difficulties. This report not only details the challenges encountered but also offers potential solutions and methodologies that could guide future implementations of SafeSAC. It serves as a detailed description of how to tackle some of the possible challenges that arise when implementing the algorithm. This might prove helpful for future examinations of the algorithm.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. *CoRR*, abs/1705.10528, 2017. URL http://arxiv.org/abs/1705.10528.

Yarden As. Repository: Safe control as inference, 2023. URL https://github.com/yardenas/safe-control-as-inference.

Yarden As. Repository: Safe adaptation gym, 2024. URL https://github.com/lasgroup/safe-adaptation-gym.

Farama-Foundation. Repository: Gymnasium, 2024. URL https://github.com/Farama-Foundation/Gymnasium.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Jonas Hübotter. Repository: Lecture notes of eth zurich course "probabilistic artificial intelligence", 2023. URL https://gitlab.inf.ethz.ch/OU-KRAUSE/pai-script.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Zuxin Liu, Zhepeng Cen, Vladislav Isenbaev, Wei Liu, Zhiwei Steven Wu, Bo Li, and Ding Zhao. Constrained variational policy optimization for safe reinforcement learning. *CoRR*, abs/2201.11927, 2022. URL https://arxiv.org/abs/2201.11927.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

- (●) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].
- ( ) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].
- ( ) I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.

Title of paper or thesis:

Probabilistic Inference for Safe Reinforcement Learning

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**
Lahmann	Pablo

With my signature I confirm the following:

- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	**Signature(s)**
Zurich, 14.07.2024	

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard