Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bachelor's Degree in Computer Science

# Distributional Safety Critic for Stochastic Latent Actor-Critic

**Thiago Silva Miranda**

JUIZ DE FORA
JULY, 2023

# Distributional Safety Critic for Stochastic Latent Actor-Critic

Thiago Silva Miranda

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bachelor's Degree in Computer Science

Supervisor: Heder Soares Bernardino

JUIZ DE FORA

JULY, 2023

# Distributional Safety Critic for Stochastic Latent Actor-Critic

Thiago Silva Miranda

MONOGRAPH SUBMITTED TO THE FACULTY OF THE INSTITUTO DE CIÊNCIAS EXATAS OF THE UNIVERSIDADE FEDERAL DE JUIZ DE FORA, AS AN INTEGRAL PART OF THE REQUIREMENTS NECESSARY TO OBTAIN THE BACHELOR'S DEGREE IN COMPUTER SCIENCE.

Approved by:

Heder Soares Bernardino
PhD in Computational Modeling

Saulo Moraes Villela
PhD in Systems and Computing Engineering

Thiago Dias Simão
PhD in Computer Science

JUIZ DE FORA

2023, JULY 14

*To my girlfriend, friends and family.*

# Abstract

When employing reinforcement learning techniques in real-world applications, it is often desirable to constraint the agent, such that it does not perform actions that would lead to potential damage, harm or unwanted scenarios in general. In order to specify and enforce these constraints, current state-of-the-art safe reinforcement learning algorithms rely on the constrained Markov decision process framework which makes use of a cost function to inform the agent about how unsafe each transition is. Particularly, recent approaches focus on developing safe behavior under conditions where the full observability assumption is relaxed and, instead of having access to the true state of the environment, the agent receives observations with incomplete information. In this vain, we develop a method that combines distributional reinforcement learning techniques with methods used to facilitate learning in partially observable environments. Our approach, called distributional safe stochastic latent actor-critic (DS-SLAC), uses an implicit quantile network as safety critic and learns based on a stochastic latent representation of the environment. We evaluate the DS-SLAC performance on four Safety-Gym tasks. Ultimately, DS-SLAC obtained results better than those reached by sate-of-the-art algorithms in two of the evaluated environments, while being able to develop a safe policy in three of them. Lastly, we also identify the main challenges of performing distributional reinforcement learning in the safety constrained partially observable setting.

**Keywords:** Reinforcement Learning, Safety, Distributional RL.

# Resumo

Ao empregar técnicas de aprendizado por reforço em aplicações do mundo real, muitas vezes é desejável restringir o agente, de modo que ele não execute ações que levariam a possíveis danos, perigos ou cenários indesejados de forma geral. A fim de especificar e impor essas restrições, os atuais algoritmos de aprendizado por reforço seguro pertencentes ao estado da arte baseiam-se no framework chamado de processo de decisão de Markov restrito, que faz uso de uma função de custo para informar o agente sobre o quão insegura cada transição é. Particularmente, abordagens recentes focam no desenvolvimento de comportamento seguro sob condições onde a suposição de plena observabilidade é relaxada e, ao invés de ter acesso ao verdadeiro estado do ambiente, o agente recebe observações com informações incompletas. Nesse sentido, desenvolvemos um método que combina técnicas de aprendizado por reforço distributivo com métodos usados para facilitar o aprendizado em ambientes parcialmente observáveis. Nossa abordagem, chamada de *distributional safe stochastic latent actor-critic* (DS-SLAC), usa uma *implicit quantile network* (IQN) como crítico de segurança e aprende com base em uma representação estocástica latente do ambiente. A performance do método proposto DS-SLAC é avaliada em quatro tarefas do *Safety-Gym*. Em última análise, o DS-SLAC consegue resultados superiores aos alcançados por algoritmos estado da arte em dois dos ambientes avaliados, ao mesmo tempo em que é capaz de desenvolver uma política segura para três deles. Por fim, também identificamos os principais desafios de realizar o aprendizado por reforço distributivo no ambiente parcialmente observável com restrição de segurança.

**Palavras-chave:** Aprendizado por reforço, aprendizado por reforço profundo, segurança, aprendizado por reforço distribucional.

# Acknowledgments

To my family for their unconditional support.

To professor Heder for his advice, friendship and outstanding work ethic.

To my friends for inspiring me to go beyond my expectations.

To Thiago Simão and professor Nils Jansen who welcomed me wholeheartedly during my stay in the Netherlands, without which this work would not have been possible.

*\We all wear such intellectual blinders and make such inexplicable blunders that it is amazing that any progress is made at all.".*

*Richard Bellman*

# Contents

**List of Figures** **7**

**List of Tables** **8**

**List of Abbreviations** **9**

**1 Introduction** **10**

**2 Background** **13**
2.1 Reinforcement Learning . . . . . 13
2.1.1 Markov Decision Process . . . . . 14
2.1.2 Dynamic Programming . . . . . 15
2.1.3 Tabular Reinforcement Learning . . . . . 17
2.1.4 Function Approximation . . . . . 19
2.2 Deep Reinforcement Learning . . . . . 21
2.2.1 Neural Networks . . . . . 21
2.2.2 Soft Actor-Critic . . . . . 23
2.3 Distributional Reinforcement Learning . . . . . 25
2.3.1 Implicit Quantile Network . . . . . 26
2.4 Constrained Markov Decision Process . . . . . 27

**3 Related Work** **29**
3.1 Fully Observable Safe Reinforcement Learning . . . . . 30
3.2 Partially Observable Safe Reinforcement Learning . . . . . 32

**4 Distributional Safe Stochastic Latent Actor-Critic** **37**

**5 Computational Experiments** **42**
5.1 Implementation Details . . . . . 45

**6 Conclusion** **47**

**Bibliography** **49**

# List of Figures

2.1 The perception-action-learning loop. Adapted from (SUTTON; BARTO, 2018). . . . . 14
2.2 Example of a standard feedfoward deep neural network. . . . . 22

3.1 Different Safety-Gym environments. In "Goal" tasks, the agent has to reach the goal (green cilinder) while trying to avoid hazards (blue circles and blue cube) . . . . . 29
3.2 Modified observations received by the agent at each Safety-Gym environment 33

5.1 Learning curves for DS-SLAC in four different Safety-Gym environments. 43
5.2 Normalized reward return and cost return. . . . . 44

# List of Tables

5.1 Normalized final results with each cell containing a tuple $(\bar{J}_r(\pi), \bar{J}_c(\pi))$. . . 44
5.2 Hyperparemeters for DS-SLAC. . . . . 46

# List of Abbreviations

| | |
|---|---|
| RL | Reinforcement learning |
| DL | Deep learning |
| NN | Neural network |
| DNN | Deep neural network |
| DP | Dynamic programming |
| TD | Temporal difference |
| Safe RL | Safe reinforcement learning |
| MDP | Markov decision process |
| CMDP | Constrained Markov decision process |
| POMDP | Partially Observable Markov decision process |
| CPOMDP | Constrained partially observable Markvo decision process |
| SAC | Soft actor-critc |
| SLAC | Stochastic latent actor-critc |
| TRPO | Trust region policy optimization |
| PPO | Proximal policy optimization |
| IQN | Implicit Quantile Network |
| CPO | Constrained policy optimization |
| WCSAC | Worst case soft actor-critc |
| LAMBDA | Lagrangian model-based agent |
| Safe SLAC | Safe stochastic latent actor-critc |
| DS-SLAC | Distributional safe stochastic latent actor-critc |

# 1 Introduction

During recent years, the accelerated growth in the field of machine learning has become particularly evident. Specifically, the widely known Deep Neural Networks (DNNs) have been used to produce significant improvements in the state of the art in a range of different tasks. Some domains that have greatly benefited from the use of DNNs include computer vision, natural language processing, and medical applications. The great availability of data, combined with the increase in computational power, as well as the improvement of scalability techniques, has provided these function approximators with the necessary tools to generate images never seen before (RAMESH et al., 2022), understand and create texts in natural language (BROWN et al., 2020), and even help with programming (CHEN et al., 2021).

Although most current deep learning applications fall under the umbrella of methods classified as supervised learning (POUYANFAR et al., 2018), DNNs have also been used to succeed in more general tasks, which call for an agent to be able to make a series of decisions in complex environments. The methods used to tackle such problems belong to the area of Reinforcement Learning (RL) and their combination with DNNs is called Deep Reinforcement Learning (DRL) (ARULKUMARAN et al., 2017). DRL has enjoyed great success when applied to games. Notable results include achieving super human level play at games from the Atari 2600 console (MNIH et al., 2013) and beating the best Go player in the world (SILVER et al., 2016). However, as we aim to apply DRL techniques to real world scenarios, an agent's performance ceases to be our only concern. In those circumstances, how safely an agent can learn and execute his assigned tasks can be just as important as how well it performs on the tasks themselves.

For instance, a robot that is controlled by an AI must never harm a human being, and a self driving car must make every effort to avoid actions that cause harm to the vehicle, even during it's training procedure. Safe Reinforcement Learning (Safe RL) (GARCIA; FERNÁNDEZ, 2015) is the area concerned with addressing this problem. It aims to create agents which are robust enough to act in the real world without causing

harm or performing unwanted actions. This is particularly challenging, as RL agents generally learn by trial and error. These agents explore their action space in order to find the optimal action in each situation. As such, they would naturally execute undesirable actions before learning that these actions lead to poor outcomes.

Current approaches to the safe reinforcement learning problem mainly rely on Safety-Gym tasks (RAY; ACHIAM; AMODEI, 2019) to benchmark performance and use the constrained Markov decision processes (CMDP) (ALTMAN, 1999) as their formalism. In CMDPs, beyond the usual reward signal, the RL agent also receives a cost signal, which encapsulates how unsafe or undesirable a transition is. Additionally, recent methods have focused on performing safe reinforcement learning in environments with a high degree of partial observability. To this end, these methods learn from high dimensional sensory inputs as observations in a constrained partially observable Markov decision process (CPOMDP) (ISOM; MEYN; BRAATZ, 2008; LEE et al., 2018) setting.

As et al. (2022) proposed a model-based approach called LAMBDA, to learn directly from pixels. Lambda utilizes Bayesian world models to estimate optimistic upper bounds in respect to the reward signal, as well as pessimistic upper bounds in respect to the cost signal; these are, then, used to train the agent's policy. In the same vain, Hogewind et al. (2023) method also focus on pixel learning. Their Safe SLAC algorithm, on the other hand, is assisted by latent variable model that is trained to predict both the reward and cost signal. Then, it uses an inferred latent state (produced by the latent variable model) as input to the policy; in order to address the strong partial observability coming from the use of pixels as observations.

In this work, we develop a method that augments Hogewind et al. (2023) approach by using distributional RL techniques (BELLEMARE; DABNEY; ROWLAND, 2023), which are known to improve sample-efficiency and performance of RL agents (DABNEY et al., 2018b; HESSEL et al., 2018). Furthermore, the use of distributional RL also allows for risk-averse behavior (TANG; ZHANG; SALAKHUTDINOV, 2019; YANG et al., 2023). That is, for an agent to not only favor transitions that lead to a lower cost on average, but to also avoid transitions with high variance, that could eventually lead to extremely high cost, even if these high cost were to happen rarely. We call our method

distributional safe stochastic latent actor-critic (DS-SLAC). We evaluate DS-SLAC using different Safety-Gym tasks and achieve comparable performance to LAMBDA and Safe SLAC in some of the environments. Ultimately, we identify some of the challenges involved in performing distributional safe reinforcement learning under high partial observability, as well as the main obstacles to producing risk-averse behavior. Lastly, the proposed approach is capable of developing a safe policy for most of the tested environments.

# 2 Background

In this chapter, the main concepts that are necessary for understanding this work are presented. Section 2.1 discusses classic reinforcement learning, and builds the core ideas of the field, which are important for the understanding of the remaining sections. Here, (SUTTON; BARTO, 2018) is used as the main frame of reference and is not cited throughout the text to avoid repetition. Thus, the reader can assume that (SUTTON; BARTO, 2018) is the reference used for any of the concepts, unless stated otherwise. Notation also follows the book closely.

Section 2.2 is about deep reinforcement learning. It explains deep neural networks briefly, as high level understanding of the topic is already enough for the reader to comprehend this work. Furthermore, it mentions some of the main deep reinforcement learning algorithms, and mainly focus on explaining the soft actor-critic algorithm, which will be used in subsequent chapters. Section 2.3 describes some of the distributional reinforcement learning algorithms, and more heavily explains the implicit quantile network algorithm, which also is used in later chapters. Lastly, Section 2.4 discusses the constrained Markov decision process formalism.

## 2.1 Reinforcement Learning

The field of reinforcement learning took shape along two main lines. The first comes from psychology and brings concepts of learning by trial and error extracted from research on animals. The second is related to optimal control theory, which seeks to optimize the behavior of an agent in a dynamic system. Both areas provided fundamental ideas to support and allow the formation of the current theory of RL.

At its core, RL is a set of computational methods that seek to capture ideas and automate interactive learning. These methods follow a constant iteration cycle that is demarcated by the measure which is called *time step*. At each time step, an agent interacts with an environment choosing an action, based on the current state of the

environment, receiving a new state and a scalar signal that is called reward. At each step of the interaction, the agent updates its knowledge about the environment based on the new information acquired. This is the fundamental cycle present in all RL methods. It is called the *perception-action-learning loop* (ARULKUMARAN et al., 2017).

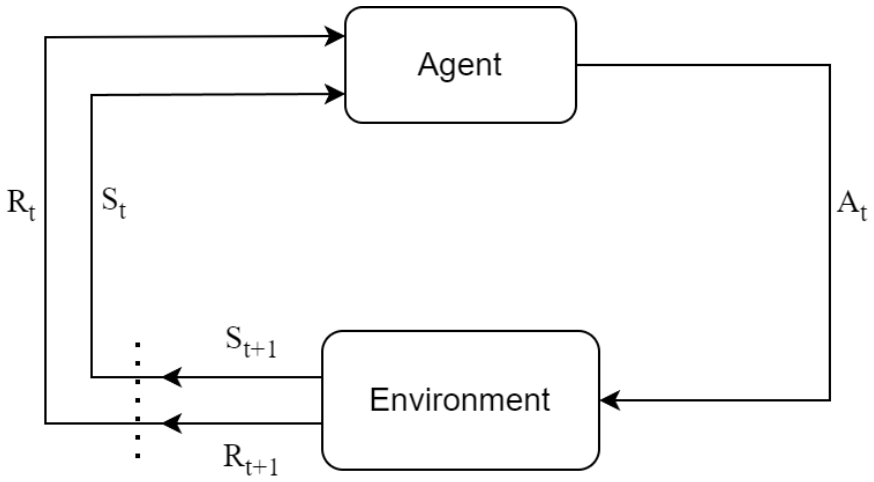



Figure 2.1: The perception-action-learning loop.
Adapted from (SUTTON; BARTO, 2018).

The agent's goal is to develop a policy, typically called $\pi$, that maximizes the value of rewards received by the environment. A policy is a mapping from states to actions, and can be deterministic or stochastic. In the first case, each state is directly mapped to a specific action. In the second each state has a probability distribution over the set of possible actions.

### 2.1.1 Markov Decision Process

The cycle of interactions presented above can be mathematically formalized through a Markov decision process (MDP), more specifically, a finite MDP. In turn, we can characterize an MDP as a tuple of four elements ($\mathsf{S}$, $\mathsf{A}$, $p$, $\gamma$), where:

- $\mathsf{S}$ is the set of all possible states.
- $\mathsf{A}$ is the set of all possible actions.
- $p(r, s' \mid s, a)$ is the joint probability of a reward $r$ and a next state $s'$ given state $s$ and action $a$.

- $\gamma \in [0, 1]$ is the discount factor.

The states of a MDP must follow the Markov property, that is, $p$ must fully characterize the dynamics of the environment. Meaning, the probabilities of state and reward transitions in a time step $t$ depend only on the state observed in the previous time step $(t-1)$. Intuitively, the state should contain all relevant and necessary information for the agent to be able to make the best decision.

Finite MDPs provide a flexible framework for encapsulating RL problems and approaching them with mathematical reasoning. However, when dealing with RL, not all features of an MDP are available. The dynamics of the environment $p$ are often unknown, states are not fully observable and may even disobey the Markov property. Even so, the theory and concepts built on the somewhat limiting assumptions of MDPs form the basis of all current RL methods. Thus, understanding it is essential to also understand the functioning of RL algorithms.

### 2.1.2 Dynamic Programming

Whenever a full model of the the transitions probabilities $p$ of the environment is available, a MDP can be solved via planning techniques. In this case, the methods employed belong to the field of Dynamic Programming (DP).

DP focuses on finding the optimal policy ($\pi^*$) given a perfect model of the environment. For this purpose, the concept of a value function $v$ is defined. The value function of a state corresponds to the expected value of the return $G_t$, where the return is the discounted sum of all rewards achieved when starting at that state:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^k R_T \tag{2.1}$$

$$v_\pi(s) = \mathsf{E}_\pi[G_t \mid S_t = s] \tag{2.2}$$

where both $R_t$ and $S_t$ are random variables.

In a similar way, one can define the concept of a value function for a state-action pair, which is commonly known as the action value function:

$$q_\pi(s,a) = \mathsf{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{2.3}$$

It is important to point out that both state and action values are conditioned on the policy $\pi$, as future rewards are decided based on the actions taken by the agent. Analogously, the concepts of optimal value functions are also defined:

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.4}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2.5}$$

these functions correspond to the value functions that are reached when the agent follows an optimal policy ($\pi^*$), i.e., a policy that results in the highest expected return for the agent.

The state and action values of a policy can be calculated using linear programming methods by solving a system of linear equations. However, these state and state-action values also have a recursive property, which allows for them to be calculated iteratively. This is expressed in the so called Bellman equations, and can be defined as:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{2.6}$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')] \tag{2.7}$$

In the same manner, the recursive property for the optimal state and state-action value functions are defined in the Bellman optimality equations:

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \tag{2.8}$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')] \tag{2.9}$$

DP algorithms make use of these Bellman equations to arrive at an optimal policy. They work as follows: an arbitrary policy $\pi$ is initialized as well as initial estimate for the

values of each state. From there, an iteration consisting of two steps begins. The first step, called *policy evaluation*, seeks to find the current state values when following the policy $\pi$; to do so, the algorithms update the current values according to the Bellman equations. The second step, called *policy improvement*, updates the policy $\pi$ so that it becomes the deterministic policy that is greedy with respect to the new calculated state values. When the policy evaluation step does not generate any change in the state values in relation to the previous iteration, it means that the optimal policy $\pi^*$ has been reached, since this policy obeys the Bellman optimality equations. This process is called *generalized policy interaction* (GPI) and guarantees asymptotic convergence for $\pi^*$.

### 2.1.3 Tabular Reinforcement Learning

Dynamic programming methods are powerful but, as mentioned earlier, they require a perfect model of the dynamics of the environment. This type of model is usually unavailable when dealing with more complex scenarios. To solve this problem, reinforcement learning uses sampling to estimate action values, allowing agents to learn through direct interaction with the environment.

There are two main ways to learn by sampling in RL. Monte Carlo methods and temporal difference (TD) learning methods. Both are used to perform policy evaluation estimating action values. From then on, the policy improvement step remains the same.

Monte Carlo methods collect the complete sequence of states, actions and rewards from various episodes. An episode consists of time steps from 1 to $T$, where $T$ is the ending time step. From these data, such methods calculate the average of the returns of each state-action pair and use this average as an estimate of the value of this pair. By the law of large numbers (HSU; ROBBINS, 1947), the average converges to the real value as the number of episodes increases.

TD learning methods, on the other hand, make use of the TD error $\delta_t$:

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t). \tag{2.10}$$

The idea is the following: it is possible to estimate the action value $Q(S_t, A_t)$,

through the use of the next action value estimate $Q(S_{t+1}, A_{t+1})$ and the received reward $R_{t+1}$. In the same way as $q_\pi(s, a)$ depends on the values of the next reward $r$ and the next action value $q_\pi(s', a')$ in Equation 2.7. Thus the temporal difference error calculates the error between the current action value estimate $Q(S_{t+1}, A_{t+1})$ and a better estimate $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$. In this way, an update is performed at each time step based on the parameter $\upsilon$, which defines the size of the update. This parameter is called *step size*. Thus, the current action value $Q(S_t, A_t)$ can be updated according to:

$$Q(S_t, A_t) = Q(S_t, A_t) + \upsilon\delta_t \tag{2.11}$$

by updating with enough samples of transitons containing data of all the possible states and all the possible actions, the action value estimates converge to true action values $q_\pi$.

A problem arises, however, when sampling methods are used to solve control problems. The agent encounters a dilemma during its process of interacting with the environment. On one hand, it can use the knowledge acquired so far to choose the actions that result in the highest reward based on its current beliefs. On the other hand, the agent can seek to improve its understanding of the environment, choosing actions that are not perceived as optimal, updating its beliefs and improving its chances of performing well. This problem is known as *exploration and exploitation tradeo* (KAELBLING; LITTMAN; MOORE, 1996).

An agent that explores too much, to get to know its environment better, never achieves a good performance by choosing sub-optimal actions. On the other hand, the agent that always chooses the action with the highest value can perform poorly if its estimates are wrong or if the environmental conditions change. One way to solve this problem is to use a policy $b$ to choose actions while learning, but at the same time learn about another policy $\pi$. Policy $b$ is the behavior policy and is constructed in a way such that it explores all possible state-action pairs. On the other hand, the devolved policy $\pi$ is only concerned in being greedy in respect to the action values and not necessarily visiting all possible states. The setting where the agent learns a policy, based on experience collected by a different policy is called the *o -policy learning* setting.

Possibly the most well-known RL algorithm is *Q-learning*. Q-learning is an off-

policy algorithm based on the TD error. It performs its value updates based on the action of the next state that has the highest estimated value. It is based on the Bellman optimallity equations from DP. Thus, Q-learning estimates the values of the optimal policy following an arbitrary policy:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \upsilon[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.12}$$

Another well-known algorithm is called SARSA. SARSA is an on-policy algorithm that learns and improves the current policy based on a sampled action of the policy itself:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \upsilon[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.13}$$

### 2.1.4 Function Approximation

Both SARSA and Q-learning perform well when working with a relatively small state space (what is known as the tabular setting). However, for many interesting applications the state space may be extremely large, to the point where it becomes unfeasible to enumerate all of the states in order to calculate their values and subsequently find the optimal policy. In this case, one forgoes calculating the exact value of each state in favor of merely approximating it. Doing so allows for the generalization of knowledge across states, resulting in a far more compact representation of the value function. In this way, RL algorithms can be extended for settings with combinatorial or even infinite state spaces.

To approximate action values, states are converted into sets of features. Then, by choosing a function $\hat{q}$, that is parametrized by a set of weights $\mathbf{w}$, one can approximate the true value function $q_\pi$ by performing stochastic gradient descent (SGD) (ROBBINS; MONRO, 1951) on the weights $\mathbf{w}$ based on the TD error and the function's gradient:

$$\mathbf{w} \leftarrow \mathbf{w} + \upsilon[R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})] \nabla \hat{q}(S_t, A_t, \mathbf{w}) \tag{2.14}$$

where $\upsilon$ is the step size.

Then, the approximated function can be used to derive a policy by choosing the

actions with the most value.

So far all RL methods presented in this section revolve around estimating action values and, then, choosing the best action according to these values. However, this can be problematic when dealing with continuous action spaces instead of discrete ones. In this case, the operation performed to find the action that maximizes the value of a given state can be complex and computationally expensive. To bypass this issue, one can use a different approach to solve RL problems: it is possible to derive a policy by directly searching the space of all possible policies, through the so-called policy gradient methods. These methods parameterise a policy $\pi$ with a set of weights $\boldsymbol{\theta}$, such that:

$$\pi(a|s, \boldsymbol{\theta}) = Pr\{A_t = a|S_t = s, \boldsymbol{\theta_t} = \boldsymbol{\theta}\} \tag{2.15}$$

Then the policy weights can be updated according to a metric of performance $J(\boldsymbol{\theta})$. Stochastic gradient ascent can be performed on the weights, such that each weight is adjusted towards the direction that maximizes the value of $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \upsilon \nabla J(\boldsymbol{\theta}) \tag{2.16}$$

When defining $J(\boldsymbol{\theta})$ to be equal to the value of the initial state, it is possible to derive an expression that is proportional to the gradient $\nabla J(\boldsymbol{\theta})$. In fact, there are many derivations that lead to slightly different updates for the policy weights. The simplest policy gradient algorithm, REINFORCE, uses the following proportional relation:

$$\nabla J(\boldsymbol{\theta}) \propto \mathsf{E}_\pi \left[ G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \right] \tag{2.17}$$

Thus, performing the update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \upsilon G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}_t) \tag{2.18}$$

## 2.2 Deep Reinforcement Learning

Deep reinforcement learning is the intersection of *Deep Learning* (DL) and RL. Combining neural networks (NN) with reinforcement learning methods is not a new idea. In fact, it was already explored decades ago, using Q-learning and a neural network to approximate its action values (RUMMERY; NIRANJAN, 1994). However, in recent years, with the advent of DL and the increase in computational power, as well as the development of techniques to improve stability of RL agents using NN as function approximators, this combination has become much more effective, being able to display some impressive feats (MNIH et al., 2015).

### 2.2.1 Neural Networks

Neural networks are formed by several units, called neurons. Each unit receives a series of input signals and outputs one singular value. The output of each neuron is calculated by multiplying each input signal with a respective weight, then summing all the results of the multiplications. Essentially producing a linear combination of the input signals. A bias term is also added to final the summation. Finally, the summed value is given as input to what is called an activation function and that produces the output of the entire neuron. Mathematically, the output of a neuron $k$ can be expressed through the following equation (HAYKIN, 2009):

$$y_k = \sigma\left(\sum_{j=1}^{m} w_{kj}x_j + b_k\right) \tag{2.19}$$

where $x_1, \ldots, x_m$ are the input signals, $w_{k1}, \ldots, w_{km}$ are the neuron weights, $b_k$ is the bias term and $\sigma$ is the activation function.

In turn, a NN is collection of multiple neurons, each one with its own set of weights. These units are organized into layers, which are arranged in sequential order. In the most common type of NN, called a feedfoward fully connected neural network (GOODFELLOW; BENGIO; COURVILLE, 2016), a neuron from layer $l$ receives as input the output of all the neurons from layer $l-1$. A neural network with several layers is called a deep neural network (DNN) (POUYANFAR et al., 2018).

The neuron operations of a fully connected NN can be expressed compactly through the use of vector notation. Let $\mathbf{h}^{(l-1)}$ be the vector containing the output of all neurons from a layer $l-1$. Then, a neuron $k$ from layer $l$ with a set of weights represented by the vector $\mathbf{w}_k$, a bias term $b_k$ and an activation function $\sigma$, computes its output $h_k^{(l)}$ according to the following equation:

$$h_k^{(l)} = \sigma(\mathbf{w}_k^T \mathbf{h}^{(l-1)} + b_k) \tag{2.20}$$

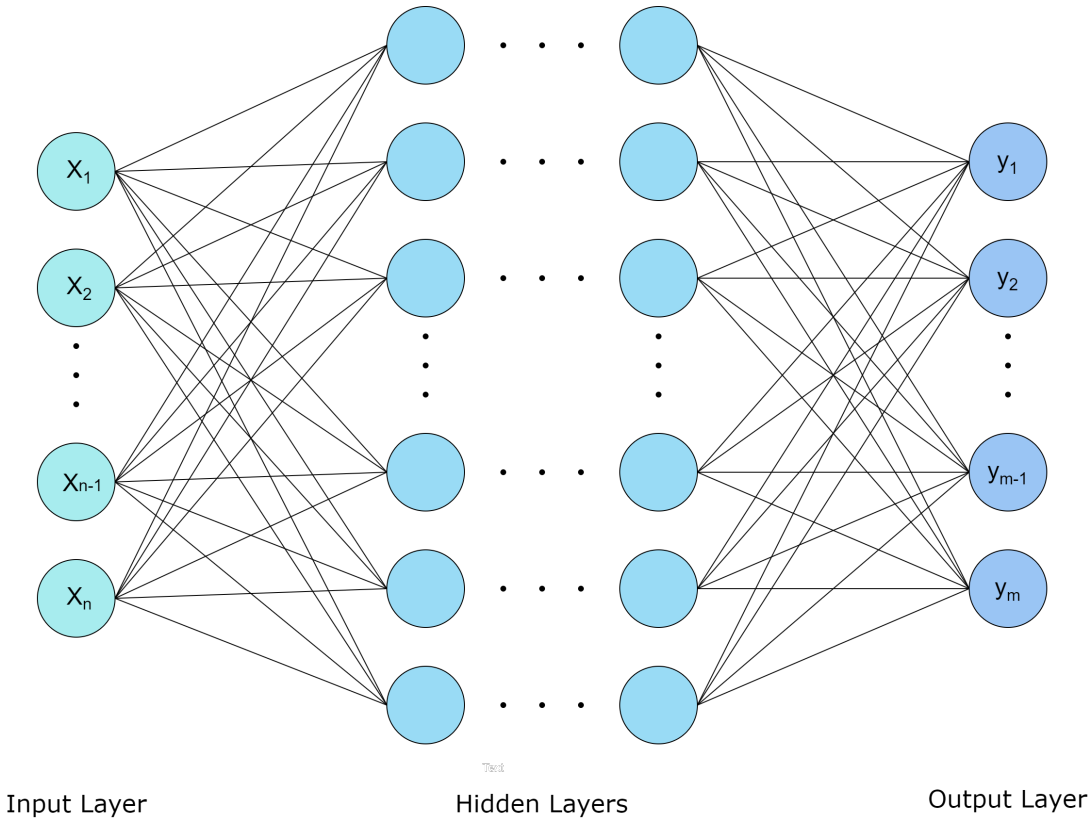



Figure 2.2: Example of a standard feedfoward deep neural network.

Given this architecture, it is possible to train a NN $f$ parametrized by $\boldsymbol{\theta}$ to approximate a function $f^*$ that maps inputs $\mathbf{x}$ to an outputs $y$. To do so, a cost function $J$ is defined. $J(\hat{y}, y)$ is a distance measure between the output of the true function $y = f^*(\mathbf{x})$ and the output of the neural network $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$. Thus the weights of the network can be updated iteratively, by performing gradient descent based on $J(\hat{y}, y)$. To this end, an algorithm called backpropagation is used (RUMELHART; HINTON; WILLIAMS, 1986).

Backpropagation allows for the computation of partial derivatives of the cost function in respect to all weights in the network, by taking advantage of the chain rule of calculus.

In RL, DNNs are used to approximate value functions and as parametrized differentiable policies, very much like it was mentioned in Section 2.1. However, many of theoretical guarantees of convergence of classic RL algorithms do not apply to deep reinforcement learning. As such, this combination is far from trivial and finding different ways to improve the stability of these algorithms has been an active topic of research ever since the popularization of deep learning techniques in the 2010s.

Today, there already exist a set of key DRL algorithms that accomplish this through different manners. (MNIH et al., 2015) and (HAUSKNECHT; STONE, 2015) are based on action value estimation, whereas (SCHULMAN et al., 2015) and (SCHULMAN et al., 2017) are some of the policy gradient methods. (HAARNOJA et al., 2018a) and (MNIH et al., 2016) are also policy gradient methods, but, additionally, they make use of action values in the policy update. In this case, the action value estimator is called the "critic", while the policy is the "actor"; consequently, these approaches are known as actor-critic methods.

### 2.2.2 Soft Actor-Critic

Soft actor-critic (SAC)(HAARNOJA et al., 2018a; HAARNOJA et al., 2018b) is an off-policy algorithm built on top of the reinforcement learning maximum entropy framework. In this setting, the agent aims to maximize the tradeoff between reward and entropy. That is, maximize the expected reward while acting as randomly as possible. This is a way to tackle the exploration and exploitation tradeoff mentioned in Section 2.1. The algorithm is off-policy, because it uses a replay buffer $\mathsf{D}$ to store agent transitions from the environment. Then, it updates both the it's action value estimate and the policy by sampling transitions from this replay buffer.

Given a random variable $X$ distributed according to a probability mass or density function $P$, the entropy of $P$ can be calculate as:

$$H(P) = \mathsf{E}\left[-\log P(X)\right] \tag{2.21}$$

Therefore, in the maximum entropy framework, the agent receives an additional bonus to the reward at each time step, that is proportional to the entropy of the policy. By taking this into account, a new version of the action value Bellman equation can be derived:

$$q_\pi(s,a) = \mathsf{E}_\pi \left[ R + \gamma(q_\pi(s',a') - \alpha \log \pi(a'|s')) \right], \tag{2.22}$$

where $\alpha$ is a coefficient that regulates the trade-off between entropy and reward. It is possible, then, to sample in order to approximate the true value $q_\pi$. Thus, the SAC critic can be trained with the following loss function:

$$J_Q(\theta) = \mathsf{E}_{s,a,r,s' \sim \mathsf{D}} \left[ (Q_\theta(s,a) - \hat{Q}(s,a))^2 \right] \tag{2.23}$$

with:

$$\hat{Q}(s,a) = r + \gamma(Q_\theta(s',a') - \alpha \log \pi(a'|s')), \quad a' \sim \pi(\cdot|s') \tag{2.24}$$

where $Q_\theta$ represents the neural network used to approximate the true action values and is parameterized by weights $\theta$.

Likewise, the actor's loss function can be defined as:

$$J_\pi(\phi) = \mathsf{E}_{s \sim \mathsf{D}, a \sim \pi_\phi} \left[ \alpha \log \pi_\phi(a|s) - Q_\phi(s,a) \right] \tag{2.25}$$

It is still not possible to update policy $\pi$ based on the equation above. This occurs because both $a$ and $\pi$ are dependent on the parameters $\phi$. To solve this problem it is possible to apply the reparameterization trick (KINGMA; WELLING, 2013) to rewrite the equation into an expectation over noise, which allows us to estimate the gradient of $J_\pi$ by sampling:

$$J_\pi(\phi) = \mathsf{E}_{s \sim \mathsf{D}, \epsilon \sim \mathsf{N}} \left[ \alpha \log \pi_\phi(f_\phi(\epsilon; s)|s) - Q_\theta(s, f_\phi(\epsilon; s)) \right] \tag{2.26}$$

Lastly, it is important to note that SAC also applies other tricks to improve stability, such as double Q-learning networks (FUJIMOTO; HOOF; MEGER, 2018) and the use of target networks (MNIH et al., 2015). These techniques were omitted in the

equations above for simplicity sake, but they play a important role with respect to the good performance achieved by the overall method.

## 2.3 Distributional Reinforcement Learning

As seen previously, classic reinforcement learning is concerned with creating a policy that is able to maximize the expected return. Thus, its only natural to ponder about what would happen if, instead of taking only the expect return into account, the full return distribution was considered. Intuitively, the distribution contains far more information than the singular scalar value. In practice, its been shown that the use of distributional reinforcement learning increases both sample efficiency and performance of RL agents (HESSEL et al., 2018).

To extend RL to the distributional case, a distributional version of the Bellman equation can be defined:

$$Z(s,a) \overset{\mathsf{D}}{=} R(s,a) + \gamma Z(S',A') \tag{2.27}$$

where $\mathsf{D}$ denotes equality between probability laws (BELLEMARE; DABNEY; MUNOS, 2017). In Equation 2.27, $Z(s,a)$ is the distribution of possible returns when action $a$ is chosen in state $s$. This distribution depends on the random variable of the reward $R$ and return distribution $Z(S',A')$, where $S'$ and $A'$ are the random variables of the next state and the next action, respectively.

Equation 2.27 can be used to estimate the return distribution. However, computational challenges are presented when trying to perform distributional reinforcement learning in the first place. As the distribution of returns can be shaped in quite a complex way, its necessary to resort to approximation in order to represent it with a finite number of parameters. Additionally, multiple metrics can be used as proxy for the distance between two distributions.

How the distributions are parameterized is often a key distinction between algorithms. For instance, C51 (BELLEMARE; DABNEY; MUNOS, 2017) used a discrete distribution parameterized by $N \in \mathsf{N}$ and $V_{min}, V_{max} \in \mathsf{R}$, where the support is a set of atoms $\{z_i = V_{min} + i\Delta z : 0 \leq i < N\}, \Delta z = \frac{V_{max}-V_{min}}{N-1}$. Meanwhile, both QR-DQN

(DABNEY et al., 2018b) and IQN (DABNEY et al., 2018a) approach the problem by approximating the quantile function of the return distribution, the inverse of the cumulative distribution function (c.d.f.).

### 2.3.1 Implicit Quantile Network

Implicit quantile network (IQN) is a distributional reinforcement learning algorithm based on deep Q-learning. It trains an implicit parametric function to reparameterize samples from a base distribution, usually the uniform distribution $U([0,1])$, to the quantile values of a target distribution, in this case $Z(s,a)$. In this way, by approximating the quantile function with $F_Z^{-1}(\tau)$, sampling $\tau \sim U([0,1])$ and applying $F_Z^{-1}$ would equate to sampling the original return distribution: $F_Z^{-1}(\tau)(s,a) \sim Z(s,a)$.

To do so, the p-Wasserstein distance is employed to measure the difference between two distributions U and V:

$$W_p(U,V) = \left( \int_0^1 |F_U^{-1}(\omega) - F_V^{-1}(\omega)|^p d\omega \right)^{\frac{1}{p}} \tag{2.28}$$

Where $F$ is the c.d.f.

In IQN's case, the Wasserstein distance is approximated by the Huber quantile regression loss (HUBER, 1992), with a threshold $\kappa$:

$$\rho_\tau^\kappa = |\tau - \mathsf{I}\{\delta_{i,j} < 0\}| \frac{\mathsf{L}(\delta_{i,j})}{\kappa}, \quad \text{with} \tag{2.29}$$

$$\mathsf{L}(\delta_{i,j}) = \begin{cases} \frac{1}{2}\delta_{i,j}^2 & \text{if } |\delta_{i,j}| < \kappa \\ \kappa(|\delta_{i,j}| - \frac{1}{2}\kappa) & \text{otherwise} \end{cases} \tag{2.30}$$

From the distributional Bellman equation, it is possible to derive the sampled temporal difference error at time step $t$, for two i.i.d. samples $\tau, \tau' \sim U([0,1])$ and a policy $\pi$:

$$\delta_t^{\tau,\tau'} = r_t + \gamma F_Z^{-1}(\tau')(s_{t+1}, \pi(s_{t+1})) - F_Z^{-1}(\tau)(s_t, a_t) \tag{2.31}$$

Thus, IQN trains its action-state pair return distribution function with the following loss function:

$$J(s_t, a_t, r_t, s_{t+1}, a_{t+1}) = \frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_{\tau_i}^{\kappa}(\delta_t^{\tau_i, \tau'_j}) \tag{2.32}$$

where $N$ and $N'$ correspond to the number of samples $\tau, \tau' \sim U([0, 1])$ used to estimate the loss.

## 2.4 Constrained Markov Decision Process

The reward hypothesis posits the following: "That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)." (SUTTON; BARTO, 2018). Although, it seems like a particularly bold assumption, it is very a important statement to reinforcement learning in general. In case the validity of the hypothesis was to be proven, it would signify that most, if not all, tasks could be formulated as a reinforcement learning problem, at least theoretically. In practice, however, RL practitioners have found difficulties in precisely expressing desired complex behavior through the use of a reward function, with reward shaping becoming an area of research of its own (NG; HARADA; RUSSELL, 1999).

In the case of safe reinforcement learning problems, state-of-the-art algorithms avoid dealing with the difficulties of shaping the reward function to express safety constraints. Instead, these algorithms rely on the constrained Markov decision process (CMDP) formalism. This allows for a clear separation between the task objective that the agent is trying to optimize and the safety violations that the agent should aim to avoid. The framework also provides an easy way to regulate the tradeoff between these two, often conflicting, goals. We describe the mathematical details as follows.

In a CMDP, at every time step $t$, in addition to the reward signal $r_t$, the agent also receives a vector containing $K$ different cost signals $c_t^k$. In this setting, the agent aims to find the optimal policy $\pi^*$, i.e. the policy that maximizes some metric $J_r(\pi)$. Where $J_r(\pi)$ is some function that depends on the reward induced from following policy

$\pi$. However, unlike in MDPs, the set of possible policies $\Pi$ is reduced to $\Pi_C$, such that only feasible policies are considered, meaning policies that satisfy a set of constraints. Thus, the CMDP objective is the following:

$$\pi^* = arg \max_{\pi \in \Pi_C} J_r(\pi) \tag{2.33}$$

And, the set of constraint-satisfying policies can be defined as:

$$\Pi_C = \{\pi : J_{c_i}(\pi) \leq d_i, \quad i = 1, .., k\} \tag{2.34}$$

where $J_{c_i}$ is some function that depends on the cost $c_i$ induced from following policy $\pi$, and $d_i$ is a hyperparameter that specifies a desired upper limit for $J_{c_i}(\pi)$.

Often, when dealing with safe reinforcement learning, a single cost function $c$ is used to express unsafety and the hyperparameter $d$ is referred to as the budget. Additionally, $J_r(\pi)$ is defined to be the expected return (from Equation 2.2) achieved by following policy $\pi$ and starting from the initial state $s_0$:

$$J_r(\pi) = v_\pi^r(s_0) \tag{2.35}$$

and $J_c(\pi)$ is defined as the expected cost return from the initial state $s_0$:

$$J_c(\pi) = v_\pi^c(s_0) \tag{2.36}$$

where the expected cost return $v_\pi^c$ is defined in an analogous manner to $v_\pi^r$:

$$G_t^c = C_{t+1} + \gamma_c C_{t+2} + \gamma_c^2 C_{t+3} + ... + \gamma_c^k R_T \tag{2.37}$$

$$v_\pi^c(s) = \mathsf{E}_\pi[G_t^c \mid S_t = s] \tag{2.38}$$

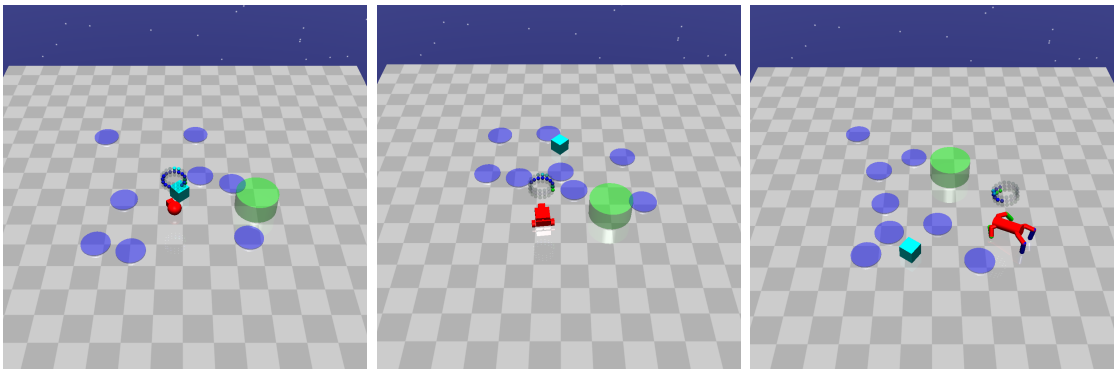
and $\gamma_c$ is the discount factor for the cost signal.

Thus, safe RL algorithms commonly use neural networks to approximate both the reward action value $Q^r$ and the cost action value $Q^c$. In the case of actor-critic methods, $Q^c$ is often referred to as a safety critic.

# 3 Related Work

In this chapter we present some of the work that focus on creating reinforcement learning agents that can learn and act safely within an environment, according to some safety measure. To this end, all of the following methods use the constrained reinforcement learning framework. Some focus on learning from fully observable states, while others aim to create methods that can work when dealing with partial observability. Each contribution is described as follows.

Ray, Achiam e Amodei (2019) made two main propositions. Fist, the authors suggested the use of constrained Markov decision process (CMDP) (ALTMAN, 1999) as the main formalism for safe reinforcement learning. CMDPs are discussed in Section 2.4. Second, they developed a benchmark suit to evaluate performance of reinforcement learning agents in safe exploration tasks. The benchmark suit, called Safety Gym, contains a series of different robots (the agent's "body"), tasks and hazards, which can be combined to form environments with a varying level of difficulty for RL agents. Figure 3.1 shows some examples of these environments.



(a) PointGoal1 (b) CarGoal1 (c) DoggoGoal1

Figure 3.1: Different Safety-Gym environments. In "Goal" tasks, the agent has to reach the goal (green cilinder) while trying to avoid hazards (blue circles and blue cube)

Furthermore, it was also provided in (RAY; ACHIAM; AMODEI, 2019) baselines versions of the trust region policy optimization (TRPO) (SCHULMAN et al., 2015) and proximal policy optimization (PPO) (SCHULMAN et al., 2017) algorithms adapted to

the CMDP setting by the use of the Lagrange multiplier method (BERTSEKAS, 2014). In conclusion, (RAY; ACHIAM; AMODEI, 2019) builds some foundations for a lot of subsequent work in the safe reinforcement learning field, by standardizing evaluation of safe RL algorithms.

## 3.1 Fully Observable Safe Reinforcement Learning

The setting where the agent gets direct access to the environment's true state is called the fully observable setting. Likewise, when the agent receives, instead, an observation that does not contain every information relevant to describe the environment's dynamics, the setting is described as partially observable and is formalized by a partially observable Markov decision process (POMDP) (KAELBLING; LITTMAN; CASSANDRA, 1998). In DRL, some methods are built based on the full observability assumption, while others relax this assumption and develop techniques to specifically mitigate partial observability. Throughout this section we talk about safe RL approaches that do the former.

Constrained policy optimization (CPO) (ACHIAM et al., 2017) performs policy search by augmenting the objective of the usual local policy search with CMDP constraints. In this context, the current policy $\pi_k$ is updated to the next policy $\pi_{k+1}$ by searching the space of all policies $\Pi_\theta$ that can be represented through the parameters $\theta$. This is done while also adhering to some constraints:

$$\begin{aligned} \pi_{k+1} = arg \max_{\pi \in \Pi_\theta} & J_r(\pi) \\ \textbf{s.t.} \quad & J_{c_i}(\pi) \leq d_i, \quad i = 1, \ldots, m \\ & D(\pi_k, \pi_{k+1}) < \upsilon \end{aligned} \tag{3.1}$$

where $D$ is some distance measure and $\upsilon$ is a step size. The main idea of local policy search is to update the policy's parameters without drastically changing its distribution at each time. Local search improves the stability of policy search methods and enables then to function in high dimensional function approximation settings.

With the augmented objective, Achiam et al. (2017) derive an approximated update that encounters a local feasible policy most of the time. Whenever this does not

happen, and the new policy $\pi_{k+1}$ is not feasible (that is, it does not satisfy all of the CMDP constraints), a recovery step is performed to bring the policy back to feasibility bounds.

Worst-case soft actor-critic (WCSAC) (YANG et al., 2023) is a SAC based algorithm that uses a distributional safety critic to produce risk-averse behavior. The authors propose two different versions of the algorithm: the first version estimates $Q_c$ using a Gaussian approximation and the second estimates $Q_c$ using a safety critic based on the IQN algorithm from Section 2.3. WCSAC-IQN outperforms its counterpart for increasingly complex environments, by being able to develop safe policies more consistently.

To perform risk-averse constrained RL, the upper tail of the estimated distribution is used. This is represented by the conditional Value-at-Risk (CVaR) (ROCKAFELLAR; URYASEV et al., 2000). To do so, a new hyper parameter is introduced called risk level, which we is defined here as $\beta \in (0, 1]$. The smaller the value of $\beta$, the more pessimistic the agent becomes, while $\beta = 1$ corresponds to the risk-neutral case. Thus, the CVaR metric, for a cost signal $c$ with random variable $C$ that follows the return distribution induced by $c$ when following policy $\pi$, is defined as:

$$\Gamma_\pi(s, a, \beta) = CVaR_\pi^\beta(C) = \mathsf{E}[C|C \geq F_C^{-1}(1-\beta)] \tag{3.2}$$

where $F_C$ is the c.d.f of the cost return distribution and, thus, $F_C^-1$ is the quantile function of the same distribution.

When using IQN as the safety critic, the CVaR metric can be approximated by, instead of sampling from a uniform distribution between zero and one ($\tau \sim U([0, 1])$), sampling from the uniform distribution between $1 - \beta$ and one ($\tau \sim U([1 - \beta, 1])$). Thus, to estimate CVaR with $K$ i.i.d. samples of $\tau \sim U([1 - \beta, 1])$:

$$\Gamma_\pi(s, a, \beta) = \frac{1}{K} \sum_{k=1}^{K} F_C^{-1}(\tau)(s, a) \tag{3.3}$$

Consequently, WCSAC-IQN augments the original SAC actor loss from equation 2.25:

$$J_\pi(\phi) = \mathsf{E}_{s\sim\mathsf{D},a\sim\pi_\phi}\left[\alpha \log \pi_\phi(a|s) - Q_\phi(s,a) - \lambda\Gamma_\pi(s,a,\beta)\right] \tag{3.4}$$

where $\lambda$ is dynamically adjusted based on the following loss:

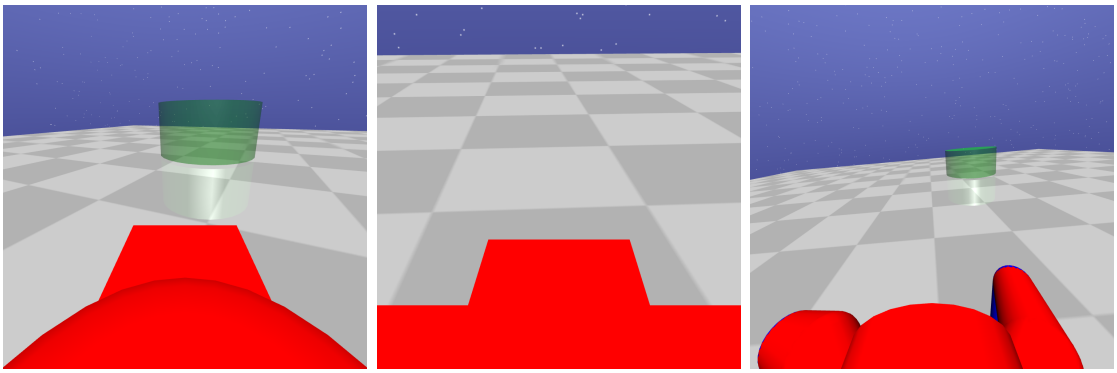
$$J_s(\lambda) = \mathsf{E}_{s\sim\mathsf{D},a\sim\pi_\phi}\left[\lambda(d - \Gamma_\pi(s,a,\beta))\right] \tag{3.5}$$

by the use of the Lagrange multiplier method. Where $d$ is the budget. In this way, when $d \geq \Gamma_\pi(s,a,\beta)$ the value of lambda will be decreased, giving less weight to the CVaR term in the actor loss; otherwise, lambda will increase, emphasizing the safety term and inducing the agent into looking for a safer policy. The reward critic is updated according to the common SAC loss, presented in Equation 2.23, while the safety critic follows the IQN loss from Equation 2.32.

## 3.2 Partially Observable Safe Reinforcement Learning

As RL applications increasingly approach real world scenarios, the challenge of partial observability becomes ever more present. To study the partial observable setting with high dimensional sensory inputs in safe reinforcement learning, the methods in this section modify the usual Safety-Gym observation. By default, agents receive multiple sensor inputs, such as joint position and velocity sensors, the robot sensors (accelerometer, gyroscope, magnetometer, and velocimeter), compasses for pointing to goals, and lidar (where each lidar sensor perceives objects of a single kind). The modified observation, however, takes a first-person view in respect to the robot perspective. It's easy to see why this would make the learning process harder: with the first-person view, the agent does not have information about the goal or hazard locations at every time step. Furthermore, before even learning to execute the task, the agent has to learn how to identify the relevant objects in the environment through the pixels values it receives. Figure 3.2 presents some examples of the agent's view.

To tackle this problem, the Lagrangian Model-Based Agent (LAMBDA) is pro-



(a) PointGoal1 (b) CarGoal1 (c) DoggoGoal1

Figure 3.2: Modified observations received by the agent at each Safety-Gym environment

posed in (AS et al., 2022). LAMBDA infers a Bayesian world model that is, then, used to simulate transitions and train the agent. Model-based techniques are used to improve sample efficiency of RL agents (DEISENROTH; RASMUSSEN, 2011), which is even more crucial in safety concerned task, as drawing experience from the model, instead of the real environment, is away to avoid general damage caused by the agent during exploration in real-world applications. Thus, by starting with a dataset of transitions $\mathsf{D}$, the method aims to fit a model $p(s_t + 1|s_t, a_t, \theta)$ with parameters $\theta$.

LAMBDA creates its world model in a similar manner to the Recurrent State Space Model (RSSM) from (HAFNER et al., 2019). First, it is assumed that, at each time step, the agent receives an observation that depends on the true state of the environment $o_t \sim p(\cdot|s_t)$. Next, an inference network with parameters $\phi$ is trained to approximate the posterior distribution $s_{\eta:\eta+H} \sim q_\phi(\cdot|o_{\eta:\eta+H}, a_{\eta-1:\eta+H-1})$, where $H$ is the predefined sequence horizon and $s_{i:j}$ represents the sequence of states received from the environment from time step $i$ to time step $j$, the same is valid for $a_{i:j}$ and $o_{i:j}$. The network $q_\phi$ infers a latent state, which is used as input to the policy. A second model is also trained to approximate the predictive distribution $p(s_{\eta:\eta+H}|s_{\eta-1}, a_{\eta-1:\eta+H-1}, \theta)$; this is later used to generate transitions. Additionally, LAMBDA estimates uncertainty in its world model by taking a prior on the model parameters and maintaining a posterior over these parameters via approximate Bayesian inference $\theta \sim p(\theta|\mathsf{D})$.

LAMBDA trains both its reward and cost critic using $TD(\lambda)$ technique (SUTTON; BARTO, 2018), a way to trade-off bias and variance by mixing bootstrapping with

Monte-Carlo estimation. Thus, they are updated according to the following loss function:

$$J_v(\psi) = \mathsf{E}_{\pi, p(s_{\eta:\eta+H}|s_{\eta-1}, a_{\eta-1:\eta+H-1}, \theta)} \left[ \frac{1}{2H} \sum_{t=\eta}^{\eta+H} (v_\psi(s_t) - \mathbf{V}_\lambda(s_t))^2 \right] \tag{3.6}$$

The policy update is performed based on the augmented Lagrangian method (WRIGHT, 2006):

$$J_\pi(\xi) = \mathsf{E}_{\pi, p(s_{\eta:\eta+H}|s_{\eta-1}, a_{\eta-1:\eta+H-1}, \theta)} \left[ \frac{1}{H} \sum_{t=\eta}^{\eta+H} -\mathbf{V}_\lambda(s_t) \right] + \sum_{i=1}^{C} \Psi(\pi, \lambda_k, \mu_k) \tag{3.7}$$

where $\Psi$ is a function that depends on the $TD(\lambda)$ value of the cost signal $\mathbf{V}_\lambda^c$.

Lastly, the authors propose the use of an optimistic pessimistic approach. In practice, when collecting transitions to perform the updates to the critics and the policy, they sample $M$ different set of weights $\theta_k \sim p(\theta|\mathsf{D})$ from their Bayesian model. For each $\theta_k$, $H$ transitions are sampled from the model parameterize by $\theta_k$: $s_{\eta:\eta+H} \sim p(s_{\eta:\eta+H}|s_{\eta-1}, a_{\eta-1:\eta+H-1}, \theta_k)$. These transitions are used to estimate $M$ different values for the $TD(\lambda)$ of the reward and the cost: $\mathbf{V}_\lambda^k$. Then, the maximum value of each set of estimates is used to perform the algorithm's updates. The main idea is to use the maximum reward return estimate to be optimistic in relation to pursuing high reward, while also being pessimistic by assuming the maximum possible cost return over the transitions generated by the $M$ sets of weights $\theta$.

(HOGEWIND et al., 2023) is another work that aims at solving partially observable problems in the context of safe reinforcement learning. The authors propose a new version of the Stochastic Latent Actor-Critic (SLAC) algorithm (LEE et al., 2020) called Safe SLAC, which is adapted to work under the CMDP framework. SLAC is an actor-critic method based on the SAC algorithm. It assumes partial observability and, in order to infer the true state of the environment $\mathbf{z}_t$, a sequential latent variable model is defined. Then, by predicting rewards and the observations based on the true hidden state, the model can be trained to find $\mathbf{z}_t$, such that the likelihood of $o_t$ and $r_t$ are maximized. This is done via variational inference (KINGMA; WELLING, 2013). In addition to the reward and observation, the Safe SLAC model is also trained to accurately predict the

cost $c_t$.

In practice, the variable $\mathbf{z}_t$ is factorized into two stochastic variables $\mathbf{z}_t^1$ and $\mathbf{z}_t^2$. Consequently, the generative part of the latent variable model with parameters $\psi$ consists of the following distributions:

$$\begin{aligned}
\mathbf{z}_1^1 &\sim p(\mathbf{z}_1^1) \\
\mathbf{z}_1^2 &\sim p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1) \\
\mathbf{z}_{t+1}^1 &\sim p_\psi(\mathbf{z}_{t+1}^1|\mathbf{z}_t^2, a_t) \\
\mathbf{z}_{t+1}^2 &\sim p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, a_t) \\
o_t &\sim p_\psi(o_t|\mathbf{z}_t^1, \mathbf{z}_t^2) \\
r_t &\sim p_\psi(r_t|\mathbf{z}_t^1, \mathbf{z}_t^2, a_t, \mathbf{z}_{t+1}^1, \mathbf{z}_{t+1}^2)
\end{aligned} \tag{3.8}$$

While the posterior contains the following factorization:

$$\begin{aligned}
\mathbf{z}_1^1 &\sim q_\psi(\mathbf{z}_1^1|o_1) \\
\mathbf{z}_1^2 &\sim p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1) \\
\mathbf{z}_{t+1}^1 &\sim q_\psi(o_{t+1}|\mathbf{z}_t^2, a_t) \\
\mathbf{z}_{t+1}^2 &\sim p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, a_t)
\end{aligned} \tag{3.9}$$

In (HOGEWIND et al., 2023), the authors augments the original SLAC generative model from Equation 3.8 with the conditional distribution $c_t \sim p_\psi(c_t|\mathbf{z}_t^1, \mathbf{z}_t^2, a_t, \mathbf{z}_{t+1}^1, \mathbf{z}_{t+1}^2)$. As the cost is always binary in Safety-Gym tasks, the conditional distribution is modeled as a Bernoulli distribution. Then, the latent model is trained with the following objective:

$$J_M(\psi) = \mathsf{E}_{\mathbf{z}_{1:\eta+1}\sim q_\psi}\left[\sum_{t=0}^{\eta} \begin{array}{c} -\log p_\psi(o_{t+1}|\mathbf{z}_{t+1}) \\ -\log p_\psi(r_{t+1}|\mathbf{z}_{t+1}) \\ -\log p_\psi(c_{t+1}|\mathbf{z}_{t+1}) \\ +D_{KL}(q_\psi(\mathbf{z}_{t+1}|o_t, \mathbf{z}_t, a_t)||p_\psi(\mathbf{z}_{t+1}|\mathbf{z}_t, a_t)) \end{array}\right] \tag{3.10}$$

where $D_{KL}$ is the Kullback-Leibler divergence (COVER, 1999). The reward and cost critic are updated in a similar manner to the original SAC critic (see Equation 2.23).

Finally, Safe SLAC actor loss is given by:

$$J_\pi(\phi) = \mathsf{E}_{\mathbf{z}_{1:\eta+1}\sim q_\psi}\left[\mathsf{E}_{a_{\eta+1}\sim\pi_\phi}\left[\begin{array}{c}\alpha\log\pi_\phi(a_{\eta+1}|\mathbf{z}_{\eta+1})\\ -Q^r_\theta(\mathbf{z}_{\eta+1},a_{\eta+1})\\ +\lambda Q^c_\xi(\mathbf{z}_{\eta+1},a_{\eta+1})\end{array}\right]\right] \tag{3.11}$$

where $Q^r_\theta$ is the reward critic and $Q^c_\xi$ is the safety critic.

In this work, we propose a method that relies on distributional reinforcement techniques, like (YANG et al., 2023), but, in contrasts, focus on solving tasks with high degree of partial observability, like (AS et al., 2022) and (HOGEWIND et al., 2023). We describe our method in detail in the following chapter.

# 4 Distributional Safe Stochastic Latent Actor-Critic

In the same vain as (AS et al., 2022) and (HOGEWIND et al., 2023), we propose a safe reinforcement learning algorithm to operate in the CPOMDP setting. Our approach is closely related to (HOGEWIND et al., 2023) in the sense that it is based on SLAC. The main difference is the use of a distributional safety critic instead of a expectation based one. Thus, we call the proposed approach as distributional safe stochastic latent actor-critic (DS-SLAC). We describe DS-SLAC in detail as follows.

DS-SLAC relies on the same latent variable model as (HOGEWIND et al., 2023) to infer the true hidden state of the environment $\mathbf{z}_t$ given an observation $o_t$. It does so using variational inference (KINGMA; WELLING, 2013) to optimize the model parameters in order to fit the observed data. The architecture is given by the following conditional probabilities:

$$
\begin{aligned}
&\mathbf{z}_1^1 \sim p(\mathbf{z}_1^1) \\
&\mathbf{z}_1^2 \sim p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1) \\
&\mathbf{z}_{t+1}^1 \sim p_\psi(\mathbf{z}_{t+1}^1|\mathbf{z}_t^2, a_t) \\
&\mathbf{z}_{t+1}^2 \sim p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, a_t) \\
&o_t \sim p_\psi(o_t|\mathbf{z}_t^1, \mathbf{z}_t^2) \\
&r_t \sim p_\psi(r_t|\mathbf{z}_t^1, \mathbf{z}_t^2, a_t, \mathbf{z}_{t+1}^1, \mathbf{z}_{t+1}^2) \\
&c_t \sim p_\psi(c_t|\mathbf{z}_t^1, \mathbf{z}_t^2, a_t, \mathbf{z}_{t+1}^1, \mathbf{z}_{t+1}^2)
\end{aligned}
\tag{4.1}
$$

$$
\begin{aligned}
&\mathbf{z}_1^1 \sim q_\psi(\mathbf{z}_1^1|o_1) \\
&\mathbf{z}_1^2 \sim p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1) \\
&\mathbf{z}_{t+1}^1 \sim q_\psi(o_{t+1}|\mathbf{z}_t^2, a_t) \\
&\mathbf{z}_{t+1}^2 \sim p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, a_t)
\end{aligned}
\tag{4.2}
$$

where Equation 4.1 corresponds to generative part of the model and Equation 4.2 corre-

sponds to the posterior portion. Given this architecture it is possible to train the latent variable model according to:

$$J_M(\psi) = \mathsf{E}_{\mathbf{z}_{1:\eta+1}\sim q_\psi}\left[\sum_{t=0}^{\eta} \begin{array}{c} -\log p_\psi(o_{t+1}|\mathbf{z}_{t+1}) \\ -\log p_\psi(r_{t+1}|\mathbf{z}_{t+1}) \\ -\log p_\psi(c_{t+1}|\mathbf{z}_{t+1}) \\ +D_{KL}(q_\psi(\mathbf{z}_{t+1}|o_t,\mathbf{z}_t,a_t)||p_\psi(\mathbf{z}_{t+1}|\mathbf{z}_t,a_t)) \end{array}\right] \tag{4.3}$$

This is the same loss as the one used to train the model in (HOGEWIND et al., 2023).

As DS-SLAC is also a SAC based-method, a reward critic is trained according to the maximum entropy framework described in Section 2.2.2. In this sense, the reward critic loss is similar to the one presented in Equation 2.23. It differs, however, in the fact that the hidden state produced by the latent variable model is used as input to the critic function. Thus, the loss used to update the parameters $\theta$ from the reward critic $Q^r_\theta$ is defined as:

$$J_{Q^r}(\theta) = \mathsf{E}_{a_t,r_t\sim\mathsf{D},\mathbf{z}_t,\mathbf{z}_{t+1}\sim q_\psi}\left[(Q_\theta(\mathbf{z}_t,a_t) - \hat{Q}(\mathbf{z}_t,a_t))^2\right] \tag{4.4}$$

where:

$$\hat{Q}(\mathbf{z}_t,a_t) = r_t + \gamma(Q_\theta(\mathbf{z}_{t+1},a_{t+1}) - \alpha\log\pi(a_{t+1}|\mathbf{z}_{t+1})), \quad a_{t+1}\sim\pi(\cdot|\mathbf{z}_{t+1}) \tag{4.5}$$

and $\mathsf{D}$ is the replay buffer.

As mentioned before, a safety critic is also trained. Our reasons for adopting a distributional perspective on the safety critic are twofold. First, inspired by the authors of (YANG et al., 2023), we believe that it is particularly interesting to be able to produce risk-averse behavior in the context of safe reinforcement learning. Second, the use of a distributional safety critic can increase the accuracy of the predictions of cost returns, which, in turn, allows for the agent to more accurately trade-off reward and cost in the CMDP setting.

Consequently, DS-SLAC estimates the cost return distribution with a implicit quantile network discussed in section 2.3.1. For brevity, we define $Q^c_{\xi\tau}$ as the function parameterized by $\xi$ that approximates $F^{-1}_{Z_c}(\tau)$, where $Z_c$ is the cost return distribution

and $F_{Z_c}$ is the c.d.f. of this distribution. In this manner, we train $Q^c_{\xi\tau}$ by sampling $N$ and $N'$ samples of $\tau, \tau' \sim U([0,1])$ and estimating IQN the loss according to:

$$J_{Q^c}(\xi) = \frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho^{\kappa}_{\tau_i}(\delta_t^{\tau_i,\tau'_j}), \quad \text{where} \tag{4.6}$$

$$\delta_t^{\tau,\tau'} = r_t + \gamma Q^c_{\xi\tau'}(\mathbf{z}_{t+1}, a_{t+1}) - Q^c_{\xi\tau}(\mathbf{z}_t, a_t), \quad a_{t+1} \sim \pi_\phi(\cdot|\mathbf{z}_{t+1}) \tag{4.7}$$

and $\rho^{\kappa}_{\tau}$ is the Huber quantile regression loss described in equation 2.29.

Through estimating the cost return distribution and the expected reward return, the policy update can be performed according to:

$$J_\pi(\phi) = \mathsf{E}_{\mathbf{z}_t \sim q_\psi, a_t \sim \pi_\phi} \begin{bmatrix} \alpha \log \pi_\phi(a_t|\mathbf{z}_t) \\ -Q^r_\theta(\mathbf{z}_t, a_t) \\ +\lambda Q^c_\xi(\mathbf{z}_t, a_t) \end{bmatrix} \tag{4.8}$$

with

$$Q^c_\xi(\mathbf{z}_t, a_t) = \frac{1}{K} \sum_{k=1}^{K} Q^c_{\xi\tau}(\mathbf{z}_t, a_t) \tag{4.9}$$

where $K$ is the number of i.i.d. samples $\tau \sim U([0,1])$ used to estimate $Q^c_\xi(\mathbf{z}_t, a_t)$, $\alpha$ is the parameter used to regulate the trade-off between reward and entropy, and $\lambda$ is the Lagrange multiplier that, in turn, regulates the trade-off between reward and cost constraint.

Similarly to (HOGEWIND et al., 2023), we find during our preliminary experiments that updating the Lagrange multiplier with off-policy data (i.e., by sampling from the replay buffer $\mathsf{D}$) results in high instability during learning, often leading the agent to developing an unsafe policy. We found no other way of mitigating this stability besides updating the Lagrange multiplier with on-policy data, like it is done in (HOGEWIND et al., 2023). For (HOGEWIND et al., 2023), this solution works well. In our case, however, this is particularly undesirable, as it prevents us from performing risk-averse updates by using the CVaR metric to update the Lagrange multiplier, as it is done by (YANG et al., 2023) and represented in equation 3.5. Thus, the update for the Lagrange multiplier is performed according to the following loss:

$$J_s(\lambda) = \mathsf{E}_\pi \left[ \lambda \sum_{t=1}^{T} c_t - d \right] \tag{4.10}$$

where $c_t$ is the cost induced by following policy $\pi$ and $d$ is the budget.

Thus, DS-SLAC works in the following manner. First, the replay buffer $\mathsf{D}$ is initialized by interacting with the environment and selecting actions according to a random policy for $W_p$ environments steps, in this way $W_p$ transitions are stored in the replay buffer $\mathsf{D}$. Next, the latent variable model is pre-trained by sampling the transitions collected with the random policy from the replay buffer $(o_t, a_t, r_t, c_t) \sim \mathsf{D}$; this is performed for $W_t$ times.

Afterwards the algorithm alternates between multiple iterations of two processes. The first consists of the agent interacting with the environment by selecting an action $a_t$ based on the latent state $z_t$ that is inferred by the latent model according to previous latent state $z_{t-1}$ and current observation $o_t$. The selected action is executed in the environment and is stored in the replay buffer along with the received observation, reward and cost $(o_{t+1}, r_{t+1}, c_{t+1})$. During this step, the Lagrange multiplier is updated, since it needs to be on-policy updated, as mentioned previously.

The second process consists of sampling a transition from the replay buffer $(o_t, a_t, r_t, c_t) \sim \mathsf{D}$, then calculating the losses for the latent model, reward critic, cost critic and policy based on the sampled transition and according to the respective loss equations. Next, the respective weights are updated by performing a single gradient step. Lastly, the target networks for the reward and safety critic are also updated through exponential averaging (MNIH et al., 2015).

The alternation of these two process can occur for as long as desired, essentially, until the agent develops a good enough policy. In practice, a maximum number of desired environment interactions (number of time the first process is repeated) is defined and when this number is reached training stops. This is done to evaluate the agent sample efficiency, i.e. its performance relative to the number of samples of environment interactions used for learning.

Finally, DS-SLAC is described in detail in Algorithm 1, where $\theta_1$ and $\theta_2$ represent the double Q-learning networks (FUJIMOTO; HOOF; MEGER, 2018), while $\bar{\theta_1}$, $\bar{\theta_2}$ and

$\bar{\xi}$ are the target networks (MNIH et al., 2015).

**Algorithm 1:** Distributional Safe Stochastic Latent Actor-Critic

**Hyperparameters**: $W_t, W_p, N, N', K$

Initialize $\mathsf{D}$ by following a random policy with $W_p$ transitions
**for** *i=1 to* $W_t$ **do**
    $(o_t, a_t, r_t, c_t) \sim \mathsf{D}$
    Update $\psi$ according to Equation 4.3
**end for**
**while** *not converged* **do**
    **for** *each environment step* **do**
        $a_t \sim \pi_\phi(a_t|\mathbf{z}_t)$
        Obtain $(o_{t+1}, r_{t+1}, c_{t+1})$ by executing $a_t$
        $\mathsf{D} \leftarrow \mathsf{D} \cup (o_{t+1}, a_t, r_{t+1}, c_{t+1})$
        Update $\lambda$ according to Equation 4.10
    **end for**
    **for** *each gradient step* **do**
        $(o_t, a_t, r_t, c_t) \sim \mathsf{D}$
        Update $\psi$ according to Equation 4.3
        Update $\theta_1$ and $\theta_2$ according to Equation 4.4
        Update $\xi$ according to Equation 4.6
        Update $\phi$ according to Equation 4.8
        $\bar{\theta_1} \leftarrow \nu\theta_1 + (1-\nu)\bar{\theta_1}$
        $\bar{\theta_2} \leftarrow \nu\theta_2 + (1-\nu)\bar{\theta_2}$
        $\bar{\xi} \leftarrow \nu\xi + (1-\nu)\bar{\xi}$
    **end for**
**end while**

# 5 Computational Experiments

In practice, DS-SLAC collects $W_p = 60K$ environments steps following a complete random policy. Next, for $W_t = 30K$ steps, the latent variable model is initialized based on the collected information. Additionally, during the training of the policy, DS-SLAC uses 100 environment steps and 100 gradient steps, meaning it collects 100 environment interactions and then performs 100 gradient updates, repeating this process until the maximum number of environment steps is reached.

We perform our experiments in a subset of the SG6 benchmark from (RAY; ACHIAM; AMODEI, 2019). Following As et al. (2022) and Hogewind et al. (2023), we perform evaluation of each run as following: for each 25K environment steps, the agent performance is measured by collecting $E = 10$ different episodes of length of $T_{ep} = 1000$ with the current policy. Then, the undiscounted reward and cost return for each episode are calculated and the mean of these metrics across all episodes represent the current policy performance. In short, the reward performance is given by $\frac{1}{E}\sum_{e=1}^{E}\sum_{t=1}^{T_{ep}} r_t$ and the cost performance is given by $\frac{1}{E}\sum_{e=1}^{E}\sum_{t=1}^{T_{ep}} c_t$. The data used for evaluation is then discarded and is not used for training.

Figure 5.1 presents the DS-SLAC learning curves for each Safety-Gym environment, generated by avaraging the performance of three different random seeds. Doted lines indicate the final results for LAMBDA, CPO and TRPO-Lag that were reported by As et al. (2022). Both CPO and TRPO-Lag learn directly from sensors, with 10 million environment steps, meanwhile LAMBDA and DS-SLAC learn for 1 million environment steps in PointGoal1, CarGoal1 and PointGoal2, and for 2 million steps in DoggoGoal1. (HOGEWIND et al., 2023) does not report the results for Safe-Slac in a tabular form, only through figures. As such, we do not include a direct comparison to Safe-Slac results. Nevertheless, it is still possible to get a sense of the comparative performance of both agents by looking at the graphics present here and in (HOGEWIND et al., 2023).

As proposed by (RAY; ACHIAM; AMODEI, 2019), we also present a comparison of the normalized final reward and cost returns. Contrary to the learning curves, the

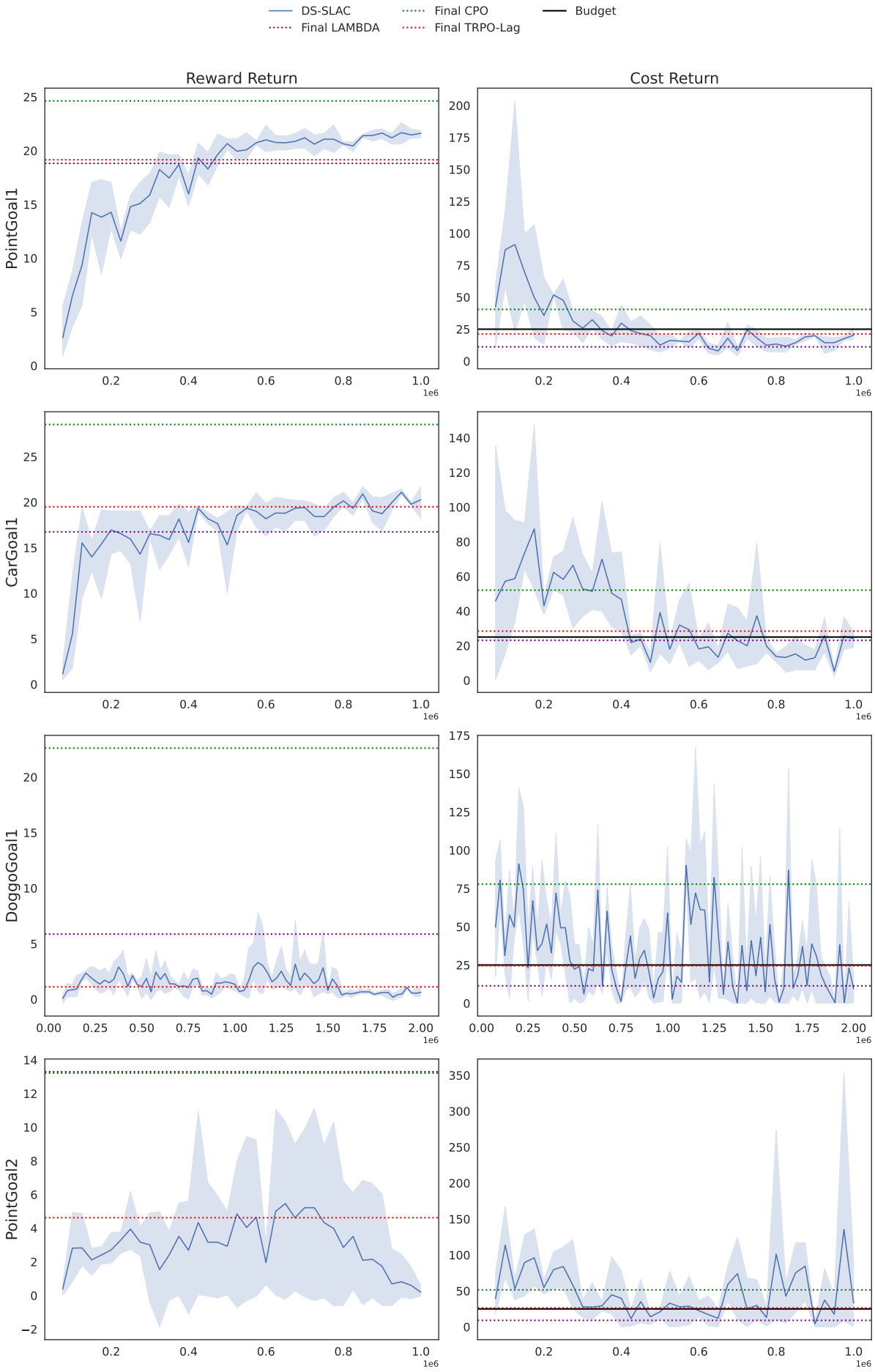



Figure 5.1: Learning curves for DS-SLAC in four different Safety-Gym environments.

normalized results are based on the final reward return and cost return after 1 million environment steps of training across all agents, including CPO and TRPO-Lag. The normalization is performed based on the result of an unconstrained proximal policy optimization (PPO) (SCHULMAN et al., 2017) agent, also reported by (AS et al., 2022). Then, for each agent with a reward return $\hat{J}_r(\pi)$ and a cost return $\hat{J}_c(\pi)$, and for the reward return $\hat{J}_r^{PPO}$ and cost return $\hat{J}_c^{PPO}$ of the PPO agent, the normalized results are given by:

$$\begin{aligned} \bar{J}_r(\pi) &= \frac{\hat{J}_r(\pi)}{\hat{J}_r^{PPO}} \\ \bar{J}_c(\pi) &= \frac{max(0, \hat{J}_c(\pi) - d)}{max(10^{-6}, \hat{J}_c^{PPO} - d)} \end{aligned} \tag{5.1}$$

The normalized metrics are presented in Figure 5.2 and in Table 5.1.

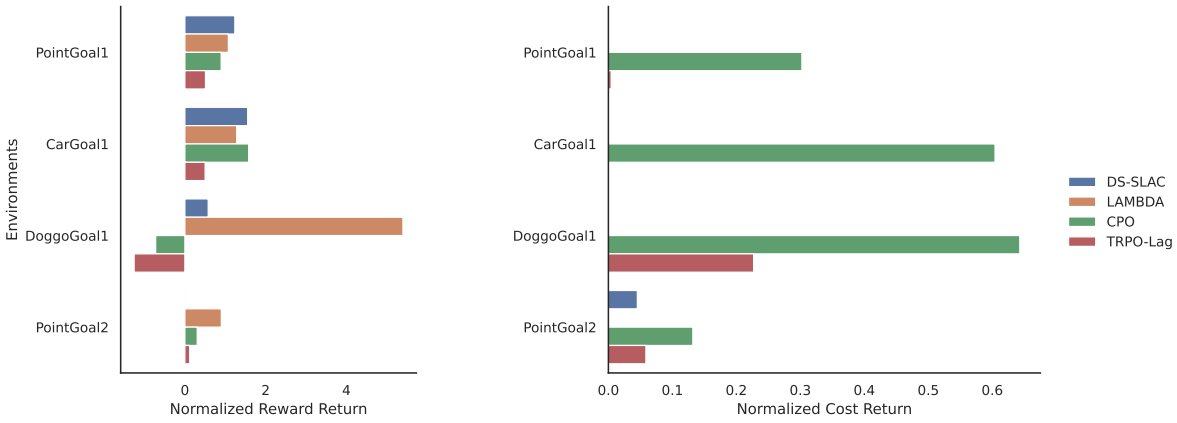



Figure 5.2: Normalized reward return and cost return.

Table 5.1: Normalized final results with each cell containing a tuple $(\bar{J}_r(\pi), \bar{J}_c(\pi))$.

| | TRPO-Lag | CPO | LAMBDA | DS-SLAC |
|---|---|---|---|---|
| PointGoal1 | 0.51, 0.004 | 0.898, 0.302 | 1.077, **0.0** | **1.237**, **0.0** |
| CarGoal1 | 0.501, **0.0** | **1.579**, 0.604 | 1.284, **0.0** | 1.555, **0.0** |
| DoggoGoal1 | -1.257, 0.227 | -0.723, 0.643 | **5.400**, **0.0** | 0.577, **0.0** |
| PointGoal2 | 0.119, 0.059 | 0.306, 0.132 | **0.902**, **0.0** | 0.014, 0.045 |

DS-SLAC demonstrates great performance in both PointGoal1 and CarGoal1. For either one of these environments, DS-SLAC outperforms all other methods by attaining the highest reward return, while also developing a safe policy.

On the other hand, for both DoggoGoal1 and PointGoal2, although DS-SLAC generates a safe policy for DoggoGoal1 and is very close to the budget value in the case of PointGoal2, our algorithm does not perform well in respect to reward return. In part, this is occurs because these environments are significantly harder for agents in general. PointGoal2 contains a lot more hazards when compared to PointGoal1. Meanwhile, DoggoGoal1 robot has a large size, making it hard to avoid safety violations. Additionally, in the DoggoGoal1 environment, the robot's point of view shakes constantly when it moves, inducing a high degree of partial observability. The difficulty of these environments can also be attested by the performance of baseline algorithms, as seen in Figure 5.2.

Nevertheless, DS-SLAC still significantly underperforms when compared to both LAMBDA and Safe SLAC. We hypothesize that, for the more complex environments, DS-SLAC safety critic begins to overestimate cost values during training, eventually preventing the agent to seek higher reward return as the policy objective becomes too heavily influenced by the safety term. We believe one possible cause for this overestimation is the use of on-policy data to update the Lagrange multiplier, while the safety term in the actor loss is calculated in an off-policy manner. This factor can make the safety critic update unstable.

## 5.1 Implementation Details

For the purpose of reproducibility, we make the code for DS-SLAC available at: ⟨https://github.com/tsmir/ds-slac⟩. The algorithm is implemented using the PyTorch library (PASZKE et al., 2017) and the repository contains the code for interfacing with Safety-Gym using first-person pixel observations.

In PointGoal1, CarGoal1 and PointGoal2, where 1 million environment steps were used, DS-SLAC training took about 9 hours for each random seed. In DoggoGoal1, as the number of environment steps was doubled, training took about 18 hours for each random seed. All the experiments were performed on a machine equipped with a i5-10400F processor and a RTX 3070 Ti GPU. Lastly, Table 5.2 presents all the hyperparameters used to execute the algorithm.

Table 5.2: Hyperparemeters for DS-SLAC.

| Parameter | Value |
|---|---|
| Action repeat | 2 |
| Image size | $64 \times 64 \times 3$ |
| Image reconstruction noise | 0.4 |
| Length of sequences sampled from replay buffer | 10 |
| Reward discount factor | 0.99 |
| Cost discount factor | 0.995 |
| $z^1$ size | 32 |
| $z^2$ size | 200 |
| Replay buffer size | $2 \cdot 10^5$ |
| Latent model update batch size | 32 |
| Actor-critic update batch size | 64 |
| Latent model learning rate | $1 \cdot 10^{-4}$ |
| Actor-critic learning rate | $2 \cdot 10^{-4}$ |
| Safety Lagrange multiplier learning rate | $2 \cdot 10^{-4}$ |
| Initial value for $\alpha$ | $4 \cdot 10^{-3}$ |
| Initial value for $\lambda$ | $2 \cdot 10^{-2}$ |
| $W_p$ | $60 \cdot 10^3$ |
| $W_t$ | $30 \cdot 10^3$ |
| $N$ | 64 |
| $N'$ | 64 |
| $K$ | 64 |
| Gradient clipping max norm | 40 |
| Target network update exponential factor | $5 \cdot 10^{-3}$ |

# 6 Conclusion

As reinforcement learning techniques increasingly transition from being applied to simulated environments to real world applications, the topic of avoiding damage, risks and unwanted scenarios during an agent's interaction with the environment becomes ever more prevalent. As such, there exists a intrinsic motivation for developing methods that are able to specify and enforce safety constraints in order to produce safe behavior.

Current state-of-the-art safe reinforcement learning algorithms work under the constrained Markov decision process (CMDP) formalism to create agents that perform well, even under a high degree of partial observability. We developed an algorithm named DS-SLAC, that combines techniques of amortized variational inference used to mitigate partial observability with a distributional reinforcement learning perspective. We obtain comparable results to state-of-the-art methods in some Safety-Gym environments, as DS-SLAC outperforms other algorithms in two of these environments. Additionally, we identify some of the challenges in performing safe reinforcement learning with a distributional safety critic under the CPOMDP framework.

Future work can focus on improving performance of DS-SLAC in more complex environments. We hypothesize that updating the Lagrange multiplier with on-policy data, while using off-policy data to calculate the safety term in the policy update results in overestimation of the cost term used in the actor loss. As such, the cost term would dominate the policy objective, preventing the agent from focusing on accumulating reward.

Furthermore, we believe that being able to perform risk-averse constrained reinforcement learning is desirable in the safety setting. This can be performed when using distributional reinforcement learning to estimate the full cost return distribution, then the agent's behavior is updated according to "worst-case" scenarios, i.e. scenarios where cost return assumes a value in the upper end of the distribution.

In DS-SLAC case, we were not able to perform risk-averse constrained reinforcement learning, due to high instability when performing off-policy updates to the Lagrange multiplier, i.e., the term that regulates the trade-off between adhering to safety constraints

and seeking a higher reward. Thus, encountering a way to perform the Lagrange multiplier update in an off-policy manner would unlock the ability to produce risk-averse behavior and, consequently, is a promising path for future work.

# Bibliography

ACHIAM, J.; HELD, D.; TAMAR, A.; ABBEEL, P. Constrained policy optimization. In: PMLR. *International conference on machine learning*. [S.l.], 2017. p. 22–31.

ALTMAN, E. *Constrained Markov decision processes*. [S.l.]: CRC press, 1999. v. 7.

ARULKUMARAN, K.; DEISENROTH, M. P.; BRUNDAGE, M.; BHARATH, A. A. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, IEEE, v. 34, n. 6, p. 26–38, 2017.

AS, Y.; USMANOVA, I.; CURI, S.; KRAUSE, A. Constrained policy optimization via bayesian world models. *arXiv preprint arXiv:2201.09802*, 2022.

BELLEMARE, M. G.; DABNEY, W.; MUNOS, R. A distributional perspective on reinforcement learning. In: PMLR. *International conference on machine learning*. [S.l.], 2017. p. 449–458.

BELLEMARE, M. G.; DABNEY, W.; ROWLAND, M. *Distributional reinforcement learning*. [S.l.]: MIT Press, 2023.

BERTSEKAS, D. P. *Constrained optimization and Lagrange multiplier methods*. [S.l.]: Academic press, 2014.

BROWN, T.; MANN, B.; RYDER, N.; SUBBIAH, M.; KAPLAN, J. D.; DHARIWAL, P.; NEELAKANTAN, A.; SHYAM, P.; SASTRY, G.; ASKELL, A. et al. Language models are few-shot learners. *Advances in neural information processing systems*, v. 33, p. 1877–1901, 2020.

CHEN, M.; TWOREK, J.; JUN, H.; YUAN, Q.; PINTO, H. P. d. O.; KAPLAN, J.; EDWARDS, H.; BURDA, Y.; JOSEPH, N.; BROCKMAN, G. et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

COVER, T. M. *Elements of information theory*. [S.l.]: John Wiley & Sons, 1999.

DABNEY, W.; OSTROVSKI, G.; SILVER, D.; MUNOS, R. Implicit quantile networks for distributional reinforcement learning. In: PMLR. *International conference on machine learning*. [S.l.], 2018. p. 1096–1105.

DABNEY, W.; ROWLAND, M.; BELLEMARE, M.; MUNOS, R. Distributional reinforcement learning with quantile regression. In: *Proceedings of the AAAI Conference on Arti cial Intelligence*. [S.l.: s.n.], 2018. v. 32, n. 1.

DEISENROTH, M.; RASMUSSEN, C. E. Pilco: A model-based and data-efficient approach to policy search. In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. [S.l.: s.n.], 2011. p. 465–472.

FUJIMOTO, S.; HOOF, H.; MEGER, D. Addressing function approximation error in actor-critic methods. In: PMLR. *International conference on machine learning*. [S.l.], 2018. p. 1587–1596.

GARCIA, J.; FERNÁNDEZ, F. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, v. 16, n. 1, p. 1437–1480, 2015.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. ⟨http://www.deeplearningbook.org⟩.

HAARNOJA, T.; ZHOU, A.; ABBEEL, P.; LEVINE, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: PMLR. *International conference on machine learning*. [S.l.], 2018. p. 1861–1870.

HAARNOJA, T.; ZHOU, A.; HARTIKAINEN, K.; TUCKER, G.; HA, S.; TAN, J.; KUMAR, V.; ZHU, H.; GUPTA, A.; ABBEEL, P. et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

HAFNER, D.; LILLICRAP, T.; FISCHER, I.; VILLEGAS, R.; HA, D.; LEE, H.; DAVIDSON, J. Learning latent dynamics for planning from pixels. In: PMLR. *International conference on machine learning*. [S.l.], 2019. p. 2555–2565.

HAUSKNECHT, M.; STONE, P. Deep recurrent q-learning for partially observable mdps. In: *2015 aaai fall symposium series*. [S.l.: s.n.], 2015.

HAYKIN, S. *Neural networks and learning machines, 3/E*. [S.l.]: Pearson Education India, 2009.

HESSEL, M.; MODAYIL, J.; HASSELT, H. V.; SCHAUL, T.; OSTROVSKI, G.; DABNEY, W.; HORGAN, D.; PIOT, B.; AZAR, M.; SILVER, D. Rainbow: Combining improvements in deep reinforcement learning. In: *Proceedings of the AAAI conference on arti cial intelligence*. [S.l.: s.n.], 2018. v. 32, n. 1.

HOGEWIND, Y.; SIMAO, T. D.; KACHMAN, T.; JANSEN, N. Safe reinforcement learning from pixels using a stochastic latent representation. In: *International Conference on Learning Representations*. [S.l.: s.n.], 2023.

HSU, P.-L.; ROBBINS, H. Complete convergence and the law of large numbers. *Proceedings of the national academy of sciences*, National Acad Sciences, v. 33, n. 2, p. 25–31, 1947.

HUBER, P. J. Robust estimation of a location parameter. *Breakthroughs in statistics: Methodology and distribution*, Springer, p. 492–518, 1992.

ISOM, J. D.; MEYN, S. P.; BRAATZ, R. D. Piecewise linear dynamic programming for constrained pomdps. In: *AAAI*. [S.l.: s.n.], 2008. v. 1, p. 291–296.

KAELBLING, L. P.; LITTMAN, M. L.; CASSANDRA, A. R. Planning and acting in partially observable stochastic domains. *Arti cial intelligence*, Elsevier, v. 101, n. 1-2, p. 99–134, 1998.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of arti cial intelligence research*, v. 4, p. 237–285, 1996.

KINGMA, D. P.; WELLING, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

LEE, A. X.; NAGABANDI, A.; ABBEEL, P.; LEVINE, S. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, v. 33, p. 741–752, 2020.

LEE, J.; KIM, G.-H.; POUPART, P.; KIM, K.-E. Monte-carlo tree search for constrained pomdps. *Advances in Neural Information Processing Systems*, v. 31, 2018.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; LILLICRAP, T.; HARLEY, T.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. In: PMLR. *International conference on machine learning*. [S.l.], 2016. p. 1928–1937.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G. et al. Human-level control through deep reinforcement learning. *nature*, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

NG, A. Y.; HARADA, D.; RUSSELL, S. Policy invariance under reward transformations: Theory and application to reward shaping. In: CITESEER. *Icml*. [S.l.], 1999. v. 99, p. 278–287.

PASZKE, A.; GROSS, S.; CHINTALA, S.; CHANAN, G.; YANG, E.; DEVITO, Z.; LIN, Z.; DESMAISON, A.; ANTIGA, L.; LERER, A. Automatic differentiation in pytorch. In: *NIPS-W*. [S.l.: s.n.], 2017.

POUYANFAR, S.; SADIQ, S.; YAN, Y.; TIAN, H.; TAO, Y.; REYES, M. P.; SHYU, M.-L.; CHEN, S.-C.; IYENGAR, S. S. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 51, n. 5, p. 1–36, 2018.

RAMESH, A.; DHARIWAL, P.; NICHOL, A.; CHU, C.; CHEN, M. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

RAY, A.; ACHIAM, J.; AMODEI, D. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, v. 7, n. 1, p. 2, 2019.

ROBBINS, H.; MONRO, S. A stochastic approximation method. *The annals of mathematical statistics*, JSTOR, p. 400–407, 1951.

ROCKAFELLAR, R. T.; URYASEV, S. et al. Optimization of conditional value-at-risk. *Journal of risk*, Citeseer, v. 2, p. 21–42, 2000.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group UK London, v. 323, n. 6088, p. 533–536, 1986.

RUMMERY, G. A.; NIRANJAN, M. *On-line Q-learning using connectionist systems*. [S.l.]: Citeseer, 1994. v. 37.

SCHULMAN, J.; LEVINE, S.; ABBEEL, P.; JORDAN, M.; MORITZ, P. Trust region policy optimization. In: PMLR. *International conference on machine learning*. [S.l.], 2015. p. 1889–1897.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. V. D.; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M. et al. Mastering the game of go with deep neural networks and tree search. *nature*, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

TANG, Y. C.; ZHANG, J.; SALAKHUTDINOV, R. Worst cases policy gradients. *arXiv preprint arXiv:1911.03618*, 2019.

WRIGHT, S. J. *Numerical optimization*. 2006.

YANG, Q.; SIMÃO, T. D.; TINDEMANS, S. H.; SPAAN, M. T. Safety-constrained reinforcement learning with a distributional safety critic. *Machine Learning*, Springer, v. 112, n. 3, p. 859–887, 2023.