Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Constrained Reinforcement Learning for Process Optimisation

Author:
Jaime Sabal Bermúdez

Supervisor:
Dr. Calvin Tsay

Submitted in partial fulfillment of the requirements for the MSc degree in Artificial Intelligence of Imperial College London

June 2022

Contents

Abstract

Abstract should go here...

Acknowledgements

Acknowledgements here...

Introduction

For all living organisms, the ability to learn by interacting with their environment is essential for survival. As humans, we learn how to interact with this world in a manner that allows us to maximise our time living in it, a primal instinct hard-wired into our consciousness through millions of years of evolution. However, existing in such a complex environment implies being subject to caveats which challenge and restrict the way in which we learn to adapt to it. For example, fish have evolved to only survive in aquatic settings and therefore act accordingly and instinctively to satisfy this constraint. In reality, there are usually a variety of constraints that an agent must consider to act safely.

In the past few decades, and as computational power has vastly increased, machine learning has gained a lot of popularity for its capacity to imitate the way humans learn. This is done by solving problems numerically through the use of algorithms that mimic different aspects of cognitive reasoning. Following this definition, it makes sense to design a method that reflects the most basic way in which organisms make decisions - by interacting with their environment and adjusting their actions iteratively after making observations on the effect they had (i.e. after receiving a ”reward” from the interaction). The sub-field of Reinforcement Learning (RL) [1] does this precisely, providing a framework for artificial agents to learn by interacting with an environment, analogously to how we as humans learn by interacting with our complex world. Reinforcement learning algorithms have proven to be very effective at solving tasks in areas such as gaming, where superhuman performance has been achieved in incredibly complex games like Go [2], or robotics and control, where often problems are too complex to be solved analytically.

Traditional RL involves an agent making decisions while continuously interacting with an environment freely, without being subject to any constraints. This means that there is no implicit cost associated with unlimited exploration of the environment and/or of specific states within it. Recent efforts have been made in the area of constrained RL, which attempts to address this limitation of traditional RL, to optimise the way in which an agent explores its environment and learns how to act whilst being subject to a set of constraints. Advancements in this area could lead the way to wide-spread use of reinforcement learning for industry applications. Currently, businesses often rely on poorly designed internal processes that lead to material and time wastes which could be easily avoided through the use of constrained reinforcement learning.

Theoretical Background

2.1 Reinforcement Learning

Reinforcement learning defines a framework for the interaction between an *agent* and an *environment* in terms of the possible configurations of the environment as observed by the agent (i.e., the set of *states* $\mathcal{S}$), the ways in which the agent can transition between states (i.e., the set of *actions* $\mathcal{A}$), and the benefit that being in state $S_t \in \mathcal{S}$ at time-step t has towards achieving the agents' goal (i.e., the *reward* $R_t \in \mathcal{R} \subset \mathbb{R}$). Moreover, the agent transitions between states according to the state transition probability function $\mathcal{P}$, which captures the dynamics of the environment. Its goal is then to learn a *policy* π that maps each state s and action a to the probability $\pi_t(A_t = a \,|\, S_t = s)$ of taking action A_t in state S_t, and which maximises the *total cumulative discounted reward* G_t received over the future for all time-steps $t \in T$. This can be expressed as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

where R_t is the reward received after taking action A_t in state S_t and $\gamma \in [0, 1]$ is the *discount rate*, which allows the infinite sum to have a finite value as long as the reward sequence $\{R_k\}$ is bounded.

Markov Decision Processes

Reinforcement learning problems are usually said to have the *Markov property* and therefore are assumed to be *Markov decision processes* (MDP's), meaning that the probability of transitioning to a new state s', given the current state s and action a, is conditionally independent of all previous states and actions and as such:

$$\mathbb{P}\left[S_{t+1} = s' \,|\, S_t, A_t\right] = \mathbb{P}\left[S_{t+1} = s' \,|\, S_0, A_0, ..., S_{t-1}, A_{t-1}, S_t, A_t\right] \tag{2.2}$$

This property implies that the current state contains all the relevant information about the past needed to predict future rewards and select actions, which is useful since it reduces problems to functions of the current state only. Formally, an MDP is defined by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, where $\mathcal{P}$ is the transition probability function, γ is the discount rate, and $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ are the state, action, and reward spaces, respectively.

Value Functions

Now that we have defined a framework for reinforcement learning problems, it is convenient to define a metric that reflects the potential of a specific state with respect to the goal of maximising future rewards. Usually, the objective in reinforcement learning problems (i.e., MDP's) is to learn the optimal *value function* $V^*(s)$, expressed as:

$$\begin{aligned} V^*(s) &= \mathbb{E}_{\pi^*}\left[G_t \,|\, S_t = s\right] \\ &= \max_{\pi} V_\pi(s) \end{aligned} \tag{2.3}$$

which gives a numerical "score" to each state s from the expected value of G_t given s under the optimal policy π^*. Learning the optimal value function directly leads us to find the optimal policy, since for two different policies π and π' we have that $\pi \geq \pi' \iff V_\pi(s) \geq V_{\pi'}(s)$, $\forall s$ is always true [1], where the subscripts on the value functions indicate the policy being followed. Moreover, it is often useful to additionally define the *state-action value function* (2.4) and its optimal equivalent (2.5) as:

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \,|\, S_t = s, A_t = a\right] \tag{2.4}$$

$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a) \tag{2.5}$$

since we express policies as probabilities of choosing an action a from a state s. The value function can then be expressed in terms of $Q_\pi(s,a)$ as:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} Q_\pi(s,a)\pi(s,a) \tag{2.6}$$

Consequently, the optimal policy π^* is obtained through the maximisation of the value function:

$$\pi^* = \arg\max_{\pi} V_\pi(s) = \arg\max_{\pi} Q_\pi(s,a) \tag{2.7}$$

Bellman's Equations

Combining (2.1) and (2.3), we can derive the following recursive relationship in value functions:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,|\, S_t = s\right] \\ &= \mathbb{E}_{a\sim\pi, s'\sim\mathcal{P}}\left[r(s,a) + \gamma V_\pi(s')\right] \end{aligned} \tag{2.8}$$

where the action a is sampled from the policy π, and the subsequent state s' from the transition probability function $\mathcal{P}$. Equivalently for the state-action value function:

$$Q_\pi(s,a) = \mathbb{E}_{s'\sim\mathcal{P}}\left[r(s,a) + \gamma \mathbb{E}_{a'\sim\pi}\left[Q_\pi(s',a')\right]\right] \tag{2.9}$$

which together with (2.8) yield *Bellmans equation's*, and provide an update rule to iteratively improve approximations of value functions through the use of *dynamic programming* [3] (see Section 2.1.2), which ensures convergence to the optimal value function. In this manner, a complex problem that is recursive in nature (often the case for problems that span over time) can be broken down into simpler sub-problems and solved optimally and efficiently.

2.1.1 Traditional Methods

Traditional or *tabular* RL refers to problems where state and action spaces are small enough that optimal value functions can be approximated using look-up tables/arrays by using the Bellman equations. RL algorithms can be divided into model-based and model-free methods. The former require knowledge of the underlying environment dynamics (i.e. state-transition probabilities) through the use of a *model*. This model can be exact (e.g. if there are a clear set of rules that govern these dynamics like in a game of chess) or an approximation learned and provided to the agent by an external source (e.g. a neural network). On the other hand, model-free methods learn from sampled experience, with an approximation of these dynamics learned intrinsically in finding the optimal value function. This is a much more practical approach for real scenarios where this information isn't readily available.

Dynamic Programming

Dynamic programming (DP) refers to a collection of algorithms that are conceptually very important in understanding any RL problem, but that are limited in that they assume a perfect model of the environment dynamics (i.e. model-based) and are computationally very expensive. Exact solutions are found for the system of equations that govern a MDP through *bootstrapping* (i.e. using an estimate of a value to get closer to its true value), without the agent really "learning" anything.

Value iteration is an algorithm whereby the optimal value function is iteratively found by taking the maximum value over all actions at every step using Bellman's equation and subsequently updating the policy at the end to its optimal form. Alternatively, *policy iteration* involves evaluating an initially random policy by finding its value function and improving it by taking the best action (as per the found value function) for all states, repeating this process until a stability condition is satisfied.

Monte-Carlo Control

Monte-Carlo (MC) control [1] is a model-free RL algorithm that addresses the limitation of dynamic programming of requiring access to the state-transition probabilities. It relies on sampling to calculate the value of a state or state-action pair from the average empirical return seen for it across all samples. The update rule on $Q(s, a)$ is thus:

$$Q(s, a) \longleftarrow Q(s, a) + \alpha \left[G_t - Q(s, a)\right] \tag{2.10}$$

where α is the learning rate and G_t the average total discounted reward at time t staring at state s.

By the law of large numbers, the state-action value function converges to its optimal value in the limit of infinite samples and thus the optimal policy can be found. However, sampling presents a lot of variance so this algorithm could take a long time to converge.

Temporal-Difference Learning

Temporal-Difference learning (TD-learning) is a model-free idea that also uses the concept of bootstrapping to update the values of the states. Similarly to Monte-Carlo control the TD target is sampled, but it also includes an estimate of $Q(S_{t+1}, A_{t+1})$ as in dynamic programming. *SARSA* [4] is an on-policy (i.e. uses a behaviour policy to estimate the next state-action value) TD control algorithm with the update step:

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{2.11}$$

where γ is the discount factor. Q-learning [5] is the off-policy alternative to SARSA, approximating the optimal state-action value function independently from the policy being followed. Rather, it uses $Q(S_{t+1}, A_{t+1}) = \max_a Q(S_{t+1}, a)$ for the TD target:

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{2.12}$$

The presence of bootstrapping comes with the caveat of adding bias to the update targets, since these will be dependent on the starting values of the states. This can potentially lead to instability in training, with value over-estimations often occurring in the case of Q-learning due to the $\max$ operator. It is also worth mentioning the trade-off that there exists between *exploration* and *exploitation* in RL problems. That is, we want to explore as much of the state space as possible to allow all states to converge to their optimal values before always choosing *greedy* actions (i.e. those with the highest value). A way of managing this trade-off is using ϵ-greedy policies that select greedy actions with probability $1 - \epsilon$ and random actions with probability ϵ.

2.1.2 Deep Reinforcement Learning

Traditional reinforcement learning algorithms suffer from the *curse of dimensionality* [6]. This phenomenon describes the problem of obtaining reliable results when dealing with a high-dimensional state space, arising from the fact that the amount of data needed to achieve a result increases exponentially with dimensionality. Using a non-linear function approximator such as a deep neural network with parameters θ, we are able to estimate the optimal state-action value function (or policy directly) as $Q(s, a; \theta) \approx Q^*(s, a)$ for very complex and large state and action spaces.

Deep Q-Networks

The concept of a Deep Q-Network (DQN) [7] was introduced by DeepMind in a successful attempt to have an artificial agent achieve better performance than humans on seven Atari 2600 games. It consists of training a deep neural network with parameters θ to estimate the optimal state-action value function by minimising the mean-squared error (MSE) with a TD target. For a given optimisation step i, this loss can be expressed as:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \rho}\left[(y_i - Q(s, a; \theta_i))^2\right] \tag{2.13}$$

where $y_i = r + \gamma \max_{a'} Q(s', a'; \hat{\theta})$ is the TD target and ρ is the behaviour distribution from which new states are sampled. The parameters θ are then updated accordingly using stochastic gradient descent. It is important to note how $\max_{a'} Q(s', a')$ is calculated using a different

network with parameters $\hat{\theta}$, which are a copy of θ from a number of This is done to stabilise training, since $Q(s,a)$ and $Q(s',a'), a' \in \mathcal{A}$ would be largely correlated otherwise. Moreover, a novel concept introduced in [8] and used in DQN's is that of *experience replay*. This consists of randomly sampling previous transitions from a buffer of saved examples, preventing learned information from being forgotten and smoothing the training distribution by reducing the correlation between samples in the same training step. In combination with using a target network, this adjustment dramatically improves the performance and stability of the DQN [9].

A variation for the DQN is the Double Deep Q-Network (DDQN) [10], which decouples the way in which actions are selected and evaluated by using the target and behaviour networks separately for these tasks. Doing this diminishes the overestimation of values resulting from the $\max$ operator being used for the TD target.

Policy Gradients

Policy gradients are useful for problems dealing with continuous action spaces, which are commonly found in reality. In policy gradient methods, we learn a parameterised function for the policy directly, written as $\pi_\theta(a \,|\, s)$, in terms of θ. To do this, a neural network is optimised to maximise the reward function:

$$\begin{aligned} J(\theta) &= \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) \\ &= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a \,|\, s) Q^\pi(s,a) \end{aligned} \tag{2.14}$$

where $d^\pi(s)$ is the stationary probability of s (i.e. the probability of s having started from an initial state s_0 under π_θ). Since $J(\theta)$ is a function of π_θ, we can use gradient ascent to perform steps on θ in the direction of $\nabla_\theta J(\theta)$. However, this gradient depends on the unknown effect of the policy parameters on the state distribution $d^\pi(s)$, which is a function of the environment. The *policy gradient theorem* [11] simplifies this problem by giving an analytic expression for the gradient excluding $\nabla_\theta d^\pi(s)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[Q^\pi(s,a) \nabla_\theta \ln \pi_\theta(a \,|\, s) \right] \tag{2.15}$$

which results in an update rule that presents no bias but a large variance, the latter commonly alleviated through the use of the advantage $A(s,a) = Q(s,a) - V(s)$ in place of $Q(s,a)$. One of the first policy gradient methods was REINFORCE [1], an on-policy algorithm which relies on finding the episodic expected value of the return through Monte-Carlo sampling to optimise θ - exploiting the fact that the expected value of the sampled gradient is equal to the true gradient.

Trust Region Policy Optimisation

As has been discussed, one of the biggest drawbacks of policy gradient methods is the presence of large variance and thus instability in training; Trust Region Policy Optimisation (TRPO) [12] addresses this issue by avoiding updates of the policy parameters that lead to very large changes in the policy. Specifically, it does this by enforcing the KL-divergence

between old and new policies to be smaller than a threshold δ, called the *trust region constraint*. In this manner, TRPO is able to guarantee monotonic improvement over policy iteration.

Need to say more about TRPO since it is essential to my project.

Other Methods

Actor-critic methods [13] attempt to learn both the value function and the policy, using the value function to address large variances in training (usually in an on-policy fashion). Specifically, at each step in training, the critic performs gradient ascent on the value function parameters ϕ, and the actor subsequently updates the policy parameters θ in the same direction as the critics gradient. The Soft Actor-Critic (SAC) [14, 15] algorithm provides an off-policy rendition of this idea with the objective of improving sampling efficiency by re-using past trajectories. It also offers a slight variation in that the policy is optimised to maximise both the expected return and the entropy $\mathcal{H}$ to promote exploration. This objective can be expressed as:

$$J(\theta) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}} \left[r(s,a) + \alpha \mathcal{H}\left(\pi_\theta(\cdot \,|\, s_t)\right] \right. \tag{2.16}$$

where ρ_{π_θ} is the state-action marginal distribution induced by the policy π_θ. It is worth mentioning that we want the policy to be tractable, so in practice we restrict the policy to a set $\mathit{\Pi}$ (e.g. Gaussian distributions). We then update it through the minimisation of the Kullback-Leibler (KL) divergence between $\pi_\theta(\cdot \mid s)$ and the normalised exponential of the updated actor network $Q_\phi(s,a)$. Another off-policy algorithm that uses the actor-critic framework is Deep Deterministic Policy Gradients (DDPG) [16]. Its objective is to learn a deterministic policy for continuous action spaces by learning both a Q-function $Q_\phi(s,a)$ (through the framework of a DQN) and a policy $\pi_\theta(a\,|\,s)$ that maximises $Q_\phi(s,a)$.

2.1.3 Distributional Reinforcement Learning

Traditionally, and when not subject to constraints, the objective of RL agents is to maximize the value function across the state space to obtain the optimal policy. In *distributional reinforcement learning* [17] (DRL), this objective changes to learning the distribution of values for all state-action pairs. This approach provides a range of auxiliary statistics (e.g. the variance in the return intrinsic to the randomness of an MDP) that aid in providing a more natural framework for inductive bias

DRL focuses on optimizing the distribution of the random return $Z^\pi(s,a)$ having started in state s and taken action a under policy π, whose expectation value is the state-action value function $Q^\pi(s,a)$. It can also be expressed as a recursive equation using the *distributional Bellman equation*:

$$\mathcal{T}^\pi Z(s,a) \overset{D}{:=} R(s,a) + \gamma Z(s',a') \tag{2.17}$$

where $\mathcal{T}^\pi : \mathcal{Z} \longrightarrow \mathcal{Z}$, with $\mathcal{Z}$ being the space of action-value distributions, is the distributional Bellman operator, and $s' \sim P(\cdot \mid s,a), a' \sim \pi(\cdot \mid a')$. This equation is characterised in terms of three independent random variables: the reward $R(s,a)$, the next state-action pair (s',a'), and its return $Z(s',a')$.

Quantile Networks

It can be shown that the distributional Bellman operator $\mathcal{T}^\pi$ is a γ-contraction in the Wasserstein distance $\mathcal{W}$ with a unique fixed point at Z^π [17]. This metric is useful because it poses an update rule analogous to the target in TD methods that converges to Z^π when $\mathcal{W}$ is minimised. However, this can't be done generally through stochastic gradient descent because the minimum expected sample Wasserstein loss is different from its true value [18], which complicates the task at hand.

Quantile Regression DQN (QRDQN) [18] aims to optimise the return distribution through its parameterization into a discrete set of N Dirac delta distributions whose locations in $R \in \mathbb{R}$ are adjusted using quantile regression to be able to minimise $\mathcal{W}$ easily. The Implicit Quantile Network (IQN) [19] builds upon this idea and reparameterizes a base distribution $\tau \sim U([0,1])$ (the implicit return distribution and prior) to fit the quantile values of Z^π; in this manner, Z^π can be learned without imposing any assumptions on its parameterization. Moreover, in [19] it was found that increasing the number of quantiles led to faster learning, corroborating that the added benefit of DRL algorithms lies in their effect as auxiliary loss functions, stabilising training and serving as regularizers.

In practice, we approximate the return distribution through a neural network $y_i(\theta, s, a)$ parameterized by θ and with N outputs indicating the number of quantiles τ [20]. We then attempt to maximise the likelihood:

$$P(D\,|\,\theta) = \prod_{j=1}^{K} \prod_{i=1}^{N} f_{\tau_i}(z_j - y_i(\theta, s, a)) \tag{2.18}$$

where $f_{\tau_i}(u) = \frac{\tau(1-\tau)}{\sigma_D} \exp(-\frac{\rho_\tau(u)}{\sigma_D})$. Here σ_D is a characteristic length scale and $\rho_\tau(u) = u \times (\tau_i - \mathbb{1}_{u<0})$. We can finally approximate the posterior distribution $P(\theta\,|\,D)$ from the likelihood and prior through Bayesian inference by using Markov Chain Monte-Carlo (MCMC) methods [21].

Estimating Risk and Uncertainty

Any model-free RL agent faces two types of uncertainty: that arising from having limited knowledge about the model governing the dynamics of the environment (*epistemic uncertainty*), and that stemming from the stochasticity present in any MDP (*aleatoric uncertainty*). However, these are in practice difficult to quantify individually because the variance of the quantiles in Z for out of distribution data can be affected by them both.

In [20] they propose a way of disentangling these two types of uncertainty and using them individually to tune the risk-adversity of the agent (using aleatoric uncertainty) and optimise exploration (through epistemic uncertainty); the former purely in the sense of maximising rewards, without considering constraints in the MDP. Following the framework of quantile networks described previously, aleatoric uncertainty $\sigma^2_{\text{aleatoric}}$ can be defined as the variance of the expected value of the quantiles taken from $P(\theta\,|\,D)$ over θ:

$$\sigma^2_{\text{aleatoric}} = \text{var}_{i\sim\mathcal{U}\{1,N\}}[\mathbb{E}_{\theta\sim P(\theta|D)} y_i(\theta, s, a)] \tag{2.19}$$

where $\mathcal{U}\{1, N\}$ is the uniform distribution over $\{1, N\}$.

Moreover, the epistemic uncertainty can then be thought of as expected value of the variance of the quantiles over θ:

$$\sigma^2_{\text{epistemic}} = \mathbb{E}_{i\sim\mathcal{U}\{1,N\}}[\text{var}_{\theta\sim P(\theta|D)}(y_i(\theta, s, a))] \tag{2.20}$$

In the limit of infinite data, when the posterior is concentrated around a single value, all of the variance in the quantile values is aleatoric, while in the absence of data all of the uncertainty is epistemic. In practice, we estimate these error sources by using two auxiliary neural networks that draw samples from $P(\theta \mid D)$ using MAP sampling [22], and use them to perform information-directed exploration by, for example, penalising actions with a high aleatoric uncertainty or foment exploration around states with a large epistemic uncertainty [20].

2.1.4 Constrained Reinforcement Learning

As described in the introduction of this investigation, it is very difficult to think about a reinforcement learning scenario in nature without considering constraints. It is also very important to consider situations where a system coexists physically with humans; in this case, it should satisfy the set of constraints that ensures maximal safety for the humans around it.

It is common for constrained reinforcement learning (CRL) algorithms to introduce safety constraints in the context of a constrained MDP (CMDP), which additionally maps transition tuples to costs $c(s, a) \in \mathbb{R}$ in a way analogous to rewards, and adds a safety threshold d to the formulation of a regular MDP [23]. The objective function of the agent becomes to maximise rewards such that the safety threshold isn't surpassed by the cost function, expressed as:

$$\begin{aligned} &\max_{\pi} \mathbb{E}_{(s,a)\sim\rho_\pi}[\sum_t \gamma^t r(s, a)], \\ &\text{s.t. } \mathbb{E}_{(s,a)\sim\rho_\pi}[\sum_t \gamma^t c(s, a)] \leq d. \end{aligned} \tag{2.21}$$

The constraint aspect of CMDP's are then usually solved using the Lagrange multiplier method, transforming Eq. (2.21) into the unconstrained dual problem:

$$\min_{\lambda\geq 0} \max_{\theta} L(\theta, \lambda) = J_R^{\pi_\theta} - \lambda(J_C^{\pi_\theta} - d) \tag{2.22}$$

where $J_R^{\pi_\theta} = \mathbb{E}_{(s,a)\sim\rho_\pi}[\sum_t \gamma^t r(s, a)]$ and $J_C^{\pi_\theta} = \mathbb{E}_{(s,a)\sim\rho_\pi}[\sum_t \gamma^t c(s, a)]$.

Constrained Policy Optimization

A way of applying constraints to a RL problem is by mapping transition tuples to costs in a way analogous to what is done with rewards in regular MDP's. Constrained Policy Optimisation (CPO) [24] uses the framework of a CMDP, whereby policies are restricted to a set $\pi \in \mathit{\Pi}_\theta$ that must satisfy the following relations:

- the cost "returns" $J_{C_i}(\pi)$ (that include the constraints we want to impose on the agent) must be smaller than a cost limit $d_i, \forall i$ for a given state s:

$$J_{C_i}(\pi_k) + \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi_k}, a\sim\pi}[A^{\pi_k}_{C_i}(s, a)] \leq d_i \quad \forall i \tag{2.23}$$

where $A^{\pi_k}_{C_i}$ denotes the cost advantage for constraint i of being in state s and taking action a under policy π_k, and d^{π_k} is the discounted future state distribution $d^{\pi}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s\,|\,\pi)$.

- a *trust region* inspired by trust region optimization methods (see Section 2.1.3) that limits the expected value of the KL divergence between the old and new policies at state s to a step size δ:

$$\mathbb{E}_{s\sim\pi_k}[\overline{D}_{KL}(\pi \,||\, \pi_k)[s]] \leq \delta \tag{2.24}$$

By doing this, we are able to perform a policy update that guarantees monotonic performance improvements and constraint satisfaction. The policy update can be expressed as:

$$\pi_{k+1} = \arg\max_{\pi\in\Pi_\theta} \mathbb{E}_{s\sim d^{\pi_k}, a\sim\pi}[A^{\pi_k}(s,a)] \tag{2.25}$$

which is very costly computationally for solutions requiring high-dimensional parameter spaces like neural networks. For small step sizes δ, however, approximating the returns and costs becomes computationally feasible by linearizing around π_k and performing policy updates that resemble primal-dual methods [24]. As a final note, it is is worth mentioning that this algorithm presents a limitation in that intermediary policies are not required to satisfy the constraints, but rather only the converged policy is.

State-Augmented Markov Decision Processes

Another way of applying constraints to a RL problem is by augmenting the state-space and reshaping the objective slightly by adding a budget for the number of constraint violations that can occur. Safety Augmented (SAUTE) Markov Decision Processes [25] do this effectively in a "plug-and-play" approach that can be applied to most RL algorithm seamlessly. The state-space is augmented through the variable $z_t \in \mathcal{Z}$ defined by the Markovian relation:

$$z_{t+1} = (z_t - l(s_t, a_t))/\gamma_l \tag{2.26}$$

where $z_0 = d$, $l(s_t, a_t) \in [0, +\infty]$ is the safety cost associated with state-action pair (s_t, a_t), and $\gamma_l \in [0, 1]$ is the safety discount factor. With this in mind, we define a SAUTE MDP in a slightly different way than a CMDP (which is represented by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \tilde{c}, \gamma_c)$) through the tuple $(\tilde{\mathcal{S}}, \mathcal{A}, \tilde{\mathcal{P}}, \tilde{c}_n, \gamma_c, \gamma_l)$, where $\tilde{\mathcal{S}} : \mathcal{S} \times \mathcal{Z}$; $\tilde{\mathcal{P}} : \tilde{\mathcal{S}} \times \mathcal{A} \times \tilde{\mathcal{S}} \longrightarrow [0, 1]$, γ_c is the cost discount factor and $\tilde{c}_n(s_t, z_t, a_t)$ is the cost expressed as:

$$\tilde{c}_n(s_t, z_t, a_t) = \begin{cases} c(s_t, a_t) & z_t \geq 0, \\ n & z_t < 0. \end{cases} \tag{2.27}$$

where n is a constant. We thus express the objective function in terms of a minimisation of costs rather than a maximisation of rewards:

$$\min_{\pi} \mathbb{E} \sum_{t=0}^{\infty} \gamma_c^t \tilde{c}_n(s_t, z_t, a_t) \tag{2.28}$$

and it can be proved that there exists a value function that converges monotonically to the optimal value function [25]. Finally, it is worth mentioning that a limitation of this approach

is that the state-space increases with the number of constraints, which could lead to scalability issues. In this regard, Lagrangian-based methods can be more sample efficient.

Inventory Management Problem

3.1 Formulation

Developing the Framework

Use `this` style to name modules!

Distributional Constrained Policy Optimization

5.1 Distributional Value Function and Safety Baseline

5.2 Cost Reshaping

Let $\rho_v = \beta\mathbb{P}(J_C > d)$ and $\rho_s = -\beta\mathbb{P}(J_C < d)$ denote the refactoring parameters when the constraints are being violated and satisfied, respectively. A general equation for the refactoring parameter ρ can then be expressed as:

$$\begin{aligned}\rho &= \begin{cases} \rho_v, & J_C > d \\ \rho_s, & J_C < d \end{cases} \\ &= \begin{cases} \beta\mathbb{P}\left(J_C > d\right)), & J_C > d \\ -\beta\mathbb{P}\left(J_C < d\right) & J_C < d \end{cases}\end{aligned} \tag{5.1}$$

An expression for the refactored costs and target is then:

$$\tilde{J}_C = J_C\left(1 + \text{clip}\left(\rho, -\epsilon, \infty\right)\right) \tag{5.2}$$

$$\tilde{d} = d\left(1 + \text{clip}\left(\rho, -\epsilon, \infty\right)\right) \tag{5.3}$$

Hey

Proof. why this works: □

5.3 From Progress Report...

We define the objective of this investigation as the approximation of the optimal distributions of costs and returns as per the Bellman optimality equations to obtain the optimal policy in situations where constraint satisfaction is of high importance. By approximating distributions of both costs and returns, we are able to exploit the more natural framework for inductive bias [17] that is presented by DRL to optimise exploration and minimise the number of constraint violations.

Algorithm 1: Distributional Constrained Policy Optimization

Input: Initial policy $\pi_0 \in \Pi_\theta$, safety margin ϵ, refactoring coefficient β
for $k = 0, 1, 2, ...$ **do**
  Sample a set of trajectories $\mathcal{D} = \tau \sim \pi_k = \pi(\theta_k)$
  Obtain reshaped constraint $\tilde{J}_C(\pi_k)$ and target $\tilde{d}$
  Form sample estimates $\hat{g}$, $\hat{b}$, $\hat{H}$, $\hat{c}$ using $\mathcal{D}$, $\tilde{J}_C(\pi_k)$ and $\tilde{d}$
  if *approximate CPO is feasible* **then**
    Solve dual problem (Eq. ref) for λ_k^*, ν_k^*
    Compute policy proposal θ^* with (Eq. ref)
  else
    Compute recovery policy proposal θ^* with (Eq. ref)
  end
  Obtain θ_{k+1} through backtracking linesearch to enforce satisfaction of sample estimates of constraints in (Eq. ref)
end

For a deep RL setting and with the distributional RL framework at our disposal, we are able to make the auxiliary loss used to optimise both the policy and returns (through actor-critic methods) a function of both the cost and reward distributions. These will be parameterised as quantile functions as per [19]; the added benefit this provides in terms of regularisation and improved training stability alone is reason for investigation. In addition, through estimations of epistemic and aleatoric uncertainties of both distributions [20], we should be able to parameterise exploration in terms of risk-adversity for safety and maximisation of returns.

We will attempt to solve the constraint satisfaction problem using this idea in combination with both the SAUTE [25] formulation of a CMDP and also by applying constraints through the Lagrangian multiplier method, and compare the two.

Evaluation

6.1 Distributional Constrained Policy Optimization

6.1.1 The Effect of N

6.1.2 The Effect of β

6.1.3 Ablation of Cost Reshaping

6.2 Performance Against State-of-the-Art Algorithms

Conclusion

7.1 Future Work

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html. pages 4, 6, 7, 9

[2] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961. pages 4

[3] Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958. doi: 10.1016/s0019-9958(58)80003-0. pages 7

[4] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, 1994. pages 8

[5] Christopher J. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. doi: 10.1007/bf00992698. pages 8

[6] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957. ISBN 9780486428093. pages 8

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602. pages 8

[8] Longxin Lin. Reinforcement learning for robots using neural networks. 1992. pages 9

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015. doi: 10.1038/nature14236. pages 9

[10] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015. URL https://arxiv.org/abs/1509.06461. pages 9

[11] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In

Proceedings of the 12th International Conference on Neural Information Processing Systems, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press. pages 9

[12] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015. URL https://arxiv.org/abs/1502.05477. pages 9

[13] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In *SIAM Journal on Control and Optimization*, pages 1008–1014. MIT Press, 2000. pages 10

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290. pages 10

[15] Lilian Weng. Policy gradient algorithms. *lilianweng.github.io*, 2018. URL https://lilianweng.github.io/posts/2018-04-08-policy-gradient/. pages 10

[16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015. URL https://arxiv.org/abs/1509.02971. pages 10

[17] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning, 2017. URL https://arxiv.org/abs/1707.06887. pages 10, 11, 17

[18] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression, 2017. URL https://arxiv.org/abs/1710.10044. pages 11

[19] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning, 2018. URL https://arxiv.org/abs/1806.06923. pages 11, 18

[20] William R. Clements, Bastien Van Delft, Benoît-Marie Robaglia, Reda Bahi Slaoui, and Sébastien Toth. Estimating risk and uncertainty in deep reinforcement learning, 2019. URL https://arxiv.org/abs/1905.09638. pages 11, 12, 18

[21] Don van Ravenzwaaij, Pete Cassey, and Scott D. Brown. A simple introduction to markov chain monte-carlo sampling. *Psychonomic Bulletin Review*, 25(1):143–154, February 2018. ISSN 1069-9384. doi: 10.3758/s13423-016-1015-8. pages 11

[22] Tim Pearce, Felix Leibfried, and Alexandra Brintrup. Uncertainty in neural networks: Approximately bayesian ensembling. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 234–244. PMLR, 26–28 Aug 2020. URL https://proceedings.mlr.press/v108/pearce20a.html. pages 12

[23] Hengrui Zhang, Youfang Lin, Sheng Han, Shuo Wang, and Kai Lv. Conservative distributional reinforcement learning with safety constraints, 2022. URL https://arxiv.org/abs/2201.07286. pages 12

[24] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization, 2017. URL https://arxiv.org/abs/1705.10528. pages 12, 13

[25] Aivar Sootla, Alexander I. Cowen-Rivers, Taher Jafferjee, Ziyan Wang, David Mguni, Jun Wang, and Haitham Bou-Ammar. Saute rl: Almost surely safe reinforcement learning using state augmentation, 2022. URL https://arxiv.org/abs/2202.06558. pages 13, 18