Generalization and Transferability in Reinforcement Learning

Generalisierung und Transfer in Reinforcement Learning
Master-Thesis von Pascal Klink aus Weinheim
März 2019

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generalization and Transferability in Reinforcement Learning
Generalisierung und Transfer in Reinforcement Learning

Vorgelegte Master-Thesis von Pascal Klink aus Weinheim

1. Gutachten: M.Sc. Hany Abdulsamad
2. Gutachten: M.Sc. Boris Belousov
3. Gutachten: Prof. Jan Peters

Tag der Einreichung:

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Pascal Klink, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: Unterschrift / Signature:

_____ _____

Abstract

Reinforcement learning has proven capable of extending the applicability of machine learning to domains in which knowledge cannot be acquired from labeled examples but only via trial-and-error. Being able to solve problems with such characteristics is a crucial requirement for autonomous agents that can accomplish tasks without human intervention. However, most reinforcement learning algorithms are designed to solve exactly one task, not offering means to systematically reuse previous knowledge acquired in other problems. Motivated by insights from homotopic continuation methods, in this work we investigate approaches based on optimization- and concurrent systems theory to gain an understanding of conceptual and technical challenges of knowledge transfer in reinforcement learning domains. Building upon these findings, we present an algorithm based on contextual relative entropy policy search that allows an agent to generate a structured sequence of learning tasks that guide its learning towards a target distribution of tasks by giving it control over an otherwise hidden context distribution. The presented algorithm is evaluated on a number of robotic tasks, in which a desired system state needs to be reached, demonstrating that the proposed learning scheme helps to increase and stabilize learning performance.

Zusammenfassung

Reinforcement Learning hat sich als Schlüsseltechnik erwiesen, die Anwendbarkeit des maschinellen Lernens auf Bereiche zu erweitern, in denen Wissen nicht anhand von Beispielen, sondern nur durch eigenständige Exploration erworben werden kann. Die Fähigkeit solche Probleme lösen zu können, ist eine entscheidende Eigenschaft autonomer Agenten, die Aufgaben ohne menschliches Zutun erfüllen. Die meisten Reinforcement Learning Algorithmen sind jedoch für das Lösen genau einer Aufgabe konzipiert und bieten deshalb keine Möglichkeit zur systematischen Wiederverwendung bereits erlernten Wissens. In dieser Arbeit untersuchen wir - motiviert durch Erkenntnisse aus homotopischen Continuation Methoden - Ansätze der Optimierungs- und nebenläufigen Systemtheorie, um ein Verständnis für die konzeptionellen und technischen Herausforderungen des Wissenstransfers im Reinforcement Learning zu erlangen. Aufbauend auf diesen Erkenntnissen präsentieren wir einen Algorithmus basierend auf Contextual Relative Entropy Policy Search, der einem Agenten die Generierung einer strukturierten Sequenz von Lernaufgaben, die sein Lernen auf eine Zielverteilung von Aufgaben lenkt, ermöglicht, indem er ihm die Kontrolle über eine ansonsten stationäre Kontextverteilung gibt. Der vorgestellte Algorithmus wird anhand verschiedener Robotikaufgaben evaluiert, in denen ein vorgegebener Systemzustand erreicht werden muss. Die Ergebnisse zeigen, dass das vorgeschlagene Lernschema hilft, die Lernleistung zu erhöhen und zu stabilisieren.

Acknowledgments

I would like to thank my supervisors M.Sc. Boris Belousov and M.Sc. Hany Abdulsamad for their time, fruitful discussions as well as encouraging and motivating me to pursue my ideas throughout the thesis, without which this would have not been such an insightful experience.
Furthermore, I want to thank Prof. Jan Peters, Head of the Intelligent Autonomous Systems (IAS) group, for sparking my interest in the topics of Machine Learning and Reinforcement Learning with his excellent lectures.
At this point, I also want to thank all IAS team members for their efforts to support and motivate their students in lectures and projects as well as the pleasant atmosphere throughout my work as a student assistant at the IAS group.

Contents

Figures and Tables

List of Figures

List of Tables

1 Introduction

Autonomous robots and machines promise to remove the need for human labor in disaster areas, hazardous workplaces and other environments that, although holding many dangers for life and well-being, still require human presence. While the successful realization of such machines can drastically reduce the number of incidents and deaths accounted to these environments, equipping machines with the necessary learning capabilities unfortunately turns out to be a very complex problem.

Reinforcement Learning (RL) has allowed to create agents that are capable of learning a variety of games to a superhuman level [1, 2, 3] and achieving complicated robotic control tasks through trial-and-error [4, 5, 6]. Those successes make RL be regarded as a key technology for equipping robots with the required autonomy to help or replace humans in aforementioned areas.

With increasing computational capabilities and the availability of powerful function approximators, RL methods are being applied to more and more complicated problems. However, there exist crucial mismatches between the way we humans learn compared to RL methods, one of them being that most methods are designed to learn one specific task, not allowing to effectively reuse knowledge from related learning tasks in novel scenarios. While so called transfer learning for RL [7] aims at providing RL agents with those missing capabilities, this remains an open problem as many transfer learning approaches employ specific assumptions that limit their general applicability. Nonetheless, a generally applicable approach that allows RL agents to subsequently choose learning tasks and transfer acquired knowledge between them would be a crucial step towards artificial intelligence that truly deserves its name. Results from cognitive science [8] underline this finding, where head cameras and eye-trackers were used to capture images of the infant's learning environment during their first two years of life, revealing that this environment gradually changes with the increasing capabilities of the infant learner, creating an implicit learning curriculum that provides the infant tasks of increasing complexity.

In this work, we will create such a scheme of transfer learning, in which the learning tasks of an agent progress from easier to more complicated ones, by gaining insights into so called homotopic predictor-corrector methods - a form of numerical continuation methods - that allow to find roots of parameterized systems of linear equations by finding roots for specific parameters and then follow these roots as the parameters change. This approach is especially appealing, if direct root finding approaches fail for specific single parameters.

Motivated by this idea of finding a chain of solutions to problems of increasing complexity, we then investigate how to apply this idea to the domain of RL, where we do not need to solve a root finding- but a constraint optimization problem. More specifically, we will investigate sensitivity analysis and bisimulation metrics. Sensitivity analysis [9] is a well-known concept from multi-parametric programming [10] that allows to compute how the solution of an optimization problem changes with respect to its parameters. Bisimulation metrics [11, 12] are approaches to quantify the similarity of states in Markov Decision Processes (MDPs) by extending the concept of bisimulation known for concurrent processes [13].

While providing promising insights into the complexity of the problem that we aim to solve in this thesis, both of the aforementioned methods suffer from technical and practical limitations that do not allow to use them in a straightforward manner for our purpose. Because of this, we ultimately take an episodic view of the RL problem by extending the so called Contextual Relative Entropy Policy Search (C-REPS) algorithm [14, 15, 16] - an episodic policy search algorithm which allows to generalize a policy over a set of reinforcement learning problems that are related by a context variable. We extend this algorithm to allow the RL agent to control the, otherwise fixed, context distribution and introduce a penalty term that allows to increasingly force the context distribution of the learning agent to match the desired one. While this is conceptually straightforward, we will see that some ingenuity is necessary to allow for a numerically stable solution of the resulting optimization problem.

We then evaluate our algorithm on three different robotic goal reaching tasks. The experiments show that our algorithm indeed uses the explicit control over the context distribution to initially focus on easier contexts and then progresses towards the desired context distribution as we increase the penalty multiplier. This allows our method to solve tasks in which the default C-REPS algorithm cannot demonstrate satisfying performance.

Finally, we discuss related work and conclude the thesis by investigating possibilities for future work.

2 A Motivating Example: Numerical Continuation Methods

To motivate our efforts in understanding sequential learning and knowledge transfer in reinforcement learning domains as well as the algorithm that will be presented in this thesis, we want to look at so called homotopic predictor-corrector methods, a specific class of numerical continuation algorithms [17], which are used to solve non-linear equation systems

$$F_i(\mathbf{x}) = 0, \quad i = 1, \dots, m, \quad \mathbf{x} \in \mathbb{R}^n.$$

A common tool for such a task is the so called newton method [18], which is able to find roots of $F(\mathbf{x})$ by computing a sequence of points $\mathbf{x}_i$ obeying

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{x}_i - F'(\mathbf{x}_i)^\dagger F(\mathbf{x}_i) \\ &= \mathbf{x}_i - \left(F'(\mathbf{x}_i)^T F'(\mathbf{x}_i)\right)^{-1} F'(\mathbf{x}_i)^T F(\mathbf{x}_i). \end{aligned} \tag{2.1}$$

In above equation, $F'(\mathbf{x}_i) \in \mathbb{R}^{m \times n}$ is the Jacobian of $F(\mathbf{x}_i) \in \mathbb{R}^m$ and $\mathbf{A}^\dagger \in \mathbb{R}^{L \times K}$ the Moore-Penrose Inverse of an arbitrary matrix $\mathbf{A} \in \mathbb{R}^{K \times L}$ [19][20]. Note that the use of the Moore-Penrose Inverse is already a generalization of the original newton method, which allows for its application to over- or underdetermined systems of non-linear equations, i.e. if $m \neq n$.
While the sequence $\mathbf{x}_i$ - if convergent - quadratically approaches a root of $F(\mathbf{x})$, this convergence is not guaranteed for all initial values $\mathbf{x}_0$. The three scalar functions f, g and h shown in figure 2.1 help to illustrate this problem. While the newton method would robustly find one of the roots of the left-most function $f(x)$, this task becomes more and more brittle for the middle and right functions $g(x)$ and $h(x)$, as for more and more initial values the gradient $F'(\mathbf{x}_0)$ in equation 2.1 becomes 0 and hence the computation of $\mathbf{x}_1$ fails.
However, figure 2.1 also hints at a possible solution to this problem. The three functions $f(x)$, $g(x)$ and $h(x)$ are actually strongly related as they all arise from the function

$$H(\mathbf{x}, \lambda) = \begin{cases} \frac{1}{10^{-4c}}(\mathbf{x} - c)^4 - \frac{2}{10^{-2c}}(\mathbf{x} - c)^2 + \frac{1}{64}, & \text{if } |\mathbf{x} - c| \leq 10^{-c} \\ -\frac{63}{64}, & \text{else} \end{cases}, \; c = 4 - 5\lambda, \; \lambda \in [0, 1],$$

with $H(\mathbf{x}, 1) = f(\mathbf{x})$, $H(\mathbf{x}, 0.75) = g(\mathbf{x})$ and $H(\mathbf{x}, 0) = h(\mathbf{x})$. We call the function H a homotopy between f and h, whose parameter λ describes a continuous deformation of f into h. This also means that the roots of f are continuously deformed into roots of h as we decrease λ from 1 to 0.
It is exactly this homotopic nature of H that we can exploit to find roots of $H(\mathbf{x}, 0)$. That is, we compute a root of $H(\mathbf{x}, 1)$ using a newton-type method and then iterate between decreasing λ and re-computing the root of $H(\mathbf{x}, \lambda)$ using the previously found root as an initial guess, until we arrive at $\lambda = 0$. This scheme already corresponds to a naive version of a predictor-corrector method, where in the predictor step we choose the next value of λ and the corrector step then re-computes the root of $H(\mathbf{x}, \lambda)$ for this new value of λ. This concept is visualized in figure 2.2 for the homotopy H.
However, the figure also indicates that it is possible to improve upon this naive approach by approximating how the position $\mathbf{x}$ of the root will change as we decrease λ. In order to approximate this change, it is necessary to start thinking of the roots of $H(\mathbf{x}, \lambda)$ in a different way.

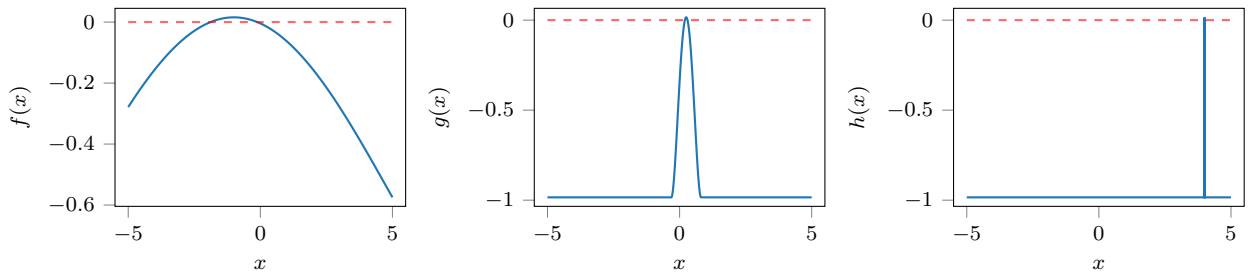

Figure 2.1.: Three functions f, g and h mapping from $\mathbb{R}$ to $\mathbb{R}$. The red dotted line shows $f(x) = g(x) = h(x) = 0$ to pronounce the roots of the functions.

2.1 Tracing Root Curves

The roots of $H(\mathbf{x}, \lambda)$, depicted by red lines in figure 2.2, actually form curves in $\mathbb{R}^{n+1}$ which are defined by the condition

$$H(\mathbf{u}) = \mathbf{0}, \quad \mathbf{u} = [\mathbf{x}\ \lambda] \in \mathbb{R}^{n+1}, H(\mathbf{u}) \in \mathbb{R}^m. \tag{2.2}$$

With above definition, a more elaborate way of thinking about predictor-corrector methods is that they trace those root curves starting from an initial point $\mathbf{u}_0 \in \mathbb{R}^{n+1}$ until they reach a point where $\lambda = 0$.
This new interpretation allows us to improve the predictor step by taking the orientation of the curve into account, which is given by its tangent at a given point $\mathbf{u}$. As shown in [17], if we parameterize the curve by the arclength (i.e. the traveled distance along the curve) we can define the tangent vector $\mathbf{t} \in \mathbb{R}^{n+1}$ using the three constraints

$$H'(\mathbf{u})\,\mathbf{t} = \mathbf{0}, \tag{2.3}$$

$$\|\mathbf{t}\| = 1, \tag{2.4}$$

$$\det \begin{pmatrix} H'(\mathbf{u}) \\ \mathbf{t}^T \end{pmatrix} > 0. \tag{2.5}$$

Condition 2.3 arises by differentiating the curve condition 2.2. Condition 2.4 ensures that the tangent vector is of constant size and finally, condition 2.5 ensures that the curve is traced in a consistent direction, i.e. that the algorithm does not arbitrarily move back and forth along the curve. Note that this last constraint implicitly requires $m = n$, i.e. that the number of constraints matches the dimension of $\mathbf{x}$ as the determinant is only defined for square matrices and $H'(\mathbf{u}) \in \mathbb{R}^{m\times(n+1)}$. For the case $m \neq n$, other conditions ensuring a consistent direction would need to be introduced. We, however, will restrict ourselves to above definition. With the notion of the tangent vector, we can now express the tracing problem as an initial value problem of the form

$$\begin{aligned} \dot{\mathbf{u}} &= \mathbf{t}\left(H\left(\mathbf{u}\right)\right) \\ \mathbf{u}(0) &= \mathbf{u}_0, \end{aligned}$$

with $H(\mathbf{u}_0) = \mathbf{0}$ and $\mathbf{t}\left(H\left(\mathbf{u}\right)\right)$ being the tangent vector of the curve at $\mathbf{u}$. By re-defining our problem of finding roots of a function as an initial value problem, it is possible to choose from a variety of solution methods for this kind of problem. However, for predictor-corrector methods, it is common to use the comparatively simple explicit euler method

$$\mathbf{u}_i = \mathbf{u}_{i-1} + h\mathbf{t}\left(H(\mathbf{u}_{i-1})\right)$$

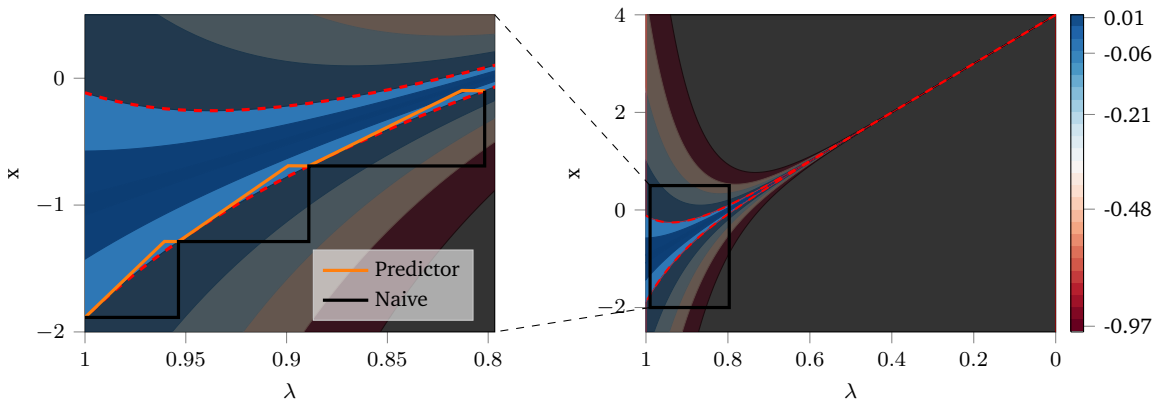

Figure 2.2.: The plots visualize the homotopy $H(\mathbf{x}, \lambda)$ for different subspaces of $\mathbb{R}^2$. The values of H are encoded using different colors and the dashed red line highlights the zero-crossings of H. The area of $\mathbb{R}^2$ where $H(\mathbf{x}, \lambda) < 0$ is shaded while the areas with $H(\mathbf{x}, \lambda) > 0$ appear in brighter colors. The left plot furthermore illustrates two different schemes of iteratively computing roots of $H(\mathbf{x}, \lambda)$ for decreasing values of λ, which differ in their way of predicting the x-position of the root of H for the next λ. The 'Naive' scheme assumes no change in position while the 'Predictor' scheme uses information about the tangent at $H(\mathbf{x}, \lambda)$ to approximate the change in position.

Input: Step size h,
Allowed prediction error ϵ_p,
Desired precision of newton-method ϵ_n (typically $\epsilon_n \ll \epsilon_p$)
Initial root guess $\mathbf{x}_0$
$\mathbf{u} = [\mathbf{x_0}\ 1]$
loop
Corrector Step: ▷ Newton method according to 2.1
while $\|H(\mathbf{u})\| > \epsilon_n$ **do**
$\mathbf{u} = \mathbf{u} - H'(\mathbf{u})^{\dagger} H(\mathbf{u})$
end while ▷ $[\mathbf{x}\ \lambda]$ does not violate 2.2 by more than ϵ_n

Predictor Step:
$\mathbf{Q}, \mathbf{R} = \text{QR}\left(H'(\mathbf{u})\right)$ ▷ QR-Decomposition of $H'(\mathbf{u})$
$\mathbf{t} = \text{sign}\left(\det(\mathbf{Q})\det(\mathbf{R})\right)\mathbf{q}_{n+1}$ ▷ t fulfills conditions 2.3, 2.4 and 2.5
$\alpha = 1$
$\mathbf{u}_{pred} = \mathbf{u} + \alpha h \mathbf{t}$
while $\|H(\mathbf{u}_{pred})\| > \epsilon_p$ **do**
$\alpha = 0.9\alpha$
$\mathbf{u}_{pred} = \mathbf{u} + \alpha h \mathbf{t}$
end while ▷ $[\mathbf{x}_{pred}\ \lambda_{pred}]$ does not violate 2.2 by more than ϵ_p

Termination Handling:
if $\lambda_{pred} < 0$ **then**
$\beta = \frac{\lambda}{\lambda - \lambda_{pred}}$ ▷ It holds that $0 = \beta\lambda_{pred} + (1-\beta)\lambda$
return $\beta\mathbf{x}_{pred} + (1-\beta)\mathbf{x}$ ▷ Interpolate between $\mathbf{x}$ and $\mathbf{x}_{pred}$ using β
else
$\mathbf{u} = \mathbf{u}_{pred}$
end if
end loop

Algorithm 1: A Basic Homotopic Predictor Corrector Method

with a given step size $h \in \mathbb{R}$, as the prediction for $\mathbf{u}_i$ is corrected by a run of the newton method at every iteration before computing $\mathbf{u}_{i+1}$ - which accounts for errors in the prediction of $\mathbf{u}_i$. This correction can be reviewed in figure 2.2, where the orange line first deviates from the curve by following the tangent at a given point and is then corrected back to the curve by the correction step. We can also see that this scheme leads to a much smaller deviation from the curve compared to the naive approach.
A practical question that arises at this point is how to compute the tangent vector $\mathbf{t}(H(\mathbf{u}))$. As detailed in [17], this is possible by a QR-Factorization of $H'(\mathbf{u})^T$

$$H'(\mathbf{u})^T = \mathbf{Q}\begin{pmatrix}\mathbf{R}\\ \mathbf{0}^T\end{pmatrix} = \begin{pmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \dots & \mathbf{q}_{n+1}\end{pmatrix}\begin{pmatrix}\mathbf{R}\\ \mathbf{0}^T\end{pmatrix}, \quad \mathbf{q}_1, \dots, \mathbf{q}_{n+1} \in \mathbb{R}^{n+1},\ \mathbf{Q} \in \mathbb{R}^{(n+1)\times(n+1)},\ \mathbf{R} \in \mathbb{R}^{n\times n}.$$

The property of $\mathbf{Q}$ that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ already guarantees $\|\mathbf{q}_i\| = 1$ for all $i \in [1, \dots, n+1]$. Furthermore it holds that

$$H'(\mathbf{u})\,\mathbf{q}_{n+1} = \begin{pmatrix}\mathbf{R}^T & \mathbf{0}\end{pmatrix}\mathbf{Q}^T\mathbf{q}_{n+1} = \begin{pmatrix}\mathbf{R}^T & \mathbf{0}\end{pmatrix}\begin{pmatrix}\mathbf{0}\\ 1\end{pmatrix} = \mathbf{0}.$$

The above equation reveals that the vector $\mathbf{q}_{n+1}$ is the tangent vector $\mathbf{t}$ we are looking for. The only thing left to ensure is that the orientation of $\mathbf{q}_{n+1}$ is consistent, i.e. that it fulfills condition 2.5. Since it holds that

$$\det\left(\begin{pmatrix}H'(\mathbf{u})\\ \mathbf{q}_{n+1}^T\end{pmatrix}\right) = \det\left(\begin{pmatrix}H'(\mathbf{u})^T & \mathbf{q}_{n+1}\end{pmatrix}\right) = \det\left(\mathbf{Q}\begin{pmatrix}\mathbf{R} & \mathbf{0}\\ \mathbf{0}^T & 1\end{pmatrix}\right) = \det(\mathbf{Q})\det(\mathbf{R}),$$

we can easily enforce this by choosing $\mathbf{t} = \pm\mathbf{q}_{n+1}$ depending whether $\det(\mathbf{Q})\det(\mathbf{R})$ is positive or negative. Having all the means to compute $\mathbf{t}(H(\mathbf{u}))$, algorithm 1 outlines an implementation of a homotopic predictor-corrector method. Note that this is still only a very basic implementation of a such a method, as we have skipped important aspects such as maxima of $H(\mathbf{x}, \lambda)$ that result in singular jacobians $H'(\mathbf{x}, \lambda)$. Nonetheless this should give a sufficient understanding of the principal concept of homotopic predictor-corrector methods.

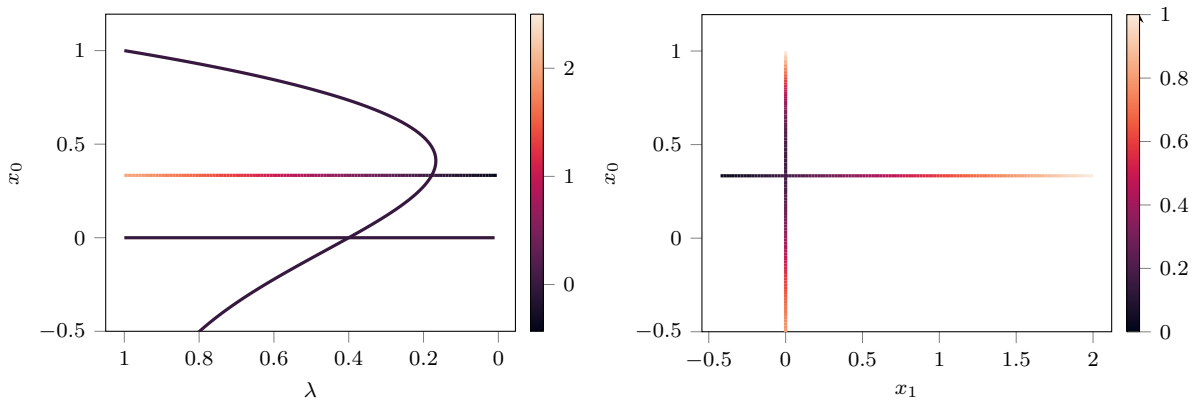

Figure 2.3.: The plots visualize the three-dimensional roots of the predator-prey model 2.6 using different projections. The roots are plotted over x_0 and λ in the left- and over x_0 and x_1 in the right plot. The third dimension (x_1 in the left plot and λ in the right one) is visualized using a color scale.

2.2 Dynamical System Analysis with Continuation Methods

The previous chapters demonstrated that homotopic predictor corrector methods allow to compute roots of a function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, for which standard newton-type approaches fail. This becomes possible by interpolating between the desired function f and another function $g : \mathbb{R}^n \mapsto \mathbb{R}^m$, for which roots can be more easily found, using a homotopy.
However, introducing an appropriate homotopy is not an easy task and although there are general purpose homotopies such as

$$H(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda f(\mathbf{x}_0),$$

where $\mathbf{x}_0$ is an initial guess for the root of f, those homotopies do not guarantee that λ can be monotonously decreased from 1 to 0 and indeed cannot guarantee that there is a curve that intersects $\lambda = 0$ after a finite arclength at all. Dynamical System Analysis, however, is a domain in which homotopies often naturally arise from e.g. open parameters of a dynamical system. Imagine a simple Predator-Prey Model

$$\begin{pmatrix} \dot{x_0} \\ \dot{x_1} \end{pmatrix} = \begin{pmatrix} 3x_0(1 - x_0) - x_0x_1 - \lambda\,(1 - \exp(-5x_0)) \\ -x_1 + 3x_0x_1 \end{pmatrix}, \tag{2.6}$$

where x_0 and x_1 are the population sizes of two animal species and the open parameter λ determines external effects that further reduce the population size x_0 of the first animal species, such as hunting. A question often asked for such systems is how the population equilibria, i.e. the roots of the system 2.6, change with regards to λ. While there are no numerical problems that prevent a newton-type algorithm from calculating those roots for individual values of λ, it is still undesirable to do that for many different values of λ individually. One reason for that being that the number of roots can actually change with the value of λ.
Numerical continuation methods naturally help to deal with such problems, as they, starting from an initial value of λ, trace the behavior of one specific root of the dynamical system as λ changes. Figure 2.3 illustrates the result of such a tracing procedure, where we started with the three roots $(x_0, x_1) = (0, 0)$, $(1, 0)$ and $(\frac{1}{3}, 2)$ of the dynamical system for $\lambda = 1$. We can see that the curve starting in $(x_0, x_1) = (1, 0)$ has a very interesting behavior as it crosses the two other curves at $\lambda \approx 0.18$ and $\lambda \approx 0.4$ and then with increasing λ takes values that are not physically sound, i.e. propose population sizes smaller than zero. Hence the investigated model has three or four physically sound equilibria, depending on the value of λ. Such a behavior would have been tedious to explore without the help of numerical continuation methods.

2.3 Continuation in Reinforcement Learning

The previous chapters showed that homotopic predictor-corrector methods can be of great benefit in root finding tasks and dynamical system analysis by either allowing to actually compute roots for a given function or making it easy to compute them for a variety of related functions.

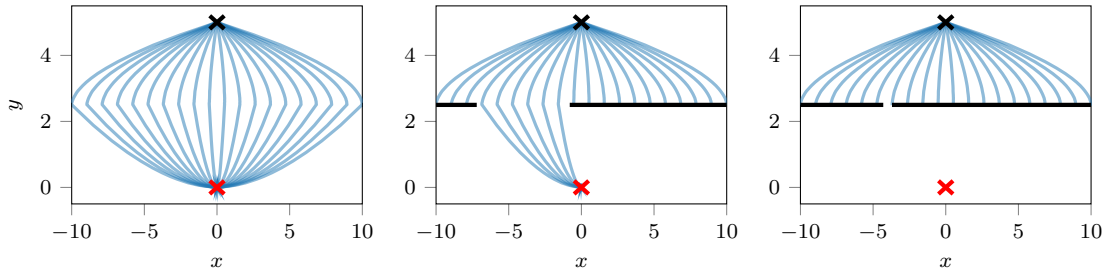

Figure 2.4.: The three plots show trajectories generated by PD-Controllers with different parameterizations $\boldsymbol{\theta}$ (blue lines) for different widths $w_\delta = 20$ (left), $w_\delta = 3$ (middle) and $w_\delta = 0.1$ (right) of the gate at position $w_x = -4$. The starting position is depicted by a black cross and the goal position by a red one.

In episodic reinforcement learning, we do not need to solve root finding problems but often express learning tasks as the maximization of an objective function $f : \mathbb{R}^n \mapsto \mathbb{R}$ - a closely related problem, as e.g. newton-type algorithms can be used to optimize a function by finding roots of its Jacobian $f' : \mathbb{R}^n \mapsto \mathbb{R}^n$.
However, the objective function f often contains many local maxima, which we want to avoid as they can encode behavior that is not able to solve the task at hand. Furthermore, many reinforcement learning algorithms do not assume knowledge about the functional form of f and hence can only gather information about it by observing its values at specific $\mathbf{x} \in \mathbb{R}^n$. To pose an example of the indicated problem, we will look at a very simple perturbed linear system

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} 5 \\ -1 \end{pmatrix} + \mathbf{u} + \boldsymbol{\delta}, \quad \mathbf{u}, \boldsymbol{\delta} \in \mathbb{R}^2, \ \boldsymbol{\delta} \sim \mathcal{N}\left(\mathbf{0}, 10^{-3}\mathbf{I}\right),$$

which models the x- and y-position of a point-mass in two dimensional space that can be controlled using the controls $\mathbf{u}$ and is subject to perturbations $\boldsymbol{\delta}$. Starting from position $(x, y) = (0, 5)$, the goal is to reach the origin $(0, 0)$. The controls $\mathbf{u}$ are generated by two PD-Controllers of the form

$$\mathrm{PD}_i(x, y) = \mathbf{K}_i \begin{pmatrix} \tilde{x}_i - x \\ \tilde{y}_i - y \end{pmatrix} + \mathbf{k}_i, \ \mathbf{K}_i \in \mathbb{R}^{2\times 2}, \ \mathbf{k}_i \in \mathbb{R}^2, \ \tilde{x}_i, \tilde{y}_i \in \mathbb{R}$$

for $i = 1, 2$. The first controller PD_1 is active until the point mass reaches the height $y = 2.5$, starting from which the second controller PD_2 becomes active. The agent now needs to find the parameters $\boldsymbol{\theta} = \{(\mathbf{K}_i, \mathbf{k}_i, \tilde{x}_i) | i = 1, 2\}$ that maximize the performance criterion

$$f(\boldsymbol{\theta}) = 10 \left(\exp\left(-\sqrt{x_f^2 + y_f^2}\right) - 10^{-5} \sum_{i=1}^{100} \mathbf{u}_i^T \mathbf{u}_i \right),$$

where x_f and y_f are the positions after 100 state transitions generated by the controller using $\boldsymbol{\theta}$ and $\mathbf{u}_i$ the generated actions. Note that we fixed $\tilde{y}_1 = 2.5$ and $\tilde{y}_2 = 0.0$ as this ensures that the aforementioned switching logic between the two controllers is sensible. While such a function is easily optimized by reinforcement learning algorithms, we now want to make this very easy task more interesting by requiring the point-mass to traverse a wall at height $y = 2.5$ through a gate at position w_x with a width of w_δ. If the point-mass crashes into the wall, the experiment terminates and x_f and y_f are set to the current position. Figure 2.4 illustrates the resulting tasks for different widths of the gate.
The position and width of the gate can be seen as additional parameters $\mathbf{c} = [w_x \ w_\delta] \in \mathbb{R}^2$ to the function f

$$f(\boldsymbol{\theta}, \mathbf{c}) : \mathbb{R}^n \times \mathbb{R}^2 \mapsto \mathbb{R}.$$

At this point, f constitutes a so called contextual reinforcement learning problem, where the context $\mathbf{c}$ modifies the reward function, representing different but related reinforcement learning problems. In the given example, it is intuitive that optimizing f is more complicated in some contexts than in others. In order to reach the origin for $\mathbf{c} = [-4 \ 0.1]$, the points mass needs to pass through $x \in [-3.9, -4.1]$ at $y = 2.5$, before progressing towards the origin, while for $\mathbf{c} = [-4 \ 20]$, this interval expands to $[-24, 16]$. Figure 2.5 shows examples of the resulting objective function f for different contexts.
We see that the presence of the wall creates discontinuities in the reward function that become more and more problematic for narrow gates. This already renders finding the global optimum of f problematic, even when its functional form is

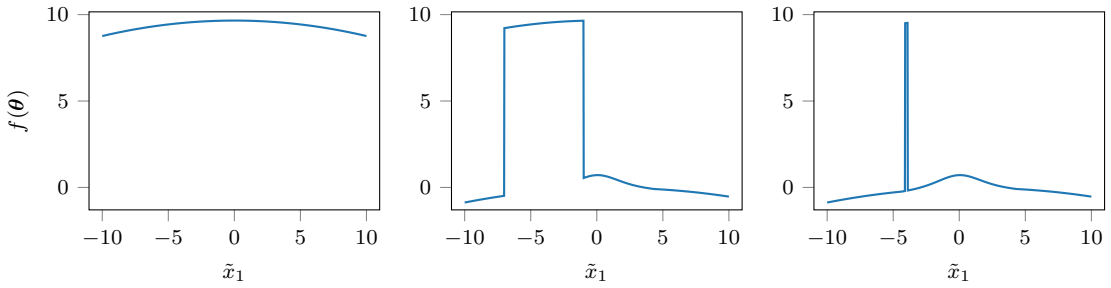

Figure 2.5.: Rewards functions for the three contexts $[-4\ 20]$ (left), $[-4\ 3]$ (middle) and $[-4\ 0.1]$ (right). The plots are generated by changing the desired x-position $\tilde{x}_1$ of the first PD Controller while keeping the remaining parameters fixed. Those remaining parameters are chosen such that the resulting controllers are capable of steering the point-mass through the gate and then towards the origin, if $\tilde{x}_1$ is chosen appropriately. Indeed, the parameters were used to generate the trajectories in figure 2.4.

known. As already mentioned, the optimization procedures in reinforcement learning problems are even more delicate in the sense that there is no initial knowledge about the functional form of f. Hence values of f can only be observed at specific instances of $\boldsymbol{\theta}$. One class of reinforcement learning algorithms alternates between modifying a probability distribution $p(\boldsymbol{\theta})$ and observing values of f for samples from $p(\boldsymbol{\theta})$ to finally assign most of the probability mass to $\boldsymbol{\theta}^*$ - the maximizer of f. However, in order to assign more probability mass to $\boldsymbol{\theta}^*$, there first needs to be evidence for its optimality in form of a sample from f. Keeping in mind that $\boldsymbol{\theta} \in \mathbb{R}^{14}$, the occurrence of such a sample for a narrow gate of the described collision-avoidance problem becomes a highly unlikely event when sampling from e.g. a uniform distribution.

At this point, the similarity of $f(\boldsymbol{\theta}, \mathbf{c})$ to the homotopy $H(\mathbf{x}, \lambda)$, with which we motivated continuation methods, already suggests a way to resolve the illustrated issue. Similar to homotopic predictor-corrector methods, one can interpolate between an easy context instantiation $f(\boldsymbol{\theta}, \mathbf{c}_i)$ and the target one $f(\boldsymbol{\theta}, \mathbf{c}_f)$, reusing the knowledge about the optimal $\boldsymbol{\theta}$ of the easier tasks. This idea seems even more natural as the necessary interpolation between the target function that we seek to optimize and the initial surrogate function that is easy to optimize arises naturally as a consequence of the context $\mathbf{c}$ of contextual reinforcement learning problems. In the following chapter, we investigate existing methods from optimization, optimal control and concurrent systems theory that can be of benefit in designing such a predictor-corrector scheme for contextual reinforcement learning problems, however face limitations that prevent their straightforward applicability. Nonetheless, these insights motivate the algorithm that we propose in chapter 4. Chapters 5 and 6 discuss conducted experiments as well as related work and finally conclude the thesis.

3 Predictor-Corrector Schemes in Reinforcement Learning: First Approaches

To implement predictor-corrector schemes in contextual reinforcement learning problems, it is necessary to adjust algorithm 1 at certain points. First of all, it is necessary to solve a constrained optimization- instead of a root finding problem in the corrector step, as reinforcement learning problems are commonly expressed as constrained optimization problems. While this conceptually does not pose a significant problem, adjusting the predictor step accordingly is somewhat more delicate, as the predictor step needs to approximate the solution for a new context using tangent information at the optimal solution for the last context. Hence we need to find possibilities to compute such a tangent vector. Furthermore, it will become necessary to quantify the similarity of two given contexts in order to find contexts that adequately interpolate between an initial- and a target one, where adequately means that the solution to the contextual problems changes somewhat gradually.
In this chapter, we focus on different concepts from optimization and concurrent systems theory that, although at a first glance seem well suited for computing tangent vectors of solutions and measuring closeness between contexts in the predictor step, by closer investigation suffer from both conceptual and practical limitations which have to our knowledge not been addressed at the moment of writing. While we will not attempt to resolve those shortcomings, they motivate the perspective that we take in chapter 4 to define an algorithm that still shares the basic scheme of predictor-corrector methods while avoiding the aforementioned limitations.

3.1 Sensitivity Analysis

Sensitivity Analysis [9] is a core concept of multi-parametric programming [10], a discipline that aims at computing optimal solutions for parameteric optimization problems

$$\begin{aligned} \boldsymbol{\theta}^*(\mathbf{c}) = \arg\min_{\boldsymbol{\theta}} \; & f(\boldsymbol{\theta}, \mathbf{c}) \\ \text{s.t. } & g(\boldsymbol{\theta}, \mathbf{c}) \leq \mathbf{0} \\ & h(\boldsymbol{\theta}, \mathbf{c}) = \mathbf{0} \end{aligned} \tag{3.1}$$

with optimization variables $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$, parameters $\mathbf{c} \in \mathbb{R}^{n_c}$, inequality constraints $g : \mathbb{R}^{n_\theta} \mapsto \mathbb{R}^{n_g}$ and equality constraints $h : \mathbb{R}^{n_\theta} \mapsto \mathbb{R}^{n_h}$. We will shortly see, why such a constrained optimization problem rather than an unconstrained one is necessary to express reinforcement learning problems. We will now deviate from the exact predictor-corrector scheme by assuming that we already know the change $\mathrm{d}\mathbf{c}$ that will be applied to the current parameters $\mathbf{c}$, as opposed to computing it alongside the change in the optimization variables $\boldsymbol{\theta}^*$. We will look at ways to compute "good" changes $\mathrm{d}\mathbf{c}$ in later sections of this chapter. The problem that now remains is to approximate the solution to problem 3.1 with parameter $\mathbf{c} + \mathrm{d}\mathbf{c}$. The expression $\boldsymbol{\theta}^*(\mathbf{c})$ already indicates that the solutions to the parameterized optimization problem 3.1 can be interpreted as a function of the parameters $\mathbf{c}$. Hence, if we are able to compute the gradient $\nabla_{\mathbf{c}}\boldsymbol{\theta}^*(\mathbf{c}) \in \mathbb{R}^{n_\theta \times n_c}$, we could easily approximate the new solution using a taylor approximation

$$\boldsymbol{\theta}^*(\mathbf{c} + \mathrm{d}\mathbf{c}) \approx \boldsymbol{\theta}^*(\mathbf{c}) + \nabla_{\mathbf{c}}\boldsymbol{\theta}^*(\mathbf{c})\mathrm{d}\mathbf{c}.$$

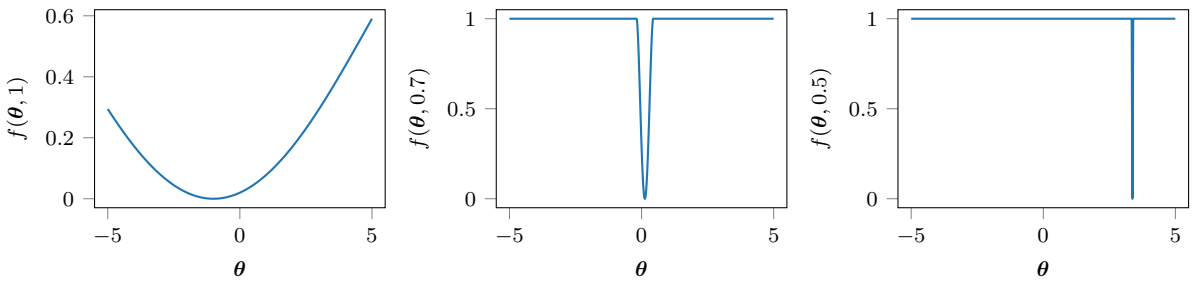

Figure 3.1.: Plots of the function $f(\boldsymbol{\theta}, \mathbf{c})$ over values of $\boldsymbol{\theta} \in [-5, 5]$ for $\mathbf{c} = 1$, $\mathbf{c} = 0.7$ and $\mathbf{c} = 0.5$.

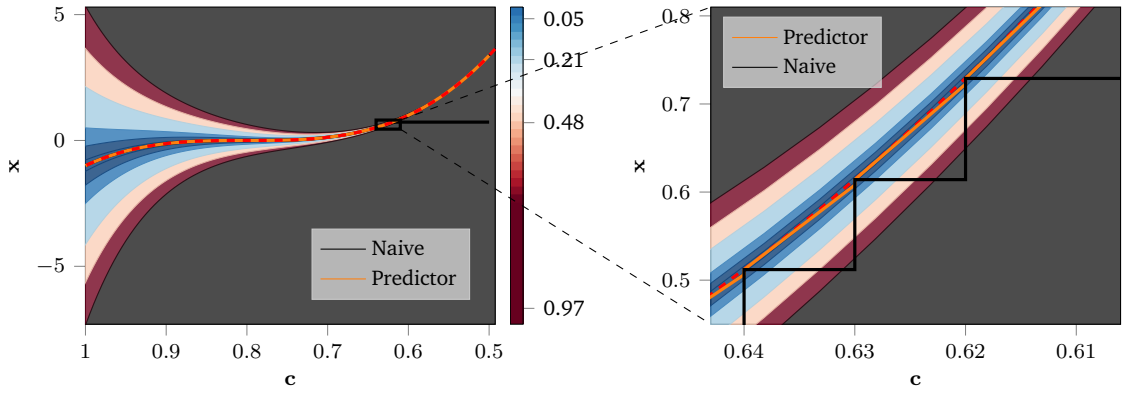

Figure 3.2.: Visualization of the function $f(\boldsymbol{\theta}, \mathbf{c})$ whose minimum should be found for $\mathbf{c} = 0.5$. The red line highlights the curve of points at which the function takes its minimum. The orange and black lines illustrate the way in which the naive approach not taking into account the gradient $\nabla_{\mathbf{c}}\boldsymbol{\theta}^*(\mathbf{c})$ and the approach that uses this information trace the curve.

Sensitivity analysis allows to compute the gradient $\nabla_{\mathbf{c}}\boldsymbol{\theta}^*(\mathbf{c})$ at a solution $\boldsymbol{\theta}^*(\mathbf{c})$. The definition of this gradient uses the Lagrangian function

$$L(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{c}) = f(\boldsymbol{\theta}, \mathbf{c}) + \boldsymbol{\lambda}^T g(\boldsymbol{\theta}, \mathbf{c}) + \boldsymbol{\mu}^T h(\boldsymbol{\theta}, \mathbf{c}), \quad \boldsymbol{\lambda} \in \mathbb{R}^{n_g}, \boldsymbol{\mu} \in \mathbb{R}^{n_h},$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are the Lagrangian multipliers that incorporate the constraints g and h. Having the notion of L, it is possible to more precisely define a solution to the optimization problem 3.1 as the tuple $(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ that fulfills both the so called Linear Independence Constraint Qualification (LICQ) and strict second-order optimality conditions. Both constraints can be reviewed in appendix A.1 and are left out at this point for brevity. If we hence have $\boldsymbol{\theta}_0$, $\boldsymbol{\lambda}_0$ and $\boldsymbol{\mu}_0$ that fulfill the LICQ and the second order optimality conditions for $\mathbf{c}_0$, the following holds

$$\begin{pmatrix} \nabla_{\mathbf{c}}\boldsymbol{\theta}^*(\mathbf{c}_0) \\ \nabla_{\mathbf{c}}\boldsymbol{\lambda}^*(\mathbf{c}_0) \\ \nabla_{\mathbf{c}}\boldsymbol{\mu}^*(\mathbf{c}_0) \end{pmatrix} = \begin{pmatrix} \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} L(\boldsymbol{\theta}_0, \boldsymbol{\lambda}_0, \boldsymbol{\mu}_0, \mathbf{c}_0) & \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}_0, \mathbf{c}_0) & \nabla_{\boldsymbol{\theta}} h(\boldsymbol{\theta}_0, \mathbf{c}_0) \\ \Delta \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}_0, \mathbf{c}_0)^T & \boldsymbol{\Gamma} & \mathbf{0} \\ \nabla_{\boldsymbol{\theta}} h(\boldsymbol{\theta}_0, \mathbf{c}_0)^T & \mathbf{0} & \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \nabla^2_{\boldsymbol{\theta}\mathbf{c}} L(\boldsymbol{\theta}_0, \boldsymbol{\lambda}_0, \boldsymbol{\mu}_0, \mathbf{c}_0) \\ \Delta \nabla_{\mathbf{c}} g(\boldsymbol{\theta}_0, \mathbf{c}_0) \\ \nabla_{\mathbf{c}} h(\boldsymbol{\theta}_0, \mathbf{c}_0) \end{pmatrix}, \tag{3.2}$$

with $\boldsymbol{\Gamma} = \operatorname{diag}(g_1(\boldsymbol{\theta}_0, \mathbf{c}_0), \ldots, g_{n_g}(\boldsymbol{\theta}_0, \mathbf{c}_0))$ and $\Delta = \operatorname{diag}(\lambda_{0_1}, \ldots, \lambda_{0_{n_g}})$. Note that we assume that all the necessary gradients of f, g and h exist and are continuous. For the proof of above result, we refer to [9]. Given a new parameter $\mathbf{c}_1 = \mathbf{c}_0 + \mathrm{d}\mathbf{c}$, we can approximate the solution tuple using

$$\begin{aligned} \boldsymbol{\theta}^*(\mathbf{c}_1) &\approx \boldsymbol{\theta}^*(\mathbf{c}_0) + \nabla_{\mathbf{c}}\boldsymbol{\theta}^*(\mathbf{c}_0)\mathrm{d}\mathbf{c}, \\ \boldsymbol{\lambda}^*(\mathbf{c}_1) &\approx \boldsymbol{\lambda}^*(\mathbf{c}_0) + \nabla_{\mathbf{c}}\boldsymbol{\lambda}^*(\mathbf{c}_0)\mathrm{d}\mathbf{c}, \\ \boldsymbol{\mu}^*(\mathbf{c}_1) &\approx \boldsymbol{\mu}^*(\mathbf{c}_0) + \nabla_{\mathbf{c}}\boldsymbol{\mu}^*(\mathbf{c}_0)\mathrm{d}\mathbf{c}. \end{aligned}$$

Just as in the previous chapter, this is a linear approximation of the solution which is only valid in an arbitrarily small region around $\mathbf{c}_0$. The error in this predicted solution is again compensated by the corrector step, where the optimization problem with parameter $\mathbf{c}_1$ is solved using the prediction of $\boldsymbol{\theta}^*(\mathbf{c}_1)$, $\boldsymbol{\lambda}^*(\mathbf{c}_1)$, $\boldsymbol{\mu}^*(\mathbf{c}_1)$ as a starting point. To illustrate the resulting algorithm, we will take a look at a function that is very similar to the one that motivated predictor corrector methods in the previous section

$$f(\boldsymbol{\theta}, \mathbf{c}) = \begin{cases} -\frac{1}{10^{-4w}} \left(\boldsymbol{\theta} - w^3\right)^4 + \frac{2}{10^{-2w}} \left(\boldsymbol{\theta} - w^3\right)^2, \text{ if } |\boldsymbol{\theta} - w^3| \leq 10^{-w} \\ 1, \text{ else} \end{cases}, \quad w = 4 - 5\mathbf{c}, \ \mathbf{c} \in [0.5, 1].$$

Figure 3.1 shows the function for various values of $\mathbf{c}$. Loosely speaking, f is a negated version of the homotopy H, whose minimum moves according to a polynomial of third order instead of along a line. However, they both share the property that for increasingly small values of $\mathbf{c}$, the gradients of f or H with respect to $\boldsymbol{\theta}$ become zero at most points. This prevents standard optimization procedures to be successfully applied, if the initial guess is not already close to the root or optimum. Figure 3.2 demonstrates that the predictor-corrector approach based on sensitivity analysis is nonetheless able to find solutions for small values of $\mathbf{c}$ by tracing solutions to f starting from $\mathbf{c} = 1$ until reaching $\mathbf{c} = 0.5$. The plots were generated by decreasing the values of $\mathbf{c}$ by 0.01 in every iteration. Just as in the previous chapter, it improves upon a naive implementation that only uses the solution to the previous problem as a starting point for the optimization of the next. This naive implementation was not able to find the optimum for $\mathbf{c} = 0.5$, as the gradient $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^*(0.62), 0.61)$ became zero and hence the optimization of $f(\boldsymbol{\theta}, 0.61)$ failed.

3.1.1 Markov Decision Processes as Constrained Optimization Problems

At this point, we have found a way to implement a simple predictor-corrector scheme for constrained optimization problems. We will try to address the lack of a good way to choose the step $\mathrm{d}\mathbf{c}$ in the next sextion and will now focus on how to use the aforementioned algorithm on reinforcement learning problems. To accomplish this, we need to express them as constrained optimization problems, which is the topic of this section. Typically, Reinforcement learning problems [21] are defined as optimization problems in Markov Decision Processes (MDPs) [22]. MDPs are 4-tuples

$$(S, A, p, r)$$

with states $\mathbf{s} \in S$, actions $\mathbf{a} \in A$, transition probabilities $p : S \times A \times S \mapsto [0, \infty)$ and reward function $r : S \times A \mapsto \mathbb{R}$. The transition probabilities $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ describe how likely it is to reach state $\mathbf{s}'$ from $\mathbf{s}$ by applying action $\mathbf{a}$. As we see, the next state $\mathbf{s}'$ only depends on $\mathbf{s}$ and $\mathbf{a}$. This so called Markov property of stochastic processes gives MDPs their name. The state-space S, action-space A and transition probabilities p together define the environment, in which an agent represented by a policy $\pi : S \mapsto A$ needs to optimize its expected reward

$$J^N(\pi) = E_{p,\mu_0,\pi}\left[\sum_{i=0}^{N} \gamma^i r(s_i, a_i) | a_i \sim \pi(a|s_i), s_{i+1} \sim p(s'|s_i, a_i), s_0 \sim \mu_0(s)\right] \tag{3.3}$$

with initial state distribution $\mu_0 : S \mapsto [0, \infty)$ and discount factor $\gamma \in [0, 1]$. As we see, the initial state distribution $\mu_0(\mathbf{s})$ describes the probability of the initial state $\mathbf{s}_0$ at the beginning of the decision process. The discount factor γ encodes the trade-off between immediate- and long-term reward. Note that it is both common to think about problems with a finite horizon N and with an infinite horizon $N = \infty$. For infinite time horizons, it is required that $\gamma < 1$ as otherwise above sum can grow without bounds.

Up until now, the introduced MDP formulation encoded one task that needed to be solved by an agent. The introduction of a context $\mathbf{c} \in C$ allows to encode multiple related tasks. A common assumption that we will adapt in this thesis is to restrict the influence of the context to the reward function and the transition probabilities, assuming that all MDPs share the same state- and action-space [23, 24]. With these restrictions, we obtain the contextual reward function $r : S \times A \times C \mapsto \mathbb{R}$ and contextual transition probabilities $p : S \times A \times S \times C \mapsto [0, \infty)$, which we will abbreviate with $r_{\mathbf{c}}(s, a)$ and $p_{\mathbf{c}}(s'|a, s)$. With these definitions, we can now define a contextual MDP as the 4-tuple

$$\mathrm{MDP}_{\mathbf{c}}(S, A, p_{\mathbf{c}}, r_{\mathbf{c}}).$$

The policy now also extends to $\pi : S \times A \times C \mapsto \mathbb{R}$ and will also be abbreviated by $\pi_{\mathbf{c}}$. The optimization objective changes to

$$J^N(\pi_{\mathbf{c}}) = E_{p_{\mathbf{c}},\mu_0^{\mathbf{c}},\pi_{\mathbf{c}}}\left[\sum_{i=0}^{N} \gamma^i r_{\mathbf{c}}(s_i, a_i) | a_i \sim \pi_{\mathbf{c}}(a|s_i), s_{i+1} \sim p_{\mathbf{c}}(s'|s_i, a_i), s_0 \sim \mu_0^{\mathbf{c}}(s)\right]. \tag{3.4}$$

Note that we also allowed the initial state distribution $\mu_0 : S \times C \mapsto [0, \infty)$ to be changed by the context $\mathbf{c}$, while keeping the discount factor γ fixed. With some adjustments, it is possible to express the optimization problem 3.4 as a parametric optimization problem 3.1 to which sensitivity analysis can be applied. For that we first of all assume that we face a continuous state-, action- and context space, i.e. $S \subseteq \mathbb{R}^{n_s}$, $A \subseteq \mathbb{R}^{n_a}$ and $C \subseteq \mathbb{R}^{n_c}$. Next, we require that we do not face probabilistic transition dynamics $p_{\mathbf{c}}$, but require them to be deterministic, i.e. have $d : S \times A \times C \mapsto S$ (abbreviated as $d_{\mathbf{c}}(s, a)$). We will discuss the implications and especially problems that arise from this assumption in the conclusion on sensitivity analysis and will just accept this restriction for now. As a consequence of assuming deterministic system dynamics, we also require a deterministic initial state s_0 instead of a probability distribution over it. Furthermore, we will not optimize the policy $\pi_{\mathbf{c}}$, but choose to directly optimize the actions

$$\mathbf{a}_{0:N} = \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \dots & \mathbf{a}_N \end{bmatrix} \in \mathbb{R}^{N n_a}.$$

This does not constitute a severe limitation, as it is well known that if a probabilistic policy maximizing objective 3.4 exists, there also exists a deterministic policy with the same level of performance [25]. Instead of optimizing this deterministic policy (or typically some parameters of it), we choose to directly optimize the actions that would be generated by it. And if we assume that we are working with a perfect model of $d_{\mathbf{c}}$, there is no need to adjust the computed

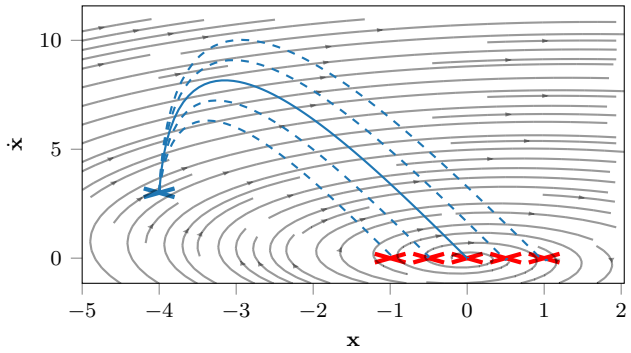

Figure 3.3.: The plot shows an LQR System with the state $\mathbf{s} = [\mathbf{x}\ \dot{\mathbf{x}}] \in \mathbb{R}^2$ being the position and velocity of a point mass to which forces $\mathbf{a} \in \mathbb{R}$ can be applied. The uncontrolled system is unstable, i.e. increasingly diverges from the origin if not stabilized by the applied forces as indicated by the black lines that visualize the uncontrolled dynamics. The solid blue line shows the trajectory generated from the optimal actions to optimization problem 3.5 with $r_{\mathbf{c}}$ as defined in 3.6 and goal state $\mathbf{c}_0 = [0\ 0]$. The dashed blue lines show the trajectories that were generated with actions computed with a linear interpolation using the gradient information $\nabla_{\mathbf{c}}\mathbf{a}^*_{0:N}(\mathbf{c}_0)$ for different goal positions illustrated by red crosses.

"feed-forward" actions during execution. The previous assumptions now allow to express objective 3.4 as the following optimization problem

$$\mathbf{a}^*_{0:N}(\mathbf{c}) = \underset{\mathbf{a}_{0:N}, \mathbf{s}_{1:N}}{\arg\min} \; -\sum_{i=0}^{N} \gamma^i r_{\mathbf{c}}(\mathbf{s}_i, \mathbf{a}_i) \tag{3.5}$$

$$\text{s.t.} \begin{pmatrix} \mathbf{s}_1 - d_{\mathbf{c}}(\mathbf{s}_0, \mathbf{a}_0) \\ \mathbf{s}_2 - d_{\mathbf{c}}(\mathbf{s}_1, \mathbf{a}_1) \\ \vdots \\ \mathbf{s}_N - d_{\mathbf{c}}(\mathbf{s}_{N-1}, \mathbf{a}_{N-1}) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where the dynamics of the MDP are expressed using equality constraints. However, we see that with above formulation, we can only solve finite-time horizon problems, i.e. with $N \in \mathbb{N}$. Because of this, the following section will introduce ways to resolve this conceptual problem and turn the "feed-forward" actions into a "closed-loop" policy.

3.1.2 Sensitivity Analysis for LQR Problems

Linear Quadratic Regulator (LQR) problems are well known in the domain of optimal control. Indeed optimal control and reinforcement learning are inherently connected, as described in [26], making LQR problems an interesting first benchmark for the "performance" of sensitivity analysis. For that, we will define the transition dynamics $d_{\mathbf{c}}$ to be linear with

$$\mathbf{s}_{i+1} = d_{\mathbf{c}}(\mathbf{s}_i, \mathbf{a}_i) = (\mathbf{I} + \mathbf{A})\,\mathbf{s}_i + \mathbf{B}\mathbf{a}_i,$$

where $\mathbf{s}_i, \mathbf{s}_{i+1}, \in \mathbb{R}^{n_s}$, $\mathbf{a}_i \in \mathbb{R}^{n_a}$, $\mathbf{A} \in \mathbb{R}^{n_s \times n_s}$ and $\mathbf{B} \in \mathbb{R}^{n_s \times n_a}$. The context $\mathbf{c} = \mathbf{s}_g$ will represent the goal state $\mathbf{s}_g$, that the system should be driven to. This desired behavior is encoded by the reward function

$$r_{\mathbf{c}}(\mathbf{s_i}, \mathbf{a_i}) = -(\mathbf{s}_i - \mathbf{s}_g)^T \mathbf{Q}(\mathbf{s}_i - \mathbf{s}_g) - \mathbf{a}_i^T \mathbf{R}\mathbf{a}_i. \tag{3.6}$$

We see that the reward function further penalizes large actions $\mathbf{a}_i$, typically leading to a smoother trajectory of the system state $\mathbf{s}$. Having a definition of $r_{\mathbf{c}}$ and $d_{\mathbf{c}}$, it is now possible to compute the optimal actions $\mathbf{a}^*_{0:N}(\mathbf{c})$ by solving optimization problem 3.5. Being able to compute solutions for individual contexts $\mathbf{c}$, the question arises how much we can gain from the gradient $\nabla_{\mathbf{c}}\mathbf{a}^*_{0:N}(\mathbf{c})$. As a closer look at the definition 3.2 reveals that those gradients are very powerful for LQR problems. Indeed, the approximation

$$\mathbf{a}^*_{0:N}(\mathbf{c}_1) \approx \mathbf{a}^*_{0:N}(\mathbf{c}_0) + \nabla_{\mathbf{c}}\mathbf{a}^*_{0:N}(\mathbf{c}_0)\mathrm{d}\mathbf{c}$$

is actually exact for any $\mathbf{c}_0$ and $\mathbf{c}_1$, as the gradients of f and h that appear in equation 3.2 are constant, making obtained gradient $\nabla_{\mathbf{c}}\mathbf{a}^*_{0:N}(\mathbf{c})$ constant across all contexts $\mathbf{c}$. This beneficial property is exploited in figure 3.3, where the solution obtained for context $\mathbf{c} = [0\ 0]$ is generalized to other contexts without needing to solve another optimization problem. Indeed, this property also allows for a straight-forward implementation of a Model Predictive Control (MPC) scheme [27] for LQR problems, where a solution to the optimization problem 3.5 is constantly refined to compensate for errors in the dynamics model $d_{\mathbf{c}}$. To pose an example, imagine that the dynamics were actually perturbed by some noise

$$\mathbf{s}_{i+1} = d_{\mathbf{c}}(\mathbf{s}_i, \mathbf{a}_i) = (\mathbf{I} + \mathbf{A})\,\mathbf{s}_i + \mathbf{B}\mathbf{a}_i + \boldsymbol{\delta}, \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}).$$

Obviously, $d_{\mathbf{c}}$ is now a probability distribution over the next system state. This creates problems as we require deterministic dynamics functions. If we simply ignore the perturbation during planning and only assume the initial state $\mathbf{s}_0$ to be exactly given, all actions $\mathbf{a}^*_{1:N}$ become non-optimal, since the actual state $\mathbf{s}_1$ that resulted from applying action $\mathbf{a}^*_0$ in state $\mathbf{s}_0$ may differ from the one the actions $\mathbf{a}^*_{1:N}$ were planned with. If we however extend the context vector to not only contain the goal state $\mathbf{s}_g$ but also the initial state $\mathbf{s}_0$, i.e. $\mathbf{c} = [\mathbf{s}_0\ \mathbf{s}_g]$, we obtain a straight-forward way of correcting the actions $\mathbf{a}^*_{1:N}$ after executing action $\mathbf{a}^*_0$ and observing the true state $\mathbf{s}_1$ by computing

$$\mathbf{a}^*_{0:N}([\mathbf{s}_1\ \mathbf{s}_g]) = \mathbf{a}^*_{0:N}([\mathbf{s}_0\ \mathbf{s}_g]) + \nabla_{\mathbf{c}}\mathbf{a}^*_{0:N}([\mathbf{s}_0\ \mathbf{s}_g])[(\mathbf{s}_1 - \mathbf{s}_0)\ \mathbf{0}].$$

The outlined scheme turns the feed-forward actions into a closed-loop control, as the actions are corrected after every step. This also allows to approximately solve infinite time-horizon problems through a repeated solution of finite horizon problems. Figure 3.4 shows the result of such an MPC scheme for a perturbed linear system. We can see that the random perturbations that occur during the system transitions can be compensated through the repeated correction of the planned actions. Without such corrections, the trajectory increasingly deviates from the desired position. Investigating the change in the optimal solution with regards to a change in the initial state has an interesting interpretation, which we will investigate in the following section.

3.1.3 Sensitivity Analysis and Value Functions

The scheme of encoding the initial state $\mathbf{s}_0$ of optimization problem 3.5 as a part of the context $\mathbf{c}$ and adjusting the optimal actions $a^*_{0:N}(\mathbf{c})$ according to a change in this initial state creates a noteworthy connection to classical concepts known in MDP theory. To highlight this connection, it is necessary to look back at the non-contextual reinforcement learning objective 3.3, for which we assume a finite horizon N for now. We have already mentioned that if there is a probabilistic policy maximizing this objective, there also exists a deterministic one with the same performance. This deterministic optimal policy π^*_N can be defined by

$$\pi^*_N(\mathbf{s}) = \arg\max_{\mathbf{a}} \left(r(\mathbf{s}, \mathbf{a}) + \gamma E_p\left[V^*_{N-1}(\tilde{\mathbf{s}}) | \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a})\right]\right).$$

While the notation π^*_N highlights that for every timestep i, there is a distinct policy π^*_i, we will still only talk about *the* optimal policy π^*_i, since we could e.g. extend the policy to take the current step i as an argument. The function $V^*_N(\mathbf{s})$ is typically referred to as the optimal value function and defined as

$$V^*_N(\mathbf{s}) = \max_{\mathbf{a}} \left(r(\mathbf{s}, \mathbf{a}) + \gamma E_p\left[V^*_{N-1}(\tilde{\mathbf{s}}) | \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a})\right]\right).$$

with

$$V^*_0 = \max_{\mathbf{a}} r(\mathbf{s}, \mathbf{a}).$$

Looking at above definitions, we can identify two important properties of V^*_N. First of all, the value function is defined in a recursive fashion. Furthermore, the definitions of π^*_N and V^*_N only differ in employing the $\arg\max$ instead of the $\max$ operator. Hence, the value function $V^*_N(\mathbf{s})$ encodes the expected accumulated reward (i.e. the value) that is obtained when acting according to the optimal policy π^*_i starting from state $\mathbf{s}$ for N steps. Now assuming deterministic system dynamics d, the definition of the value function reduces to

$$\begin{aligned} V^*_N(\mathbf{s}) &= \max_{\mathbf{a}} \left(r(\mathbf{s}, \mathbf{a}) + \gamma V^*_{N-1}\left(d(\mathbf{s}, \mathbf{a})\right)\right) \\ &= \max_{\mathbf{a}_{0:N}} \sum_{i=0}^{N} \gamma^i r(s_i, a_i) \\ &= \min_{\mathbf{a}_{0:N}} -\sum_{i=0}^{N} \gamma^i r(s_i, a_i), \quad s_0 = s,\ s_{i+1} = d(s_i, a_i). \end{aligned}$$

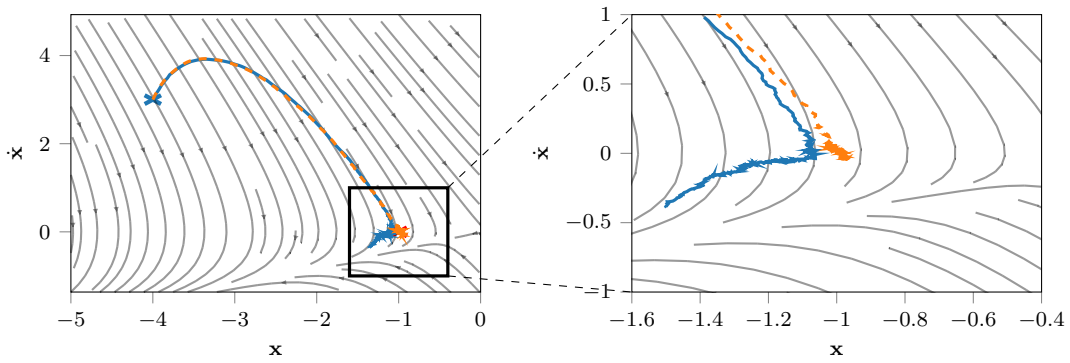

Figure 3.4.: Visualization of a perturbed LQR System with forces $\mathbf{a} \in \mathbb{R}$ that can be applied to a points mass represented by $\mathbf{s} = [\mathbf{x}\ \dot{\mathbf{x}}] \in \mathbb{R}^2$. The solid blue line shows the trajectory generated by applying the optimal actions $\mathbf{a}^*_{0:N}(\mathbf{c}_0)$ according to optimization problem 3.5 with initial state $\mathbf{s}_0 = [-4\ 3]$, goal state $\mathbf{s}_g = [-1\ 0]$ and reward $r_\mathbf{c}$ according to equation 3.6 - without taking the random perturbations of the LQR system into account. The dashed orange line shows the trajectory resulting from correcting the actions according to the current system state using the gradient information $\nabla_\mathbf{c}\mathbf{a}^*_{0:N}(\mathbf{c})$.

This definition resembles objective 3.5 in which the $\arg\min$ operator is replaced by the $\min$ operator, showing that objective 3.5 indeed computes the actions that would have been generated by the optimal policy π^*_i at time-steps $i = 0, \dots, N$. And more importantly, we now know that we indirectly compute the optimal value function $V^*_N(\mathbf{s})$ when computing the actions $\mathbf{a}^*_{0:N}$. Hence, sensitivity analysis also allows us to approximate value functions $V^*_N(\tilde{\mathbf{s}})$ for nearby states $\tilde{s}$ from a state $\mathbf{s}$ in which we know $V^*_N(\mathbf{s})$.
While this connection to value functions and the promising performance on LQR systems may lead to the assumption that sensitivity analysis is indeed a well-suited tool for the implementation of predictor-corrector schemes in reinforcement learning domains, the following section will show why this first impression does not last.

3.1.4 Limitations of Sensitivity Analysis

We first concentrate on a rather technical problem arising from the conditions that need to be fulfilled in order to compute the gradient $\nabla_\mathbf{c}\boldsymbol{\theta}^*(\mathbf{c})$. More specifically, it is the LICQ that can become violated at a solution $\boldsymbol{\theta}^*(\mathbf{c})$, if there exist additional constraints to the optimization problem 3.5. Such constraints may indeed be necessary to e.g. bound the actions that are applied to a system or to prevent a system from reaching catastrophic states using inequality constraints. However, a large number of additional active constraints at a solution $\boldsymbol{\theta}^*$ may lead to a violation of the LICQ. More precisely, this happens if the number of active constraints becomes larger than the number of optimization variables of 3.5. This problem becomes even more pronounced, if we do not optimize the individual actions $\mathbf{a}_{0:N}$ but instead optimize the parameters $\boldsymbol{\theta} \in \mathbb{R}^{n_{\boldsymbol{\theta}}}$ of a deterministic policy $\pi_\mathbf{c}$. If the parameterization is somewhat low-dimensional, the LICQ may be violated even for a small number of additional active constraints. However, this problem may already be alleviated by employing a different constraint qualification than the LICQ. Besides this problem, so called active set changes, i.e. a change in the set of inequality constraints that are active at two solution points, require additional treatment as detailed in [28, 29, 30]. Another technical problem becomes obvious when looking back at the different reward functions of the gate environment in figure 2.5. While the functions share the same characteristics as the one in figure 3.1, they are only piece-wise continuous, due to the possibility of hitting the wall during the experiment. Such discontinuities can prohibit the computation of the gradient, as we required all derivates of the objective and the constraints to be continuously differentiable. So to be applicable to a wide range of reward functions and transition dynamics, it would be necessary to deal with such discontinuities.

Furthermore, there is another, more implicit continuity assumption that we want to mention at this point. The predictor-corrector method introduced in chapter 2 was defined using homotopies between two functions. These homotopies are, per definition, continuous., i.e. a small change in the parameter λ can only slightly change the value of the homotopy at a given point $\mathbf{x}$. With that, also the roots of the homotopy can only gradually change with regards to λ. However, for general MDPs we do not have any guarantee that a slight change in context $\mathbf{c}$ will lead to a small change in the optimal policy. To pose an example, we can think of a simple pendulum to which a certain maximum amount of force can be applied. Hanging downwards, the pendulum should be brought to an upright position. If the context $\mathbf{c}$ encodes the gravitational force that the pendulum is exposed to, it becomes clear that for a certain amount of gravitational force,

the maximum amount of force that can be applied to the pendulum does not suffice to directly swing up the pendulum. Hence, the optimal policy abruptly changes from a direct swing up to a policy that needs to first gather kinetic energy by swinging the pendulum back and forth before reaching the upright position. This kind of discontinuity does not arise from any discontinuities in the reward function or the dynamics of the MDP, but arises from the bounded action-space that is available to the agent.

We want to conclude with another important problem. The reinforcement learning optimization problem 3.3 is defined as an expected reward over the policy execution for N steps. The need for computing such an expected reward arises because of the non-deterministic transition probabilities p, as we already know that an optimal policy does not need to be probabilistic. However, even if the transitions of a system are completely deterministic, chapter 2 already outlined that reinforcement learning algorithms do not only need to compute an optimal policy but also need to learn about the behavior of the system for which the optimal policy should be found. Hence, even for a deterministic system, the initial uncertainty about its dynamics would again require to model the dynamics as probability distributions. Consequently, dealing with probabilistic transition dynamics is a core requirement for a successful reinforcement learning algorithm, regardless of the true dynamics. Unfortunately, sensitivity analysis does not offer any means to deal with such uncertainty as the optimization problem 3.1 requires the objective and the constraints to be deterministic functions. We can however alleviate this problem, if we define an alternative state space S_{prob} of probability distributions over the actual system state

$$(\tilde{p}: \ S \mapsto [0, \infty)) \in S_{\text{prob}},$$

with which we can now express the transitions between two states $\tilde{p}_{i+1}$ and $\tilde{p}_i$ as

$$\tilde{p}_{i+1}(\mathbf{s}') = \int_S p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \tilde{p}_i(\mathbf{s}) d\mathbf{s} = \tilde{f}(\tilde{p}_i, \mathbf{a}),$$

where p are the (learned) transition probabilities of the system. We see that with such a definition, we obtain a deterministic transition function d, that maps from one probability distribution $\tilde{p}_i$ to another. We now need to introduce an alternative reward function $\tilde{r}$, that computes the expected reward

$$\tilde{r}(\tilde{p}, \mathbf{a}) = E_{\tilde{p}} \left[r(\mathbf{s}, \mathbf{a}) | \mathbf{s} \sim \tilde{p}(\mathbf{s}) \right].$$

Having a deterministic reward function and deterministic transition probabilities, it looks like we can now readily define the parametric optimization problem 3.1 with S_{prob}, $\tilde{f}$ and $\tilde{r}$. However, the very definition of the parametric optimization problem requires $S_{\text{prob}} \subseteq \mathbb{R}^{n_s}$, which is obviously violated as S_{prob} is a function space. One way to escape these problems is to restrict oneself to a distribution with real parameters and define S_{prob} to be the space of those real-valued parameters. This was done in [31] using Gaussian distributions. However, as e.g. shown in [32] this treatment of uncertainty comes at very high computational costs that prevents the resulting algorithm to be applied to high-dimensional state-action spaces.

All in all, it seems that although very promising at first glance, a predictor-corrector scheme based on sensitivity analysis poses several technical and conceptual problems that prevent it from being applied to a wide range of reinforcement learning problems in a straightforward way.

3.2 MDP Measures

Sensitivity analysis allowed us to compute how a change in the context $\mathbf{c}$ is reflected by a change in the optimal solution $\boldsymbol{\theta}^*(\mathbf{c})$. However, it does not help to select to which context $\mathbf{c}'$ the current solution should be transferred to. An obvious first idea is to just use a linear interpolation between an initial context $\mathbf{c}_0$ and the final one $\mathbf{c}_f$

$$\mathbf{c}_i = \mathbf{c}_0 + \frac{i}{N}(\mathbf{c}_f - \mathbf{c}_0), \quad i = 0, \ldots, N.$$

However, such an interpolation may not be adequate, since a small change in context $\mathbf{c}$ according to a geometric norm may lead to a severe change in the structure of the MDP or its solution, as we have already outlined in the previous section.
To pose another example of this problem, we look at a 4×4 grid-world with four different height levels. The grid-world has $n_s = 16$ states and $n_a = 5$ actions. Given a state $s \in S = \{1, \ldots, 16\}$, the x- and y-position of the agent in the grid-world can be computed by $p(s) = [x \ y] = [\lfloor s/4 \rfloor \ (s \bmod 4)]$. In each state, the agent can decide whether it wants to move up, left, down or right - or stay at the current position. There is no non-determinism in the state-transitions.

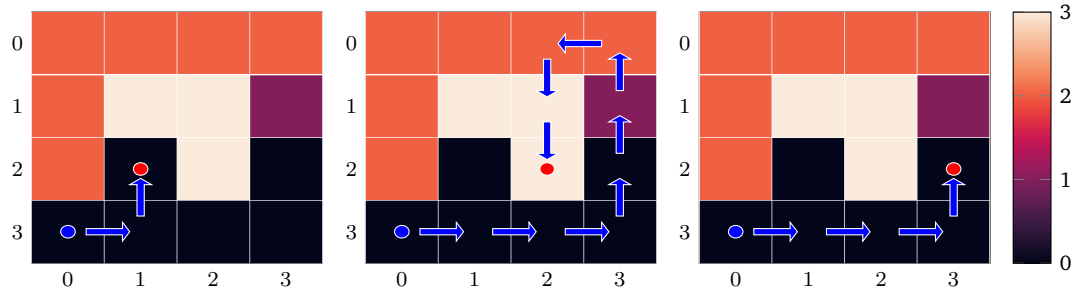

Figure 3.5.: Three 4×4 grid-worlds in which an agent needs to reach different goal positions $[2\ 1]$, $[2\ 2]$ and $[2\ 3]$ (red dots) starting from position $[0\ 3]$ (blue dot). Different height levels are visualized by different colors and the agent can only move to positions that are at most one height level above or below the current one. The shortest path to the three positions is represented by blue arrows.

Hence, if an agent decides to move up, it will end up in the position above the current one in the next step. However, not all actions are valid in a state, since an agent cannot go left if it is at the left border of the grid-world. Furthermore, it cannot move to a position s' whose height $h(s')$ (we will also use the notation $h(p(s'))$) is more than 1 level above or below the current one. The agent starts in the left-bottom corner, i.e. at $p(s_0) = [0\ 3]$, and then needs to reach a goal position that is encoded by the context, i.e. $\mathbf{c} = [g_x\ g_y]$. The reward function for this task is defined in terms of the L1-norm as well as the absolute difference in height levels between current and desired position

$$r(s, \mathbf{c}) = \frac{36 - 10|h(s) - h(\mathbf{c})| - \|p(s) - \mathbf{c}\|_1}{36}.$$

Assuming we would like to gradually transfer a policy from context $\mathbf{c}_0 = [1\ 2]$ to $\mathbf{c}_f = [3\ 2]$, the linear interpolation scheme from above would yield an intermediate step $\mathbf{c}_1 = [2\ 2]$. Figure 3.5 visualizes how problematic this interpolation can be for a specific instance of such a grid-world. As we can see, the policy changes drastically between $\mathbf{c}_0$ and $\mathbf{c}_1$ as well as $\mathbf{c}_1$ and $\mathbf{c}_f$ while the policies of $\mathbf{c}_0$ and $\mathbf{c}_f$ do not change as drastically. Obviously, we need a more elaborate way of interpolating between contexts of MDPs that ensures that the policy only gradually changes with a change in context. More precisely, we need a distance measure $d_{\text{MDP}}(\text{MDP}_{\mathbf{c}_0}, \text{MDP}_{\mathbf{c}_1})$ between MDPs that allows us to quantify how different the two MDPs parameterized by $\mathbf{c}_0$ and $\mathbf{c}_1$ are in terms of the optimal policy. In the following section we will first review concepts from the study of concurrent processes and building upon that investigate ways to express such distance measures.

3.2.1 Bisimulation Metrics

Bisimulation [13] is a concept that describes whether two states of a process or system are equivalent in their behavior and has been extended to finite MDPs in [33]. The term "finite" refers to the finite number of states and actions in the MDP. Hence we can represent the state- and action space by a finite subset of the natural numbers: $S = \{i | i = 1, \ldots, n_s\}$ and $A = \{i | i = 1, \ldots, n_a\}$, as was the case for the previously described grid-world. Bisimulation is defined as an equivalence relation B on the state-space S of two MDPs $\text{MDP}_1 = (S, A, p_1, r_1)$ and $\text{MDP}_2 = (S, A, p_2, r_2)$. Note that we require that the MDPs share the same state-action-space for the following exposition, although the definition in [33] allows the state-spaces to differ. A bisimulation relation fulfills the following property

$$s_1 \sim_{\text{B}} s_2 \Leftrightarrow \forall a \in A : (r_1(s_1, a) = r_2(s_2, a) \wedge \forall C \in S/\text{B} :\ p_1(C|s_1, a) = p_2(C|s_2, a)),$$

where S/B is the partition of the state-space induced by the relation B and $p(C|s, a) = \sum_{\tilde{s} \in C} p(\tilde{s}|s, a)$ is the probability of transitioning from the current state s to a partition C using action a. To put above formula in words, two states are bisimulation equivalent, if their immediate rewards for all actions and the probability of transitioning into the partitions according to the bisimulation relation are the same. Consequently, such a bisimulation relation aggregates the state-space into clusters that behave identical - not only in the immediate but also in a long-term sense. This property makes such relations interesting to reduce the number of sates for planning in an MDP. However, we pursue a quite different goal of quantifying the difference between two MDPs. For this purpose, the binary information whether states in two MDPs are bisimulation equivalent or not is obviously not sufficient. [11] addresses this shortcoming by introducing a metric $d_{\text{B}} : S \times S \mapsto [0, \infty)$ which fulfills

$$d_{\text{B}}(s_1, s_2) = 0 \Leftrightarrow s_1 \sim_{\text{B}} s_2$$

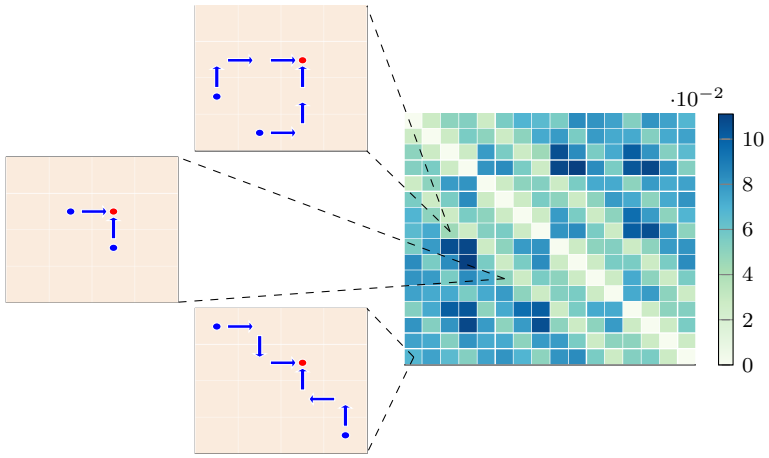

Figure 3.6.: The right plot visualizes the distance between any two states in a 4×4 grid-world according to the bisimulation metric d_{B}. Examples of this grid-world are shown on the left, where two states (blue dots) as well as the optimal policy (blue arrows) starting from these states towards the goal position (red dot) are shown. The grid-world has only one height level, so it is possible to go to any neighboring position without restrictions. The dotted lines show the corresponding distance of the two initial states according to d_{B}.

and hence allows to quantify the similarity of states in MDP_1 and MDP_2, where a value of zero means that they are bisimulation equivalent and larger values encode more distinct long-term behaviour of those states. However, the derivations and proofs in [11] are only given for the case that $\mathrm{MDP}_1 = \mathrm{MDP}_2$. We will nonetheless introduce the general case, since we expect the proofs to straightforwardly generalize. Please note, however, that this is not guaranteed. The metric d_{B} is expressed as the fixed point of the following operator on a metric $d : S \times S \mapsto [0, \infty)$

$$F_{\mathrm{B}}(d)(s_1, s_2) = \max_{a \in A} \left(c_R |r_1(s_1, a) - r_2(s_2, a)| + c_T T_k(d)(p_1(\cdot|s_1, a), p_2(\cdot|s_2, a)) \right), \tag{3.7}$$

with $c_R = \gamma$ and $c_T = 1 - \gamma$. $T_k(d)(\cdot)$ is the kantorovich or wasserstein distance [34] between the probability distributions $p_1(\cdot|s_1, a)$ and $p_2(\cdot|s_2, a)$ and can be defined as the following linear program

$$\begin{aligned} T_k(d)(p_1(\cdot|s_1, a), p_2(\cdot|s_2, a)) = \min_{l: S\times S \mapsto [0,\infty)} & \sum_{s_1, s_2 \in S} l(s_1, s_2) d(s_1, s_2) \\ \text{s.t. } p_1(\tilde{s}|s_1, a) &= \sum_{s_2 \in S} l(\tilde{s}, s_2) \; \forall \tilde{s} \in S \\ p_2(\tilde{s}|s_2, a) &= \sum_{s_1 \in S} l(s_1, \tilde{s}) \; \forall \tilde{s} \in S. \end{aligned} \tag{3.8}$$

Being also referred to as the "Earth Movers Distance", above optimization problem has a very intuitive interpretation. Viewing the two probability distributions $p_1(\cdot|s_1, a)$ and $p_2(\cdot|s_2, a)$ as heaps of dirt, $T_k(d)$ expresses the minimum amount of work that is necessary to transform one heap into another, where the work necessary to move one unit of dirt from $\tilde{s}_1$ to $\tilde{s}_2$ is expressed by d. As already pointed out, we can use the operator 3.7 to, starting from an initial metric d_0, converge towards a bisimulation metric via

$$d_{i+1}(s_1, s_2) = F(d_i)(s_1, s_2).$$

Figure 3.6 visualizes the metric d_{B} between states in a different version of the 4×4 grid-world, where $h(\mathbf{s}) = 0$ for all states $\mathbf{s}$. Looking at the visualizations, we can however already see that the bisimulation metric d_{B} is very strict, as it assigns a non-zero distance to many states, whose optimal policies are actually mirrored versions of each other, such as e.g. $d_{\mathrm{B}}([0\ 2], [1\ 3])$. This results from the requirement that *all* actions "behave" identical in the two compared states in order to label them bisimilar. If we however think in terms of the solutions to MDPs, we could also only require that for every (or even only the optimal action) in one state, there exists at least one action in the other state that behaves similarly.

3.2.2 Lax Bisimulation-Metrics

Building up on the work by [11], [12] introduced a metric that reflects the so called lax bisimulation equivalence relation

$$s_1 \sim_{\mathrm{LB}} s_2 \Leftrightarrow \forall a_1 \in A \; \exists a_2 \in A : \; (r_1(s_1, a_1) = r_2(s_2, a_2) \wedge \forall C \in S/\mathrm{LB} : \; p_1(C|s_1, a_1) = p_2(C|s_2, a_2)) .$$

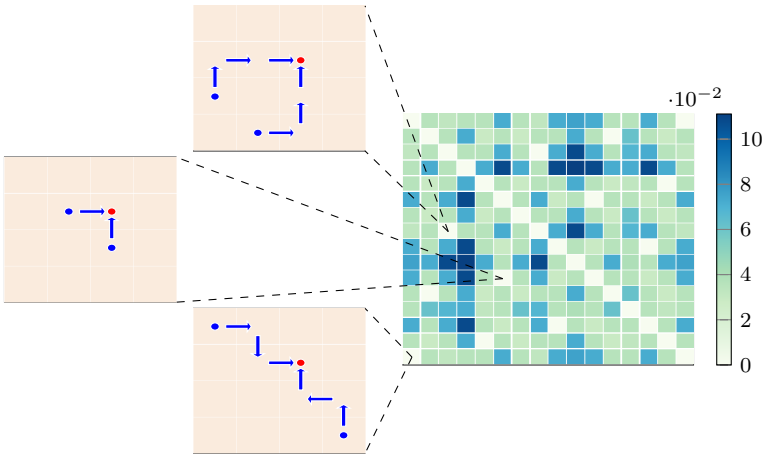

Figure 3.7.: Three 4×4 grid-worlds with different states (blue dots) as well as their optimal policies (blue arrows) towards the goal position (red dot) starting from these states are shown on the left. The right plot visualizes the distance between any two states in the 4×4 grid-world according to the bisimulation metric d_{LB}. The dotted lines mark the distance between the two states in the left plots according to d_{LB}.

This is a less strict version of bisimulation that only requires that for every action a_1 in state s_1 there exists at least one action a_2 in s_2 that has the same immediate reward and the same transition probabilities w.r.t to the state partitioning of LB. This relaxation will later allow the metric to reflect symmetric structures in a MDP. Note that again, the definitions and proofs in [12] were only given for the case $\mathrm{MDP}_1 = \mathrm{MDP}_2$ but we introduce the formulas in the more general context, assuming that the proofs generalize as well. A metric encoding this relaxed definition of bisimulation can, just as in [11], be obtained by computing the fixed point of an operator. For that we will first of all define the following intermediate metric

$$\delta(d)\left([s_1\ a_1],[s_2\ a_2]\right) = c_R|r_1(s_1,a_1) - r_2(s_2,a_2)| + c_T T_k(d)\left(p_1(\cdot|s_1,a_1), p_2(\cdot|s_2,a_2)\right).$$

Note that this metric is defined on the state-action space $S \times A$. The reason for that becomes obvious when looking at the actual operator

$$\begin{aligned} F_{\mathrm{LB}}(d)(s_1,s_2) &= H(\delta(d))(X_{s_1}, X_{s_2}) \\ &= \max\left(\sup_{x_1 \in X_{s_1}} \inf_{x_2 \in X_{s_2}} \delta(d)(x_1,x_2), \sup_{x_2 \in X_{s_2}} \inf_{x_1 \in X_{s_1}} \delta(d)(x_1,x_2)\right), \end{aligned} \tag{3.9}$$

where $X_s = \{s\} \times A$ and $H(\delta(d))$ is the so called Hausdorff metric between non-empty compact subsets of $S \times A$. The application of the Hausdorff metric to X_{s_1} and X_{s_2} reflects the property that given a state s_1 and action a_1, there only needs to be one action a_2 in state s_2 with equal reward and transition probabilities. Just as in [11], the update according to 3.9 can be iterated until convergence to obtain a metric d_{LB} that fulfills

$$d_{\mathrm{LB}}(s,s') = 0 \Leftrightarrow s \sim_{\mathrm{LB}} s'$$

starting from any metric d_0. Figure 3.7 illustrates the metric for the grid-world examined in figure 3.6. We can see that it correctly "detects" the symmetric relation between the states $[0\ 2]$ and $[1\ 3]$ and hence yields $d\left([0\ 2],[1\ 3]\right) = 0$. In general, the lax bisimulation metric is able to detect the symmetries of all states with regards to the main diagonal of the grid-world in figure 3.7.

3.2.3 Relation to Value Functions

A very revealing property of (lax) bisimulation metrics is that they bound the change in the optimal value function. Until now, however, we have only seen a definition of the optimal value function $V_N^*(\mathbf{s})$ for a finite-time horizon N. Since the aforementioned bound on the value function is defined for $N \to \infty$, we need to introduce the infinite-time horizon value function

$$V^*(\mathbf{s}) = \max_{\mathbf{a}} \left(r(\mathbf{s},\mathbf{a}) + \gamma E_p\left[V^*(\tilde{\mathbf{s}})|\tilde{\mathbf{s}} \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})\right]\right).$$

As we can see, the difference to V_N^* is that the infinite-time horizon value function is not dependent on the current horizon i. And indeed, we can compute V^* by computing V_N^* for larger and larger time horizons, i.e. iterate

$$\begin{aligned} V_{i+1}(\mathbf{s}) &= \max_{\mathbf{a}} \left(r(\mathbf{s}, \mathbf{a}) + \gamma E_p \left[V_i(\tilde{\mathbf{s}}) | \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \right] \right) \\ &= F_V(V_i)(\mathbf{s}) \end{aligned}$$

until $V_{i+1} = V_i$. To put it differently, the optimal value function V^* is a fixed point of the operator F_V. Just as before, the optimal policy for the infinite-time horizon objective is defined analogously as V^*, only replacing the $\max$ with the $\arg\max$ operator

$$\pi^*(\mathbf{s}) = \arg\max_{\mathbf{a}} \left(r(\mathbf{s}, \mathbf{a}) + \gamma E_p \left[V^*(\tilde{\mathbf{s}}) | \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \right] \right).$$

Looking at the definitions of the operators for the (lax) bisimulation metric given in 3.7 and 3.9, it is actually not that surprising anymore that the following results could be proved in [11] and [12]

$$c_R |V^*(\mathbf{s}_1) - V^*(\mathbf{s}_2)| \leq d_{\mathrm{LB}}(\mathbf{s}_1, \mathbf{s}_2) \leq d_{\mathrm{B}}(\mathbf{s}_1, \mathbf{s}_2). \tag{3.10}$$

Hence both metrics actually give an upper bound on the change in value between two states, where the bound by the lax bisimulation d_{LB} is tighter than the one by d_{B}. We can indeed see this by looking back at figures 3.7 and 3.6, where the lax bisimulation metric correctly yields a distance of 0 between two distinct states s_1 and s_2 with a symmetric optimal policy, while the regular bisimulation metric assigns a non-zero distance to them. Note that those results were again only proved in the case that $\mathrm{MDP}_1 = \mathrm{MDP}_2$. Nonetheless, we hypothesize that it should be possible to generalize this result to $\mathrm{MDP}_1 \neq \mathrm{MDP}_2$.

3.2.4 Bisimulation measures between MDPs

Although [11] and [12] only measured distances of states within one MDP, already the definition of bisimulation given in [33] was defined on two distinct MDP instances. Furthermore, lax bisimulation was shown to be equivalent to so called MDP homomorphisms [35] - a specific class of transformations between two MDPs - in [12] and has already been used to transfer policies between two MDPs with different state-spaces in [36]. We guess that the reason for measuring the similarity of states in two distinct MDP instances has not been investigated by [11] and [12] is twofold. First of all, [11] and [12] considered bisimulation metrics as a tool to compress the state-space of an MDP by aggregating those states that are assigned near zero distance. Secondly, the metric interpretation of d_{B} and d_{LB} does not hold in the case $\mathrm{MDP}_1 \neq \mathrm{MDP}_2$ due to the possibly different state spaces S_1 and S_2 in the MDPs. Because of this, the triangle equality

$$d(s_1, s_2) \leq d(s_1, \tilde{s}) + d(\tilde{s}, s_2)$$

is not well-defined anymore as we cannot assume that $\tilde{s} \in S_1$ and $\tilde{s} \in S_2$. And even if this would hold, e.g. if $S_1 = S_2$, the interpretation of this inequality is somewhat crooked, as $\tilde{s}$ is once interpreted as a state in MDP_1 and once in MDP_2. However, in terms of our goal of measuring closeness between two MDP instances, the computed "distance measure" (we will avoid the term metric from now on) between states may still be useful. We want to at least empirically underline this by defining the closeness between two MDPs in terms of the expected difference of their initial states w.r.t to the lax bisimulation metric

$$d_{\mathrm{MDP}}(\mathrm{MDP}_1, \mathrm{MDP}_2) = E_{\mu_0^1, \mu_0^2} \left[d_{\mathrm{LB}}(s_0^1, s_0^2) | s_0^1 \sim \mu_0^1(s), s_0^2 \sim \mu_0^2(s) \right].$$

Having such a distance measure allows to define many different ways of finding context sequences $\{\mathbf{c}_i | i = 0, \ldots, M \wedge \mathbf{c}_M = \mathbf{c}_f\}$ that interpolate between the contextual MDPs $\mathrm{MDP}_{\mathbf{c}_0}$ and $\mathrm{MDP}_{\mathbf{c}_f}$. One of those possibilities is to find the shortest path between $\mathbf{c}_0$ and $\mathbf{c}_f$ while interpreting $d_{\mathrm{MDP}}\left(\mathrm{MDP}_{\mathbf{c}_i}, \mathrm{MDP}_{\mathbf{c}_{i+1}}\right)$ as the cost of moving from $\mathbf{c}_i$ to $\mathbf{c}_{i+1}$. Such a shortest path can easily be computed with standard algorithms such as [37]. The "shortest" path interpolating the contexts $\mathbf{c}_0 = [1\ 2]$ and $\mathbf{c}_f = [3\ 2]$ from the initial example in figure 3.5 is shown in 3.8. Note that we enforced $\|\mathbf{c}_{i+1} - \mathbf{c}_i\|_1 \leq 1$ by pruning the graph of the corresponding shortest path problem accordingly. We see that the optimal interpolation between $\mathbf{c}_0$ and $\mathbf{c}_f$ according to d_{MDP} is indeed different from the linear interpolation. Not surprisingly, this sequence takes a "detour" to avoid the hill in between the initial and the final context. Furthermore, the visualization of d_{MDP} at least suggests that it is indeed a metric. However, whether this really holds or not is left to be determined.

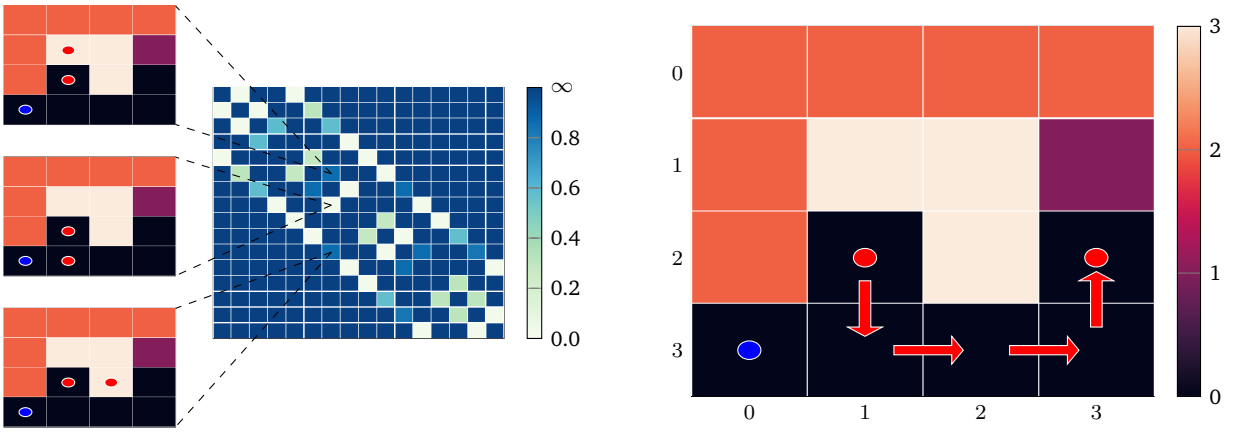

Figure 3.8.: The right plot shows the shortest path (red arrows) between two contexts $\mathbf{c}_0 = [1\ 2]$ and $\mathbf{c}_f = [3\ 2]$ (red dots) in a 4×4 grid-world according to the distance d_{MDP} when the allowed change between two contexts $\|\mathbf{c}_{i+1} - \mathbf{c}_i\|_1$ on the path is limited to 1. The blue dot is the initial position of the agent - which is the same in all grid-worlds. The left plot visualizes the distance d_{MDP}, which was pruned to enforce the aforementioned restriction in context change. The smaller pictures on the very left shows different contexts (red dots), whose distance according to d_{MDP} is marked by dotted lines.

3.2.5 Issues of Bisimulation Metrics

While the previously mentioned lack of mathematical interpretation as well as proofs for the properties of d_{B}, d_{LB} and d_{MDP} in the case of distinct MDP instances is probably easily resolved by a more thorough study of those constructs, we would now like to highlight more fundamental issues.

First of all, it would be necessary to extend the definition of bisimulation and the corresponding metrics to cases in which uncertainty about the reward function is present to be generally applicable in reinforcement learning problems.

Secondly, computing the updates of d_i according to 3.7 and 3.9 are computationally very demading, as we need to solve $|S|^2|A|$ linear programs for 3.7 and $|S|^2|A|^2$ for 3.9. As stated in [11], the complexity of the linear program 3.8 is in $O\left(|S|^2 \log(|S|)\right)$, yielding a total complexity of $O\left(|S|^4|A| \log(|S|)\right)$ and $O\left(|S|^4|A|^2 \log(|S|)\right)$ per update-step 3.7 and 3.9 respectively. This quickly becomes infeasible for any non-trivial number of states and actions. Looking at equation 3.10, we see that both d_{LB} and d_{B} are upper bounds to the difference in value function. With the complexity of operator F_V in $O(|S|^2|A|)$, it seems more reasonable to just compute the value functions for the individual MDP instances and then express the similarity of their initial states in terms of the difference in the value functions.

Allthough there is work on extending the notion of bisimulation and the corressponding metrics to MDPs with continuous state- and action-spaces [38], only theoretical results could be achieved and we do not know of an algorithm that even approximately computes such a metric at the time of writing. This problem seems natural, since the computation of exact value functions in continuous state-action MDPs is known to be infeasible. However, successful approaches for approximating value functions in continuous MDPs may be able to approximate bisimulation metrics for continuous MDPs as well.

3.2.6 MDP measures based on optimal policies

The previously described MDP distance measure based on bisimulation metrics has an, in our opinion, undesirable property, which we want to investigate now. Imagine two MDP instances $\text{MDP}_{\mathbf{c}_1} = (S, A, p, r_{\mathbf{c}_1}, \mu_0)$ and $\text{MDP}_{\mathbf{c}_2} = (S, A, p, r_{\mathbf{c}_2}, \mu_0)$, which only differ in their reward functions $r_{\mathbf{c}_1}$ and $r_{\mathbf{c}_2}$. As shown in appendix A.2, it holds that

$$r_{\mathbf{c}_2}(s,a) = \alpha r_{\mathbf{c}_1}(s,a) + \beta \Rightarrow V^*_{\mathbf{c}_2}(\mathbf{s}) = \alpha V^*_{\mathbf{c}_1}(\mathbf{s}) + \frac{\beta}{1-\gamma}$$

for any given discount factor $\gamma < 1$. From the very definition of the operators F_{LB} and F_{B} we can see that in the case of convergence of these operators, it holds that $d_{\text{LB}}(s_1, s_2) \geq c_R\beta$ and $d_{\text{B}}(s_1, s_2) \geq c_R\beta$ for any state s_1 in $\text{MDP}_{\mathbf{c}_1}$ and any state s_2 in $\text{MDP}_{\mathbf{c}_2}$. Note that this holds in any case, regardless whether the previously mentioned bound on the value

function also holds for $\text{MDP}_1 \neq \text{MDP}_2$. This is somewhat counter-intuitive as both MDPs require the same policy to be solved optimally, which can be easily shown by looking back at the definition of the optimal policy of an MDP

$$\begin{aligned}
\pi^*_{\mathbf{c}_2}(\mathbf{s}) &= \arg\max_{\mathbf{a}} \ \left(r_{\mathbf{c}_2}(s,a) + \gamma E_p\left[V^*_{\mathbf{c}_2}(\tilde{\mathbf{s}})|\tilde{\mathbf{s}} \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})\right]\right) \\
&= \arg\max_{\mathbf{a}} \ \left(\alpha r_{\mathbf{c}_1}(s,a) + \beta + \gamma\left(\alpha E_p\left[V^*_{\mathbf{c}_1}(\tilde{\mathbf{s}})|\tilde{\mathbf{s}} \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})\right] + \frac{\beta}{1-\gamma}\right)\right) \\
&= \arg\max_{\mathbf{a}} \ \alpha\left(r_{\mathbf{c}_1}(s,a) + \gamma E_p\left[V^*_{\mathbf{c}_1}(\tilde{\mathbf{s}})|\tilde{\mathbf{s}} \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})\right]\right) + \beta + \frac{\gamma\beta}{1-\gamma} \\
&= \arg\max_{\mathbf{a}} \ \left(r_{\mathbf{c}_1}(s,a) + \gamma E_p\left[V^*_{\mathbf{c}_1}(\tilde{\mathbf{s}})|\tilde{\mathbf{s}} \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})\right]\right) = \pi^*_{\mathbf{c}_1}(\mathbf{s}).
\end{aligned}$$

With that knowledge, it seems more sensible to express the difference between $\text{MDP}_{\mathbf{c}_1}$ and $\text{MDP}_{\mathbf{c}_2}$ in terms of the expected difference of the actions generated by their optimal policies $\pi^*_{\mathbf{c}_2}$ and $\pi^*_{\mathbf{c}_1}$ along the state trajectory generated by one of the optimal policies

$$\tilde{d}_{\text{MDP}}(\text{MDP}_1, \text{MDP}_2) = E_{\mu_0^{\mathbf{c}_2}, p}\left[\sum_{i=0}^{N} \|\pi^*_{\mathbf{c}_2}(\mathbf{s}_i) - \pi^*_{\mathbf{c}_1}(\mathbf{s}_i)\|_{\mathbf{a}} \middle| \mathbf{s}_0 \sim \mu_0^{\mathbf{c}_2}(\mathbf{s}), \mathbf{s}_i \sim p(\mathbf{s}'|\mathbf{s}_{i-1}, \pi^*_{\mathbf{c}_2}(\mathbf{s}_{i-1}))\right].$$

As you can see, above definition requires some form a norm $\|\cdot\|_{\mathbf{a}}$ defined on the action space. We do not use a specific norm here, as especially for discrete MDPs one is typically required to specify a norm based on domain knowledge. Furthermore, for some MDPs one needs to explicitly express the mapping $- : A \times A \mapsto A$. Regardless of the specific norm and mapping, this approach of measuring MDP similarity has the benefit that the $\arg\max$ operator of the optimal policies removes any constant scaling and offset of the reward functions that would otherwise lead to a nonzero distance according to d_{MDP}. Above distance metric can also be interpreted as computing the optimal policy π^*_{alt} in $\text{MDP}_{\text{alt}} = (S, A, p_{\text{alt}}, r_{\mathbf{c}_2})$ with

$$p_{\text{alt}} : S \times A \times S \mapsto \mathbb{R}, \quad p_{\text{alt}}(\mathbf{s}'|\mathbf{s},\mathbf{a}) = p\left(\mathbf{s}'|\mathbf{s}, \mathbf{a} + \pi^*_{\mathbf{c}_1}(\mathbf{s})\right)$$

and then computing

$$\tilde{d}_{\text{MDP}}(\text{MDP}_1, \text{MDP}_2) = E_{\mu_0^{\mathbf{c}_2}, p_{\text{alt}}}\left[\sum_{i=0}^{N} \|\pi^*_{\text{alt}}(\mathbf{s}_i)\|_{\mathbf{a}} \middle| \mathbf{s}_0 \sim \mu_0^{\mathbf{c}_2}(\mathbf{s}), \mathbf{s}_i \sim p_{\text{alt}}(\mathbf{s}'|\mathbf{s}_{i-1}, \pi^*_{\text{alt}}(\mathbf{s}_{i-1}))\right].$$

Above definition now requires the existence of a function $+ : A \times A \mapsto A$ expressing the result of a combination of two actions instead of the mapping $- : A \times A \mapsto A$. In the previous grid-world example, one could define a mapping encoding that e.g. movements in opposite directions cancel out.

3.2.7 A final thought on MDP measures

As we have seen, it is not an easy task to actually quantify the similarity between MDP instances and there exist a lot of pitfalls in designing such similarity measures as well as computational burdens.
An interesting insight that can be obtained when looking at the presented measures, is that they are all defined as some form of optimization problem themselves. Especially the previously described similarity measure based on optimal policies intrinsically requires to compute the optimal policies for both MDPs before comparing them. For the case of bisimulation metrics, we at least compute an upper bound to the difference in optimal value functions - which in turn encode the optimal policy. This may seem flawed at first glance. On second thought however, it seems natural that it is necessary to have a rough idea about the optimal solution of two tasks in order to judge whether these tasks are similar or not - and especially at which points they differ. This problem becomes very intuitive, if we picture two different MDPs as abstractions of two different board games. Without any knowledge of the rules of both games, making statements about their similarity is basically not possible. However, the more profound the knowledge of both games becomes, the more detailed a player would be able to explain the similarities and differences of both games. This also illustrates that uncertainty about the specific characteristics of a MDP hinders the ability to precisely compare it to other MDPs.
At this point we want to stop thinking about explicit ways of measuring similarities and transferring policies between MDPs and abstract from all those details by taking a probabilistic and episodic view of the reinforcement learning problem. As we will see, this abstraction will allow for a more straightforward definition of a learning algorithm that shares similarities with the motivating predictor-corrector methods while avoiding technicalities.

4 Self-Paced Contextual Reinforcement Learning

In the previous chapter we saw that applying the promising predictor-corrector scheme from chapter 2 to reinforcement learning problems creates several technical as well as conceptual challenges. While all those challenges only stress the complexity of creating distance measures and transferring solutions between Markov Decision Processes (MDPs) and with that show that there is more theoretical work to be done in those areas, we now want to take a viewpoint of reinforcement learning as the optimization of a black-box function $r(\boldsymbol{\theta}, \mathbf{c})$ - just as introduced in chapter 2. This black-box view naturally avoids the challenges that we faced in the previous chapter and allows us to create a reinforcement learning algorithm that gradually transforms the solution of an initial - easy - task to the solution of a desired one. For that we will first introduce the Contextual Relative Entropy Policy Search (C-REPS) algorithm on which we then built our approach that allows for the outlined learning scheme.

4.1 Episodic Contextual Relative Entropy Policy Search

As already introduced, we take the view of (contextual) reinforcement learning as the optimization of a (contextual) black box reward function

$$r : \Theta \times C \mapsto \mathbb{R}$$

with "parameters" $\theta \in \Theta \subseteq \mathbb{R}^{n_{\boldsymbol{\theta}}}$ and contexts $\mathbf{c} \in C \subseteq \mathbb{R}^{n_c}$. We speak of a "black box" function, since the agent initially has no knowledge about r and can only query the function at specific points $(\boldsymbol{\theta}, \mathbf{c})$ to find the optimal parameters $\boldsymbol{\theta}^*$ for a given context $\mathbf{c}$.
The term "episodic" refers to the property that the function $r(\boldsymbol{\theta}, \mathbf{c})$ typically expresses the reward obtained by executing an actual low-level policy $\pi_{\boldsymbol{\theta}}$, whose behavior is defined by the parameters $\boldsymbol{\theta}$, in a MDP. This abstraction will be one of the key elements that allow us to avoid the problems that we faced in the previous chapter.
The probability of encountering a specific context can be expressed using a probability distribution $\mu : C \mapsto [0, \infty]$. This context distribution is typically assumed to be hidden, meaning that an agent cannot influence this distribution. With that, the episodic contextual reinforcement learning objective is defined as

$$J(\pi) = \int_{C,\Theta} r(\boldsymbol{\theta}, \mathbf{c}) \pi(\boldsymbol{\theta}|\mathbf{c}) \mu(\mathbf{c}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{c}, \tag{4.1}$$

where the agent is represented by the non-deterministic policy $\pi : C \times \Theta \mapsto [0, \infty)$ that expresses with which probability it will choose a specific parameter $\boldsymbol{\theta}$ for the low-level policy in a given context. So called policy search algorithms [39, 40, 41] optimize objectives similar to 4.1 by subsequently changing an initial distribution π_0 to create a sequence of distributions $\pi_i,\ i = 1, \ldots$ which assign more and more probability to the optimal parameters in those contexts that can be encountered according to the context distribution. While there are many possible ways how to subsequently change this initial distribution, we will focus on so called C-REPS [14, 15, 16] which bounds the change between two successive distributions π_{i+1} and π_i using the KL-Divergence - a distance measure between two probability distributions. The update of the policy π_i in episodic C-REPS is defined as the solution of the optimization problem

$$\max_{p:\Theta \times C \mapsto \mathbb{R}} \int_{C,\Theta} r(\boldsymbol{\theta}, \mathbf{c}) p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{c} \tag{4.2}$$

$$\text{s.t. } D_{\mathrm{KL}}(p\|q) \leq \epsilon \tag{4.3}$$

$$\int_{C,\Theta} p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{c} = 1 \tag{4.4}$$

$$\int_{\Theta} p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta} = \mu(\mathbf{c}) \quad \forall \mathbf{c} \in C \tag{4.5}$$

with $q(\boldsymbol{\theta}, \mathbf{c}) = \pi_i(\boldsymbol{\theta}|\mathbf{c})\mu(\mathbf{c})$. The optimization is done over the joint distribution $p(\boldsymbol{\theta}, \mathbf{c})$ instead of the policy $\pi(\boldsymbol{\theta}|\mathbf{c})$ as this makes dealing with the optimization problem easier, which we will see later when we derive the solution for a related optimization problem. However, now that we are optimizing over the joint distribution, we need to ensure that the improvement of the objective 4.1 does not result from actually altering the context distribution that is implicitly encoded in the joint distribution p. Constraint 4.5 prevents exactly this behavior. Constraint 4.4 enforces that the probability mass

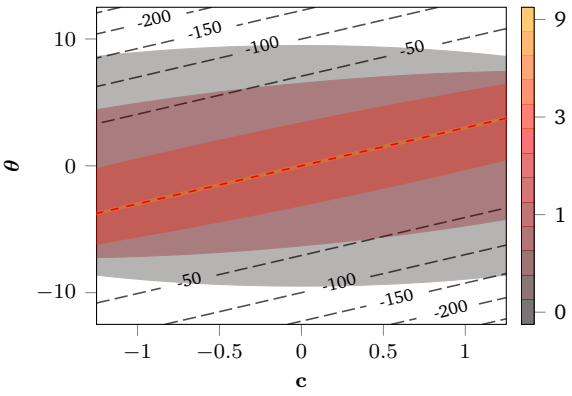

Figure 4.1.: C-REPS example on a simple quadratic function. The contours of the reward function $r(\boldsymbol{\theta}, \mathbf{c}) = (\boldsymbol{\theta} - 3\mathbf{c})^2$ are shown by the black dotted lines. The numbers represent the values that r takes on the contour lines. The red dotted line visualizes the maxima of the function. The shaded areas represent the joint distribution $p(\boldsymbol{\theta}, \mathbf{c}) = \pi(\boldsymbol{\theta}|\mathbf{c})\mu(\mathbf{c})$ at specific C-REPS iterations. The context distribution μ is a Gaussian $\mathcal{N}(0, 1)$ while the policy π is a Gaussian with linear mean $\mathcal{N}(\mathbf{A}\mathbf{c} + \mathbf{b}|\boldsymbol{\Sigma})$. $\mathbf{A}$, $\mathbf{b}$ and $\boldsymbol{\Sigma}$ are learned using weighted linear regression from the set of weighted samples. In each iteration, 100 samples are drawn from r.

of p indeed sums to one and constraint 4.3 is the aforementioned bound in KL-Divergence. The KL-Divergence between two probability distributions $p_1 : \mathbb{R}^n \mapsto [0, \infty]$ and $p_2 : \mathbb{R}^n \mapsto [0, \infty]$ is defined as

$$D_{\mathrm{KL}}(p_1 \| p_2) = \int_{\mathbb{R}^n} p_1(\mathbf{x}) \log\left(\frac{p_1(\mathbf{x})}{p_2(\mathbf{x})}\right) \mathrm{d}\mathbf{x}$$

and in above optimization problem enforces that the change between two consecutive policies π_i and π_{i+1} is limited. Interestingly, the very definition of the KL-Divergence enforces that $p_1(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$ even if we do not assume that p_1 is a valid probability distribution, as the $\log$ function is only defined for values greater than zero. In the C-REPS optimization problem, this, together with constraint 4.4, enforces that p is indeed a probability distribution. Just as done for the constrained optimization problem in section 3.1, we will work with the Lagrangian function

$$\begin{aligned} L(p, \eta, \lambda, V) = &\int_{C,\Theta} r(\boldsymbol{\theta}, \mathbf{c}) p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} \\ &+ \eta\left(\epsilon - D_{\mathrm{KL}}(p\|q)\right) \\ &+ \lambda\left(1 - \int_{C,\Theta} p(\boldsymbol{\theta}, \mathbf{c})\mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c}\right) \\ &+ \int_C V(\mathbf{c})\left(\mu(\mathbf{c}) - \int_\Theta p(\boldsymbol{\theta}, \mathbf{c})\mathrm{d}\boldsymbol{\theta}\right)\mathrm{d}\mathbf{c}, \end{aligned}$$

where the so-called Lagrangian multipliers $\eta, \lambda \in \mathbb{R}$ with $\eta \geq 0$ and $V : C \mapsto \mathbb{R}$ incorporate the three constraints of the C-REPS optimization problem. Interestingly, the multiplier V is a function instead of a scalar, since there is one constraint for every $\mathbf{c} \in C$ and C is a continuous space - which makes constraint 4.5 actually represent an infinite number of constraints. The optimal joint distribution p^* of above optimization problem and with that π_{i+1} can now be expressed in terms of the optimal multipliers η^* and V^* (we will shortly see how those optimal multipliers can be found)

$$\begin{aligned} p^*(\boldsymbol{\theta}, \mathbf{c}) &\propto q(\boldsymbol{\theta}, \mathbf{c}) \exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V^*(\mathbf{c})}{\eta^*}\right) \\ \Leftrightarrow \pi_{i+1}(\boldsymbol{\theta}|\mathbf{c})\mu(\mathbf{c}) &\propto \pi_i(\boldsymbol{\theta}|\mathbf{c})\mu(\mathbf{c}) \exp\left(\frac{\delta(\boldsymbol{\theta}, \mathbf{c})}{\eta^*}\right) \\ \Leftrightarrow \pi_{i+1}(\boldsymbol{\theta}|\mathbf{c}) &\propto \pi_i(\boldsymbol{\theta}|\mathbf{c}) \exp\left(\frac{\delta(\boldsymbol{\theta}, \mathbf{c})}{\eta^*}\right), \end{aligned} \tag{4.6}$$

with $\delta(\boldsymbol{\theta}, \mathbf{c}) = r(\boldsymbol{\theta}, \mathbf{c}) - V^*(\mathbf{c})$. We see that π_{i+1} is an exponential re-weighting of π_i. The "aggressiveness" of this re-weighting is controlled by the multiplier η, where large values of η will lead to a smaller deviation of π_{i+1} from π_i while the deviation will become more pronounced for smaller values. The new probability mass of parameter $\boldsymbol{\theta}$ in

context $\mathbf{c}$ depends on the difference between the reward $r(\boldsymbol{\theta}, \mathbf{c})$ and the optimal Lagrangian multiplier $V^*(\mathbf{c})$. In [42], it was shown that $V^*(\mathbf{c})$ corresponds to some form of expected reward in context $\mathbf{c}$ with the current policy $\pi_i(\boldsymbol{\theta}|\mathbf{c})$, which is why we will refer to it as the value function sometimes as it encodes the "value" of the context $\mathbf{c}$. The difference between $r(\boldsymbol{\theta}, \mathbf{c})$ and $V(\mathbf{c})$ hence expresses how much better parameter $\boldsymbol{\theta}$ is than the "average" parameter in context $\mathbf{c}$. The optimal parameters η^* and V^* can be found by minimizing the so-called dual function

$$\mathcal{G}(\eta, V) = \eta\epsilon + E_\mu\left[V(\mathbf{c})\right] + \eta \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right)\right]\right).$$

This dual function arises from plugging the definition of the joint distribution p with regards to the multipliers back into the Lagrangian. We skip the exact derivation here, since we will face a strongly related optimization problem in later sections for which we will derive the aforementioned results. Above dual function, however, poses several difficulties. First of all, optimizing with respect to the function V is infeasible. Because of this, we need to approximate it with some parametric model, such as e.g. a weighted sum of features

$$V(\mathbf{c}) = \boldsymbol{\omega}^T \phi(\mathbf{c}),$$

with $\boldsymbol{\omega} \in \mathbb{R}^{n_\omega}$ and $\phi : C \mapsto \mathbb{R}^{n_\omega}$. The feature function ϕ needs to be chosen by the user and only $\boldsymbol{\omega}$ is left to be optimized. Besides that, the function r is unknown and we can only gather information by sampling from it. Because of this, the expectation in the dual is approximated by samples from the joint distribution q. With the mentioned adaptations, the dual becomes

$$\mathcal{G}(\eta, \boldsymbol{\omega}) = \eta\epsilon + \boldsymbol{\omega}^T E_\mu\left[\phi(\mathbf{c})\right] + \eta \log\left(\frac{1}{N}\sum_{i=1}^{N}\left[\exp\left(\frac{r(\boldsymbol{\theta}_i, \mathbf{c}_i) - \boldsymbol{\omega}^T\phi(\mathbf{c}_i)}{\eta}\right)\right]\right).$$

Note that while we typically cannot compute the expectation over the context features analytically, we can easily compute an average feature vector $\bar{\phi}$ by computing an average over feature samples $\phi(\mathbf{c}_i)$ from the context distribution $\mu(\mathbf{c})$. Aforementioned approximations also imply that the new policy π_{i+1} cannot be expressed analytically but only in terms of the N samples that were used to approximate the expectation over q in the dual

$$D_\pi = \left\{\left(w_j^\pi, \boldsymbol{\theta}_j, \mathbf{c}_j\right) | w_j^\pi = \exp\left(\frac{\delta(\boldsymbol{\theta}_j, \mathbf{c}_j)}{\eta^*}\right),\ j = 1, \ldots, N\right\}.$$

This set of weighted samples can then be used to infer the new policy π_{i+1} using e.g. weighted linear regression. Figure 4.1 visualizes the functionality of C-REPS on a simple parabolic function $r(\boldsymbol{\theta}, \mathbf{c})$ with a Gaussian context distribution $\mu(\mathbf{c})$ and a linear Gaussian policy $\pi(\boldsymbol{\theta}|\mathbf{c}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{A}\mathbf{c} + \mathbf{b}, \boldsymbol{\Sigma})$, which was updated from the re-weighted samples using weighted linear regression. As expected, the - initially very broad - distribution becomes more and more concentrated on the optimas of the function.

4.2 Self-Paced Contextual Reinforcement Learning

As we have seen, episodic C-REPS allows to find a policy $\pi(\boldsymbol{\theta}|\mathbf{c})$ that maximizes a function $r(\boldsymbol{\theta}, \mathbf{c})$ for various contexts $\mathbf{c}$, whose likelihood is encoded by a hidden context distribution $\mu(\mathbf{c})$. Indeed, the policy already generalizes from the set of weighted samples, with which it is learned, to contexts that were not contained in the set of samples. This behavior is already very desirable when thinking about our goal of generalizing from easy tasks to harder - desired - ones. However, the context distribution that encodes the possibility of encountering a specific task is assumed to be hidden and hence cannot be changed by the agent, which prevents it to gradually sample more and more "complex" contexts. We now resolve this issue by allowing to modify the sampling distribution during learning. This will allow to not only generate sequences of policies $\pi_i, i = 1, \ldots$ but to also generate context distributions $\tilde{\mu}_i, i = 1, \ldots$. The idea is that the initial context distributions assign high probability mass to contexts that are easy to solve while the later distributions increasingly coincide with the desired context distribution μ. We can express this idea as the following optimization problem

$$\max_{p:\Theta\times C\mapsto\mathbb{R}, \tilde{\mu}:C\mapsto\mathbb{R}} \left(\int_{C,\Theta} r(\boldsymbol{\theta}, \mathbf{c}) p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} - \alpha D_{\mathrm{KL}}(\tilde{\mu}\|\mu)\right) \tag{4.7}$$

$$\text{s.t. } D_{\mathrm{KL}}(p\|q) \leq \epsilon \tag{4.8}$$

$$\int_{C,\Theta} p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} = 1 \tag{4.9}$$

$$\int_{\Theta} p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta} = \tilde{\mu}(\mathbf{c}) \quad \forall \mathbf{c} \in C \tag{4.10}$$

Input: Relative entropy bound ϵ,
α-schedule $\tau\left(i\right)$,
initial policy $\pi_0(\boldsymbol{\theta}|\mathbf{c})$,
initial sampling distribution $\mu_0(\mathbf{c})$,
number of iterations K.
for $i = 1$ **to** K **do**
Collect Data:
Sample contexts $\mathbf{c}_j \sim \mu_{i-1}(\mathbf{c}),\ j = 1, \ldots, N$
Sample and execute actions $\boldsymbol{\theta}_j \sim \pi_{i-1}(\boldsymbol{\theta}|\mathbf{c}_j)$
Observe reward $R_j = r\left(\boldsymbol{\theta}_j, \mathbf{c}_j\right)$
Update Policy and Context Distributions:
Optimize dual function:
$\alpha_i = \tau\left(i\right)$
$\left[\eta^*_{\tilde{\mu}}, \eta^*_p, V^*\right] \leftarrow \arg\max \mathcal{G}(\eta_{\tilde{\mu}}, \eta_p, V)$ ▷ Equation 4.15
Calculate sample weights:
$\left[w^{\pi}_j, w^{\tilde{\mu}}_j\right] \leftarrow \left[\exp\left(\frac{\delta(\boldsymbol{\theta}_j, \mathbf{c}_j)}{\eta^*_p}\right), \exp\left(\frac{\beta(\mathbf{c}_j)}{\alpha_i + \eta^*_{\mu}}\right)\right]$
Infer new policy and context distributions:
$\pi_i \leftarrow D_{\pi} = \left\{(w^{\pi}_j, \boldsymbol{\theta}_j, \mathbf{c}_j)|j = 1, \cdots, N\right\}$ ▷ E.g. with weighted linear regression
$\tilde{\mu}_i \leftarrow D_{\tilde{\mu}} = \left\{(w^{\tilde{\mu}}_j, \mathbf{c}_j)|j = 1, \cdots, N\right\}$ ▷ E.g. with weighted maximum likelihood
end for

Algorithm 2: Self Paced Reinforcement Learning

with $q(\boldsymbol{\theta}, \mathbf{c}) = \pi_i(\boldsymbol{\theta}|\mathbf{c})\tilde{\mu}_i(\mathbf{c})$. The multiplier $\alpha \geq 0$ encodes how much we penalize a deviation of the new context distribution $\tilde{\mu}$ from the desired one. In initial iterations, α should hence be set to zero or close to zero. Before continuing with the derivation of the optimal joint- and context distribution according to optimization problem 4.7 as well as the dual, we first need to ensure that the constraints of above optimization problem actually enforce that the optimal $\tilde{\mu}$ is a probability distribution. First of all, the KL-Divergence between $\tilde{\mu}$ and μ in the objective ensures that $\tilde{\mu}(\mathbf{c}) \geq 0$ for all $\mathbf{c} \in C$. Furthermore, constraints 4.9 and 4.10 ensure that the probability mass of $\tilde{\mu}^*$ indeed sums to one as

$$\begin{aligned}
&\int_{C,\Theta} p^*(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} = 1 \\
\Leftrightarrow &\int_C \left(\int_{\Theta} p^*(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta}\right) \mathrm{d}\mathbf{c} = 1 \\
\Leftrightarrow &\int_C \tilde{\mu}^*(\mathbf{c}) \mathrm{d}\mathbf{c} = 1.
\end{aligned}$$

With those two properties, we can be sure that μ^* is guaranteed to be a valid probability distribution. Interestingly, we see that constraints 4.8 and 4.9 together bound the KL-Divergence between $\tilde{\mu}^*$ and $\tilde{\mu}_i$, since

$$\begin{aligned}
&D_{\mathrm{KL}}(p^* \| q) \leq \epsilon \\
\Leftrightarrow &\int_{\Theta,C} p^*(\boldsymbol{\theta}, \mathbf{c}) \log\left(\frac{p^*(\boldsymbol{\theta}, \mathbf{c})}{q(\boldsymbol{\theta}, \mathbf{c})}\right) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} \leq \epsilon \\
\Leftrightarrow &\int_{\Theta,C} p^*(\boldsymbol{\theta}, \mathbf{c}) \log\left(\frac{\tilde{\mu}^*(\mathbf{c})}{\tilde{\mu}_i(\mathbf{c})}\right) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} + \int_{\Theta,C} p^*(\boldsymbol{\theta}, \mathbf{c}) \log\left(\frac{\pi^*(\boldsymbol{\theta}|\mathbf{c})}{\pi_i(\boldsymbol{\theta}|\mathbf{c})}\right) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} \leq \epsilon \\
\Leftrightarrow &\int_C \left(\int_{\Theta} p^*(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta}\right) \log\left(\frac{\tilde{\mu}^*(\mathbf{c})}{\tilde{\mu}_i(\mathbf{c})}\right) \mathrm{d}\mathbf{c} \leq \epsilon - \int_{\Theta,C} p^*(\boldsymbol{\theta}, \mathbf{c}) \log\left(\frac{\pi^*(\boldsymbol{\theta}|\mathbf{c})}{\pi_i(\boldsymbol{\theta}|\mathbf{c})}\right) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} \\
\Leftrightarrow &\int_C \tilde{\mu}^*(\mathbf{c}) \log\left(\frac{\tilde{\mu}^*(\mathbf{c})}{\tilde{\mu}_i(\mathbf{c})}\right) \mathrm{d}\mathbf{c} \leq \epsilon - \int_{\Theta,C} \pi^*(\boldsymbol{\theta}|\mathbf{c})\tilde{\mu}^*(\mathbf{c}) \log\left(\frac{\pi^*(\boldsymbol{\theta}|\mathbf{c})}{\pi_i(\boldsymbol{\theta}|\mathbf{c})}\right) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} \\
\Leftrightarrow &\int_C \tilde{\mu}^*(\mathbf{c}) \log\left(\frac{\tilde{\mu}^*(\mathbf{c})}{\tilde{\mu}_i(\mathbf{c})}\right) \mathrm{d}\mathbf{c} \leq \epsilon - \int_C \tilde{\mu}^*(\mathbf{c}) \left(\int_{\Theta} \pi^*(\boldsymbol{\theta}|\mathbf{c}) \log\left(\frac{\pi^*(\boldsymbol{\theta}|\mathbf{c})}{\pi_i(\boldsymbol{\theta}|\mathbf{c})}\right) \mathrm{d}\boldsymbol{\theta}\right) \mathrm{d}\mathbf{c} \\
\Leftrightarrow &D_{\mathrm{KL}}(\tilde{\mu}^* \| \tilde{\mu}_i) \leq \epsilon - E_{\tilde{\mu}^*}\left[D_{\mathrm{KL}}(\pi^*(\cdot|\mathbf{c}) \| \pi_i(\cdot|\mathbf{c}))\right].
\end{aligned}$$

Appendix A.3 shows how to derive expressions of the joint- and context distribution p and $\tilde{\mu}$ in terms of the optimal Lagrangian multipliers η, λ and V as well as the dual function $\mathcal{G}$, whose minimization allows to find the optimal multi-

pliers. While we still obtain the same formula for the optimal joint distribution as in 4.6, investigating the expression for the optimal context distribution

$$\tilde{\mu}^*(\mathbf{c}) \propto \mu(\mathbf{c}) \exp\left(\frac{V^*(\mathbf{c})}{\alpha}\right)$$

reveals several problems. First of all, we would rather like to express $\tilde{\mu}^*$ in terms of the current context distribution $\tilde{\mu}_i$, as we will only have access to policy samples in contexts drawn from this distribution and hence the estimation of V will only provide meaningful values for those contexts. While we can conceptually alleviate this problem by reformulating above expression

$$\begin{aligned}\tilde{\mu}^*(\mathbf{c}) &\propto \mu(\mathbf{c}) \exp\left(\frac{V^*(\mathbf{c})}{\alpha}\right) \\ &= \mu(\mathbf{c}) \frac{\tilde{\mu}_i(\mathbf{c})}{\tilde{\mu}_i(\mathbf{c})} \exp\left(\frac{V^*(\mathbf{c})}{\alpha}\right) \\ &= \tilde{\mu}_i(\mathbf{c}) \exp\left(\frac{\alpha\left(\log(\mu(\mathbf{c})) - \log(\tilde{\mu}_i(\mathbf{c}))\right) + V^*(\mathbf{c})}{\alpha}\right),\end{aligned}$$

this reformulation, however, cannot alleviate the numerical problems of this specific exponential re-weighting that we want to discuss now. As we have already mentioned, we would like to set $\alpha = 0$ in initial iterations to allow the agent to freely choose the context distribution for learning. However, with $\alpha = 0$, the expression in the exponent is not well defined. And even for values close to zero, this term would drastically amplify the value of the numerator, leading to very large negative or positive values in the exponent, which than lead to numerical over- or underflows of the $\exp$ term.

4.3 Stable Self-Paced Contextual Reinforcement Learning

We have seen that we need to numerically stabilize the optimization problem from the previous section in order to actually make use of it. Interestingly, we can easily achieve this by adding "redundant" constraints to optimization problem 4.7. We say "redundant", because the optimal joint- and context distribution according to 4.7 are already guaranteed to fulfill those constraints by fulfilling constraints 4.8, 4.9 and 4.10. This is because the "redundant" constraints enforce exactly the properties $\int_C \tilde{\mu}^*(\mathbf{c})\mathrm{d}\mathbf{c} = 1$ and $D_{\mathrm{KL}}(\tilde{\mu}^* \| \tilde{\mu}_i) \leq \epsilon$ that we have shown in the previous section. This yields the following optimization problem

$$\max_{p:\Theta\times C\mapsto\mathbb{R}, \tilde{\mu}:C\mapsto\mathbb{R}} \left(\int_{C,\Theta} r(\boldsymbol{\theta},\mathbf{c}) p(\boldsymbol{\theta},\mathbf{c}) \mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} - \alpha D_{\mathrm{KL}}(\tilde{\mu}\|\mu)\right) \tag{4.11}$$

$$\begin{aligned}\text{s.t. } & D_{\mathrm{KL}}(p\|q) \leq \epsilon \\ & D_{\mathrm{KL}}(\tilde{\mu}\|\tilde{\mu}_i) \leq \epsilon \qquad (4.12) \\ & \int_{C,\Theta} p(\boldsymbol{\theta},\mathbf{c})\mathrm{d}\boldsymbol{\theta}\mathrm{d}\mathbf{c} = 1 \\ & \int_C \tilde{\mu}(\mathbf{c})\mathrm{d}\mathbf{c} = 1 \qquad (4.13) \\ & \int_\Theta p(\boldsymbol{\theta},\mathbf{c})\mathrm{d}\boldsymbol{\theta} = \tilde{\mu}(\mathbf{c}) \quad \forall \mathbf{c} \in C.\end{aligned}$$

As shown in appendix A.4, this again does not change the expression for p (we only renamed the multiplier η to η_p) but allows to define the optimal policy $\tilde{\mu}^*$ in a numerically much more stable way

$$\tilde{\mu}^*(\mathbf{c}) \propto \tilde{\mu}_i(\mathbf{c}) \exp\left(\frac{\alpha\left(\log\left(\mu(\mathbf{c})\right) - \log\left(\tilde{\mu}_i(\mathbf{c})\right)\right) + V^*(\mathbf{c})}{\alpha + \eta^*_{\tilde{\mu}}}\right) = \tilde{\mu}_i(\mathbf{c}) \exp\left(\frac{\beta(\mathbf{c})}{\alpha + \eta^*_{\tilde{\mu}}}\right) \tag{4.14}$$

where $\eta_{\tilde{\mu}}$ is the multiplier that corresponds to constraint 4.12. We see that exactly this multiplier allows to set α to zero, as it prevents the denominator in the exponent from becoming zero. Readers that are familiar with constrained optimization problems may question this anticipated effect, as an important property of Lagrangian multipliers that arise from inequality constraints is that these multipliers are zero if the corresponding constraint is not active. In the case of $\eta_{\tilde{\mu}}$, this implies that this multiplier is zero, if the KL-Divergence $D_{\mathrm{KL}}(\tilde{\mu}^*\|\tilde{\mu}_i)$ is smaller than ϵ. This will, however, be a very likely event given that any deviation of π_{i+1} from π_i implies that the KL-Divergence $D_{\mathrm{KL}}(\tilde{\mu}^*\|\tilde{\mu}_i)$ is strictly smaller than ϵ. In this likely case, the multiplier $\eta_{\tilde{\mu}}$ would not stabilize above expression, as it becomes zero during the

optimization. To argue why this problem does not arise in practice, we need to take a look at the dual function, with which we find the optimal multipliers η_p^*, $\eta_{\tilde{\mu}}^*$ and V^* through minimization and whose derivation can be reviewed in appendix A.4

$$\mathcal{G}(\eta_p, \eta_{\tilde{\mu}}, V) = \eta_p \epsilon + \eta_p \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta_p}\right)\right]\right) + \eta_{\tilde{\mu}}\epsilon + (\alpha + \eta_{\tilde{\mu}}) \log\left(E_{\tilde{\mu}_i}\left[\exp\left(\frac{\alpha\left(\log\left(\mu(\mathbf{c})\right) - \log\left(\tilde{\mu}_i(\mathbf{c})\right)\right) + V(\mathbf{c})}{\alpha + \eta_{\tilde{\mu}}}\right)\right]\right).$$

This dual function again allows for a straightforward approximation of the expectations over q and $\tilde{\mu}_i$ using samples from q. Note that context samples from q are also samples from $\tilde{\mu}_i$, since $q(\boldsymbol{\theta}, \mathbf{c}) = \pi_i(\boldsymbol{\theta}|\mathbf{c})\tilde{\mu}_i(\mathbf{c})$. With the already introduced approximation of V using a weighted sum of features we obtain

$$\mathcal{G}(\eta_p, \eta_{\tilde{\mu}}, \boldsymbol{\omega}) = \eta_p \epsilon + \eta_p \log\left(\frac{1}{N}\sum_{j=1}^{N} \exp\left(\frac{r(\boldsymbol{\theta}_j, \mathbf{c}_j) - \boldsymbol{\omega}\phi(\mathbf{c}_j)}{\eta_p}\right)\right) + \eta_{\tilde{\mu}}\epsilon + (\alpha + \eta_{\tilde{\mu}}) \log\left(\frac{1}{N}\sum_{j=1}^{N} \exp\left(\frac{\alpha\left(\log\left(\mu(\mathbf{c}_j)\right) - \log\left(\tilde{\mu}_i(\mathbf{c}_j)\right)\right) + \boldsymbol{\omega}\phi(\mathbf{c}_j)}{\alpha + \eta_{\tilde{\mu}}}\right)\right). \tag{4.15}$$

We think that exactly these approximations of the expectations over q and $\tilde{\mu}_i$ and the value function prevent $\eta_{\tilde{\mu}}$ from becoming zero in practice, as the discussed implications of $D_{\mathrm{KL}}(p^*\|q^*)$ on $D_{\mathrm{KL}}(\tilde{\mu}^*\|\tilde{\mu}_i)$ only hold in the case of an exact representation of all expectations and the value function. Just as before, we can now approximate $\tilde{\mu}_{i+1}$ from the set of weighted samples from $\tilde{\mu}_i$

$$D_{\tilde{\mu}} = \left\{\left(w_j^{\tilde{\mu}}, \mathbf{c}_j\right) | w_j^{\tilde{\mu}} = \exp\left(\frac{\beta(\mathbf{c}_j)}{\alpha + \eta_{\tilde{\mu}}^*}\right),\ j = 1, \ldots, N\right\}$$

using e.g. a weighted maximum likelihood approach. Algorithm 2 outlines the resulting algorithm which can be used to compute a sequence of policies π_i and context distributions $\tilde{\mu}_i$. The algorithm requires the specification of a so called "alpha-schedule", i.e. a function $\tau(i)$ encoding the value of α that should be used in optimization problem 4.11 in iteration i.

Please note that this algorithm introduces an approximation error by using the weighted set of samples D_π that we introduced in the previous section to infer the new policy. Although the expression for p^* in terms of the optimal multipliers has not changed compared to C-REPS, we would nonetheless need to adjust the weights in the set D_π, as we have

$$\begin{aligned}
& p^*(\boldsymbol{\theta}, \mathbf{c}) \propto q(\boldsymbol{\theta}, \mathbf{c}) \exp\left(\frac{\delta(\boldsymbol{\theta}, \mathbf{c})}{\eta_p^*}\right) \\
\Leftrightarrow & \pi^*(\boldsymbol{\theta}|\mathbf{c})\tilde{\mu}^*(\mathbf{c}) \propto \pi_i(\boldsymbol{\theta}|\mathbf{c})\tilde{\mu}_i(\mathbf{c}) \exp\left(\frac{\delta(\boldsymbol{\theta}, \mathbf{c})}{\eta_p^*}\right) \\
\Leftrightarrow & \pi^*(\boldsymbol{\theta}|\mathbf{c}) \propto \pi_i(\boldsymbol{\theta}|\mathbf{c}) \frac{\tilde{\mu}_i(\mathbf{c})}{\tilde{\mu}^*(\mathbf{c})} \exp\left(\frac{\delta(\boldsymbol{\theta}, \mathbf{c})}{\eta_p^*}\right) \\
\Leftrightarrow & \pi^*(\boldsymbol{\theta}|\mathbf{c}) \propto \pi_i(\boldsymbol{\theta}|\mathbf{c}) \exp\left(\frac{\delta(\boldsymbol{\theta}, \mathbf{c})}{\eta_p^*} - \frac{\beta(\mathbf{c})}{\alpha + \eta_{\tilde{\mu}}^*}\right).
\end{aligned}$$

However, using this exponential re-weighting in practice led to numerical problems in our experiments. Those problems occurred because the $\beta(\mathbf{c}_j)/(\alpha + \eta_{\tilde{\mu}}^*)$ terms could be several orders of magnitude larger than the $\delta(\boldsymbol{\theta}_j, \mathbf{c}_j)/\eta_p^*$ terms. Because of this, all samples then got assigned a near equal weight, leading to a poor policy improvement even in initial iterations (where there clearly exist poor samples $\boldsymbol{\theta}_j$). We guess that this problem arises due to the limited number of samples with which the expectations in both duals are approximated. Especially, there is typically only one sample $\boldsymbol{\theta}_j$ for a given context $\mathbf{c}_j$.

We want to conclude this section by investigating whether the outlined scheme will actually focus on easy tasks for $\alpha = 0$ as we anticipated. To argue that the algorithm indeed does, we investigate the expression for $\tilde{\mu}^*$ with $\alpha = 0$

$$\tilde{\mu}^*(\mathbf{c}) \propto \tilde{\mu}_i(\mathbf{c}) \exp\left(\frac{V^*(\mathbf{c})}{\eta_{\tilde{\mu}}}\right).$$

We see that the new distribution will assign higher probability mass to those contexts with a higher value of $V^*(\mathbf{c})$. As already mentioned, it was shown that $V^*(\mathbf{c})$ corresponds to a softmax operator of the average reward in context $\mathbf{c}$. Hence, $\tilde{\mu}^*$ will assign higher probability mass to contexts with higher average reward. Assuming that the maximum obtainable reward does not differ between contexts, a higher average reward hence means that the agent performs better in these contexts. Consequently, $\tilde{\mu}^*$ indeed focuses on those tasks in which it can more easily obtain a high reward, if it is not forced to match the target distribution μ by $\alpha \neq 0$.

4.4 Numerically Stable Log-Sum-Exp Computation

An important aspect of both the C-REPS and Self-Paced Contextual Reinforcement Learning (SPRL) algorithm is the proper handling of the log-sum-exp terms in the dual. We will abstract from the actual formulas and exemplify the idea on the following log-sum-exp term

$$\log\left(\sum_{i=1}^{N}\exp(f(\mathbf{x}_i))\right).$$

The main problem here is that although the $\log$ and the $\exp$ term conceptually cancel out each other, the individual $\exp(f(\mathbf{x}_i))$ terms in the sum may still become very large and hence lead to numerical overflows. However, we can reformulate above expression to obtain a much more stable expression

$$\begin{aligned}
&\log\left(\sum_{i=1}^{N}\exp(f(\mathbf{x}_i))\right)\\
=\ &\log\left(\sum_{i=1}^{N}\exp(f(\mathbf{x}_i)-f_{\max}+f_{\max})\right)\\
=\ &\log\left(\exp(f_{\max})\sum_{i=1}^{N}\exp(f(\mathbf{x}_i)-f_{\max})\right)\\
=\ &\log\left(\sum_{i=1}^{N}\exp(f(\mathbf{x}_i)-f_{\max})\right)+f_{\max},
\end{aligned}$$

where $f_{\max}=\max_{i\in[1,N]}f(\mathbf{x}_i)$. This ensures that the $\exp(f(\mathbf{x}_i)-f_{\max})$ terms attain a maximum value of 1. If we hence face a large range of values $f(\mathbf{x}_i)$, the $\exp$ term of the smaller values $f(\mathbf{x}_i)$ will be very close to zero or even underflow. This is actually not that problematic, which we can see by looking at the update equations 4.6 and 4.3 of the joint- and context distribution, which can be expressed in an abstract way as the following equation

$$p(\mathbf{x}_i)\propto q(\mathbf{x}_i)\exp(f(\mathbf{x}_i)).$$

We obviously also need to subtract $f_{\max}$ in the $\exp$ terms of these equations to avoid overflows, which yields

$$\begin{aligned}
p(\mathbf{x}_i)\ \propto&q(\mathbf{x}_i)\exp(f(\mathbf{x}_i))\\
=&q(\mathbf{x}_i)\exp(f(\mathbf{x}_i)-f_{\max}+f_{\max})\\
=&q(\mathbf{x}_i)\exp(f(\mathbf{x}_i)-f_{\max})\exp(f_{\max})\\
\propto&q(\mathbf{x}_i)\exp(f(\mathbf{x}_i)-f_{\max}).
\end{aligned}$$

We see that if one of the $\exp(f(\mathbf{x}_i)-f_{\max})$ terms in above equations becomes zero, this only means that the corresponding context $\mathbf{c}$ or parameter $\boldsymbol{\theta}$ should not be assigned considerable probability mass by the new context- or joint-distribution anyways. Hence, with the approach of subtracting $f_{\max}$ from the $f(\mathbf{x}_i)$ terms, we only loose information due to the limited floating point precision for samples which are not "important" anyways.

5 Experiments

In this section, we want to evaluate whether the introduced Self-Paced Contextual Reinforcement Learning (SPRL) algorithm can hold its promise of learning to accomplish complicated tasks by training on a sequence of intermediate tasks that guide its learning - just as predictor-corrector methods were able to compute roots of a "complicated" function by finding roots for a sequence of "easier" functions. We choose three different environments for this evaluation. In the experiments, we compare our algorithm to Contextual Relative Entropy Policy Search (C-REPS), which serves as a representative for "default" policy search algorithms which do not have control over a context distribution. We think that this comparison is the most natural one as SPRL results from a straightforward extension of C-REPS but otherwise shares many similarities which truely allow to investigate the benefit of having control over the otherwise hidden context distribution. The first environment is the task of steering a point-mass through a gate of varying size and position in order to reach a goal position that already served as a motivation in chapter 2. The second task is the three-dimensional reaching task implemented in the OpenAI Gym simulation environment [43], where the end-effector of a robotic arm needs to be controlled to reach various goal positions. The final environment is a modified version of this reaching task, which includes additional obstacles.

In all experiments, we use Radial Basis Function (RBF) features to approximate the value function $V(\mathbf{c})$. We represent the policies $\pi_i(\boldsymbol{\theta}|\mathbf{c})$ by a Gaussian whose mean is a linear weighting of the context features , i. e.

$$\pi_i(\boldsymbol{\theta}|\mathbf{c}) = \mathcal{N}\left(\boldsymbol{\theta}|\mathbf{A}_{\pi_i}\phi_\pi(\mathbf{c}), \boldsymbol{\Sigma}_{\pi_i}\right).$$

The features ϕ_π are chosen to be linear in the first two environments and are chosen to be RBF features in the third one. The weighting $\mathbf{A}_{\pi_i}$ and covariance matrix $\boldsymbol{\Sigma}_{\pi_i}$ are learned from D_π using weighted linear regression for C-REPS and SPRL. In order to solely evaluate the benefits of the generated learning curriculum, C-REPS and SPRL use the same number of RBF features to approximate $V(\mathbf{c})$ and $\pi_i(\boldsymbol{\theta}|\mathbf{c})$ for a given environment. We choose to represent the context distributions $\tilde{\mu}_i(\mathbf{c})$ by Gaussian distributions, which are approximated via maximum-likelihood estimation on $D_{\tilde{\mu}}$ after every iteration for SPRL.

One iteration of SPRL and C-REPS consists of $M = 100$ policy executions, which are generated by sampling a task from $\tilde{\mu}_i(\mathbf{c})$ or $\mu(\mathbf{c})$, for SPRL and C-REPS respectively, and then sampling from $\pi_i(\boldsymbol{\theta}|\mathbf{c})$ to evaluate the reward function using those samples.

In all experiments, we have some form of a control policy $\pi_{\boldsymbol{\theta}} : S \mapsto A$ or action representation $f_{\boldsymbol{\theta}} : [0, N] \mapsto A$ that are parameterized by $\boldsymbol{\theta}$ and generate the actions during the experiments, whose outcomes are encoded by the reward

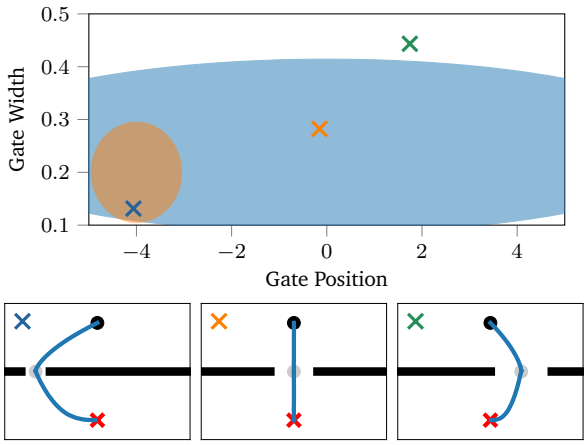

Figure 5.1.: The small pictures show gates of different positions and widths, that need to be traversed in order to reach the goal (red cross) from the starting position (black dot). Crosses of matching color mark the position of those tasks in the task space of the gate environment. Finally, the orange and blue areas show three times standard deviation of the task distributions for the "precision" and "global" setting.

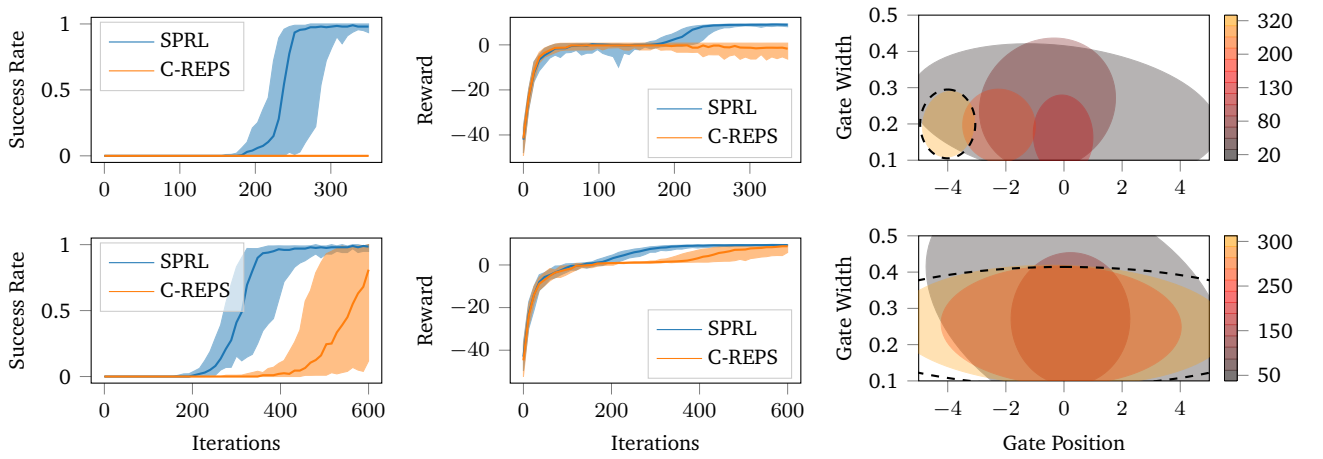

Figure 5.2.: The left plots show the success rate and reward achieved by SPRL and C-REPS on the "precision" (top row) and "global" setting (bottom row) of the gate environment. Thick lines represent the 50%-quantiles and shaded areas show the intervals from 10%- to 90%-quantile. The quantiles were computed using 50 algorithm executions. The right plot shows the evolution of the sampling distribution $\tilde{\mu}(\mathbf{c})$ of one run (colored areas) together with the target distribution $\mu(\mathbf{c})$ (dotted line). Later iterations are represented by brighter colors and iteration numbers are shown in the legend on the right.

function $r(\boldsymbol{\theta}, \mathbf{c})$. The interval $[0, N]$ corresponds to individual time steps for which actions are generated. The reward function for all of the three tasks is an exponential of the final distance to the desired state with additional action penalties

$$r(\boldsymbol{\theta}, \mathbf{c}) = \kappa \, \exp\left(-\|\mathbf{x}_f(\boldsymbol{\theta}) - \mathbf{x}_g(\mathbf{c})\|_2\right) - \nu \sum_{i=0}^{N} \mathbf{a}_i(\boldsymbol{\theta})^T \mathbf{a}_i(\boldsymbol{\theta}).$$

In above formula, $\mathbf{x}_f(\boldsymbol{\theta})$ is the position of the point-mass or end-effector at the end of the policy execution or after applying all feed-forward actions, $\mathbf{x}_g(\mathbf{c})$ the desired final position for the task and $\mathbf{a}_i(\boldsymbol{\theta})$ the action applied at time-step i. The two multipliers κ and ν are chosen individually for each task and will be described for each environment individually.
Besides the obtained rewards, we also evaluate the success rate of both algorithms. Following the metric of the OpenAI Gym reacher environment, an experiment is said to be successful, if the distance between final- and desired state is less than 0.05

$$\text{Success}(\boldsymbol{\theta}, \mathbf{c}) = \begin{cases} 1, & \text{if } \|\mathbf{x}_f(\boldsymbol{\theta}) - \mathbf{x}_g(\mathbf{c})\|_2 < 0.05 \\ 0, & \text{else.} \end{cases}$$

In all experiments, we start with a wide initial sampling distribution $\tilde{\mu}_0$ to allow the SPRL algorithm to automatically choose the initial tasks on which learning should take place. To allow the algorithm to make this choice, we set α to zero or close to zero in the initial iterations of the experiments and then progressively increase its value. As the tasks have different reward structures and target distributions, the exact α-schedules τ were chosen for each task individually and are shown in appendix A.5.

5.1 Gate Environment

Although we have already outlined the experiment details in chapter 2, we want to recapitulate the experiment setup before presenting the results to spare the reader the effort to leaf back and forth between pages. In this environment, the agent needs to steer a point-mass in two-dimensional space from the starting position $[0 \;\; 5]^T$ to the origin. The dynamics of the point mass are described by a simple perturbed linear system

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} 5 \\ -1 \end{pmatrix} + \mathbf{a} + \boldsymbol{\delta}, \quad \boldsymbol{\delta} \sim \mathcal{N}\left(\mathbf{0}, 10^{-3}\mathbf{I}\right),$$

where x is the x-position of the point mass, y its y-position and $\mathbf{a}$ the action of the agent. Complexity is introduced by a wall at height $y = 2.5$, which can only be traversed through a gate. The x-position and width of the gate together

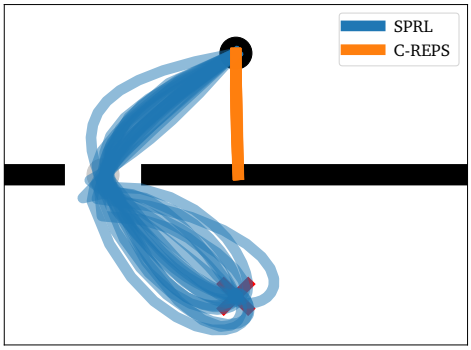

Figure 5.3.: Trajectories generated by the final policies of SPRL and C-REPS for the "precision" setting in the gate environment. Starting from the black dot, the goal is to traverse the gate and reach the desired position, which is marked by a red cross.

define a task $\mathbf{c}$. If the point-mass crashes into the wall, the experiment is stopped and the reward computed based on the current position.
We evaluate two setups in this gate environment: In the first one, the agent needs to be able to steer through a very small gate far from the origin ("precision") and in the second it is required to steer through gates with a variety of positions and widths ("global"). Figure 5.1 illustrates the environment and the two settings that are used for evaluation.
The point-mass is controlled by two PD-Controllers

$$\mathrm{PD}_i(x, y) = \mathbf{K}_i \begin{pmatrix} \tilde{x}_i - x \\ \tilde{y}_i - y \end{pmatrix} + \mathbf{k}_i, \ \mathbf{K}_i \in \mathbb{R}^{2\times 2}, \ \mathbf{k}_i \in \mathbb{R}^2, \ \tilde{x}_i, \tilde{y}_i \in \mathbb{R},$$

whose parameters need to be tuned by the agent. The first controller PD_1 is responsible for steering the point mass towards the gate, which is always at height $y = 2.5$. When reaching this height, the second PD-Controller becomes active and then needs to drive the point-mass to the origin. Only $\tilde{y}_1$ and $\tilde{y}_2$ are fixed to 2.5 and 0, while all other parameters are controlled by the policy π, making $\boldsymbol{\theta}$ a 14-dimensional vector. For this environment, we set the multipliers κ and ν of the reward function to 100 and 10^{-4} respectively. Furthermore, we clip all rewards to a minimum value of -100, as very large negative rewards can occur in initial iterations of the algorithm if the feedback matrices $\mathbf{K}_i$ contain negative values on their diagonal. These negative entries make the control signals grow out of bounds over time, leading to a very large penalty for the generated actions.
Figure 5.2 visualizes the obtained rewards and success rate for the SPRL and C-REPS algorithm in both settings as well as the evolution of the sampling distribution $\tilde{\mu}(\mathbf{c})$. We can see that the SPRL algorithm performs significantly better than C-REPS. Especially in the "precision" setting, only SPRL is able to find a controller that successfully traverses the wall by avoiding the local optima, to which C-REPS converges. With C-REPS achieving a success rate of zero, it is imaginable that this local optima does not encode desirable behavior. However, figure 5.3 demonstrates how poor the resulting behavior actually is.
Besides that, the visualized sampling distributions in Figure 5.2 show that in early iterations of both settings, $\tilde{\mu}$ focuses on tasks with wide gates positioned at the origin and then subsequently changes the sampling distributions to match

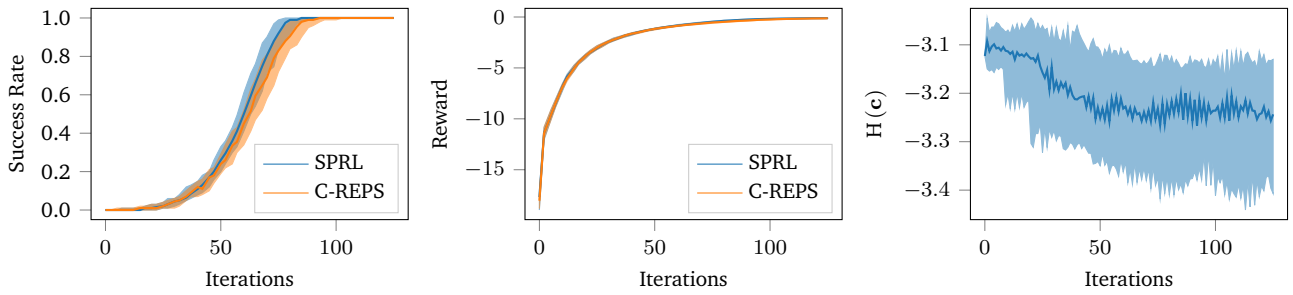

Figure 5.4.: The success rate and reward achieved by SPRL and C-REPS in the reacher environment are shown in the left plots, where thick lines represent the 50%-quantiles and shaded areas show the intervals from 10%- to 90%-quantile. The right plot shows those quantiles for the entropy of $\tilde{\mu}(\mathbf{c})$ over the SPRL iterations. The quantiles were computed using 50 trials.

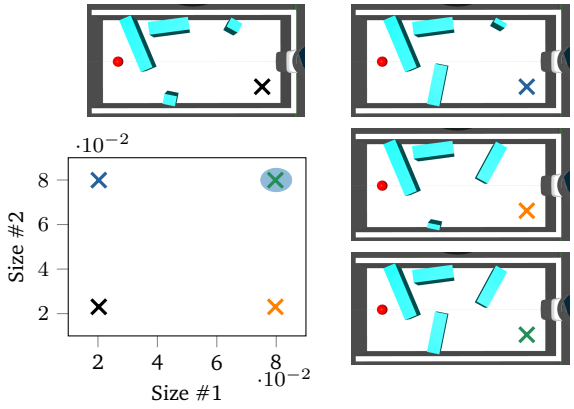

Figure 5.5.: The small pictures show a top-down view on the table and the obstacles (cyan boxes) on top of it in different tasks of the modified reacher environment. The goal in each task is to move the robot end-effector from its initial position at the right end of the table to the red dot, while avoiding the obstacles. Crosses of matching color show to which areas of the task space the individual tasks correspond. The target distribution is visualized by the blue shaded area, which represents three times its standard deviation.

the desired ones. Looking back at figure 2.5, it seems that $\tilde{\mu}$ indeed focuses on easy tasks in early iterations, as a wider gate leads to a higher chance of traversing the wall even with a random policy. While figure 2.5 does not explain why $\tilde{\mu}$ initially favors gates with an x-position near zero, this can easily done by closer examination of the initial policy π_0, which is chosen to be a zero mean Gaussian with "wide" variance for all experiments, i.e. $\mathbf{\Sigma}_{\pi_0} = \zeta \mathbf{I}$ with a large value of ζ. Despite the large initial variance, the chance of sampling a parameter vector $\boldsymbol{\theta}$ with near zero entries is higher than sampling one with larger positive or negative values. Since $\boldsymbol{\theta}$ also contains the desired x-positions of the two PD-Controllers $\tilde{x}_1$ and $\tilde{x}_2$, it is more likely that the initial PD-Controllers will steer the points mass near the positions $[0\ 2.5]$ and $[0\ 0]$. This leads to a higher probability of successfully traversing the wall with a gate whose x-position is near zero for initial controllers. It hence looks like the capabilities of SPRL to control the context distribution also allow to deal with a sub-optimal initial policy π_0 for the desired task by first focusing on tasks in which this initial policy performs well.

5.2 Reacher Environment

The gate environment seems to consist of tasks with varying difficulty. Starting from a random policy, it seems to be easier to find controllers that navigate the agent through a wider gate at the origin than through a narrower one far away from it, especially with the initial policy being a zero-mean Gaussian.
The experiments in the following environment serve to evaluate the performance of SPRL when there are no tasks which are inherently more or less complicated than others. For that evaluation, we choose the three-dimensional reacher environment of the OpenAI Gym toolkit. In this environment, a task is defined by the three-dimensional end-effector position of a robotic arm that should be reached at the end of the trajectory. The desired end-effector position is drawn uniformly from

$$C = [1.19, 1.49] \times [0.60, 0.90] \times [0.38, 0.68] \subset \mathbb{R}^3.$$

We believe that in this setting, there are no hard and easy tasks, since the reward signal always smoothly guides the agent towards the optimal policy. This is not the case in the previous environment, where the possibility of crashing into the wall creates plateaus and discontinuities in the reward function as can be again seen in figure 2.5.
Since the actions in this environment correspond to the desired changes dx, dy and dz in the x-, y- and z-position of the end-effector, the controls do not need to take the dynamics of the robot into the account. Hence we choose to make the actions only depend on the current time-step t_i. These actions are encoded using Gaussian Basis Functions over time

$$\begin{aligned}
\mathbf{a}_i(\boldsymbol{\theta}) &= \boldsymbol{\theta}^T \boldsymbol{\psi}(t_i), \\
\boldsymbol{\psi}_j(t_i) &= \frac{b_j(t_i)}{\sum_{l=1}^{L} b_l(t_i)}, \\
b_j(t_i) &= \exp\left(\frac{(t_i - c_j)^2}{0.6L}\right), \quad j = 1, \ldots, L,
\end{aligned}$$

such as e.g. used in [44]. We evenly spread the $L = 10$ centers c_j over the time interval of the movement, which gives $\boldsymbol{\theta} \in \mathbb{R}^{10\times 3}$, as $\mathbf{a}_i(\boldsymbol{\theta})$ is a three-dimensional vector. The multipliers of the reward function are set to $\kappa = 10$ and

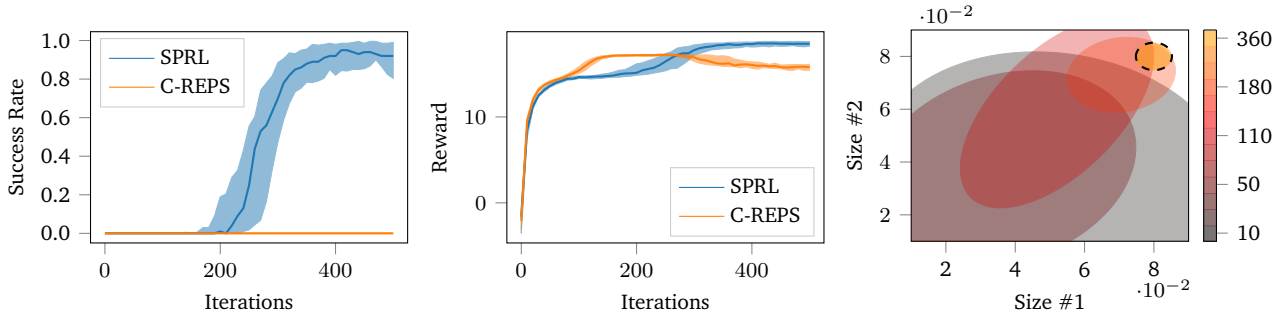

Figure 5.6.: The left plots show the 50%-quantiles (thick lines) and the intervals from 10%- to 90%-quantile (shaded areas) of success rate and reward achieved by SPRL and C-REPS in the modified reacher environment. The quantiles were computed using 25 algorithm executions. Colored areas in the right plot show the sampling distribution $\tilde{\mu}(\mathbf{c})$ at different iterations of one SPRL run together with the target distribution (dotted line). The legend on the right shows the iteration that corresponds to a specific color.

$\nu = 10^{-2}$. We see that the value of ν is considerably higher than in the previous experiment. This is because the actions encode the desired change in end-effector position. Looking at the space C of possible goal positions, we see that the actions hence do not need to be as large as in the previous environment, as e.g. only a total displacement of 0.3 in the x-position is necessary to move across C.
Investigating figure 5.4, we can see that the learning curves of SPRL and C-REPS are basically identical. This matches our belief that a learning curriculum cannot significantly improve the learning performance in this environment, since all tasks provide very good reward signals to learn from. Furthermore, the entropy of the initially wide sampling distribution is not drastically reduced by the SPRL algorithm. This indicates that SPRL covered the set of target tasks right away instead of first focusing on a specific subset. Note that we choose to plot the entropy of the sampling distribution, since actual visualizations of $\tilde{\mu}(\mathbf{c})$ are somewhat problematic with $\mathbf{c}$ being three-dimensional.

5.3 Modified Reacher Environment

In order to demonstrate the beneficial properties of SPRL in the previously investigated reacher environment, we introduce obstacles that the agent needs to avoid while moving towards a goal position. These obstacles lie on a table and the end-effector needs to be moved along the surface of this table towards the goal position, which is at the opposite end of the table. Hence, the agent truly needs to navigate through the obstacles to reach the goal and cannot just avoid the obstacles by moving the robotic arm over the obstacles.
With the obstacles becoming larger, the robot needs to introduce a more and more pronounced curve movement in order to reach the goal without collisions. To allow for an easy visualization of the task distribution, we only allow two of the four obstacles to vary in size. The sizes of those two obstacles make up a task $\mathbf{c}$ in this environment.
Figure 5.5 visualizes the target task distribution and illustrates different tasks that occur in this environment. Just as in the first environment, the robot should not crash into the obstacles and hence the movement is stopped if one of the four obstacles is touched.

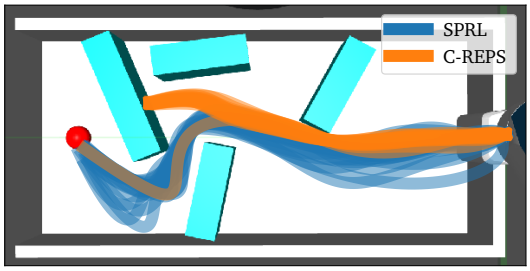

Figure 5.7.: Trajectories generated by the final policies learned with SPRL and C-REPS in the modified reacher environment. The trajectories should reach the red dot while avoiding the cyan boxes. Please note that the visualization is not completely accurate, as we did not account for the viewpoint of the simulation camera when plotting the trajectories.

The actions are again encoded using Gaussian Basis Functions over time, as the actions again encode the desired change in end-effector position and hence do not need to account for possibly complicated dynamics of the robot. However, since the robot is forced to move along the surface of the table, it can only control the movement of its end-effector in the xy-plane, while the position in the z-plane is fixed. With $L = 20$ basis functions, we hence obtain $\boldsymbol{\theta} \in \mathbb{R}^{20\times 2}$. The multipliers for the reward function are set to $\kappa = 20$ and $\nu = 10^{-2}$. The higher value of κ is explained as additional movement needs to be introduced to the end-effector compared to the previous reacher environment. The resulting high penalty is compensated by the higher value of κ.
When looking at figure 5.5, one can imagine that it is hard to solve the target tasks starting from a random policy, as they require a precise curve movement to reach the desired position without crashing into the obstacles. The performance comparison of both algorithms in Figure 5.6 confirms this assumption, showing that SPRL is able to find a better optimum of the reward function and that, just as in the "precision" setting of the gate environment, reaching the one or the other optimum decides whether the resulting policy is able to solve the task. This issue is as also illustrated in Figure 5.7, which visualizes the final policies of SPRL and C-REPS. The figure also shows the only algorithm execution of C-REPS, in which the final policy solves the target tasks. This indicates that indeed the high likelihood of finding a poor local optima causes the low performance of C-REPS. Note that this one successful run of C-REPS is not reflected by the plots in Figure 5.6, since the 10%- and 90%-quantiles remove such outliers from the visualization.
Furthermore, the distributions shown in figure 5.6 indicate that again, SPRL focuses on easier tasks with smaller obstacle sizes first and then moves on to the harder desired tasks. Especially as opposed to the first environment, the easiness of tasks is not implicitly influenced by the initial policy π_0, as a zero-mean Gaussian in combination with the Gaussian Basis Functions over time assigns the most probability to a sequence of actions that do not move the end-effector at all. Such a behavior is equally good or bad in all contexts.

6 Related Work

In this section, we want to briefly discuss ideas that share similarities to the introduced Self-Paced Contextual Reinforcement Learning (SPRL) algorithm.

Obviously, our method shares conceptual ideas with homotopic continuation methods [17]. Tracing the roots of a function for which those are easy to find towards the roots of those for which a default root search cannot be expected to be successful corresponds to the idea of SPRL of first training in easy contexts and then subsequently transferring the found solution to the one of the desired context. In behavioral psychology, a similar concept is known as shaping [45], and has shown to have direct links to computational reinforcement learning [46].

The idea of co-evolving the task together with the learner was explored under different names in various contexts. In evolutionary robotics, simulation parameters describing a robot were gradually evolved to match the observations from the real system, while intermediate controllers were learned entirely in simulation [47]. Recently, this idea got into the spotlight of reinforcement learning under the name 'sim-to-real transfer' [48].

In supervised learning, the paradigm of curriculum learning [49] shares strong similarities to our approach. In curriculum learning, the function approximator is trained on a sequence of training sets with increasing difficulty, reusing the parameters from the previous optimization as the initialization for the next. This has been shown to avoid local optima of the objective function. However, the authors of [49] required the learning curriculum to be specified by hand. This requirement was avoided in [50] by introducing a multiplier that was used to force the function approximator to include more and more samples into the training set. The authors then showed that the algorithm tends to first include samples in the training set, for which it can predict the most accurate values. Indeed, the name for the approach introduced in our work is derived from the name of this algorithm, since the parameter α has similar behavior and interpretation as the multipler in [50]. The approach presented in [51] extends the work of [50] by allowing to take prior knowledge about "good" curricula into account. Our approach also allows to incorporate such prior knowledge through the specification of the initial context distribution $\tilde{\mu}_0$.

The authors of [52] investigated the much harder problem of learning the *optimal* curriculum by formulating the context selection problem as a Markov Decision Process (MDP) and learning an optimal policy in it. However, this turns out to be computationally harder than learning the entire task from scratch at once. Our method employs local auxiliary optimization instead as a surrogate for global optimization, which significantly improves sample efficiency of learning.

The process of adaptive task selection in SPRL can be seen as a form of active learning [53], which allows machine learning algorithms to explicitly choose the data from which to learn instead of using a pre-determined set of training samples. This has been shown to increase prediction accuracy while reducing the required number of samples. The results in the "global" setting of the gate environment demonstrate a similar beneficial property of our algorithm, where initially focusing on easy contexts led to a faster convergence to an optimal policy across all contexts compared to Contextual Relative Entropy Policy Search (C-REPS). Active learning in turn is closely related to curiosity-driven learning [54], which introduced such approaches as intelligent adaptive curiosity [55] and intrinsic motivation [56] that suggest focusing learning on tasks that promise high change in reward based on the recent history of experiences [57]. Curiosity-driven learning was combined with multi-armed bandit algorithms for automatic task selection in reinforcement learning problems [58] and was applied in robotics to learn goal-reaching movements with sparse rewards [59].

The idea of reusing knowledge across related tasks is at the heart of transfer learning in general [60] and transfer in RL in particular [7, 61]. Prioritization of tasks for which the agent obtains rewards falling into a certain interval of values combined with additional reversibility assumptions was shown to enable learning of high-dimensional object manipulation and maze navigation tasks [62, 63].

Assuming shared dynamics between tasks and knowledge about the functional form of the reward function allowed to solve a variety of tasks in the classical computer game Doom by learning to predict measurements for various time horizons [64]. These predicted measurements could then be combined in a linear way to define the current task that the agent should solve. Enhanced with universal value function approximators [65], such a reward-based transfer was extremely successful in robotics applications with sparse reward functions [66]. Later, this approach was argued in [67] to acutally learn a long-term dynamics prediction due to the specific from of the reward function and the fact that the agents learned goal reaching tasks. And indeed, if the assumptions of a shared environment and a known reward function hold, model-based reinforcement learning approaches such as [68, 32] seem to be a natural and appealing way to transfer knowledge between tasks.

Finally, an important theoretical relation to the presented algorithm is the already mentioned work on contextual MDPs [23, 24]. Especially the considerations on the smoothness of MDPs in [24] is a topic that is also important for our algorithm. Indeed, SPRL implicitly assumes a smooth change not only in the MDP structure but also in the optimal policy as we typically represent the policy using a linear re-weighting of features.

7 Discussion and Outlook

We have introduced homotopic predictor-corrector methods to motivate the potential of a predictor-corrector scheme in reinforcement learning domains to avoid problematic local optima of reward functions. We then introduced and investigated concepts from optimization and concurrent systems theory that can possibly be leveraged to implement such a scheme. This investigation, however, revealed that a straightforward application of these concepts to general reinforcement learning problems requires more theoretical work, since there are technical and practical aspects that prevent their general application. Because of these shortcomings, we then took an episodic view of the contextual reinforcement learning problem. This view allowed us to straightforwardly extend an existing policy search algorithm to allow a reinforcement learning agent to control the, otherwise hidden, context distribution. In the experimental section, we have demonstrated that this control over the context distribution is indeed utilized by the agent to focus on easier tasks in initial iterations. The agent can then be increasingly "forced" to match the desired context distribution by increasing the value of a penalty multiplier. This learning scheme allowed to solve obstacle avoidance tasks in different environments, in which a default contextual policy search algorithm could not achieve satisfying results, by reusing acquired knowledge from the initial algorithm iterations in easier tasks.

The results of this work leave many points for future work. First of all, addressing the encountered shortcomings of sensitivity analysis and bisimulation in chapter 3 may allow to create an algorithm similar to the one presented in chapter 4 that, however, relies on theoretical insights from optimization and process theory. This could drastically improve sample efficiency of the overall algorithm.

Extending the SPRL algorithm to step-based policy search that learns a policy $\pi(\mathbf{a}|\mathbf{s})$, which directly maps from states $\mathbf{s}$ in a Markov Decision Process (MDP) to actions $\mathbf{a}$, could be another way to improve the sample efficiency of the algorithm by removing the abstraction of episodic policy search, allowing to leverage more information about the structure of the underlying MDP.

Furthermore, enforcing the KL-Divergence $D_{\mathrm{KL}}(\tilde{\mu}|\mu)$ to be smaller than a certain threshold instead of penalizing it in optimization problem 4.11 could allow for a more fine-grained control over the evolution of the context distributions $\tilde{\mu}_i$. This is straightforward to implement, however needs to be evaluated in future experiments.

Another point for future work is to improve the inference of π_{i+1} and $\tilde{\mu}_{i+1}$ from the data set D_π and $D_{\tilde{\mu}}$. Currently, the employed weighted linear regression and maximum likelihood estimation do not guarantee that the the bound of the KL-Divergences $D_{\mathrm{KL}}(\pi_{i+1}\|\pi_i)$ and $D_{\mathrm{KL}}(\mu_{i+1}\|\mu_i)$ is satisfied. As mentioned at the end of chapter 4, we were also forced to use the sample weights that are - theoretically speaking - not completely accurate. Appendix A.6 shows a potential way of how to learn both the new policy π_{i+1} and μ_{i+1} only from the dataset D_π while respecting aforementioned KL bounds. Whether this work in practice, however, is left to be shown by additional experiments.

Currently, the parameter α needs to be chosen by the user in every iteration. While the experiments demonstrated that typically a simple step-function representing the values of α for the individual iterations is sufficient, finding ways to automatically determine the appropriate α would allow to apply our algorithm while requiring less fine-tuning. At the point of writing this thesis, we already obtained promising preliminary results by choosing α such that in every iteration, the penalty due to the KL-Divergence between current- and target context distribution makes up a certain percentage β of the current average reward. With this approach, β becomes the only open parameter.

Finally, the KL-Divergence is a non-geometric distance measure between probability distributions. While this may be a sensible choice for the distance between successive policies π_{i+1} and π_i, the context space C may indeed have a geometric interpretation that should be utilized by the SPRL algorithm. The so-called wasserstein metric [69] allows to incorporate a metric on the space C when computing the distance between two probability distributions. Inspired by the computation of the wasserstein metric in [70], appendix A.7 allows to use the wasserstein metric to penalize the distance between $\tilde{\mu}$ and μ instead of the KL-Divergence in optimization problem 4.11. Again, actual experiments need to be conducted to demonstrate that this approach works in a practical scenario.

Addressing aforementioned points will help to further increase the capabilities of the presented algorithm, allowing to even better leverage the benefits of knowledge transfer between related tasks - and with that taking further steps towards truly intelligent machines.

Bibliography

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[3] G. Tesauro, "Td-gammon: A self-teaching backgammon program," in *Applications of Neural Networks*, pp. 267–285, Springer, 1995.

[4] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems*, pp. 849–856, 2009.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[6] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*, pp. 363–372, Springer, 2006.

[7] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[8] L. B. Smith, S. Jayaraman, E. Clerkin, and C. Yu, "The developing infant creates a curriculum for statistical learning," *Trends in cognitive sciences*, 2018.

[9] T. Nikolayzik, *Korrekturverfahren zur numerischen Loesung nichtlinearer Optimierungsprobleme mittels Methoden der parametrischen Sensitivitaetsanalyse*. PhD thesis, Universitaet Bremen, 2012.

[10] E. N. Pistikopoulos, M. C. Georgiadis, and V. Dua, *Multi-parametric programming*, vol. 1. 2011.

[11] N. Ferns, P. Panangaden, and D. Precup, "Metrics for finite markov decision processes," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 162–169, AUAI Press, 2004.

[12] J. Taylor, D. Precup, and P. Panagaden, "Bounding performance loss in approximate mdp homomorphisms," in *Advances in Neural Information Processing Systems*, pp. 1649–1656, 2009.

[13] K. G. Larsen and A. Skou, "Bisimulation through probabilistic testing," *Information and computation*, vol. 94, no. 1, pp. 1–28, 1991.

[14] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-efficient generalization of robot skills with contextual policy search," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[15] C. Daniel, G. Neumann, and J. Peters, "Hierarchical relative entropy policy search," in *Artificial Intelligence and Statistics*, pp. 273–281, 2012.

[16] S. Parisi, H. Abdulsamad, A. Paraschos, C. Daniel, and J. Peters, "Reinforcement learning vs human programming in tetherball robot games," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6428–6434, IEEE, 2015.

[17] E. L. Allgower and K. Georg, *Numerical continuation methods: an introduction*, vol. 13. Springer Science & Business Media, 2012.

[18] T. J. Ypma, "Historical development of the newton–raphson method," *SIAM review*, vol. 37, no. 4, pp. 531–551, 1995.

[19] R. Penrose, "A generalized inverse for matrices," in *Mathematical proceedings of the Cambridge philosophical society*, vol. 51, pp. 406–413, Cambridge University Press, 1955.

[20] E. H. Moore, "On the reciprocal of the general algebraic matrix, abstract," *Bull. Amer. Math. Soc.*, vol. 26, pp. 394–395, 1920.

[21] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*, vol. 135. MIT press Cambridge, 1998.

[22] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[23] A. Hallak, D. Di Castro, and S. Mannor, "Contextual markov decision processes," *arXiv preprint arXiv:1502.02259*, 2015.

[24] A. Modi, N. Jiang, S. Singh, and A. Tewari, "Markov decision processes with continuous side information," in *Algorithmic Learning Theory*, pp. 597–618, 2018.

[25] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[26] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems*, vol. 12, no. 2, pp. 19–22, 1992.

[27] M. Morari, C. E. Garcia, and D. M. Prett, "Model predictive control: theory and practice," *IFAC Proceedings Volumes*, vol. 21, no. 4, pp. 1–12, 1988.

[28] J. Kadam and W. Marquardt, "Sensitivity-based solution updates in closed-loop dynamic optimization," in *Proceedings of the DYCOPS*, vol. 7, 2004.

[29] H. Pirnay, R. López-Negrete, and L. T. Biegler, "Optimal sensitivity based on ipopt," *Mathematical Programming Computation*, vol. 4, no. 4, pp. 307–331, 2012.

[30] V. M. Zavala, "Computational strategies for the optimal operation of large-scale chemical processes," 2008.

[31] S. Kamthe and M. P. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control," *arXiv preprint arXiv:1706.06491*, 2017.

[32] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Advances in Neural Information Processing Systems*, pp. 4759–4770, 2018.

[33] R. Givan, T. Dean, and M. Greig, "Equivalence notions and model minimization in markov decision processes," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 163–223, 2003.

[34] L. Kantorovich, "On the transfer of masses (in russian)," in *Doklady Akademii Nauk*, vol. 37, pp. 227–229, 1942.

[35] B. Ravindran and A. G. Barto, "Relativized options: Choosing the right transformation," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 608–615, 2003.

[36] P. S. Castro and D. Precup, "Using bisimulation for policy transfer in mdps," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 1399–1400, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[37] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[38] N. Ferns, P. Panangaden, and D. Precup, "Metrics for markov decision processes with infinite state spaces," *arXiv preprint arXiv:1207.1386*, 2012.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[40] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[41] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization.," in *Icml*, vol. 37, pp. 1889–1897, 2015.

[42] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[43] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016.

[44] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems*, pp. 2616–2624, 2013.

[45] B. F. Skinner, *The behavior of organisms: An experimental analysis*. BF Skinner Foundation, 1990.

[46] T. Erez and W. D. Smart, "What does shaping mean for computational reinforcement learning?," in *7th IEEE International Conference on Development and Learning (ICDL)*, 2008.

[47] J. Bongard and H. Lipson, "Once more unto the breach: Co-evolving a robot and its simulator," in *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pp. 57–62, 2004.

[48] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," *arXiv preprint arXiv:1810.05687*, 2018.

[49] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 41–48, ACM, 2009.

[50] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Advances in Neural Information Processing Systems*, pp. 1189–1197, 2010.

[51] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. G. Hauptmann, "Self-paced curriculum learning," in *AAAI*, vol. 2, p. 6, 2015.

[52] S. Narvekar and P. Stone, "Learning curriculum policies for reinforcement learning," *arXiv preprint arXiv:1812.00285*, 2018.

[53] B. Settles, "Active learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.

[54] P.-Y. Oudeyer, "Computational theories of curiosity-driven learning," *arXiv preprint arXiv:1802.10546*, 2018.

[55] P.-Y. Oudeyer, "Intelligent adaptive curiosity: A source of self-development," in *Proceedings of the 4th International Workshop on Epigenetic Robotics*, vol. 117, pp. 127–130, Lund University Cognitive Studies, 2004.

[56] A. G. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *Proceedings of the 3rd International Conference on Development and Learning*, pp. 112–19, 2004.

[57] A. Baranes and P.-Y. Oudeyer, "Intrinsically motivated goal exploration for active motor learning in robots: A case study," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, 2010.

[58] A. Fabisch and J. H. Metzen, "Active contextual policy search," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3371–3399, 2014.

[59] P. Fournier, O. Sigaud, M. Chetouani, and P.-Y. Oudeyer, "Accuracy-based curriculum learning in deep reinforcement learning," *arXiv preprint arXiv:1806.09614*, 2018.

[60] S. J. Pan, Q. Yang, *et al.*, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[61] A. Lazaric, "Transfer in reinforcement learning: a framework and a survey," in *Reinforcement Learning*, pp. 143–173, Springer, 2012.

[62] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *International Conference on Machine Learning*, pp. 1514–1523, 2018.

[63] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *Proceedings of the 1st Annual Conference on Robot Learning* (S. Levine, V. Vanhoucke, and K. Goldberg, eds.), vol. 78 of *Proceedings of Machine Learning Research*, pp. 482–495, PMLR, 13–15 Nov 2017.

[64] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," *arXiv preprint arXiv:1611.01779*, 2016.

[65] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International Conference on Machine Learning*, pp. 1312–1320, 2015.

[66] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

[67] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep rl for model-based control," in *International Conference on Learning Representations*, 2018.

[68] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

[69] C. Villani, *Optimal transport: old and new*, vol. 338. Springer Science & Business Media, 2008.

[70] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Advances in neural information processing systems*, pp. 2292–2300, 2013.

[71] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

A Appendix

A.1 Multi-Parametric Programming Conditions

Given the parametric optimization problem

$$\begin{aligned}\boldsymbol{\theta}^*(\mathbf{c}) = \arg\min_{\boldsymbol{\theta}}\ & f(\boldsymbol{\theta}, \mathbf{c}) \\ \text{s.t. } & g(\boldsymbol{\theta}, \mathbf{c}) \leq \mathbf{0} \\ & h(\boldsymbol{\theta}, \mathbf{c}) = \mathbf{0},\end{aligned}$$

with $\boldsymbol{\theta} \in \mathbb{R}^{n_{\boldsymbol{\theta}}}$, $\mathbf{c} \in \mathbb{R}^{n_{\mathbf{c}}}$, $g : \mathbb{R}^{n_{\boldsymbol{\theta}}} \times \mathbb{R}^{n_{\mathbf{c}}} \mapsto \mathbb{R}^{n_g}$ and $h : \mathbb{R}^{n_{\boldsymbol{\theta}}} \times \mathbb{R}^{n_{\mathbf{c}}} \mapsto \mathbb{R}^{n_h}$, a point $\boldsymbol{\theta}$ fulfills the so called Linear Independence Constraint Qualification (LICQ) for a given parameter $\mathbf{c}$, if the set of gradients

$$T_C(\boldsymbol{\theta}, \mathbf{c}) = \{\nabla_{\boldsymbol{\theta}} h_i(\boldsymbol{\theta}, \mathbf{c}) | i = 1, \ldots, n_h\} \cup \{\nabla_{\boldsymbol{\theta}} g_i(\boldsymbol{\theta}, \mathbf{c}) | i = 1, \ldots, n_g \wedge g_i(\boldsymbol{\theta}, \mathbf{c}) = 0\}$$

is linearly independent. We also refer to points $\boldsymbol{\theta}$ that satisfy the LICQ as normal points.

Now, as detailed in [9], if $\boldsymbol{\theta}$ is a normal point and there exist $\boldsymbol{\lambda} \in \mathbb{R}^{n_g}$ and $\boldsymbol{\mu} \in \mathbb{R}^{n_h}$ such that

$$\begin{aligned}\lambda_i &\geq 0 && \forall i \in [1, \ldots, n_g], \\ \lambda_i &= 0 && \forall i \in \{i | i = 1, \ldots, n_g \wedge g_i(\boldsymbol{\theta}, \mathbf{c}) < 0\}, \\ \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{c}) &= \mathbf{0}, \\ \mathbf{d}^T \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{c}) \mathbf{d} &> 0 && \forall \mathbf{d} \in \ker(T_C(\boldsymbol{\theta}, \mathbf{c})) \setminus \{\mathbf{0}\}\end{aligned}$$

hold, then $\boldsymbol{\theta}$ fulfills the so called strict second order optimality conditions. Above conditions assume that f, g and h are twice continuously differentiable in a region around $\boldsymbol{\theta}$.

A.2 Value Function Relations

Given two Markov Decision Processes (MDPs) $\text{MDP}_1 = (S, A, p, r_1)$ and $\text{MDP}_2 = (S, A, p, r_2)$ with

$$r_2(\mathbf{s}, \mathbf{a}) = \alpha r_1(\mathbf{s}, \mathbf{a}) + \beta,$$

we will show that

$$V_2^*(\mathbf{s}) = \alpha V_1^*(\mathbf{s}) + \frac{\beta}{1 - \gamma}.$$

For that, we will iterate the value iteration operator

$$V^{i+1}(\mathbf{s}) = F_V(V^i)(\mathbf{s}) = \max_{\mathbf{a}} \left(r(\mathbf{s}, \mathbf{a}) + \gamma E_p \left[V^i(\tilde{\mathbf{s}}) | \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \right] \right),$$

which is known to converge to the optimal value functions V_1^* and V_2^* from any initial guess V_0 in the limit $i \to \infty$. Due to simplicity, we will assume that $V_2^0(\mathbf{s}) = V_1^0(\mathbf{s}) = 0$ for all $\mathbf{s} \in S$. However, since operator F_V is known to converge from any initial V^0, the result naturally generalizes. With the aforementioned assumptions we obtain

$$\begin{aligned}V_2^1(\mathbf{s}) &= \max_{\mathbf{a}} \left(r_2(\mathbf{s}, \mathbf{a}) + \gamma E_p \left[V_2^0(\tilde{\mathbf{s}}) | \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \right] \right) \\ &= \max_{\mathbf{a}} \left(r_2(\mathbf{s}, \mathbf{a}) \right) \\ &= \max_{\mathbf{a}} \left(\alpha r_1(\mathbf{s}, \mathbf{a}) + \beta \right) \\ &= \alpha \max_{\mathbf{a}} \left(r_1(\mathbf{s}, \mathbf{a}) \right) + \beta \\ &= \alpha V_1^1(\mathbf{s}) + \beta.\end{aligned}$$

Now assuming that $V_2^i(\mathbf{s}) = \alpha V_1^i(\mathbf{s}) + \sum_{j=0}^{i-1} \gamma^i \beta$, we show that this still holds for $i+1$ by investigating $V_2^{i+1}(\mathbf{s}) = F_V(V_2^i)(\mathbf{s})$

$$
\begin{aligned}
V_2^{i+1}(\mathbf{s}) &= \max_{\mathbf{a}} \left(r_2(\mathbf{s}, \mathbf{a}) + \gamma E_p \left[V_2^i(\tilde{\mathbf{s}}) | \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \right] \right) \\
&= \max_{\mathbf{a}} \left(\alpha r_1(\mathbf{s}, \mathbf{a}) + \beta + \gamma E_p \left[\alpha V_1^i(\tilde{\mathbf{s}}) + \sum_{j=0}^{i-1} \gamma^i \beta \middle| \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \right] \right) \\
&= \alpha \max_{\mathbf{a}} \left(r_1(\mathbf{s}, \mathbf{a}) + \gamma E_p \left[V_1^i(\tilde{\mathbf{s}}) | \tilde{\mathbf{s}} \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \right] \right) + \beta + \gamma \sum_{j=0}^{i-1} \gamma^i \beta \\
&= \alpha V_1^{i+1}(\mathbf{s}) + \beta + \gamma \sum_{j=0}^{i-1} \gamma^i \beta \\
&= \alpha V_1^{i+1}(\mathbf{s}) + \beta + \sum_{j=0}^{i-1} \gamma^{i+1} \beta \\
&= \alpha V_1^{i+1}(\mathbf{s}) + \beta + \sum_{j=1}^{i} \gamma^i \beta \\
&= \alpha V_1^{i+1}(\mathbf{s}) + \sum_{j=0}^{i} \gamma^i \beta.
\end{aligned}
$$

Now taking the limit $i \to \infty$ gives the desired result, as it holds that

$$
\lim_{i \to \infty} \sum_{j=0}^{i} \gamma^i \beta = \frac{\beta}{1-\gamma}
$$

when $\gamma < 1$.

A.3 SPRL Derivation

The Lagrangian of optimization problem 4.7

$$
\begin{aligned}
L(p, \tilde{\mu}, \eta, \lambda, V) = &\int_{C,\Theta} r(\boldsymbol{\theta}, \mathbf{c}) p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{c} - \alpha \int_C \tilde{\mu}(\mathbf{c}) \log \left(\frac{\tilde{\mu}(\mathbf{c})}{\mu(\mathbf{c})} \right) \mathrm{d}\mathbf{c} \\
&+ \eta \left(\epsilon - \int_{C,\Theta} p(\boldsymbol{\theta}, \mathbf{c}) \log \left(\frac{p(\boldsymbol{\theta}, \mathbf{c})}{q(\boldsymbol{\theta}, \mathbf{c})} \right) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{c} \right) \\
&+ \lambda \left(1 - \int_{C,\Theta} p(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{c} \right) \\
&+ \int_C V(\mathbf{c}) \left(\tilde{\mu}(\mathbf{c}) - \int_\Theta p(\boldsymbol{\theta}, \mathbf{c}) d\boldsymbol{\theta} \right) \mathrm{d}\mathbf{c} \\
= &\eta\epsilon + \lambda + E_p \left[r(\boldsymbol{\theta}, \mathbf{c}) - \eta \log \left(\frac{p(\boldsymbol{\theta}, \mathbf{c})}{q(\boldsymbol{\theta}, \mathbf{c})} \right) - \lambda - V(\mathbf{c}) \right] + E_{\tilde{\mu}} \left[-\alpha \log \left(\frac{\tilde{\mu}(\mathbf{c})}{\mu(\mathbf{c})} \right) + V(\mathbf{c}) \right]
\end{aligned}
$$

allows to express the optimal joint- and context distribution p^* and μ^* in terms of the Lagrangian multipliers. For this we set the derivatives of L with respect to p and $\tilde{\mu}$ to zero

$$\begin{aligned}
\frac{\partial L}{\partial p} &= r(\boldsymbol{\theta}, \mathbf{c}) - \eta \left(1 + \log\left(\frac{p(\boldsymbol{\theta}, \mathbf{c})}{q(\boldsymbol{\theta}, \mathbf{c})}\right)\right) - \lambda - V(\mathbf{c}) = 0 \\
&\Leftrightarrow r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c}) - \lambda - \eta = \eta \log\left(\frac{p(\boldsymbol{\theta}, \mathbf{c})}{q(\boldsymbol{\theta}, \mathbf{c})}\right) \\
&\Leftrightarrow p^*(\boldsymbol{\theta}, \mathbf{c}) = q(\boldsymbol{\theta}, \mathbf{c}) \exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right) \exp\left(\frac{-\lambda - \eta}{\eta}\right), \\
\frac{\partial L}{\partial \mu} &= -\alpha \left(1 + \log\left(\frac{\tilde{\mu}(\mathbf{c})}{\mu(\mathbf{c})}\right)\right) + V(\mathbf{c}) = 0 \\
&\Leftrightarrow V(\mathbf{c}) - \alpha = \alpha \log\left(\frac{\tilde{\mu}(\mathbf{c})}{\mu(\mathbf{c})}\right) \\
&\Leftrightarrow \tilde{\mu}^*(\mathbf{c}) = \mu(\mathbf{c}) \exp\left(\frac{V(\mathbf{c})}{\alpha}\right) \exp(-1).
\end{aligned}$$

The dual of the optimization problem can be obtained by plugging the derived expressions back into the Lagrangian. Before we do that, we will first derive a result that will later helps us to eliminate the Lagrangian multiplier λ by plugging the derived expression for p into constraint 4.9

$$\begin{aligned}
&\int_{C,\Theta} p^*(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{c} = 1 \\
\Leftrightarrow &\int_{C,\Theta} q(\boldsymbol{\theta}, \mathbf{c}) \exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right) \exp\left(\frac{-\lambda - \eta}{\eta}\right) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{c} = 1 \\
\Leftrightarrow &E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right)\right] = \exp\left(\frac{\lambda + \eta}{\eta}\right) \\
\Leftrightarrow &\eta + \lambda = \eta \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right)\right]\right).
\end{aligned}$$

Now we are ready to derive the dual function

$$\begin{aligned}
\mathcal{G}(\eta, \lambda, V) =& \eta\epsilon + \lambda + E_p\left[r(\boldsymbol{\theta}, \mathbf{c}) - \eta \log\left(\frac{p(\boldsymbol{\theta}, \mathbf{c})}{q(\boldsymbol{\theta}, \mathbf{c})}\right) - \lambda - V(\mathbf{c})\right] + E_{\tilde{\mu}}\left[-\alpha \log\left(\frac{\tilde{\mu}(\mathbf{c})}{\mu(\mathbf{c})}\right) + V(\mathbf{c})\right] \\
=& \eta\epsilon + \lambda + E_p\left[r(\boldsymbol{\theta}, \mathbf{c}) - \eta \log\left(\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right) \exp\left(\frac{-\lambda - \eta}{\eta}\right)\right) - \lambda - V(\mathbf{c})\right] \\
&+ E_{\tilde{\mu}}\left[-\alpha \log\left(\exp\left(\frac{V(\mathbf{c})}{\alpha}\right) \exp(-1)\right) + V(\mathbf{c})\right] \\
=& \eta\epsilon + \lambda + E_p\left[r(\boldsymbol{\theta}, \mathbf{c}) - (r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c}) - \lambda - \eta) - \lambda - V(\mathbf{c})\right] + E_{\tilde{\mu}}\left[-(V(\mathbf{c}) - 1) + V(\mathbf{c})\right] \\
=& \eta\epsilon + \lambda + E_p\left[\eta\right] + E_{\tilde{\mu}}\left[1\right] \\
=& \eta\epsilon + \lambda + \eta E_p\left[1\right] + E_{\tilde{\mu}}\left[1\right].
\end{aligned}$$

Assuming that we choose the multiplier λ according to the multipliers η and V, such that it enforces 4.9, we know that $E_p\left[1\right] = 1$. With that we can simplify G to

$$\begin{aligned}
\mathcal{G}(\eta, V) &= \eta\epsilon + \lambda + \eta + E_{\tilde{\mu}}\left[1\right] \\
&= \eta\epsilon + \eta \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right)\right]\right) + E_{\tilde{\mu}}\left[1\right].
\end{aligned}$$

However, we cannot easily apply the same trick to the expectation over $\tilde{\mu}$, as although we know that constraint 4.9 holds, we can only be sure that $E_{\tilde{\mu}}\left[1\right] = 1$ if we would also choose the multiplier V to also enforce constraint 4.10. Consequently, we can only replace this expectation by the definition of $\tilde{\mu}$

$$\begin{aligned}
\mathcal{G}(\eta, V) &= \eta\epsilon + \eta \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right)\right]\right) + E_{\mu}\left[\exp\left(\frac{V(\mathbf{c})}{\alpha}\right) \exp(-1)\right] \\
&= \eta\epsilon + \eta \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta}\right)\right]\right) + \exp(-1) E_{\mu}\left[\exp\left(\frac{V(\mathbf{c})}{\alpha}\right)\right].
\end{aligned}$$

A.4 Stable SPRL Derivation

To derive the optimal joint- and context distribution p^* and $\tilde{\mu}^*$, we first define the Lagrangian of optimization problem 4.7

$$
\begin{aligned}
L\left(p, \tilde{\mu}, \eta_p, \eta_{\tilde{\mu}}, V, \lambda_p, \lambda_{\tilde{\mu}}\right) = & \int_{C,\Theta} p\left(\boldsymbol{\theta}, \mathbf{c}\right) r(\boldsymbol{\theta}, \mathbf{c}) \mathrm{d}\mathbf{c}\mathrm{d}\boldsymbol{\theta} - \alpha \int_C \tilde{\mu}\left(\mathbf{c}\right) \log\left(\frac{\tilde{\mu}\left(\mathbf{c}\right)}{\mu\left(\mathbf{c}\right)}\right) \mathrm{d}\mathbf{c} \\
& + \eta_p \left(\epsilon - \int_{C,\Theta} p\left(\boldsymbol{\theta}, \mathbf{c}\right) \log\left(\frac{p\left(\boldsymbol{\theta}, \mathbf{c}\right)}{q\left(\boldsymbol{\theta}, \mathbf{c}\right)}\right) \mathrm{d}\mathbf{c}\mathrm{d}\boldsymbol{\theta}\right) \\
& + \eta_{\tilde{\mu}} \left(\epsilon - \int_C \tilde{\mu}\left(\mathbf{c}\right) \log\left(\frac{\tilde{\mu}\left(\mathbf{c}\right)}{\tilde{\mu}_i\left(\mathbf{c}\right)}\right) \mathrm{d}\mathbf{c}\right) \\
& + \int_C V\left(\mathbf{c}\right) \left(\tilde{\mu}\left(\mathbf{c}\right) - \int_\Theta p\left(\boldsymbol{\theta}, \mathbf{c}\right) \mathrm{d}\boldsymbol{\theta}\right) \mathrm{d}\mathbf{c} \\
& + \lambda_p \left(1 - \int_{C,\Theta} p\left(\boldsymbol{\theta}, \mathbf{c}\right) \mathrm{d}\mathbf{c}\mathrm{d}\boldsymbol{\theta}\right) \\
& + \lambda_{\tilde{\mu}} \left(1 - \int_C \tilde{\mu}\left(\mathbf{c}\right) \mathrm{d}\mathbf{c}\right) \\
= & \eta_p \epsilon + \eta_{\tilde{\mu}} \epsilon + \lambda_p + \lambda_{\tilde{\mu}} \\
& + E_p \left[r(\boldsymbol{\theta}, \mathbf{c}) - \eta_p \log\left(\frac{p\left(\boldsymbol{\theta}, \mathbf{c}\right)}{q\left(\boldsymbol{\theta}, \mathbf{c}\right)}\right) - V\left(\mathbf{c}\right) - \lambda_p\right] \\
& + E_{\tilde{\mu}} \left[-\alpha \log\left(\frac{\tilde{\mu}\left(\mathbf{c}\right)}{\mu\left(\mathbf{c}\right)}\right) - \eta_{\tilde{\mu}} \log\left(\frac{\tilde{\mu}\left(\mathbf{c}\right)}{\tilde{\mu}_i\left(\mathbf{c}\right)}\right) + V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}}\right].
\end{aligned}
$$

We see that compared to the Lagrangian of optimization problem 4.7, the derived Lagrangian has multiple parameters η_p and $\eta_{\tilde{\mu}}$ instead of only one η. This reflects the additional KL-Bound constraint on $\tilde{\mu}$. The same holds for the parameters λ_p and $\lambda_{\tilde{\mu}}$. To derive p and $\tilde{\mu}$ as expression of the multipliers, we again take the derivative of the Lagrangian with respect to p and $\tilde{\mu}$ and set them to zero

$$
\begin{aligned}
\frac{\partial L}{\partial p} &= r(\boldsymbol{\theta}, \mathbf{c}) - \eta_p \left(\log\left(\frac{p\left(\boldsymbol{\theta}, \mathbf{c}\right)}{q\left(\boldsymbol{\theta}, \mathbf{c}\right)}\right) + 1\right) - V\left(\mathbf{c}\right) - \lambda_p = 0 \\
&\Leftrightarrow p\left(\boldsymbol{\theta}, \mathbf{c}\right) = q\left(\boldsymbol{\theta}, \mathbf{c}\right) \exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V\left(\mathbf{c}\right)}{\eta_p}\right) \exp\left(\frac{-\lambda_p - \eta_p}{\eta_p}\right), \\
\frac{\partial L}{\partial \mu} &= -\alpha \left(\log\left(\frac{\tilde{\mu}\left(\mathbf{c}\right)}{\mu\left(\mathbf{c}\right)}\right) + 1\right) - \eta_{\tilde{\mu}} \left(\log\left(\frac{\tilde{\mu}\left(\mathbf{c}\right)}{\tilde{\mu}_i\left(\mathbf{c}\right)}\right) + 1\right) + V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} = 0 \\
&\Leftrightarrow V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha = \alpha \log\left(\frac{\tilde{\mu}\left(\mathbf{c}\right)}{\mu\left(\mathbf{c}\right)}\right) + \eta_{\tilde{\mu}} \log\left(\frac{\tilde{\mu}\left(\mathbf{c}\right)}{\mu_i\left(\mathbf{c}\right)}\right) \\
&\Leftrightarrow V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha = \alpha \log\left(\tilde{\mu}\left(\mathbf{c}\right)\right) - \alpha \log\left(\mu\left(\mathbf{c}\right)\right) + \eta_{\tilde{\mu}} \log\left(\tilde{\mu}\left(\mathbf{c}\right)\right) - \eta_{\tilde{\mu}} \log\left(\tilde{\mu}_i\left(\mathbf{c}\right)\right) \\
&\Leftrightarrow \alpha \log\left(\mu\left(\mathbf{c}\right)\right) + \eta_{\tilde{\mu}} \log\left(\tilde{\mu}_i\left(\mathbf{c}\right)\right) + V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha = \left(\alpha + \eta_{\tilde{\mu}}\right) \log\left(\tilde{\mu}\left(\mathbf{c}\right)\right).
\end{aligned}
$$

We want to emphasize above relation between $\tilde{\mu}$, μ and $\tilde{\mu}_i$ as we will need it later. A more intuitive way of expressing $\tilde{\mu}$ requires some more reformulations

$$
\begin{aligned}
\log\left(\tilde{\mu}\left(\mathbf{c}\right)\right) &= \frac{\alpha \log\left(\mu\left(\mathbf{c}\right)\right) + \eta_{\tilde{\mu}} log\left(\tilde{\mu}_i\left(\mathbf{c}\right)\right) + V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha}{\alpha + \eta_{\tilde{\mu}}} \\
\Leftrightarrow \log\left(\tilde{\mu}\left(\mathbf{c}\right)\right) &= \log\left(\mu\left(\mathbf{c}\right)^{\frac{\alpha}{\alpha+\eta_{\tilde{\mu}}}} \tilde{\mu}_i\left(\mathbf{c}\right)^{\frac{\eta_{\tilde{\mu}}}{\alpha+\eta_{\tilde{\mu}}}}\right) \frac{V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha}{\alpha + \eta_{\tilde{\mu}}} \\
\Leftrightarrow \tilde{\mu}\left(\mathbf{c}\right) &= \mu\left(\mathbf{c}\right)^{\frac{\alpha}{\alpha+\eta_{\tilde{\mu}}}} \tilde{\mu}_i\left(\mathbf{c}\right)^{\frac{\eta_{\tilde{\mu}}}{\alpha+\eta_{\tilde{\mu}}}} \exp\left(\frac{V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha}{\alpha + \eta_{\tilde{\mu}}}\right) \\
\Leftrightarrow \tilde{\mu}\left(\mathbf{c}\right) &= \mu\left(\mathbf{c}\right)^{\frac{\alpha}{\alpha+\eta_{\tilde{\mu}}}} \tilde{\mu}_i\left(\mathbf{c}\right)^{\frac{\eta_{\tilde{\mu}}}{\alpha+\eta_{\tilde{\mu}}}} \exp\left(\frac{V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha}{\alpha + \eta_{\tilde{\mu}}}\right) \\
\Leftrightarrow \tilde{\mu}\left(\mathbf{c}\right) &= \mu\left(\mathbf{c}\right)^{\frac{\alpha}{\alpha+\eta_{\tilde{\mu}}}} \tilde{\mu}_i\left(\mathbf{c}\right)^{\frac{\eta_{\tilde{\mu}}}{\alpha+\eta_{\tilde{\mu}}}-1} \tilde{\mu}_i\left(\mathbf{c}\right) \exp\left(\frac{V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha}{\alpha + \eta_{\tilde{\mu}}}\right) \\
\Leftrightarrow \tilde{\mu}\left(\mathbf{c}\right) &= \tilde{\mu}_i\left(\mathbf{c}\right) \exp\left(\frac{\alpha}{\alpha + \eta_{\tilde{\mu}}} \log\left(\mu\left(\mathbf{c}\right)\right) + \left(\frac{\eta_{\tilde{\mu}}}{\alpha + \eta_{\tilde{\mu}}} - 1\right) \log\left(\tilde{\mu}_i\left(\mathbf{c}\right)\right) + \frac{V\left(\mathbf{c}\right) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha}{\alpha + \eta_{\tilde{\mu}}}\right) \\
\Leftrightarrow \tilde{\mu}\left(\mathbf{c}\right) &= \tilde{\mu}_i\left(\mathbf{c}\right) \exp\left(\frac{\alpha \log\left(\mu\left(\mathbf{c}\right)\right) - \alpha \log\left(\tilde{\mu}_i\left(\mathbf{c}\right)\right) + V\left(\mathbf{c}\right)}{\alpha + \eta_{\tilde{\mu}}}\right) \exp\left(\frac{-\lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha}{\alpha + \eta_{\tilde{\mu}}}\right).
\end{aligned}
$$

Following the derivation in appendix A.3, we can again show that

$$\eta_p + \lambda_p = \eta_p \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta_p}\right)\right]\right).$$

However, since we introduced constraint 4.13 in optimization problem 4.11, we are now also able to show

$$\begin{aligned}
&\int \tilde{\mu}(\mathbf{c})\,\mathrm{d}\mathbf{c} = 1 \\
\Leftrightarrow &\int \tilde{\mu}_i(\mathbf{c}) \exp\left(\frac{\alpha\log(\mu(\mathbf{c})) - \alpha\log(\tilde{\mu}_i(\mathbf{c})) + V(\mathbf{c}) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha}{\alpha + \eta_{\tilde{\mu}}}\right)\mathrm{d}\mathbf{c} = 1 \\
\Leftrightarrow &\int \tilde{\mu}_i(\mathbf{c}) \exp\left(\frac{\alpha\log(\mu(\mathbf{c})) - \alpha\log(\tilde{\mu}_i(\mathbf{c})) + V(\mathbf{c})}{\alpha + \eta_{\tilde{\mu}}}\right)\mathrm{d}\mathbf{c} = \exp\left(\frac{\lambda_{\tilde{\mu}} + \eta_{\tilde{\mu}} + \alpha}{\alpha + \eta_{\tilde{\mu}}}\right) \\
\Leftrightarrow &E_{\tilde{\mu}_i}\left[\exp\left(\frac{\alpha\log(\mu(\mathbf{c})) - \alpha\log(\tilde{\mu}_i(\mathbf{c})) + V(\mathbf{c})}{\alpha + \eta_{\tilde{\mu}}}\right)\right] = \exp\left(\frac{\lambda_{\tilde{\mu}} + \eta_{\tilde{\mu}} + \alpha}{\alpha + \eta_{\tilde{\mu}}}\right) \\
\Leftrightarrow &\lambda_{\tilde{\mu}} + \eta_{\tilde{\mu}} + \alpha = (\alpha + \eta_{\tilde{\mu}}) \log\left(E_{\tilde{\mu}_i}\left[\exp\left(\frac{\alpha\log(\mu(\mathbf{c})) - \alpha\log(\tilde{\mu}_i(\mathbf{c})) + V(\mathbf{c})}{\alpha + \eta_{\tilde{\mu}}}\right)\right]\right).
\end{aligned}$$

Again we can make use of these properties when deriving the dual function by plugging in the expressions for p and $\tilde{\mu}$ into the Lagrangian

$$\begin{aligned}
\mathcal{G}(\eta_p, \eta_{\tilde{\mu}}, V) &= \eta_p\epsilon + \eta_{\tilde{\mu}}\epsilon + \lambda_p + \lambda_{\tilde{\mu}} \\
&\quad + E_p\left[r(\boldsymbol{\theta}, \mathbf{c}) - \eta_p \log\left(\frac{p(\boldsymbol{\theta}, \mathbf{c})}{q(\boldsymbol{\theta}, \mathbf{c})}\right) - V(\mathbf{c}) - \lambda_p\right] \\
&\quad + E_{\tilde{\mu}}\left[-\alpha\log\left(\frac{\tilde{\mu}(\mathbf{c})}{\mu(\mathbf{c})}\right) - \eta_{\tilde{\mu}}\log\left(\frac{\tilde{\mu}(\mathbf{c})}{\tilde{\mu}_i(\mathbf{c})}\right) + V(\mathbf{c}) - \lambda_{\tilde{\mu}}\right] \\
&= \eta_p\epsilon + \eta_{\tilde{\mu}}\epsilon + \lambda_p + \lambda_{\tilde{\mu}} \\
&\quad + E_p\left[r(\boldsymbol{\theta}, \mathbf{c}) - \eta_p \log\left(\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta_p}\right)\exp\left(\frac{-\lambda_p - \eta_p}{\eta_p}\right)\right) - V(\mathbf{c}) - \lambda_p\right] \\
&\quad + E_{\tilde{\mu}}\left[-(\alpha + \eta_{\tilde{\mu}})\log(\tilde{\mu}(\mathbf{c})) + \alpha\log(\mu(\mathbf{c})) + \eta_{\tilde{\mu}}\log(\tilde{\mu}_i(\mathbf{c})) + V(\mathbf{c}) - \lambda_{\tilde{\mu}}\right] \\
&= \eta_p\epsilon + \lambda_p + \eta_{\tilde{\mu}}\epsilon + \lambda_{\tilde{\mu}} + E_p[\eta_p] \\
&\quad + E_{\tilde{\mu}}\big[-(\alpha\log(\mu(\mathbf{c})) + \eta_{\tilde{\mu}}\log(\tilde{\mu}_i(\mathbf{c})) + V(\mathbf{c}) - \lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}} - \alpha) \\
&\qquad\qquad + \alpha\log(\mu(\mathbf{c})) + \eta_{\tilde{\mu}}\log(\tilde{\mu}_i(\mathbf{c})) + V(\mathbf{c}) - \lambda_{\tilde{\mu}}\big] \\
&= \eta_p\epsilon + \lambda_p + \eta_p + \eta_{\tilde{\mu}}\epsilon + \lambda_{\tilde{\mu}} + E_{\tilde{\mu}}[\eta_{\tilde{\mu}} + \alpha] \\
&= \eta_p\epsilon + \lambda_p + \eta_p + \eta_{\tilde{\mu}}\epsilon + \lambda_{\tilde{\mu}} + \eta_{\tilde{\mu}} + \alpha \\
&= \eta_p\epsilon + \eta_p \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta_p}\right)\right]\right) \\
&\quad + \eta_{\tilde{\mu}}\epsilon + (\alpha + \eta_{\tilde{\mu}})\log\left(E_{\tilde{\mu}_i}\left[\exp\left(\frac{\alpha\log(\mu(\mathbf{c})) - \alpha\log(\tilde{\mu}_i(\mathbf{c})) + V(\mathbf{c})}{\alpha + \eta_{\tilde{\mu}}}\right)\right]\right).
\end{aligned}$$

Note that we used the same "trick" as in appendix A.3 and assumed that the parameters λ_p and $\lambda_{\tilde{\mu}}$ are chosen such that $E_p[1] = 1$ and $E_{\tilde{\mu}}[1] = 1$.

A.5 Alpha Schedules

The experiments in section 5 were run with the alpha schedules shown in tables A.1.

A.6 Improved Distribution Updates

We are given a dataset of N weighted samples $D = \left\{(w_i, \mathbf{x}_i, \mathbf{y}_i) | i = 1, \ldots, N, \mathbf{x}_i \in \mathbb{R}^{d_\mathbf{x}}, \mathbf{y}_i \in \mathbb{R}^{d_\mathbf{y}}\right\}$ and we would like to fit a joint-distribution

$$p(\mathbf{x}, \mathbf{y}) = p_\mathbf{y}(\mathbf{y}|\mathbf{x})p_\mathbf{x}(\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\phi(\mathbf{x}), \boldsymbol{\Sigma}_\mathbf{y})\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\Sigma}_\mathbf{x})$$

Table A.1.: Schedules for α in the two gate environment settings (left) and the two reacher environments (right), showing at which iterations (left column) which value of α was set. An '$\times$' symbolizes that SPRL was not run for the corresponding number of iterations in this setting.

Iteration	Gate - Precision	Gate - Global
0	0	0
80	1e-6	0
120	1e-6	0.05
150	0.1	0.05
200	2.5	2.5
300	2.5	50
350	$\times$	500
400	$\times$	1e4
450	$\times$	1e5
500	$\times$	1e6

Iteration	Reacher	Modified Reacher
0	1e-5	0
30	1e-3	0
70	1e-2	0
100	1e-1	5e-13
140	$\times$	8e-13
180	$\times$	1e-12
210	$\times$	2e-12
240	$\times$	3e-12
260	$\times$	7e-12
300	$\times$	1e-10

to this dataset, while limiting the change with regards to a reference distribution

$$q(\mathbf{x}, \mathbf{y}) = q_{\mathbf{y}}(\mathbf{y}|\mathbf{x})q_{\mathbf{x}}(\mathbf{x}) = \mathcal{N}(\mathbf{y}|\tilde{\mathbf{A}}\phi(\mathbf{x}), \tilde{\mathbf{\Sigma}}_{\mathbf{y}})\mathcal{N}(\mathbf{x}|\tilde{\boldsymbol{\mu}}_{\mathbf{x}}, \tilde{\mathbf{\Sigma}}_{\mathbf{x}}),$$

where the feature function $\phi : X \mapsto \mathbb{R}^o$ can be arbitrary. We can express this as the following constrained optimization problem

$$\begin{aligned}
&\max_{\mathbf{A}, \mathbf{\Sigma}_{\mathbf{y}}, \boldsymbol{\mu}_{\mathbf{x}}, \mathbf{\Sigma}_{\mathbf{x}}} \sum_{i=1}^{N} w_i \log(p(\mathbf{x}_i, \mathbf{y}_i)) \\
&\qquad \text{s.t. } D_{\mathrm{KL}}(q\|p) \leq \epsilon \\
=\ &\max_{\mathbf{A}, \mathbf{\Sigma}_{\mathbf{y}}, \boldsymbol{\mu}_{\mathbf{x}}, \mathbf{\Sigma}_{\mathbf{x}}} \sum_{i=1}^{N} w_i \log(p(\mathbf{x}_i, \mathbf{y}_i)) \\
&\qquad \text{s.t. } E_{q_{\mathbf{x}}}\left[D_{\mathrm{KL}}(q_{\mathbf{y}}\|p_{\mathbf{y}})\right] + D_{\mathrm{KL}}(q_{\mathbf{x}}\|p_{\mathbf{x}}) \leq \epsilon \\
\approx\ &\max_{\mathbf{A}, \mathbf{\Sigma}_{\mathbf{y}}, \boldsymbol{\mu}_{\mathbf{x}}, \mathbf{\Sigma}_{\mathbf{x}}} \sum_{i=1}^{N} w_i \log(p(\mathbf{x}_i, \mathbf{y}_i)) \\
&\qquad \text{s.t. } \frac{1}{N}\sum_{i=1}^{N} D_{\mathrm{KL}}(q_{\mathbf{y}}(\cdot|\mathbf{x}_i)\|p_{\mathbf{y}}(\cdot|\mathbf{x}_i)) + D_{\mathrm{KL}}(q_{\mathbf{x}}\|p_{\mathbf{x}}) \leq \epsilon \\
=\ &\max_{\mathbf{A}, \mathbf{\Sigma}_{\mathbf{y}}, \boldsymbol{\mu}_{\mathbf{x}}, \mathbf{\Sigma}_{\mathbf{x}}} \sum_{i=1}^{N} w_i \Big(-\frac{1}{2}\left(\log(\det(\mathbf{\Sigma}_{\mathbf{y}})) + (\mathbf{y}_i - \mathbf{A}\phi(\mathbf{x}_i))^T \mathbf{\Sigma}_{\mathbf{y}}^{-1}(\mathbf{y}_i - \mathbf{A}\phi(\mathbf{x}_i))\right) \\
&\qquad\qquad -\frac{1}{2}\left(\log(\det(\mathbf{\Sigma}_{\mathbf{x}})) + (\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}})^T \mathbf{\Sigma}_{\mathbf{x}}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}})\right)\Big) \\
&\qquad \text{s.t. } \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}\left(\mathrm{tr}(\mathbf{\Sigma}_{\mathbf{y}}^{-1}\tilde{\mathbf{\Sigma}}_{\mathbf{y}}) + (\mathbf{A}\phi(\mathbf{x}_i) - \tilde{\mathbf{A}}\phi(\mathbf{x}_i))^T \mathbf{\Sigma}_{\mathbf{y}}^{-1}(\mathbf{A}\phi(\mathbf{x}_i) - \tilde{\mathbf{A}}\phi(\mathbf{x}_i)) + \log(\det(\mathbf{\Sigma}_{\mathbf{y}})) - \log(\det(\tilde{\mathbf{\Sigma}}_{\mathbf{y}})) - d_{\mathbf{y}}\right) \\
&\qquad\qquad + \frac{1}{2}\left(\mathrm{tr}(\mathbf{\Sigma}_{\mathbf{x}}^{-1}\tilde{\mathbf{\Sigma}}_{\mathbf{x}}) + (\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}})^T \mathbf{\Sigma}_{\mathbf{x}}^{-1}(\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}}) + \log(\det(\mathbf{\Sigma}_{\mathbf{x}})) - \log(\det(\tilde{\mathbf{\Sigma}}_{\mathbf{x}})) - d_{\mathbf{x}}\right) \leq \epsilon.
\end{aligned}$$

Note that we have already omitted constant terms in the Gaussian log likelihood, as they do not change the objective. In order to solve this problem, we need to set the derivative of the Lagrangian

$$L(\mathbf{A}, \mathbf{\Sigma}_{\mathbf{y}}, \boldsymbol{\mu}_{\mathbf{x}}, \mathbf{\Sigma}_{\mathbf{x}}) = \sum_{i=1}^{N} w_i \log(p_{\mathbf{y}}(\mathbf{y}_i|\mathbf{x}_i)p_{\mathbf{x}}(\mathbf{x}_i)) + \eta\left(\epsilon - \left(\sum_{i=1}^{N} D_{\mathrm{KL}}(q_{\mathbf{y}}(\cdot|\mathbf{x}_i)\|p_{\mathbf{y}}(\cdot|\mathbf{x}_i)) + D_{\mathrm{KL}}(q_{\mathbf{x}}\|p_{\mathbf{x}})\right)\right)$$

with respect to the optimization variables to zero and solve for them to obtain expressions of the optimization variables in terms of the multiplier η and the samples from D

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{A}} =& -\frac{1}{2}\sum_{i=1}^{N} w_i(-2)\mathbf{\Sigma}_{\mathbf{y}}^{-1}(\mathbf{y}_i - \mathbf{A}\phi(\mathbf{x}_i))\phi(\mathbf{x}_i)^T - \eta\frac{1}{2}\frac{1}{N}\sum_{i=1}^{N} 2\mathbf{\Sigma}_{\mathbf{y}}^{-1}(\mathbf{A}-\tilde{\mathbf{A}})\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T = \mathbf{0} \\
\Leftrightarrow \mathbf{A} =& \left[\sum_{i=1}^{N}\left(w_i\mathbf{y}_i + \frac{\eta}{N}\tilde{\mathbf{A}}\phi(\mathbf{x}_i)\right)\phi(\mathbf{x}_i)^T\right]\left[\sum_{i=1}^{N}\left(w_i + \frac{\eta}{N}\right)\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\right]^{-1},
\end{aligned}$$

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{\Sigma}_{\mathbf{y}}} =& -\frac{1}{2}\sum_{i=1}^{N} w_i\left(\mathbf{\Sigma}_{\mathbf{y}}^{-1} - \mathbf{\Sigma}_{\mathbf{y}}^{-1}(\mathbf{y}_i - \mathbf{A}\phi(\mathbf{x}_i))(\mathbf{y}_i - \mathbf{A}\phi(\mathbf{x}_i))^T\mathbf{\Sigma}_{\mathbf{y}}^{-1}\right) \\
& - \eta\frac{1}{2}\left(-\mathbf{\Sigma}_{\mathbf{y}}^{-1}\tilde{\mathbf{\Sigma}}_{\mathbf{y}}\mathbf{\Sigma}_{\mathbf{y}}^{-1} + \mathbf{\Sigma}_{\mathbf{y}}^{-1} - \frac{1}{N}\sum_{i=1}^{N}\mathbf{\Sigma}_{\mathbf{y}}^{-1}(\mathbf{A}-\tilde{\mathbf{A}})\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T(\mathbf{A}-\tilde{\mathbf{A}})^T\mathbf{\Sigma}_{\mathbf{y}}^{-1}\right) = \mathbf{0} \\
\Leftrightarrow \mathbf{\Sigma}_{\mathbf{y}} =& \frac{\sum_{i=1}^{N} w_i(\mathbf{y}_i - \mathbf{A}\phi(\mathbf{x}_i))(\mathbf{y}_i - \mathbf{A}\phi(\mathbf{x}_i))^T + \eta\left(\tilde{\mathbf{\Sigma}}_{\mathbf{y}} + \frac{1}{N}(\mathbf{A}-\tilde{\mathbf{A}})\sum_{i=1}^{N}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T(\mathbf{A}-\tilde{\mathbf{A}})^T\right)}{\sum_{i=1}^{N} w_i + \eta},
\end{aligned}$$

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\mu}_{\mathbf{x}}} =& -\frac{1}{2}\sum_{i=1}^{N} w_i(-2)\mathbf{\Sigma}_{\mathbf{x}}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}}) - \eta\frac{1}{2}2\mathbf{\Sigma}_{\mathbf{x}}^{-1}(\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}}) = \mathbf{0} \\
\Leftrightarrow \boldsymbol{\mu}_{\mathbf{x}} =& \frac{\sum_{i=1}^{N} w_i\mathbf{x}_i + \eta\tilde{\boldsymbol{\mu}}_{\mathbf{x}}}{\sum_{i=1}^{N} w_i + \eta},
\end{aligned}$$

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{\Sigma}_{\mathbf{x}}} =& -\frac{1}{2}\sum_{i=1}^{N} w_i\left(\mathbf{\Sigma}_{\mathbf{x}}^{-1} - \mathbf{\Sigma}_{\mathbf{x}}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}}))(\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}})^T\mathbf{\Sigma}_{\mathbf{x}}^{-1}\right) \\
& - \eta\frac{1}{2}\left(-\mathbf{\Sigma}_{\mathbf{x}}^{-1}\tilde{\mathbf{\Sigma}}_{\mathbf{x}}\mathbf{\Sigma}_{\mathbf{x}}^{-1} + \mathbf{\Sigma}_{\mathbf{x}}^{-1} - \mathbf{\Sigma}_{\mathbf{x}}^{-1}(\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}})(\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}})^T\mathbf{\Sigma}_{\mathbf{x}}^{-1}\right) = \mathbf{0} \\
\Leftrightarrow \mathbf{\Sigma}_{\mathbf{x}} =& \frac{\sum_{i=1}^{N} w_i(\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}})(\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}})^T + \eta\left(\tilde{\mathbf{\Sigma}}_{\mathbf{x}} + (\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}})(\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}})^T\right)}{\sum_{i=1}^{N} w_i + \eta}.
\end{aligned}$$

With above equations, we can easily find the optimal variables $\mathbf{A}$, $\mathbf{\Sigma}_{\mathbf{y}}$, $\boldsymbol{\mu}_{\mathbf{x}}$ and $\mathbf{\Sigma}_{\mathbf{x}}$ by plugging in above equations into the Lagrangian and then minimize the resulting dual with respect to η. This yields the optimal η^* which can then be used to compute the optimal variables. Note that we obtain the default maximum likelihood estimates for $\eta = 0$. This seems natural, since η only becomes zero if the constraint on the allowed KL-Divergence is not active. In this case, the maximum likelihood estimate does not need any regularization as it is close enough to the reference distribution.

A.7 Approximate Wasserstein Metric for SPRL

The wasserstein metric between the next and desired context distribution $\tilde{\mu}$ and μ under a metric $d : C \times C \mapsto \mathbb{R}$ on the context space can be expressed in terms of the following optimization problem

$$\begin{aligned}
D_{\mathrm{W}}(\tilde{\mu}, \mu) = \min_{\gamma} & \int_{C,C} \gamma(\mathbf{c}_1, \mathbf{c}_2) d(\mathbf{c}_1, \mathbf{c}_2) \mathrm{d}\mathbf{c}_1 \mathrm{d}\mathbf{c}_2 \\
\text{s.t.} & \int_{C} \gamma(\mathbf{c}_1, \mathbf{c}_2) \mathrm{d}\mathbf{c}_2 = \tilde{\mu}(\mathbf{c}_1) \; \forall \mathbf{c}_1 \in C \\
& \int_{C} \gamma(\mathbf{c}_1, \mathbf{c}_2) \mathrm{d}\mathbf{c}_1 = \mu(\mathbf{c}_2) \; \forall \mathbf{c}_2 \in C \\
& \gamma(\mathbf{c}_1, \mathbf{c}_2) \geq 0 \; \forall \mathbf{c}_1, \mathbf{c}_2 \in C.
\end{aligned}$$

While it is conceptually straightforward to incorporate this metric into optimization objective 4.11 by replacing the term $D_{\mathrm{KL}}(\tilde{\mu}\|\mu)$ with the term $\min_{\gamma} \int_{C,C} \gamma(\mathbf{c}_1, \mathbf{c}_2) d(\mathbf{c}_1, \mathbf{c}_2) \mathrm{d}\mathbf{c}_1 \mathrm{d}\mathbf{c}_2$, adding the constraints of above optimization problem to the constraints of 4.11 and then also optimizing over γ, this poses several technical difficulties. These problems are

similar to the problems that we faced in the naive definition of the Self-Paced Contextual Reinforcement Learning (SPRL) objective 4.7. However, we can just as before alleviate these problems by defining the optimization problem in the following way

$$
\begin{aligned}
&\max_{\tilde{\mu},\gamma,p} \left(\int_{C,\Theta} p(\boldsymbol{\theta},\mathbf{c}) r(\boldsymbol{\theta},\mathbf{c}) \mathrm{d}\mathbf{c}\mathrm{d}\boldsymbol{\theta} - \alpha \int_{C,C} \gamma(\mathbf{c}_1,\mathbf{c}_2) d(\mathbf{c}_1,\mathbf{c}_2) \mathrm{d}\mathbf{c}_1 \mathrm{d}\mathbf{c}_2 \right) \\
&\text{s.t. } D_{\mathrm{KL}}(p||q) \leq \epsilon \\
&\quad D_{\mathrm{KL}}(\tilde{\mu}||\tilde{\mu}_i) \leq \epsilon \\
&\quad \int_{C,\Theta} p(\boldsymbol{\theta},\mathbf{c}) \mathrm{d}\mathbf{c}\mathrm{d}\boldsymbol{\theta} = 1 \\
&\quad \int_{C} \tilde{\mu}(\mathbf{c}) \mathrm{d}\mathbf{c} = 1 \\
&\quad \int_{C,\Theta} p(\boldsymbol{\theta},\mathbf{c}) \mathrm{d}\boldsymbol{\theta} = \tilde{\mu}(\mathbf{c}) \; \forall \mathbf{c} \in C \\
&\quad D_{\mathrm{KL}}(\gamma||\tilde{\mu}_i \mu) \leq \epsilon_\gamma \\
&\quad \int_{C,C} \gamma(\mathbf{c}_1,\mathbf{c}_2) \mathrm{d}\mathbf{c}_1 \mathrm{d}\mathbf{c}_2 = 1 \\
&\quad \int_{C} \gamma(\mathbf{c}_1,\mathbf{c}_2) \mathrm{d}\mathbf{c}_2 = \tilde{\mu}(\mathbf{c}_1) \forall \mathbf{c}_1 \in C \\
&\quad \int_{C} \gamma(\mathbf{c}_1,\mathbf{c}_2) \mathrm{d}\mathbf{c}_1 = \mu(\mathbf{c}_2) \forall \mathbf{c}_2 \in C.
\end{aligned}
$$

The last two constraints are directly taken from the definition of the wasserstein metric. While those two constraints implicitly enforce that the total probability mass of γ indeed sums to one, we nonetheless enforce this explicitly by the third to last constraint. This yields, just as for the SPRL algorithm, easier to handle formulas for the dual. However, we avoided the constraint that explicitly enforces $\gamma(\mathbf{c}_1,\mathbf{c}_2) \geq 0$ by introducing a surrogate constraint $D_{\mathrm{KL}}(\gamma \| \tilde{\mu}\mu) \leq \epsilon_\gamma$. We already know that the definition of the KL-Divergence implicitly prevents negative values of γ. Furthermore, it will allow us to express γ in terms of the "reference" joint distribution $\tilde{\mu}(\mathbf{c}_1)\mu(\mathbf{c}_2)$. This is desirable because we can sample from both these distributions and hence can approximate expectations with regards to them. The KL-Bound ϵ_γ can be set arbitrarily large to allow γ to significantly deviate from the reference distribution, while allowing to express γ in terms of an exponential re-weighting of $\tilde{\mu}\mu$. This beneficial property of defining γ in terms of aforementioned reference distribution was used in [70] to significantly speed up the computation of the wasserstein metric. In our case this allows us to derive the following expressions for p, $\tilde{\mu}$ and γ in terms of the multipliers of the optimization problem

$$
\begin{aligned}
p(\boldsymbol{\theta},\mathbf{c}) &= q(\boldsymbol{\theta},\mathbf{c}) \exp\left(\frac{r(\boldsymbol{\theta},\mathbf{c}) - V(\mathbf{c})}{\eta_p}\right) \exp\left(\frac{-\lambda_p - \eta_p}{\eta_p}\right), \\
\gamma(\mathbf{c}_1,\mathbf{c}_2) &= \tilde{\mu}_i(\mathbf{c}_1)\mu(\mathbf{c}_2) \exp\left(\frac{-\alpha d(\mathbf{c}_1,\mathbf{c}_2) - \tilde{\Gamma}(\mathbf{c}_1) - \Gamma(\mathbf{c}_2)}{\eta_\gamma}\right) \exp\left(\frac{-\lambda_\gamma - \eta_\gamma}{\eta_\gamma}\right), \\
\tilde{\mu}(\mathbf{c}) &= \tilde{\mu}_i(\mathbf{c}) \exp\left(\frac{V(\mathbf{c}) + \tilde{\Gamma}(\mathbf{c})}{\eta_{\tilde{\mu}}}\right) \exp\left(\frac{-\lambda_{\tilde{\mu}} - \eta_{\tilde{\mu}}}{\eta_{\tilde{\mu}}}\right).
\end{aligned}
$$

We skipped the derivation of these results here, as it is analogous to the ones in appendices A.3 and A.4. The multipliers $\tilde{\Gamma}(\mathbf{c}_1)$ and $\Gamma(\mathbf{c}_2)$ correspond to second to last constraints, while η_γ and λ_γ correspond to the constraints enforcing the KL-Bound and a total probability mass of one. The dual of this optimization problem is given by

$$
\begin{aligned}
\mathcal{G}(\eta_p,\eta_{\tilde{\mu}},\eta_\gamma,V,\Gamma,\tilde{\Gamma}) =& \eta_p\epsilon + \eta_{\tilde{\mu}}\epsilon + \eta_\gamma\epsilon_\gamma \\
&+ \eta_p \log\left(E_q\left[\exp\left(\frac{r(\boldsymbol{\theta},\mathbf{c}) - V(\mathbf{c})}{\eta_p}\right)\right]\right) \\
&+ \eta_\gamma \log\left(E_{\tilde{\mu}_i\mu}\left[\exp\left(\frac{-\alpha d(\mathbf{c}_1,\mathbf{c}_2) - \tilde{\Gamma}(\mathbf{c}_1) - \Gamma(\mathbf{c}_2)}{\eta_\gamma}\right)\right]\right) \\
&+ \eta_{\tilde{\mu}} \log\left(E_{\tilde{\mu}_i}\left[\exp\left(\frac{V(\mathbf{c}) + \tilde{\Gamma}(\mathbf{c})}{\eta_{\tilde{\mu}}}\right)\right]\right) \\
&+ E_\mu\left[\Gamma(\mathbf{c})\right].
\end{aligned}
$$

We see that we again obtain expectations over distributions from which we can easily sample. However, we now also need to approximate $\Gamma(\mathbf{c}_1)$ and $\tilde{\Gamma}(\mathbf{c}_2)$ by some parametric function approximators to be able to optimize the dual.