Planning in Stochastic Computation Graphs: Solving Stochastic Nonlinear Problems with Backpropagation

Thiago Pereira Bueno

Thesis submitted to
the
Institute of Mathematics and Statistics
of
University of São Paulo
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Program: Computer Science

Supervisor: Professor Leliane Nunes de Barros, PhD

Co-Supervisor: Professor Denis Deratani Mauá, PhD

During the development of this research project
the author received financial support from
National Council for Scientific and Technological Development (CNPq), grant #141379/2015-4,
and São Paulo Research Foundation (FAPESP), grant #2016/22900-1.

São Paulo, August 31st, 2021

Thesis Committee:

Leliane N. de Barros, Chair

Fabio Cozman

Felipe Meneguzzi

Scott Sanner

Felipe Trevizan

Happiness only real when shared.

Abstract

BUENO, T. P. **Planning in Stochastic Computation Graphs: Solving Stochastic Nonlinear Problems with Backpropagation**. 2021. Thesis (PhD) - Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

Deep Learning has achieved remarkable success in a range of complex perception tasks, games, and other real-world applications. At a high level, it can be argued that the main reason behind the astonishing performance of deep neural networks is the stochastic gradient descent method, which is based on the well-known error backpropagation algorithm. Inspired by the recent applications of deep learning, we propose to investigate the opportunities and challenges in adapting the backpropagation algorithm as a planning technique in continuous sequential decision-making problems. We make the key observation that if a differentiable model of the dynamics of a system can be made available, then an autonomous agent can leverage the advanced gradient-based optimizers developed in the context of learning algorithms to solve long-horizon planning problems. Besides reformulating the recently-proposed deterministic planning through backpropagation algorithm as a form of gradient-based trajectory optimization technique, we propose several extensions to the more general setting of stochastic decision processes in AI planning. In particular, we propose a framework to train Deep Reactive Policies offline for fast decision-making based on stochastic computation graphs and the re-parametrization trick. In addition, we investigate how the duality theory of information relaxation can be adapted to obtain a gradient-based online planning algorithm that interleaves optimization and execution. Empirical experiments show the effectiveness of our proposed approaches in a variety of sequential decision-making problems exhibiting nonlinear dynamics and stochastic exogenous events, such as path planning, multi-reservoir control and HVAC systems.

Keywords: probabilistic planning, Markov Decision Process, MDP, stochastic computation graphs, policy search, trajectory optimization, information relaxation, stochastic gradient descent, deep neural nets, deep learning.

Resumo

BUENO, T. P. **Planejamento em Grafos de Computação Estocástica: Resolvendo Problemas Estocásticos Não-Lineares com Retropropagação de Erros**. 2021. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

A área de Aprendizado Profundo tem obtido grande sucesso em tarefas complexas de percepção, jogos e outras aplicações práticas. Pode-se argumentar, de uma perspectiva geral, que a principal razão do desempenho surpreendente de redes neurais profundas está relacionada ao método do gradiente descendente, o qual por sua vez é baseado no reconhecido algoritmo de retropropagação de erros. Inspirado por aplicações recentes de aprendizado profundo, propõe-se investigar os desafios e oportunidades em adaptar a retropropagação de erros como uma técnica de planejamento em problemas de tomada de decisão sequencial em espaços contínuos. Observa-se, em particular, que se um modelo diferenciável da dinâmica do sistema sob controle estiver disponível, então é possível que um agente autônomo possa se aproveitar de otimizadores avançados baseados em gradientes desenvolvidos no contexto de algoritmos de aprendizado para resolução de problemas de planejamento de longo horizonte. Além de reformular a técnica recentemente proposta de planejamento via retropropagação como uma forma de otimização de trajetória baseada em gradiente, essa tese de doutorado propõe diversas extensões para o caso mais geral de problemas de decisão sequencial em espaços contínuos sob incerteza estocástica estudados em planejamento automatizado em inteligência artificial. Em particular, propõe-se um arcabouço de pré-treinamento de Políticas Reativas Profundas com foco na abordagem de tomada de decisão rápida baseado em grafos de computação estocástica e na técnica de re-parametrização de distribuições. Além disso, investiga-se como a teoria da dualidade de relaxação de informação pode ser adaptada para o desenvolvimento de algoritmos de planejamento baseados em gradientes que intercalam otimização e execução. Resultados empíricos mostram a efetividade da abordagem proposta em problemas de tomada de decisão sequencial envolvendo dinâmica não-linear e eventos exógenos estocásticos, como por exemplo, planejamento de caminho, controle de recursos em multi-reservatórios e controle de sistemas físicos de aquecimento, ventilação e ar condicionado.

Palavras-chave: planejamento probabilístico, Processo de Decisão Markoviano, MDP, grafo de computação estocástica, busca de política, otimização de trajetória, relaxação de informação, gradiente descendente estocástico, redes neurais profundas, aprendizado profundo.

Acknowledgments

Being a PhD student in the Laboratory of Artificial Intelligence and Formal Methods (LIAMF) at the Institute of Mathematics and Statistics (IME) of the University of São Paulo (USP) was truly an opportunity of a lifetime for me.

First of all, I am very fortunate to have been encouraged to apply to the graduate program by Prof. Leliane Nunes de Barros who subsequently accepted me as one of her fellow students. I am very thankful for her initial vote of confidence, and above all, for her support all the way during my formative years at IME-USP. I honestly do not think I would have made to the end of my doctorate without her enthusiasm and optimism. I also recognize that her encouragement for me to apply to several summer schools and the ICAPS doctoral consortium was providential for me to have contact with world-renowned researchers and other fellow students; an experience that certainly helped consolidate my research interests in AI planning and machine learning.

Furthermore, I want to thank Prof. Denis Deratani Mauá, my co-adviser, for introducing me to the fascinating world of probabilistic modeling and inference, and perhaps more importantly, for teaching me the value of healthy skepticism in the face of all the buzzwordiness of current trends in machine learning; though I feel the lesson was hard at times, I believe it made me a better critical thinker in general and a more pragmatic researcher, and for that I am very grateful. I will forever remember in great delight the many informal chats we had about AI, engineering, and computer science that were casually joined by various professors during coffee breaks. These moments were truly special and served to keep my curiosity in AI research always high.

I also want to thank Prof. Scott Sanner of University of Toronto for having accepted me as an international visiting graduate student at the Data-Driven Decision Making (D3M) lab for a year. Prof. Sanner is a beacon of vision and clarity in this rapidly-evolving and sometimes twisted world of AI and machine learning. I certainly have a much broader vision of the world of AI now. More importantly, though, his honesty and openness about the personal life of a researcher showed me a whole different perspective on the true meaning of being a Doctor of Philosophy.

Last but not least, I want to express my deepest gratitude to all my fellow colleagues and friends at University of São Paulo, specially Felipe Salvatore, Paula Moraes, Erika Guetti, Thiago Simão, Viviane Bonadia, Ignasi Andres, Thiago Maia, Angelo Lovatto, and Renato Scaroni. The academic life with you was a much richer experience than I could have ever imagined.

Agradecimentos

Toda a minha jornada acadêmica durante os anos de graduação e pós-graduação não teria sido possível sem o apoio incondicional de minha família. Sou e serei eternamente grato a meus pais, tanto pelo incentivo à educação que me deram desde a minha infância, quanto pelos inúmeros exemplos de carinho, ética e dedicação com os quais me presentearam durante toda a minha vida. Hoje tenho a lucidez para perceber que todos seus esforços sempre foram na direção de me ajudar a ser uma pessoa melhor e a realizar todos os meus sonhos. Sem o amor de vocês eu jamais teria chegado até esse momento.

Agradeço também à minha companheira de todas as horas, Amanda Ferrari. Sinto que necessitaria de pelo menos o mesmo número de horas de que precisei para escrever essa tese para realmente conseguir colocar em palavras todo o meu sentimento por você. Ao relembrar nossos primeiros encontros, não consigo pensar senão em Oscar Wilde que em sua poesia dizia: "*Escolho meus amigos não pela pele ou outro arquétipo qualquer, mas pela pupila. Tem que ter brilho questionador e tonalidade inquietante.*" E de todas as pessoas que conheci, você certamente se destaca como aquela cujo brilho no olhar, carinhoso e sagaz, mais me inspira e me desafia. É completamente desnecessário dizer o quão fundamental na minha vida você se tornou.

Por fim, gostaria de deixar registrado meus agradecimentos especiais a todos os entes queridos da família Ferrari: Naná, Girsão, Bruno, Gilsinho e Nataliya, e por último, mas não menos importante, Schummy. Sinto-me sinceramente privilegiado em ter sido acolhido da forma que fui em sua casa desde o primeiro dia. Agradeço, em particular, todos os incontáveis almoços, jantares, banquetes, churrascos, sardelas e pãezinhos do Gonçalo com café que tive o prazer de dividir ao lado de pessoas tão sinceras e verdadeiras como vocês. Espero poder um dia retribuir toda a atenção, cuidado, amor e paciência que tiveram comigo ao longo de todos esses anos.

Com carinho, dedico esta tese a todos vocês.

Contents

List of Abbreviations

AC	Actor-Critic
ADP	Approximate Dynamic Programming
AI	Artificial Intelligence
BPTT	Backpropagation-Through-Time
BRTDP	Bounded Real-Time Dynamic Programming
CEC	Certainty-Equivalent Control
CVaR	Conditional Value-at-Risk
E2C	Embed-To-Control
DBN	Dynamic Bayesian Network
DCP	Deterministic Control Problem
DDP	Differential Dynamic Programming
DP	Dynamic Programming
DDPG	Deep Deterministic Policy Gradient
DRP	Deep Reactive Policy
FF	Fast-Forward
GP	Gaussian Process
GPS	Guided Policy Search
GAMPS	Gradient-Aware Model-based Policy Search
GRU	Gated Recurrent Network
HOLOP	Hierarchical Open-Loop Optimization
HOOT	Hierarchical Optimistic Optimization applied to Trees
HOP	Hindsight Optimization

HVAC	Heating, Ventilation, and Air Conditioning
iLQR	Iterative Linear Quadratic Regulator
ICAPS	International Conference on Automated Planning and Scheduling
KKT	Karush-Kuhn-Tucker conditions
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
LRTA*	Learning Real-Time A*
LRTDP	Labeled Real-Time Dynamic Programming
LSTM	Long-Short Term Memory
MAAC	Model-Augmented Actor-Critic
MAB	Multi-Armed Bandit
MAGE	Model-based Action-Gradient Estimator
MBPO	Model-Based Policy Optimization
MCTS	Monte-Carlo Tree Search
MDP	Markov Decision Process
MILP	Mixed-Integer Linear Programming
MLE	Maximum-Likelihood Estimation
MLP	Multi-Layer Perceptron
MPC	Model-Predictive Control
MSE	Mean Square Error
MVE	Model-based Value Estimation
NDP	Neuro-Dynamic Programming
NPG	Natural Policy Gradient
PBT	Population-Based Training
PEGAUS	Policy Evaluation-of-Goodness And Search Using Scenario
PETS	Probabilistic Ensembles with Trajectory Sampling
PI	Policy Iteration
PILCO	Probabilistic Inference for Learning Control

POMDP	Partially-Observable Markov Decision Process
PPO	Proximal Policy Optimization
PSGD	Projected Stochastic Gradient Descent
RDDL	Relational Dynamic Influence Diagram Language
ReLU	Rectifier Linear Unit
RL	Reinforcement Learning
RMS	Root Mean Square
RNN	Recurrent Neural Net
SAC	Soft Actor-Critic
SCG	Stochastic Computation Graphs
SCP	Stochastic Control Problem
SDM	Sequential Decision-Making
SDP	Symbolic Dynamic Programming
SGD	Stochastic Gradient Descent
SOGBOFA	Symbolic Online Gradient Based Optimization for Factored Actions
SQP	Sequential Quadratic Programming
STEVE	Stochastic Ensemble Value Expansion
SVG	Stochastic Value Gradient
TD	Temporal Difference
TD3	Twin Delayed Deep Deterministic policy gradient
TRPO	Trust Region Policy Optimization
UCB	Upper-Confidence Bound
UCT	Upper-Confidence Tree
UPN	Universal Planning Networks
VAE	Variational Auto-Encoder
VI	Value Iteration
VIN	Value Iteration Networks
X-MDP	eXogenous Markov Decision Process

List of Symbols

x	state vector
u	action vector
$\mathcal{X}$	state space
$\mathcal{A}$	action space
T	transition function
R	reward function
t	timesteps
H	truncated horizon
π	policy
V^π	value function under policy
$\hat{Q}_t$	reward-to-go from timestep t
$\mathbb{E}$	expectation
$\mathbb{E}_\pi$	expectation w.r.t. policy
τ	state-action trajectory
$\mathbf{x}$	state trajectory
$\mathbf{u}$	action trajectory
J	loss function
θ	input parameter
Θ	input parameter set
∇	gradient vector
∂	partial derivative
ϵ	noise random variable
ξ	scenario random variable
Ξ	set of possible scenarios
$p(\cdot)$	probability density function
$p(\cdot\|\cdot)$	conditional probability density function

List of Figures

Chapter 1

Introduction

Decision making is a core component of any autonomous intelligent agent. In one form or another, the very concept of agency is intrinsically dependent on the ability to leverage a decision-making process. Needless to say, both humans and machines are faced on a daily basis with a large number of complex decision-making problems. In this context, it is very easy to foresee the ever-increasing need of effective automated decision procedures in our modern society, be it to help humans in possibly error-prone and tedious decisions or to make machines more cost effective and flexible in its capabilities.

An important type of decision-making problems commonly studied in Artificial Intelligence (AI) and engineering is the class of *sequential decision-making* problems. In this setting, an agent is mainly concerned with the following key question:

> *How to best choose what to do now when a decision made at one time might affect future decisions in complex and unpredictable ways?*

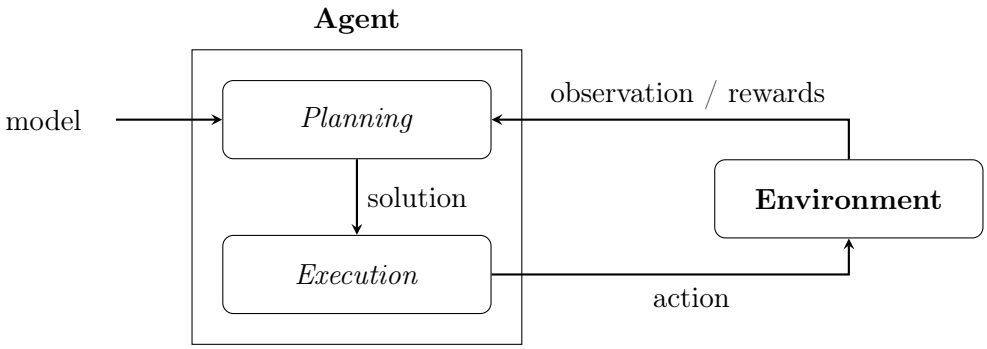


Figure 1.1: *Agent-Environment interaction in a sequential decision-making problem: an autonomous agent selects its next action to execute based on automated planning.*

Generally speaking, the formulation of a sequential decision-making problem consists of an autonomous agent (also known as a decision maker or controller) that interacts with an environment (also known as a system) with the objective of influencing its evolution according to a given metric of interest and/or with the intent of achieving a particular set of goals. Figure 1.1 illustrates the agent-environment cycle of interaction in which an action to be executed in the environment is chosen by the agent as a function of current and possibly previous observations and rewards received

from the environment. We are specifically interested in this work in agents that use a model of the environment for selecting actions. In other words, we shall consider agents that leverage an automated planning component whose responsibility will be to reason over possible future outcomes of its actions using the model as a guiding reference in order to come up with optimized decisions in each situation.

Examples of sequential decision-making problems abound both in the scientific literature and in real-world practical applications. For instance, in goal-oriented navigation tasks an agent must find a sequence of moves that will lead it to a target position while avoiding obstacles and other undesired regions. In dynamic resource management, a decision maker has to decide how to best allocate scarce resources to maximize payoff while striving to meet future demands. In control applications in physical systems, a controller has to reason over energy consumption, nominal constraints, and other safety and robustness considerations in order to achieve some intended behavior. Regardless of the specifics of each problem, the inevitable tension between maximizing (minimizing) short-term and long-term benefits (costs) must be balanced accordingly. In other words, the current decision must forcibly incorporate essential aspects of possible futures that could unfold, if an agent seeks to optimize its long-term performance.

Historically, there has been two alternatives in AI to tackle sequential decision-making problems: (*i*) automated AI planning and (*ii*) reinforcement learning. From a high-level perspective, automated planning is concerned with finding a solution through automatic reasoning over a model description of the system, whereas reinforcement learning attempts to learn the solution from directly interacting with the system without the need of any model. Both approaches have achieved remarkable successes in several applications, though with very different strengths and weaknesses.

In this thesis, we will be interested in studying **model-based planning approaches to the particular class of continuous state-action sequential decision-making problems**.

This chapters is organized as follows. First, we introduce the class of sequential decision-making in continuous domains and discuss some of the challenges to solve it in optimal form, thus making the case that approximations are necessary for practical applications. Next, we provide an overview and contextualization of *differentiable planning* as an efficient and flexible class of algorithms amenable to solve continuous planning both in the deterministic and stochastic settings. Finally, we present the overall thesis proposal and contributions and close the chapter with some examples of problems used in experiments in this thesis and with an outline of the following chapters.

1.1 Sequential Decision-Making in Continuous Spaces

We shall investigate a particular subset of problems in sequential decision-making involving continuous spaces and stochastic disturbances. A sequential decision problem is described in terms of continuous spaces if either the agent's action variables or the system's state variables are defined over a real-valued set. In addition, a system is said to operate under stochastic disturbances if the state transitions induced by the agent's actions cannot be deterministically explained by the actions' effects. We provide examples of sequential decision-making problems in continuous spaces in Section 1.5.

Although the AI community has achieved tremendous progress in discrete combinatorial decision problems, either through model-based approaches in AI planning or by model-free techniques in

reinforcement learning, we believe that decision-making tasks featuring continuous state and action variables and stochastic transitions have not been fully explored in the general case. This is not gratuitous though. As a matter of fact, applying classical planning and search methods to continuous domains is notoriously difficult due to the infinite action space and the infinite branching factor inherently present in continuous transition functions. In addition, continuous spaces also impose major challenges in the learning setting of RL. For instance, Q-learning (Watkins and Dayan, 1992), an instance of a value-based learning method, would have to commit to a (necessarily coarse) discretization of the action space in order to solve its inner maximization problem. This method leads to a complicated trade-off: overdiscretization can substantially increase the computation resources needed and underdiscretization might likely aggregate radically different actions or states leading to catastrophic failure in applications that require precise control. Policy optimization approaches such as policy gradients (Sutton *et al.*, 1999; Williams and Zipser, 1995), in the other hand, could potentially avoid the issues related to discretization. However, the methods based exclusively on policy optimization are likely to suffer from high-variance.

All and all, it remains a challenge to handle arbitrary stochastic continuous planning problems, specially when the system under control might exhibit arbitrary nonlinearities, in which case the decision-making problem can rapidly become intractable for methods that attempt to find optimal solutions. Therefore, some form of approximation has to be introduced to make the problem solvable in reasonable time. In the next section, we discuss different approximations both in problem and solution spaces.

1.2 Problem Approximations and Approximate Solutions

As discussed in the previous section, it is typically intractable to compute globally optimal solutions in the general case of continuous stochastic domains in all but the smallest problems. Indeed, in several complex applications, the best we can hope for is to find good approximate solutions. In this context, model-based approaches to sequential decision-making in continuous domains can be broadly divided into the following two classes:

i. methods that attempt to find exact solutions to approximations of the problem, and

ii. methods that aim to find approximate solutions to the exact problem.

The objective of problem approximation schemes is to apply a set of simplifying modeling assumptions to derive a related problem that can be efficiently solved, possibly obtaining an optimal solution to the simpler problem. On the other hand, methods that approximate solutions to the original problem do not bother with problem simplifications, but focus on leveraging simple algorithms that could in principle achieve better scalability through increasing computations.

Examples of methods that approximate the original nonlinear planning problem through discretization include Monte-Carlo Tree Search (MCTS) (Chang *et al.*, 2005; Kocsis and Szepesvári, 2006), numeric planning, and Q-learning (Watkins and Dayan, 1992), as well as approaches that resort to first and/or second order approximations such as Symbolic Dynamic Programming (SDP) (Sanner *et al.*, 2011; Vianna *et al.*, 2015; Zamani *et al.*, 2012), MILP-based planning (Say, 2021; Say *et al.*, 2017), and differentiable dynamic programming (DDP) (Jacobson and Mayne, 1970) and

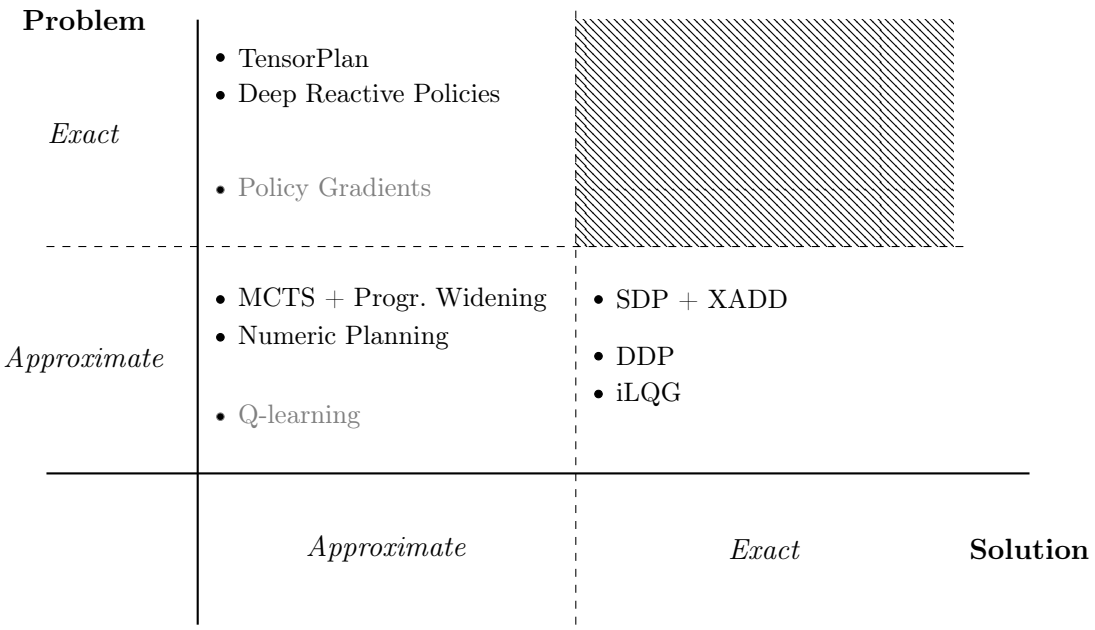


Figure 1.2: *Algorithms for nonlinear sequential-decision making problems: model-based planning approaches in black and model-free learning methods in light gray; the vertical axis represents whether or not the original problem can be solved without problem simplification/approximation and the horizontal axis represents whether or not the solution to the original or approximate problem can be obtained exactly.*

iLQG (Li and Todorov, 2004) from the optimal control literature. In contrast, methods that are general enough to avoid the need to approximate the original nonlinear problem at the cost of settling for approximate solutions include differentiable planning (i.e., methods based on planning through backpropagation such as TensorPlan (Wu *et al.*, 2017) planner for optimizing plans and Deep Reactive Policies (Bueno *et al.*, 2019) for learning policies parametrized as neural networks in continuous stochastic problems) and model-free policy gradients (Sutton *et al.*, 1999; Williams and Zipser, 1995). Figure 1.2 classifies these approaches according the nature of approximations leveraged in each method. We review these techniques and discuss other related works in Chapter 2.

1.3 Differentiable Planning

In this thesis we investigate and further develop a class of algorithms that depend upon the ideas proposed in the context of *differentiable planning*. In recent years, there has been a resurgence of interest in differentiable models, mainly boosted by the wide applicability of Deep Learning techniques in high-dimensional non-convex prediction problems. Inspired by those recent successes in connectionist machine learning, the methods based on differentiable planning propose to adapt the well-known backpropagation algorithm and gradient descent methods to solve sequential decision-making problems. For this reason, the area of differentiable planning is also known as *gradient-based planning*.

There are two main components of a differentiable planning algorithm: (*i*) the *gradient estimation*, and (*ii*) the *optimization step*. Different methods can be derived depending on how these components are defined. We shall discuss in depth in this thesis a number of alternatives concerning their implementations. Nonetheless, the underlying concept of any differentiable planner

is the framework of *stochastic computation graphs*. In connectionist machine learning, computation graphs play an important role to the general implementation of the forward and backward passes in gradient-based optimization algorithms. More generally though, by providing a structured functional representation, computation graphs provide a way to leverage the compositionality of differentiable local models to build bigger, more expressive models for prediction and inference at the same time that it allows for efficient computation of gradients and high-order Jacobians and Hessians. In cases where such models attempt to capture probabilistic uncertainty, stochastic computation graphs extend the basic computation graphs with stochastic nodes representing sampling procedures based on probability distributions.

From the perspective of gradient estimation, it is important to note that the introduction of stochastic nodes creates difficulties to backpropagate gradients throughout the model as the sampling procedures associated with stochastic nodes might not be directly differentiable. Indeed, the *raison d'être* of stochastic computation graphs is to define the necessary conditions to circumvent this issue. We will present in Chapter 3 a number of generative models and decision-making models in AI and machine learning that leverages the theory of stochastic computation graphs for gradient estimation. In the particular case of differentiable planning, stochastic computation graphs are used to build an unbiased approximation of value function for a given policy through the composition of differentiable dynamics and reward functions. With this differentiable approximation of the value function available, a surrogate loss can be obtained from Monte-Carlo evaluations for which the policy parameters might be optimized through gradient-based optimization approaches such as stochastic gradient descent.

Despite technical differences in implementation, the common theme of differentiable planning algorithms is that the availability of gradients of the dynamics and reward function provides a sound and flexible foundation to develop techniques for reasoning over long-horizon decision-making tasks. As a matter of fact, instead of discretizing the state and action spaces in a sequential decision problem to search for solutions, gradient-based planners have the advantage to directly search over the original continuous space by following gradient directions thus avoiding hard-to-control discretization errors. Furthermore, the underlying formulation of continuous planning as a smooth optimization problem that can be solved by gradient descent techniques also avoids the intractable brute-force maximization over actions that is a necessary component of value-based methods in reinforcement learning. In other words, the existence of gradients through differentiable models of the environment allows to derive efficient mechanisms to approximate the solution of the original planning problem in contrast to methods that first approximates the problem itself. As we will see in the remainder of this thesis, the application of gradient-based optimization allows to effectively solve an important class of planning problems whilst being flexible enough to be adapted to different settings such as offline and online planning.

1.4 Thesis Proposal and Contributions

In this thesis, we ponder the question:

> *How can we find scalable solutions to continuous planning problems with possibly non-linear dynamics and stochastic exogenous events without resorting to discretization or problem simplification?*

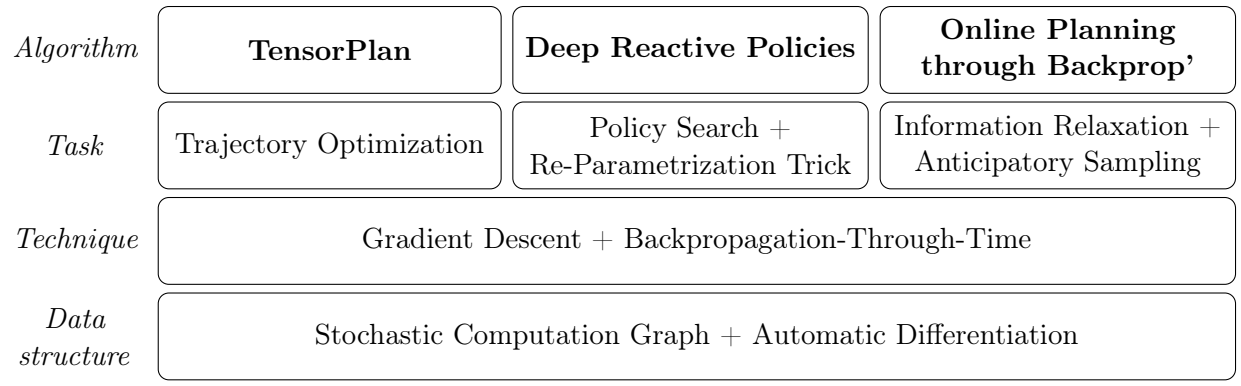


Figure 1.3: *Differentiable Planning: an overview of the algorithms investigated and developed in this thesis, i.e.,* TensorPlan *for deterministic domains, the offline stochastic extension of* Deep Reactive Policies*, and* Online Planning through Backpropagation *based on anticipatory sampling. Every* algorithm *solves a particular* task *formulation, but overall, all algorithms in Differentiable Planning leverage* techniques *based on gradient-based optimization which are all based on the abstraction of stochastic computation graphs that support automatic differentiation.*

This thesis provide an answer to this question by making the key observation that **gradients through differentiable models of the environment can be efficiently leveraged to guide planners to find effective solutions in continuous domains**, therefore avoiding the need to search over discretized spaces and allowing to re-purpose a set of deep learning techniques to optimize solutions for large-scale model-based planning domains. In particular, we explore in depth two classes of differentiable planning algorithms:

(i) *Planning through Backpropagation* for optimizing plans in deterministic domains (or as a sub-routine in online determinization-based solutions for stochastic domains); and

(i) *Deep Reactive Policies* for training parametric policies for fast decision-making during execution in stochastic problems.

In their seminal paper Wu *et al.* (2017) introduced the foundations of planning through backpropagation in deterministic planning domains in their algorithm (informally) known as *TensorPlan*. Inspired by the recurrent computation of Recurrent Neural Networks (RNN), TensorPlan leverages the backpropagation-through-time technique to optimize the model inputs (i.e., the agent's actions) instead of the internal neural representations. In this thesis, we reinterpret TensorPlan and propose to formulate **planning through backpropagation as trajectory optimization** (Scaroni *et al.*, 2020). We remark that this reinterpretation makes interesting connections to control theory and expands the understanding of differentiable planning as general gradient-based methods that can be built on top of different optimization formulations which may lead to novel algorithms in the future.

In addition, we propose to extend planning through backpropagation to solve **offline stochastic planning problems through Deep Reactive Policies trained with policy search** (Bueno *et al.*, 2019) over stochastic computation graphs whose stochastic nodes are amenable to reparametrization. Finally, we extend and combine these ideas to implement an online differentiable planner based on information relaxation and anticipatory sampling. Figure 1.3 summarizes the approaches studied in this thesis.

The main contributions of this thesis are:

- The formulation of *Planning through Backpropagation as a Trajectory Optimization* method and a comparison with differential dynamic programming (chapter 4);
- A training method of *Deep Reactive Policies* for offline differentiable planning in stochastic nonlinear domains (chapter 5); and
- An extension of planning through backpropagation to online planning via information relaxations, anticipatory sampling, model predictive control and rollouts (chapter 6).

1.5 Examples

In this section, we informally describe different examples of planning problems that can be solved with differentiable planning techniques. In following chapters, we shall present and discuss specific models inspired by the applications introduced here. Our goal in this section is therefore to present these examples in order to motivate their practical importance from a high-level planning perspective. We defer technical details concerning their formulation as benchmarks used in experiments to Chapters 4, 5, and 6.

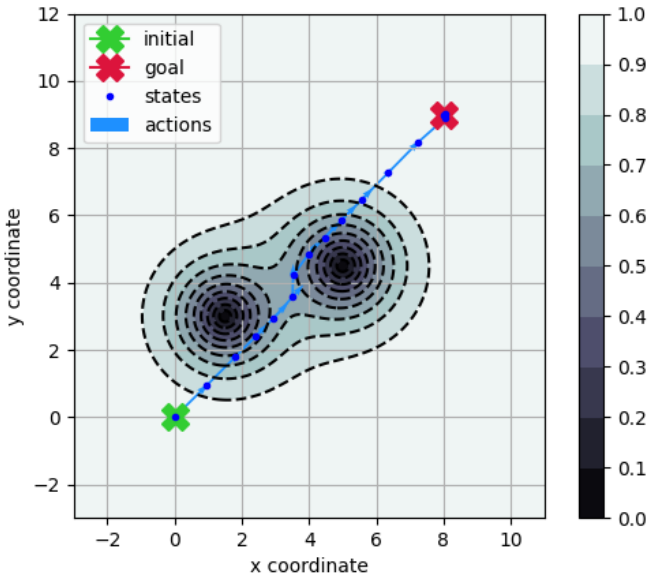


Figure 1.4: *Navigation 2D: an agent is concerned with finding a cost-effective path from an initial position to a fixed goal position. The degree of deceleration in the 2D grid scales from 0 to 1 as shown by color bar on the right and depends on the distance to the center of the deceleration zones. Darker colors indicate points of high deceleration to be avoided. The dashed contour lines show coordinates with the same deceleration value.*

Example 1.5.1. (Path and Motion Planning)

In a path planning problem (Faulwasser and Findeisen, 2009), a decision maker is concerned with finding a sequence of actions (i.e., a path) that lead the environment from an initial state to a goal state, subject to dynamics, safety, and other task-dependent constraints. In general, path planning problems are formulated as 1-, 2-, or 3-dimensional spatial problems. Motion control extends the formulation of path planning to the general n-dimensional space of configurations. It has been extensively applied to robotics. Both classes of problems are very interesting from the point

of view of sequential decision making because early actions can seem quite promising if considered in a myopic, isolated way. For instance, consider a typical path planning task. It is not hard to imagine a short-term scenario in which the best course of action might be perceived by the agent to move straight to the goal. However, if there are longer-term important effects, such as distant, but unavoidable obstacles, the agent might be obliged to take a costly detour at the beginning of its path in order to make progress in the future. These trade-off decisions are at the core of difficult planning tasks whose challenges can be exacerbated in nonlinear domains. Figure 1.4 illustrates a path planning task in 2D space in which the task is to get to goal position as fast as possible. The problem is complicated by the effect of obstacles modeled as deceleration zones that force the agent to consider detours in the straight-line ideal plan. We shall formalize and solve similar problems in Chapter 4.

Example 1.5.2. (Water Reservoir Control)

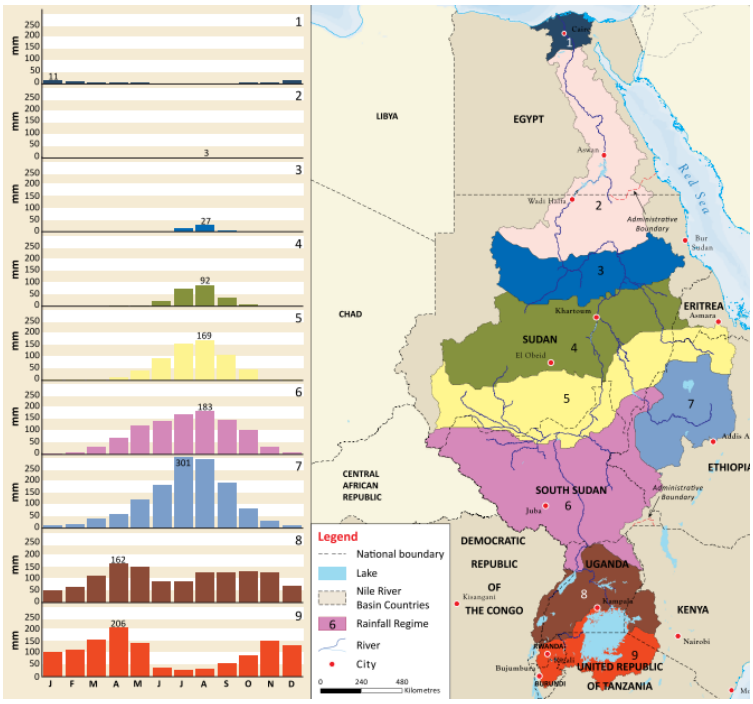


Figure 1.5: *Water reservoir control: (left) rainfall regimes over the Nile basin (period from 1961 to 1990); (right) geography of the Nile basin. Water management is a fundamental issue in the region that depends on an effective planning over several months to avoid critical lack of resources. Note that different regions must operate within radically different hydrology perspectives. Image extracted from UNEP (2013).*

Water reservoir management (Castelletti et al., 2008; Sangiorgio and Guariso, 2018) is an increasingly important and very active area of research and development. It is mainly concerned with the control of a system of interconnected water reservoirs (e.g., natural lakes, artificial reservoirs) and consumers (e.g., hydroelectric power plants, agricultural districts) so that each reservoir operates within a nominal level range in order to optimally attend the consumers' demand. The problem is substantially complicated by a number of uncertainties and hard-to-model characteristics (e.g., rainfall, uncontrolled natural and anthropized catchments). In this context, the decision maker must reason over possible uncertain future events whilst managing conflicting interests and prepare contingencies accordingly. From a planning viewpoint, the problem becomes challenging as the trivial policy that increases the outflows of reservoirs in overflow and reduces the outflows of reservoirs in

underflow needs to be carefully and dynamically fine-tuned to the topology of the problem (i.e., how different reservoirs are interconnected) according to the evolution of the uncertain exogenous events such as the rainfall. In addition, we remark that in practice, due to scarce resources and political instabilities, water reservoir management in several parts of the world must also satisfy a number of complex economic, social, and environmental constraints (UNEP, 2013). Figure 1.5 outlines a real situation in which effective water reservoir management is critical for the economical development of several counties in the Nile basin.

Example 1.5.3. HVAC (Heating, Ventilation, and Air Conditioning)

HVAC control systems (Agarwal et al., 2010; Ghahramani et al., 2020; Lazic et al., 2018) attempt to maintain the temperature and humidity in a given area under a nominal range. This is important in industrial applications for the reasons of process efficiency and safety, but also for thermal comfort in buildings. An HVAC system typically controls the volume of conditioned air sent by supply fans and the heating or cooling coil responsible for injecting or removing thermal energy, respectively. The decision problem is complicated due to the complex interaction between the heat transfer among adjacent rooms and uncontrolled external and internal thermal loads. In addition, a decision maker must strike a good trade-off between the monetary cost associated with the energy consumption of the system in operation and the nominal temperature and humidity range requirements. Figure 1.6 illustrates an HVAC system for controlling the temperature of six rooms.

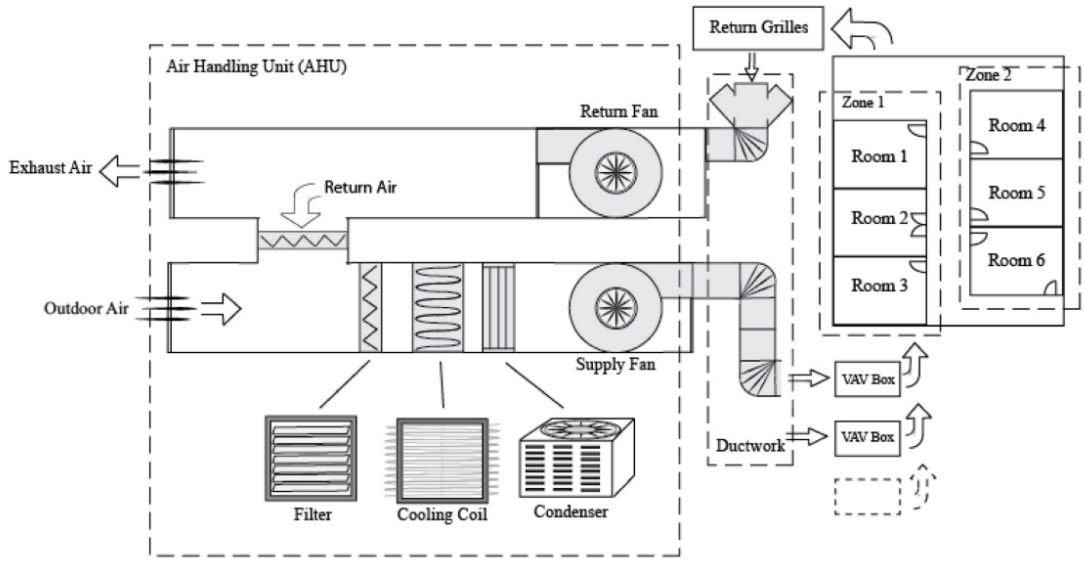


Figure 1.6: *HVAC operational system: the schematic diagram shows six adjacent rooms whose temperature and humidity are to be controlled by an air handling unit that can heat or cool the air temperature supplied through the fan. Image extracted from Ghahramani et al. (2020).*

1.6 Outline

This thesis is organized as follows.

Chapter 1: In this chapter, we introduced the class of sequential decision-making we are interested to address in this thesis, namely *stochastic nonlinear planning problems* and provided

an overview and contextualization of the techniques based on *differentiable planning* that we will develop in the following chapters.

Chapter 2: We start by reviewing basic concepts of sequential decision-making in AI and engineering. We provide an overview of the formulations and methods in probabilistic planning in AI, reinforcement learning, and optimal control as our main contributions are closely related to important ideas in those areas.

Chapter 3: We formalize the notion of *Stochastic Computation Graphs* and discuss Monte-Carlo techniques used for gradient estimation of parametric functions defined over expectations. This is important both from the high-level perspective of unifying the presentation of the proposed methods and from the low-level implementation of the underlying differentiable planning algorithms.

Chapter 4: We introduce the basic ideas of differentiable planning and apply them in the specific case of deterministic domains. We review important prior work on *Planning through Backpropagation* and provide additional experiments and detailed discussions of its computational challenges.

Chapter 5: We extend differentiable planning to the case of stochastic domains. We discuss our proposed approach of offline training *Deep Reactive Policies* (DRPs) via stochastic gradient descent in order to implement fast decision-making in problems involving stochastic exogenous events and nonlinear dynamics and costs.

Chapter 6: We continue to investigate different ways to make planning through backpropagation amenable to sequential decision-making under uncertainty. In particular, we propose to interleave planning and execution to obtain an open-loop policy by leveraging the certainty-equivalence principle and other forms of *information relaxation* amenable to *online planning*.

Chapter 7: We close this thesis by providing an overall discussion of the interplay of planning, acting, and learning as well as discussing some promising avenues for research in differentiable planning and model-based sequential decision-making.

Part I

Literature Review

Chapter 2

Sequential Decision Making in Artificial Intelligence and Engineering

The problem of making decisions in environments that are dynamic and not fully predictable or observable is still a great challenge in the development of autonomous agents. Researchers and practitioners in AI and engineering have been interested in finding efficient solutions for such problem for decades. Of course, different areas come with different sets of assumptions, expectations, and focus on particular applications. Nevertheless, there has been a great deal of cross-fertilization between these areas both in terms of conceptual ideas and practical methods.

In this section, we present a high-level introduction to the general framework of *Sequential Decision-Making* (SDM). We start by identifying the overall components of the SDM problem as well as the requirements for an autonomous agent to be able to achieve intelligent behavior in the framework. In addition, we present a discussion over the terminology issues around models and simulators. Finally, we instantiate the general ideas of the SDM framework to the specific settings of *Probabilistic Planning* in AI and Reinforcement Learning, as well as in the context of the area of *Optimal Control* in engineering.

Our objective in this chapter is threefold:

(i) to review the relevant literature in AI planning, RL and control to provide a broad view of the similarities and differences of their formulations and methods;

(ii) to present the mathematical background required for understanding the technical details of the contributions of this thesis as presented in following chapters; and

(iii) to introduce the scientific notation used throughout the manuscript.

2.1 Sequential Decision-Making Framework

The theory of SDM is mainly concerned with the tension between maximizing short-term benefits and incurring temporary losses in the hope of bigger gains in the future. This tension is always unavoidable if the agent's ultimate goal is to find the optimal behavior in a dynamic environment. The hardness of problem comes from the fact that there is an intrinsic, possibly noisy and delayed, interference between decisions in different stages of the process. In this regard, SDM problems

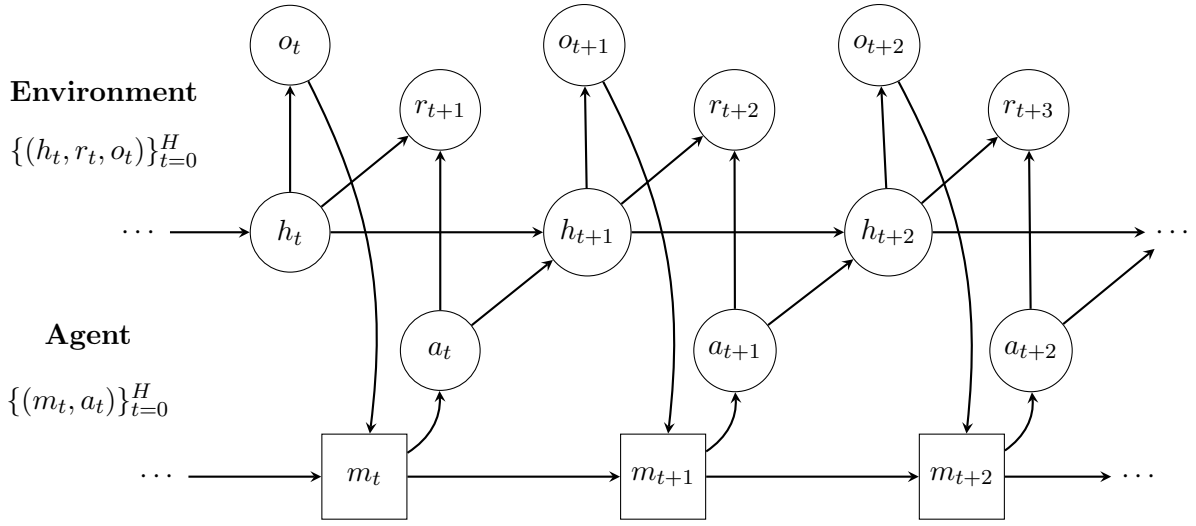


Figure 2.1: *Sequential Decision-Making: an agent sequentially interacts with an environment generating streams of experience, $\{(m_t, a_t)\}$ and $\{(h_t, r_t, o_t)\}$, respectively. At each decision stage t, an agent selects an action a_t to execute in the environment based on its internal knowledge or memory m_t. The environment transitions to a new state generating an updated history h_{t+1} and responds to the agent with an observation o_{t+1} and reward r_{t+1}. Depending on the specific formulation, these interleaving processes may continue indefinitely or until a fixed horizon or condition is met.*

are very different from static tasks in decision theory as the information necessary for solving the problem is entangled in a temporally-correlated stream of experiences generated by the agent.

The SDM framework breaks down the complex multi-stage process into a series of decision steps. It provides a principled understanding of how a dynamical system evolves over time under the effects of the agent's actions and also how the internal state of the agent is updated in the light of new observations. Figure 2.1 shows a fragment of the overall sequential decision-making process unrolling in time. Definition 2.1.1 describes the main components of an SDM process[1].

Definition 2.1.1. (SDM components) *The main components of an SDM problem are:*

1. *the* decision stages*: a sequence of discrete steps, $t = 1, 2, \cdots, H$;*
2. *the* state space*: a (finite or infinite) set of possible states of the system;*
3. *the* action space*: a (finite or infinite) set of available actions to the agent;*
4. *the* transition kernel*: a mapping from state s_t and action a_t to next state, $s_{t+1} = T(s_t, a_t)$;*
5. *the* history*: a sequence of past states and actions, $h_{t+1} = (h_t, a_t, T(s_t, a_t))$;*
6. *the* reward function*: the numeric preference of a state-action pair, $r_{t+1} = R(s_t, a_t)$;*
7. *the* observation function*: the sensory data generated from the current state, $o_t = O(s_t)$;*
8. *the* perception function*: the aggregation of information across time, $m_t = O^{-1}(m_{t-1}, o_t)$; and*
9. *the* policy*: the action selection process conditioned on the filtered past experiences, $a_t = \pi(m_t)$.*

[1]Note that at this point in the presentation, we focus on an abstract and intuitive definition of an SDM problem. We defer the full mathematical details to sections that will make specific assumptions over its components.

States and Actions. The evolution of a dynamical system is described by a sequence of *states*. Generally speaking, a state is a set of variables that can model any quantity of interest for the decision maker. A few examples include: (*i*) *resources* (e.g., position/velocity, temperature, energy, fuel, monetary budget, inventory levels, etc), (*ii*) *information* (e.g., prices and demands of a product in previous decision stages in an inventory control problem, historic patterns of rainfall in a water management application, etc), and (*iii*) *beliefs* (e.g., the possible interval of validity of a production capacity, a distribution over future demands and prices, a probabilistic estimate of sensory information such radar/lidar distances in a SLAM (simultaneous localization and mapping) applications, or even epistemic estimates of Q-values for different actions, etc).

Actions correspond to the available decisions under control by the agent. Typically, it can include (*i*) *control* actions to physically induce transitions in the system, and (*ii*) *sensing* actions to gather additional information about the current state.

States and actions are related by the decision made by the agent and the transitions of the dynamical system under control. At a high level, a system starts in a given current state s_t and after the execution of action a_t evolves to a next state s_{t+1} according to its transition function $T(s_t, a_t)$.[2] Though not essentially necessary for the mathematical formalization of SDM problems, we remark that it is usually convenient for practical modeling to break a single transition in a series of intermediate steps. In this context, we say that a system evolve from a pre-decision state s_t, to an action a_t, to a post-decision state-action pair (s_t, a_t), to a new information state (s_t, a_t, ξ_{t+1}), and finally, to the next state x_{t+1}. The additional information variable ξ_{t+1} might correspond to the demand realized for the given decision stage in an inventory control application or to a stochastic disturbance observed after the execution of an action.

We refer to any sequence of state and actions generated according to sequence of steps discussed above as a *trajectory*. In particular, if all states and actions in a trajectory have been realized in the past (w.r.t. the current decision stage), then such trajectory forms part of the *history* experienced by the agent. In the other hand, if the trajectory refers to a possible sequence of states and actions to be realized in the future (usually obtained from a model), then we say it is a future trajectory or a *scenario*. A trajectory can be associated with its *return*, i.e., the total sum of stage-wise rewards $G = \sum_{t=1}^{H} r_t$.

Uncertainty. An SDM problem can naturally incorporate different sources of uncertainty in its modeling. Apart from aleatoric uncertainty inherent to the system under control, epistemic uncertainty can be accommodated in a number of ways. In imperfect information systems, observational noise (e.g., sensoring) and prognostic errors (e.g., forecasting) can be probabilistically modeled. Also, other sorts of epistemic uncertainty typically related learning setting can be be taken into account, for instance, model uncertainty (i.e., the structure of a function approximator) and inferential uncertainty (i.e., the parameters of the model).

Objective Functions. If the system is deterministic, then an objective function can be derived directly from the return of the trajectory induced by a sequence of actions. However, if the system is stochastic, then the return of a trajectory is a random variable, i.e., a random return G, and care must be taken to formalize a sound objective function. We remark that depending on the nature

[2] At this point, we do not distinguish between deterministic or stochastic transition functions.

of the stochasticities considered, different objective functions for the SDM problem might be used. Usually, the expected return is defined as the objective function, though the variance of the return can also be considered in risk-sensitive applications or portfolio optimization applications.

2.1.1 Models and Simulators

In the last section, we abstractly described the main components of the SDM framework, however, in order to tackle an SDM problem in practice, a decision-maker must first adhere to a specific formulation of the decision process at hand. This is the starting point for the theoretical understanding of how the system that an agent aims to control might evolve over time. Particular formulations will naturally dictate a set of assumptions upon which effective algorithms might be developed, e.g., stochastic problems might lead to the need of searching for contingent plans or policies mapping state to actions, partial observability might require a particular type of memory or observation filtering mechanism available to the agent, etc.

In addition to the formulation assumed for the mathematical description of the system's dynamics, an agent may optionally build an internal *model* of the system. Note that this internal representation might be completely separate from the overall SDM formulation and therefore can be simplified for the purpose of computational efficiency accordingly. Model-based approaches enable an important class of algorithms that might allow agents to reason over the long-term outcomes of its decisions in regards to safety, risk, and efficiency before committing to an action.

Model is one of the most overloaded terms used in AI, and also in science in general. It may have very different connotations depending on the viewpoint and traditions of each community. It is usually the case that attempting to fully formalize the notion of a model can bring more confusion than clarity to the matter. Therefore, we shall limit ourselves to an introductory discussion, focusing on relevant examples and highlighting some key aspects in the nature of how models can be used.

Informally speaking, a model is an abstraction that explicitly encodes knowledge about the environment and task. Examples include: a transition/dynamics model $s_{t+1} = T(s_t, a_t)$, a reward model $r_{t+1} = R(s_t, a_t)$, an inverse dynamics model $a_t = T^{-1}(s_t, s_{t+1})$ or $s_t = T^{-1}(s_{t+1}, a_t,)$, a model of state distance $d_{ij} = f_d(s^i, s^j)$, a model of future returns $G_t = Q(s_t, a_t)$ or $G_t = V(s_t)$.

In particular, a dynamics model can be categorized along two orthogonal dimensions:

(i) *type of information*: A dynamics model is said to be *analytic* or *descriptive* if the distribution of next states conditioned on a state-action pair is analytically defined, therefore querying the model for transition probabilities can be made tractable. On the other hand, if only samples can be efficiently generated for the next states, then the dynamics model is said to be a *sample* or *generative* model. Analytic/descriptive models and sample/generative models are also known as *declarative* and *procedural* models, respectively.

(ii) *type of access*: A *reversible* or *resettable* model is one that can be queried for any state at any time, therefore allowing to generate many transition samples from the same current state. An *irreversible* or *non-resettable* can only be queried for the current state and afterwards must compulsorily transition to the next state, i.e., it only allows forward sampling. To avoid confusion, in the remainder of the thesis we shall refer to an irreversible dynamics model as a *simulator*.

Classical examples of reversible analytic models are Bayesian Networks (Darwiche, 2009; Pearl, 1989) and Influence Diagrams (Boutilier, 2005; Howard and Matheson, 2005). Probabilistic programming languages and deep generative models can be used to define reversible sample models.

Example 2.1.1. (DBN application - Bayesian Automated Taxi)

Figure 2.2 shows an example of a Dynamic Bayesian Network modeling the stochastic transition in an automated taxi application. Actions (e.g., "Lateral Action", "Fwd Action") interact with state variables (e.g., "Xpos", "Xdot", "Ypos", "Ydot", "Engine Status") and noisy observations (e.g., "Left Clear", "Right Clear", "Front Clear", "Back Clear" to model the motion of the automated taxi vehicle. Though not shown in the figure, each node in the network has a corresponding conditional probabilistic tables (CPT) that defines the probabilistic dependencies.

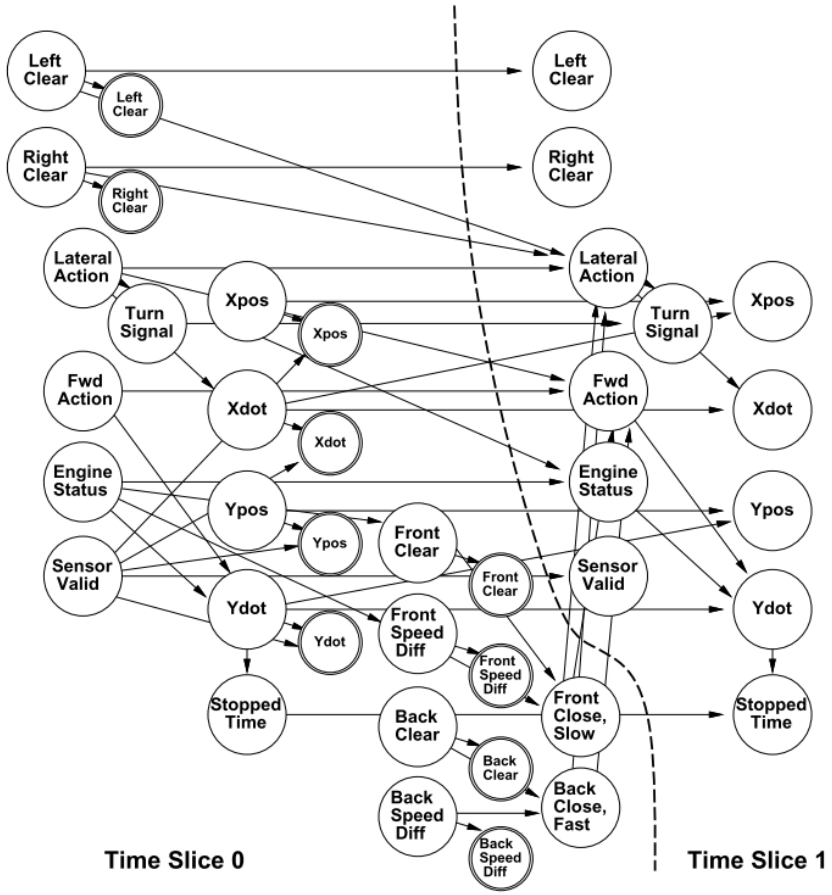


Figure 2.2: *Dynamic Bayesian Network (DBN): Bayesian Automated Taxi (BATmobile) application. Extracted from (Forbes et al., 1995).*

2.1.2 Representational Formalisms

Markov Decision Processes (MDPs)[3] are useful mathematical models for defining a wide class of SDM problems. However, they are not the most suitable languages for describing such problems due to the number and size of the relations and parameters needed to fully specify the planning task. In AI, it has become commonplace to use modular and high-level languages that allows to

[3]We will formalize MDPs in details in Section 2.2.1.

compactly represent problems that are exponentially large in the size of the representation. Representational formalisms such as the *Probabilistic Planning Domain Description Language* (PPDDL) (Younes and Littman, 2004) and the *Relational Dynamic influence Diagram Language* (RDDL) (Sanner, 2010) can be viewed as a convenient way to define semi-analytic models in the sense that in the general case they may not be able to define an analytical form of the next state distributions but can provide some level of information about the structural dependencies between state and action variables. These dependencies are important to be made explicit as they can be exploited computationally.

Finally, it is important to notice that there is an implicit hierarchy in terms of the types of access and information available for the different classes of models. It is always possible to sample next states from an analytic model. Moreover, a reversible model can have its use restricted to that of a forward-only simulator – after all, irreversible sampling is a key aspect of the real world.

Example 2.1.2. RDDL: Game of Life

A RDDL domain defines the dynamics of a system structured by a set of interacting objects. Each object has a set of nonfluent and fluent variables, corresponding to static and dynamic properties. The state of the system is given by the set of fluent variables defined as parametric variables algebraically specified by CPFs (conditional probability functions), i.e., formal sampling rules that describe the evolution of its values. In addition, a reward function describes the preference of a state-action pair from the viewpoint of the planning task. Optionally, state-action constraints can also be considered. The following listing is an example of a RDDL model for the popular Conway's cellular automata "Game of Life".

```
domain game_of_life {
  requirements = { reward-deterministic };

  types {x_pos: object; y_pos : object};

  pvariables {
    PROB_REGENERATE : { non-fluent, real, default = 0.5 };
    NEIGHBOR(x_pos,y_pos,x_pos,y_pos) : {non-fluent, bool, default=false};
    alive(x_pos,y_pos) : {state-fluent, bool, default=false };
    count-neighbors(x_pos,y_pos) : {interm-fluent, int, level = 1};
    set(x_pos, y_pos) : {action-fluent, bool, default=false };
  };

  cpfs {
    count-neighbors(?x,?y) =
      KronDelta(sum_{?x2 : x_pos, ?y2 : y_pos}
        [NEIGHBOR(?x,?y,?x2,?y2) ^ alive(?x2,?y2)]);

    alive'(?x,?y) = if (forall_{?y2 : y_pos} ~alive(?x,?y2))
            then Bernoulli(PROB_REGENERATE)
            else if ([alive(?x,?y) ^ (count-neighbors(?x,?y) >= 2)
```

```
                ^ (count-neighbors(?x,?y) <= 3)]
                | [~alive(?x,?y) ^
                (count-neighbors(?x,?y) == 3)]
                | set(?x,?y))
            then Bernoulli(PROB_REGENERATE)
            else Bernoulli(1.0 - PROB_REGENERATE);
  };

  reward = sum_{?x : x_pos, ?y : y_pos} alive(?x,?y);

  state-action-constraints {
    (PROB_REGENERATE >= 0.0) ^ (PROB_REGENERATE <= 1.0);
    forall_{?x : x_pos, ?y : y_pos} alive(?x,?y) => ~set(?x,?y);
  };
}
```

2.2 Algorithms and Methods

Approximations. From a high-level point of view, the necessary requirements for an autonomous agent to solve an SDM problem consist of: (1) a summary of the past, (2) a knowledge of its affordances, (3) an efficient mechanism to peak into the long-term future, and (4) a set of predictive capabilities. Depending on the specifics of each of this components, several algorithms and methods can be derived. In general, most algorithms that attempt to solve large-scale SDM problems can be categorized either as *Approximations in Value Space* or as *Approximations in Policy Space*.

Approximations in value space leverage lookahead approximations for planning future actions and reasoning over possible outcomes in a short-term way. Together with a simulation mechanism and possibly an evaluation function, a decision maker can improve upon the basic limited lookahead reasoning to incorporate heuristic information about the long-term consequences of its actions. An implicit policy is obtained by computation over the approximated values. Figure 2.3 shows graphically the general components of a planning method based on approximation in value space. On the other hand, methods based on approximations in policy space directly attempt to obtain a policy representation. An example of such methods is policy search. Usually, policy search over a class of parametric functions is implemented as the direct optimization of a surrogate objective function correlated with the performance of the policy in the given SDM problem.

Finite vs. Infinite Horizon. Besides approximations in value or policy space, a planner can also leverage approximations in term of the number of decision stages it shall consider in its long-term reasoning. In particular, even if the original decision-making task is defined to have an infinite horizon, an agent might want to consider a fixed number of decision stages in order to accelerate the planning computations. And furthermore, it might want to also constrain itself to only search for stationary policies as a way to reduce the memory footprint of the algorithm, even if such policies can be non-optimal for finite horizon problems. This type of horizon approximation is subtle and is typically employed behind the scenes in implementations of learning-based SDM algorithms.

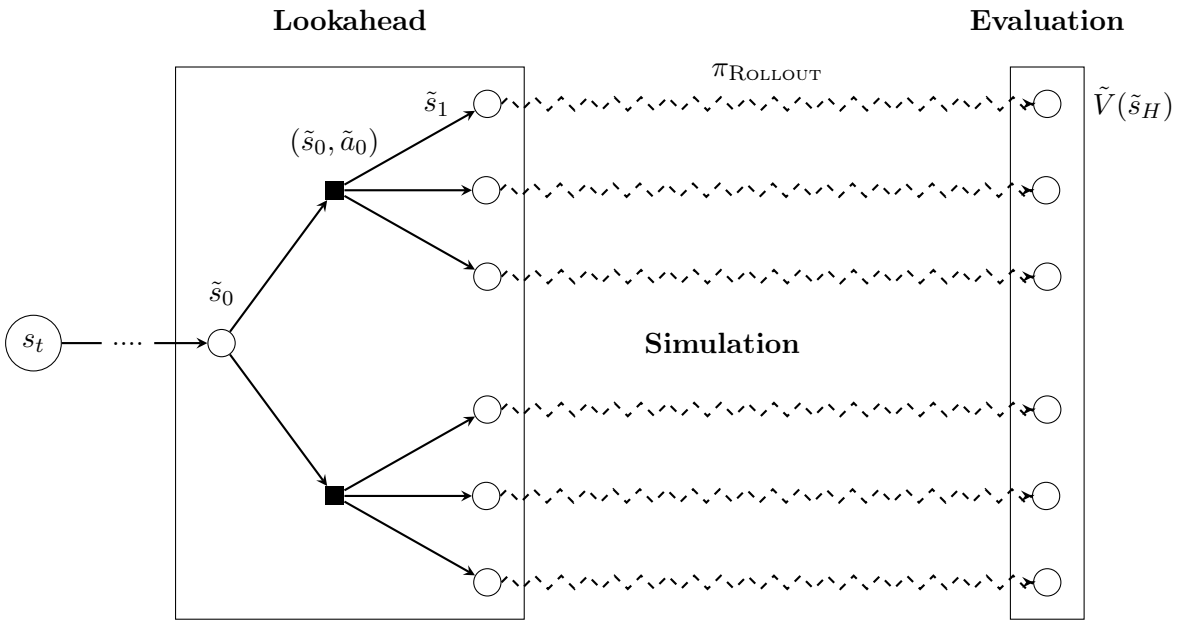


Figure 2.3: *Schematics of search done by approximation in value space.*

Offline vs. Online. Solutions to an SDM can be implemented in two settings: (1) *offline* (a.k.a. *background planning*); or (2) *online* (a.k.a. *decision-time planning*). Offline planners attempt to globally solve the decision-making problem, i.e., their objective is to find a closed-loop policy that can be readily applied to any state in the state space. Conversely, online planners handle the decision-making problem locally, i.e., their objective is to find the best possible action for the current state in each timestep. One limitation of global planning is that the space of policies must be at the same time expressive enough to represent effective strategies but simple enough that the computational resources needed do not become prohibitively high.

In the following sections, we review several methods proposed in the literature of planning and learning algorithms in the context of probabilistic planning, reinforcement learning, and optimal control.

2.2.1 Automated Planning

Planning is the model-based approach to the problem of autonomous agents in traditional AI. In general, planning methods assume knowledge about the environment in the form of a reversible model. Broadly speaking, the overall goal of any planner is to transform model predictions into executable actions. There is a vast literature on AI planning for deterministic, fully observable, and discrete-space problems. However, for the purposes of literature review, we shall focus on the stochastic setting in which the system dynamics is explicitly modeled with a probabilistic model.

Probabilistic Planning

The SDM problem in *Probabilistic Planning* (Boutilier *et al.*, 1995) is typically formulated as a *Markov Decision Process* (MDP) (Puterman, 1994) in the case of perfect state information or as a *Partially-Observable Markov Decision Process* (POMDP) when the system's state is not fully observable by the agent. Informally, MDPs and POMDPs encode the temporal evolution of a stochastic

process controlled by an agent. For the purposes of this thesis, we restrict the presentation to the case of discrete-time, infinite-horizon MDPs.

Definition 2.2.1. (**MDP**) *A* Markov Decision Process *is defined by the tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \rho, T, R, \gamma)$ *where:* $\mathcal{S}$ *is the state space,* $\mathcal{A}$ *is the action space,* $\rho \in \mathcal{P}(\mathcal{S})$ *is the initial state distribution,* $T \colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ *is the transition kernel mapping a state-action pair to a distribution over next states,* $R \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the reward function, and* $\gamma \in [0, 1)$ *is the discount factor. Additionally, we denote by* $\mathcal{A}(s)$ *the set of applicable/valid actions in state* s*.*

Let $\mathcal{M}$ be a discrete-time MDP. The behavior of an agent is defined by a *policy* π. A policy can be stochastic or deterministic.

Definition 2.2.2. (**Stochastic Policy**) *A Markov policy is a mapping from states to distributions over actions,* $\pi \colon \mathcal{S} \to \mathcal{P}(\mathcal{A})$*. A valid policy assigns non-zero probability exclusively to actions in the set of applicable actions in a given state,* $\mathcal{A}(s) \subseteq \mathcal{A}$*.*

Definition 2.2.3. (**Deterministic Policy**) *A deterministic Markov policy is a direct mapping from states to actions,* $\pi \colon \mathcal{S} \to \mathcal{A}(s)$*.*

Whenever an agent interacts with a system modeled as an MDP, it generates state-action trajectories $\tau = (s_0, a_0, s_1, a_1, \cdots) \sim \pi$, where we use the shorthand $\tau \sim \pi$ to denote the stochastic process induced by $s_{t+1} \sim p(\cdot|s_t, a_t) = T(s_t, a_t)$, $a_t \sim \pi(\cdot|s_t)$, and $s_0 \sim \rho(\cdot)$. We associate with a trajectory τ a return defined as the discounted cumulative reward, $G(\tau) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$.

We define the *value function* of a policy π in a given state s by the expected return induced by following π from s:

$$V^{\pi}(s) \doteq \mathbb{E}_{\tau \sim \pi}[G(\tau) \mid \mathcal{M}, s] \ . \tag{2.1}$$

The objective of an agent is to find an optimal policy π^* such that for any state $s \in \mathcal{S}$:

$$V^{\pi^*}(s) \geq V^{\pi}(s), \text{ for any policy } \pi \ . \tag{2.2}$$

It is well known in the MDP literature that a deterministic and stationary policy $\pi \colon \mathcal{S} \to \mathcal{A}$ is sufficient for optimality in fully-observable problems Puterman (1994). Therefore, we shall assume onward that π refers to a deterministic policy, unless explicitly noted otherwise.

In the remainder of this section, we review the main classes of MDP methods typically studied in probabilistic planning: *dynamic programming*, *heuristic search*, and *simulation-based planning*.

Dynamic Programming

Dynamic Programming (DP) was first proposed by Bellman (1957) as an efficient computational approach to solve multi-stage decision processes. The key observation is summarized in its *Principle of Optimality* that naturally allows to break a complex multi-step problem into a series of simpler single-step problem.

Theorem 2.2.1. (Bellman's Optimality) *Let* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \rho, T, R, \gamma)$ *be a discrete-time MDP. Then, the* optimal value function *is given by:*

$$V^*(s) \doteq \max_{\pi} V^{\pi}(s) = \max_{a \in \mathcal{A}(s)} \left[R(s, a) + \gamma \mathbb{E}_{S' \sim T(s,a)}[V^*(s')] \right] \ .$$

Early research on probabilistic planning has mostly focused on exact Dynamic Programming algorithms that exhibit optimality guarantees in the tabular case. *Value Iteration* (VI) (Bellman, 1957) and *Policy Iteration* (PI) (Howard, 1960) are two representatives of this kind of algorithms.

Theorem 2.2.2. (Value Iteration) *Let $\mathcal{T}\colon V_i \mapsto \mathcal{T}V_i$ be the Bellman operator defined by:*

$$(\mathcal{T}V_i)(s) \doteq \max_{a\in\mathcal{A}(s)} \left[R(s,a) + \gamma \mathbb{E}_{s'\sim T(s,a)}[V_i(s')]\right] \ . \tag{2.3}$$

Then, for any $V_0 \in \mathbb{R}^{|\mathcal{S}|}$, it holds that successive applications of $(\mathcal{T})$ converge to the optimal value function, i.e., $(\mathcal{T})^k V_0 \xrightarrow{k\to\infty} V^*$. We remark that a single application of the Bellman operator is also known as a Bellman backup.

Theorem 2.2.3. (**Policy Iteration**) *Let $\mathcal{T}^\pi\colon V_i \mapsto \mathcal{T}^\pi V_i$ be the Bellman evaluation operator for policy π defined by:*

$$(\mathcal{T}V_i^\pi)(s) \doteq R(s,\pi(s)) + \gamma \mathbb{E}_{s'\sim T(s,\pi(s))}[V_i(s')] \ . \tag{2.4}$$

Let π^V be the greedy policy w.r.t. the value estimate V:

$$\pi^V(s) \doteq \arg\max_{a\in\mathcal{A}(s)} \left[R(s,a) + \gamma \mathbb{E}_{s'\sim T(s,a)}[V(s')]\right] \ . \tag{2.5}$$

Then, for any $V_0 \in \mathbb{R}^{|\mathcal{S}|}$ and $\pi_0 \in \mathcal{A}^{|\mathcal{S}|}$, it holds that:

- *successive applications of $\mathcal{T}^\pi$ converge to the value of policy π, i.e., $(\mathcal{T}^\pi)^k V_0 \xrightarrow{k\to\infty} V^\pi$; and*
- *the interleaving of policy evaluation and policy greedification converges to the optimal policy, i.e, $\pi_{k+1} \xrightarrow{k\to\infty} \pi^*$ with $V_k = V^{\pi_k}$ and $\pi_{k+1} = \pi^{V_k}$.*

Heuristic Search

Exact DP algorithms such as VI and PI need to store the whole state space in memory to even get started. The number of states being exponential in the number of state variables, which is known as the *curse of dimensionality*, make the application of exact DP prohibitive to most problems of interest in AI. Asynchronous versions of these methods have been proposed to alleviate the memory cost and to reduce the number of Bellman backups (e.g, Prioritized VI (Wingate and Seppi, 2005), Focussed VI (Ferguson and Stentz, 2004), Topological VI (Dai and Goldsmith, 2007)). Nevertheless, they only partially address the disadvantages of DP as they all still need to visit all states infinitely often to guarantee convergence.

One step towards solving MDPs with large state spaces is to rely on *forward search*, i.e., to only update states that are reachable from the start state, thus obtaining a partial policy defined exclusively at states that can be visited by an optimal policy. The key insight is that VI can be seen as a shortest path algorithm formulated over the state graph induced by the MDP and the optimal policy. This is most clear in deterministic settings, but it is also a sound interpretation in stochastic domains if the search is performed over an AND/OR graph. In any case, however, the implicit search implemented by VI is not guided by any heuristic, a setting in which computation can be easily wasted in irrelevant parts of the state space. In order to remedy this inefficiency, *Planning as*

Heuristic Search (Bonet and Geffner, 1999, 2001) was proposed. The key idea is to leverage declarative representations of the underlying MDP to automatically extract from the problem structure an informative heuristic function to accelerate planning time.

*Learning Real-Time A** (LRTA*) (Korf, 1990) was one of the first algorithms that implemented the idea of heuristic search to deterministic setting. *Real-Time Dynamic Programming* (RTDP) (Barto *et al.*, 1995) extended the approach to the probabilistic setting by using greedy action selection and an admissible heuristic combined with Monte-Carlo trajectory samples. Extensions were subsequently proposed to improve convergence, e.g., LRTDP (Bonet and Geffner, 2003), BRTDP (McMahan *et al.*, 2005), and others. Simultaneously, other heuristic search algorithms that gradually build an optimal solution AND/OR graph beginning from the root node representing the initial state were proposed, e.g., iLAO* (Hansen and Zilberstein, 2001).

Symbolic Dynamic Programming

Symbolic Dynamic Programming (SDP) (Hoey *et al.*, 1999; Sanner *et al.*, 2011; Vianna *et al.*, 2015; Zamani *et al.*, 2012) leverages efficient data structures (i.e., Binary Decision Diagrams (BDDs) (Bahar *et al.*, 1997) (Bryant, 1986), Algebraic Decision Diagrams (ADDs) (Bahar *et al.*, 1997), and other extensions) in order to carry out dynamic programming over finite or infinite sets of states and actions. State-of-the-art approaches based on SDP can optimally solve huge problems in symbolic form, but are rather limited to special cases when continuous variables are used in the state and/or action representations.

Simulation-based Planning

Simulation-based tree search is a class of anytime methods that address the curse of dimensionality through Monte-Carlo sampling (Chang *et al.*, 2005; Kearns *et al.*, 1999; Kocsis and Szepesvári, 2006). The classical solutions purely based on simulations exhibit interesting theoretical properties of convergence to the optimal policy in the limit case independently of the size of the state space. In general, the principle of optimism in the face of uncertainty is instantiated via some statistical test in order to specify an action selection mechanism that achieves a good enough trade-off between exploration and exploitation (e.g., UCB (Auer *et al.*, 2002) is used in MCTS-UCT (Kocsis and Szepesvári, 2006)). Combined with automatically-generated heuristics for value initialization, simulation-based planning has achieved tremendous successes in challenging applications and is considered state-of-the-art in several discrete planning domains.

Extensions of a number of these model-based AI planning algorithms have been proposed to handle continuous state-action spaces. However, a number of theoretical and practical issues hold back the potential of such methods. In particular, methods built on top of search over continuous action spaces have to discretize the values an action can take in one form or another. The issue with action discretization in sequential decision-making is twofold. Firstly, there is no principled way in general to define a fixed or even an adaptive grid to search over. Though a number of heuristics (e.g., progressive widening (Chaslot *et al.*, 2007) or optimistic hierarchical tree expansion (Mansley *et al.*, 2011)) have been successfully applied, in practice, considerable effort and careful attention must be devoted to tuning its hyper-parameters for each task. Secondly, there is a combinatorial explosion of paths from the current state that a search-based agent has to deal with after discretization. Indeed, even if a very coarse grid is employed, the search complexity can rapidly become unmanageable

as its horizon dependence remains exponential even if sparse sampling is used. A possible remedy for this is to combine the search with a generalization mechanism in order to truncate the task horizon and/or to prune search branches. For that matter, the standard solution nowadays is to rely on function approximators learned from experience data. Such approaches seem very promising as a number of recent applications has demonstrated, but unfortunately their success if oftentimes predicated on the availability of huge computational power.

Although heuristic search does not need to visit all states infinitely often to guarantee convergence, the nearly-optimal solution obtained by such methods depends on the size of the subset of reachable states and the quality of the heuristic used to initialize value estimates.

Simulation-based planning (a.k.a., Monte-Carlo planning) help alleviate the curse of dimensionality by allowing selective state and/or state-action backups according to the state visitation induced by the policy. In other words, sampling can be leverage to steer the search to states most likely to be visited by an optimal policy. Differently than heuristic search algorithms that rely on full DP backups, Monte-Carlo planning algorithms use sample backups. A huge body of research was developed on the idea of simulation-based planning. Examples include *Sparse Sampling* (Kearns *et al.*, 1999), POMCP (Silver and Veness, 2010), *Adaptive Sampling* (Chang *et al.*, 2005), *Rollout* (Tesauro and Galperin, 1996), *Monte-Carlo Tree Search* (MCTS) (Browne *et al.*, 2012), PROST (Keller and Eyerich, 2012).

MCTS is likely the most successful example of simulation-based planning. Originally based on the UCB1 (Auer *et al.*, 2002) algorithm first proposed in the Multi-Armed Bandit (MAB) literature, the *Upper Confidence Tree* (UCT) (Kocsis and Szepesvári, 2006) achieved tremendous success in games and other problems with discrete action spaces. Since then, MCTS has been adapted to continuous action spaces as well through adaptive discretization, either by hierarchical methods based on the continuous-armed bandits (e.g., HOOT (Mansley *et al.*, 2011), HOLOP (Weinstein and Littman, 2012)) or by progressive widening techniques (Chaslot *et al.*, 2007; Coulom, 2007). *Trial-based Heuristic Tree Search* (THTS) (Keller and Helmert, 2013) subsumes MCTS, DP, and Heuristic Search by identifying a few common design dimensions among seemingly different methods, i.e., action and outcome selection mechanisms, trial length, heuristic function, and backup function.

Numeric and Hybrid Planning

In numerical planning (Hoffmann, 2011; Ivankovic *et al.*, 2014), the flow of time is adaptively discretized and the system is controlled by a finite set of operators defined over continuous parameters that represent the pre-defined rates of change of numerical fluents. In addition, the transition function describing the evolution of the system is modeled with event-based processes. Planning is then accomplished by iteratively interleaving discretization, search, and plan validation steps until convergence is obtained. On the other hand, symbolic dynamic programming and simulation-based tree search assume a previously obtained and fixed time discretization and therefore directly operate on the discrete time setting.

2.2.2 Reinforcement Learning

Reinforcement Learning (RL) (Sutton and Barto, 2018) is the model-free approach to the problem of intelligent agents in AI. Model-free approaches optimize policies directly from experience

without attempting to explicitly leverage anything about the underlying components of the MDP a priori. Therefore, RL methods usually applied to irreversible models (i.e., simulators) which is in stark contrast to AI planning that in its most standard form assume complete access to reversible models. RL goes by many names depending on the focus on a particular solution class, e.g., Approximate Dynamic Programming (ADP) (Powell, 2007), Neuro-Dynamic Programming (NDP) (Bertsekas and Tsitsiklis, 1996)

RL methods typically assume a mathematical formulation based on MDPs. So, at least from an abstract point of view, RL and AI planning both attempt to address the same class of decision-making problems. Nevertheless, the way those areas of research find effective solutions in the algorithmic design space are rather different. RL is mostly concerned with a set of approximations, either in value space or in policy space. The former gives rise to *approximate dynamic programming* methods and the latter to *policy optimization* techniques. Of course, it is also possible to combine ideas from both approaches, either in a fully model-free or model-learned setting. For the remainder of this section, we review the relevant works for these classes of algorithms.

Approximate Dynamic Programming

The most successful algorithm in ADP is certainly Watkin's *Q-learning* (Watkins and Dayan, 1992). The key observation is that the Bellman optimality (Equation 2.2.1) can be reformulated in terms of the Q-function to a form that is amenable to sampling-based estimation. In addition, once a nearly-optimal approximation of Q^* is found, the optimal action in a given state can be directly obtained in a complete model-free fashion.

Definition 2.2.4. (Optimal Q-function) *Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \rho, T, R, \gamma)$ be a discrete-time MDP. Then, the optimal Q-function is given by:*

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim T(s,a)} \left[\max_{a' \in \mathcal{A}(s')} Q^*(s',a') \right] . \tag{2.6}$$

Theorem 2.2.4. (Q-learning) *Let $Q_0 \in \mathbb{R}^{|S| \times |A|}$ be any initial estimate of Q^*. Then, under the Robbins & Monroe stochastic approximation conditions (Robbins and Monro, 1951) for the learning rate α, the following iterative algorithm converges to the optimal Q^*:*

$$Q_{i+1}(s,a) \leftarrow (1-\alpha) Q_i(s,a) + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}(s')} Q_i(s',a') \right) , \tag{2.7}$$

if $s' \sim T(s,a)$ and $r = R(s,a)$ and assuming all state-action pairs (s,a) are visited infinitely often.

Other important ADP algorithms are based on the method of *Temporal Differences* (TD)[4] (Sutton, 1988). Early on, approaches based on the combination of TD and neural networks used as function approximators had a tremendous impact in board games, e.g., TD-Gammon (Tesauro, 1995). In recent years, a resurgence of interest in neural network-based methods in the area of ADP has been seen due to the advancement of Deep Learning (Goodfellow *et al.*, 2016) and the increase in compute power. Algorithms such as NFQ for discrete action spaces (Riedmiller, 2005) and NFQA for continuous action spaces (Hafner and Riedmiller, 2011) paved the way to more modern works in ADP, e.g., DQN (Mnih *et al.*, 2013), Rainbow (Hessel *et al.*, 2018).

[4]For the interested reader, an extensive survey of TD methods can be found in Sutton's influential book on RL (Sutton and Barto, 2018).

Policy Optimization

In contrast to ADP methods learn an approximation to the optimal Q-function as a proxy for finding an optimal policy, methods based on policy optimization attempts to directly optimize the final solution of the decision-making problem (i.e., a policy) given previously-generated experiences. This is typically achieved by considering a parametric policy π_θ and defining an objective function $J(\theta)$ that can be efficiently optimized.

Policy optimization is typically performed either by 0th-order methods (e.g., Cross-Entropy Method (CEM) or evolutionary methods (Hansen, 2006; Salimans *et al.*, 2017)) or 1st-order methods (e.g., REINFORCE (Williams, 1992), Policy Gradients (Barto *et al.*, 1983, 2021; Sutton *et al.*, 1999)). In the context of model-free RL, policy optimization is sometimes known as *Policy Search* (Deisenroth *et al.*, 2013).

Theorem 2.2.5. (Policy Gradients) *Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \rho, T, R, \gamma)$ be a discrete-time MDP. In addition, let π_θ be a parametric policy and $J(\theta) = \mathbb{E}_{S\sim\rho}[V^{\pi_\theta}(S)]$ be an objective function. If π_θ and $J(\theta)$ are differentiable w.r.t. the parameters θ, then:*

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_\theta}}\left[\mathbb{E}_{a\sim\pi_\theta}[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)]\right] \ , \tag{2.8}$$

where $\rho^{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s)$ is the (unnormalized) state visitation distribution of the policy π_θ in the given MDP.

Actor-Critic (AC) methods (Konda and Tsitsiklis, 1999) combine policy optimization with value function learning as a way to reduce the variance of policy gradient estimators. By first learning an approximation of the value function, AC approaches attempt to follow the direction of improvement in the space of policy parameters using the ascent direction of the learned value function as a surrogate. Recent on-policy AC methods include algorithms based on natural gradients, e.g., NPG (Kakade, 2001), and trust-region optimization, e.g., TRPO (Schulman *et al.*, 2015b), PPO (Schulman *et al.*, 2017). Off-policy AC methods have also been derived, e.g., DDPG (Lillicrap *et al.*, 2016), TD3 (Fujimoto *et al.*, 2018). Among these methods, Soft Actor-Critic (SAC) (Haarnoja *et al.*, 2018a,b) deserves a special mention. It reformulates the original problem of expected return maximization to incorporate policy entropy regularization as a mechanism to add stability and robustness to the policy optimization.

Model-based Reinforcement Learning

Model-based RL integrates planning capabilities into the overall learning approach. In general, these model-based methods explicitly learn predictive models of the environment's dynamic and/or reward functions. In this regard, the learned models can be used (*i*) to generate additional experience (a.k.a. data augmentation), (*ii*) to assist in the optimization formulations used during learning, or (*iii*) to strengthen an offline-learned policy during execution time.

Dyna (Sutton, 1990, 1991) is among the first algorithms in RL capable of some form of rudimentary planning. Dyna and other related extensions such as PILCO (Deisenroth and Rasmussen, 2011) and MBPO (Janner *et al.*, 2019) learn a model of the system through *Maximum Likelihood Estimation* (MLE) from transition data previously generated by the agent's interactions. Their use of the model is simply to generate imaginary/virtual data to be processed by model-free RL

algorithms. Dyna proposes to use a lookup table as a non-parametric model. PILCO also uses a non-parametric model, albeit a much more powerful one based on Gaussian Processes (GP). MBPO uses an ensemble of neural networks to approximate the dynamics of the environment and proposes to use the learned model to sample short rollouts branched from real trajectories as a way to accelerate exploration and learning.

Other methods attempt to use MLE-learned models to incorporate multi-step targets into value learning and policy optimization objectives, e.g., MVE (Feinberg *et al.*, 2018), STEVE (Buckman *et al.*, 2018), PETS (Chua *et al.*, 2018). Of particular importance to this thesis are the works based on the idea of *value gradients*. Recent approaches such as Stochastic Value Gradients (SVG) (Heess *et al.*, 2015), Model-Augmented Actor-Critic (MAAC) (Clavera *et al.*, 2020), Gradient-Aware Model-based Policy Search (GAMPS) (D'Oro *et al.*, 2020), and Model-based Action-Gradient Estimator (MAGE) (D'Oro and Jaskowski, 2020) are closely related to the general theme of this thesis. They all rely on differentiable models and leverage stochastic gradients through the dynamics to obtain value functions approximations of some sort. Guided Policy Search (GPS) (Levine and Abbeel, 2014; Levine and Koltun, 2013) is somewhat similar in their reliance on differentiable models, but it adapts methods from trajectory optimization in optimal control to find a near optimal policy.

Previously discussed methods are all based on learned models that are specifically grounded in the MDP in the sense that their predictions of state and rewards make sense under the dynamics of the environment. Indeed, this is the natural way to think about learning models. However, the representations learned via MLE does not necessarily carry the most relevant information for the subsequent use of models during planning. This is most clear in tasks for which typical observations are based on raw data, e.g., pixels. In those cases, minimizing prediction errors of visual features that are merely aesthetic only makes the learning process inefficient.

To address this limitation, recent research on learning latent space models attempt to compress the state representation used in the learned models in a way that *plannable representations* naturally emerge. The particular details of how the different latent-space model-based RL methods define the learning objective vary wildly. For instance, seminal work on *World Models* (Ha and Schmidhuber, 2018) make use of Variational Auto-Encoders (VAEs) and unsupervised learning to obtain compressed spatial and temporal representation of the environment. Other works learn internal representations that either help to predict future rewards, e.g., Predictron (Silver *et al.*, 2017b), or makes the planning invariant to goal selection, e.g., UPN (Srinivas *et al.*, 2018)), or even introduce inductive bias to the architecture of the models based on properties of the MDP or classical planning algorithms, e.g., VIN (Tamar *et al.*, 2016), TreeQN/ATreeC (Farquhar *et al.*, 2018), E2C (Watter *et al.*, 2015). Some more ambitious approaches attempt to directly embed the entire MDP in the latent space using recurrent models and dynamic unrolling, e.g., DeepMDP (Gelada *et al.*, 2019), PlaNet, DreamerV2 (Hafner *et al.*, 2019, 2020). A very important class of model-based RL that has achieved tremendous success in practice are those based on neurally-guided MCTS. AlphaGo (Silver *et al.*, 2016) and its extension AlphaZero (Silver *et al.*, 2017a) leverage Monte-Carlo search over a known model aided by value and policy networks to focus exploration. MuZero (Schrittwieser *et al.*, 2020) extends its predecessors by learning a compressed latent state model which allowed the algorithm to achieve super-human level not only at board games such as chess, Shogi, and Go, but also in pixel-based Atari games. Finally, various works also use the learned model at decision time using *Model Predictive Control* (MPC) to make the policy learned more robust to certain types of uncertain

events never seen during training, e.g., (Nagabandi *et al.*, 2018).

For a more in-depth discussion about model-based RL, we recommend the interested reader to consult the recent surveys (Moerland *et al.*, 2020b; Plaat *et al.*, 2020; Polydoros and Nalpantidis, 2017; Wang *et al.*, 2019).

2.2.3 Optimal Control

Control is the science of real-time decision making in engineering. Reliability, robustness, and safety bounds are the hallmarks of control theory. The standard control model consists of (i) an input-output process that an agent aims to control; (ii) an observer for estimating state from noisy observations; (iii) a controller deciding the control actions to execute in the system. The goal of the controller is to steer the system towards regions of well-behaved states as defined in a task description.[5] In classical control theory, the task description typically consists of several (possibly conflicting) requirements that together specify the desired transient and steady-state properties of the system (e.g., rise time, overshoot, setting time in step response, magnitude and phase for sinusoids in frequency response) as well as safety and liveness constraints that the controller must satisfy.

In optimal control, the high-level task requirements are rather specified through simple cost functions that associate an scalar signal for each state-control pair with the goal of incentivizing a particular behavior to emerge. The immediate consequence of designing cost functions is that a natural notion of optimality is defined. Among other benefits, the optimality criterion induced by cost functions allows to easily compare different controllers by virtue of comparing a single numeric value.

The traditional view in control is to derive a controller based on *reactive compensation* where state feedback is employed to provide stability, performance, and robustness in tasks involving disturbances and other uncertainties. This is typically used to solve regulation or tracking problems for which a nominal steady-state or trajectory is provided as part of the task specification. The modern view of control, however, revolves around the concept of *predictive compensation* where a model is used to derive reference trajectories for low-level controllers to regulate.

In deterministic systems, the dynamics in continuous time are modelled by ODEs (Ordinary Differential Equations) and PDEs (Partial Differential Equations) with task specifications given by cost functions defined over time- and/or space- dependent properties of the system. For example, the system dynamics is modelled by a set of structural equations, either in continuous time, $\dot{x}(t) = f(x, u)$, or in discrete time, $x_{k+1} = f(x_k, u_k)$, where x and u denote the state and action variables, respectively. The goal of the controller is defined in terms of the minimization of a total cost function subject to a set of constraints over the states and control actions. In this thesis, we restrict ourselves to discrete-time control problems involving continuous states and actions.

Definition 2.2.5. (DCP) *A* Deterministic Control Problem *is defined by the tuple* $\mathcal{M}_{DCP} = (\mathcal{X}, \mathcal{U}, f, g, x_0)$*, where* $\mathcal{X} \in \mathbb{R}^n$ *is the state space,* $\mathcal{U} \in \mathbb{R}^m$ *is the input space,* $f \colon \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ *is the dynamics function,* $g \colon \mathcal{X} \times \mathcal{U} \to \mathbb{R}_+$ *is the cost function, and* $x_0 \in \mathcal{X}$ *is the start state.*

[5] In AI planning terms, the system is the domain specified by a transition function, the task is specified by the reward/cost function (and possibly hard-constraints) and the environment refers to a specific instance/problem with all its particular parameters.

In probabilistic systems, dynamics models can be defined using stochastic ODEs, Kolmogorov equations, or Markov Decision Processes, with task specifications defined over expected values and other distribution moments. In addition, unmodeled aspects of the system's dynamics can be bounded in a robust setting.

Definition 2.2.6. (SCP) *A* Stochastic Control Problem *is given by* $\mathcal{M}_{SCP} = (\mathcal{X}, \mathcal{U}, \Omega, p, f, g, x_0)$, *where* $\mathcal{X} \in \mathbb{R}^n$ *is the state space,* $\mathcal{U} \in \mathbb{R}^m$ *is the input space,* Ω *is the space of random disturbances,* $p \in \mathcal{P}(\Omega)$ *is the noise distribution,* $f \colon \mathcal{X} \times \mathcal{U} \times \Omega \to \mathcal{X}$ *is the dynamics function,* $g \colon \mathcal{X} \times \mathcal{U} \to \mathbb{R}_+$ *is the cost function, and* $x_0 \in \mathcal{X}$ *is the start state.*

From the viewpoint of solutions, optimal control approaches typically fall into one of two broad categories: *indirect* and *direct* methods. In indirect methods, the necessary and sufficient conditions of optimality (i.e., Karush–Kuhn–Tucker (KKT) conditions) are first written down for the original problem in continuous time and only then discretized for numeric resolution. On the other hand, direct methods[6] discretize the time before formulating an optimization problem whose solution is afterwards interpolated back to continuous time.

Trajectory Optimization and *Model Predictive Control* (MPC) are the main methods used to solve control problems that need long-horizon planning capabilities. In the remainder of this section, we review related problems and control algorithms that leverage dynamic programming in the context of trajectory optimization and MPC methods for online control.

Linear-Quadratic Regulator

Linear Quadratic Regulation (LQR) problems assume a linear transition and a quadratic cost function. Though the LQR model may seem too simplistic at first glance, there is actually a large class of decision-making problems that can be either exactly modeled or at least approximated by LQR formulations.

Definition 2.2.7. (LQR) *A* Linear Quadratic Problem *is a DCP where:*

- *the* dynamics function *is given by* $x_{t+1} = f(x_t, u_t) = Ax_t + Bu_t$, *where* A *and* B *are matrices of appropriate dimensions; and*
- *the* cost function *is given by* $g(x_t, u_t) = x_t^\top Q x_t + u_t^\top R u_t$, *where* Q *and* R *are positive semi-definite matrices of appropriate dimensions.*

Definition 2.2.8. (LQG) *A* Linear Quadratic Gaussian *is an SCP where:*

- *the* dynamics function *is given by* $x_{t+1} = f(x_t, u_t) = Ax_t + Bu_t + w_t$, *where* A *and* B *are matrices of appropriate dimensions and* $w_t \sim \mathcal{N}(0, \Sigma)$ *is a 0-mean Gaussian noise; and*
- *the* cost function *is given by* $g(x_t, u_t) = x_t^\top Q x_t + u_t^\top R u_t$, *where* Q *and* R *are positive semi-definite matrices of appropriate dimensions.*

[6] In this thesis, we will be exclusively interested in direct methods that bear a number of important connections with the underlying theme of planning through backpropagation.

Dynamic programming can be easily applied to LQR problems due to the convex minimization problem resulting from the interplay of the quadratic cost function and the linear dynamics. At each time step, the Bellman optimality equations can be solved analytically. The solution is a linear controller, $u_t = \pi(x_t) = -K_t x_t$, for which the gain matrix K can be efficiently computed by solving the *Riccati* equations. The same set of equations can be used to solve LGQ tasks as any 0-mean disturbances does not affect in expectation the underlying cost minimization problem. This is an exact application of the *Certainty-Equivalence Principle* (CEP).

Theorem 2.2.6. (Riccati Equations) *Let $\mathcal{M} = (A, B, Q, R)$ be a DCP as described in Definition 2.2.7. The optimal controller for $\mathcal{M}$, $\pi(x_t) = -K_t x_t$, has matrix gain K_t given by the solution of the following discrete Riccati equations:*

$$K_t = (R + B^\top V_{t+1} B)^{-1} B^\top V_{t+1} A \ , \tag{2.9}$$

$$V_t = Q + A^\top V_{t+1} A - (A^\top V_{t+1} B)(R + B^\top V_{t+1} B)^{-1}(B^\top V_{t+1} A) \ . \tag{2.10}$$

Differential Dynamic Programming

Differential Dynamic Programming (DDP) attempts to solve nonlinear tasks by first approximating the control problem and then applying the LQR algorithm, iteratively. At each optimization epoch an LQR problem is constructed by local approximations around a candidate trajectory. Typically, transitions are linearized and cost functions are quadraticized, obtained through first and/or second order Taylor approximations which require access to a differentiable model.

The seminal work in DDP was proposed by Jacobson and Mayne (1970). Important extensions have been developed to deal with box-constrained action spaces (Tassa *et al.*, 2007; Theodorou *et al.*, 2010) and stochastic transitions with Gaussian noise (Li and Todorov, 2004; Roulet *et al.*, 2019; Tassa *et al.*, 2014; Todorov and Li, 2005). DDP performs local optimization, which calls for the need of cost shaping in domains with many local optima. Therefore, During problem approximation, regularization techniques must be employed to prevent divergence, thus leading to a difficult trade-off between stability and speed of convergence.

Model Predictive Control

Model Predictive Control (MPC) (Camacho and Alba, 2013) is a broad class of model-based online methods widely used in industry. It aims to obtain the current control action by minimizing a surrogate cost function defined over a finite-time horizon. Typically, it uses some a form of problem approximation that is amenable to online solutions via mathematical programming. For instance, it is usual to leverage the certainty-equivalence principle to derive nominal deterministic trajectories with future disturbances and other uncertain quantities sampled beforehand and assumed to be fully known during optimization.

The solution of the online approximated problem is a sequence of actions from which only the first action is executed in the system; the other remaining actions are discarded. Then, the system evolves to the next state and the procedure restarts anew. This online approach can be viewed as an open-loop optimal control formulation in the sense that information about future uncertainties are

Algorithm 1: Differential Dynamic Programming

Input: DCP $\mathcal{M} = (\mathcal{X}, \mathcal{U}, f, g, x_0)$, horizon H
Output: a sub-optimal plan $\hat{\mathbf{u}}_{0:H-1}$

$\hat{\mathbf{u}}_{0:H-1} \sim p(\mathcal{U}^H)$; ▷ Sample a valid initial plan
$\hat{\mathbf{x}} \leftarrow ()$; ▷ Initialize empty trajectory
▷ Generate an initial trajectory
for $t = 0, \cdots, H-1$ **do**
  $\hat{x}_{t+1} = f(\hat{x}_t, \hat{u}_t)$
  $\hat{\mathbf{x}} \leftarrow (\mathbf{x}, u_t, x_{t+1})$
while *not converged* **do**
  ▷ Model approximation
  for $t = 0, \cdots, H-1$ **do**
    $\tilde{A}_t, \tilde{B}_t \leftarrow \nabla_{x,u} f(x,u)|_{x=x_t, u=u_t}$; ▷ First-order dynamics approx.
    $\tilde{Q}_t, \tilde{R}_t \leftarrow \nabla^2_{x,u} g(x,u)|_{x=x_t, u=u_t}$; ▷ Second-order cost approx.
  ▷ Backward pass
  $\mathbf{K}_{0:H-1} \leftarrow \textsc{Ricatti}(\tilde{\mathbf{A}}_{0:H-1}, \tilde{\mathbf{B}}_{0:H-1}, \tilde{\mathbf{Q}}_{0:H-1}, \tilde{\mathbf{R}}_{0:H-1})$; ▷ LQR controller
  ▷ Forward pass
  $\hat{x}_0 \leftarrow x_0$
  for $t = 0, \cdots, H-1$ **do**
    $u_t \leftarrow \hat{u}_t + K_t(x_t - \hat{x}_t)$
    $x_{t+1} \leftarrow f(x_t, u_t)$; ▷ Simulation
    $\hat{u}_t, \hat{x}_{t+1} \leftarrow u_t, x_{t+1}$
return $\hat{\mathbf{u}}_{0:H-1}$

incrementally revealed and not fully internalized by a global policy. It is the interleaving of planning, execution, and re-planning that attempts to cope with the progressive unrolling of uncertainty. In addition, this online setting forces MPC approaches, in particular in real-time decision-making, to use simple yet accurate models. However, such properties do not usually correlate well in complex nonlinear large-scale applications which makes the solutions obtained via MPC highly sensitive to the problem approximation used.

Nevertheless, a very useful aspect of MPC methods based on mathematical programming is that in principle it can easily incorporate important classes of state and action constraints, e.g., input saturation, maximum power consumption, budget limits, and other state safety constraints, including important finite-horizon terminal constraints.

2.3 Summary of Related Work

In this chapter, we presented an overall view of models and algorithms for sequential decision-making in AI and optimal control in engineering. In this section, we classify the algorithms previously surveyed along the dimensions of: (1) model-based or model-free, (2) space discretization, (3) nonlinear dynamics, (4) stochastic events, and (5) algorithmic scalability. Table 2.1 compares the classes of heuristic search in MDPs, hybrid symbolic dynamic programming, MILP-based planning (Mixed Integer Linear Programming), Monte-Carlo planning, metric planning, model-free RL, differential dynamic programming in optimal control, and differentiable planning.

Model-based / model-free. With the exception of model-free RL algorithms, most methods considered here assume access a model of the environment, i.e., its dynamics and cost functions. We refrain to list all the methods from the model-based RL literature as they are vastly diverse, but most are extensions of Approximate Dynamic Programming or Policy Gradients, and therefore can be classified much as their model-free counterparts. MILP-based planning does not necessarily depend on a model, but needs to obtain data from somewhere (e.g., a simulator, another model, logged data, etc).

Space discretization. In general, methods that are originally developed for discrete spaces such as heuristic search, numeric and metric planning, and Monte-Carlo planning are not directly applicable to continuous action spaces. Hence, they require an offline (or an adaptive) step of discretization.

Nonlinearities. Most methods can cope with some form of nonlinearities. In particular, simulation and search methods are in principle agnostic to the dynamics after action discretization. However, the degree of non-linearity is likely to have an impact on the granularity needed in the action discretization to obtain good performance, i.e., the more nonlinear a problem is, more fine-grained the discretization mesh needs to be. Additionally, some methods in numeric and metric planning and hybrid symbolic dynamic programming can only afford specific functional nonlinearities. On the other hand, differential dynamic programming can handle nonlinear domains, but first needs to linearize the dynamics and quadratize the cost functions around a given trajectory. This process is somewhat involved as sensitive regularizations have to be applied to maintain the convexity of the approximated cost function and heuristic line search methods have to be used to trade-off computational time and performance. Differentiable planning methods can cope with various forms of nonlinearities

Stochastic events. Most methods can handle some form of stochastic events, except for MILP-base planning. The most general class of algorithms from the viewpoint of stochasticity in the model are the ones based on simulations, either Monte-Carlo planning or model-free RL methods, as those methods only require samples. We remark that differential dynamic programming can only support in its original formulation Gaussian noise (or at least 0-mean additive random variables) in the stochastic transition function. Additionally, differentiable planning methods for Deep Reactive Policies (as presented in this thesis) can only cope with continuous stochastic variables that can be re-parametrized (as defined in Section 3.2.2).

Scalability. We consider as non-scalable all the works that depend on discretization, with the exception of Monte-Carlo Planning + Adaptive Sampling that can in theory manage the exponential explosion in representation due to the discretization (even though at the cost of expensive trial-and-error to tune the adaptive sampling schemes). Conversely, we consider as scalable methods those that do not rely on discretization, with the caveat that for Policy Gradients the higher the number of action variables, the more sample inefficient the method becomes, even though the computational cost of estimating the policy gradient remains linear in the dimension of the action space. Finally, we remark that symbolic dynamic programming can solve huge problems, but its scalability depends on the order of the variables in the underlying decision diagrams, i.e., some order may lead to

very compact representations and other orders may very quickly consume all the available memory. Thus, we classify such methods to exhibit limited scalability.

As per our survey of the literature presented in this chapter, we conclude that differentiable planning methods are uniquely positioned to flexibly address continuous planning problems with exogenous stochastic events without the need of discretization or other problem-dependent approximations.

Table 2.1: *Comparison of sequential decision-making algorithms in AI and optimal control in Engineering*

Approaches	Model	Discretization	Nonlinearities	Stochastic events	Scalability
Heuristic Search Bonet and Geffner (2003) Hansen and Zilberstein (2001)	Model-based	State + Action	Yes	Discrete	No
Hybrid Symbolic Dynamic Programming Sanner *et al.* (2011) Zamani *et al.* (2012) Vianna *et al.* (2015)	Model-based	No	Limited	Discrete	Limited
MILP-based Planning Say *et al.* (2017)	Data-driven	No	Yes	No	Limited
Monte-Carlo Planning Kocsis and Szepesvári (2006) Keller and Eyerich (2012)	Model-based	Action	Yes	Discrete / Continuous	No
Monte-Carlo Planning + Adaptive Sampling Chaslot *et al.* (2007) Mansley *et al.* (2011)	Model-based	Action	Yes	Discrete / Continuous	Yes*
Numeric and Metric Planning Hoffmann (2011) Benton *et al.* (2012) Piotrowski *et al.* (2016)	Model-based	Time / Action	Limited	No	No
Approximate Dynamic Programming (RL) Watkins and Dayan (1992)	Model-free	Action	Yes	Discrete / Continuous	No
Policy Gradients (RL) Sutton *et al.* (1999)	Model-free	No	Yes	Discrete / Continuous	Yes*
Differential Dynamic Programming Li and Todorov (2004) Todorov and Li (2005)	Model-based	No	Yes*	Continuous (Gaussian noise)	Limited
Differentiable Planning Wu *et al.* (2017)	Model-based	No	Yes	No	Yes
Bueno *et al.* (2019)	Model-based	No	Yes	Continuous*	Yes

Chapter 3

Stochastic Computation Graphs

In this chapter, we introduce the formalism of *Stochastic Computation Graphs* (SCG) (Schulman, 2016; Schulman *et al.*, 2015a) which will form the mathematical foundation for applying automatic differentiation to stochastic models. Informally speaking, SCGs specify parametric models that mix deterministic computations with random variables drawn from distributions that may depend on the results of previous computations. In this context, SCGs have been used to represent a growing number of recently-proposed parametric stochastic models developed in machine learning, notably, probabilistic models with latent variables (Kingma and Welling, 2014; Mnih and Gregor, 2014; Rezende *et al.*, 2014) and reinforcement learning (Guez *et al.*, 2018; Sutton *et al.*, 1999; Williams, 1992; Zaremba and Sutskever, 2015).

Given that such stochastic models attempt to capture uncertainty with probabilistic distributions, they typically attempt to obtain the best set of parameters by optimizing an objective function that depend on a complex expectation over its random variables, which rarely can be known in an analytically tractable form. This difficulty in obtaining a closed-form, analytical objective function precludes the direct application of gradient descent to optimize the model, given that error backpropagation (Rumelhart *et al.*, 1986), and more generally automatic differentiation (Griewank *et al.*, 1989), can only be applied to known deterministic and differentiable functions. The formalism of SCGs aim to tackle this issue by specifying the necessary conditions to obtain sound and efficient gradient estimators for the class of differentiable stochastic models, thus allowing gradient-based optimization to be used to optimize its parameters.

In summary, SCGs can be interpreted as both a *language* to define the dependencies between the variables of an acyclic generative stochastic model with the intent to reason over the applicability of gradient estimators and as a *data structure* that allows the development of efficient sampling and automatic differentiation procedures that are required for all gradient-based optimization schemes. Even though the ultimate goal of this thesis is to extend differentiable planning to stochastic models, we remark that SCGs underlie a vast class of models in related fields in machine learning and AI. For this reason, we shall present in this chapter the theory of SCGs in general terms and defer details specific to differentiable planning to the following chapters.

This chapter is organized as follows. First, we lay out important definitions and the main theorem of gradient estimation in SCGs. Then, we present and discuss the differences between the two broad classes of gradient estimators that can be obtained for differentiable stochastic models. Finally, we conclude the chapter with an intuitive introduction to reverse-mode automatic differentiation and present a few illustrative examples.

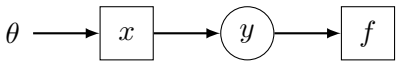
3.1 Definitions

The modern view of stochastic computation graphs as extensions of computation graphs for stochastic models was recently proposed by Schulman *et al.* (2015a). In this section, we adapt important definitions and notations from this seminal work with the ultimate goal of presenting the necessary conditions to obtain unbiased gradient estimators of a problem-dependent objective function w.r.t. to its inputs which are to be understood as the model parameters to be optimized.

We first start by defining stochastic computation graphs in Definition 3.1.1.

Definition 3.1.1. (SCG) *A* stochastic computation graph $\mathcal{G} = (V, E)$ *is a directed, acyclic graph defined over three disjoint set of nodes:* (i) input nodes I*, directly observed or externally set beforehand;* (ii) deterministic nodes D*, corresponding to functions of its parent nodes; and* (iii) stochastic nodes S*, representing random variables conditionally distributed accordingly to a function whose parameters depend on its parent nodes. The set of nodes* V *is partitioned as* $V = I \cup D \cup S$*. An edge* $(u, v) \in E$ *if node* v *or its probability distribution depends on node* u*.*

A first example of an SCG is outlined in Figure 3.1 illustrating a function f that depends on the Gaussian random variable y whose mean parameter depend on the the value of variable x which is itself a function of the input parameter θ. This is a simple example of an SCG. Additional examples will be discussed in Section 3.4 and many more examples appearing in differentiable planning algorithms will be presented in details in the remainder of the text. Throughout this thesis, we follow the graphical notation of Schulman *et al.* (2015a) in which *squared* nodes depict deterministic dependencies, *rounded* nodes represent stochastic nodes, and input nodes are root nodes depicted with no border. Input nodes can be usually interpreted as parameter nodes or other values conditioned a priori on the model.



$$x(\theta) = \text{sigmoid}(\theta) = \frac{1}{1 + e^{-\theta}} \tag{3.1}$$

$$y \sim \mathcal{N}(\mu = x + 0.5, \sigma = 0.1) \tag{3.2}$$

$$f(y) = y(y - 2) \tag{3.3}$$

Figure 3.1: *Stochastic computation graph example: (left) the graphical representation of an SCG with a single path from input node to cost node; (right) the functional equations and probabilistic distribution for each node in the SCG. Squared nodes* x *and* f *are deterministic nodes and the rounded node* y *is a stochastic node. The node* θ *is the single input node represented without border.*

Once an SCG is specified[1], graph algorithms can explore its graphical structure to infer the deterministic and probabilistic dependency paths as formalized in Definition 3.1.2. As we will see shortly, it is essential to obtain these paths to define which gradient estimator can be used for each part of the stochastic model. Additionally, we remark that visiting such paths in an efficient way is of practical importance to derive automatic differentiation algorithms that remain linear in the size of the computation graph (Griewank and Walther, 2008). In the Example 3.1, we observe that there is a single path from parameter θ to the leaf node f and this path is mediated by the random variable y. Therefore, there is no deterministic path joining nodes f and θ, i.e., there is no explicit deterministic functional dependencies between them. The direct consequence is that the

[1] At this point the presentation of SCGs might seem abstract at a first glance, but we remind the reader that SCGs are concrete representations of computations and form the underlying data structure used in automatic differentiation software, e.g., TensorFlow (Abadi *et al.*, 2016).

well-known backpropagation algorithm cannot be applied directly in this computation graph to obtain the gradient $\nabla_\theta f$.

Definition 3.1.2. (Probabilistic and Deterministic Paths) *Let v and w be nodes of a stochastic computation graph $\mathcal{G}$. Then, we denote by $v \prec w$ the property that it exists a dependency path from node v to node w in $\mathcal{G}$. Additionally, we denote by $v \prec^D w$ the property that it exists a dependency path from node v to node w traversing only deterministic nodes.*

From the exclusive point of view of sampling the output of all nodes in the graph given the input nodes, the previous definitions completely define the underlying semantics of simulations in SCGs. However, in order to differentiate the model to obtain gradients, we need to establish the differentiability requirements for an SCG. Definition 3.1.3 outlines the necessary conditions to extend the semantics of SCGs to represent differentiable stochastic models.

Definition 3.1.3. (Differentiable SCG) *Let* $\textsc{Parents}(w)$ *denote the set of parent nodes of the node w. An SCG $\mathcal{G}$ is said to be differentiable w.r.t. node θ if, for all edges (v, w) satisfying $\theta \prec^D v$ and $\theta \prec^D w$, it holds that:*

1. *if $w \in D$, then the Jacobian $\frac{\partial w}{\partial v}$ exists; or*
2. *if $w \in S$, then the gradient of the probability density function $\frac{\partial}{\partial v} p(w|\textsc{Parents}(w))$ exists or $p(w|\textsc{Parents}(w))$ is a re-parametrizable probability density function as defined in Section 3.2.2.*

In addition, an SCG is said to be differentiable if it is differentiable w.r.t. to all input nodes, $\theta \in I$.

If an SCG representing a parametric stochastic model is differentiable w.r.t. its input nodes θ (i.e., the model parameters), then gradient descent methods can be applied to optimize an objective function, $J(\theta)$, defining how good or bad the represented model is w.r.t. its current parametrization. We remark once again that in machine learning (and also in our context of differentiable planning in following chapters), it is often the case that an expectation of a loss function or metric of interest is used as the objective function (or at least part of it). In this context, we notice that the only requirement for obtaining unbiased gradients is that the composition of the objective function with the remaining nodes of the SCG remains differentiable. In Definition 3.1.4, we introduce the notion of cost nodes and define the notation of the objective function used in the remainder of this chapter.

Definition 3.1.4. (Objective function) *The* objective function $J \colon \Theta \to \mathbb{R}$ *of an SCG is denoted by the expectation of the* total cost *taken w.r.t. all stochastic nodes:*

$$J(\theta) = \mathbb{E}_{w \in S} \left[\sum_{c \in C} c(\theta) \right], \tag{3.4}$$

where Θ is the set of input parameters, $c \in C \subseteq D$ are scalar-valued leaf nodes known as cost nodes.

The final concept we need before stating the main theorem in the theory of SCGs is the notion of *cost-to-go* from a particular node. Recall that an SCG might contain several stochastic nodes representing random variables. From the viewpoint of a given a node w, a particular realization

of those random variables will induce a particular realization of all descendants of w, including all dependent cost nodes. The total sum of the values of these cost nodes are referred to as the cost-to-go of node w. Definition 3.1.5 formalizes this notion.

Definition 3.1.5. (Cost-to-Go) *The* cost-to-go $\hat{Q}_w$ *from node* $w \in V$ *(also known as* downstream total cost*) is defined by:*

$$\hat{Q}_w = \sum_{\hat{c} \in C, w \prec \hat{c}} \hat{c}. \tag{3.5}$$

Remark. *In general, we will use the hat symbol, for instance* $\hat{v}$*, to represent a given realization of a random variable* v*, which shall be treated as constant in gradient and expectation formulas.*

Finally, we are now able to present the *Gradient Estimation Theorem* in SCGs. This is the most important result of the theory of SCGs. It is a formidable result in the sense that it provides the general form of the gradient of any differentiable SCG regardless of its specific graphical structure and functional dependencies. In principle, the procedure to obtain this general gradient estimator for any given SCG could be completely automatized in modern automatic differentiation software. Unfortunately, this has not been implemented by the time of the writing of this thesis in any mainstream library, except for very specific cases. Consequently, it becomes the user's responsibility to correctly implement (as another computation graph) the correct gradient estimator, and for this, we remark it is fundamental to understand the implications of the structure of the underlying SCG from the point of view of Theorem 3.1.1.

Theorem 3.1.1. (Gradient Estimation)

Let $\theta \in \Theta$ *be the set of parameters of a stochastic model and* $J(\theta)$ *be the objective function to be optimized. If the SCG representing the objective function composed over the computations of the model is differentiable, then the* gradient of the objective function *w.r.t. input parameter* θ *is given by:*

$$\nabla_\theta J(\theta) = \mathbb{E}_{w \in S}\left[\sum_{\substack{w \in S, \\ \theta \prec^D w}} \nabla_\theta \log p(w | \textsc{Parents}(w)) \hat{Q}_w + \nabla_\theta \sum_{\substack{c \in C, \\ \theta \prec^D c}} c(\theta) \right], \tag{3.6}$$

where $\hat{Q}_w = \sum_{\hat{c} \in C, w \prec \hat{c}} \hat{c}$ *is the* cost-to-go *from node* w*.*

Proof. See Appendix A of Schulman *et al.* (2015a). □

Note that the Theorem 3.1.1 transforms the gradient of an expectation, $\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}[\sum c(\theta)]$, into the expectation of a (log) gradient for all stochastic nodes. It is important to note that we cannot simply push the gradient operator inside the expectation because the stochastic nodes that the expectation is taking w.r.t. might depend on the parameters θ. In other words, the linearity of the gradient operation w.r.t. to the expectation does not apply in the general case of stochastic nodes depending on nodes θ.

The practical consequence of the transformation of Theorem 3.1.1 is that now we have access to a principled, and unbiased way to estimate the gradient ∇_θ by Monte-Carlo approximations of the expectation. In the next sections, we will discuss the implications of each gradient term inside the expectation in Theorem 3.1.1.

3.2 Gradient Estimators

There are two basic classes of gradient estimators in SCGs directly corresponding to the two terms inside the expectation in Theorem 3.1.1. First the term where the gradient of the log-probabilities are multiplied by the cost-to-go is generally referred as *likelihood ratio* or *score function* estimator whereas the second term is known as the *pathwise derivative* or the *re-parameterized gradient*. In this section, we present these types of gradient estimators and compare them on the simple example of Figure 3.1. For a comprehensive survey on the topic of gradient estimators in SCGs, we refer the interested reader to the work of Mohamed *et al.* (2020). Additionally, we make the observation that the formal proof of Theorem 3.1.1 is a somewhat generalizations of the derivations presented in Corollaries 3.2.1 and 3.2.2.

3.2.1 Likelihood Ratio Estimator

The *likelihood ratio estimator* (Glynn, 1986) is the most general form of gradient estimator in differentiable SCGs. It is applicable whenever the distributions of the stochastic nodes have differentiable probability density functions with respect to its parameters. In addition, its practical appeal comes from the fact that it does not require the downstream cost function to be differentiable. As a matter of fact, it only requires samples of the cost nodes, which makes it quite useful for model-free RL algorithms such as Policy Gradients (Barto *et al.*, 1983, 2021; Sutton *et al.*, 1999).

To illustrate why the likelihood ratio gradient has its name, consider the Corollary 3.2.1. The key observation is that this gradient estimator leverages the fact that $\nabla \log F = \frac{\nabla F}{F}$ for any differentiable function F, which is known as likelihood ratio trick for historical reasons (Mohamed *et al.*, 2020).

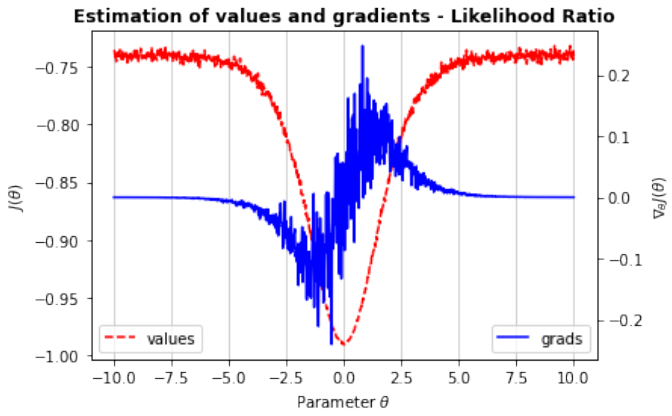
Corollary 3.2.1. *The score function gradient of the objective function of the SCG in Figure 3.1 is given by* $\nabla_\theta J = \mathbb{E}_y \left[\frac{\partial x}{\partial \theta} \frac{\partial}{\partial x} \log p(y|x) f(y) \right]$.

Proof.

$$
\begin{aligned}
\nabla_\theta J &= \nabla_\theta \mathbb{E}_{y \sim p(\cdot|x(\theta))} \left[f(y) \right] \\
&= \nabla_\theta \int_y p(y|x(\theta)) f(y) \, dy && \text{expectation defn} \\
&= \int_y \left[\nabla_\theta p(y|x(\theta)) \right] f(y) \, dy && \text{Leibniz integral rule} \\
&= \int_y \left[p(y|x(\theta)) \nabla_\theta \log p(y|x(\theta)) \right] f(y) \, dy && \text{likelihood ratio trick} \\
&= \int_y p(y|x(\theta)) \left[\nabla_\theta \log p(y|x(\theta)) f(z) \right] \, dy && \text{associativity} \\
&= \mathbb{E}_y \left[\nabla_\theta \log p(y|x(\theta)) f(y) \right] && \text{expectation defn} \\
&= \mathbb{E}_y \left[\frac{\partial x}{\partial \theta} \frac{\partial}{\partial x} \log p(y|x) f(y) \right] && \text{chain rule}
\end{aligned}
$$

□

Figure 3.2 shows a numerical example of the gradient estimates of the SCG of Figure 3.1. Both the value of the objective function $J(\theta)$ and its gradient $\nabla_\theta J(\theta)$ were computed via the empirical mean approximation $\mathbb{E}[X] \approx \frac{1}{N} \sum_{i=1}^{N} \hat{X}_i$, i.e., Monte-Carlo approximation. It is widely



$$\hat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N} f(y_i)$$

$$\nabla_\theta \hat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\partial x}{\partial \theta}\frac{\partial}{\partial x}\log p(y_i|x) f(y_i)\right]$$

Figure 3.2: *Likelihood Ratio (Score Function) estimator: values of the objective function and gradient estimates for the example of Figure 3.1. The plots were obtained via Monte-Carlo approximation with* $N = 1024$ *samples.*

known (Mohamed *et al.*, 2020; Tokui and Sato, 2017) that this gradient estimator can be very noisy depending of the variance of the random variables in the SCG. We notice that in this example we are only dealing with a single scalar random variable, thus one can easily imagine that in higher dimensions this noisy effect can become quite accentuated[2]. In summary, the high variance of this estimator can be seen as the price to pay for such a general gradient estimator. In the next section, we will see how to obtain better estimators provided that certain conditions on the SCG are met.

3.2.2 Pathwise Derivative

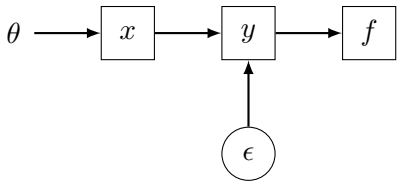
An effective way to circumvent the high variance in gradient estimation in SCGs is to exploit the probability density **re-parameterization trick** (Kingma and Welling, 2014)[3], which allows to sample a random variable $y \sim p_\theta(\cdot)$ by transforming it into a deterministic function $y = \phi(\theta, \xi)$ and using an independent marginal density to sample an auxiliary random variable $\xi \sim p(\cdot)$. For the location-scale family of distributions (e.g., Normal, Gamma, Uniform, Cauchy) the function $\phi(\theta, \xi)$ is given by $\phi(\theta, \xi) = \mu_\theta + \sigma_\theta \cdot \xi$, where μ_θ and σ_θ are functions defining the location and scale of probability density $p_\theta(\cdot)$.

The re-parameterization trick is the key ingredient to obtain the *pathwise derivative* gradient estimator (Schulman *et al.*, 2015a). The resulting effect of re-parameterizing the distributions to which the expectation is taken w.r.t. is that it becomes valid to push the gradient operator inside the expectation, and without resorting to the likelihood ratio trick. The new SCG obtained when the re-parametrization trick is applied to the example of Figure 3.1 and its corresponding equations is illustrated in Figure 3.3. Also, Corollary 3.2.2 shows the analytical derivation of the pathwise gradient estimator for this particular example.

Corollary 3.2.2. *The gradient of the objective function of the SCG in Figure 3.3 is given by* $\nabla_\theta J = \mathbb{E}_{\xi\sim p}[\nabla_\theta f(y(\theta, \xi))]$.

[2]This is one of the major reasons why policy gradients (which are based on the score function estimator) typically exhibit such a poor sample efficiency in practice.

[3]The reparametrization trick is an old technique used in statistics and therefore it is hard to pinpoint the seminal or the most known paper to cite. Here, we follow the current trend in machine learning to cite the VAE paper that recently put this technique in evidence in the community.



$$x(\theta) = \text{sigmoid}(\theta) = \frac{1}{1+e^{-\theta}} \tag{3.7}$$

$$\epsilon \sim \mathcal{N}(\mu = 0.0, \sigma = 1.0) \tag{3.8}$$

$$y = x + 0.5 + 0.1\epsilon \tag{3.9}$$

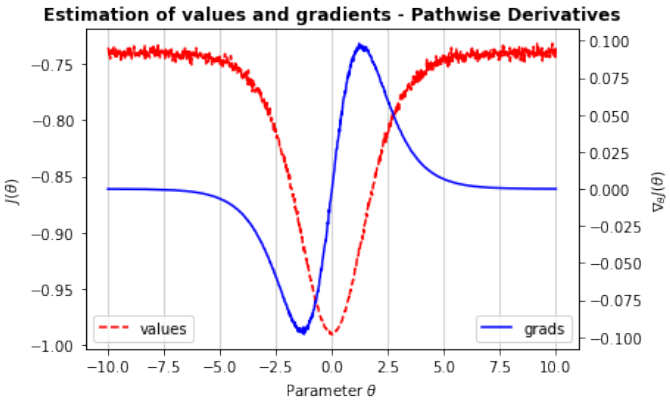
$$f(y) = y(y-2) \tag{3.10}$$

Figure 3.3: *Stochastic computation graph with a single path from input node to cost node*

Proof.

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{y\sim p_\theta}[f(y)] \\
&= \nabla_\theta \mathbb{E}_{\xi\sim p}[f(y(\theta,\xi))] && \text{re-parametrization trick} \\
&= \nabla_\theta \int p(\xi) f(y(\theta,\xi)) d\xi && \text{expectation defn} \\
&= \int p(\xi) \nabla_\theta f(y(\theta,\xi)) d\xi && \text{Leibniz integral rule} \\
&= \mathbb{E}_{\xi\sim p}[\nabla_\theta f(y(\theta,\xi))] && \text{expectation defn}
\end{aligned}$$

□



$$\hat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N} f(y_i)$$

$$\nabla_\theta \hat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\partial x}{\partial \theta}\frac{\partial y(x,\epsilon_i)}{\partial x}\frac{\partial f}{\partial y}\right]$$

Figure 3.4: *Pathwise Derivative (Re-Parametrization) estimator: objective function and gradient estimates for the example of Figure 3.3. The plots were obtained via Monte-Carlo approximation with $N = 1024$ samples.*

Figure 3.4 shows a numerical example for the SCG of the Figure 3.3. As before, both the value of the objective function $J(\theta)$ and its gradient $\nabla_\theta J(\theta)$ were obtained via Monte-Carlo approximation with the same $N = 1024$ samples. We cannot help but to notice that the noise decreased considerably, despite the fact that the same number of samples were used in the empirical mean. This example illustrates the fact that re-parametrization trick can give rise to more efficient gradient estimators in SCGs whenever we can have access to the functional form of the differentiable models.

In the next section, we continue the discussion of gradient computation once the stochastic nodes have been dealt with via one of the gradient estimators presented above. In particular, we will see from the point of view of reverse-mode automatic differentiation, how the gradient estimates inside the expectations in Corollaries 3.2.1 and 3.2.2 can be obtained automatically for any differentiable

SCG.

3.3 Reverse-Mode Automatic Differentiation

Reverse-mode automatic differentiation[4] is a particular technique for automatically computing the gradient of a single (in general scalar-valued) output node w.r.t. to many upstream input nodes (Griewank *et al.*, 1989), in contrast to the *forward-mode automatic differentiation* that is more suited to compute the gradient of many output nodes w.r.t. to a single input node. Reverse-mode automatic differentiation can be viewed as a generalization of the well-known *error backpropagation* algorithm popularized by Rumelhart *et al.* (1986) in the connectionist machine learning community, even though it was studied and developed independently in the scientific computation community (Griewank and Walther, 2008). In very general terms, the basic idea of reverse-mode automatic differentiation (and also error backpropagation) is to implement in an algorithm the *derivative chain rule* for computing the derivative of a composition of functions, as shown below for the composition of functions f and g:

$$(f \circ g)(x) = f(g(x)) \Rightarrow (f \circ g)'(x) = f'(y)|_{y=g(x)} g'(x) \ . \tag{3.11}$$

The key point to observe is that the functional dependencies between the nodes in an SCG can be much more complex than the single composition shown in Equation 3.11, and therefore the algorithmic implementation of the chain rule in reverse-mode automatic differentiation will need to consider all the parents of a given node as summarized in Equation 3.12.

$$\frac{\partial J}{\partial x_i} = \sum_{j:\, i \in \textsc{Parents}(j)} \frac{\partial J}{\partial x_j} \frac{\partial x_j}{\partial x_i} \ . \tag{3.12}$$

Note that for a standard *Multi-Layer Perceptron* (MLP) neural network (Goodfellow *et al.*, 2016), which is essentially a linear, layered composition of many transformations, the Equation 3.12 will simplify to the product of the Jacobians of all the layer transformations. Nevertheless, for models with many paths and branches, a graph algorithm will be needed to recursively visit in topological order all upstream nodes of the node we want to compute the gradient. We outline this idea in Algorithm 2 that simulates all the computations in the SCG in the *forward pass* in Algorithm 3 and then backpropagates the gradients in the *backward pass* in Algorithm 4.

Algorithm 2: Reverse-Mode Automatic Differentiation

Input: node J, input nodes Θ
Output: the set of gradients $\{\nabla_\theta J \mid \theta \in \Theta\}$

for $\theta \in \Theta$ **do**
  $\theta.\text{grad} \leftarrow 0$
ForwardPass(J, Θ)
BackwardPass(J, 1, Θ)
return $\{\theta.\text{grad} \mid \theta \in \Theta\}$

In Algorithm 3, each node w has a value and a function associated. To compute the value of a

[4] Automatic differentiation is not be confused with other approaches such as finite differences or symbolic differentiation (e.g., as available in Mathematica® software (Wolfram *et al.*, 1999)).

given node, the algorithm first computes the values of its parent nodes recursively, and then uses its function to compute its current value and also to build the composition over the functions of the parent nodes. It is the functional dependency created by this composition that will guide the computation of the gradients in the backward pass.

Algorithm 3: ForwardPass

```
Input: node w, input nodes Θ
if w ∉ Θ then
    for v ∈ Pa(w) do
        ForwardPass(v, Θ)
    w.value ← w.function(Parents(w))
```

Finally, once the values and the functional dependencies are registered after the forward pass is executed, the Algorithm 4 visits all the nodes in the SCG and computes the product of the gradients resulting from local compositions, which are then combined on the fly when an input node is reached in the search. After the backward pass is finished, the nodes of interest will have its gradient correctly computed according to Equation 3.12.

As a final remark, we notice that in this section we presented an intuitive explanation of the subject of automatic differentiation. In practice, both the forward and backward passes are not implemented exactly as we outlined here, but are rather highly pre-processed and optimized in professional-grade automatic differentiation software such as TensorFlow (Abadi *et al.*, 2016) and others.

Algorithm 4: BackwardPass

```
Input: node w, gradient g_w, input nodes Θ
if w ∈ Θ then
    w.grad ← w.grad + g_w
else
    for v ∈ Parents(w) do
        g_v ← g_w · ∂w/∂v
        BackwardPass(w, g_v, Θ)
```

3.4 Examples

In this section, we present two illustrative examples where the previous gradient estimators have been put to use in practical applications in reinforcement learning and probabilistic generative models.

REINFORCE. The algorithm REINFORCE (Williams, 1992) is perhaps one of the oldest model-free reinforcement algorithms based on policy optimization. It belongs to the general class of Policy Gradients methods which has been extensively studied in robotics applications (Deisenroth *et al.*, 2013). The basic idea is to explicitly represent a policy by a parametric function approximator and update its parameters in the steepest direction of the policy's Q-value function averaged according

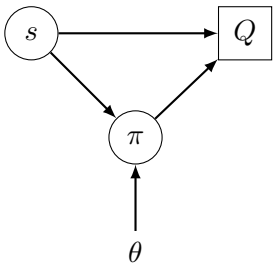
to the state visitation distribution of the policy in the MDP. Therefore, REINFORCE attempts to maximize the objective function outlined in the Theorem 2.2.5, for which the gradient is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_\theta}}\left[\mathbb{E}_{a\sim\pi_\theta}[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)]\right] \ . \tag{3.13}$$

Note that the gradient in Equation 3.13 is exactly the result of applying the likelihood ratio estimator to the Value function V_θ^π and averaging the gradient estimates over the states visited by the policy:

$$V_\theta^\pi(s) = \mathbb{E}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)\right] \ . \tag{3.14}$$

The stochastic computation graph for the gradient estimator of the REINFORCE algorithm is given by Figure 3.5. Note that there is no deterministic path from the node θ to the node representing the action-valued cost function $Q(s,\cdot)$ and the only dependency path connecting those node goes through the stochastic node of the stochastic policy π_θ. We can now clearly see that only the first term (related to the likelihood ratio estimator) of the general gradient formula in SCGs as given by the Theorem 3.1.1 applies to the REINFORCE estimator. As a matter of fact, the REINFORCE-based algorithms is so ubiquitous in model-free RL that it is informally equated to the likelihood gradient estimator itself, being used interchangeably in the community.



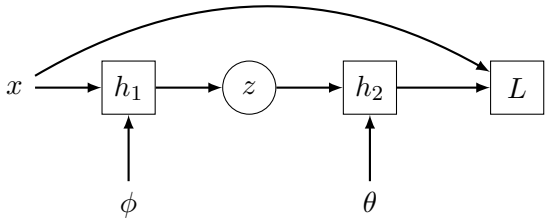
$$s \sim \rho^{\pi_\theta} \tag{3.15}$$
$$a \sim \pi_\theta(\cdot|s) \tag{3.16}$$
$$Q = Q^{\pi_\theta}(s,a) \tag{3.17}$$

Figure 3.5: *Stochastic computation graph for the REINFORCE algorithm*

The great appeal of the likelihood ratio estimator (and consequently of the REINFORCE and policy gradients method) for model-free RL is that a policy can be optimized by only obtaining samples of the action-value Q-function, either via Monte Carlo simulation or via function approximation. In other words, there is absolutely no need to have access to an analytical model of the MDP. As we have previously discussed, this type of gradient estimator, albeit very general, can exhibit high variance even if specific variance-reduction techniques are used. Nevertheless, this is the price to pay to go completely model-free.

Variational Auto-Encoder. VAEs (Kingma and Welling, 2014) are an example of stochastic generative models whose goal is to solve a problem of density estimation. In other words, given a training dataset their goal is to learn in an unsupervised way the underlying data distribution that generated the training dataset in the first place. In particular, VAEs attempt to encode the inputs in a low-dimensional space effectively generating an internal latent-space representation that can be later decoded to generate new outputs that hopefully belong to the input distribution. The basic implementation learns both the encoder and the decoder by minimizing a re-construction loss between the input and the generated output whose goal is to try to imitate the input from the low-dimensional representation. Intuitively, VAEs attempt to compress the input distribution in a

way that the model can be queried for sampling new outputs. Figure 3.6 shows an example of a VAE whose encoder and decoder are given by the neural networks $h_1(\cdot;\phi)$ and $h_2(\cdot;\theta)$, respectively. The stochastic node z represents a Gaussian distribution whose goal is to add smoothing noise to the latent representation in order to make the model more robust, and the re-construction loss L is essentially a differentiable similarity metric.



$$h_1 = h_1(x;\phi) \tag{3.18}$$
$$z \sim \mathcal{N}(\cdot|\mu(h_1),\sigma(h_1)) \tag{3.19}$$
$$\tilde{x} = h_2(z;\theta) \tag{3.20}$$
$$L = L(x,\tilde{x}) \tag{3.21}$$

Figure 3.6: *Stochastic computation graph for the VAE generative model*

As we can observe from the SCG of the VAE, in principle, we would need to resort to a similar likelihood ratio estimator to train the encoder as there are no deterministic paths from its parameters ϕ to the re-construction loss L. However, we can leverage the fact that node z follows a Gaussian distribution which can be effectively re-parametrized (much like the example in Figure 3.3). Thus, it can use the pathwise derivative gradient estimator as discussed in Section 3.2.2. In fact, VAEs and derived works have been known for promoting the use of the re-parametrization trick in machine learning in recent years. Notice that unlike in the case of model-free Policy Gradients, in the case of VAEs the stochastic model is actually fully known as it is in fact defined by construction.

3.5 Conclusion

This chapter introduced the formalism of SCGs and discussed the general types of gradient estimators and how they relate to internal dependencies between the internal variables of a stochastic model. Additionally, we presented an intuitive view of reverse-mode automatic differentiation and how the gradient of an arbitrary composition of functions can be efficiently obtained if it is given as a computation graph. Starting in the following chapter, we shall present a number of differentiable planning algorithms making the necessary connections with the theory of stochastic computation graph as presented here.

Part II

Differentiable Planning

Chapter 4

Planning through Backpropagation in Deterministic Domains

Planning through backpropagation has been recently proposed from the viewpoint of the AI planning community as an scalable alternative to address decision making in continuous state-action spaces (Wu *et al.*, 2017). The main motivation was based on the fact that the methods traditionally developed for search-based planning (e.g., MCTS (Kocsis and Szepesvári, 2006), numeric planning (Hoffmann, 2011; Scala *et al.*, 2016)) could not effectively handle problems with arbitrary nonlinear dynamics and/or cost functions, specially in high-dimensional and/or long-horizon tasks, due to the intractable branching factor implied by the required discretization of the action space.

In order to avoid the issues inherent to search-based methods, planning through backpropagation turns to gradient-based optimization techniques as the solution to the problem of searching in high-dimensional continuous state-action spaces. In this chapter, we define the approach of planning through backpropagation from a perspective slightly different to its original presentation based on the conceptual parallel with Recurrent Neural Networks (RNN). In particular, **we propose to view planning through backpropagation as a trajectory optimization technique** in the hope to present its advantages and limitations in a more principled theoretical way and also to allow future cross-fertilization with optimal control.

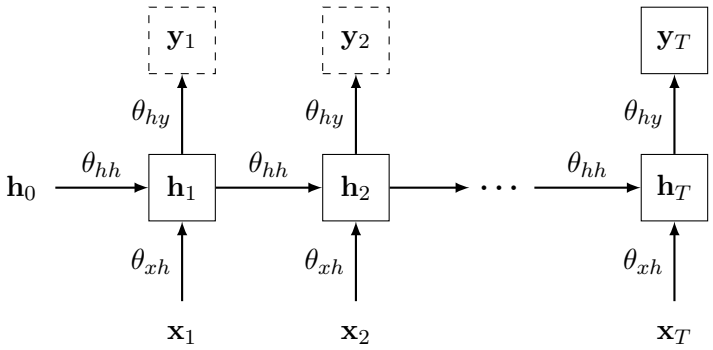
This chapter is organized as follows. We first introduce planning through backpropagation through the lens of trajectory optimization in connection with the framework of computation graphs. Besides defining the planning problem, the notation, and the high-level algorithmic formulation, it is our objective to clarify some conceptual misconceptions regarding the RNN-based characterization of the planning through backpropagation approach. After this introduction, we review and discuss a number of difficulties related to finding long-horizon plans through gradient-based optimization, handling action-constrained spaces, and mitigating local optima. These issues constitute the main computational challenges any practical implementation of planning through backpropagation must address. We conclude the chapter with a number of empirical experiments chosen to highlight the impact of different practical implementations. In addition, we demonstrate the effectiveness of planning through backpropagation in nonlinear problems and provide an empirical analysis of the optimality gap of TensorPlan (Wu *et al.*, 2017) in problems that can be exactly solved through optimal control techniques.

4.1 Planning through Backpropagation as Trajectory Optimization

Planning through backpropagation is a specific framework built around the application of stochastic gradient descent (Ruder, 2016) to the task of finding optimal sequences of actions in continuous spaces. It is particularly effective for an important class of planning problems exhibiting nonlinear transition and cost function. The key idea of planning through backpropagation in deterministic domains is best summarized by Wu *et al.* (2017) in their seminal paper:

> *"[...] we reverse the idea of training parameters of the network given fixed inputs to instead optimizing the inputs (i.e., actions) subject to fixed parameters [...]".*

Their observation is based on the conceptual analogy between dynamical systems and Recurrent Neural Networks (RNNs) in supervised learning. Those recurrent models are trained to predict output sequences given input sequences by updating a context-dependent hidden latent state in each time step. Therefore, RNNs are a class of generative models that attempts to build an explicit memory mechanism as a way to capture long-term relationships in the input sequence. The evolution of the RNN is governed by the recursive application of a parametric model known as the RNN's core, as illustrated in Figure 4.1.



$$\mathbf{h}_t = \phi_{\theta_1}(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad (4.1)$$

$$\mathbf{y}_t = \psi_{\theta_2}(\mathbf{h}_t) \quad (4.2)$$

Figure 4.1: *Recurrent Neural Net (*RNN*): $\boldsymbol{x}_t$, $\boldsymbol{h}_t$, and $\mathbf{y}_t$ are the input, the hidden latent state, and the output at time step t, respectively. The parametric functions ϕ_{θ_1} and ϕ_{θ_2} correspond to the recurrent and output kernels, respectively.*

According to Wu *et al.* (2017), it is rather natural to draw a parallel between the RNN inputs and outputs and the system's actions and costs trajectories. However, the analogy starts to break when one consider the role of the hidden state, $\mathbf{h}_i$, in the recurrent model. In effect, RNNs attempt to learn from data a compact representation of past inputs as a set of features correlated with the target outputs. This is usually done by fitting the inner RNN's core parameters θ to minimize some loss function defined over a dataset of input/output pairs. In contrast, the goal in the planning through backpropagation approach is not related to representation learning. It is to find near optimal inputs (i.e., the control actions) with respect to a total cost minimization problem; hence the idea of Wu *et al.* (2017) to "reverse" the training procedure to optimize the inputs rather than the parameters of the "network". In other words, there is no parameter learning involved, but the repurposing of the deep learning-based machinery (namely *Backpropagation-Through-Time* (BPTT)) to perform sensitivity analysis of the dynamical system with the goal of finding good inputs.[1]

[1]We wished to make this distinction clear between RNN training and planning through backpropagation as there has been some confusion within the community in the past.

In this section, we take an intentionally different approach to motivate planning through backpropagation. We argue that *Trajectory Optimization* in optimal control forms the theoretical foundation of planning through backpropagation. Trajectory optimization in control theory represents an important perspective shift from the classical setting of controller synthesis whose goal is to find a global mapping (i.e., a control law) from observations to control actions. By restricting the focus to only the current state, the optimization space is simplified considerably. Instead of optimizing over the infinite dimension of (feedback) functions, one can now aspire to optimize over the finite-dimension space of state-action trajectories. **We remark that the exact same modeling assumptions of trajectory optimization apply to planning through backpropagation, namely the differentiability of the transition and cost functions**. In this context, we formulate the planning task in discrete-time nonlinear deterministic domains as a constrained optimization problem. Under the assumptions of perfect information and deterministic actions, the solution of a planning problem defined over a finite horizon H for a particular initial state, $x_0 = \bar{x}$, is a sequence of valid control actions, $\mathbf{u} = (u_0, \dots, u_{H-1})$, that induces a state-action trajectory, $\tau_{0:H} = (x_0, u_0, \dots, x_{H-1}, u_{H-1}, x_H)$, minimizing the cumulative sum of costs, $G(\tau_{0:H}) = \sum_{t=0}^{H-1} g(x_t, u_t)$.

Definition 4.1.1. (Planning as Trajectory Optimization) *Let $\mathcal{M} = (\mathcal{X}, \mathcal{U}, f, g, \bar{x})$ be a deterministic control problem. We formulate the finite-horizon, continuous planning task as the following constrained optimization problem:*

$$\min_{\mathbf{x},\mathbf{u}} \quad \sum_{t=0}^{H-1} g(x_t, u_t) \tag{4.3}$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t), \tag{4.4}$$

$$u_t \in \mathcal{U}(x_t), \tag{4.5}$$

$$x_0 = \bar{x} \ . \tag{4.6}$$

The planning problem as directly transcribed in Definition 4.1.1 is in its general form a nonlinear constrained optimization problem defined over the state-action trajectory. The feasibility of a solution $(\mathbf{x}, \mathbf{u}) = (x_0, u_0, x_1, \cdots, x_{H-1}, u_{H-1}, x_H)$ is predicted on satisfying the dynamics of the system, $x_{t+1} = f(x_t, u_t)$, the applicability of actions, $u_t \in \mathcal{U}(x_t)$, and the initial state constraint, $x_0 = \bar{x}$. It is easy to make sure the initial state constraint is satisfied; it suffices to condition the initial state decision variable throughout the optimization procedure. Also, guaranteeing the applicability of actions can be made practical with a number of existing alternatives. We shall explore a subset of these possibilities in Section 4.3.2. However, how to effectively handle the dynamics constraints from the point of view of numerical optimization is not as straightforward. Trajectory optimization methods in control typically deals with the issue of enforcing the dynamics constraints either by *shooting* methods or by *direct transcription / collocation* methods. In the next sections, we overview these approaches.

4.1.1 Shooting Methods

The shooting method is the simplest way of handling the dynamics constraints in trajectory optimization. It implicitly encodes them in the objective function via forward simulation of states according to the transition function. Indeed, in shooting methods the variables corresponding to the state trajectory are no longer decision variables and the optimization is performed exclusively with respect to the actions. The Definition 4.1.2 formalizes the shooting optimization formulation. In optimal control, shooting methods are typically implemented with nonlinear programming techniques (e.g., Sequential Quadratic Programming (SQP) Nocedal and Wright (1999)). In contrast, the algorithm proposed in the seminal work in planning through backpropagation (Wu *et al.*, 2017) (informally known as *TensorPlan*) solves the shooting formulation via gradient descent techniques (Ruder, 2016).

Definition 4.1.2. (Optimization by shooting)

$$\min_{\mathbf{u}} \quad g(x_0, u_0) + g(f(x_0, u_0), u_1) + \cdots + g(f(\cdots f(x_0, u_0), u_1), u_{H-1}) \tag{4.7}$$

$$\text{s.t.} \quad u_t \in \mathcal{U}(x_t), \tag{4.8}$$

$$x_0 = \bar{x} \ . \tag{4.9}$$

Note that, in contrast to the general constrained formulation of Definition 4.1.1, the shooting formulation does not include the state trajectory, $\mathbf{x}$, as decision variables in the optimization task. The states are implicitly generated through "shooting", i.e., the recursive application of the transition function. In other words, the dynamics constraints are satisfied by construction. The computation graph in Figure 4.2 depicts the functional dependencies implied by the shooting method.

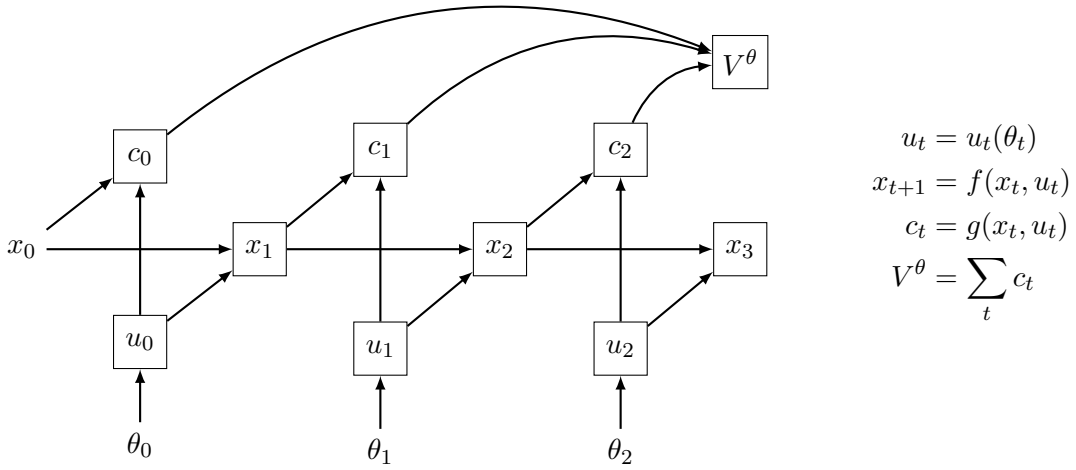


Figure 4.2: *Computation graph of a Deterministic Control Problem (DCP): the value function computed by the recursive application of the transition function composed with the cost function. Equivalently, the sequence of actions* $\mathbf{u} = (u_0, \cdots, u_{H-1})$ *is unrolled in time in order to obtain the parametric value function* $V^\theta(x_0; \mathbf{u})$*. Note that control actions,* $u_t \in \mathcal{U}(x_t)$*, can be parametrized by* $\theta_t \in \mathbb{R}^n$ *to deal with the case of action-constrained problems.*

4.1.2 Direct Transcription

On the other hand, direct transcription and collocation methods make the dynamics constraints explicit in the optimization problem. It typically resorts to constrained nonlinear programming techniques (e.g., interior point or log-barrier methods) or Lagrangian methods in order to solve the constrained optimization problem. Of particular interest to the framework of planning through backpropagation is the Lagrangian formulation. The key observation in Lagrangian methods applied to trajectory optimization is the fact that violations to the dynamics can be penalized in the objective function in such a way that upon convergence a minimizing cost and feasible solution is obtained.

Definition 4.1.3. (Optimization by direct transcription via augmented Lagrangian)

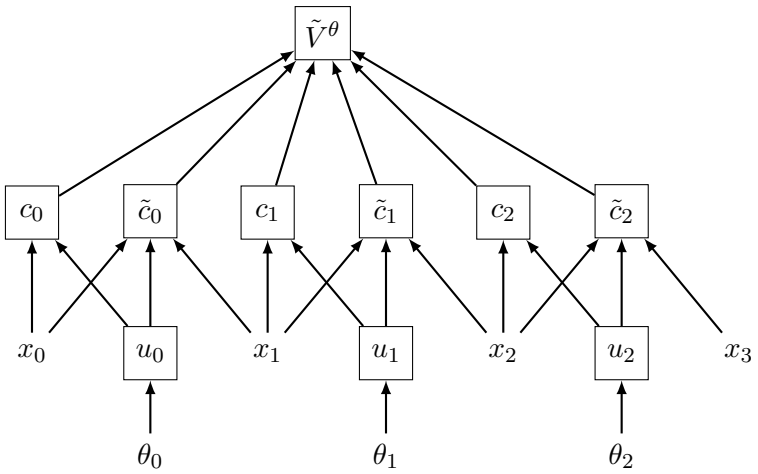
$$\min_{\mathbf{x},\mathbf{u}} \max_{\lambda} \quad \sum_{t=0}^{H-1} g(x_t, u_t) + \lambda_t \|x_{t+1} - f(x_t, u_t)\|^2 \tag{4.10}$$

$$\text{s.t.} \quad u_t \in \mathcal{U}(x_t), \tag{4.11}$$

$$\lambda_t \geq 0, \tag{4.12}$$

$$x_0 = \bar{x} \ . \tag{4.13}$$

It can be shown that under some regularity conditions (Nocedal and Wright, 1999), the penalty formulation can be solved with a primal-dual approach defined as the saddle point problem shown in Definition 4.1.3. In the primal problem, both the state and the action variables are optimized with the aim at minimizing the total cost added to an artificial cost corresponding to the degree of feasibility violation. In the dual problem the weight of the violation cost is then optimized accordingly.[2]



$$u_t = u_t(\theta_t)$$
$$c_t = g(x_t, u_t)$$
$$\tilde{c}_t = \lambda_t \|x_{t+1} - f(x_t, u_t)\|^2$$
$$\tilde{V}^\theta = \sum_t (c_t + \tilde{c}_t)$$

Figure 4.3: *Computation graph for the direct transcription method applied to a Deterministic Control Problem (DCP): both the states and actions (i.e., parameters θ) are to be understood as the inputs to the computation graph. Unlike in the shooting formulation, the violations of dynamics constraints are treated as penalties $\tilde{c}_t$ in the overall objective function $\tilde{V}^\theta$.*

[2] We refer the interested reader to the extensive treatment of numerical optimization in Nocedal and Wright (1999) for further details on Lagrangian methods.

One important consequence of formulating the approach of planning through backpropagation as trajectory optimization is that in principle we can take inspiration from the optimal control literature to explore other formulations that could be better suited for different problems. As a matter of fact, Planning through Backpropagation as initially proposed in Wu *et al.* (2017) is a shooting method as the states variables are implicitly generated by forward simulation of the input actions starting from the initial state. However, the numerical optimization algorithm used to find the sequence of optimal actions is borrowed from the deep learning literature. Instead of resorting to nonlinear mathematical programming as is typically used in optimal control, planning through backpropagation (as its name implies) leverages gradient descent and, in particular, the Backpropagation-through-Time (BPTT) algorithm as its iterative optimization procedure.

Nevertheless, it suffers from the same difficulties as in traditional shooting methods in optimal control. As shooting methods globally optimize the actions, the underlying optimization problem is oftentimes poorly conditioned due to recursive application of the dynamics function. In other words, the effect on the total cost of the first action in the plan is considerably higher than the effect of the last action as even a small change of the values of the first action can considerably change the course of the entire trajectory. In practice, this poor conditioning can lead to vanishing or exploding gradients depending on the particular characteristics of the system dynamics (as discussed in Section 4.3.1). Consequently, shooting-based methods can suffer from severe local optima in tasks requiring long-term reasoning. The intuition is that shooting methods attempt to solve a difficult conflicting optimization problem in which at the same time it must steer the state towards low-cost regions and search for the actions that will get the system there. In contrast, direct transcription methods exploit the Markovian structure of the sequential decision problem to build an objective that only deals with pairwise dependencies between temporally adjacent states without the recursive application of the transition function. Intuitively, it disentangles the sub-problems of finding low-cost regions in the state space and constructing a feasible plan to drive the system there. Those are the main reasons why in optimal control direct transcription and collocation methods are usually preferred over direct shooting methods.

However, the choice between shooting and direct transcription methods is not a clear cut. On one hand, the core issue of shooting methods related to vanishing and exploding gradients is a well-studied problem in deep learning (and specifically in RNN research) for which a number of mitigating solutions have been successfully proposed. On the other hand, direct transcription is a much more involved optimization approach for which the issue of the initialization of state-action trajectories has a direct impact on its ability to find a feasible and high-quality solution.

In the next sections, we shall discuss in details the necessary techniques for the effective application of the shooting-based planning through backpropagation as originally proposed by Wu *et al.* (2017). In the final section of this chapter, we shall empirically compare both the shooting and direct transcription formulations. We remark that regardless of the optimization formulation (i.e., Definition 4.1.2 or 4.1.3), in this work we investigate solutions obtained via the gradient descent method to solve the DCP problem.

4.2 Optimizing Plans via Gradient Descent

As discussed previously, planning through backpropagation applies gradient descent to optimize trajectories. Therefore, the only assumption made by the approach is that a differentiable dynamics model and cost function are available.[3] Denoting the value function of a plan $\mathbf{u} = (u_0, \cdots, u_{H-1})$ executed from a start state x by $V(x; \mathbf{u}) = \sum_{t=0}^{H-1} g(x_t, u_t)$, the basic idea is to update a current plan $\mathbf{u}^k$ by the following first-order iterative optimization procedure:

$$\mathbf{u}^{k+1} = \mathbf{u}^k - \alpha_k \nabla_{\mathbf{u}} V(x; \mathbf{u})\big|_{x=\bar{x}, \mathbf{u}=\mathbf{u}^k} \, . \tag{4.14}$$

If the value function $V(x; \mathbf{u})$ is convex in the sequence of actions $\mathbf{u}$ and the learning rate α_k follows a schedule satisfying the Robbins–Monro conditions (Robbins and Monro, 1951) (i.e., $\sum_{k=0}^{\infty} \alpha_k = \infty$ and $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$), then $\mathbf{u}^k \xrightarrow{k \to \infty} \mathbf{u}^*$, where $\mathbf{u}^* = \arg\max_{\mathbf{u}} V(x; \mathbf{u})$ is an optimal plan for the state x. In general, however, the trajectory optimization problem will rarely be convex (except in a few important cases) and hence the best we can hope for is to find a good local optimum. We will see in Section 4.3.3 a practical implementation improvement that can alleviate the local optimum problem.

We remark that the presentation so far focused on the most basic algorithmic formulation of planning through backpropagation for the purpose of clarity of exposition. We shall remind ourselves that one of the motivation of planning through backpropagation is to leverage advanced techniques from deep learning to perform trajectory optimization. In particular, there is no reason we should restrict ourselves to: (i) directly optimize the cost-to-go of a plan – we shall see in following chapters that any increasing smooth function of the return can be optimized; and more importantly, (ii) directly update the plan with the $\nabla_{\mathbf{u}} V(x; \mathbf{u})$ – we can leverage advanced non-convex optimizers (Ruder, 2016) with adaptive learning rate and momentum to preprocess the gradient in order to obtain better convergence properties. Therefore, in order to present a more general view of the proposed approach, we abstract away these optimization details and summarize the overall steps of planning through backpropagation in Algorithm 5.

Algorithm 5: Planning through Backpropagation

Input: DCP $\mathcal{M} = (\mathcal{X}, \mathcal{U}, f, g)$, start state $\bar{x}$, horizon H, iterations K
Output: plan $\mathbf{u}^K$

$\mathbf{u}^0 \sim q(\mathcal{U}^H)$; ▷ Initialization
for $k = 0, \ldots, K-1$ **do**
  $(\tau_{0:H}^k, V^k) \leftarrow \textsc{Forward}(\mathcal{M}, \bar{x}, \mathbf{u}^k)$; ▷ Simulation
  $\nabla_{\mathbf{u}^k} V^k \leftarrow \textsc{Backward}(\tau_{0:H}^k, V^k)$; ▷ Backpropagation-Through-Time
  $\mathbf{u}^{k+1} \leftarrow \textsc{Update}(\mathbf{u}^k, \nabla_{\mathbf{u}^k} V^k)$; ▷ Optimization
return $\mathbf{u}^K$

Planning through backpropagation works by alternating *forward* and *backward* passes on top of the computation graph illustrated in Figure 4.2. At the beginning of training, a valid plan $\mathbf{u}^0$ is initialized at random from a proposal distribution $q(\cdot)$. Then, for K iterations the algorithm interleaves the forward and backward subroutines. First, in the forward pass, the current plan,

[3] In principle, gradient-based planning is agnostic to the nature of the model and how it was first obtained. Therefore, the techniques presented in this chapter can be applied to models specified by a specialist or directly learned from data. Also, it can be used both for models grounded in the decision process or for latent-space models.

$\mathbf{u}^k$, is simulated from the start state $\bar{x}$ according to the system's dynamics which generates the trajectory $\tau^k_{0:H}$ with total cost (i.e., value function) given by V^k. Afterwards, in the backward pass, the gradient of the total cost for each control action u_t is recursively computed and stored in the tensor $\nabla_{\mathbf{u}^k} V^k = (\nabla_{u^k_0} V^k, \cdots, \nabla_{u^k_{H-1}} V^k)$. Finally, the gradient previously computed is processed by a non-convex optimizer and the current plan, $\mathbf{u}^k$, is updated thus generating an improved plan, $\mathbf{u}^{k+1}$.

4.2.1 Gradient Computation with Backpropagation-Through-Time

The most important part of the planning through backpropagation algorithm (as its name implies) is the backward pass. Indeed, the trajectory simulation performed at the forward pass is a simple iterative application of the dynamics, $x_{t+1} = f(x_t, u_t)$, and the cost function $c_t = g(x_t, u_t)$, and the final update step also performs simple computations to generate the new plan. In this section, we present the computational technique behind the backward pass, namely *Backpropagation-Through-Time* (BPTT).

The algorithm of BPTT (Werbos, 1990; Williams and Zipser, 1995) is an extension of the well-known error *backpropagation* (Rumelhart *et al.*, 1986) algorithm for neural networks which is in turn an specialized form of *reverse-mode automatic differentiation* (Griewank and Walther, 2008) in general computation graphs. Informally, the backpropagation algorithm computes the gradient of the output error of a neural network for each of its parameters by recursively exploiting the derivative chain rule for composition of functions. BPTT extends this idea for recurrent models that can be *unrolled* in time, thus obtaining a time-dependent layered model that is in nature not very different than the traditional multi-layer feed-forward networks. The key value of this conceptualization is that training a recurrent model is similar to training feedforward networks with the only difference that certain constraints need to be impose as the parameters of the recurrent model must be shared across time steps.

4.2.2 Dynamic Programming in the Space of Gradients

The particular instantiation of BPTT for the deterministic control problem leads to the differentiation of the value function of a plan with respect to each action. If we recall the principle of optimality for the deterministic setting, i.e., $V(x_0; \mathbf{u}_{0:H-1}) = g(x_0, u_0) + V(f(x_0, u_0); \mathbf{u}_{1:H-1})$, then we can apply the chain rule of derivatives to obtain the following gradient equations:

$$\nabla_{u_t} V(x_t; \mathbf{u}_{t:H}) = [\nabla_u g(x_t, u) + \nabla_u f(x_t, u) \nabla_{x'} V(x'; \mathbf{u}_{t+1:H})]\big|_{u=u_t,\ x'=f(x_t,u_t)} , \tag{4.15}$$

$$\nabla_{x_t} V(x_t; \mathbf{u}_{t:H}) = [\nabla_x g(x, u_t) + \nabla_x f(x, u_t) \nabla_{x'} V(x'; \mathbf{u}_{t+1:H})]\big|_{x=x_t,\ x'=f(x_t,u_t)} . \tag{4.16}$$

Therefore, by the recursive application of Equations 4.15 and 4.16, planning through backpropagation leverages BPTT to efficiently implement its backward pass. Algorithm 6 outlines the details of BPTT for the deterministic trajectory optimization problem. We remark that by making use of the computation graph of Figure 4.2, BPTT can compute all the necessary gradients symbolically at about the same computational cost of evaluating the underlying functions. In addition, it has been demonstrated that symbolic differentiation can evaluate partial derivatives more accurately and cheaply than methods that rely on numerical differences (Griewank *et al.*, 1989). Finally, we note that BPTT has space complexity in $\Theta((m+n)H)$ and average time complexity per time step

in $\Theta(n^2)$, where H represents the truncated horizon and n and m are the size of the state and action spaces, respectively (Williams and Zipser, 1995).

Algorithm 6: Backpropagation-Through-Time in Deterministic Control Problems

Input: DCP $\mathcal{M} = (\mathcal{X}, \mathcal{U}, f, g)$, trajectory $\tau_{0:H} = (x_0, u_0, \cdots, u_{H-1}, x_H)$, value function V
Output: gradient of total cost $\nabla_{\mathbf{u}} V = (\nabla_{u_0} V, \cdots, \nabla_{u_{H-1}} V)$

$\nabla_x V \leftarrow 0$
for $t = H-1, \ldots, 0$ **do**
  $\nabla_{u_t} V \leftarrow \nabla_u g(x_t, u)|_{u=u_t} + \nabla_u f(x_t, u)|_{u=u_t} \nabla_x V$; ▷ action value gradient
  $\nabla_x V \leftarrow \nabla_x g(x, u_t)|_{x=x_{t-1}} + \nabla_x f(x, u_t)|_{x=x_{t-1}} \nabla_x V$; ▷ state value gradient
return $\nabla_{\mathbf{u}} V$

4.3 Limitations of Planning through Backpropagation

4.3.1 Vanishing and Exploding Gradients over Long Horizons

As discussed in the last section, BPTT computes an improvement direction by a simple backward calculation, which can be seen as a direct application of a *dynamic programming algorithm over the space of gradients*. Though computational efficient, this approach hides some potential problems from the point of view of trajectory optimization. In this section, we unfold the dynamic programming used in BPTT to analyze the interactions over long horizons between the Jacobians of the transition function, $\nabla_{x,u} f(x, u)$, and the value gradients, $\nabla_{x,u} V(x; \mathbf{u})$. As usual, let $G_t = \sum_{i=t}^{H-1} c_i$ denote the cost-to-go from time step t. Then, by the Markov property and the linearity of the gradient operator, the action-value gradient is given by:

$$\nabla_{\mathbf{u}} V(x; \mathbf{u}) = \sum_{t=1}^{H-1} \frac{\partial G_t}{\partial u_t} \ , \tag{4.17}$$

where $\frac{\partial G_t}{\partial u_t}$ denotes the partial derivative of the cost-to-go G_t with respect to action u_t, i.e., the rate of change of the cumulative downstream cost when subsequent actions are kept fixed.

In order to evaluate the global importance of the action at time step t in the computation of the value gradient, we expand the term $\frac{\partial G_t}{\partial u_t}$ in the following sum-of-products form:

$$\frac{\partial G_t}{\partial u_t} = \frac{\partial}{\partial u_t} \sum_{i=t}^{H-1} c_i \tag{4.18}$$

$$= \frac{\partial}{\partial u_t} \left(c_t + \sum_{i=t+1}^{H-1} c_i \right) \tag{4.19}$$

$$= \frac{\partial c_t}{\partial u_t} + \sum_{i=t+1}^{H-1} \frac{\partial c_i}{\partial u_t} \tag{4.20}$$

$$= \frac{\partial c_t}{\partial u_t} + \sum_{i=t+1}^{H-1} \frac{\partial x_{t+1}}{\partial u_t} \frac{\partial c_i}{\partial x_{t+1}} \tag{4.21}$$

$$= \frac{\partial c_t}{\partial u_t} + \frac{\partial x_{t+1}}{\partial u_t} \sum_{i=t+1}^{H-1} \frac{\partial c_i}{\partial x_{t+1}} \tag{4.22}$$

$$= \frac{\partial c_t}{\partial u_t} + \frac{\partial x_{t+1}}{\partial u_t} \sum_{i=t+1}^{H-1} \left(\prod_{j=t+1}^{i-1} \frac{\partial x_{j+1}}{\partial x_j} \right) \frac{\partial c_i}{\partial x_i} \tag{4.23}$$

By rewriting the gradients in this form we can identify the *temporal* components, $\frac{\partial c_i}{\partial x_{t+1}}$, from which we loosely distinguish between *long term* and *short term* contributions, depending on whether the condition $t \ll i$ holds or not. By the chain rule of derivatives, we can observe that each temporal component is given by the product of the Jacobians of the transition function, $\frac{\partial x_{j+1}}{\partial x_j}$. Similarly to the case where a product of $i-t-1$ real numbers can diverge to infinity or shrink to zero as $i \to \infty$ depending on its absolute value, a product of matrices can explode or vanish. It can be proven that it is sufficient that $\rho < 1$, where ρ is the largest eigenvalue of the Jacobian matrix $\frac{\partial x_{t+1}}{\partial x_t}$, for long-term components to decrease exponentially fast in the horizon, and conversely it is necessary that $\rho > 1$ for the gradients to explode.

These numerical issues are behind the most important challenge for the application of BPTT in long-horizon tasks (either in planning or supervised learning). In the literature of recurrent models, such complications are know as the *exploding* and the *vanishing* gradient problems and are typically the culprit behind the difficulty of training RNNs (Bengio *et al.*, 1993). As a matter of fact, exploding gradients can lead to catastrophic failures due to uncontrolled divergences and vanishing gradients can make it impossible to capture long-term dependencies present in the data, thus leading to highly myopic models. Both the vanishing and the exploding gradient problems have been extensively studied in deep learning. Besides from standard techniques of parameter regularization and teacher forcing in neural network training, several solutions have been specifically proposed for RNNs, e.g., specialized architectures such as *Long-Short Term Memory* (LSTM) (Greff *et al.*, 2017) or *Gated Recurrent Networks* (GRU) (Chung *et al.*, 2014), gradient norm clipping and Jacobian regularization (Pascanu *et al.*, 2013).

For the case of trajectory optimization in continuous planning tasks, we observe that there is an important class of domains for which BPTT does not exhibit pronounced vanishing gradients for moderately large horizons (i.e., $H < 1000$). For instance, if the system dynamics is of the general form $x_{t+1} = x_t + \phi(x_t, u_t)$, it has been shown empirically that the linear dependence on x_t keeps the largest determiner of the Jacobian $\frac{\partial x_{t+1}}{\partial x_t}$ above one for most time steps assuming $\phi(x_t, u_t)$ is a smooth function, thus limiting the effect of the vanishing gradients. At a closer look,

the aforementioned general form of the dynamics has straight resemblance with the architecture of LSTMs and residual neural networks (He *et al.*, 2016; Veit *et al.*, 2016; Zaeemzadeh *et al.*, 2018); typical models in deep learning that can be trained for dozens or in some cases for hundreds of layers / time steps. Additionally, we remark that in the case of recurrent models the natural implementation of teacher forcing (i.e., setting the values of specific layers to given targets) naturally lead to the notion of cost shaping (Ng *et al.*, 1999), which for a number of problems can be easily incorporated in the domain description.

Finally, concerning the exploding gradient problem, we remark that the simple solution of gradient clipping proposed by Pascanu *et al.* (2013) has been quite effective for the continuous problems we consider in this thesis. Gradient clipping is implemented by using

$$\nabla_{\mathbf{u}}\tilde{V} = \frac{\nabla_{\mathbf{u}}V}{\|\nabla_{\mathbf{u}}V\|}\min(\mathcal{G}_{\max}, \|\nabla_{\mathbf{u}}V\|) \ , \tag{4.24}$$

as a surrogate gradient where $\mathcal{G}_{\max}$ is a constant defined via experimental hyperparameter tuning.

4.3.2 Handling Continuous Action-Constrained Spaces

In most continuous planning tasks, it is usually the case that hard constraints apply to actions, thus restricting the set of valid actions, $u_t \in \mathcal{U}(x_t)$. Therefore, in order to find feasible plans such domains it is fundamental to find an efficient way to handle such constraints in gradient-based planning approaches. At the very least, planning through backpropagation should be able to accommodate actions whose range are an open, a closed or a half-open/half-closed interval of the reals.

There are several alternatives for handling action constraints: penalty functions, constructive approaches based on squashing functions or action clipping, projected gradient descent, and projected newton methods. Action clipping is the most basic alternative and is akin to transform the original planning domain into another one with the actions unconstrained but with a transition and reward functions that encompass an action preprocessing step that guarantees that an interval-based hard constraint is always satisfied. Projected stochastic gradient descent (PSGD)(Nocedal and Wright, 1999) is a well-known descent method that can handle constrained optimization problems by projecting the parameters (actions) into a feasible range after each gradient update. In the case of box-constrained actions, i.e., $u_t \in [u_{\text{MIN}}, u_{\text{MAX}}]$, projected gradient descent can be easily implemented via clipped gradients. Squashing functions are also a natural option in deep learning and by extension to planning through backpropagation. Typical activation functions used in neural networks such as the sigmoid function, $\sigma(x) = \frac{1}{1+\exp -x}$, the hyperbolic tangent, $\tanh(x) = \frac{\exp x - \exp x}{\exp x + \exp x}$, or the ReLU function, $ReLU(x) = \max(0, x)$, can be adapted to guarantee that the output will be limited in the valid range. Figure 4.4 shows the graphs of different kinds of activation functions.

We note that it is in general hard to characterize the effects of each of these alternatives in the final performance as the technical details are domain-dependent. Therefore, in this thesis, unless specifically noted otherwise, we subscribe to the solution based on squashing functions, but with a minor addition. To avoid big changes in the norm of action gradients, $\frac{\partial u_t}{\partial \theta_t}$, we parametrize the actions as per the function $u_t \colon \mathbb{R} \to [u_{\text{MIN}}, u_{\text{MAX}}]$ defined as follows for $b = \frac{u_{\text{MIN}} + u_{\text{MAX}}}{2}$, and $w = \frac{u_{\text{MAX}} - u_{\text{MIN}}}{2}$:

$$u_t(\theta_t) = b + w \tanh\left(\frac{1}{w}\theta_t\right) \ . \tag{4.25}$$

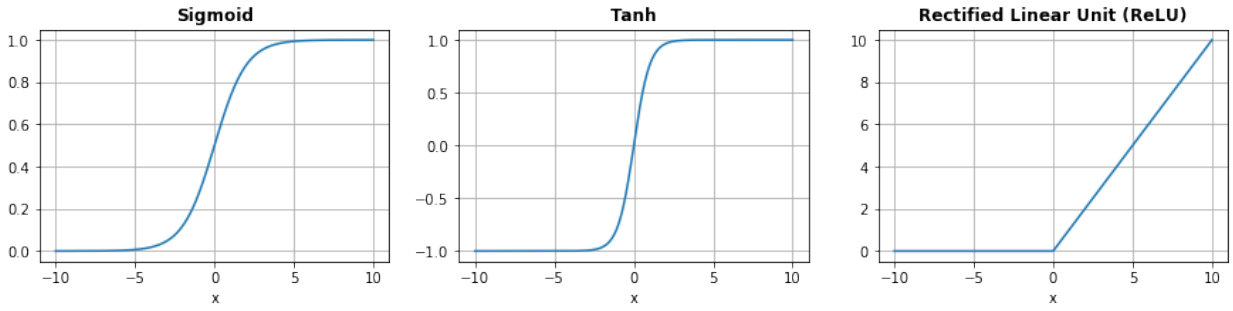


Figure 4.4: *Activation functions commonly used in parametric models in deep learning: planning through backpropagation can leverage these nonlinearities to constructively impose box constraints in the action space.*

Equation 4.25 allows to optimize the free parameters θ_t while it keeps the values of actions u_t satisfying their box-constrained applicability requirements. We remark that for such squashing function to work in practice, it is import to add a regularization loss over the norm of parameters θ_t to avoid local minima induced by the saturation of the actions whenever θ_t become too large in absolute value.

4.3.3 Mitigating Local Optima

Planning through backpropagation is a trajectory optimization method based on gradient descent. Henceforth, the best we can hope for is to obtain good local optima as its underlying optimization is basically performing a local search on the space of plans. A simple mitigation solution is to simultaneously run multiple parallel planners initialized with different candidate plans. We note this is akin to random restarts, but with the advantage of the massive parallelization that automatic differentiation libraries naturally allow (Abadi *et al.*, 2016).

In addition, besides the previously discussed issues related to the vanishing and exploding gradient problems, we observe that another difficulty may impact the convergence of planning through backpropagation. The very nature of shooting methods that compose the transition function for many time steps may make the optimization of long-horizon plans harder. We argue that, even in the rather benign case of norm-preserving state Jacobians (i.e., $\left\| \frac{\partial x_{t+1}}{\partial x_t} \right\|_2 = 1$) where gradient information flows without much hassle, the actions at the beginning of the trajectory may have a much prominent effect on the total cost, whenever immediate costs, $g(x_t, u_t)$, are densely distributed across time steps.

The practical consequence of the unbalanced effects on the norm of action gradients $\frac{\partial G_t}{\partial u_t}$ is that it can lead to an ill-conditioned optimization problem. In this case, if the planning horizon is large enough, the basic version of the gradient descent optimizer may suffer from overshooting the optimum and eventually never converge. It has been observed that such ill-conditioning issue is likely to be more pronounced the larger the dimensionality of the parameters being optimized (in the case of planning through propagation this amounts to the number of time steps multiplied by the size of the action space). This problem is so pervasive in deep learning that several advanced optimizers (Ruder, 2016) have been proposed to preprocess the gradients so that its norm is appropriately normalized across the dimensions. In particular, RMSProp (Hinton *et al.*, 2012b) and Adam (Kingma and Ba, 2015) have shown to be robust enough to such ill-conditioning problems in many trajectory optimization problems (Wu *et al.*, 2017).

For the purposes of the experiments in this thesis, we used the adaptive optimizer RMSProp that implements momentum by dividing the learning rate η by an exponentially decaying average of squared gradients, thus maintaining independent learning rates for each parameter, according to the following equations:

$$\mathbb{E}[g^2]_{(k)} = \gamma \mathbb{E}[g^2]_{(k-1)} + (1-\gamma) g^2_{(k)} \tag{4.26}$$

$$\theta_{(k+1)} = \theta_{(k)} + \frac{\eta}{\sqrt{\mathbb{E}[g^2]_{(k)} + \epsilon}} g_{(k)}, \tag{4.27}$$

where $g = \nabla_\theta J(\theta)$ and ϵ is a small non-negative smoothing term that avoids division by zero.

4.4 Experiments

4.4.1 Effectiveness of TensorPlan in Simple Problems

In this section, we investigate the empirical performance of planning through backpropagation on three domains: (*i*) double integrator, (*ii*) Navigation 2D, and (*iii*) LQR problems. The main goal with the experiments is to demonstrate the effectiveness of gradient-based planning in deterministic nonlinear planning domains. We point out that in the experiments in this section all negative rewards in the plots are to be interpreted as costs.

Double Integrator. The double integrator is a classic problem in discrete-time continuous control in which a second-order system needs to be steered from an initial, unstable state to an equilibrium state, i.e., the point 0, conventionally. It can be seen as an idealized model for motion control in physical systems. For this particular example, we consider a 1-dimensional integrator model defined as follows. The state variables, $\mathbf{x}_t = (p_t, v_t)$, model the position and velocity of the system. The control action u_t applies an acceleration to the system changing its velocity:

$$p_{t+1} = p_t + v_t \ , \tag{4.28}$$

$$v_{t+1} = v_t + u_t \ . \tag{4.29}$$

Note that the action only influences the position indirectly, i.e., via integration over time; hence the name double integrator. The action u_t is bounded in the interval $[-1.0, 1.0]$.

Also, the cost function is given by the square distance to the position point 0 added to the η-weighted square of the acceleration applied to the motion, a term typically associated with the energy expenditure in physical systems:

$$g(\mathbf{x}_t, u_t) = p_t^2 + \eta u_t^2 \ . \tag{4.30}$$

We ran TensorPlan for the double integrator with initial state $\mathbf{x}_0 = (-1.0, 0.0)$ and compare it against a random baseline plan. We train a shooting plan with horizon $H = 20$ for 1000 iterations with RMSProp optimizer with learning rate 2e-2 and batch size 256.

Figure 4.5 shows the performance of the random plan which very rapidly destabilizes the system. In contrast, TensorPlan manages to steer the system towards its minimum cost state, i.e., the goal

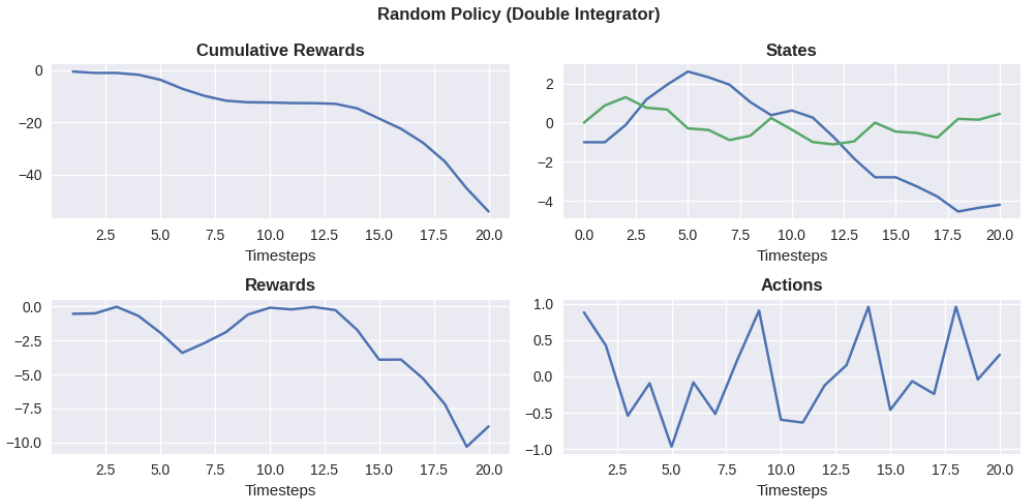


Figure 4.5: *Double Integrator: a single run of a random plan. The system is quickly destabilized. In the top right chart, the velocity and the position of the system are given by the green and blue curves, respectively.*

position in point 0. Figure 4.6 shows that the algorithm finds a plan that leads the system to equilibrium in approximately 5 time steps. We can observe that as expected the plan obtained first accelerates the system until a certain velocity threshold and then decelerates in accordance to the inertia of the integrator.

Figure 4.6: *Double Integrator: a single run of TensorPlan. The near optimal plan obtained via shooting steers the system to the goal position in approximately 5 timesteps. In the top right chart, the velocity and the position of the system are given by the green and blue curves, respectively.*

Navigation 2D. In the Navigation 2D problem, we model a fixed-horizon path finding problem (inspired by the formulation proposed in Faulwasser and Findeisen (2009)) in which an agent is supposed to move from a start to a goal position while avoiding specific regions along its way. For the purposes of demonstrating the ability of planning through backpropagation to handle nonlinear dynamics and costs, we use a simplified vehicle dynamics discretized to simulation step h. Let $\mathbf{x} = (x_t, y_t, \theta_t, v_t)$ be the state vector describing the 2D position, $pos(x_t, y_t)$, the orientation θ_t, and the velocity v_t. The action vector $\mathbf{u}_t = (u_t^\theta, u_t^v)$ consists of the steering angle $u_t^\theta \in [-\pi, \pi]$ and the

forward acceleration $u_t^v \in [-1.0, 1.0]$. The system dynamics are given by the following equations:

$$x_{t+1} = x_t + hv_t \sin(\theta_t) \ , \tag{4.31}$$

$$y_{t+1} = y_t + hv_t \cos(\theta_t) \ , \tag{4.32}$$

$$\theta_{t+1} = \theta_t + hv_t u_t^\theta \ , \tag{4.33}$$

$$v_{t+1} = v_t + hu_t^v \ . \tag{4.34}$$

We define the cost function $g(\mathbf{x}_t, \mathbf{u}_t)$ in Equation 4.35. It promotes proximity to the goal position, pos_{goal}, while penalizing the paths that move close to the disallowed regions specified by the parameters η_i and r_i. In addition, we also penalize energy expenditure in order to discourage action saturation and consequently to obtain smoothed trajectories.

$$g(\mathbf{x}_t, \mathbf{u}_t) = \|pos(x_t, y_t) - pos_{\text{goal}}\|^2 + \sum_i \eta_i \exp\left(-\frac{\|pos(x_t, y_y) - pos_i\|^2}{r_i}\right) + \eta\|\mathbf{u}\|^2 \tag{4.35}$$

In addition to the double integrator, we benchmark TensorPlan against the 2-dimensional nonlinear Navigation problem as described above. Note that unlike in the previous experiment, the Navigation problem exhibits a nonlinear transition dynamics and a more complex cost function with additional conflicting terms to be optimized.

We ran the TensorPlan algorithm for 1000 iterations with RMSProp optimizer with learning rate 2e-2 with batch size 256. We solved the problem for a fixed goal position, but varied the start position to highlight different paths found by the planner. We have chosen appropriate horizons for each start position, giving more time steps to distant starting points. Figure 4.7 shows how the current best plan in the batch evolves along the training iterations. As expected, the planner finds a near optimal path for each starting point whilst avoiding the high-cost regions.

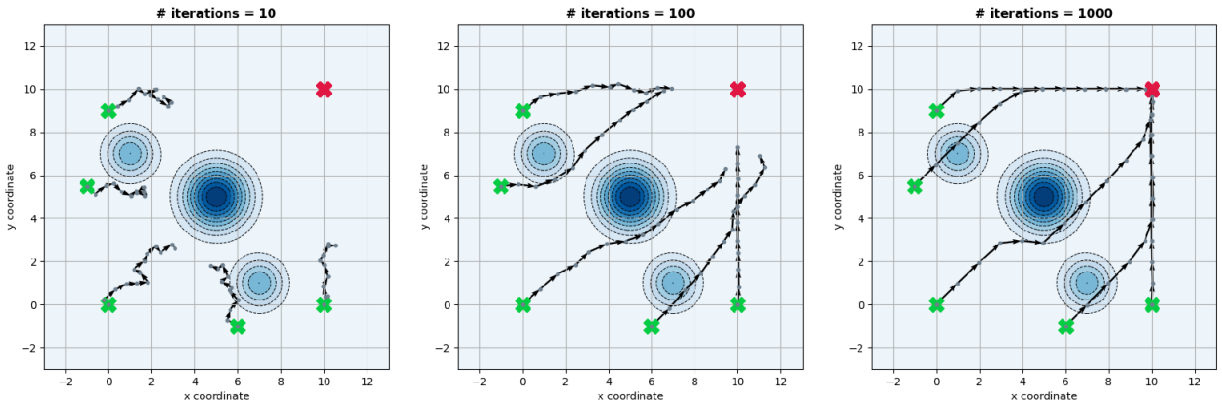


Figure 4.7: *Evolution of trajectories during training in Navigation 2D. The current best plan: (left) at the very beginning of training, (middle) after 100 iterations, and (right) at convergence after 1000 iterations.*

In order to compare the effects of the shooting formulation in Definition 4.1.2 to the collocation formulation in Definition 4.1.3, we implemented a version of TensorPlan that optimizes the computation graph given in Figure 4.3 instead of the one in Figure 4.2 by jointly minimizing the total cost of the state-action trajectory, $(\mathbf{x}, \mathbf{u})$, and the dual penalty term for dynamics violation.

We ran the training for 3000 iterations with RMSProp optimizer with learning rate 5e-2 with

batch size 256 and used a simple linear schedule for the dual variable λ_t ranging from 0.1 to 1000 in 3000 iterations. Figure 4.8 shows how a single trajectory evolves during training. We can observe at the beginning of training that a randomly-generated initial state-action trajectory drastically violates the dynamics constraints, meaning that the overall path is completely disconnected. After a number of iterations, however, the cost of violating the dynamics dominates the total training cost and forces the optimizer to sequentially connect all the states. Once the penalty term is minimized, the optimizer manages to find a near optimal path towards the goal position, thus minimizing the original cost function of the problem.

We remark that the hyperparameters used here were chosen to clearly separate the three distinct training phases illustrated in Figure 4.8 and that different hyperparameters can substantially accelerate the training. Regardless, we point out that our main intent with this experiment is to demonstrate an alternative to the shooting formulation as the underlying optimization formulation that can be used in differentiable planning.

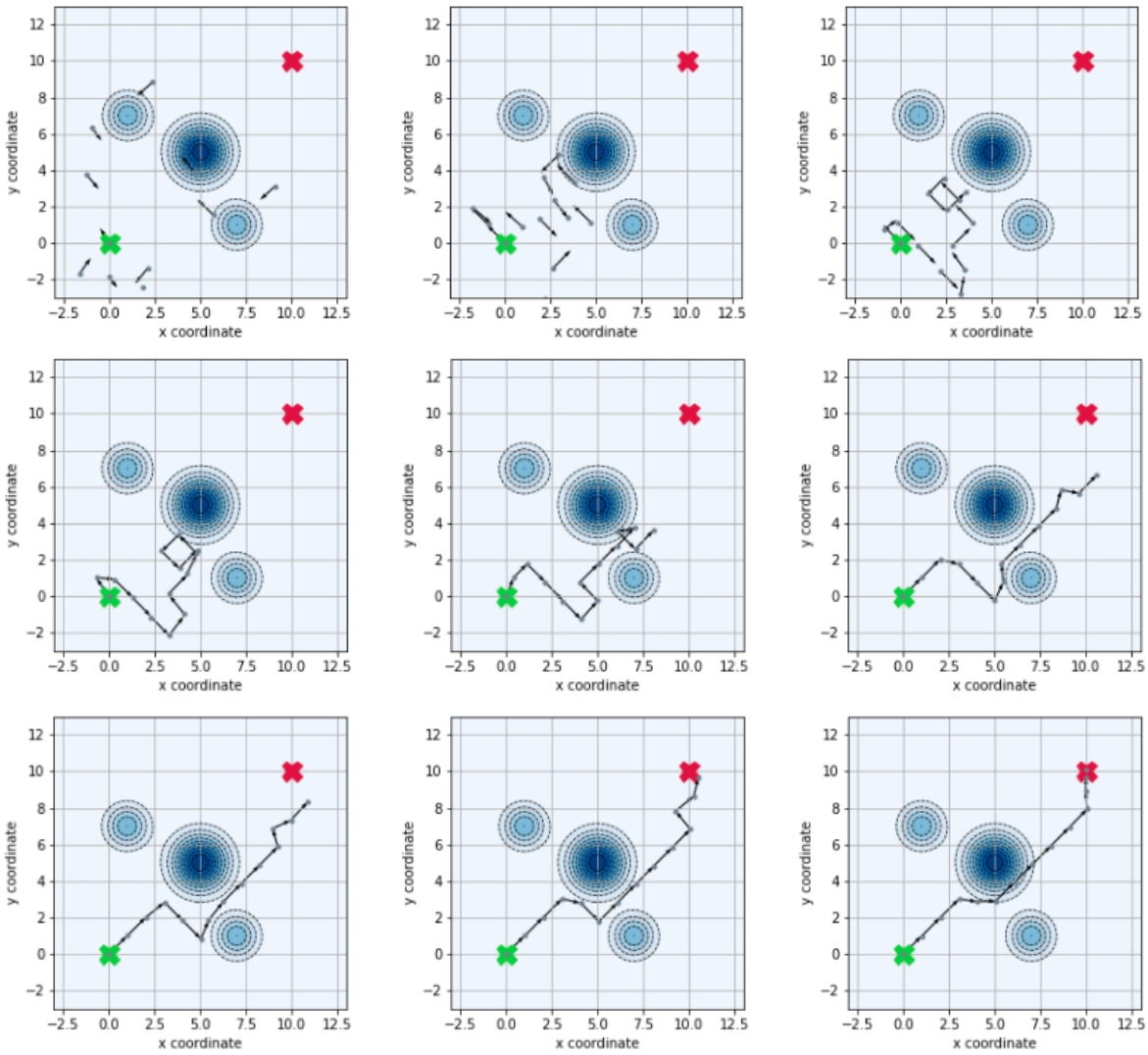


Figure 4.8: *Evolution of trajectories during training in Navigation 2D for the collocation method. (top row) at beginning of training the dynamics constraints are violated leading to unconnected paths; (middle row) after 200 iterations the dynamics are no longer violated and the overall trajectory starts to move towards the goal; (bottom row) the optimization converges to a feasible plan that minimizes the total distance to the goal.*

In addition the previous experiments, we provide a comparison across different training hyper-

parameters to show the impact of learning rates and optimizers in the shooting formulation for the Navigation 2D problem. In particular, we show in Figure 4.9 the effect of different values of learning rates for the most used optimizers in Deep Learning (i.e., Adam, RMSprop, and SGD) in a setting of limited training iterations (only 50 iterations). We can observe that the RMSProp optimizer performs best among other optimizers in general. Specifically, we see that a very aggressive learning rate (i.e., lr=1e-1) can be used to obtain a very fast convergence in only 50 iterations.

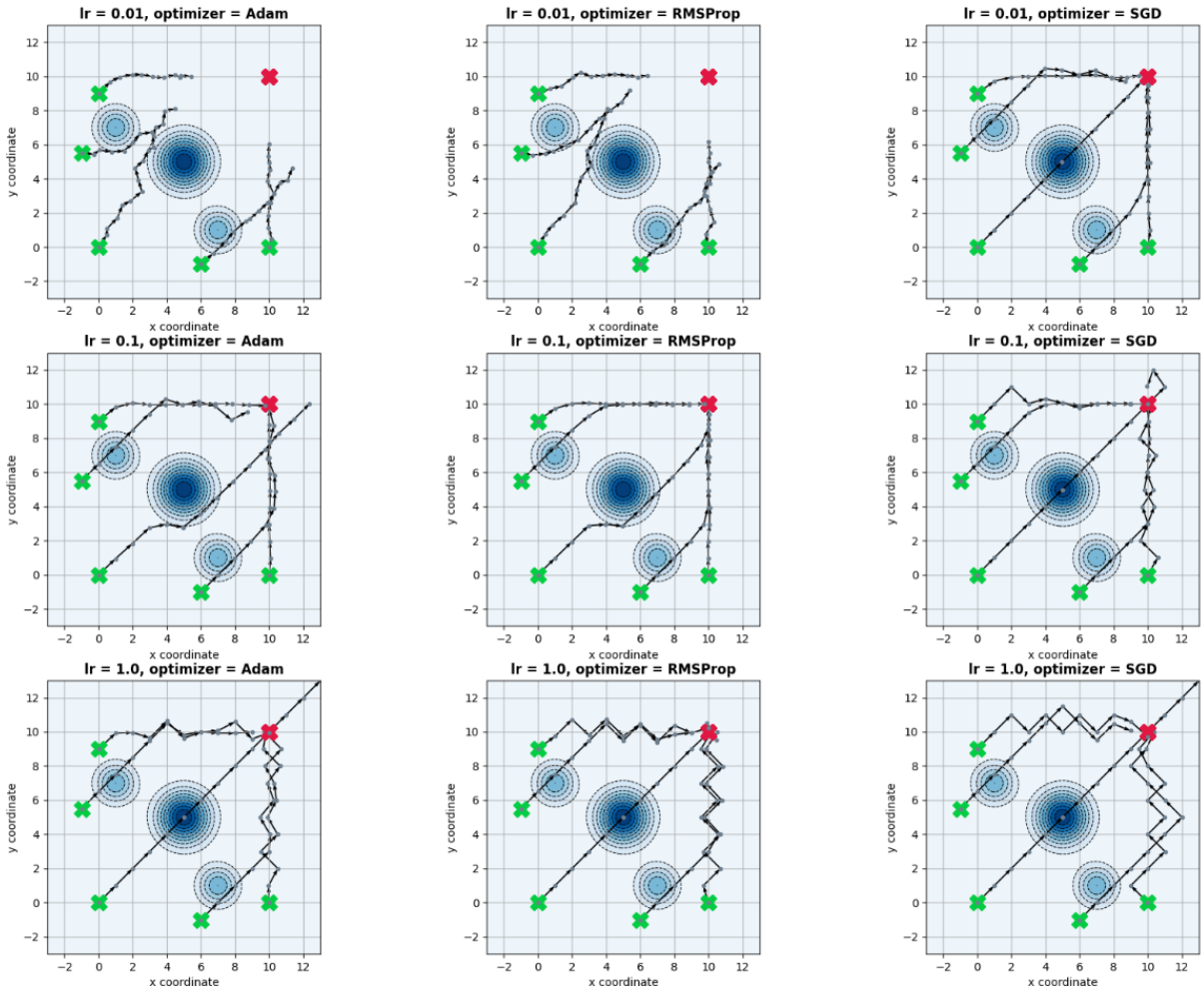


Figure 4.9: *Optimization hyper-parameters: learning rate vs. optimizer (iterations=50)*

To show the training behavior of the gradient-based planning algorithm, we plot in Figures 4.10 and 4.11 the training curves for a Navigation problem with start position $\mathbf{x}_0 = (0.0, 0.0)$. We observe that for this particular problem the learning rate can be tuned around 0.1 for a smooth, but fast convergence. We notice that the basic SGD optimizer attains a very sharp convergence, however, this behavior should be avoided for more complex problems as it can easily lead to divergence.

4.4.2 Comparisons in Linear-Quadratic Tasks

In this section, we evaluate the performance of planning through backpropagation in LQR problems, as described in Definition 2.2.7. We recall that the linear dynamics and the quadratic

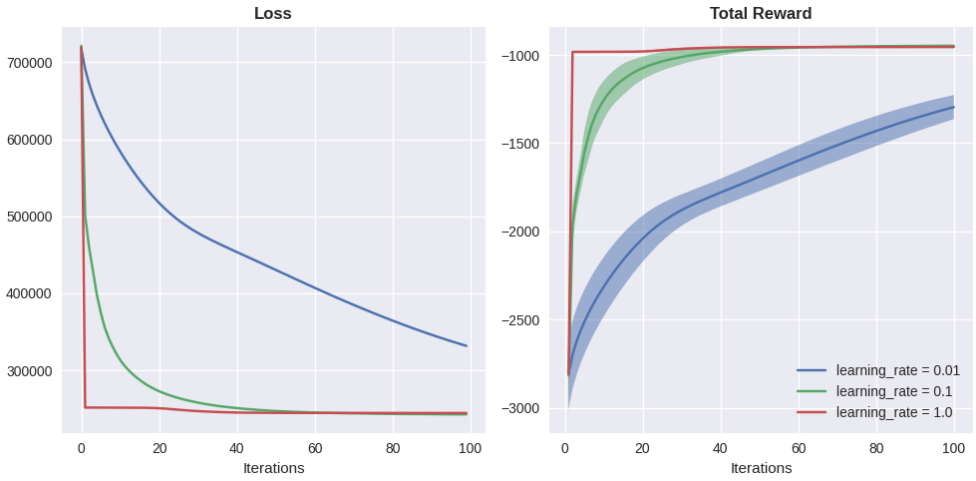


Figure 4.10: *Navigation 2D: Training curves for different learning rates with the RMSProp optimizer: (left) the loss function and (right) the total reward averaged across the batch (solid line corresponds to the mean value, and transparent band to 1 standard deviation).*

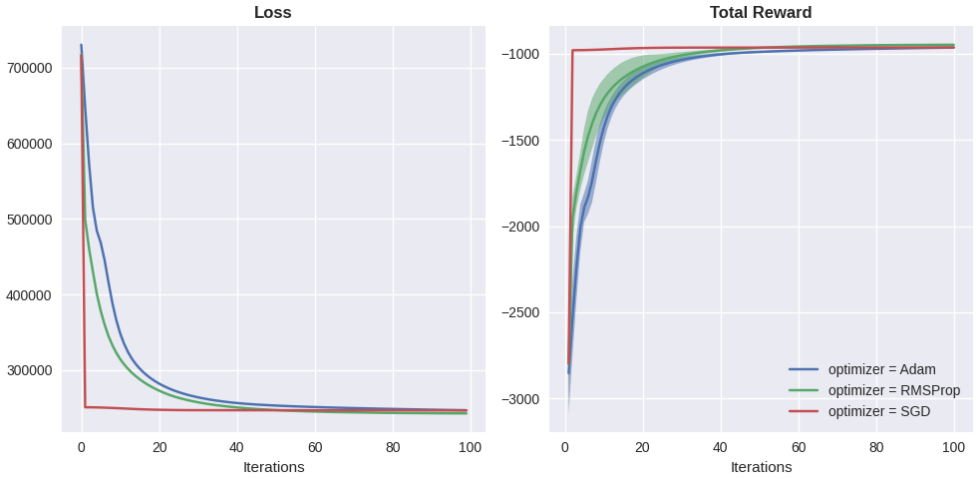


Figure 4.11: *Navigation2D: training curves for different optimizers with learning rate = 1e-1. (left) the loss function and (right) the total reward averaged across the batch (solid line corresponds to the mean value, and transparent band to 1 standard deviation).*

cost function of LQR problems are given by the following equations:

$$f(x,u) = Ax + Bu \ , \tag{4.36}$$

$$g(x,u) = x^\top Qx + u^\top Ru \ , \tag{4.37}$$

where Q is a positive definite matrix and R is a positive semidefinite matrix.

For the empirical experiments, we synthetically generated LQR problems in the following fashion. Let n and m be the number of the state and action dimensions, respectively. Let $\mathbf{I}_n$ be the identity matrix of size n, $\mathbf{N}(n,m)$ be an $n \times m$ matrix of normally distributed elements $\mathbf{N}_{ij} \sim \mathbb{N}(\mu = 0, \sigma = 1)$, c_u be a control-cost coefficient, and h be the discretization step. The

Table 4.1: *Optimality Gap in LQR problems: comparison between trajectory optimization methods of shooting and collocation. Total cost is averaged over 50 problems and the optimality gap %Δ is the relative difference w.r.t. to the optimal cost.*

	LQR	**Shooting**		**Collocation**	
Instance	**Optimal Cost**	**Total Cost**	%Δ	**Total Cost**	%Δ
$n = 5$, $m = 1$	1.82211	1.82304	0.051	1.82301	0.049
$n = 5$, $m = 5$	1.24731	1.25538	0.647	1.25544	0.652
$n = 10$, $m = 1$	35.46004	35.46100	0.002	35.46081	0.002
$n = 10$, $m = 5$	10.53660	10.56196	0.240	10.56542	0.273
$n = 10$, $m = 10$	1.32250	1.58230	19.64	1.54401	16.74
$n = 25$, $m = 5$	25.17057	28.30125	12.43	28.74188	14.18
$n = 25$, $m = 10$	11.23706	12.31894	9.62	12.14420	8.07
$n = 25$, $m = 25$	1.91856	2.43143	26.73	2.46386	28.42

parameters of the LQR tasks were then generated as follows:

$$A = \mathbf{I}_n + h\mathbf{N}(n, n) \tag{4.38}$$

$$B = h\mathbf{N}(n, m) \tag{4.39}$$

$$Q = h\mathbf{I}_n \tag{4.40}$$

$$R = c_u h\mathbf{I}_m \tag{4.41}$$

Also, the initial state was sampled from the normal distribution, $x_0 \sim \mathbf{N}(n, 1)$ and the actions are bounded to the interval $[-1, 1]$, and the horizon was fixed to 100 timesteps.

Optimality Gap. In this set of experiments, we ran both the shooting and the collocation version of TensorPlan and compare them with the optimal plans obtained via the analytical LQR solution obtained via the Riccati equations according to the Theorem 2.2.6. We generate 50 problems according to the sampling procedure outlined above for each pair (n, m) and averaged the results. We trained both trajectory optimization methods for 1000 iterations with RMSProp and with batch size 256 and learning rate 3e-3. Table 4.1 shows the obtained results for several LQR instances. We can observe that as the size of the LQR increases, the problem becomes harder for both optimization formulations, starting at almost no optimality gap for smaller instances and going up to 26.73% and 28.42% gaps for the shooting and collocation methods, respectively. We remark that as the number of actions m increases within the set of problems with the same number of states n, the problem becomes more likely to be controllable[4] and therefore the total costs naturally decrease.

Trajectory analysis. In this experiment we analyze the near optimal trajectory obtained with TensorPlan and compare it to the random plan and no-op plan baselines in order to give a sense of how planning through backpropagation can deal with high-dimensional problems. To show results we randomly chose an LQR problem with $n = 50$ and $m = 40$ and fix it for all results shown in this section. Initially, we compare the behavior of the solution plan obtained via TensorPlan with a no-op plan and a random plan. Figure 4.12a shows the state-action trajectory along with the

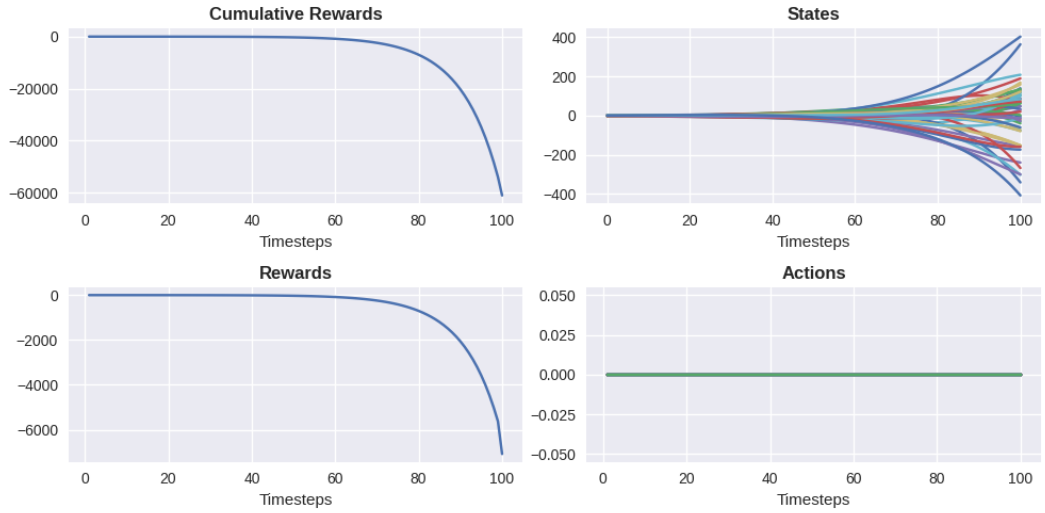
[4]There are a number of definitions of controllability in optimal control (Vinter and Vinter, 2010). Here, we consider the basic definition that an LQR is controllable if a linear controller can steer the system to any desired state in a fixed number of timesteps depending on the number of state variables.

stage-wise cost and cumulative cost for the 0-action plan, i.e., a sequence of no-ops. We can observe that after 60 time steps the LQR system starts to divergence, meaning that the system is not inherently stable. A similar observation can be made for the random plan as shown in Figure 4.12b. In contrast to the no-op and random plan cases, TensorPlan manages to stabilized the system as it can be seen in in Figure 4.12c. As a matter of fact, when planning through backpropagation is used, a very complex sequence of actions can quickly regulate the system around its equilibrium. Note that at the first glance the actions generated with TensorPlan seem quite random when visually compared to the actions of the random plan. However, as it can be seen in the state trajectories, the solution obtained manages to keep the state variables within the interval [-3.0, 3.0] for most of the trajectory, whereas the states visited by the random plan diverges to more than 300 near the end of the trajectory. Also, we notice the time-dependent character of the solution of TensorPlan as the last actions are quickly reduced to zero to minimize its costs as the final states of an already-stabilized system cannot diverge too much in a linear system as the LQR problem.

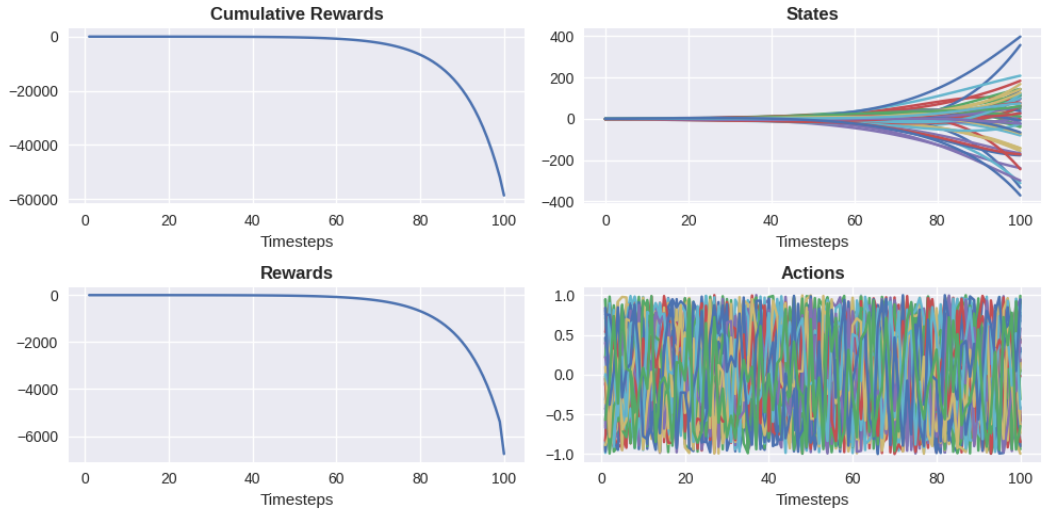
4.5 Conclusion

In this chapter, we introduced the idea of differentiable planning for deterministic sequential decision-making problems in continuous spaces. The only requirement for the applicability of the algorithms discussed here is that the system's dynamics and cost function need to be differentiable, thus implying a computation graph whose input parameters can be optimized via gradient descent methods. We proposed to reinterpret the idea of planning through backpropagation as introduced in Wu *et al.* (2017) in terms of trajectory optimization in optimal control. As a direct consequence, we pointed out that the original implementation of TensorPlan is equivalent to solving the shooting formulation via gradient descent and showed that we can obtain similar empirical results with the alternative direct transcription formulation. Finally, we presented novel experiments for TensorPlan in a 2D path finding problem and analyzed the optimality gap of planning through backpropagation in LQR tasks, hinting that with the appropriate training hyperparameters the algorithm can be arbitrarily close the optimal solution in such problems.

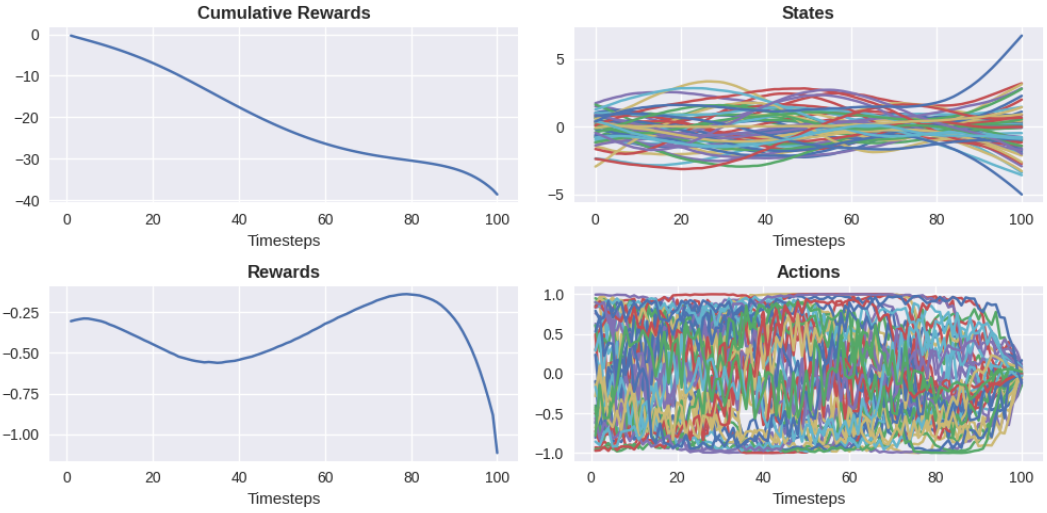
In the following chapter, we start to extend the ideas presented here to the more general setting of stochastic nonlinear domains.



(a) *No-op (baseline)*



(b) *Random plan (baseline)*



(c) *TensorPlan*

Figure 4.12: *Single run of LQR with $n = 50$ and $m = 40$: a trajectory comparison between a random plan, the no-op plan, and the near-optimal plan generated by TensorPlan.*

Chapter 5

Training Deep Reactive Policies in Stochastic Continuous Domains

In the last chapter, we discussed the framework of planning through backpropagation as a promising model-based planning approach capable of handling domains exhibiting arbitrary non-linear dynamics and cost functions, provided the underlying control problem could be completely defined with differentiable functions. Though somewhat flexible in terms of the class of continuous domains it can handle, the gradient-based planning formulation presented so far can only directly handle deterministic systems, i.e., tasks that do not involve any notion of uncertainty, either in the effects of the agent's actions, as in a standard MDP, or in exogenous events or disturbances.

The goal of this chapter is to start extending the capabilities of planning through backpropagation to stochastic domains, i.e., to solve continuous MDP problems. Overall, the issue with applying gradient-based trajectory optimization (and consequently planning through backpropagation) to stochastic domains is similar to the difficulties of applying deterministic planners to decision problems under uncertainty: no single previously-planned sequence of actions can achieve an optimal total cost in all situations. In other words, to solve sequential decision problems whose future outcomes depend on the realization of random events, an agent needs a contingency mechanism to be able to cope with unpredictable effects during execution.

Two natural solutions to implement such contingencies have been extensively studied in the AI planning literature: (*i*) *closed-loop policies* and (*ii*) *open-loop policies*. Recall that a policy in the context of a fully observable, perfect information problem is a simple mapping between states and actions. A closed-loop policy is the final solution of an offline planning algorithm that attempts to reason over the entire state space (or at least the relevant states w.r.t a set of optimal policies). Unlike closed-loop policies, an open-loop policy is the incremental solution of a planning algorithm that focus its computation on finding the best possible action for the current state, exclusively. Therefore, closed-loop policies are typically obtained via offline planning and then deployed for online execution without any further updates. In contrast, open-loop policies can only be implemented in decision time, through the interleaving of planning and execution[1].

Currently, state-of-the-art planners for finite horizons are in large part based upon online search (Bonet and Geffner, 2012; Cui *et al.*, 2015; Keller and Eyerich, 2012), thus implementing some form

[1]In optimal control, closed-loop policies are akin to feedback controllers whereas open-loop policies are related to model-predictive control.

of open-loop policy. Even though capable of handling very large decision problems, such planners usually need non-trivial computation time per decision step, which might make them difficult to apply in practical settings requiring fast decisions. We postpone further discussions of a possible online adaptation of planning through backpropagation to the next chapter. Additionally, it is important to note that search-based planners currently can only be effective in handling domains defined over discrete state and action spaces, therefore they are not directly applicable to the stochastic continuous domains. This motivates our investigation of training reactive policies with planning through backpropagation as a way to achieve fast decision-making in stochastic continuous domains.

This chapter is organized as follows. We start the next section by presenting the method of policy search via gradient-based optimization for training deep reactive policies in contrast to the trajectory optimization formulation leveraged by planning through backpropagation in deterministic domains. Following that, we address the modeling issues that naturally appears in any gradient-based planning method that attempts to solve stochastic problems and the challenges associated with approximation in policy space with deep reactive policies. We conclude the chapter with a number of experiments and a preliminary investigation of the combination of deep reactive policies with online planning that will naturally lead to the developments that shall be presented in the next chapter. We remark that the exposition here is mostly based on our work (Bueno *et al.*, 2019) with some extended discussion and additional experiments.

5.1 Policy Search via Gradient-Based Optimization

As introduced in the previous section, an alternative to apply planning through backpropagation to stochastic domains is to give up the idea of trajectory optimization in favor of more general policy search methods. The overall approach of policy search via gradient-based optimization can be summarized in the following steps:

1. explicitly encode a policy using a parametric function, π_θ,
2. embed the policy π_θ in a SCG approximating $V^{\pi_\theta}(x_0)$,
3. define a suitable objective function $J(\theta)$, and
4. optimize the policy parameters θ via stochastic gradient descent over $J(\theta)$.

Inspired by the recent successes in Deep Learning in perception tasks (Hinton *et al.*, 2012a; Krizhevsky *et al.*, 2012; Sutskever *et al.*, 2014), game applications (Mnih *et al.*, 2013; Silver *et al.*, 2016), and other hard problems that arise in artificial intelligence and machine learning where large datasets are available, we turn our attention to the use of deep neural networks as parametric function approximators. We shall denominate the policies parametrized as neural networks in this chapter by *Deep Reactive Policies* (DRPs) for reasons that we will discuss in the following sections. We notice that despite the remarkable recent applications of neural networks in computer vision, speech recognition and other classification/regression tasks, control theory was historically amongst the earliest adopters of such function approximators (Barto *et al.*, 1983; Doncieux *et al.*, 2004; Narendra and Parthasarathy, 1990; Nguyen and Widrow, 1990; Rickard and Bartholomew, 1993) in complex nonlinear systems. At least from a high-level point of view, these results motivate

the investigation of neural networks for planning given their demonstrated capabilities in inducing powerful domain-independent features.

The theoretical challenge in training neural networks to approximate optimal policies lies in the fact that the data needed to optimize a possibly large set of parameters is somewhat harder to obtain in contrast to the traditional static setting in supervised learning. Indeed, a DRP must operate not on i.i.d. samples, but should be able to exploit a learning signal defined on temporally-correlated data that is usually generated in a closed-loop system. This dynamic optimization setting can easily amplify over time existing approximation and sampling errors and consequently cause severe instabilities during training. Another challenge is the fact that in high-dimensional continuous state spaces, there is always a real risk to over-fitting and poor generalization since only a very small subset of relevant states can be typically visited in simulation. Therefore, we remark that the successful application of DRPs in stochastic continuous planning problems is not due to the blind application of off-the-shelf tools and deep learning architectures, but necessitates specialized training procedures where the dataset of trajectories and the network parameters are optimized together.

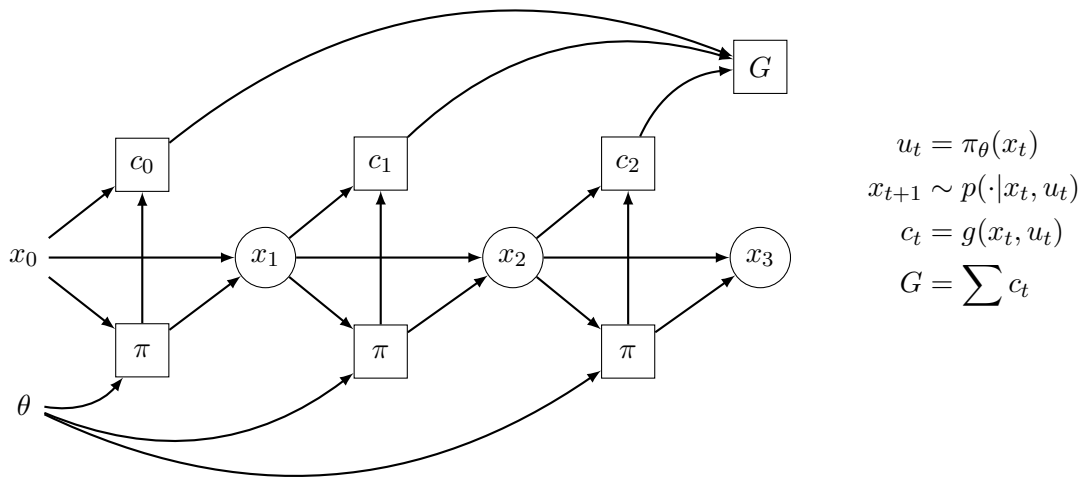


Figure 5.1: *Stochastic computation graph of an MDP and DRP: the node G represents the total return obtained by unrolling in time the transition probability, $p(\cdot|x_t, u_t)$, and cost function, $g(x_t, u_t)$, for the DRP π_θ. Rounded and squared nodes represent stochastic and deterministic nodes.*

5.1.1 Monte-Carlo Simulations

In this chapter, we explore a class of solutions to the dynamic optimization of parametric policies that are based on Monte-Carlo simulations. The key observation is that the stochastic sequential decision-making problems we will interested in solving can be naturally represented as the SCG shown in Figure 5.1. In comparison with Figure 4.2 for the deterministic control problem solved in the previous chapter, we notice three important changes: (1) the transition is now stochastic, (2) we now optimize a policy (i.e., a mapping between states to actions), and (3) the optimization parameters θ are now shared across the timesteps representing a stationary policy. From the viewpoint of SCGs, the interpretation is that a recursive composition of the dynamics and the policy parametrized as a DRP can symbolically implement a random return $G = \sum_t g(x_t, \pi_\theta(x_t))$ that

can in turn be aggregated through Monte-Carlo integration to estimate a performance metric of the current policy. In the simplest case, we can consider the Monte-Carlo approximation of the value function of the start state, x_0, for the current policy, π_θ:

$$V^{\pi_\theta}(x_0) = \mathbb{E}_{\pi_\theta}[G|x_0] \approx \frac{1}{N}\sum_{i=1}^{N} G^i \ . \tag{5.1}$$

We note that, in principle, any non-increasing function of the value function V^{π_θ} could be used as a surrogate objective function $J(\theta)$. Specifically, for the particular kinds of cost-based problems we are interested in this thesis, we use the Mean-Square Error (MSE):

$$J(\theta) = \mathbf{MSE}[V^{\pi_\theta}(x_0), 0] = \mathbb{E}\left[\left(\sum_{t=0}^{H-1} g(x_t, \pi_\theta(x_t))\right)\right]^2 \ . \tag{5.2}$$

The MSE objective function can considerably accelerate the optimization process as the gradients of the objective function w.r.t. the policy parameters will have their norms conveniently scaled up in the beginning of the training when the total cost is most-likely far from its minimum value. Moreover, we shall use Monte-Carlo sampling as a tractable approximation of the expectation in the objective function:

$$J(\theta) \approx \hat{J}(\theta) = \left(\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H-1} g(x_{i,t}, \pi_\theta(x_{i,t})\right)^2 , \tag{5.3}$$

where i is the index of the state-action trajectory, N is the number of trajectories, t is the time step and H is the horizon. In order to optimize the policy parameters θ in the SCG of Figure 5.1, we shall use an optimization method based on stochastic gradient descent:

$$\theta_{(k+1)} = \theta_{(k)} - \eta \ \nabla_\theta \hat{J}(\theta)|_{\theta=\theta_{(k)}}, \tag{5.4}$$

where η is the learning rate.

Algorithm 7: Planning through Backpropagation with Deep Reactive Policies

Input: MDP $\mathcal{M}$, start state $\bar{x}$, horizon H, number of trajectories N, iterations K
Output: policy $\pi_\theta \approx \pi^*$

$\mathcal{M}_\Xi \leftarrow$ Re-Parametrization$(\mathcal{M})$; ▷ SCG
$\pi_\theta \sim q(\mathbb{R}^d)$; ▷ Policy initialization
for $k = 0, \ldots, K-1$ **do**
  for $i = 0, \ldots, N-1$ **do**
    Sample noise $\xi \in \Xi$
    $(\tau^i_{0:H}, G^i) \leftarrow$ Forward$(\mathcal{M}_\Xi, \bar{x}, \pi_\theta, \xi)$; ▷ Simulation
  $\hat{J}(\theta) \leftarrow \frac{1}{N}\sum_{i=1}^{N}\left(G^i\right)^2$; ▷ Monte-Carlo approximation
  $\nabla_\theta \hat{J}(\theta) \leftarrow$ Backward$(\tau_{0:H}, \hat{J}(\theta))$; ▷ Backpropagation-Through-Time
  $\theta \leftarrow$ Update$(\theta, \nabla_\theta \hat{J})$; ▷ Optimization
return π_θ

5.2 Backpropagating Gradients in Stochastic Computation Graphs

At this point, the only remaining question is how to obtain the performance gradient $\nabla_\theta \hat{J}(\theta)$ as the backpropagation-through-time algorithm can only operate in deterministic computation graphs according to the discussion in Chapter 3. For that matter, we will discuss in the next section how a pathwise gradient estimator can be derived for a particular class of MDPs, thus leading to an efficient implementation of the backward pass step of the Algorithm 7. We subsequently discuss the formulation of exogenous Markov Decision Processes (X-MDPs) as a generalization of the re-parametrization trick that will allow DRPs to be trained by gradient-based planning for an important class of stochastic nonlinear domains.

Before we proceed to the next section, notice that the overall structure of the Algorithm 7 for training DRPs follows the general framework of differentiable planning as first outlined in Algorithm 5, except that the original MDP is initially re-parametrized, the simulation step first sample the noise variables ξ before running the forward pass in the SCG, and the objective function is now given by the Monte-Carlo approximation $\hat{J}$.

5.2.1 Re-Parametrization Trick

Policy search via gradient-based optimization relies on the fact that a differentiable model of the system dynamics and cost functions is available. In this case, a backpropagation-through-time algorithm can be derived to estimate a descent direction to update the policy's parameters. An important complication, however, is that the chain rule of derivatives can only be propagated along deterministic paths in the underlying SCG of the MDP which in principle disallows a direct estimation of the gradient of an expected return, e.g., $\nabla_\theta \mathbb{E}_{\pi_\theta}[G|x_0]$.

Indeed, if we recall the semantics of a stochastic node in the computation graph, we can conclude that it may not be possible to obtain the gradient of a stochastic node's sample w.r.t. its input parameters for arbitrary distributions. As an example, consider a categorical distribution over k values parametrized by logits $\rho \in \mathbb{R}^k$. It is a known fact that there is no easy way to obtain unbiased gradients as a categorical discrete sample is a discontinuous function of the logits[2].

However, there is an important class of continuous distributions for which differentiable sampling schemes have been successfully derived (Mohamed *et al.*, 2020). According to the literature of machine learning, we say that such distributions leverage the so-called *re-parametrization trick* (Kingma and Welling, 2014). Formally speaking, we say that a distribution $p(y|\phi)$ is re-parametrizable if there exist (i) a function $z(\cdot;\phi)$ differentiable in the parameters ϕ, and (ii) a canonical distribution $p_\xi(\cdot)$ independent of y and ϕ such as a sample of the random variable $y \sim p(\cdot|\phi)$ can be obtained by the following equivalent sampling process:

$$y \sim p(\cdot|\phi) \quad \Leftrightarrow \quad y = z(\xi;\phi),\ \xi \sim p_\xi(\cdot)\ . \tag{5.5}$$

A classical example is the Gaussian distribution that can naturally be re-parametrized as:

$$y \sim \mathcal{N}(\cdot|\mu,\sigma) \quad \Leftrightarrow \quad y = \mu + \sigma\xi,\ \ \xi \sim \mathcal{N}(0,1)\ . \tag{5.6}$$

[2]We note that continuous relaxations of categorical distribution (Jang *et al.*, 2017; Maddison *et al.*, 2017) have been proposed to obtain gradient estimators of categorical random variables, however, these gradients are only unbiased in the limit and typically exhibit high variance.

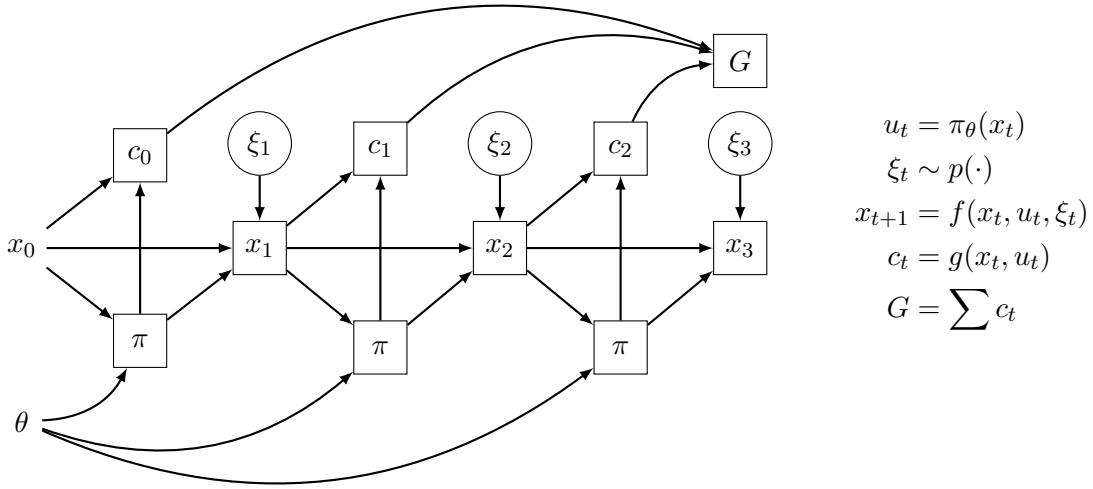


Figure 5.2: *Re-parameterization trick applied to the SCG of an MDP: the source of randomness in the transitions is now independent of the previous states and actions, and as a result, the next state x_{t+1} becomes deterministically-defined by the previous state x_t, the current action recommended by the policy $u_t = \pi_\theta(x_t)$, and the external noise variable ξ_t.*

5.2.2 Pathwise Gradient Estimator

The important property of the re-parametrization trick, as illustrated above for the Gaussian random variable, is the fact that it implements an equivalent sampling procedure through a deterministic and differentiable sequence of transformations over an independent source of randomness. Henceforth, it becomes possible to simply backpropagates the gradient computation through the sampling procedure itself in order to obtain a *pathwise gradient* estimator (Figurnov *et al.*, 2018; Jankowiak and Obermeyer, 2018; Ruiz *et al.*, 2016).

Consequently, if we assume that the dynamics of an MDP is defined by a re-parametrizable distribution, i.e., $x_{t+1} \sim p(\cdot|x_t, \pi_\theta(x_t)) \Leftrightarrow x_{t+1} = f(x_t, \pi_\theta, \xi_t)$, where $\xi_t \sim p(\cdot)$ in an independent random noise, then we can transform the SCG of Figure 5.1 into the equivalent SCG of Figure 5.2. It becomes clear that after the application of the re-parametrization trick, a backpropagation-through-time algorithm can now be directly applied to the MDP as all the paths in its computation graph from the policy parameters θ to the return G do not contain stochastic nodes.

Algebraically speaking, the pathwise gradient estimator applied to an MDP whose dynamics is re-parametrizable implies that the gradient of the expected return can be obtained by pushing the gradient operator inside the expectation:

$$\nabla_\theta V^{\pi_\theta} = \nabla_\theta \mathbb{E}_{x_{t+1} \sim p(\cdot|x_t, \pi_\theta(x_t))} \left[\sum_{t=0}^{H-1} g(x_t, \pi_\theta(x_t)) \right] \tag{5.7}$$

$$= \mathbb{E}_{\xi_t \sim p(\cdot)} \left[\nabla_\theta \sum_{t=0}^{H-1} g(x_t, \pi_\theta(x_t)) \;\middle|\; x_{t+1} = f(x_t, \pi_\theta, \xi_t) \right] . \tag{5.8}$$

We remark that the swapping between the operators of expectation and the gradient is sound only because, after the reparametrization, the distribution the expectation is taken with respect to

no longer depends on any random variables that are directly or indirectly dependent on the policy parameters θ. This is essentially the same result developed in Section 3.2.2.

5.3 Exogenous Markov Decision Process

From the viewpoint of backpropagation-through-time, the main benefit of the re-parametrization trick is that the gradient of the policy performance can be back-propagated through the dynamics as the source of randomness is now external and independent of the states and actions. In other words, we can therefore pre-sample all random variables ξ_t in the SCG of Figure 5.2 before committing to selecting actions. This is in close connection to the idea first proposed in the PEGASUS (Policy Evaluation-of-Goodness And Search Using Scenario) (Ng and Jordan, 2000) algorithm of reducing the original (PO)MDP problem to a deterministic simulative model in which all the uncertainty is specified in the initial state distribution. In PEGASUS, the state space is augmented with the additional noise variables pre-sampled, $\tilde{x}_t = (x_t, \xi_{t:H})$, and the transition consumes the current noise instantiation ξ_t inducing a deterministic next state, $\tilde{x}_{t+1} = (f(x_t, u_t, \xi_t), \xi_{t+1:H})$.

The natural interpretation of the re-parametrization in the context of sequential decision-making sudden becomes clear: the pre-sampled noise variables define a simulation scenario for the decision process. Hence, the goal of an optimal policy is to find a set of contingencies for all possible scenarios given by the valid instantiations of the noise variables $\xi_{0:H}$. In addition, it is now possible to notice that it is not necessary that an external noise variable ξ_t be completely independent from all other variables in the computation graph. Indeed, the key requirement is that the noise variables forming the simulation scenario must only be independent from the effects of the actions recommended by the policy. In other words, policy search via gradient-based optimization can be applied to the more general class of X-MDP models, i.e., decision processes in which the uncertainty in transitions is defined by an exogenous event process. Definition 5.3.1 formalizes the components of an X-MDP[3].

Definition 5.3.1. (Exogenous Markov Decision Process) *An X-MDP is defined by the tuple* $\mathcal{M}_\Xi = (\mathcal{X}, \mathcal{U}, \Xi, p_\Xi, f, g, \bar{s})$, *where* $\mathcal{X} \subseteq \mathbb{R}^n$ *is the* state space, $\mathcal{U} \subseteq \mathbb{R}^m$ *is the* action space, $\Xi \subseteq \mathbb{R}^p$ *is the* noise space, $p_\Xi \colon \Xi \to \mathcal{P}(\Xi)$ *is the* noise process distribution, $f \colon \mathcal{X} \times \mathcal{U} \times \Xi \to \mathcal{X}$ *is the* transition function, $g \colon \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ *is the* cost function, *and* $\bar{x}$ *is a* start state. *Additionally, we denote by* $\mathcal{U}(x) \subseteq \mathcal{U}$ *the set of valid actions in state* x.

Figure 5.3 shows the SCG of an X-MDP. Notice that there are no paths from the policy parameters θ to the trajectory return G that goes through the random noise variables ξ_t implying that gradients can flow directly through the system's dynamics therefore allowing backpropagation-through-time to be used to optimize the policy.

5.4 Approximation in Policy Space with Deep Reactive Policies

Policy search via gradient-based optimization with Deep Reactive Policies is an example of an approximation in policy space method. Initially, a random, most-likely low-performance policy is instantiated. Then, several derived policies are generated in turn through a sequence of hopefully

[3]We adapt the terminology of X-MDPs that was initially proposed in the context of anticipatory sampling methods in operations research (Mercier, 2009; Mercier and Hentenryck, 2007).

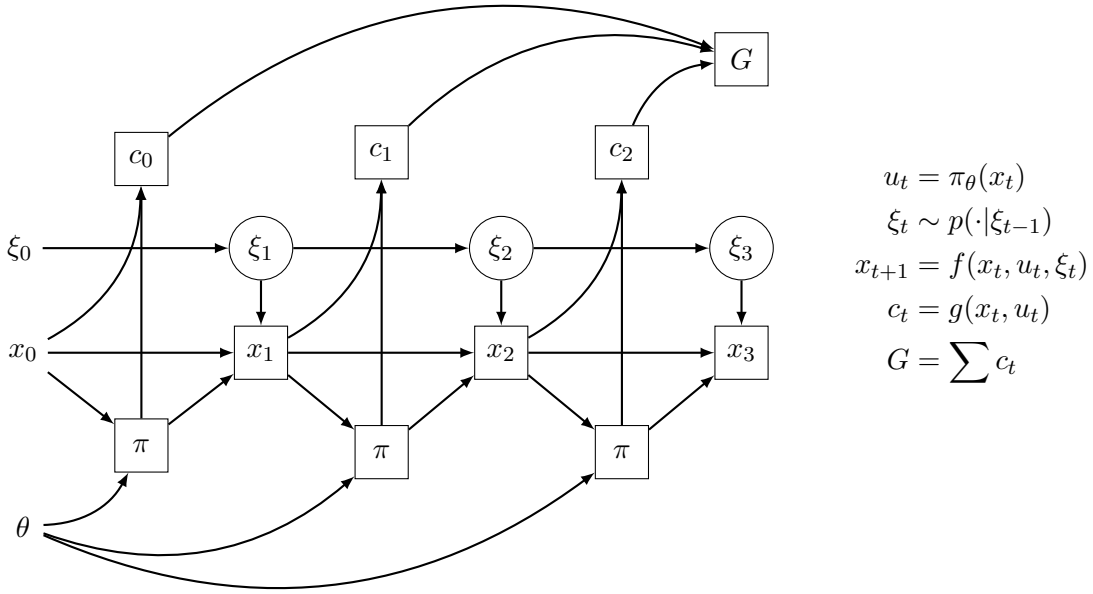


Figure 5.3: *Stochastic computation graph of an X-MDP: the state trajectory x_t is induced by the actions recommended by the policy π_θ and by the external scenario specified by the realization of the exogenous random process, $\xi_{t+1} \sim p(\cdot|\xi_t)$.*

improving iterations. Practically speaking, in the current context, a sequence of policy parameters θ^k, for $k = 0, 1, \cdots, K-1$, is obtained by gradient descent steps over the surrogate objective $\tilde{J}(\theta)$.

As for any approximation in policy space algorithm, special care must be taken to define the class of policies over which the sequential optimization must be performed. We notice that representing a policy by different parametric functions can have considerable impact both on the training time and in the capacity of approximating the optimal behavior in the given MDP. Of course, a number of alternatives can be used in the context of policy search via gradient-based optimization. Usual choices vary from simple linear models to radial basis functions and deep neural nets. The only hard requirements are that the parametric policy must be quickly evaluated to select an action and that the function's outputs must be differentiable w.r.t. its parameters for all input states.

For the purposes of this chapter, we focus on the use of Deep Reactive Policies as the function approximator of choice for the parametrization of policies. Generally speaking, a reactive policy is a model that attempts to explicitly represent a mapping from states to actions. It should therefore be an efficient mechanism to select a valid action given any valid state. Additionally, we denote by a deep policy any deep learning-based model built from the stacked layers of feed-forward neural nets or other differentiable architectures.

Figure 5.4 shows the graphical representation of a deep feed-forward neural net whose input is the state vector x_t and output is the action vector u_t. The hidden layers of the policy network typically use ReLU-based units for improved convergence, except the last layer that might use specialized action layers based on an squashing function (e.g., the tanh activation function) to constrain the set of valid actions:

$$\mathbf{h}_{l+1} = \begin{cases} \text{ReLU}(W_l\ \mathbf{h}_l + b_l), & l = 1, \cdots, L-1 \quad (5.9a) \\ \psi(\tanh(W_l\ \mathbf{h}_l + b_l)), & l = L\ . \quad (5.9b) \end{cases}$$

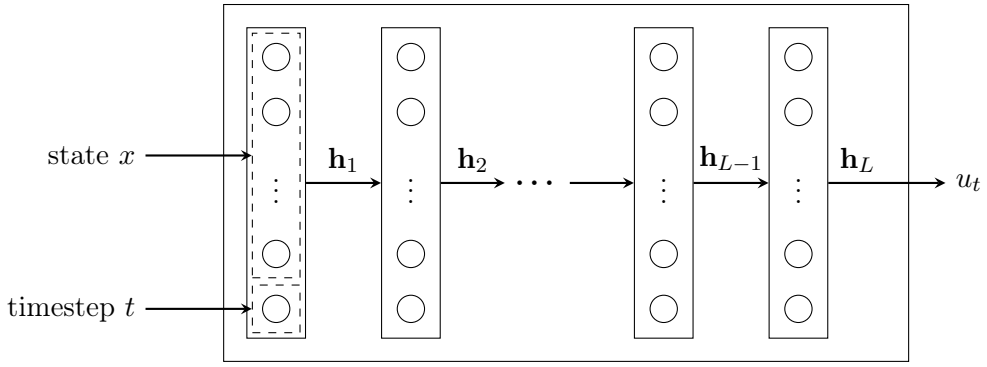


Figure 5.4: *A neural network representation of a DRP: the input layer of the policy network corresponds to the current state, x_t, and the output layer corresponds to the action variables, u_t.*

Currently, most works on DRPs in AI planning have focused on discrete stochastic policies (Groshev *et al.*, 2018; Issakkimuthu *et al.*, 2015; Toyer *et al.*, 2018). Instead, we recall that our aim here is to train continuous deterministic DRPs. Even though the general idea of an efficient action selection mechanism enabling fast decision-making remains the same, some special care is needed in order to extend DRPs to continuous domains. In particular, we identify the need for handling action constraints and state normalization in order to obtain well-behaved gradients.

5.4.1 Box-Constrained Action Spaces

It is common in continuous action spaces to constrain the valid values of the action fluents. Therefore, at the very least, a DRP should be able to accommodate action fluents whose range are either an open, a closed or a half-open/half-closed interval of the reals. An alternative to constrain $u_t = \pi_\theta(x_t)$ to be within the bounds $[\mathbf{l}, \mathbf{u}]$ is to apply a transformation on the logits z^L of the output layer of the DRP such that:

$$\pi(x_t) = \begin{cases} \frac{(\mathbf{l}+\mathbf{u})}{2} + \frac{(\mathbf{u}-\mathbf{l})}{2} \tanh(z^{(L)}), & \text{if } \mathbf{l} > -\infty \text{ and } \mathbf{u} < +\infty \\ \mathbf{l} + \exp(z^{(L)}), & \text{if } \mathbf{l} > -\infty \text{ and } \mathbf{u} = +\infty \\ \mathbf{u} - \exp(-z^{(L)}), & \text{if } \mathbf{l} = -\infty \text{ and } \mathbf{u} < +\infty \\ z^{(L)} & \text{otherwise,} \end{cases} \tag{5.10}$$

where tanh is the hyperbolic tangent squashing function and $\mathbf{l}$ and $\mathbf{u}$ denote lower and upper bound vectors, respectively.

5.4.2 State Normalization

In addition to handle continuous outputs representing continuous action fluents, a DRP in continuous domains shall pay special attention to the range of the state fluents. It is a well-known issue in training neural nets that the distribution of a layer's activation can vary widely as a function of its previous inputs, which can significantly slow down the training, especially in the case of saturating nonlinearities. This phenomenon often referred to as internal covariate shift can be particularly pronounced in DRPs for continuous spaces given that state fluents can vary by many orders of magnitude, which is in direct contrast to discrete domains that typically deal with

state boolean variables. Recently, layer normalization (Ba *et al.*, 2016b) addressed this issue by normalizing the summed inputs of a given layer. Its effects are to re-center and re-scale the inputs by using simple statistics of the distribution of activations:

$$h^{(l)} = f\left(\frac{g}{\sigma^{(l)}}(z^{(l)} - \mu^{(l)}) + b\right), \tag{5.11}$$

where g and b the gain and bias parameters of the same dimension as $h^{(l)}$, and $\mu^{(l)}$ and $\sigma^{(l)}$ are the empirical mean and standard deviation of the pre-activations $z^{(l)}$.

5.5 Experiments

5.5.1 Training Performance

Benchmarks. We extended three domains previously proposed in Wu *et al.* (2017), i.e., Navigation (Faulwasser and Findeisen, 2009), HVAC (Agarwal *et al.*, 2010), and Reservoir Control (Yeh, 1985)) in order to incorporate stochastic transitions and additional nonlinearities with the overall goal of making the problems more appealing and challenging to our proposed approach.

Navigation is a path planning problem in a 2-dimensional space in which an agent is supposed to get to a goal position from a start position as fast as possible while avoiding deceleration zones. The position of the agent at timestep t is given by the state variable $x_t \in \mathbb{R}^2$. The agent's movement is given by the action fluent $u_t \in [-1, 1]^2$. The dynamics of the agent's movement is defined by:

$$\lambda = \prod_j \frac{2}{1 + \exp(-\alpha_j ||x_t - c_j||_2)} - 1\,, \tag{5.12}$$

$$x_{t+1} \sim \mathcal{N}(\mu = x_t + \lambda u_t, \sigma = \frac{\sigma_{\max}}{\sqrt{2}}||u_t||)\,, \tag{5.13}$$

where each deceleration zone j is characterized by its center position c_j and decay constant α_j, and its effect depends upon the Euclidean distance between its center and the agent's position. The joint deceleration factor λ is given by the multiplicative effect of each independent deceleration zone. The cost function is simply the Euclidian distance from the current position to the goal position:

$$C_t = ||x_t - x_{\text{GOAL}}||_2\,. \tag{5.14}$$

HVAC is a centralized continuous decision task in which the objective is to control the temperature of multiple rooms within a comfortable range subject to energy costs. The transition dynamics is defined by the nonlinear heat transfer through walls between adjacent spaces (e.g., rooms, hallway, or outside area). Each room i has a state variable $\delta_t^i \in \mathbb{R}$ denoting its temperature at timestep t. The action $u_t^i \in [0, u_{\max}]$ corresponds to the volume of heated air sent via vent actuation to room i at each timestep t. The transition function of the problem is given by the following dependencies:

$$\delta_t^{\text{outside}} \sim \mathcal{N}(\mu_{\text{out}}, \sigma_{\text{out}})\,, \tag{5.15}$$

$$\delta_{t+1}^i = \delta_t^i + \mathcal{N}(\mu_a = u_t^i(\delta_a - \delta_t^i), \sigma_a) + \sum_{\text{ADJ}(i,j)} \mathcal{N}(\mu_{ij} = (\delta_t^j - \delta_t^i)^3 / r_{ij}, \sigma_{ij})\,, \tag{5.16}$$

where δ_a is the temperature of the air being sent by the central actuator and r_{ij} is the thermal

resistance between room i and the adjacent space j. Standard deviations σ_a and σ_{ij} are constants of the domain. The cost function takes into consideration the temperature comfort zone $[l^i, u^i]$ of each room i and the energy cost k:

$$C_t = \sum_i \left| \delta_t^i - \frac{(l^i + u^i)}{2} \right| + k u_t^i \,.$$

Reservoir Control is a system to maintain the level of interconnected water reservoirs within a nominal range. Each reservoir i has a state fluent denoting its level $l_t^i \in [0, \mathrm{l}_{\mathrm{max}}^i]$ and a corresponding action fluent u_t^i related to the water flow to a downstream reservoir (or to the sea). Note that even though a given reservoir has a single downstream reservoir, it can have zero or more upstream reservoirs. The dynamics of the system is given by the following dependencies:

$$\begin{aligned} r_t^i &\sim \mathrm{Gamma}(k, \alpha)\,, \\ e_t^i &= 0.5 \log(l_t^i + 1) \left(\frac{l_t^i}{\mathrm{l}_{\mathrm{max}}^i} \right)^2 , \\ l_{t+1}^i &= \max\Big(0,\, l_t^i + r_t^i - e_t^i - u_t^i + \sum_{\mathrm{DOWN(j,\,i)}} a_t^j \Big) , \end{aligned}$$

where r_t^i and e_t^i denote the rain and evaporation over reservoir i, and $\mathrm{DOWN}(j, i)$ is a topology predicate indicating reservoir i is the downstream reservoir of j. The cost function is such that a penalty is given for each reservoir out of its nominal range:

$$C_t = \sum_i \begin{cases} \delta_{\mathrm{lower}} (l_{\mathrm{min}}^i - l_t^i)^2 & \text{if } \; l_t^i < l_{\mathrm{min}}^i \,, \\ \delta_{\mathrm{upper}} (l_{\mathrm{max}}^i - l_t^i)^2 & \text{if } \; l_t^i > l_{\mathrm{max}}^i \,, \\ 0 & \text{otherwise.} \end{cases}$$

DRP architectures. As there is an unbounded number of architectures to investigate, we set ourselves to evaluate two architectures representatives of distinct classes of models. The first DRP, denoted "tf-mdp (1)", is a shallow, but wide model. The second model, denoted "tf-mdp (2)", is a deep, but narrow model. Table 5.1 shows the number of hidden layers and units in each layer. In both models, we use the ELU activation function (Clevert *et al.*, 2015) and layer normalization (Ba *et al.*, 2016a) in the input layer. Table 5.2 shows the number of parameters for each model/architecture.

Table 5.1: *DRP architectures*

DRP	Hidden Layers	Number of Units
tf-mdp (1)	1	2048
tf-mdp (2)	4	256, 128, 64, 32

Implementation We implemented tf-mdp in TensorFlow (Abadi *et al.*, 2016).[4] We specified the domains/instances using RDDL (Relational Dynamic Influence Diagram Language) (Sanner, 2010) and compiled the models to stochastic computation graphs in TensorFlow using a compiler specifically built for this work.[5] We conducted all experiments on a single 2.4 GHz Intel Core i5 8GB

[4] https://github.com/thiagopbueno/tf-mdp
[5] https://github.com/thiagopbueno/rddl2tf

Table 5.2: *Number of parameters per domain/instance*

Domain	tf-plan	tf-mdp (1)	tf-mdp (2)
Nav2	40	10,246	44,070
HVAC3	120	14,345	44,361
HVAC6	240	26,642	45,234
Res10	400	43,038	46,398
Res20	800	84,028	49,308
Res30	1,200	125,018	52,218

RAM machine.

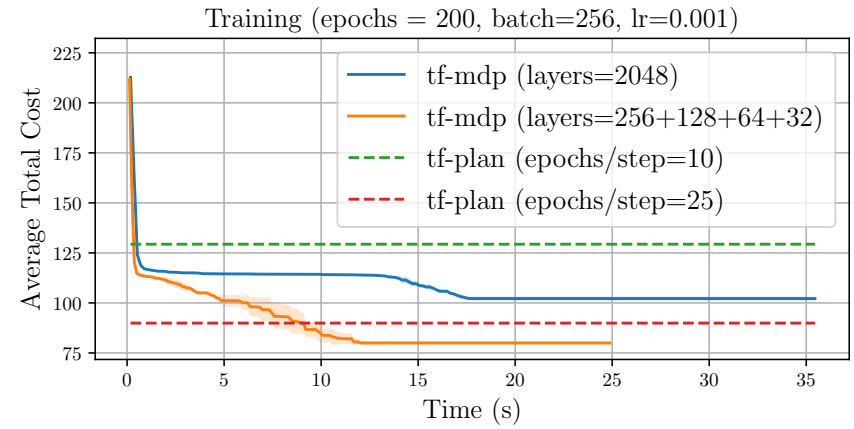
Methodology. Training neural nets and especially deep neural nets such as DRPs can be especially sensitive to the choice of training hyperparameters (e.g., learning rate, batch size, number of training epochs). Our objective with the experiments is not necessarily to achieve the best possible outcome by carefully fine-tuning hyperparameters, but instead to provide a reasonable comparison between the models. Hence, we selected the sensible default values shown in Table 5.3 and fix them for all training runs.

Table 5.3: *Training hyperparameters for tf-mdp*

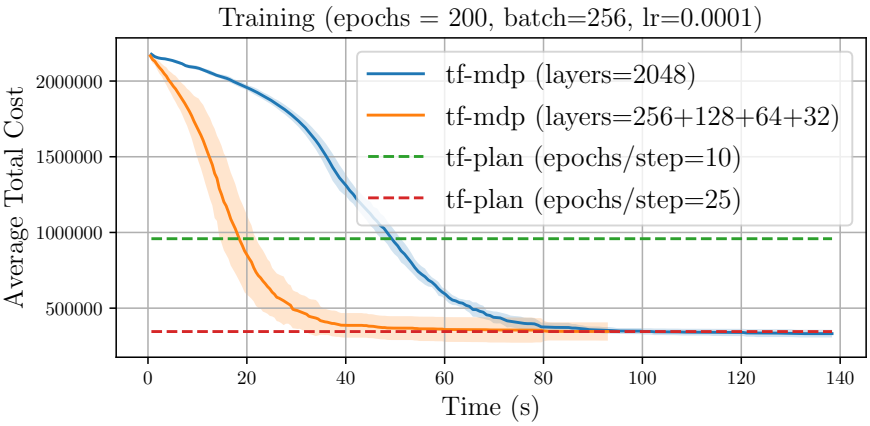
Domain	Batch	Learning rate	Epochs	Horizon
Nav	256	0.001	200	20
HVAC	256	0.0001	200	40
Res	256	0.001	200	40

We report average and standard deviation values over 10 training runs for each model/architecture in terms of quality of solution (i.e., average total cost from start state) and computational times. Additionally, in order to avoid hazardous fluctuations and divergences during training, we keep the best policy so far as a way to smooth out the gradient-based policy search. We compare the results of "tf-mdp (1)" and "tf-mdp (2)" to an online extension of tf-plan implemented as an open-loop feedback controller that attempts to optimize the average of sampled state-action trajectories. We run tf-plan for 10 and 25 training epochs per timestep, denoted "tf-plan (10)" and "tf-plan (25)" , respectively. Note that 10 and 25 training epochs per step of tf-plan corresponds to 4 and 10 times the total number of training epochs given to tf-mdp. We benchmark the results for the following domains/instances: Nav2 (Navigation with 2 deceleration zones), HVAC3 (3 rooms) and HVAC6 (6 rooms), and Res10/20/30 (Reservoir Control with 10, 20 and 30 reservoirs, respectively).

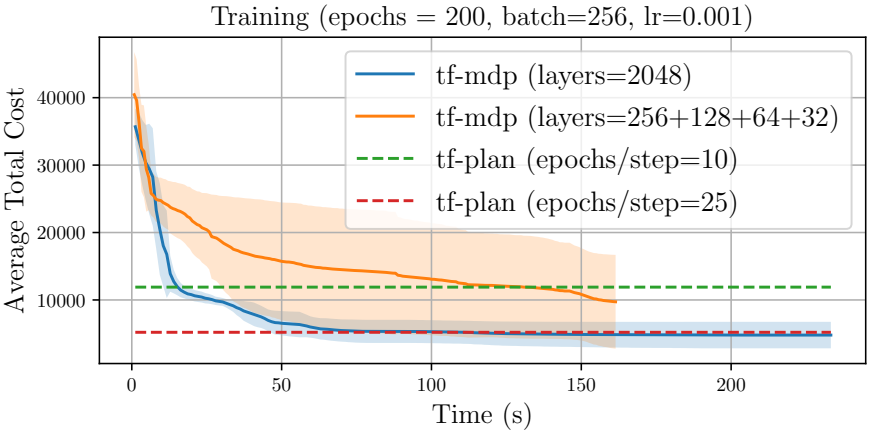
Results. Figures 5.5a, 5.5b and 5.5c show that DRPs can achieve comparable or better results than the online tf-plan planner. Additionally, we notice that tf-mdp (2) (i.e., deep and narrow DRP) performed better in Navigation and HVAC problems. However, tf-mdp (1) (i.e., shallow and wide DRP) was able to achieve better results for the number of training epochs pre-defined in Reservoir instances, even though it seems that tf-mdp (2) would be able to catch up if more training epochs were given. Also, we observe that for bigger problems in the Reservoir Control domain (as shown in Figure 5.5c), both architectures presented a considerable variance across different training runs



(a) *Navigation2*



(b) *HVAC3*



(c) *Reservoir20*

Figure 5.5: *DRP training: average total cost vs. training time*

5.5.2 Inference and Total Computing Time

Tables 5.4 and 5.5 show the empirical time estimates of inference (i.e., average time to compute the next action) and total computing time for each model and domain instance. We can observe that the average inference time of tf-mdp is within a fraction of the time required by the online planner tf-plan, which empirically validates the claim that DRPs can be useful for fast decision-making. Note that tf-mdp is on average 5 orders of magnitude faster than tf-plan in terms of inference time. Also, even when amortizing the offline computational cost over the timesteps of the horizon, tf-mdp remains competitive in terms of total computing time.

Domain	tf-plan (10)	tf-plan (25)	tf-mdp (1)	tf-mdp (2)
Nav2	1.2 ± 0.2	1.7 ± 0.3	$2.4 \cdot 10^{-5}$	$1.8 \cdot 10^{-5}$
HVAC3	3.2 ± 0.3	4.5 ± 0.5	$2.3 \cdot 10^{-5}$	$1.6 \cdot 10^{-5}$
HVAC6	3.6 ± 0.5	5.1 ± 1.3	$3.2 \cdot 10^{-5}$	$1.9 \cdot 10^{-5}$
Res10	2.9 ± 0.5	4.1 ± 0.2	$3.2 \cdot 10^{-5}$	$2.6 \cdot 10^{-5}$
Res20	4.8 ± 0.3	7.3 ± 2.2	$3.8 \cdot 10^{-5}$	$3.2 \cdot 10^{-5}$
Res30	7.4 ± 1.5	14.6 ± 6.1	$5.5 \cdot 10^{-5}$	$4.1 \cdot 10^{-5}$

Table 5.4: *Average inference time per step (sec)*

Domain	tf-plan (10)	tf-plan (25)	tf-mdp (1)	tf-mdp (2)
Nav2	26.6 ± 1.9	34.2 ± 2.3	35.4 ± 1.7	24.8 ± 1.0
HVAC3	126.4 ± 2.8	183.6 ± 4.7	138.3 ± 2.2	93.0 ± 2.5
HVAC6	138.9 ± 3.7	198.8 ± 6.5	150.5 ± 1.7	106.4 ± 2.7
Res10	117.7 ± 2.8	164.0 ± 4.7	190.7 ± 3.4	134.9 ± 4.5
Res20	194.4 ± 3.6	295.6 ± 5.6	233.2 ± 4.7	161.4 ± 2.7
Res30	296.3 ± 4.5	587.7 ± 5.8	344.9 ± 3.1	244.1 ± 3.7

Table 5.5: *Total computing time (sec)*

5.5.3 Trajectory Analysis

In this section, we analyze the state-action trajectory generated by DRPs for the problem of Reservoir Control. Our goal is to show that complex nonlinear parametric approximators can implement complex policies in a factored X-MDP. We train a 2-layer DRP with 256 units and ELU activation with RMSProp with learning rate 1e-3 and batch size 256 for 1000 iterations and we analyze the results in three settings of a reservoir control problem: (a) 5 reservoirs with initial level all below the nominal range (i.e., underflow), (b) 5 reservoirs with initial level all above the nominal range (i.e., overflow), and (c) 10 reservoirs with half of the reservoirs in overflow and the other half in underflow. In all instances we consider a linear topology for the interconnection between the reservoirs, i.e., each reservoir has at a single upstream and a single downstream reservoir, except the first reservoir which does not have an upstream connection and the last one with no downstream neighbor. We notice that in this set of experiments we subscribe to a simple rainfall model based on independent Gamma random variables with the same mean and variance across all reservoirs. A sample path of the corresponding exogenous random process is shown in Figure 5.6.

Figure 5.7 shows a sample of typical trajectories of a trained DRP as described above. We can observe that the reactive policy manages to control each reservoir level as expected in order to steer the system to its nominal range of operation $[50.0, 60.0]$. Notice how the difference in the scale of

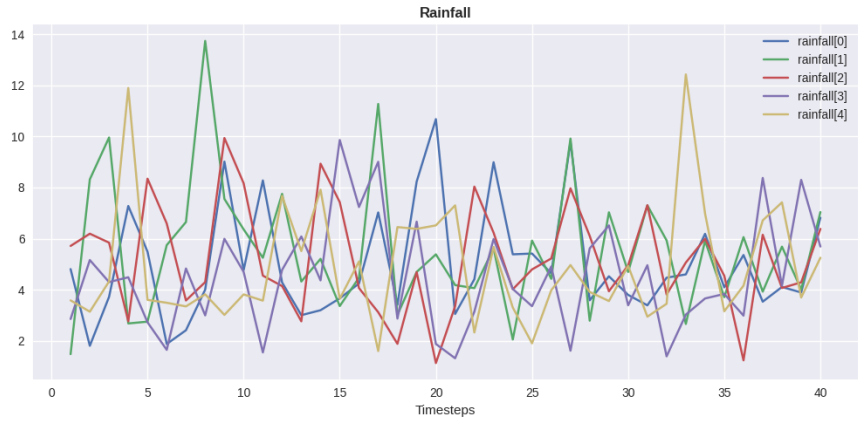


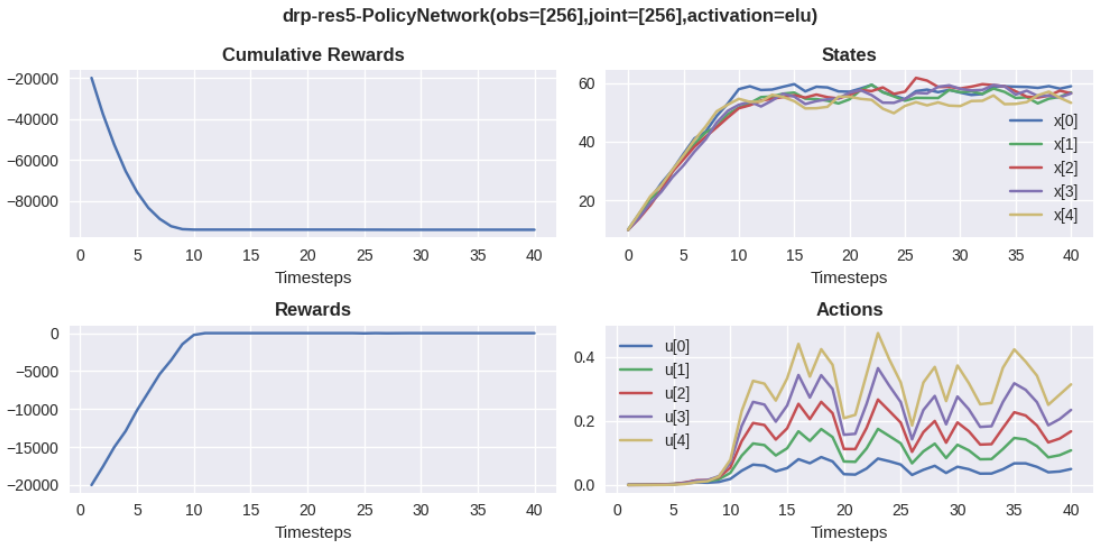
Figure 5.6: *Rainfall (5 reservoirs): sample of a random trajectory of the exogenous event variables.*

the actions is related to the topology of the problem. As a matter of fact, the 4th reservoir (the last in the linear chain) has to have a larger outflow in comparison to the 1st reservoir (the first in the topology) as all previous reservoirs have to move water downstream, which tends to overflow the last reservoir. In addition, we can observe that regardless of the initial level of each reservoir the DRP properly handles the regulation process obtaining convergence to the nominal range after 20 time steps as shown in the in Figure 5.7c.

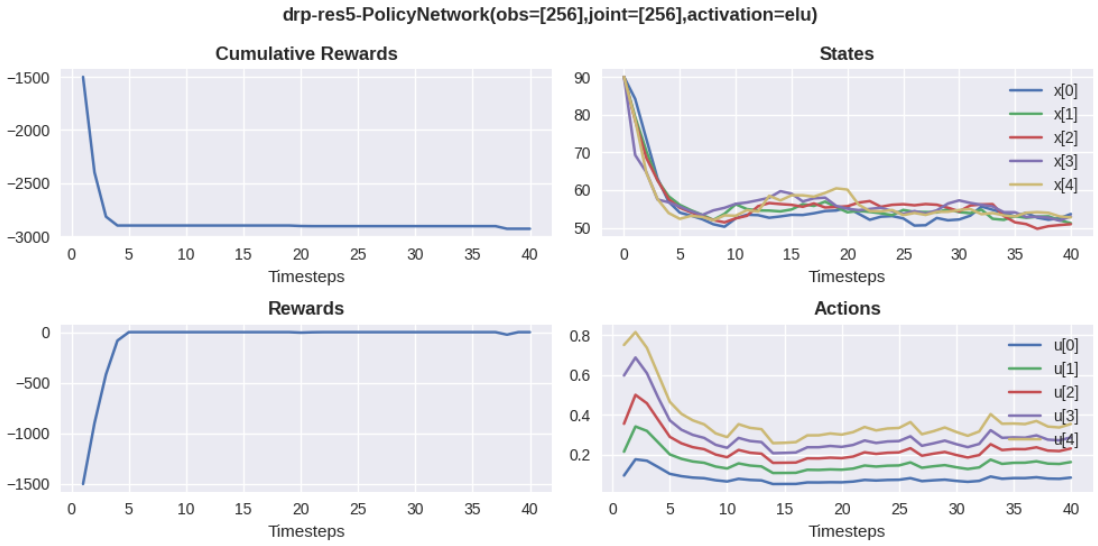
5.6 Conclusion

In this chapter, we investigated a promising approach for optimizing DRPs by leveraging non-convex optimization techniques commonly used in Deep Learning, but not frequently applied to AI planning. We presented a gradient-based optimization approach for training DRPs that exploits the re-parameterization of the stochastic computation graph of MDPs. We focused on continuous stochastic domains with concurrent actions and exogenous events exhibiting nonlinear transition and cost functions. We presented the results published in (Bueno *et al.*, 2019) and showed that training large DRPs with hundred of thousands of continuous action parameters can be carried out within minutes without the need of high-performance hardware. Finally, comparing the DRPs trained by our approach with online state-of-the-art gradient-based planners, we observed a speedup of several orders of magnitude on the time to select actions, which highlights the potential of DRPs for fast decision-making in continuous domains.

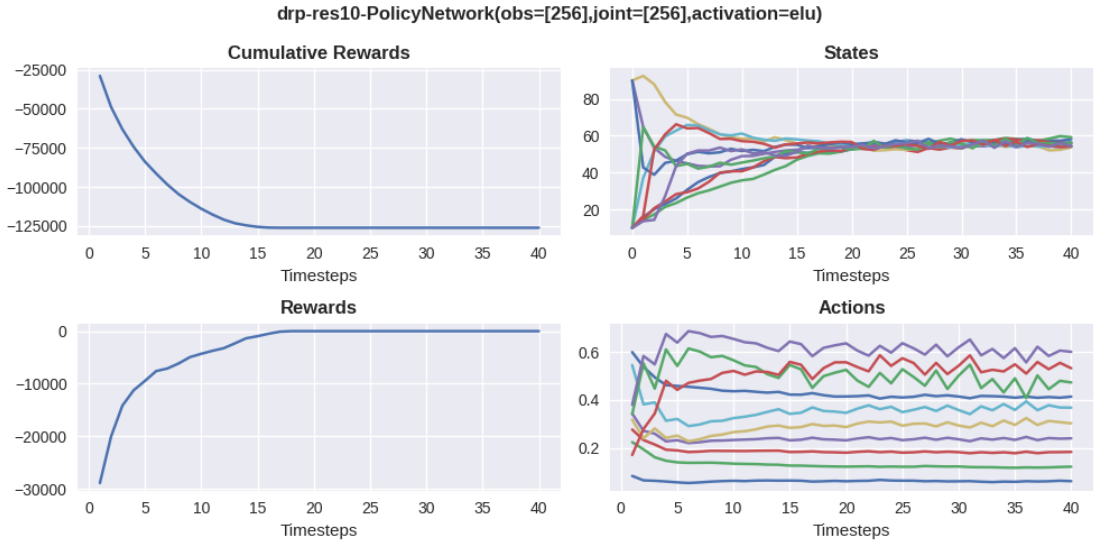
In the next section, we shall continue our investigation of ways to extend planning through backpropagation to stochastic domains. In particular, we will study the sequential decision-making problem from the viewpoint of online planning, i.e., the setting where we interleave planning and execution steps and forgo the idea of training an explicitly-represented policy, such as a DRP, in favor of focusing the computation solely on the next best action to take for a given current state.



(a) *Initial state (5 reservoirs in underflow)*



(b) *Initial state (5 reservoirs in overflow)*



(c) *Initial state (10 reservoirs 50% underflow + 50% overflow)*

Figure 5.7: *Reservoir Control: Performance of a DRP in different initial conditions: (a) underflow of all reservoirs, (b) overflow of all reservoirs, and (c) mixed underflow and overflow.*

Chapter 6

Stochastic Online Planning

Stochastic planning methods address sequential decision-making problems under uncertainty by computing a policy, i.e., a mapping from states to actions (Boutilier *et al.*, 1995). In cases where the policy is explicitly represented, the planning task can be typically solved by offline approaches, i.e., an agent uses all the available computational budget to reason and plan before executing any action in the system. Deep Reactive Policies (DRPs) (Bueno *et al.*, 2019) can be learned for problems in which all probability distributions are amenable to be reparametrized thus allowing gradients to flow through the transition model (Chapter 5). We remark that training a DRP via planning through backpropagation is an offline method that requires a priori knowledge of a good policy representation and a fixed planning horizon which are hyperparameters that need to be defined in advance. In addition, attempting to optimize such a policy in continuous state spaces can eventually become difficult as its performance is intrinsically dependent on the generalization capacity of the function approximator used and offline training might not be able to train the policy to be robust to rare but important random exogenous events.

An alternative solution to solving stochastic planning tasks in continuous domains is therefore to avoid the explicit representation of the policy and compute actions on the fly. This overall approach is known as *online planning* and it leads to methods that interleave planning and execution. The key observation is that every time an agent reaches a newly-visited state, it will plan accordingly, thus striking a trade-off between fast decision-making and quality of action.

Unlike in the case of training Deep Reactive Policies as discussed in the previous chapter, we focus here on extending planning through backpropagation to handle stochastic domains, but in the context of online planning. Our key observation is that the re-parameterization trick does not only make it possible to estimate gradients in stochastic computation graphs, a vital part of the planning through backpropagation formulation, but also allows to derive relaxations of the planning task via anticipatory sampling techniques. In this context, we investigate the opportunities and challenges of combining gradient-based optimization and online algorithms based on *certainty-equivalent control*, *hindsight optimization*, and *primal-dual information relaxation*. This chapter is organized as follows. First, we lay out the overall framework of online planning and its underlying plan-execute-monitor cycle and start investigating basic methods based on the principle of certainty-equivalent control. Next, we adapt the formulation of anticipatory sampling to our specific setting of gradient-based planning and discuss important issues related to different information relaxation schemes used as problem approximation. Finally, we introduce a version of the rollout algorithm that allows to integrate previously-trained DRPs with online planning through backpropagation.

6.1 Plan-Execute-Monitor

Online planning[1] is typically implemented by a 3-phase process. First, an agent gets to observe the current state x_t and then leverages its internal dynamics model to *plan* the next action to take. Once the planning procedure is terminated, an action u_t is selected for *execution* in the system. Finally, the agent *monitors* the next state x_{t+1} and the process restarts. The combined process of interleaving planning and execution is known as the *Plan-Execute-Monitor* cycle and is outlined in the Algorithm 8.

Algorithm 8: Online Planning

Input: MDP $\mathcal{M}$, start state $\bar{x}$, number of trials N, task horizon H, planning horizon T
Output: state-action trajectory τ, average return $\tilde{G}$

for $i = 0, \ldots, N-1$ **do**
  $G^i \leftarrow 0$
  $x_0 \leftarrow \bar{x}$
  $\tau^i \leftarrow (x_0)$
  $\mathcal{I} \leftarrow \{\}$
  for $t = 0, \ldots, H-1$ **do**
    $u_t^i \leftarrow \textsc{Plan}(\mathcal{M}, x_t^i, T, \mathcal{I})$
    $x_{t+1}^i, c_t^i \leftarrow \textsc{Execute}(\mathcal{M}, u_t^i)$
    $\mathcal{I} \leftarrow \textsc{Monitor}(\mathcal{M}, u_t^i, x_{t+1}^i)$
    $G^i \leftarrow G^i + c_t$
    $\tau^i \leftarrow (\tau^i, u_t^i, x_{t+1}^i)$
$\tilde{G} \leftarrow \frac{1}{N} \sum_{i=0}^{N-1} G^i$
return $\tilde{G}$, $\{\tau^i\}$

Online planning is based on the idea of forward lookahead from the current state. Specifically, by focusing the planning step over the possible future trajectories rooted at the current state, an agent can amortize the computational cost over many decision stages and also eventually control how much time is spent on each state based on previous computations. For instance, if an agent first computes a short-term near-optimal trajectory and temporarily maintains it in memory as candidate trajectory for the subsequent steps, then it can decide whether or not it makes sense to replan from scratch or reuse the previously-computed action at the current decision state. In Algorithm 8, the MONITOR procedure illustrates this possibility.

It is important to note that online methods naturally avoid unnecessary global exploration of the state space as only the actions for the visited states need to be computed. In other words, by postponing to commit to an action to the last minute, online planning circumvents the need to explicitly represent a policy function. Additionally, we remark that algorithms based on the plan-execute-monitor may be more robust when compared to other approximate offline planning algorithms in the case of rare events never seen during offline training as the online nature allow some degree of on-the-fly adaptability.

[1] The overall idea of online planning is closely related to receding-horizon and model-predictive control, albeit with different solution techniques.

6.2 Certainty-Equivalent Control

Certainty-Equivalent Control (CEC) (Van de Water and Willems, 1981) approximates the original probabilistic dynamics by choosing a nominal transition for each state and action. In its basic formulation, it amounts to replacing all random transitions with their most-likely outcomes, i.e., the next state that maximizes the likelihood of the transition. Figure 6.1 shows the resulting computation graph after the application of the CEC principle to a given X-MDP.

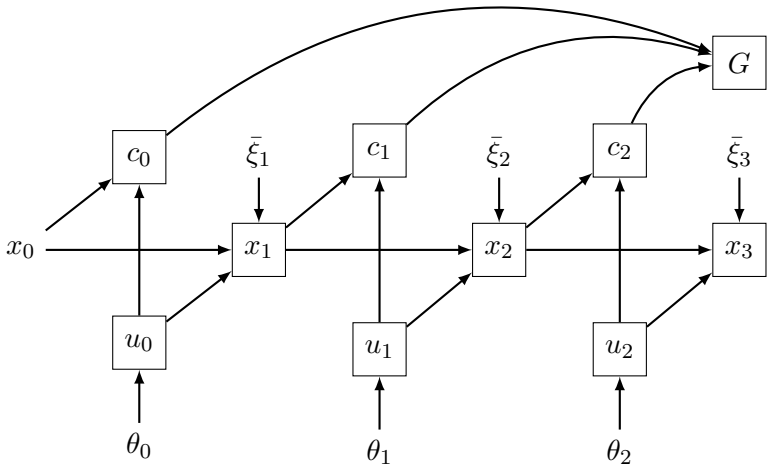


Figure 6.1: *Certainty-Equivalent Control: the determinization of the sources of random transitions in an X-MDP, i.e., $\xi_t = \bar{x}_t$, transforms its stochastic computation graph into a deterministic computation graph in which the state-trajectory depends on the nominal values of the exogenous events.*

In very specific cases where the uncertain system disturbances can be modeled as 0-mean random variables, the solution of the CEC formulation can still be optimal for the original problem. The LQG problem optimal control is an example of a family of tasks that can be optimally solved by simply ignoring the disturbances and thus considering a deterministic LQR problem. The intuition is that the net effect over the expected cost-to-go of an additive 0-mean noise is nullified on average. In optimal control, CEC is typically used in an online setting in which the uncertainty of system is incrementally incorporated into the controller via the observation of the current state after executing a control action, i.e., feedback is only used to ground the start state of each optimization.

In AI planning a number of simulation-based planning algorithms are based on the CEC principle. Of special mention is the FF-Replan (Yoon *et al.*, 2007) algorithm that made quite an impact in early editions of the probabilistic competition track at the ICAPS conference. In its original formulation, FF-Replan simplifies the problem via a single-outcome most-likely determinization. It determinizes the problem (e.g., assume that each action deterministically leads to one of its most probable outcomes), solves the deterministic problem obtained with an efficient heuristic search planner (e.g., Fast-Forward (FF) (Hoffmann, 2001)), executes the first action of the recommended plan, and then follows a simple replanning strategy. Despite the simplicity of the FF-Replan algorithm, the pattern it exploits of simplifying a complex stochastic planning problem in a way that an efficient deterministic planner can be used is very powerful. In this chapter, we make the observation that, in the same spirit of FF-Replan, the framework of deterministic gradient-based planning discussed in Chapter 4 can be readily adapted to implement an online planning through

backpropagation capable of solving stochastic continuous domains.

6.3 Anticipatory Sampling

Certainty-Equivalent Control (CEC) has been successfully applied to a number of applications, specially in model-predictive control. However, it is not the only option to derive online approximation methods for solving stochastic decision-making problems. As a matter of fact, CEC-based methods are particular examples of a broader class of *anticipatory sampling* algorithms, whose main idea is to reduce a complex probabilistic planning problem into a set of simpler deterministic problems via forecasting. In the light of this general methodology, we can easily understand the CEC principle as a particular case where the determinization of the exogenous events is built by sequentially chaining the most-likely transitions at each decision step.

As we will see in this section, anticipatory sampling (Brown *et al.*, 2010; Mercier and Hentenryck, 2008, 2007, 2011) in its most general form attempts to build a determinization of the original problem by considering alternative information structures of the decision-making. This is a powerful approach that has been applied in operations research work (Balseiro and Brown, 2019; Chen and Farias, 2013; Jiang *et al.*, 2020), discrete probabilistic planning (Issakkimuthu *et al.*, 2015; Yoon *et al.*, 2008), and (to a limited extent) to hybrid planning (Raghavan *et al.*, 2017), but to the best of our knowledge, it has not been thoroughly applied to the more general stochastic nonlinear setting, and in particular, to gradient-based planning as we have used in this thesis.

6.3.1 Information Relaxation

The key idea behind anticipatory sampling is to leverage a model of the exogenous events of the decision process in order to reason over possible future scenarios in a computationally simplified setting. Practically speaking, if a decision maker can efficiently compute (or somehow obtain from external sources) a sample path of the exogenous events, then a deterministic method might be applied to obtain a solution to a related, but easier problem, which in turn can provide a useful heuristic insight into the original stochastic problem. This approach might be seen as a dynamic, on-the-fly determinization technique that can form the basis of an online approximate decision-making procedure, akin to model-predictive control. The overall point is that sampling can be used to condition the exogenous random variables of the original stochastic problem leading to a set of scenario-based deterministic problem whose costs can be averaged to provide a lower bound on the optimal value function[2].

In the context of planning through backpropagation, anticipatory sampling allows to reuse all the deep learning machinery we discussed in previous chapters to optimize a specific set of plans from which a near-optimal action can be extracted. In particular, we note that by making the original problem deterministic, the gradient estimates can be naturally computed via backpropagation-through-time as no random variables are present to block the gradient flow in the computation graph of the Monte-Carlo approximated value function.

As we alluded earlier, the determinization performed by anticipatory sampling methods can be conceptually understood as a modifying the natural information structure of the decision process.

[2]We state everything in this chapter in terms of lower bounds as we are interested in cost minimization problems. For reward maximization tasks, the same results apply for upper bounds.

In other words, conditioning the exogenous events in the stochastic computation graph of an MDP and attempting to find plans to the derived deterministic sub-problems lead to a violation of the *non-anticipativity constraints*[3] that a decision maker have to subscribe to in order to implement any viable policy in real-world applications. Intuitively, the nonanticipativity constraints simply refer to the fact that a decision maker is bound to make a decision by only exploiting information contained in its past history, i.e., it is strictly forbidden to leverage any information related to the concrete future for the purpose of finding the next best action to take. For example, in inventory control problems the decision maker does not perfectly know in advance the future demand or the buying prices of products that it needs to maintain in store. Similarly, in water multi-reservoir control applications, the current decisions related to the outflows between reservoirs cannot be expected to be made over the perfect assumption that a specific amount of precipitation will happen in the next period.

However, a decision maker that has access to a model of such uncertain outcomes can momentarily work under the assumption that a particular realization of the exogenous event will in fact unfold, and therefore it can plan ahead using some form of scenario-based reasoning. This technique of bluntly ignoring the nonanticipativity constraints of the original Markov problem to reduce it to a computationally easier problem is known as *information relaxation*, a formalism studied in the community of operations research (Balseiro and Brown, 2019). From the viewpoint of the decision process, an information relaxation approach leads to an optimization problem defined over a set of policies that leverage privileged future information.

Algorithm 9 presents the abstract procedure employed by an online planner that uses information relaxation and deterministic planning to approximate the best action for the current state x_t. Given a computational budget specified in the form of a maximum number of scenarios N and a planning horizon H, the online planner first computes the sample paths corresponding to the scenarios $\xi^i_{t:t+H}$ and uses them to condition the exogenous random variables defined in the given X-MDP formulation $\mathcal{M}_\Xi$ in order to instantiate a deterministic version of the problem that can be solved via a deterministic planning algorithm (such as planning through backpropagation as discussed in Chapter 4). Notice that the deterministic planner computes a plan $\tilde{u}^i_{t:t+H}$ for each scenario $\xi^i_{t:t+H}$. Finally, depending on the specifics of the algorithm, the next action is selected among the set of starting actions of the scenario-based plans, $\tilde{u}^i_t$. The anticipatory sampling algorithm is subsequently used as the planning step, i.e., PLAN, used in Algorithm 8.

Algorithm 9: Online Planning via Anticipatory Sampling

Input: X-MDP $\mathcal{M}_\Xi$, current state x_t, number of scenarios N, planning horizon H
Output: action u_t

for $i = 0, \dots, N-1$ **do**
  $\xi^i_{t:t+H} \sim p(\Xi)$; ▷ Scenario sampling
  $\mathcal{M}^i \leftarrow \mathcal{M}_\Xi\left[\xi^i_{t:t+H}\right]$; ▷ Information relaxation
  $\tilde{u}^i_{t:t+H} \leftarrow \text{OPTIMIZE}(\mathcal{M}^i, x_t)$; ▷ Deterministic planning
$u_t \leftarrow \text{SELECTBESTACTION}(\{\tilde{u}^1_t, \dots, \tilde{u}^N_t\})$
return u_t

[3]Technically speaking, a violation of the nonanticipativity constraints is equivalent to a relaxation of the natural filtration of the controlled stochastic process.

Note that unlike the CEC approach that solves at each execution step a single determinized problem, anticipatory sampling in its general form randomly generates a set of non-stationary determinized problems (where the outcome selected for an action varies with time) and combines their solutions.

The most natural information relaxation is the *perfect information relaxation*. In this particular setting, a decision maker is assumed to have full knowledge of the future uncertainties for the purposes of approximated planning. We remark that it is important to make the distinction clear between being able to leverage the information about the realization of the external disturbances to obtain heuristic values and knowing with certainty which state trajectory is going to be unfolded as the result of an agent behavior during execution. As previously pointed out, the main goal of any information relaxation is practically speaking to provide a lower bound on the optimal value function that can be useful for in online planning. Theorem 6.3.1 algebraically formalizes the notion of perfect information in the context of X-MDPs.

Theorem 6.3.1. (Perfect Information) *Let $\mathcal{M}_{\Xi} = (\mathcal{X}, \mathcal{U}, \Xi, p_{\Xi}, f, g, \bar{s})$ be an X-MDP. Then, under the assumption of perfect information, the expected value of a clairvoyant policy $\mathbf{u}(\xi_{0:H-1})$ is a lower bound on the optimal value of $\mathcal{M}_{\Xi}$:*

$$V^*(x) \geq \mathbb{E}_{\xi_{0:H-1}} \left[\min_{\mathbf{u}(\xi_{0:H-1})} \sum_{t=0}^{H-1} g(x_t, u_t) \;\middle|\; x_{t+1} = f(x_t, u_t, \xi_t), x_0 = x \right] , \tag{6.1}$$

where the clairvoyant policy $\mathbf{u}(\xi_{0:H-1})$ is a mapping from a pre-sampled scenario $\xi_{0:H-1}$ to a sequence of actions $u_{0:H-1}$.

It is not hard to see the validity of Theorem 6.3.1 given that any information relaxation (and in particular the extreme case of perfect information) provides the planner with more information than is naturally available and therefore the average performance of the scenario-based policies cannot imply an underestimation on the optimal value function. In other words, the set of clairvoyant policies contains the set of viable policies (i.e., the policies typically studied in AI that do not violate any nonanticipativity constraints). Henceforth, the best clairvoyant policy should always obtain an improved value function when compared to any state-dependent policy. We refer the reader interested in a more comprehensive discussion of the concepts of clairvoyant policies and perfect information relaxation to the work of Mercier and Hentenryck (2007).

In the following sections, we will discuss specific implementations of the idea of information relaxation in the context of AI planning, and specifically for gradient-based planning.

6.3.2 Hindsight Optimization

In AI planning, the notion of perfect information relaxation has been proposed to solve goal-oriented problems via *Hindsight Optimization* (HOP) (Issakkimuthu *et al.*, 2015; Yoon *et al.*, 2008), an algorithm later extended to a hybrid planning setting where the system dynamics were defined over location-scale probability distributions parametrized by piecewise linear functions of states and actions (Raghavan *et al.*, 2017) and a solution was obtained via a Mixed-Integer Linear Programming (MILP).

The general idea behind the hindsight formulation is to rely on the perfect information relaxation of an X-MDP, $\mathcal{M}_{\Xi} = (\mathcal{X}, \mathcal{U}, \Xi, p_{\Xi}, f, g, \bar{s})$, to optimize online the action value function $\tilde{Q}_{\text{HS}}(x, u)$

using Monte-Carlo average over the future scenarios:

$$\tilde{Q}^{\text{HS}}(x_t, u_t) = g(x_t, u_t) + \frac{1}{N}\sum_{i=0}^{N}\left[\min_{u^i_{t:t+H}} \sum_{t'=t+1}^{H-1} g(x^i_{t'}, u^i_{t'}) \;\middle|\; x^i_{t'+1} = f(x^i_{t'}, u^i_{t'}, \xi^i_{t'}), x^i_t = x_t\right] \tag{6.2}$$

where i is the index of the pre-sampled scenario $\xi^i_{0:H-1}$ and $u^i_{t:t+H}$ are the actions recommended by the clairvoyant policy $\mathbf{u}(\xi^i_{0:H-1})$ for the i-th scenario.

The HOP planner executes at each decision stage the greedy action w.r.t. the action value $\tilde{Q}^{\text{HS}}(x_t, u_t)$, i.e.:

$$u^{\text{HS}}_t = \arg\min_{u \in \mathcal{U}(x_t)} \tilde{Q}^{\text{HS}}(x_t, u) \; . \tag{6.3}$$

We make the key observation that, in principle, Equations 6.2 and 6.3 are completely agnostic to the underlying optimization method used to obtain the current action u_t. In fact, assuming that the X-MDP is defined with differentiable transition and cost functions, it suffices to approximate the minimization operators in these equations using gradient descent to implement a hindsight-based planning through backpropagation algorithm. Indeed, after pre-sampling all the exogenous random variables composing the scenarios $\xi^i_{0:H-1}$, the stochastic computation graph of the X-MDP is reduced to the deterministic computation graph illustrated in Figure 6.2 which is almost identical to one shown in Figure 4.2 (as presented in Chapter 4) with the exception that the transition dynamics becomes non-stationary as induced by the concrete realizations of the noise variables ξ_t.

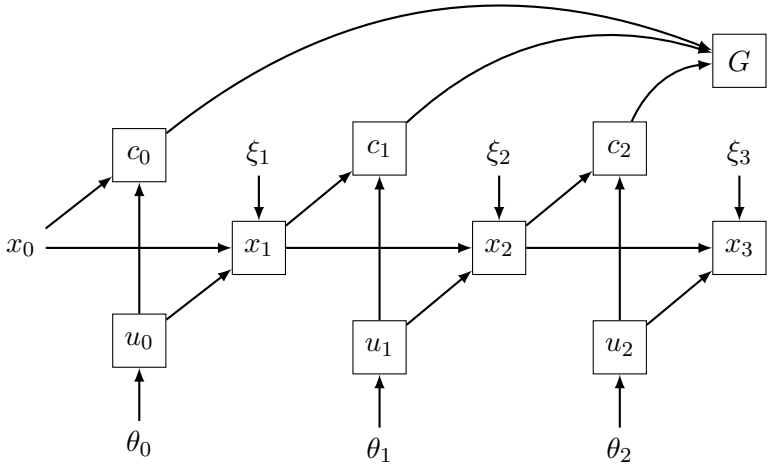


Figure 6.2: *Computation graph of Hindsight Optimization: the perfect information relaxation turns the stochastic computation graph of an X-MDP into a non-stationary version of a deterministic computation graph. Note that once a sample path $\xi_{0:H-1}$ is obtained by a realization of the exogenous event process, the induced dynamics becomes deterministic but nonstationary, i.e., $f_t(x_t, u_t) = f(x_t, u_t, \xi_t)$.*

Algorithm 10 outlines the general implementation of the idea of perfect information and hindsight optimization in the context of planning through backpropagation. The important part is to notice that the current action u_t is shared across the optimization of each scenario-based plan, implying that a good current action might be the one that in retrospect leads to good subsequent returns for most-likely scenarios. Also, note that both the current action u_t and all the scenario-based plans $\mathbf{u}^i_{t+1:t+H}$ are jointly optimized via backpropagation-through-time.

Algorithm 10: Gradient-based Hindsight Optimization

Input: X-MDP $\mathcal{M}_{\Xi}$, state x_t, scenarios N, planning horizon H, iterations K
Output: action u_t

$u_t \sim q(\mathcal{U})$; ▷ Initialize current action
for $k = 0, \cdots, K-1$ **do**
  for $i = 0, \cdots, N-1$ **do**
    $\xi^i_{t:t+H} \sim p(\Xi)$; ▷ Scenario sampling
    $x^i_{t+1} \leftarrow f(x_t, u_t, \xi^i_t)$; ▷ Compute next state
    $\mathbf{u}^i_{t+1:t+H} \sim q(\mathcal{U})$; ▷ Initialize scenario-based plan
    $G^i \leftarrow g(x_t, u_t)$; ▷ Initialize scenario return
    for $t' = t+1, \cdots, H-1$ **do**
      $x^i_{t'+1} \leftarrow f(x^i_{t'}, u^i_{t'}, \xi^i_{t'})$; ▷ Trajectory unrolling
      $G^i \leftarrow G^i + g(x^i_{t'}, u^i_{t'})$; ▷ Cumulative return
  $J \leftarrow \frac{1}{N} \sum_{i=0}^{N-1} G^i$; ▷ Monte-Carlo Approximation
  $\theta \leftarrow \{u_t\} \cup \{\mathbf{u}^i_{t+1:t+H}, i = 0, \cdots, N-1\}$; ▷ Parameters
  $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$; ▷ Backpropagation-Through-Time
return u_t

6.3.3 Penalty Functions and Dual Bounds

The effectiveness of an anticipatory sampling approach is influenced by two factors: (*i*) the inaccuracy of the lower bound implied by the underlying information relaxation w.r.t. the optimal value function, and (*ii*) the computational complexity of solving the relaxed problem. From the viewpoint of planning through backpropagation, we notice there is a design space of optimization algorithms and hyper-parameters we can tune to control the trade-off between the computational efficiency and the solution error for a given relaxed problem. However, the issue of inaccuracy of the problem approximation induced by the information relaxation is something that needs to be tackle with techniques outside the landscape of deep learning.

It has been empirically shown that for some decision-making problems, the perfect information relaxation, as presented in the context of hindsight optimization in the previous section, might lead to a lower bound that is too weak to be useful in practice (Brown *et al.*, 2010). For this reason, in order to come up with tighter and therefore more useful lower bounds, (Brown *et al.*, 2010) have proposed to incorporate a penalty function in the relaxed problem. As we will discuss in this section, it is possible to strengthen the induced problem determinization by imposing virtual costs associated with the usage of future information by the clairvoyant policy. The intuition is based on the fact that disallowing the set of clairvoyant policies to be too "smart" in regards to the possible future scenarios might reduce the gap between the (unrealizable) lower bound of the information relaxation and the (realizable) optimal value function.

The addition of a penalty function to the induced lower bound approximation leads directly to an *information relaxation primal-dual formulation* (Brown *et al.*, 2010). In the remainder of this section, we lay out the technical details of the information relaxation duality theory needed to introduce our proposed algorithm based on hindsight optimization with penalty functions. We remark that our goal is not to provide an in-depth formalization but to adapt the main theoretical results to our context of differentiable planning. For all technical details and proofs of the theorems, we refer the reader to the comprehensive work of Balseiro and Brown (2019).

We start by introducing a dual feasible penalty function in Definition 6.3.1.

Definition 6.3.1. (Dual feasible penalty function) *Let $\mathcal{M}_\Xi = (\mathcal{X}, \mathcal{U}, \Xi, p_\Xi, f, g, \bar{s})$ be an X-MDP. A function $z\colon \mathcal{X} \times \mathcal{A}^H \times \Xi^H \to \mathbb{R}$ is a dual feasible penalty function for the H-horizon decision problem of $\mathcal{M}_\Xi$ if and only if:*

$$\mathbb{E}_{\xi_{0:H-1}}\left[z(x, \mathbf{u}, \xi_{0:H-1})\right] \geq 0 \ . \tag{6.4}$$

The main results of the duality theory applied to the information relaxation setting in decision-making build upon the weak duality lemma that states that a dual feasible penalty function implies a lower bound on the optimal cost-to-go value function.

Lemma 6.3.1. (Weak Duality) *Let $\mathcal{M}_\Xi = (\mathcal{X}, \mathcal{U}, \Xi, p_\Xi, f, g, \bar{s})$ be an X-MDP. If π is a viable policy, $\mathbf{u}(\xi_{0:H-1})$ is a clairvoyant policy for a given information relaxation, and the function z is a dual feasible penalty for $\mathcal{M}_\Xi$, then it holds that:*

$$V^\pi(x) \geq \mathbb{E}_{\xi_{0:H-1}}\left[\min_{\mathbf{u}(\xi_{0:H-1})} \left\{G(x, \mathbf{u}, \xi_{0:H-1}) + z(x, \mathbf{u}, \xi_{0:H-1})\right\}\right] \ , \tag{6.5}$$

where $G(x, \mathbf{u}, \xi_{0:H-1})$ denotes the return obtained from state x after following the plan $\mathbf{u}$ conditioned on the scenario $\xi_{0:H-1}$ having been realized.

The intuition behind Lemma 6.3.1 is best understood from a simulation viewpoint. If a decision maker somehow gets access to the realization of the future trajectory, it can exploits this additional information to search for optimal actions conditioned on this specific scenario. Now, if we keep giving the decision maker such future realizations according to the natural distribution of the exogenous events, then the average of the scenario-based optimal total costs obtained this way will necessarily be a lower bound on the value of any nonanticipative policy.

An important consequence of the weak duality lemma is that by maximizing its r.h.s. over the set of dual feasible penalty functions the value of an optimal nonanticipative policy can be obtained exactly, provided the optimal value is bounded. This result, presented in Theorem 6.3.2, is similar to the strong duality theorem of linear programming, and correspondingly, also comes with a set of complementary slackness conditions (not discussed here for simplicity).

Theorem 6.3.2. (Strong Duality) *Let $\mathcal{M}_\Xi = (\mathcal{X}, \mathcal{U}, \Xi, p_\Xi, f, g, \bar{s})$ be an X-MDP. If $\mathbf{u}(\xi_{0:H-1})$ is a clairvoyant policy for a given information relaxation and $\mathcal{Z}$ is the set of dual feasible penalty for $\mathcal{M}_\Xi$, then it holds that:*

$$V^*(x) = \max_{z \in \mathcal{Z}} \mathbb{E}_{\xi_{0:H-1}}\left[\min_{\mathbf{u}} \left\{G(x, \mathbf{u}, \xi_{0:H-1}) + z(x, \mathbf{u}, \xi_{0:H-1})\right\}\right] \ , \tag{6.6}$$

where $G(x, \mathbf{u}, \xi_{0:H-1})$ denotes the return obtained from following the plan $\mathbf{u}$ conditioned on the scenario $\xi_{0:H-1}$ been realized.

For each time step t, let $v_t\colon \mathcal{X} \to \mathbb{R}$ be any (possibly) time-dependent function defined over the state space of a given X-MDP $\mathcal{M}_\Xi = (\mathcal{X}, \mathcal{U}, \Xi, p_\Xi, f, g, \bar{s})$. We call v_t the *generating function*.

Additionally, define the function $\bar{v}_t$ as follows:

$$\bar{v}_t(x_t, u_t, \xi_t) = \mathbb{E}_{\tilde{\xi}_t}\left[v_{t+1}(f(x_t, u_t, \tilde{\xi}_t)\right] - v_{t+1}(f(x_t, u_t, \xi_t)) \ . \tag{6.7}$$

Brown *et al.* (2010) showed that a dual feasible penalty function $z^v(x, \mathbf{u}, \xi_{0:H-1})$ for state x and plan $\mathbf{u}$ conditioned on scenario $\xi_{0:H-1}$ can be obtained by the following additive form induced by the generating function v_t:

$$z^v(x, \mathbf{u}, \xi_{0:H-1}) = \sum_{t=0}^{H-1} \bar{v}_t(x_t, u_t, \xi_t) \ . \tag{6.8}$$

The duality theory of information relaxation proves that if the optimal value function is used as the underlying generating function, then there is no gap between the dual bound and the optimal value function. This amounts to the ideal penalty and essentially corresponds to the "value gained from knowing the future", also known as the expected clairvoyant value. Intuitively, if one knows precisely how much can be gained by using future information, then a perfect penalty can be constructed so as to recover the optimal value of the primal problem. In practice, however, strong duality can only be aspired to as the optimal value function is never available. Nevertheless, the theory also suggests that a viable strategy can be constructed via approximations of the optimal value function. As a matter of fact, the penalties generated through approximation schemes of the optimal value function shall lead to valid bounds as long as the estimate of the expectation on the r.h.s of Equation 6.7 is unbiased.

Algorithm 11: Primal-Dual Gradient-based Hindsight Optimization

Input: X-MDP $\mathcal{M}_\Xi$, penalty function $\tilde{v}$, state x_t, scenarios N, planning horizon H, iterations K

Output: action u_t

$u_t \sim q(\mathcal{U})$; ▷ Initialize current action

for $k = 0, \cdots, K-1$ **do**

  for $i = 0, \cdots, N-1$ **do**

    $\xi^i_{t:t+H} \sim p(\Xi)$; ▷ Scenario sampling

    $x^i_{t+1} \leftarrow f(x_t, u_t, \xi^i_t)$; ▷ Compute next state

    $\mathbf{u}^i_{t+1:t+H} \sim q(\mathcal{U})$; ▷ Initialize scenario-based plan

    $G^i \leftarrow g(x_t, u_t) + \tilde{v}_t(x_t, u_t, \xi^i_t)$; ▷ Initialize scenario return

    for $t' = t+1, \cdots, H-1$ **do**

      $x^i_{t'+1} \leftarrow f(x^i_{t'}, u^i_{t'}, \xi^i_{t'})$; ▷ Trajectory unrolling

      $G^i \leftarrow G^i + g(x^i_{t'}, u^i_{t'}) + \bar{v}_t(x_{t'}, u_{t'}, \xi^i_{t'})$; ▷ Cumulative penalty-added cost

  $J \leftarrow \frac{1}{N}\sum_{i=0}^{N-1} G^i$; ▷ Monte-Carlo Approximation

  $\theta \leftarrow \{u_t\} \cup \{\mathbf{u}^i_{t+1:t+H}, i = 0, \cdots, N-1\}$; ▷ Parameters

  $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$; ▷ Backpropagation-Through-Time

return u_t

Algorithm 11 shows the pseudocode of a primal-dual extension of the Gradient-based Hindsight Optimization procedure outlined in Algorithm 10. The only significant difference is the addition of the penalty function to the cumulative cost used as the loss function fed to the backpropagation-through-time algorithm.

We propose a number of implementations of the primal-dual gradient-based hindsight optimiza-

tion algorithm based on using the approximate value function of a heuristic policy as the generating function v_t. We note that there is a number of options for choosing heuristic policies: a random policy, problem-dependent heuristic policy, ϵ-greedy w.r.t. the Q-value of another policy or the solution of a CEC-based method, and even a pre-trained DRP. Note that these value function approximators (possibly learned via regression) may not necessarily be realizable by any particular policy and should be understood as numeric approximations that might be valuable for punishing the use of future information by a decision maker in the context of anticipatory sampling.

6.4 Rollout Algorithms

Anticipatory sampling algorithms are not the only option for implementing online planning methods. Their main goal is to reduce a complex stochastic problem into a set of simpler deterministic problems that can be solved efficiently to guide the search for the next best action. However, an alternative to problem approximation is to adapt an approximate policy iteration algorithm to the online setting, thus leading to the well-known class of *rollout* algorithms (Bertsekas, 2010; Bertsekas *et al.*, 1997; Goodson *et al.*, 2017; Tesauro and Galperin, 1996).

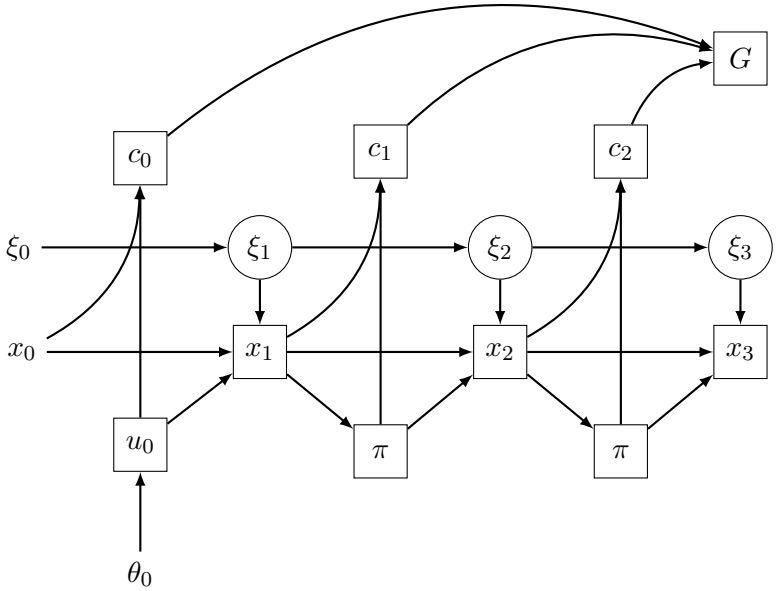


Figure 6.3: *Stochastic computation graph of the Rollout algorithm: the current action u_0 and a given heuristic policy π induce a trajectory for a sampled exogenous event path $\xi_{0:4}$; only action u_0 is optimized via backpropagation-through-time, the heuristic policy remains fixed and only allows for Monte-Carlo estimation of the return G.*

The basic idea is to start with an initial heuristic policy, possibly a mediocre one, and at each time step explore small improvements for the current action conditioned on the fact that after its execution the agent will continue to follow the heuristic policy until the end of the trajectory. This idea is akin to a single step of policy improvement (i.e., the greedification step) in a policy iteration scheme. The crucial difference is that by only considering a single step of policy improvement, the algorithm can now be implemented in an online fashion, meaning that planning and execution can be naturally interleaved. Recall that policy iteration is an offline method that typically requires

several iterations of policy evaluation and policy improvement to obtain an approximately-optimal global policy. In contrast, rollout algorithms focus only in approximating a good enough local action, which is key point in an online planning method.

Rollout algorithms have been mainly developed for discrete planning domains (Bertsekas *et al.*, 1997). We remark, however, that it is possible to extend such algorithms to the continuous case, provided that the an effective optimization method is available to greedify the current action selection. We propose to leverage planning through backpropagation as the optimization method for improving the current action upon the rollout policy. In addition, we make the key observation that we can use a previously-trained DRP as the heuristic policy for rollout, which leads to the integration of all techniques of gradient-based planning developed thus far in this thesis.

Algorithm 12: Rollout Planning through Backpropagation

Input: X-MDP $\mathcal{M}_\Xi$, rollout policy π, state x_t, scenarios N, horizon H, iterations K
Output: action u_t

$u_t \sim q(\mathcal{U})$; ▷ Initialize current action
for $k = 0, \cdots, K-1$ **do**
  for $i = 0, \cdots, N-1$ **do**
    $\xi^i_{t:t+H} \sim p(\Xi)$; ▷ Scenario sampling
    $x^i_{t+1} \leftarrow f(x_t, u_t, \xi^i_t)$; ▷ Compute next state
    $\mathbf{u}^i_{t+1:t+H} \sim q(\mathcal{U})$; ▷ Initialize scenario-based plan
    $G^i \leftarrow g(x_t, u_t)$; ▷ Initialize scenario return
    for $t' = t+1, \cdots, H-1$ **do**
      $u^i_{t'} \leftarrow \pi(x^i_{t'})$; ▷ Rollout policy
      $x^i_{t'+1} \leftarrow f(x^i_{t'}, u^i_{t'}, \xi^i_{t'})$; ▷ Trajectory unrolling
      $G^i \leftarrow G^i + g(x^i_{t'}, u^i_{t'})$
  $J \leftarrow \frac{1}{N} \sum_{i=0}^{N-1} G^i$; ▷ Monte-Carlo Approximation
  $\theta \leftarrow \{u_t\}$; ▷ Parameters
  $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$; ▷ Backpropagation-Through-Time
return u_t

Algorithm 12 outlines the integration of the online rollout method with backpropagation-through-time. Notice that, unlike in anticipatory sampling, only the current action u_t is optimized and the trajectory unrolling is carried out with the given heuristic policy and not with a set of scenario-based plans.

6.5 Experiments

6.5.1 Empirical Comparison between CEC and HOP on HVAC Systems

In this section, we compare the performance of both the CEC and HOP gradient-based planners. We conduct experiments on an HVAC control benchmark as an example of physical control task with noisy disturbances on the heat and conduction flows.

Benchmark. The HVAC domain with 6 rooms we consider in this section follows the dynamics and cost defined in Section 5.5. In particular, the control task is modeled with a start state for which the temperature in all rooms is below the comfort zone, therefore the goal of the planner is

Figure 6.4: *Cumulative reward of CEC and HOP gradient-based planners for different planning iterations.*

to rapidly heat the rooms and then keep the temperature within the comfort zone between 20 and 23.5 degrees while balancing the energy cost.

Methodology. We ran the online gradient-based planners based on the perfect information relaxation of the HVAC system with 6 rooms with a particular adjacency topology fixed across experiments. We used the RMSProp optimizer with learning rate 1e-3 and batch size 256 and compare the performance of CEC and HOP for an instance with a 40-timestep horizon.

Hyperparameter Analysis. As a first experiment, we tried different number of planning iterations per decision step. Figure 6.4 show the cumulative rewards for 10, 50, 100 and 500 iterations. We observe that past a certain number of iterations the planner starts to overfit to the pre-sampled scenarios, thus obtaining a worsen performance.

Trajectory Analysis. As expected, both planners find a sequence of actions that steer the system towards the comfort zone by first increasing the heat transfer and as the temperature gets close to the optimum the heating is decreased accordingly. Figure 6.5 shows the conduction profile between the rooms, the hall and with the outside of the building. Note that some rooms are not adjacent to the hall or the outside, thus no conduction is showed in the last two charts (i.e., 0-valued constant at the top of the chart). Figures 6.6a and 6.6b show the overall performance of the CEC-based planner and HOP-based planner, respectively. We observe that for this particular instance of the problem the performance of both algorithms is quite similar. We conjecture that this is due to the homogeneous nature of the multiplicative noise that can be managed on average to decision upon the best action at each timestep, which suggests that the upper bound optimized by the HOP planner is very close to the nominal determinization induced by the CEC principle.

6.5.2 Empirical Comparison of Rollout in Reservoir Control

Benchmark. In this section, we compare the Rollout algorithm in the Reservoir Control domain described in Section 5.5 with $n = 20, 30, 40$ reservoirs with initial state in underflow (i.e., level =

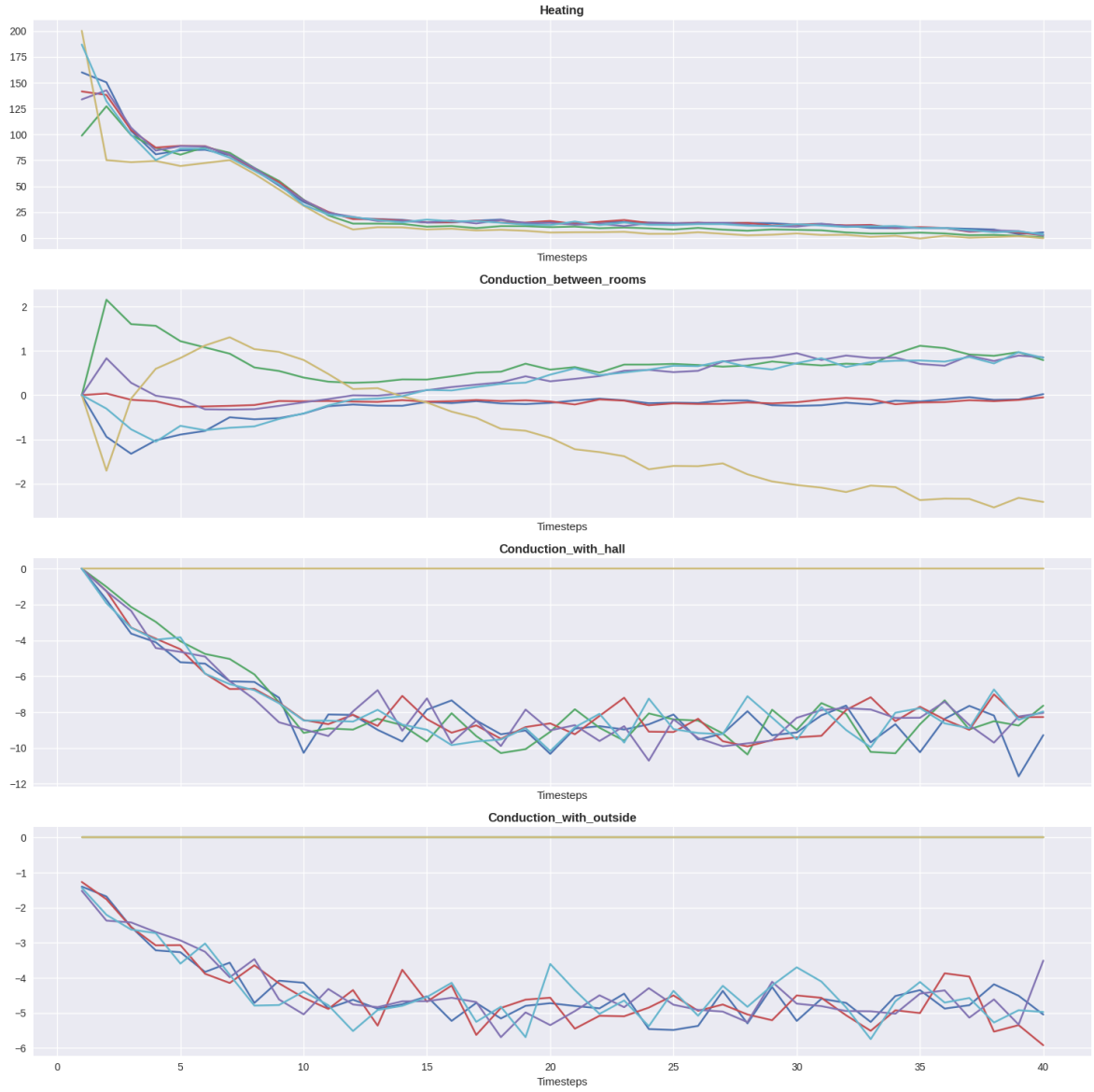
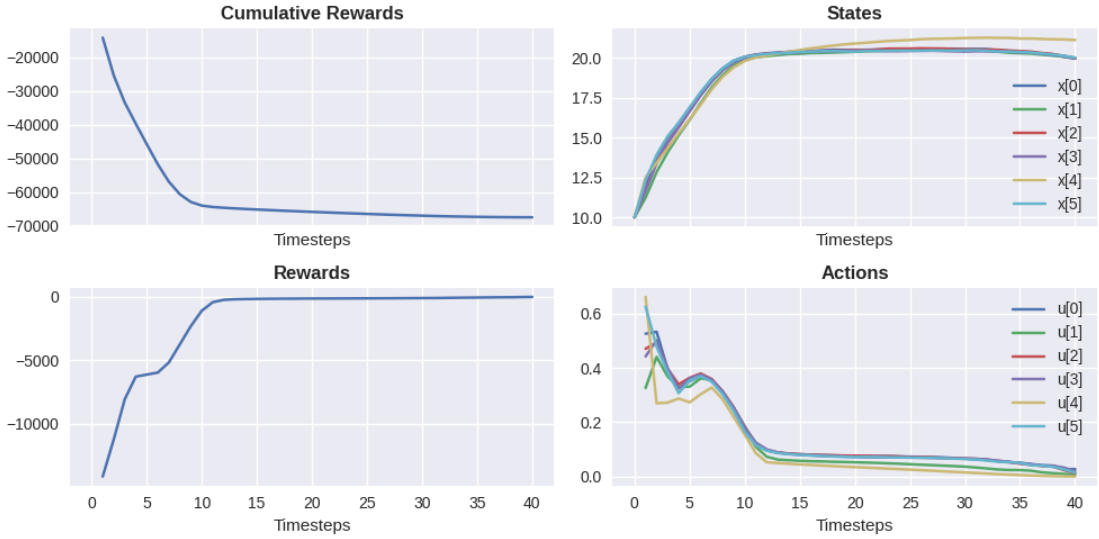


Figure 6.5: *HVAC (6 rooms): sample path of heat transfer and conduction between rooms, hall and with the outside.*

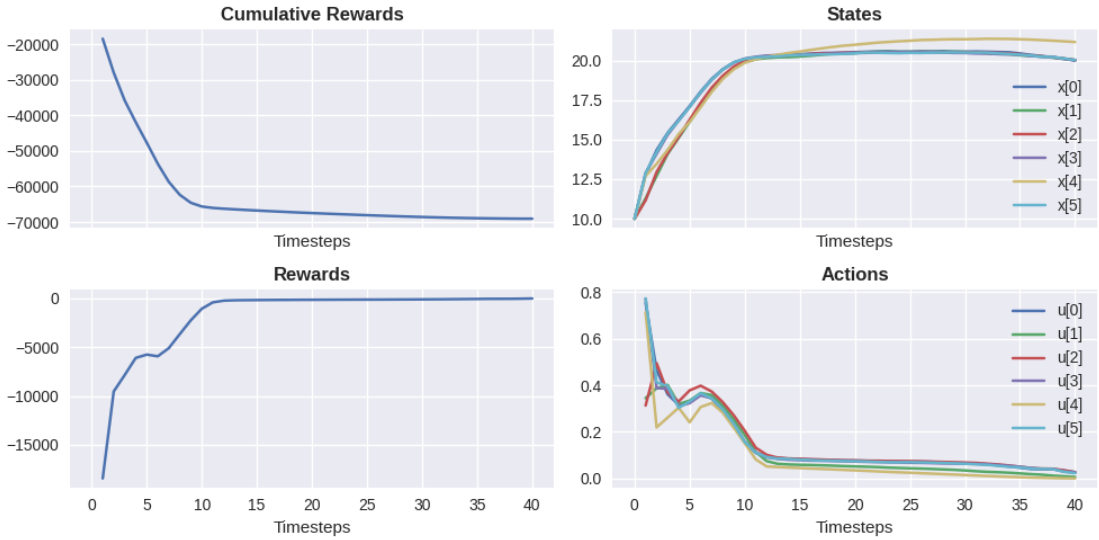
10.0, lower bound = 50, upper bound = 60) and horizon = 40.

Methodology. We run Algorithm 12 both with a random policy and a partially-trained DRP policy as the rollout policy. We perform the online optimization with batch size 256, learning rate 5e-3 with 400 iterations per timestep. The partially-trained DRP is trained with 50 iterations with learning rate 1e-4. We remark that with this very limited number of iterations and small learning rate we guarantee that the resulting DRP is far from convergence. We compare the Rollout algorithm with three baseline policies: random policy, partially-trained DRP, and the greedy policy. The greedy policy was trained with the same hyperparameters but it only optimizes the current cost in each timestep. For each instance, we run each algorithm 20 times and average the results.

Results. Table 6.1 shows the total reward that Rollout outperforms the baseline policies as expected. In particular, we notice that the Rollout+Random combination achieves over 45% improve-



(a) *CEC planner*



(b) *HOP planner*

Figure 6.6: *HVAC (6 rooms): performance of online planning through backpropagation*

ment over the random policy when $n = 20$ reservoirs and the Rollout+DRP combination achieves over 7% improvement over the partially-trained DRP. Also, we observe that Rollout+Random consistently outperforms the Greedy policy, thus showing that planning over longer horizons even with a random policy can be beneficial.

Table 6.1: *Comparison between Rollout and baseline policies: average and standard deviation of total reward.*

		Baselines			**Rollout**	
		Random	**DRP**	**Greedy**	**+ Random**	**+ DRP**
$n = 20$	Avg	-715062.70	-417646.21	-413469.12	-386252.81 (45.98%)	-385154.90 (7.78%)
	Std	18127.49	9571.52	13719.94	19559.85 (-7.90%)	11281.19 (17.86%)
$n = 30$	Avg	-1023759.00	-612477.12	-601157.90	-589034.84 (42.46%)	-577318.31 (5.74%)
	Std	26195.96	14719.50	18371.71	22670.27 (13.45%)	11538.91 (21.60%)
$n = 40$	Avg	1308585.00	-838397.33	-788574.98	-780927.80 (159.67%)	-777660.46 (7.24%)
	Std	22322.08	19155.44	12497.69	21112.46 (5.41%)	13103.54 (31.59%)

6.6 Conclusion

In this chapter, we introduced an alternative way to extend planning through backpropagation to stochastic domains. Unlike in Chapter 5 that covered offline planning, the algorithms discussed here all operate in the online setting, meaning that they interleave planning and execution and focus its computation in finding the next best action to take. Therefore, the methods developed here do not maintain an explicit policy represented in memory. In particular, we presented the theory of certainty-equivalent control and anticipatory sampling and related it to the determinization and hindsight optimization techniques used in probabilistic planning in the context of discrete MDPs. In addition, we introduce the Rollout algorithm as middle ground solution that implements a lightweight online planning framework that only updates the current action in the stochastic computation graph and uses another policy to estimate the value of the next state via Monte-Carlo simulation.

In the next chapter, we conclude this thesis and provide additional discussions on possible future works on the topic of planning in stochastic computation graphs.

Chapter 7

Conclusion

In this thesis, we aimed at studying the opportunities of applying deep learning techniques to improve continuous planning. We made the key observation that if a differentiable model of the sequential decision-making process can be made available to an autonomous agent, then **gradients through differentiable models of the environment can be efficiently leveraged to guide planners to find effective solutions in continuous domains**. This simple idea leads to differentiable planning methods that can be effectively applied to solve continuous planning problems, both in the deterministic and in the offline and online stochastic settings.

The flexibility and efficiency of the solutions presented in this thesis are based on the general theory of *stochastic computation graphs* (SCGs) (Schulman, 2016) and the *reverse-mode automatic differentiation* (Griewank and Walther, 2008) algorithm. SCGs are important in and of itself as a mathematical formulation for differentiable stochastic models, but for the purposes of this thesis, its main goal is to outline the necessary conditions to properly obtain unbiased gradient estimators whenever stochastic nodes are part of the computation graph. In addition, after the stochasticity of the computation graph is dealt accordingly, the automatic differentiation algorithm efficiently implements the derivative chain rule to obtain the gradient of an objective function w.r.t. to the input parameters of the model.

From the viewpoint of differentiable planning in stochastic domains, the theory SCGs leads to the identification of the *re-parameterization trick* (and more generally to the formulation of X-MDPs in Definition 5.3.1) as the necessary ingredient to adapt gradient descent optimization methods (Hinton *et al.*, 2012b; Ruder, 2016) to solve continuous stochastic planning problems. Also, it is important to note that automatic differentiation (also known as *error backpropagation* (Rumelhart *et al.*, 1986) in the context of deep learning) is the ultimate enabler of differentiable planning, either in the deterministic continuous setting leading to planning through backpropagation and TensorPlan as discussed in Chapter 4 or in the stochastic setting leading to the offline training of Deep Reactive Policies via gradient-based policy search in Chapter 5 or the several online planners based on anticipatory sampling or rollout techniques in Chapter 6. We remark that an efficient, parallelized implementation of automatic differentiation is key to allow the scalability of differentiable planning as first brought the attention of the planning community in the seminal work of Wu *et al.* (2017).

In this context, we argued in this thesis that backpropagating gradients through the mode in SCGs to optimize the value function of a plan (in the deterministic case) or a policy (in the stochastic case) approximated via Monte-Carlo simulations is an effective mechanism to allow continuous

planning. We believe that the framework we developed in this thesis provide an interesting answer to the problem of finding scalable solutions to stochastic nonlinear domains without resorting to discretization or problem-dependent simplifications. We recall our main contributions:

(i) we reformulate the original planning through backpropagation algorithm (Wu *et al.*, 2017) as a trajectory optimization technique in Chapter 4; and

(ii) we adapt the underlying gradient-based planner in several ways to the more general case of stochastic problems represented as X-MDPs, both for offline training in Chapter 5 and online planning in Chapter 6.

In particular, we proposed to train Deep Reactive Policies (DRPs) (Bueno *et al.*, 2019) represented by deep neural networks via backpropagation-through-time and showed that special care must be taken to stabilize the underlying optimization in long horizons. We demonstrated that such DRPs can allow for fast decision-making with the same level of performance when compared to gradient-based online planners. In addition, we proposed a different formulation planning through backpropagation as trajectory optimization thus making clear the distinction between learning internal representations in Recurrent Neural Networks (RNNs) (Goodfellow *et al.*, 2016) and optimizing trajectories (either action trajectories, i.e., plans, in the shooting formulation of Definition 4.1.2 or state-action trajectories in direct transcription of Definition 4.1.3). Preliminary results of our formulation and an analysis of the optimality gap has been recently published (Scaroni *et al.*, 2020). Finally, we also showed that the same planing through backpropagation approach can be made efficient for online planning via the framework of information relaxation and hindsight optimization, which results are to be published yet.

In the remainder of this chapter, we present a final discussion that wrap up the high-level ideas we touched upon in this work and close the thesis with some final remarks and directions for future work.

7.1 Discussion

7.1.1 The Interplay of Planning and Learning Processes

The ability of combining planning and learning has long been recognized as a necessary attribute for the development of effective and efficient autonomous agents in AI (Russell and Norvig, 2002). On one hand, except in small problems, a planner is most likely to depend upon good heuristics to advance its lookahead search which can in principle be learned from experience. And on the other hand, learning as the principal mechanism for generalization absolutely needs access to rich and diverse data sets that can only be obtained via systematic search and planning.

Historically, however, both disciplines have advanced relatively separate from each other. Even though the same sequential decision-making problems could in principle be solved with planning or learning techniques, for practical reasons, the fields have focused on different benchmarks in order to improve their proposed algorithms. The underlying matter has been exacerbated by the erroneous view that planning is intrinsically a symbolic approach while learning can only operate on latent-space representations. Furthermore, the superficial division between reasoning and perception tasks have also contributed to the separation of these communities.

In contrast to this rather limited view, a number of recent initiatives have started to provide a deeper understanding of the design space of planning and learning algorithms and showed that a great deal of untapped opportunities lie at the intersection of such techniques (Bertsekas, 2019; Moerland *et al.*, 2020a; Powell, 2019; Recht, 2019). Perhaps, the most notorious recent example of what can be achieved with a powerful combination of planning and learning is the *MuZero* algorithm (Schrittwieser *et al.*, 2020). By leveraging powerful deep neural networks to approximate the optimal value function and the optimal policy, MuZero manages to boost the capabilities of a Monte-Carlo Tree Search to solve from scratch a number of very complex combinatorial decision problems such as the games of go and chess. In particular, it should be noted that it was long believed that no algorithm would be able to beat a human professional player in the game of Go due to the enormous branching factor that considerably limits the depth of search and the complex strategic patterns needed to master the game (Silver *et al.*, 2016). Additionally, the role of planning should not be undermine. As a matter of fact, planning is used both during offline training to provide the necessary exploration to feed the neural networks with rich data to learn from and during online play in which online planning is used to adapt to previously unseen situations on the fly as planning and execution is interleaved as a mechanism to increase robustness.

In the light of the success of MuZero, we expect (and hope) that a number of derivative works combining planning and learning techniques will be developed in the near future.

7.1.2 Stochastic Computation Graphs for Searching in Continuous Spaces

As shown in this thesis, gradients in computation graphs representing the objective function of planning problems can be used to guide the optimization of a sequence of actions (i.e., a plan) in the deterministic setting or for policy search in stochastic setting. At a high-level, both approaches implement a form of local search method, either in the space of plans or in the space of policy parameters. Therefore, the algorithms presented here all share the same disadvantage of local search methods, namely they can be trapped in local minima. However, since automatic differentiation software allows to run multiple such planners in parallel (hence one of the reasons of the scalability of TensorPlan (Wu *et al.*, 2017)), this problem of local minima can be somewhat attenuated. A key limitation still persists though. These parallel planners share no information, and consequently may waste lots of computation trying without much hope to optimize badly-initialized plans or policies.

Recently, a novel method for efficiently training neural networks based on the idea of Population-based Training (PBT) (Jaderberg *et al.*, 2017) has shown that it is possible to share information in parallel local search methods to focus computation on the more promising solutions found so far, thus obtaining consistent improvements in training time, accuracy, and stability for a number of applications. We conjecture that a similar information-sharing mechanism can be specialized to improve the local search in differentiable planning methods based on stochastic computation graphs.

7.2 Future Work

We remark that the framework of planning through backpropagation as developed in this thesis is quite flexible, which can lead to a number of future works. Recall that the only requirement for gradient-based planning is differentiability of the dynamics model and cost functions. Consequently, we expect a number of possible future works to be built upon the ideas presented in this thesis. In

particular, we identify three main areas for further extensions of differentiable planning: (*i*) *learned models*, (*ii*) *discrete spaces*, and (*iii*) *risk-sensitive policies*.

Learned models. We observe that recently proposed works in model-based reinforcement learning have investigated the use of differentiable planning in robotics simulation under the designation of stochastic value gradients (Heess *et al.*, 2015) or model-based policy optimization (Janner *et al.*, 2019). However, due to compounding errors in the learned model, the horizon of the planning through backpropagation needs to be severely reduced to no more than 5 or 10 time steps, which places a hard bottleneck on the generalization capacity of the terminal value function used to evaluate the remaining cost-to-go. The sample complexity of these proposed model-based reinforcement algorithms is substantially improved when compared to their model-free counterparts, albeit at a usually very high computational cost. Lots of opportunities lie in finding solutions that can achieve a better trade-off between model errors, sample complexity, and computational cost.

Discrete spaces. As crazy as it may sound, there are a number of recently-proposed alternatives to backpropagate gradients through discrete stochastic computations (Bengio *et al.*, 2013; Jang *et al.*, 2017; Kool *et al.*, 2020; Maddison *et al.*, 2017; Tokui and Sato, 2016). These estimators are in general based on some form of continuous relaxations of the discrete random variables or the straight-through gradient which basically ignores the sampling step of the random variables in the backward pass in automatic differentiation. Of course, there is a price to pay for all these methods. They can be hard to implement it right in all cases as the underlying hyperparameters can be quite sensitive in practice, and they can only obtain biased gradients or at least unbiased in the limit of the continuous relaxations. Nevertheless, recent works have managed to adapt some of discrete gradient estimators to obtain differentiable planning algorithms capable of achieving state-of-the-art results in Atari games (Hafner *et al.*, 2020). In a different vein, a model-based planning algorithm known as SOGBOFA (Cui and Khardon, 2016) has managed to leverage the idea of algebraic rollouts (Cui *et al.*, 2015) to derive an online planner that uses gradient-based optimization in discrete stochastic domains. By using transformations over the original logical representation of RDDL (Sanner, 2010) models to obtain numerical expressions governing the evolution of each state variable, SOGBOGFA relaxes the discrete stochastic transition into a continuous deterministic transition for which gradients can easily be backpropagate through, effectively implementing a form of online differentiable planning. It remains an interesting challenge to combine these discrete approaches with the ideas in this thesis to obtain a hybrid (i.e., discrete and continuous) differentiable planner.

Risk-sensitive policies. Another very interesting direction for extending differentiable planning is to define specialized objective functions in stochastic planning problems in order to obtain risk-sensitive policies, a very active area of research in probabilistic planning and in safe reinforcement learning. As a matter of fact, there is no reason why to only optimize the value function (i.e., the expected total cost) in differentiable planning. Indeed, it is naturally possible to extend the overall optimization formulation presented in this thesis to account for different risk metrics provided that the risk measure used remains differentiable. In particular, we remark that risk-sensitive formulations based either on CVaR (conditional value-at-risk) (Chow *et al.*, 2015) or worst-case optimizations can be built on top of planning through backpropagation. Recent works have leveraged

similar formulations to learn model-free risk-sensitive reinforcement learning agents (Tang *et al.*, 2019; Yang *et al.*, 2021) or to train model-based risk-aware Deep Reactive Policies (Patton *et al.*, 2021). We expect different extensions to be built on top of differentiable planning methods to allow for different trade-offs involving risks in the future.

7.3 Final Remarks

In summary, we believe that the overall idea of planning through backpropagation in stochastic computation graphs will be an important component in the development of AI agents capable of autonomous reasoning and learning, specially given the great number of successful practical applications of gradient-based machine learning that we witness almost every day in the media and in the literature. In this context, we expect that this thesis has contributed to lay the foundations of future innovations in the area of differentiable planning.

As exciting as it has been to the author to investigate the research area of differentiable planning, this topic was not the starting project of his PhD. Initially, we investigated probabilistic logic programming (Dries *et al.*, 2015) as a formal language and inference engine to represent and solve Markov Decision Processes (MDPs) (Puterman, 1994) in discrete spaces. This initial study led to the publication of the MDP-ProbLog framework (Bueno *et al.*, 2016) and a preliminary extension to MDPs with imprecise probabilities (Bueno *et al.*, 2017). In this thesis, we refrained to additionally cover those works for the sake of unity of exposition as the theory and practical machinery used in probabilistic logic programming is very different from the stochastic computation graph theory and automatic differentiation techniques we have explored here. Nevertheless, the author is very thankful for having developed these works, both personally, as a first contact with research in AI, and academically, as the gateway to a greater appreciation of modeling languages in MDPs which in turn led to study of the RDDL language and the development of the RDDL2TensorFlow compiler[1] and subsequently to the ideas developed here in the context of differentiable planning in continuous domains.

[1] https://github.com/thiagopbueno/rddl2tf

Bibliography

Abadi *et al.***(2016)** Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard *et al.* Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283. Cited in page 36, 43, 60, 81

Agarwal *et al.***(2010)** Yuvraj Agarwal, Bharathan Balaji, Rajesh Gupta, Jacob Lyles, Michael Wei and Thomas Weng. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*, pages 1–6. ACM. Cited in page 9, 80

Auer *et al.***(2002)** Peter Auer, Nicolò Cesa-Bianchi and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256. Cited in page 23, 24

Ba *et al.***(2016a)** Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*. Cited in page 81

Ba *et al.***(2016b)** Lei Jimmy Ba, Jamie Ryan Kiros and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450. Cited in page 80

Bahar *et al.***(1997)** R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo and Fabio Somenzi. Algebraic decision diagrams and their applications. *Formal Methods Syst. Des.*, 10(2/3):171–206. Cited in page 23

Balseiro and Brown(2019) Santiago R. Balseiro and David B. Brown. Approximations to stochastic dynamic programs via information relaxation duality. *Oper. Res.*, 67(2):577–597. Cited in page 90, 91, 94

Barto *et al.***(1983)** Andrew G. Barto, Richard S. Sutton and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.*, 13(5):834–846. Cited in page 26, 39, 72

Barto *et al.***(1995)** Andrew G. Barto, Steven J. Bradtke and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artif. Intell.*, 72(1-2):81–138. Cited in page 23

Barto *et al.***(2021)** Andrew G. Barto, Richard S. Sutton and Charles W. Anderson. Looking back on the actor-critic architecture. *IEEE Trans. Syst. Man Cybern. Syst.*, 51(1):40–50. Cited in page 26, 39

Bellman(1957) Richard Bellman. *Dynamic programming*. Princeton University Press, New Jersey. Cited in page 21, 22

Bengio *et al.***(1993)** Yoshua Bengio, Paolo Frasconi and Patrice Y. Simard. The problem of learning long-term dependencies in recurrent networks. In *ICNN*, pages 1183–1188. IEEE. Cited in page 58

Bengio *et al.***(2013)** Yoshua Bengio, Nicholas Léonard and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432. Cited in page 106

Benton *et al.***(2012)** J. Benton, Amanda Jane Coles and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*. AAAI. Cited in page 34

Bertsekas(2010) Dimitri P Bertsekas. Rollout algorithms for discrete optimization: A survey. *Handbook of Combinatorial Optimization, D. Zu and P. Pardalos, Eds. Springer*. Cited in page 97

Bertsekas(2019) Dimitri P Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA. Cited in page 105

Bertsekas and Tsitsiklis(1996) Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*, volume 3 of *Optimization and neural computation series*. Athena Scientific. Cited in page 25

Bertsekas *et al.***(1997)** Dimitri P. Bertsekas, John N. Tsitsiklis and Cynara Wu. Rollout algorithms for combinatorial optimization. *J. Heuristics*, 3(3):245–262. Cited in page 97, 98

Bonet and Geffner(2012) Blai Bonet and Hector Geffner. Action selection for mdps: Anytime ao* versus UCT. In *AAAI*. AAAI Press. Cited in page 71

Bonet and Geffner(2003) Blai Bonet and Hector Geffner. Labeled RTDP: improving the convergence of real-time dynamic programming. In *ICAPS*, pages 12–21. AAAI. Cited in page 23, 34

Bonet and Geffner(1999) Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *ECP*, volume 1809 of *Lecture Notes in Computer Science*, pages 360–372. Springer. Cited in page 23

Bonet and Geffner(2001) Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33. Cited in page 23

Boutilier(2005) Craig Boutilier. The influence of influence diagrams on artificial intelligence. *Decis. Anal.*, 2(4):229–231. Cited in page 17

Boutilier *et al.***(1995)** Craig Boutilier, Thomas Dean and Steve Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Second European Workshop on Planning*. Cited in page 20, 87

Brown *et al.***(2010)** David B. Brown, James E. Smith and Peng Sun. Information relaxations and duality in stochastic dynamic programs. *Oper. Res.*, 58(4-Part-1):785–801. Cited in page 90, 94, 96

Browne *et al.***(2012)** Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis and Simon Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intell. AI Games*, 4(1):1–43. Cited in page 24

Bryant(1986) Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691. Cited in page 23

Buckman *et al.***(2018)** Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *NeurIPS*, pages 8234–8244. Cited in page 27

Bueno *et al.***(2016)** Thiago Pereira Bueno, Denis Deratani Mauá, Leliane N. de Barros and Fábio Gagliardi Cozman. Markov decision processes specified by probabilistic logic programming: Representation and solution. In *BRACIS*, pages 337–342. IEEE Computer Society. Cited in page 107

Bueno *et al.***(2017)** Thiago Pereira Bueno, Denis Deratani Mauá, Leliane N. de Barros and Fábio Gagliardi Cozman. Modeling markov decision processes with imprecise probabilities using probabilistic logic programming. In *ISIPTA*, volume 62 of *Proceedings of Machine Learning Research*, pages 49–60. PMLR. Cited in page 107

Bueno *et al.***(2019)** Thiago Pereira Bueno, Leliane N. de Barros, Denis Deratani Mauá and Scott Sanner. Deep reactive policies for planning in stochastic nonlinear domains. In *AAAI*, pages 7530–7537. AAAI Press. Cited in page 4, 6, 34, 72, 85, 87, 104

Camacho and Alba(2013) Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media. Cited in page 30

Castelletti *et al.***(2008)** Andrea Castelletti, Francesca Pianosi and Rodolfo Soncini-Sessa. Water reservoir control under economic, social and environmental constraints. *Autom.*, 44(6):1595–1607. Cited in page 8

Chang *et al.***(2005)** Hyeong Soo Chang, Michael C. Fu, Jiaqiao Hu and Steven I. Marcus. An adaptive sampling algorithm for solving markov decision processes. *Oper. Res.*, 53(1):126–139. Cited in page 3, 23, 24

Chaslot *et al.***(2007)** GMJB Chaslot, Mark Winands, JWHM Uiterwijk, H Van Den Herik, Bruno Bouzy and P Wang. Progressive strategies for monte-carlo tree search. In *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. Cited in page 23, 24, 34

Chen and Farias(2013) Yiwei Chen and Vivek F. Farias. Simple policies for dynamic pricing with imperfect forecasts. *Oper. Res.*, 61(3):612–624. Cited in page 90

Chow *et al.***(2015)** Yinlam Chow, Aviv Tamar, Shie Mannor and Marco Pavone. Risk-sensitive and robust decision-making: a cvar optimization approach. In *NIPS*, pages 1522–1530. Cited in page 106

Chua *et al.***(2018)** Kurtland Chua, Roberto Calandra, Rowan McAllister and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, pages 4759–4770. Cited in page 27

Chung *et al.***(2014)** Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555. Cited in page 58

Clavera *et al.***(2020)** Ignasi Clavera, Yao Fu and Pieter Abbeel. Model-augmented actor-critic: Backpropagating through paths. In *ICLR*. OpenReview.net. Cited in page 27

Clevert *et al.***(2015)** Djork-Arné Clevert, Thomas Unterthiner and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR*, abs/1511.07289. URL http://arxiv.org/abs/1511.07289. Cited in page 81

Coulom(2007) Rémi Coulom. Computing "elo ratings" of move patterns in the game of go. *J. Int. Comput. Games Assoc.*, 30(4):198–208. Cited in page 24

Cui and Khardon(2016) Hao Cui and Roni Khardon. Online symbolic gradient-based optimization for factored action mdps. In *IJCAI*, pages 3075–3081. IJCAI/AAAI Press. Cited in page 106

Cui *et al.***(2015)** Hao Cui, Roni Khardon, Alan Fern and Prasad Tadepalli. Factored MCTS for large scale stochastic planning. In *AAAI*, pages 3261–3267. AAAI Press. Cited in page 71, 106

Dai and Goldsmith(2007) Peng Dai and Judy Goldsmith. Topological value iteration algorithm for markov decision processes. In *IJCAI*, pages 1860–1865. Cited in page 22

Darwiche(2009) Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press. Cited in page 17

Deisenroth and Rasmussen(2011) Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *ICML*, pages 465–472. Omnipress. Cited in page 26

Deisenroth *et al.***(2013)** Marc Peter Deisenroth, Gerhard Neumann and Jan Peters. A survey on policy search for robotics. *Found. Trends Robotics*, 2(1-2):1–142. Cited in page 26, 43

Doncieux *et al.***(2004)** Stephane Doncieux, Jean-Arcady Meyer and France Animatlab-LIP. Evolution of neurocontrollers for complex systems: alternatives to the incremental approach. In *Proceedings of the international conference on artificial intelligence and applications (AIA 2004)*, pages 1–6. Cited in page 72

D'Oro and Jaskowski(2020) Pierluca D'Oro and Wojciech Jaskowski. How to learn a useful critic? model-based action-gradient-estimator policy optimization. In *NeurIPS*. Cited in page 27

D'Oro *et al.***(2020)** Pierluca D'Oro, Alberto Maria Metelli, Andrea Tirinzoni, Matteo Papini and Marcello Restelli. Gradient-aware model-based policy search. In *AAAI*, pages 3801–3808. AAAI Press. Cited in page 27

Dries *et al.***(2015)** Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer and Luc De Raedt. Problog2: Probabilistic logic programming. In *ECML/PKDD (3)*, volume 9286 of *Lecture Notes in Computer Science*, pages 312–315. Springer. Cited in page 107

Farquhar *et al.***(2018)** Gregory Farquhar, Tim Rocktäschel, Maximilian Igl and Shimon Whiteson. Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. In *ICLR (Poster)*. OpenReview.net. Cited in page 27

Faulwasser and Findeisen(2009) Timm Faulwasser and Rolf Findeisen. Nonlinear model predictive path following control. *Nonlinear Model Predictive Control*, 384:335–343. Cited in page 7, 62, 80

Feinberg *et al.***(2018)** Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101. Cited in page 27

Ferguson and Stentz(2004) David I. Ferguson and Anthony Stentz. Focussed propagation of mdps for path planning. In *ICTAI*, pages 310–317. IEEE Computer Society. Cited in page 22

Figurnov *et al.***(2018)** Mikhail Figurnov, Shakir Mohamed and Andriy Mnih. Implicit reparameterization gradients. In *NeurIPS*, pages 439–450. Cited in page 76

Forbes *et al.***(1995)** Jeff Forbes, Timothy Huang, Keiji Kanazawa and Stuart J. Russell. The batmobile: Towards a bayesian automated taxi. In *IJCAI*, pages 1878–1885. Morgan Kaufmann. Cited in page 17

Fujimoto *et al.***(2018)** Scott Fujimoto, Herke van Hoof and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1582–1591. PMLR. Cited in page 26

Gelada *et al.***(2019)** Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum and Marc G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2170–2179. PMLR. Cited in page 27

Ghahramani *et al.***(2020)** Ali Ghahramani, Parson Galicia, David Lehrer, Zubin Varghese, Zhe Wang and Yogesh Pandit. Artificial intelligence for efficient thermal comfort systems: Requirements, current applications and future directions. *Frontiers in Built Environment*, 6. Cited in page 9

Glynn(1986) Peter W. Glynn. Stochastic approximation for monte carlo optimization. In *Winter Simulation Conference*, pages 356–365. Cited in page 39

Goodfellow *et al.***(2016)** Ian Goodfellow, Yoshua Bengio, Aaron Courville and Yoshua Bengio. *Deep learning*. MIT press Cambridge. Cited in page 25, 42, 104

Goodson *et al.***(2017)** Justin C. Goodson, Barrett W. Thomas and Jeffrey W. Ohlmann. A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs. *Eur. J. Oper. Res.*, 258(1):216–229. Cited in page 97

Greff *et al.***(2017)** Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Trans. Neural Networks Learn. Syst.*, 28(10):2222–2232. Cited in page 58

Griewank and Walther(2008) Andreas Griewank and Andrea Walther. *Evaluating derivatives - principles and techniques of algorithmic differentiation, Second Edition*. SIAM. Cited in page 36, 42, 56, 103

Griewank *et al.***(1989)** Andreas Griewank *et al.* On automatic differentiation. *Mathematical Programming: recent developments and applications*, 6(6):83–107. Cited in page 35, 42, 56

Groshev *et al.***(2018)** Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava and Pieter Abbeel. Learning generalized reactive policies using deep neural networks. In *ICAPS*, pages 408–416. AAAI Press. Cited in page 79

Guez *et al.***(2018)** Arthur Guez, Theophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos and David Silver. Learning to search with mctsnets. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1817–1826. PMLR. Cited in page 35

Ha and Schmidhuber(2018) David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, pages 2455–2467. Cited in page 27

Haarnoja *et al.***(2018a)** Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR. Cited in page 26

Haarnoja *et al.***(2018b)** Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905. Cited in page 26

Hafner *et al.***(2019)** Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee and James Davidson. Learning latent dynamics for planning from pixels. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR. Cited in page 27

Hafner *et al.***(2020)** Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi and Jimmy Ba. Mastering atari with discrete world models. *CoRR*, abs/2010.02193. Cited in page 27, 106

Hafner and Riedmiller(2011) Roland Hafner and Martin A. Riedmiller. Reinforcement learning in feedback control - challenges and benchmarks from technical process control. *Mach. Learn.*, 84(1-2):137–169. Cited in page 25

Hansen and Zilberstein(2001) Eric A. Hansen and Shlomo Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62. Cited in page 23, 34

Hansen(2006) Nikolaus Hansen. The CMA evolution strategy: A comparing review. In *Towards a New Evolutionary Computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 75–102. Springer. Cited in page 26

He *et al.***(2016)** Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society. Cited in page 59

Heess *et al.***(2015)** Nicolas Heess, Gregory Wayne, David Silver, Timothy P. Lillicrap, Tom Erez and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *NIPS*, pages 2944–2952. Cited in page 27, 106

Hessel *et al.***(2018)** Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, pages 3215–3222. AAAI Press. Cited in page 25

Hinton *et al.***(2012a)** Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath *et al.* Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97. Cited in page 72

Hinton *et al.***(2012b)** Geoffrey Hinton, Nitish Srivastava and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8). Cited in page 60, 103

Hoey *et al.***(1999)** Jesse Hoey, Robert St-Aubin, Alan J. Hu and Craig Boutilier. SPUDD: stochastic planning using decision diagrams. In *UAI*, pages 279–288. Morgan Kaufmann. Cited in page 23

Hoffmann(2001) Jörg Hoffmann. FF: the fast-forward planning system. *AI Mag.*, 22(3):57–62. Cited in page 89

Hoffmann(2011) Jörg Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *CoRR*, abs/1106.5271. Cited in page 24, 34, 49

Howard(1960) Ronald A Howard. *Dynamic programming and markov processes*. John Wiley. Cited in page 22

Howard and Matheson(2005) Ronald A. Howard and James E. Matheson. Influence diagrams. *Decis. Anal.*, 2(3):127–143. Cited in page 17

Issakkimuthu *et al.***(2015)** Murugeswari Issakkimuthu, Alan Fern, Roni Khardon, Prasad Tadepalli and Shan Xue. Hindsight optimization for probabilistic planning with factored actions. In *ICAPS*, pages 120–128. AAAI Press. Cited in page 79, 90, 92

Ivankovic *et al.***(2014)** Franc Ivankovic, Patrik Haslum, Sylvie Thiébaux, Vikas Shivashankar and Dana S. Nau. Optimal planning with global numerical state constraints. In *ICAPS*. AAAI. Cited in page 24

Jacobson and Mayne(1970) David H Jacobson and David Q Mayne. *Differential dynamic programming*. North-Holland. Cited in page 3, 30

Jaderberg *et al.***(2017)** Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846. Cited in page 105

Jang *et al.***(2017)** Eric Jang, Shixiang Gu and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR (Poster)*. OpenReview.net. Cited in page 75, 106

Jankowiak and Obermeyer(2018) Martin Jankowiak and Fritz Obermeyer. Pathwise derivatives beyond the reparameterization trick. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 2240–2249. PMLR. Cited in page 76

Janner *et al.***(2019)** Michael Janner, Justin Fu, Marvin Zhang and Sergey Levine. When to trust your model: Model-based policy optimization. In *NeurIPS*, pages 12498–12509. Cited in page 26, 106

Jiang *et al.***(2020)** Daniel R. Jiang, Lina Al-Kanj and Warren B. Powell. Optimistic monte carlo tree search with sampled information relaxation dual bounds. *Oper. Res.*, 68(6):1678–1697. Cited in page 90

Kakade(2001) Sham M. Kakade. A natural policy gradient. In *NIPS*, pages 1531–1538. MIT Press. Cited in page 26

Kearns *et al.***(1999)** Michael J. Kearns, Yishay Mansour and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*, pages 1324–1231. Morgan Kaufmann. Cited in page 23, 24

Keller and Eyerich(2012) Thomas Keller and Patrick Eyerich. PROST: probabilistic planning based on UCT. In *ICAPS*. AAAI. Cited in page 24, 34, 71

Keller and Helmert(2013) Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon mdps. In *ICAPS*. AAAI. Cited in page 24

Kingma and Ba(2015) Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*. Cited in page 60

Kingma and Welling(2014) Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*. Cited in page 35, 40, 44, 75

Kocsis and Szepesvári(2006) Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer. Cited in page 3, 23, 24, 34, 49

Konda and Tsitsiklis(1999) Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *NIPS*, pages 1008–1014. The MIT Press. Cited in page 26

Kool *et al.***(2020)** Wouter Kool, Herke van Hoof and Max Welling. Estimating gradients for discrete random variables by sampling without replacement. In *ICLR*. OpenReview.net. Cited in page 106

Korf(1990) Richard E. Korf. Real-time heuristic search. *Artif. Intell.*, 42(2-3):189–211. Cited in page 23

Krizhevsky *et al.***(2012)** Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114. Cited in page 72

Lazic *et al.***(2018)** Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, M. K. Ryu and Greg Imwalle. Data center cooling using model-predictive control. In *NeurIPS*, pages 3818–3827. Cited in page 9

Levine and Abbeel(2014) Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *NIPS*, pages 1071–1079. Cited in page 27

Levine and Koltun(2013) Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1–9. JMLR.org. Cited in page 27

Li and Todorov(2004) Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229. INSTICC Press. Cited in page 4, 30, 34

Lillicrap *et al.***(2016)** Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*. Cited in page 26

Maddison *et al.***(2017)** Chris J. Maddison, Andriy Mnih and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR (Poster)*. OpenReview.net. Cited in page 75, 106

Mansley *et al.***(2011)** Christopher R. Mansley, Ari Weinstein and Michael L. Littman. Sample-based planning for continuous action markov decision processes. In *ICAPS*. AAAI. Cited in page 23, 24, 34

McMahan *et al.***(2005)** H. Brendan McMahan, Maxim Likhachev and Geoffrey J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 569–576. ACM. Cited in page 23

Mercier(2009) Luc Mercier. *Amsaa: An anticipatory algorithm for online stochastic combinatorial optimization*. Tese de Doutorado, Brown University. Cited in page 77

Mercier and Hentenryck(2008) Luc Mercier and Pascal Van Hentenryck. Amsaa: A multistep anticipatory algorithm for online stochastic combinatorial optimization. In *CPAIOR*, volume 5015 of *Lecture Notes in Computer Science*, pages 173–187. Springer. Cited in page 90

Mercier and Hentenryck(2007) Luc Mercier and Pascal Van Hentenryck. Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *IJCAI*, pages 1979–1984. Cited in page 77, 90, 92

Mercier and Hentenryck(2011) Luc Mercier and Pascal Van Hentenryck. An anytime multistep anticipatory algorithm for online stochastic combinatorial optimization. *Annals OR*, 184(1):233–271. Cited in page 90

Mnih and Gregor(2014) Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1791–1799. JMLR.org. Cited in page 35

Mnih *et al.***(2013)** Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602. Cited in page 25, 72

Moerland *et al.***(2020a)** Thomas M. Moerland, Joost Broekens and Catholijn M. Jonker. A framework for reinforcement learning and planning. *CoRR*, abs/2006.15009. Cited in page 105

Moerland *et al.***(2020b)** Thomas M. Moerland, Joost Broekens and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *CoRR*, abs/2006.16712. Cited in page 28

Mohamed *et al.***(2020)** Shakir Mohamed, Mihaela Rosca, Michael Figurnov and Andriy Mnih. Monte carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21:132:1–132:62. Cited in page 39, 40, 75

Nagabandi *et al.***(2018)** Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*, pages 7559–7566. IEEE. Cited in page 28

Narendra and Parthasarathy(1990) Kumpati S. Narendra and Kannan Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks*, 1(1):4–27. Cited in page 72

Ng and Jordan(2000) Andrew Y. Ng and Michael I. Jordan. PEGASUS: A policy search method for large mdps and pomdps. In *UAI*, pages 406–415. Morgan Kaufmann. Cited in page 77

Ng *et al.***(1999)** Andrew Y. Ng, Daishi Harada and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pages 278–287. Morgan Kaufmann. Cited in page 59

Nguyen and Widrow(1990) Derrick H Nguyen and Bernard Widrow. Neural networks for self-learning control systems. *IEEE Control systems magazine*, 10(3):18–23. Cited in page 72

Nocedal and Wright(1999) Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer. Cited in page 52, 53, 59

Pascanu *et al.***(2013)** Razvan Pascanu, Tomás Mikolov and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org. Cited in page 58, 59

Patton *et al.***(2021)** Noah Patton, Jihwan Jeong, Michael Gimelfarb and Scott Sanner. RAPTOR: end-to-end risk-aware MDP planning and policy learning by backpropagation. *CoRR*, abs/2106.07260. Cited in page 107

Pearl(1989) Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann. Cited in page 17

Piotrowski *et al.***(2016)** Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni and Fabio Mercorio. Heuristic planning for hybrid systems. In *AAAI*, pages 4254–4255. AAAI Press. Cited in page 34

Plaat *et al.***(2020)** Aske Plaat, Walter A. Kosters and Mike Preuss. Model-based deep reinforcement learning for high-dimensional problems, a survey. *CoRR*, abs/2008.05598. Cited in page 28

Polydoros and Nalpantidis(2017) Athanasios S. Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *J. Intell. Robotic Syst.*, 86(2): 153–173. Cited in page 28

Powell(2019) Warren B Powell. From reinforcement learning to optimal control: A unified framework for sequential decisions. *arXiv preprint arXiv:1912.03513*. Cited in page 105

Powell(2007) Warren Buckler Powell. *Approximate Dynamic Programming - Solving the Curses of Dimensionality*. Wiley. Cited in page 25

Puterman(1994) Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley. Cited in page 20, 21, 107

Raghavan *et al.***(2017)** Aswin Raghavan, Scott Sanner, Roni Khardon, Prasad Tadepalli and Alan Fern. Hindsight optimization for hybrid state and action mdps. In *AAAI*, pages 3790–3796. AAAI Press. Cited in page 90, 92

Recht(2019) Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279. Cited in page 105

Rezende *et al.***(2014)** Danilo Jimenez Rezende, Shakir Mohamed and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1278–1286. JMLR.org. Cited in page 35

Rickard and Bartholomew(1993) Susan Rickard and Susan Bartholomew. Neurocontrollers for complex systems. In *31st Aerospace Sciences Meeting*, page 5. Cited in page 72

Riedmiller(2005) Martin A. Riedmiller. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328. Springer. Cited in page 25

Robbins and Monro(1951) Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*. Cited in page 25, 55

Roulet *et al.***(2019)** Vincent Roulet, Dmitriy Drusvyatskiy, Siddhartha S. Srinivasa and Zaïd Harchaoui. Iterative linearized control: Stable algorithms and complexity guarantees. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 5518–5527. PMLR. Cited in page 30

Ruder(2016) Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*. Cited in page 50, 52, 55, 60, 103

Ruiz *et al.***(2016)** Francisco J. R. Ruiz, Michalis K. Titsias and David M. Blei. The generalized reparameterization gradient. In *NIPS*, pages 460–468. Cited in page 76

Rumelhart *et al.***(1986)** David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536. Cited in page 35, 42, 56, 103

Russell and Norvig(2002) Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. Cited in page 104

Salimans *et al.***(2017)** Tim Salimans, Jonathan Ho, Xi Chen and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR*, abs/1703.03864. Cited in page 26

Sangiorgio and Guariso(2018) Matteo Sangiorgio and Giorgio Guariso. Nn-based implicit stochastic optimization of multi-reservoir systems management. *Water*, 10(3):303. Cited in page 8

Sanner(2010) Scott Sanner. Relational dynamic influence diagram language (RDDL): Language description. http://users.cecs.anu.edu.au/s̃sanner/IPPC_2011/RDDL.pdf, 2010. Cited in page 18, 81, 106

Sanner *et al.***(2011)** Scott Sanner, Karina Valdivia Delgado and Leliane Nunes de Barros. Symbolic dynamic programming for discrete and continuous state mdps. In *UAI*, pages 643–652. AUAI Press. Cited in page 3, 23, 34

Say(2021) Buser Say. A unified framework for planning with learned neural network transition models. In *AAAI*, pages 5016–5024. AAAI Press. Cited in page 3

Say *et al.***(2017)** Buser Say, Ga Wu, Yu Qing Zhou and Scott Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *IJCAI*, pages 750–756. ijcai.org. Cited in page 3, 34

Scala *et al.***(2016)** Enrico Scala, Patrik Haslum, Sylvie Thiébaux and Miquel Ramírez. Interval-based relaxation for general numeric planning. In *ECAI*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 655–663. IOS Press. Cited in page 49

Scaroni *et al.***(2020)** Renato Scaroni, Thiago Pereira Bueno, Leliane N. de Barros and Denis Deratani Mauá. On the performance of planning through backpropagation. In *BRACIS (2)*, volume 12320 of *Lecture Notes in Computer Science*, pages 108–122. Springer. Cited in page 6, 104

Schrittwieser *et al.***(2020)** Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel *et al.* Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609. Cited in page 27, 105

Schulman(2016) John Schulman. *Optimizing expectations: From deep reinforcement learning to stochastic computation graphs*. Tese de Doutorado, UC Berkeley. Cited in page 35, 103

Schulman *et al.***(2015a)** John Schulman, Nicolas Heess, Theophane Weber and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *NIPS*, pages 3528–3536. Cited in page 35, 36, 38, 40

Schulman *et al.***(2015b)** John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan and Philipp Moritz. Trust region policy optimization. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1889–1897. JMLR.org. Cited in page 26

Schulman *et al.***(2017)** John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347. Cited in page 26

Silver and Veness(2010) David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *NIPS*, pages 2164–2172. Curran Associates, Inc. Cited in page 24

Silver *et al.***(2016)** David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489. Cited in page 27, 72, 105

Silver *et al.***(2017a)** David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel and Demis Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359. Cited in page 27

Silver *et al.***(2017b)** David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David P. Reichert, Neil C. Rabinowitz, André Barreto and Thomas Degris. The predictron: End-to-end learning and planning. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3191–3199. PMLR. Cited in page 27

Srinivas *et al.***(2018)** Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 4739–4748. PMLR. Cited in page 27

Sutskever *et al.***(2014)** Ilya Sutskever, Oriol Vinyals and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112. Cited in page 72

Sutton(1990) Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ML*, pages 216–224. Morgan Kaufmann. Cited in page 26

Sutton(1988) Richard S. Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3:9–44. Cited in page 25

Sutton(1991) Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163. Cited in page 26

Sutton and Barto(2018) Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press. Cited in page 24, 25

Sutton *et al.***(1999)** Richard S. Sutton, David A. McAllester, Satinder P. Singh and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063. The MIT Press. Cited in page 3, 4, 26, 34, 35, 39

Tamar *et al.***(2016)** Aviv Tamar, Sergey Levine, Pieter Abbeel, Yi Wu and Garrett Thomas. Value iteration networks. In *NIPS*, pages 2146–2154. Cited in page 27

Tang *et al.***(2019)** Yichuan Charlie Tang, Jian Zhang and Ruslan Salakhutdinov. Worst cases policy gradients. In *CoRL*, volume 100 of *Proceedings of Machine Learning Research*, pages 1078–1093. PMLR. Cited in page 107

Tassa *et al.***(2007)** Yuval Tassa, Tom Erez and William D. Smart. Receding horizon differential dynamic programming. In *NIPS*, pages 1465–1472. Curran Associates, Inc. Cited in page 30

Tassa *et al.***(2014)** Yuval Tassa, Nicolas Mansard and Emo Todorov. Control-limited differential dynamic programming. In *ICRA*, pages 1168–1175. IEEE. Cited in page 30

Tesauro(1995) Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68. Cited in page 25

Tesauro and Galperin(1996) Gerald Tesauro and Gregory R. Galperin. On-line policy improvement using monte-carlo search. In *NIPS*, pages 1068–1074. MIT Press. Cited in page 24, 97

Theodorou *et al.***(2010)** Evangelos A. Theodorou, Yuval Tassa and Emo Todorov. Stochastic differential dynamic programming. In *ACC*, pages 1125–1132. IEEE. Cited in page 30

Todorov and Li(2005) Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.* Cited in page 30, 34

Tokui and Sato(2017) Seiya Tokui and Issei Sato. Evaluating the variance of likelihood-ratio gradient estimators. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3414–3423. PMLR. Cited in page 40

Tokui and Sato(2016) Seiya Tokui and Issei Sato. Reparameterization trick for discrete variables. *CoRR*, abs/1611.01239. Cited in page 106

Toyer *et al.***(2018)** Sam Toyer, Felipe W. Trevizan, Sylvie Thiébaux and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *AAAI*, pages 6294–6301. AAAI Press. Cited in page 79

UNEP(2013) N UNEP. Adaptation to climate change induced water stress in the nile basin: A vulnerability assessment report, ed. *E. Kironde-Gowa*. Cited in page 8, 9

Van de Water and Willems(1981) Henk Van de Water and J Willems. The certainty equivalence property in stochastic control theory. *IEEE Transactions on Automatic Control*, 26(5):1080–1087. Cited in page 89

Veit *et al.***(2016)** Andreas Veit, Michael J. Wilber and Serge J. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, pages 550–558. Cited in page 59

Vianna *et al.***(2015)** Luis Gustavo Rocha Vianna, Leliane N. de Barros and Scott Sanner. Real-time symbolic dynamic programming. In *AAAI*, pages 3402–3408. AAAI Press. Cited in page 3, 23, 34

Vinter and Vinter(2010) Richard B Vinter and RB Vinter. *Optimal control*. Springer. Cited in page 67

Wang *et al.***(2019)** Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel and Jimmy Ba. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057. Cited in page 28

Watkins and Dayan(1992) Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*. Cited in page 3, 25, 34

Watter *et al.***(2015)** Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, pages 2746–2754. Cited in page 27

Weinstein and Littman(2012) Ari Weinstein and Michael L. Littman. Bandit-based planning and learning in continuous-action markov decision processes. In *ICAPS*. AAAI. Cited in page 24

Werbos(1990) Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560. Cited in page 56

Williams(1992) Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256. Cited in page 26, 35, 43

Williams and Zipser(1995) Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 433:17. Cited in page 3, 4, 56, 57

Wingate and Seppi(2005) David Wingate and Kevin D. Seppi. Prioritization methods for accelerating MDP solvers. *J. Mach. Learn. Res.*, 6:851–881. Cited in page 22

Wolfram *et al.***(1999)** Stephen Wolfram *et al. The MATHEMATICA® book, version 4*. Cambridge university press. Cited in page 42

Wu *et al.***(2017)** Ga Wu, Buser Say and Scott Sanner. Scalable planning with tensorflow for hybrid nonlinear domains. In *NIPS*, pages 6273–6283. Cited in page 4, 6, 34, 49, 50, 52, 54, 60, 68, 80, 103, 104, 105

Yang *et al.***(2021)** Qisong Yang, Thiago D Simão, Simon H Tindemans and Matthijs TJ Spaan. Wcsac: Worst-case soft actor critic for safety-constrained reinforcement learning. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence. AAAI Press, online*. Cited in page 107

Yeh(1985) William W-G Yeh. Reservoir management and operations models: A state-of-the-art review. *Water resources research*, 21(12):1797–1818. Cited in page 80

Yoon *et al.***(2007)** Sung Wook Yoon, Alan Fern and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, page 352. AAAI. Cited in page 89

Yoon *et al.***(2008)** Sung Wook Yoon, Alan Fern, Robert Givan and Subbarao Kambhampati. Probabilistic planning via determinization in hindsight. In *AAAI*, pages 1010–1016. AAAI Press. Cited in page 90, 92

Younes and Littman(2004) Håkan LS Younes and Michael L Littman. Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*. Cited in page 18

Zaeemzadeh *et al.***(2018)** Alireza Zaeemzadeh, Nazanin Rahnavard and Mubarak Shah. Norm-preservation: Why residual networks can become extremely deep? *CoRR*, abs/1805.07477. Cited in page 59

Zamani *et al.***(2012)** Zahra Zamani, Scott Sanner and Cheng Fang. Symbolic dynamic programming for continuous state and action mdps. In *AAAI*. AAAI Press. Cited in page 3, 23, 34

Zaremba and Sutskever(2015) Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *CoRR*, abs/1505.00521. Cited in page 35