Learning Optimal Actions Using Differentiable Simulation

Hashim Al-Obaidi

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2025

Abstract

Existing reinforcement learning algorithms for robotics are sample and time inefficient requiring millions of samples in order to learn a policy [SOURCE]. Research indicates that algorithms that use derivatives obtained from differentiable simulators can reduce the variance associated with policy optimisation, and learn policies more efficiently [SOURCE].

However, there are several challenges related to the use of differentiable simulators, namely the computational cost, as well as the reliability and stability of the computed gradients [SOURCE]

This project aims to compare the accuracy and stability of gradients computed through three methodologies: auto-differentiation via MuJoCo's solver, differentiation through finite differences, and my own implementation based on the implicit function theorem.

The goal was to empirically and experimentally assess the trade-offs associated with each gradient computation method to allow for informed decision-making for trajectory optimization and reinforcement learning applications.

Preliminary results indicate that despite being slower to compute, finite differences produce the most stable gradients. Conversely, gradients obtained through auto-differentiation have been found to be unreliable during contact phases.

My implementation of the implicit function theorem demonstrates the capacity to provide accurate gradients for straightforward experiments; however, it performs inadequately for more complex tasks such as trajectory or policy optimization due to the occurrence of exploding gradients.

The primary conclusion drawn from this research is that further investigation is required to enhance the stability and rapid computation of gradients generated through auto-differentiation, as this would allow gradient-based learning to become a more viable approach.

Acknowledgements

Michael Mistry (Supervisor). Extra thanks to Daniel Layeghi (PhD)

Table of Contents

Chapter 1

Introduction

1.1 Reinforcement Learning

Reinforcement learning came about due to work done by Bellman in the 1960's on dynamic programming and probabilistic Markov chains.

– Introduce notation for RL and key details —

In the 2010's neural networks were taking off, and their application to reinforcement learning revived interest in the field giving birth to a new branch of RL: *Deep Reinforcement Learning*.

This innovation set DeepMind on a track for incredible success - especially within the realm of games. The innovation that set off Google DeepMind's incredible success RL (especially within the realm of games) was the use of *Nueral Networks* to learn Q-values for states **?**. DeepMind later made history with AlphaGo **?**, an RL algorithm trained on the game Go that became the first AI to outperform humans, something long regarded as impossible. DeepMind would later build more advanced AI's to learn Chess, Shogi and more complex video games like Atari **?** by learning a *model* of the game so that it could predict the outcome of a given move, and not just the expected reward.

DeepMind's *value-based* techniques however, are not directly transferable to the realm of robotics. This is because robotics involves far more complex state transitions, with continous actions and in continous time meaning that attempts to learn the value of each state for even simple physical problems quickly becomes infeasible due to the curse of dimensionality [SOURCE].

Instead, reinforcement learning for robotics rely on learning the policy directly (*policy optimisation*) rather than on learning the value of given states and then using this information to construct an optimal policy (although there are hybrid 'actor' 'critic' approaches which are also widely used). [SOURCE?]

Review of Policy Optimization - REINFORCE

The REINFORCE algorithm is a foundational policy gradient method in reinforcement learning that seeks to optimize a parameterized policy π_θ to maximize the expected return $J(\pi_\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$, where $R(\tau)$ denotes the cumulative reward of trajectory τ.

To perform gradient ascent on $J(\pi_\theta)$, we require the gradient $\nabla_\theta J(\pi_\theta)$. Utilizing the log-derivative trick, this gradient can be expressed as:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]$$

In practice, this expectation is approximated using a finite set of sampled trajectories $\mathcal{D} = \{\tau_i\}_{i=1}^N$, leading to the empirical estimate:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau\in\mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

Here, $\hat{g}$ serves as a stochastic estimate of the true gradient $\nabla_\theta J(\pi_\theta)$. This estimation process is referred to as a 0th-order gradient method because it relies solely on sampled trajectories and does not require explicit knowledge of the environment's dynamics or the computation of exact gradients.

Linking with Analytic Policy Gradient Optimization

In contrast to REINFORCE, Analytic Policy Gradient (APG) methods leverage differentiable models of the environment to compute gradients more efficiently. Assuming the state transition function $s_{t+1} = f(s_t, a_t)$ is differentiable, we can apply the chain rule to derive the gradient of the expected return with respect to the policy parameters θ.

The gradient $\nabla_\theta J(\pi_\theta)$ can be decomposed as:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta a_t \frac{\partial s_{t+1}}{\partial a_t} \nabla_{s_{t+1}} R(s_{t+1})\right]$$

Where: - $\nabla_\theta a_t = \nabla_\theta \pi_\theta(a_t|s_t)$ represents the sensitivity of the action with respect to the policy parameters. - $\frac{\partial s_{t+1}}{\partial a_t}$ denotes the Jacobian of the state transition function with respect to the action. - $\nabla_{s_{t+1}} R(s_{t+1})$ is the gradient of the reward function with respect to the next state.

By explicitly computing these derivatives, APG methods can achieve more accurate and potentially lower-variance gradient estimates compared to sampling-based methods like REINFORCE. This approach is particularly advantageous in environments where the dynamics are known and differentiable, allowing for the application of gradient-based optimization techniques directly to the policy parameters.

1.2 Differentiable Simulators

There are many different differentiable simulator's available on the market, and they can be classified by their Gradient-Method, Dynamics-Model, Contact-Model and Integrator. **?**

Policy gradient methods for reinforcement learning as well as trajectory optimisation require the calculation of derivatives of the next state with respect to the previous state and the input.

Chapter 2

Background Chapter

2.1 MuJoCo Computation

The contact model defines how the physics simulator responds to contact with the ground, tools or other objects, yet modelling this realistically is a difficult task due to the inherently non-smooth abrupt nature of contacts.

2.2 Chain Rule and Auto-Differentiation

2.3 Implicit Function Theorem

Chapter 3

Methodology

Environments

One Bounce

The bounce environment consists of a walled enclosure, a ball and the floor. The environment models the ball being projected at a velocity v = [2, 0, -2] i.e. to the right and downwards (without gravity) to the ground and then bouncing back up. This environment and idea comes from **?** and the purpose is to be able to compare the gradients produced against an analytic or ground truth gradient. In **?** the researchers considered the analytic ground truth gradient of d(final$_position$)/d(start$_position$)(bothintermsofyandx), howeverthiswasonly

Research here titled "SimBenchmark" verifies that MuJoCo is unable to simulate elastic collisions (not a paper just a website) https://leggedrobotics.github.io/SimBenchmark/bouncing/. This is because of MuJoCo's soft contact model. Instead MuJoCo let's you define 'solref' and 'solimp' values. I did my best to modify these values to give the most elastic collision.

Even though the final position is not exact, the state transition is. As such, instead of looking at the gradients of the final position, I instead look at the state transition jacobians df/ds where f is the state transition function, and s is the state (q, v).

State Transition:

q' = q + delta$_t$ $*$ vv

Finger

This is an environment taken from Daniel.

Chapter 4

Results

Chapter 5

Conclusion

Chapter 6

Discussion

Appendix A

Appendix

A.1 First section

Any appendices, including any required ethics information, should be included after the references.

Markers do not have to consider appendices. Make sure that your contributions are made clear in the main body of the dissertation (within the page limit).