Reward Redistribution for Reinforcement Learning in Gas Storage Problems

Federico Zilli

Born 9th May 2001 in Udine, Italy

10th July 2026

Master's Thesis Mathematics

Advisor: Prof. Dr. Jochen Garcke

Second Examiner: Dr. Daniel Oeltz

Institut für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Acknowledgments

I would like to start this thesis by thanking Prof. Jochen Garcke for giving me the opportunity to work on this thesis in reinforcement learning, as well as for his comments and suggestions. I also want to thank Dr. Daniel Oeltz for his attentive supervision, the many discussions which were both very helpful and motivating, and for creating a stimulating and supporting working environment.

I am especially grateful to my parents for their constant and unconditional support during my studies. Together with the rest of my family, my sister, my grandmother, my aunt, my uncle, and my two cousins, they always showed me love and affection every time I came back home.

To Nicolò, Thomas, Andrea, Klara, Riccardo, Giacomo, Chiara, Luca, Chiara, Davide, it is difficult to express in words how grateful I am to all of you. Thank you for being there for so many years and for all the memories we have created together.

To Julio, Jimena, Jakob, Joe, Tudor, and Bailee, and the other friends I made in Bonn, a special thank you for the long walks, the dinners and the many hours spent together, chatting, discussing and playing mus. Without you, I would not have loved these years in Bonn nearly as much as I did.

Contents

1 Introduction

Reinforcement learning is one of three basic machine learning paradigms, alongside supervised and unsupervised learning. While the latter two attempt to discover patterns in the data, reinforcement learning deals with the training of an agent through its interaction with the environment, often under uncertainty and delayed rewards. Landmark results include deep reinforcement learning agents that reached human-level performance on Atari games directly from visual input [Mni+15], as well as AlphaGo, which combined deep neural networks with reinforcement learning and self-play to defeat world-class Go players and achieve superhuman play [Sil+16] [Sil+17]. Beyond games, reinforcement learning has been applied to robotics, autonomous driving, resource management, recommendation systems, finance, healthcare, energy systems, and industrial control.

Environments with delayed rewards create significant difficulties for standard reinforcement learning algorithms, as the effects of actions chosen by the agent become visible only at a later time, making it challenging to assign credit or blame. Reward engineering techniques are commonly used to address this problem, for example by adding artificial reward signals to guide the agent toward desired behaviour. This procedure was formalized in [NHR99], with the introduction of potential reward shaping.

Motivated by the widespread success of supervised learning, recent work has asked whether the pattern-recognition capabilities of such models can be leveraged to identify which actions are responsible for future rewards and to assign credit accordingly. The foundational contribution in this direction is RUDDER [Arj+19], where a long short-term memory, a type of recurrent neural network, was used to redistribute the delayed reward to earlier actions.

In this thesis we will investigate whether supervised learning can successfully be applied to redistribute reward in gas storage environments. Gas storage operators face a sequential decision problem: they must decide when to inject gas, when to withdraw it, and when to remain inactive. The value of a decision is often not revealed immediately. Buying gas today may only become profitable many time steps later, when it is eventually sold at a higher price. This delayed feedback makes gas storage a natural example of a temporal credit-assignment problem. We evaluate RUDDER alongside an algorithm developed in this thesis, which uses integrated gradients, an attribution method introduced by [STY17], to redistribute rewards to earlier actions.

The thesis is structured as follows. Chapter 2 introduces the basic concepts of reinforcement learning and the basic algorithms that serve as building blocks for the methods developed

later in the thesis, including Markov decision processes, policy-gradient methods, Proximal Policy Optimization, value functions, and advantage estimation. Chapter 3 provides a theoretical overview of the delayed-reward problem and of reward redistribution methods designed to address it, namely potential reward shaping, RUDDER and integrated gradients. Chapter 4 describes the resulting algorithms in detail and applies them to gas storage environments. The experiments show that RUDDER struggles in both discrete and continuous gas storage settings, while the integrated gradients based method produces more meaningful reward redistributions, requires significantly less training time and can be trained offline.

The contribution of this thesis is threefold. First, we develop a theoretical framework for applying integrated gradients to reinforcement learning. Second, building on this framework, we propose a practical integrated gradients-based algorithm that replaces the long short-term memory network used in RUDDER with a simpler multilayer perceptron, resulting in faster training and inference. Third, we evaluate both RUDDER and our proposed algorithm on gas storage environments, providing an empirical assessment of their performance in this setting.

2 Reinforcement-learning preliminaries

Reinforcement learning studies sequential decision problems in which an agent interacts with an environment and learns how to select actions so as to maximize a cumulative reward. In contrast with supervised learning, the agent is not given a target action at every decision point. Instead, it observes the consequences of its actions through state transitions and rewards. Because an action may influence rewards obtained much later, reinforcement learning must address the problem of temporal credit assignment.

This chapter introduces the reinforcement learning concepts that will be used throughout the thesis. We begin by formalizing the basic reinforcement learning setting, including states, actions, rewards, policies, and value functions. We then discuss policy gradient methods, with particular attention to Proximal Policy Optimization (PPO). Finally, we examine methods for value function estimation, which play a central role in both policy evaluation and modern policy optimization algorithms.

The organization of this chapter is partly inspired by the pedagogical structure of *Spinning Up in Deep Reinforcement Learning* by Achiam [Ach18], which provides an accessible introduction to the main concepts and algorithms of deep reinforcement learning.

2.1 Markov decision processes

In this section we will introduce the basic definitions and concepts in reinforcement learning. For a comprehensive treatment of the subject, see Sutton and Barto [SB18].

We model the interaction between the agent and the environment as a Markov decision process.

Definition 2.1.1 (Markov decision process)**.** *A discounted Markov decision process is a tuple*

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma, \mu_0),$$

where:

(i) $\mathcal{S}$ *is the state space;*

(ii) $\mathcal{A}$ *is the action space;*

(iii) $P(\,\mathrm{d}s' \mid s, a)$ *is a Markov transition kernel on* $\mathcal{S}$*;*

(iv) $\mathcal{R}(\,\mathrm{d}r \mid s, a, s')$ *is the reward distribution;*

(v) $\gamma \in [0, 1]$ *is the discount factor; and*

(vi) μ_0 *is the distribution of the initial state.*

At time t, the agent observes a state $S_t \in \mathcal{S}$, selects an action $A_t \in \mathcal{A}$, receives a reward, and transitions to the next state. We use the convention

$$S_t \sim \mu_0 \quad \text{when } t = 0, \qquad A_t \sim \pi(\cdot \mid S_t), \qquad S_{t+1} \sim P(\cdot \mid S_t, A_t).$$

We denote by R_{t+1} the reward produced by the transition from S_t to S_{t+1}. Its conditional expectation may be written as

$$r(s, a) \coloneqq \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a].$$

More generally, the reward distribution may depend on the next state as well.

Throughout the text, we will use capital letters to indicate random variables, such as S_t, A_t, R_t, while the respective lowercase letters s_t, a_t, r_t denote their realizations.

The Markov property states that, conditional on the current state and action, the distribution of the next state and reward does not depend on the earlier interaction history. If

$$H_t \coloneqq (S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t)$$

denotes the history available at time t, then

$$\mathcal{P}(S_{t+1} \in B, R_{t+1} \in C \mid H_t, A_t) = \mathcal{P}(S_{t+1} \in B, R_{t+1} \in C \mid S_t, A_t)$$

for measurable sets B and C.

A policy describes how the agent chooses its actions.

Definition 2.1.2 (Policy)**.** *A stochastic Markov policy is a conditional distribution*

$$\pi(\,\mathrm{d}a \mid s)$$

over actions given the current state. A deterministic policy is a measurable map $\pi : \mathcal{S} \to \mathcal{A}$ *and may be viewed as the degenerate stochastic policy concentrated at* $\pi(s)$*.*

The policy and the environment jointly induce a distribution over trajectories

$$\tau = (S_0, A_0, R_1, S_1, \ldots, S_T, A_T, R_{T+1}, S_{T+1}).$$

The return summarizes the future rewards following a given time step. For an episode terminating at time T, we define the discounted return from time t by

$$G_t^{(T)} \coloneqq \sum_{k=t}^{T} \gamma^{k-t} R_{k+1}. \tag{2.1}$$

We may omit T when it is clear from the context.

In a continuing problem, the infinite-horizon return is

$$G_t \coloneqq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

When $\gamma < 1$ and the rewards are bounded, this sum is well defined. In a finite-horizon problem, we may also take $\gamma = 1$.

We now introduce value functions, which quantify the expected future return under a policy.

Definition 2.1.3 (State- and action-value functions)**.** *For a policy π, the finite-horizon state-value and action-value functions at time t are*

$$V_t^\pi(s) \coloneqq \mathbb{E}_\pi \left[G_t^{(T)} \,\middle|\, S_t = s \right], \tag{2.2}$$

$$Q_t^\pi(s,a) \coloneqq \mathbb{E}_\pi \left[G_t^{(T)} \,\middle|\, S_t = s, A_t = a \right]. \tag{2.3}$$

In a time-homogeneous infinite-horizon problem, we omit the time subscript and write V^π and Q^π.

The value functions satisfy the Bellman expectation equations. In the infinite-horizon setting,

$$V^\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V^\pi(S_{t+1}) \,|\, S_t = s \right], \tag{2.4}$$

$$Q^\pi(s,a) = \mathbb{E} \left[R_{t+1} + \gamma \mathbb{E}_{A_{t+1} \sim \pi(\cdot|S_{t+1})} \left[Q^\pi(S_{t+1}, A_{t+1}) \right] \,\middle|\, S_t = s, A_t = a \right]. \tag{2.5}$$

In particular,

$$V^\pi(s) = \mathbb{E}_{A \sim \pi(\cdot|s)} \left[Q^\pi(s,A) \right].$$

In the finite-horizon setting, the value functions satisfy the time-dependent Bellman expectation equations. For $t = 0, \ldots, T$,

$$V_t^\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_{t+1}^\pi(S_{t+1}) \,|\, S_t = s \right], \tag{2.6}$$

$$Q_t^\pi(s,a) = \mathbb{E} \left[R_{t+1} + \gamma \mathbb{E}_{A_{t+1} \sim \pi(\cdot|S_{t+1})} \left[Q_{t+1}^\pi(S_{t+1}, A_{t+1}) \right] \,\middle|\, S_t = s,\ A_t = a \right]. \tag{2.7}$$

In particular,

$$V_t^\pi(s) = \mathbb{E}_{A \sim \pi(\cdot|s)} \left[Q_t^\pi(s,A) \right].$$

If no rewards are received after time $T+1$, the terminal condition is

$$V^{\pi}_{T+1}(s) = 0.$$

Equivalently, one may set

$$Q^{\pi}_{T+1}(s,a) = 0.$$

If the problem includes a terminal reward $g(S_{T+1})$, the terminal condition instead becomes

$$V^{\pi}_{T+1}(s) = g(s).$$

The advantage function compares an action with the average action selected by the policy in the same state.

Definition 2.1.4 (Advantage function)**.** *The advantage function of a policy π is*

$$A^{\pi}(s,a) \coloneqq Q^{\pi}(s,a) - V^{\pi}(s).$$

By construction,

$$\mathbb{E}_{A\sim\pi(\cdot|s)}\left[A^{\pi}(s,A)\right] = 0. \tag{2.8}$$

Thus, a positive advantage indicates that an action performs better than the policy's average action at that state, whereas a negative advantage indicates that it performs worse.

2.2 Policy-gradient methods

Value-based methods first estimate an optimal or near-optimal value function and then derive a policy from it. Policy-gradient methods instead optimize a parameterized policy directly. We consider a differentiable family

$$\{\pi_\theta : \theta \in \mathbb{R}^p\},$$

where $\pi_\theta(\,\mathrm{d}a \mid s)$ is a probability distribution over actions. Direct policy optimization is particularly natural when the action space is continuous, when the policy must remain stochastic, or when the policy itself is the primary object of interest.

Our goal is to maximize the expected discounted return

$$J(\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[\sum_{t=0}^{T}\gamma^t R_{t+1}\right] = \mathbb{E}_{\tau\sim p_\theta}\left[G_0^{(T)}\right]. \tag{2.9}$$

The dependence on θ is indirect: changing θ changes the distribution of actions, which changes the distribution of subsequent states and rewards. The key observation behind policy-gradient methods is that we can differentiate an expectation with respect to the parameters of the distribution from which we sample.

We begin with a general identity. Let X be a random variable with differentiable probability density or probability mass function $p_\theta(x)$. We assume that its support does not depend on θ and that differentiation may be interchanged with integration or summation. For details on the precise conditions that allow the interchange in this context, see [Kro11, Theorem 7.1.1, p 124].

Lemma 2.2.1 (Score-function identity)**.** *Let f be a function that does not depend explicitly on θ. Then*

$$\nabla_\theta \mathbb{E}_{X\sim p_\theta}\left[f(X)\right] = \mathbb{E}_{X\sim p_\theta}\left[f(X)\nabla_\theta \log p_\theta(X)\right]. \tag{2.10}$$

Proof. Writing the expectation as an integral, we obtain

$$\begin{aligned}\nabla_\theta \mathbb{E}_{X\sim p_\theta}[f(X)] &= \nabla_\theta \int f(x)p_\theta(x)\,\mathrm{d}x \\ &= \int f(x)\nabla_\theta p_\theta(x)\,\mathrm{d}x. \end{aligned}\tag{2.11}$$

Whenever $p_\theta(x) > 0$, we have

$$\nabla_\theta p_\theta(x) = p_\theta(x)\nabla_\theta \log p_\theta(x).$$

Substituting this identity into Equation (2.11) gives

$$\begin{aligned}\nabla_\theta \mathbb{E}_{X\sim p_\theta}[f(X)] &= \int f(x)p_\theta(x)\nabla_\theta \log p_\theta(x)\,\mathrm{d}x \\ &= \mathbb{E}_{X\sim p_\theta}\left[f(X)\nabla_\theta \log p_\theta(X)\right].\end{aligned}$$

The proof for a discrete random variable is identical, with the integral replaced by a sum. □

The vector

$$\nabla_\theta \log p_\theta(X)$$

is called the score. An immediate and fundamental consequence of Lemma 2.2.1 is that the score has zero expectation.

Lemma 2.2.2 (Expected score)**.** *Under the assumptions of Lemma 2.2.1,*

$$\mathbb{E}_{X\sim p_\theta}\left[\nabla_\theta \log p_\theta(X)\right] = 0. \tag{2.12}$$

Proof. We apply Lemma 2.2.1 with $f(x) = 1$. Since every probability distribution integrates to one,

$$\begin{aligned}\mathbb{E}_{X\sim p_\theta}\left[\nabla_\theta \log p_\theta(X)\right] &= \nabla_\theta \mathbb{E}_{X\sim p_\theta}[1] \\ &= \nabla_\theta \int p_\theta(x)\,\mathrm{d}x \\ &= \nabla_\theta 1 = 0.\end{aligned}$$

Equivalently, we may calculate directly that

$$\int p_\theta(x)\nabla_\theta \log p_\theta(x)\,\mathrm{d}x = \int \nabla_\theta p_\theta(x)\,\mathrm{d}x = \nabla_\theta 1 = 0.$$

□

The same result holds conditionally for a policy. For every state s,

$$\mathbb{E}_{A\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(A \mid s)\right] = 0. \tag{2.13}$$

This apparently simple identity is what allows us to introduce baselines without changing the expected policy gradient.

For a finite-horizon Markov decision process, the trajectory density induced by π_θ is

$$p_\theta(\tau) = \mu_0(S_0)\prod_{t=0}^{T}\pi_\theta(A_t \mid S_t)P(S_{t+1} \mid S_t, A_t), \tag{2.14}$$

We assume that the initial-state distribution and the environment transition kernel do not depend on θ. This factorization is important because the parameters θ affect the trajectory distribution only through the policy.

Applying the score-function identity at the trajectory level yields

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[G_0^{(T)}\nabla_\theta \log p_\theta(\tau)\right]. \tag{2.15}$$

Taking the logarithm of Equation (2.14), we obtain

$$\begin{aligned}\log p_\theta(\tau) = \log \mu_0(S_0) &+ \sum_{t=0}^{T}\log \pi_\theta(A_t \mid S_t) \\ &+ \sum_{t=0}^{T}\log P(S_{t+1} \mid S_t, A_t).\end{aligned} \tag{2.16}$$

Only the policy terms depend on θ, and therefore

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(A_t \mid S_t). \tag{2.17}$$

Substituting Equation (2.17) into Equation (2.15) gives

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(A_t \mid S_t)G_0^{(T)}\right]. \tag{2.18}$$

This expression is correct, but it assigns the complete trajectory return to every action. In particular, it multiplies the score at time t by rewards received before the action A_t

was selected. These earlier rewards cannot have been causally affected by A_t and only add variance to the estimator.

To remove them, write

$$G_0^{(T)} = \sum_{k=0}^{t-1} \gamma^k R_{k+1} + \gamma^t \sum_{k=t}^{T} \gamma^{k-t} R_{k+1}.$$

The second sum is the reward-to-go $G_t^{(T)}$. The first sum depends only on the history available before selecting A_t. Conditional on this history, its product with the policy score has expectation zero. More precisely,

$$\begin{aligned} &\mathbb{E}\left[\nabla_\theta \log \pi_\theta(A_t \mid S_t) \sum_{k=0}^{t-1} \gamma^k R_{k+1}\right] \\ &\quad = \mathbb{E}\left[\left(\sum_{k=0}^{t-1} \gamma^k R_{k+1}\right) \mathbb{E}\left[\nabla_\theta \log \pi_\theta(A_t \mid S_t) \mid H_t\right]\right] \\ &\quad = 0, \end{aligned} \tag{2.19}$$

where the final equality follows from Equation (2.13). Hence,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t=0}^{T} \gamma^t \nabla_\theta \log \pi_\theta(A_t \mid S_t) G_t^{(T)}\right]. \tag{2.20}$$

Equation (2.20) is the basic reward-to-go policy gradient. The explicit factor γ^t follows from the objective in Equation (2.9). It is sometimes omitted by adopting a different finite-horizon objective or by absorbing discount factors into the state occupancy measure. We should therefore state the discounting convention whenever comparing policy-gradient formulas.

2.2.1 Vanilla policy gradients and REINFORCE

Given N trajectories sampled independently from the current policy, we can estimate Equation (2.20) by

$$\widehat{g}_{\mathrm{RF}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T_i} \gamma^t \nabla_\theta \log \pi_\theta\left(A_t^{(i)} \mid S_t^{(i)}\right) G_t^{(i)}. \tag{2.21}$$

Here the superscript (i) indicates the trajectory number. For notational simplicity we omit the horizon T and write $G_t^{(i)}$ instead of $G_t^{(i),(T)}$.

We then perform a stochastic gradient-ascent step

$$\theta_{\mathrm{new}} = \theta_{\mathrm{old}} + \alpha\, \widehat{g}_{\mathrm{RF}}(\theta_{\mathrm{old}}), \tag{2.22}$$

where $\alpha > 0$ is a learning rate. This procedure is commonly called REINFORCE and was introduced in [Wil92]. More generally, the term *vanilla policy gradient* refers to updates based directly on a stochastic estimate of the ordinary Euclidean gradient of the expected return.

The update has an intuitive interpretation. For a sampled state–action pair,

$$\nabla_\theta \log \pi_\theta(A_t \mid S_t)$$

points locally in a direction that increases the log-probability of the sampled action. Multiplying this direction by a positive return reinforces the action, whereas multiplication by a negative return suppresses it. The magnitude of the return determines the size of the update.

This interpretation also exposes a weakness. The absolute return is often a poor measure of whether a particular action was good. For example, in a state from which every action leads to a large positive return, all sampled actions receive a large positive update even if some are much worse than others. Moreover, trajectory returns can vary substantially because of randomness in future transitions and actions. The resulting estimator is unbiased under standard assumptions but may have very high variance.

We can reduce variance by subtracting a baseline. Let b_t be any random variable that may depend on the state, time, or history before A_t is selected, but not on the sampled action A_t itself. Then

$$\mathbb{E}\left[\nabla_\theta \log \pi_\theta(A_t \mid S_t) b_t\right] = 0. \tag{2.23}$$

Indeed, using iterated expectation,

$$\begin{aligned} &\mathbb{E}\left[\nabla_\theta \log \pi_\theta(A_t \mid S_t) b_t\right] \\ &\quad = \mathbb{E}\left[b_t \mathbb{E}\left[\nabla_\theta \log \pi_\theta(A_t \mid S_t) \mid H_t\right]\right] \\ &\quad = 0. \end{aligned} \tag{2.24}$$

Consequently,

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \nabla_\theta \log \pi_\theta(A_t \mid S_t)(G_t^{(T)} - b_t)\right]. \tag{2.25}$$

The baseline changes the variance of the stochastic estimator but not its expectation.

The simplest choice is a constant baseline, such as the mean return in the current batch. A more informative choice is a state-dependent baseline $b_t = b(S_t)$. In that case, the update compares the sampled return with what we normally expect from the current state. A natural choice is

$$b(S_t) = V^{\pi_\theta}(S_t),$$

because

$$\begin{aligned} \mathbb{E}_{\pi_\theta}\left[G_t^{(T)} - V^{\pi_\theta}(S_t) \,\middle|\, S_t = s, A_t = a\right] &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s,a). \end{aligned} \tag{2.26}$$

Thus, subtracting the state value replaces the absolute desirability of an outcome by the relative quality of the chosen action.

It is important that the baseline be independent of the sampled action. An arbitrary action-dependent term $b(s, a)$ does not generally vanish:

$$\mathbb{E}_{A\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(A \mid s) b(s, A)\right] \neq 0.$$

Subtracting such a term without a correction would generally bias the policy gradient.

The baseline does not need to equal the exact value function in order to preserve unbiasedness. Even an imperfect approximation V_ϕ is a valid action-independent baseline. Its accuracy matters because it determines how much variance is removed, not because it determines whether Equation (2.25) is valid.

In practice, we replace the exact advantage by an estimator $\widehat{A}_t$, obtaining the generic vanilla policy-gradient estimator

$$\widehat{g}_{\mathrm{VPG}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i} \gamma^t \nabla_\theta \log \pi_\theta(A_t^{(i)} \mid S_t^{(i)})\widehat{A}_t^{(i)}. \tag{2.27}$$

The advantage estimator may be a Monte Carlo return minus a baseline, a temporal-difference residual, an n-step estimate, or generalized advantage estimation, as we shall see in later sections.

Frequently, implementations normalize the estimated advantages within a batch:

$$\widetilde{A}_t = \frac{\widehat{A}_t - \widehat{\mu}_A}{\widehat{\sigma}_A + \varepsilon_{\mathrm{num}}},$$

where $\widehat{\mu}_A$ and $\widehat{\sigma}_A$ are the batch mean and standard deviation. This operation is a practical rescaling heuristic. It does not form part of the exact policy-gradient theorem, and finite-batch centering and scaling can alter the stochastic update, but it often improves numerical conditioning.

An exact value or advantage function is rarely available. We therefore introduce a critic V_ϕ with parameters ϕ and use it to construct an estimate $\widehat{A}_t$. The policy π_θ is called the actor because it selects actions, while V_ϕ is called the critic because it evaluates the states visited by the actor.

A basic actor–critic update has the form

$$\theta \leftarrow \theta + \alpha_\theta \sum_t \nabla_\theta \log \pi_\theta(A_t \mid S_t)\widehat{A}_t, \tag{2.28}$$

$$\phi \leftarrow \phi - \alpha_\phi \nabla_\phi L^{\mathrm{V}}(\phi). \tag{2.29}$$

The critic loss is typically a regression loss,

$$L^{\mathrm{V}}(\phi) = \mathbb{E}_t\left[\left(V_\phi(S_t) - \widehat{V}_t^{\mathrm{targ}}\right)^2\right].$$

The target may be a Monte Carlo return, a bootstrapped temporal-difference target, an n-step return, or a λ-return. In the next section we will explore these methods in detail.

A learned critic introduces an important distinction. Subtracting $V_\phi(S_t)$ as a baseline does not itself bias the policy gradient, because it remains independent of A_t. However, an advantage estimator that bootstraps through an imperfect critic, may be biased as an estimator of the true advantage. We accept this bias because the reduction in variance can substantially improve learning.

2.2.2 Trust-region policy optimization

Vanilla policy-gradient methods update the parameters in the direction suggested by an estimate of the gradient,

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha \widehat{g}.$$

The learning rate α controls the size of the parameter update, but it does not directly control the size of the corresponding change in the policy.

This distinction is important because the map

$$\theta \longmapsto \pi_\theta(\cdot \mid s)$$

is generally nonlinear. A small change in the parameters may produce a large change in the action probabilities, while a comparatively large parameter change may have little effect on the policy. Consequently, Euclidean distance in parameter space is not a reliable measure of distance between policies.

A large policy update can be harmful for a second reason. Policy-gradient methods estimate an improvement direction using trajectories generated by the current policy $\pi_{\theta_{\text{old}}}$.

If the updated policy differs substantially from $\pi_{\theta_{\text{old}}}$, then the sampled states and actions may no longer be representative of the behavior of the new policy. The local gradient approximation may therefore become inaccurate.

Trust-region policy optimization, or TRPO, addresses this problem by maximizing a local surrogate objective while explicitly limiting the change in the policy distribution. This method was introduced in [Sch+15b].

Suppose that trajectories are collected using $\pi_{\theta_{\text{old}}}$. For each sampled state–action pair, define the likelihood ratio

$$\widehat{\rho}_t(\theta) := \frac{\pi_\theta(A_t \mid S_t)}{\pi_{\theta_{\text{old}}}(A_t \mid S_t)}. \tag{2.30}$$

The ratio compares the probability assigned to the sampled action by the new and old policies. If $\widehat{\rho}_t(\theta) > 1$, the candidate policy assigns more probability to A_t than the behavior policy did; if $\widehat{\rho}_t(\theta) < 1$, it assigns less probability.

Using advantage estimates computed under the old policy, we define the surrogate objective

$$L^{\mathrm{sur}}(\theta) \coloneqq \mathbb{E}_t \left[\widehat{\rho}_t(\theta) \widehat{A}_t \right]. \tag{2.31}$$

This objective has a simple interpretation. If $\widehat{A}_t > 0$, the sampled action appears better than the old policy's average action at S_t, and the surrogate favors increasing its probability. If $\widehat{A}_t < 0$, the action appears worse than average, and the surrogate favors decreasing its probability.

At $\theta = \theta_{\mathrm{old}}$, we have

$$\widehat{\rho}_t(\theta_{\mathrm{old}}) = 1.$$

Moreover,

$$\nabla_\theta \widehat{\rho}_t(\theta) = \widehat{\rho}_t(\theta) \nabla_\theta \log \pi_\theta(A_t \mid S_t),$$

and therefore

$$\nabla_\theta L^{\mathrm{sur}}(\theta)\big|_{\theta=\theta_{\mathrm{old}}} = \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(A_t \mid S_t)\Big|_{\theta=\theta_{\mathrm{old}}} \widehat{A}_t \right]. \tag{2.32}$$

Thus, the surrogate agrees locally with the usual policy-gradient objective.

If we maximize Equation (2.31) without any restriction, however, the likelihood ratios may become very large or very small. The new policy may then move far from the policy that generated the data, precisely where the surrogate is no longer a reliable approximation.

TRPO therefore imposes a trust-region constraint based on the Kullback–Leibler divergence:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \mathbb{E}_t \left[\widehat{\rho}_t(\theta) \widehat{A}_t \right] \\ \text{subject to} \quad & \mathbb{E}_t \left[\mathrm{KL}\left(\pi_{\theta_{\mathrm{old}}}(\cdot \mid S_t) \,\|\, \pi_\theta(\cdot \mid S_t) \right) \right] \le \delta_{\mathrm{KL}}, \end{aligned} \tag{2.33}$$

where $\delta_{\mathrm{KL}} > 0$ is a prescribed trust-region radius.

The KL divergence measures the change in the complete action distribution, rather than merely the change in the probability of the sampled action. The constraint therefore prevents the updated policy from departing too far, on average over the sampled states, from the behavior policy.

For a small parameter displacement

$$\Delta\theta = \theta - \theta_{\mathrm{old}},$$

we approximate the surrogate objective to first order:

$$L^{\mathrm{sur}}(\theta) \approx L^{\mathrm{sur}}(\theta_{\mathrm{old}}) + g^\top \Delta\theta,$$

where

$$g = \nabla_\theta L^{\mathrm{sur}}(\theta)\big|_{\theta=\theta_{\mathrm{old}}}.$$

Around θ_{old}, the average KL divergence admits the quadratic approximation

$$\mathbb{E}_t \left[\mathrm{KL}\left(\pi_{\theta_{\mathrm{old}}}(\cdot \mid S_t) \,\|\, \pi_\theta(\cdot \mid S_t) \right) \right] \approx \frac{1}{2} \Delta\theta^\top F \Delta\theta, \tag{2.34}$$

where F is the Fisher information matrix of the policy.

The local TRPO problem is consequently

$$\begin{aligned} \underset{\Delta\theta}{\text{maximize}} \quad & g^\top \Delta\theta \\ \text{subject to} \quad & \frac{1}{2}\Delta\theta^\top F \Delta\theta \leq \delta_{\text{KL}}. \end{aligned} \tag{2.35}$$

Its solution is proportional to

$$F^{-1}g,$$

the natural-gradient direction. This direction accounts for how changes in the parameters alter the policy distribution. In contrast, the ordinary gradient g treats all parameter directions according to Euclidean geometry.

In practice, TRPO does not explicitly construct or invert the full Fisher matrix. It approximately computes the natural-gradient step using Fisher-vector products and conjugate gradients, and then applies a backtracking line search to verify that the KL constraint is satisfied and that the surrogate objective improves.

TRPO therefore replaces the unrestricted update of vanilla policy gradients with a conservative update. This idea motivates proximal policy optimization. PPO keeps the same likelihood-ratio surrogate but replaces the explicit KL constrained optimization with a simpler clipped objective that can be optimized using ordinary first order stochastic-gradient methods.

2.2.3 Proximal policy optimization

Proximal policy optimization, introduced in [Sch+17] uses the same importance-weighted surrogate as TRPO but replaces the explicit trust-region optimization with an objective that can be optimized by ordinary stochastic gradient methods.

Suppose that we collect a batch of trajectories using $\pi_{\theta_{\text{old}}}$.

The basic surrogate objective is the same as TRPO:

$$L^{\text{sur}}(\theta) = \mathbb{E}_t \left[\widehat{\rho}_t(\theta) \widehat{A}_t \right]. \tag{2.36}$$

PPO limits the strength of gradient upgrades by defining

$$L^{\text{CLIP}}(\theta) \coloneqq \mathbb{E}_t \left[\min \left\{ \widehat{\rho}_t(\theta)\widehat{A}_t,\, \text{clip}(\widehat{\rho}_t(\theta), 1-\varepsilon, 1+\varepsilon)\widehat{A}_t \right\} \right], \tag{2.37}$$

where

$$\text{clip}(x, 1-\varepsilon, 1+\varepsilon) = \begin{cases} 1-\varepsilon, & \text{if } x < 1-\varepsilon, \\ x, & \text{if } 1-\varepsilon \le x \le 1+\varepsilon, \\ 1+\varepsilon, & \text{if } x > 1+\varepsilon. \end{cases}$$

The minimum is best understood by considering the sign of the advantage.

If $\widehat{A}_t > 0$, then the sampled action appears better than the policy's average action at S_t. The objective favors increasing its probability. In this case,

$$\min\left\{\widehat{\rho}_t \widehat{A}_t,\ \text{clip}(\widehat{\rho}_t, 1-\varepsilon, 1+\varepsilon)\widehat{A}_t\right\} = \min\{\widehat{\rho}_t, 1+\varepsilon\}\widehat{A}_t. \tag{2.38}$$

Once $\widehat{\rho}_t$ exceeds $1+\varepsilon$, the clipped term is constant, and the sample no longer provides an incentive to increase the action probability further.

If $\widehat{A}_t < 0$, then the sampled action appears worse than average, and the objective favors reducing its probability. Because multiplication by a negative number reverses inequalities,

$$\min\left\{\widehat{\rho}_t \widehat{A}_t,\ \text{clip}(\widehat{\rho}_t, 1-\varepsilon, 1+\varepsilon)\widehat{A}_t\right\} = \max\{\widehat{\rho}_t, 1-\varepsilon\}\widehat{A}_t. \tag{2.39}$$

Once $\widehat{\rho}_t$ falls below $1-\varepsilon$, the clipped term is constant, and the sample no longer rewards a further reduction in probability.

The clipping is deliberately one-sided. For example, if $\widehat{A}_t > 0$ but the update decreases the action probability, then $\widehat{\rho}_t < 1$ and PPO does not clip the resulting deterioration. Similarly, if $\widehat{A}_t < 0$ but the update increases the probability of the bad action, the unfavorable change remains visible. The objective clips excessively optimistic improvements, not harmful movements. This explains the use of the minimum in Equation (2.37).

Equivalently, for an individual sample,

$$L_t^{\text{CLIP}}(\theta) = \begin{cases} \min\{\widehat{\rho}_t(\theta), 1+\varepsilon\}\widehat{A}_t, & \widehat{A}_t \ge 0, \\ \max\{\widehat{\rho}_t(\theta), 1-\varepsilon\}\widehat{A}_t, & \widehat{A}_t < 0. \end{cases} \tag{2.40}$$

The clipped objective is a pessimistic modification of the unconstrained surrogate:

$$L_t^{\text{CLIP}}(\theta) \le \widehat{\rho}_t(\theta)\widehat{A}_t$$

for every sample. However, clipping is not a hard trust-region constraint. It only removes the objective's incentive to move certain sampled action ratios beyond the interval. Parameter sharing can still cause the overall policy or the KL divergence to change substantially.

PPO is usually implemented as an actor–critic algorithm. The critic is fitted to targets $\widehat{V}_t^{\text{targ}}$ by minimizing

$$L^{\text{V}}(\phi) = \mathbb{E}_t\left[\left(V_\phi(S_t) - \widehat{V}_t^{\text{targ}}\right)^2\right]. \tag{2.41}$$

When generalized advantage estimation is used, a common target is

$$\widehat{V}_t^{\text{targ}} = V_{\phi_{\text{old}}}(S_t) + \widehat{A}_t^{\text{GAE}},$$

where the target is computed before the critic update and then treated as fixed.

An entropy bonus can encourage exploration and prevent the policy from becoming prematurely concentrated:

$$\mathcal{H}(\pi_\theta(\cdot \mid S_t)) = -\mathbb{E}_{A\sim\pi_\theta(\cdot|S_t)}\left[\log \pi_\theta(A \mid S_t)\right]. \tag{2.42}$$

A common combined maximization objective is

$$L^{\text{PPO}}(\theta, \phi) = L^{\text{CLIP}}(\theta) - c_{\text{V}} L^{\text{V}}(\phi) + c_{\text{H}} \mathbb{E}_t\left[\mathcal{H}(\pi_\theta(\cdot \mid S_t))\right], \tag{2.43}$$

with coefficients $c_{\text{V}}, c_{\text{H}} \geq 0$. If the implementation uses minimization, it minimizes the negative of the policy and entropy terms plus the value loss.

Some implementations also clip the value-function update. This is separate from policy-ratio clipping and is an implementation choice rather than an essential part of the PPO policy objective.

A typical PPO iteration proceeds as follows:

(i) We fix the behavior parameters θ_{old} and collect trajectories using $\pi_{\theta_{\text{old}}}$.

(ii) We estimate returns, critic targets, and advantages, commonly using generalized advantage estimation.

(iii) We freeze the old action log-probabilities $\log \pi_{\theta_{\text{old}}}(A_t \mid S_t)$.

(iv) We perform several epochs of minibatch stochastic optimization of Equation (2.43).

(v) We set $\theta_{\text{old}} \leftarrow \theta$ and collect a new batch.

PPO is commonly described as an on-policy algorithm. It reuses each batch for several gradient epochs, but only while keeping the updated policy reasonably close to the behavior policy. Old experience from substantially earlier policies is generally unsuitable without an additional off-policy correction.

Conceptually, the progression from vanilla policy gradients to TRPO and PPO is therefore:

policy-gradient direction

$\Downarrow$

variance reduction through value baselines

$\Downarrow$

TRPO: explicit KL-constrained policy improvement

$\Downarrow$

PPO: first-order conservative surrogate optimization.

Vanilla policy gradients identify a local direction of improvement, TRPO controls the distributional size of the update through a trust region, and PPO approximates this conservative behavior with a simple clipped objective that can be optimized using standard minibatch gradient methods.

2.3 Value-function and Advantage estimation

Policy-gradient and actor-critic methods require estimates of values or advantages. These estimates generally trade bias against variance. Monte Carlo estimators rely on complete sampled returns and have low approximation bias but potentially high variance. Temporal-difference estimators bootstrap from estimated values, which usually lowers variance while introducing bias. Intermediate estimators interpolate between these extremes. This section is inspired by [Sch+15a], where a more in-depth treatment can be found.

2.3.1 Monte Carlo estimation

Given a complete episode, the Monte Carlo target for $V^\pi(S_t)$ is the observed return

$$\widehat{V}_t^{\mathrm{MC}} \coloneqq G_t^{(T)} = \sum_{k=t}^{T} \gamma^{k-t} R_{k+1}. \tag{2.44}$$

A parameterized critic may be fitted by minimizing

$$L^{\mathrm{MC}}(\phi) = \mathbb{E}_t \left[\left(V_\phi(S_t) - G_t^{(T)} \right)^2 \right].$$

Provided that trajectories are sampled from π,

$$\mathbb{E}_\pi[G_t^{(T)} \mid S_t = s] = V_t^\pi(s),$$

so the sampled return is an unbiased target for the finite-horizon value. Nevertheless, its variance may be large because it contains all random rewards and transitions occurring after time t. Monte Carlo estimation also requires waiting until the relevant trajectory segment has terminated.

The corresponding Monte Carlo advantage estimate is

$$\widehat{A}_t^{\mathrm{MC}} = G_t^{(T)} - V_\phi(S_t). \tag{2.45}$$

Although subtracting the baseline reduces the variance of the policy-gradient estimator, errors in the fitted value function affect the quality of the estimated advantage.

2.3.2 Temporal-difference estimation

Temporal-difference methods exploit the Bellman equation and bootstrap from the value estimate at the next state. The one-step temporal-difference target is

$$\widehat{V}_t^{\mathrm{TD}(0)} \coloneqq R_{t+1} + \gamma(1 - D_{t+1})V_{\phi^-}(S_{t+1}), \tag{2.46}$$

where D_{t+1} indicates whether the transition terminates the episode and ϕ^- denotes parameters treated as fixed when constructing the target. The associated temporal-difference residual is

$$\delta_t \coloneqq R_{t+1} + \gamma(1 - D_{t+1})V_\phi(S_{t+1}) - V_\phi(S_t). \tag{2.47}$$

If $V_\phi = V^\pi$, then

$$\mathbb{E}_\pi[\delta_t \mid S_t = s, A_t = a] = A^\pi(s, a).$$

Thus, the TD residual can be used as a one-step advantage estimator.

The critic may be trained by minimizing

$$L^{\mathrm{TD}(0)}(\phi) = \mathbb{E}_t\left[\left(V_\phi(S_t) - \widehat{V}_t^{\mathrm{TD}(0)}\right)^2\right].$$

In a semi-gradient update, we differentiate only through $V_\phi(S_t)$ and treat the bootstrapped target as constant. Compared with the Monte Carlo target, the one-step target generally has lower variance, but it is biased whenever the bootstrapped value estimate is inaccurate.

2.3.3 Multi-step returns

Monte Carlo and one-step TD estimation are endpoints of a family of multi-step estimators. For $n \geq 1$, the n-step return is

$$G_t^{(n)} \coloneqq \sum_{l=0}^{n-1} \gamma^l R_{t+l+1} + \gamma^n V_\phi(S_{t+n}), \tag{2.48}$$

provided that the episode has not terminated before $t + n$. If termination occurs earlier, the bootstrap term is omitted and the reward sum is truncated at the terminal transition. The cases $n = 1$ and $n = T - t + 1$ recover, respectively, the one-step TD target and the episodic Monte Carlo return.

Increasing n reduces the dependence on the approximate value function but introduces more random rewards and transitions. Consequently, larger values of n tend to reduce bootstrap bias while increasing sampling variance.

2.3.4 Generalized advantage estimation

The λ-return combines multi-step returns using exponentially decaying weights. In the infinite-horizon notation, it is

$$G_t^\lambda := (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}, \qquad \lambda \in [0,1]. \tag{2.49}$$

For a finite trajectory, the sum is truncated and the remaining probability mass is assigned to the longest available return. More explicitly, if $m = T - t + 1$, we may define

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{m-1}\lambda^{n-1}G_t^{(n)} + \lambda^{m-1}G_t^{(m)}. \tag{2.50}$$

When $\lambda = 0$, the λ-return reduces to the one-step TD target. When $\lambda = 1$, it reduces to the longest available return, which is the Monte Carlo return for a complete episode.

Generalized advantage estimation constructs an advantage estimate from a discounted sum of temporal-difference residuals. Given

$$\delta_t = R_{t+1} + \gamma(1 - D_{t+1})V_\phi(S_{t+1}) - V_\phi(S_t),$$

we define

$$\widehat{A}_t^{\mathrm{GAE}} := \sum_{l=0}^{T-t}(\gamma\lambda)^l\delta_{t+l}, \qquad \lambda \in [0,1], \tag{2.51}$$

where each term is truncated when a genuine terminal state is reached.

The extreme values of λ recover familiar estimators. For $\lambda = 0$,

$$\widehat{A}_t^{\mathrm{GAE}} = \delta_t,$$

so GAE reduces to the one-step TD advantage estimate. For $\lambda = 1$, the sum telescopes:

$$\begin{aligned}\widehat{A}_t^{\mathrm{GAE}} &= \sum_{l=0}^{T-t}\gamma^l\delta_{t+l} \\ &= G_t^{(T)} - V_\phi(S_t),\end{aligned} \tag{2.52}$$

assuming that the episode terminates at T and that the terminal value is zero. Thus, $\lambda = 1$ yields the Monte Carlo advantage estimate.

For intermediate values of λ, GAE interpolates between the low-variance but potentially biased one-step estimate and the higher-variance Monte Carlo estimate. The parameter γ determines how future rewards enter the task objective, while λ primarily controls the estimator's bias–variance trade-off. Although both appear through the product $\gamma\lambda$ in Equation (2.51), their conceptual roles are distinct.

GAE is closely related to the λ-return. In particular,

$$\widehat{A}_t^{\mathrm{GAE}} = G_t^{\lambda} - V_\phi(S_t), \tag{2.53}$$

when the same bootstrap values and boundary conventions are used. A natural critic target is therefore

$$\widehat{V}_t^{\mathrm{targ}} = V_\phi(S_t) + \widehat{A}_t^{\mathrm{GAE}}. \tag{2.54}$$

In practice, the value on the right-hand side is usually computed using the critic parameters available when the rollout is collected and is then treated as a fixed regression target.

GAE can be evaluated efficiently by a backward recursion. Setting the advantage after the final transition to zero, we compute

$$\widehat{A}_t^{\mathrm{GAE}} = \delta_t + \gamma\lambda(1 - D_{t+1})\widehat{A}_{t+1}^{\mathrm{GAE}}. \tag{2.55}$$

This recursion requires time linear in the trajectory length and is the form typically used in PPO implementations.

The estimators introduced above differ in how strongly they rely on sampled rewards and bootstrapped value predictions:

$$\begin{aligned}
\text{one-step TD:} &\quad R_{t+1} + \gamma V_\phi(S_{t+1}),\\
n\text{-step return:} &\quad \sum_{l=0}^{n-1} \gamma^l R_{t+l+1} + \gamma^n V_\phi(S_{t+n}),\\
\text{Monte Carlo:} &\quad \sum_{l=0}^{T-t} \gamma^l R_{t+l+1},\\
\text{GAE:} &\quad \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l}.
\end{aligned}$$

One-step TD relies heavily on the critic and usually has relatively low variance. Monte Carlo estimation does not bootstrap and is therefore less sensitive to value-function approximation error, but it can have high variance. Multi-step returns, λ-returns, and GAE provide intermediate estimators. PPO commonly uses GAE because it supplies a practical and computationally efficient mechanism for controlling this bias–variance trade-off while producing both actor advantages and critic regression targets.

3 Delayed Rewards and Reward Redistribution

Environments with delayed rewards pose a significant challenge for standard reinforcement learning algorithms, as the consequences of an action may only become observable many time steps later. This chapter examines this problem and develops a theoretical framework for mitigating it through reward redistribution. In particular, it discusses three reward redistribution procedures: potential-based reward shaping, optimal reward redistribution in the sense of RUDDER, and integrated gradients.

3.1 The problem of delayed rewards

This section is inspired by the related discussions in [Arj+19]. Reinforcement learning with significant delayed rewards presents significant challenges to traditional algorithms. To understand why, we can imagine an environment with finite horizon where all reward is given at time $T + 1$, at the end of the episode.

To estimate the value function we can use one of the three techniques described in Section 2.3: Monte Carlo estimation, temporal differences or general advantage estimation. We now describe more precisely how the delayed-reward structure affects these three families of estimators. We will show that, in the presence of long reward delays, they either suffer from high variance, slow propagation of information, or both.

Assume that the horizon is finite and that the only nonzero reward is observed at the end of the episode. That is,

$$R_{t+1} = 0 \quad \text{for } t = 0, \dots, T-1, \qquad R_{T+1} = R(\tau),$$

where τ denotes the full trajectory. For $t \leq T$, the return is therefore

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} R_{k+1} = \gamma^{T-t} R_{T+1}.$$

Consequently, all value estimates at all times are functions of the same terminal random variable. The only difference between the return at different times is the discount factor γ^{T-t}. This already indicates the main difficulty: the reward contains information about the whole trajectory, but it arrives only after all decisions have been made.

For Monte Carlo estimation, the value function is estimated using sampled returns. In the delayed-reward setting, a Monte Carlo target for $V^\pi(S_t)$ is

$$G_t = \gamma^{T-t} R_{T+1}.$$

Thus Monte Carlo estimation remains unbiased, since

$$\mathbb{E}_\pi[G_t \mid S_t = s] = V^\pi(s),$$

provided that V^π is defined with the same finite horizon and discount factor. However, the variance of the target is

$$\mathrm{Var}_\pi(G_t \mid S_t = s) = \gamma^{2(T-t)} \mathrm{Var}_\pi(R_{T+1} \mid S_t = s).$$

This expression shows that Monte Carlo estimation inherits the full conditional variance of the terminal reward, since we cannot decompose the target into local pieces that can be attributed to individual decisions. If many different trajectories pass through the same state s, and if their terminal outcomes vary substantially, then the Monte Carlo estimate of $V^\pi(s)$ can have high variance.

We encounter the same issue more severely when estimating state-action values. For a state-action pair (s, a), the Monte Carlo target is

$$G_t = \gamma^{T-t} R_{T+1},$$

and therefore

$$Q^\pi(s, a) = \mathbb{E}_\pi[\gamma^{T-t} R_{T+1} \mid S_t = s, A_t = a].$$

The action $A_t = a$ may have only a small causal effect on the terminal outcome, or its effect may depend strongly on later states and actions. Nevertheless, the entire realized terminal reward is used as the target for that single action. Hence the estimator assigns the full terminal variability to every action in the trajectory. This is the classical credit-assignment problem: the return reveals whether the trajectory was good or bad, but it does not reveal which decisions were responsible.

The problem becomes even clearer for policy-gradient methods, like PPO. The policy-gradient theorem motivates estimators of the form

$$\widehat{\nabla_\theta J(\theta)} = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(A_t \mid S_t)\, G_t,$$

or, more generally, estimators using an advantage function in place of G_t. In the delayed-reward setting this becomes

$$\widehat{\nabla_\theta J(\theta)} = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(A_t \mid S_t)\, \gamma^{T-t} R_{T+1}.$$

A single terminal random variable multiplies all score terms along the trajectory. Even if the estimator is unbiased, its variance may be large because all policy-gradient terms

are correlated through the same terminal reward. Moreover, when $\gamma < 1$, the gradient contributions from early decisions are attenuated by γ^{T-t}, even though early decisions may be crucial for reaching a successful terminal outcome. Thus Monte Carlo policy-gradient estimation can be statistically inefficient in long-horizon delayed-reward problems.

Temporal-difference methods address part of the Monte Carlo variance problem by replacing full returns with bootstrapped targets. The one-step TD target for the value function is

$$R_{t+1} + \gamma V(S_{t+1}),$$

and the corresponding TD error is

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$

In the delayed-reward setting, for every nonterminal time $t < T$, this reduces to

$$\delta_t = \gamma V(S_{t+1}) - V(S_t).$$

The immediate reward term is identically zero. Therefore, before the final transition, the TD error contains no direct sample of the terminal reward. The only way information about R_{T+1} can influence earlier value estimates is through bootstrapping from later value estimates.

This creates a propagation problem. Suppose, for example, that value estimates are initialized near zero. At the final transition, the TD error is

$$\delta_T = R_{T+1} - V(S_T),$$

assuming that the value after termination is zero. Thus the terminal reward first affects the value estimate near time T. Only after $V(S_T)$ has been updated can the previous value $V(S_{T-1})$ receive a meaningful bootstrapped target. In this way, information about the terminal reward propagates backward one step at a time. For large T, many updates may be needed before early states receive an accurate learning signal.

We can also see this phenomenon by writing the Bellman equation. For $t < T$,

$$V_t^\pi(s) = \mathbb{E}_\pi[\gamma V_{t+1}^\pi(S_{t+1}) \mid S_t = s],$$

whereas at the final time,

$$V_T^\pi(s) = \mathbb{E}_\pi[R_{T+1} \mid S_T = s].$$

The terminal reward appears explicitly only in the boundary condition. All earlier values are obtained by repeated application of the transition operator. Thus the learning problem requires reconstructing a long chain of dependencies from the terminal boundary condition back to the initial states. In exact dynamic programming this is not necessarily problematic, but in sample-based learning with function approximation the process can be slow and unstable.

There is therefore a trade-off. Compared with Monte Carlo estimation, TD estimation can reduce variance because each update uses a local bootstrapped target rather than the full

random return. However, this variance reduction comes at the cost of bias whenever the current value approximation is inaccurate. In delayed-reward problems, the approximation is particularly inaccurate early in learning because the terminal information has not yet propagated backward. Hence TD methods may produce low-variance but uninformative updates for much of the trajectory.

The same issue affects n-step TD returns. The n-step return is

$$G_t^{(n)} = \sum_{k=t}^{t+n-1} \gamma^{k-t} R_{k+1} + \gamma^n V(S_{t+n}).$$

If $t + n - 1 < T$, then all rewards in the explicit sum are zero, and therefore

$$G_t^{(n)} = \gamma^n V(S_{t+n}).$$

Thus an n-step method sees the terminal reward directly only when the n-step window reaches the end of the episode. Otherwise, it still depends entirely on bootstrapping. Larger n allows reward information to travel farther in a single update, but it also makes the estimator closer to Monte Carlo and can increase variance. Smaller n reduces variance but worsens the delay in propagating reward information.

Generalized advantage estimation provides a continuous interpolation between these extremes. Recall that GAE constructs an advantage estimator from exponentially weighted TD errors:

$$\widehat{A}_t^{\mathrm{GAE}} = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l},$$

where

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$

In the delayed-reward setting, all TD errors before the terminal transition have the form

$$\delta_t = \gamma V(S_{t+1}) - V(S_t), \qquad t < T,$$

while the terminal TD error is

$$\delta_T = R_{T+1} - V(S_T).$$

Therefore, the only TD error containing the observed reward directly is δ_T. Its contribution to $\widehat{A}_t^{\mathrm{GAE}}$ is weighted by

$$(\gamma\lambda)^{T-t}.$$

Hence, for early time steps, the direct contribution of the terminal reward to the advantage estimate is exponentially damped in the remaining horizon whenever $\gamma\lambda < 1$.

This observation explains the role of λ. If λ is close to zero, GAE behaves similarly to one-step TD. The resulting estimator has relatively low variance, but it relies strongly on the learned value function to transmit terminal information backward. In delayed-reward environments, this can make the early-time advantages almost entirely determined by the

current critic, rather than by the newly observed terminal reward. If λ is close to one, GAE approaches a Monte Carlo-style estimator. The terminal reward then has a stronger influence on early decisions, but the estimator inherits more of the Monte Carlo variance.

Equivalently, GAE can be interpreted as a weighted average of n-step advantage estimators. In delayed-reward problems, short n-step components do not include the terminal reward unless n is large enough. Therefore, much of the weighted average may consist of bootstrapped targets that have not yet received direct reward information. The parameter λ controls how much mass is placed on longer returns, but no choice of λ fully removes the fundamental tension between variance and temporal propagation.

We encounter therefore theoretical difficulties with all three estimators, but it appears in different forms. Monte Carlo estimation observes the correct terminal outcome immediately for every time step, but it uses a high-variance global target and provides poor temporal credit assignment. Temporal-difference estimation reduces variance by bootstrapping, but the terminal reward must propagate backward through many local updates. Generalized advantage estimation interpolates between the two: it can reduce variance relative to Monte Carlo and propagate information farther than one-step TD, but the contribution of the delayed terminal reward is still discounted by powers of $\gamma\lambda$.

This motivates reward redistribution. The goal is to construct an alternative reward process $(\widetilde{R}_{t+1})_{t=0}^{T}$ that preserves the relevant return while making the reward signal more temporally informative. Ideally, the redistributed rewards satisfy a return-equivalence condition such as

$$\sum_{t=0}^{T} \gamma^t \widetilde{R}_{t+1} = \sum_{t=0}^{T} \gamma^t R_{t+1},$$

or at least preserve this equality in expectation under the policy class of interest. If such a transformation moves reward mass closer to the decisions that caused the outcome, then Monte Carlo estimates can have lower variance, TD methods can receive informative local targets earlier, and GAE can rely less on long chains of bootstrapping or high-variance terminal returns.

3.2 Theoretical Framework

Our goal in this section is to introduce formally the concept of reward redistribution. This section is modelled after [Arj+19, Section A2].

While for the motivational section above we worked with a generic discount rate γ, from this point onward, for simplicity, we restrict the reward-redistribution theory to the finite-horizon undiscounted setting, i.e. $\gamma = 1$. The discounted case requires modified formulas, especially for potential-based shaping.

When we redistribute the reward, our goal is for the optimal policy of the environment to

remain invariant, to avoid cases where the agent learns tricks to maximize the new artificial reward signal, without actually solving task; this is known as *reward hacking*, or *reward gaming*. This motivates the following theory.

Definition 3.2.1 (Return-equivalent Markov decision processes)**.** *Two Markov decision processes $\widetilde{\mathcal{M}}$ and $\mathcal{M}$ are return-equivalent if they differ only in the rewards $\widetilde{r}(s,a)$ and $r(s,a)$, but have the same expected return*

$$\mathbb{E}_\pi\left[\widetilde{G}_0\right] = \mathbb{E}_\pi[G_0]$$

for each policy π. $\widetilde{\mathcal{M}}$ and $\mathcal{M}$ are strictly return-equivalent if they have the same expected return for every episode and for each policy π:

$$\mathbb{E}_\pi\left[\widetilde{G}_0 \,\middle|\, s_0, a_0, \ldots, s_T, a_T\right] = \mathbb{E}_\pi[G_0 \,|\, s_0, a_0, \ldots, s_T, a_T]\,. \tag{3.1}$$

As the name suggests, strictly return-equivalent MDPs are return equivalent.

Proposition 3.2.1. *Strictly return-equivalent Markov decision processes are return equivalent.*

Proof. Suppose $\widetilde{\mathcal{M}}$ and $\mathcal{M}$ are strictly return equivalent, then using the law of total expectation

$$\mathbb{E}_\pi\left[\widetilde{G}_0\right] = \int_{\mathcal{T}} \mathbb{E}_\pi\left[\widetilde{G}_0 \,\middle|\, \tau\right] \pi(\,\mathrm{d}\tau) = \int_{\mathcal{T}} \mathbb{E}_\pi[G_0 \,|\tau]\, \pi(\,\mathrm{d}\tau) = \mathbb{E}_\pi[G_0]$$

.

We used the notation $\tau = (s_0, a_0, \ldots s_T, a_T)$. □

The next proposition shows that return-equivalence is the key property to preserve optimal policies.

Proposition 3.2.2. *Return-equivalent Markov decision processes have the same optimal policies*

Proof. A policy is optimal if it maximizes $\mathbb{E}_\pi[G_0]$, the expected return at time 0. Since for each policy $\mathbb{E}_\pi[G_0] = \mathbb{E}_\pi\left[\widetilde{G}_0\right]$, the optimal policies are the same. □

Now that we have defined return equivalent MDP we can introduce the concept of a reward redistribution.

Definition 3.2.2 (Reward redistribution)**.** *A reward redistribution given an MDP $\mathcal{M}$ is a fixed procedure that, for each state-action sequence $s_0, a_0, \ldots, s_T, a_T$, redistributes the realization of the associated return variable*

$$G_0 = \sum_{t=0}^{T} R_{t+1}$$

or its expectation

$$\mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_T, a_T]$$

along the sequence. The redistribution creates a new SDP $\widetilde{\mathcal{M}}$ with redistributed reward $\widetilde{R}_{t+1}$ at time $(t+1)$ and return variable

$$\widetilde{G}_0 = \sum_{t=0}^{T} \widetilde{R}_{t+1}.$$

The redistribution procedure ensures for each sequence either $G_0 = \widetilde{G}_0$ or

$$\mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_T, a_T] = \mathbb{E}_\pi\left[\widetilde{G}_0 \mid s_0, a_0, \ldots, s_T, a_T\right]. \tag{3.2}$$

In the following sections we will construct different types of reward redistributions.

3.3 Potential Reward Shaping

We will follow the outline given in [NHR99], the work that introduced potential reward shaping.

For the theoretical framework developed in this section, we assume the existence of an *absorbing state* $\bar{s}$, such that the MDP terminates after transitioning to $\bar{s}$. We also assume that all transitions are *proper*, in the sense that, starting from any state and under any policy, the process reaches $\bar{s}$ with probability 1. For simplicity, we restrict attention to finite-horizon MDPs with finite state spaces, where these assumptions are naturally satisfied. Moreover, throughout this section we remain in the undiscounted setting introduced in the previous section. Nevertheless, the theory can be extended to more general settings.

We first start with a general definition of reward shaping.

Definition 3.3.1. *Given a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \mu)$, we can construct a transformed Markov decision process $\widetilde{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, P, \widetilde{R}, \mu)$ which differs only in the reward function. This new reward has the form $\widetilde{R} = R + F$, where $F \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow \mathbb{R}$ is a bounded real valued function, called the* shaping reward function*. The transformation form $\mathcal{M}$ to $\widetilde{\mathcal{M}}$ is called **reward shaping**.*

Therefore, if the original MDP, when transitioning from state s to state s' with action a, would receive a reward $R(s, a, s')$, the new MDP would receive $\widetilde{R}(s, a, s') = R(s, a, s') + F(s, a, s')$. An important observation is that $\widetilde{\mathcal{M}}$ remains an MDP, since the new reward $\widetilde{R}$ is still Markov, that is to say, it does not depend on the history of the trajectory before s.

The natural question now is to find out what conditions need to be enforced on F to guarantee the invariance of optimal policies. To get an idea, we will start by taking a look at some examples of reward hacking.

In [RA98] the authors studied an environment where a simulated bicycle has to ride from a starting point to a target. If the agent reaches the target, it receives a reward inversely proportional to the time taken. This environment suffers severely from delayed rewards, since only one reward is given at the end of the episode. Moreover, the target is very small compared to the size of the environment, so it would take a long time for the agent to reach it by chance for the first time. An estimate of the required time to reach the goal by doing a correlated random walk is 10^{10} steps.

Therefore, shaping the reward signal is a sensible idea, but it must be implemented carefully. In the first experiments the authors rewarded the agent for driving closer to the target, but did not punish it for driving away from it. Consequently, the agent drove in circles around the starting point. The new artificial reward signal led the agent to an optimal policy that was not optimal in the original environment; therefore the new MDP was not return-equivalent with the original one.

This example suggests that a crucial problem in the construction of the new reward signal are positive reward cycles, namely a sequence of states $s_1, s_2, \dots, s_n$ such that, when the agent travels through them cyclically ($s_1 \to s_2 \to \cdots \to s_n \to s1$), it gains a net positive shaped reward, so

$$F(s_1, a_1, s_2) + \cdots + F(s_{n-1}, a_{n-1}, s_n) + F(s_n, a_n, s_1) > 0.$$

These cycles seem to distract the agent and lead to suboptimal policies.

To solve this issue, a simple solution that comes to mind is to use an F obtained as a difference of potentials.

Definition 3.3.2. *We say $F\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow \mathbb{R}$ is a **potential-based** shaping function if there exists a real-valued function $\Phi\colon \mathcal{S} \longrightarrow \mathbb{R}$, such that for every $s \in \mathcal{S} \setminus \bar{s}$, $a \in \mathcal{A}, s' \in \mathcal{S}$, it holds*

$$F(s, a, s') = \Phi(s') - \Phi(s)$$

Potential-based shaping functions are immune to positive reward cycles, since the sums of the shaped reward in cycles become telescopic:

$$F(s1, a1, s2) + \cdots + F(s_{n-1}, a_{n-1}, s_n) + F(s_n, a_n, s_1) = \Phi(s_1) - \Phi(s_1) = 0.$$

Even though in this way we solved the problem of cycles, there could be other phenomena that cause reward hacking and that need additional conditions to be enforced on F. It turns out this is not the case, since, as we can see in the next theorem, potential-based shaping functions preserve the optimal policy.

Theorem 3.3.1. *Let $\mathcal{M}$ be the original MDP, with reward function R, and let $\widetilde{\mathcal{M}}$ be the MDP obtained by adding a shaping reward F. Then F being a potential shaped is a sufficient and necessary condition for it to be consistent with the optimal policies, in the following sense*

- *(Sufficiency) If F is a potential-based shaping function, then every optimal policy in $\widetilde{\mathcal{M}}$ will also be an optimal policy in M, and vice versa.*
- *(Necessity) If F is not a potential-based shaping function, then there exists a proper transition function P and a reward function R such that no optimal policy in $\widetilde{\mathcal{M}}$ is optimal in $\mathcal{M}$.*

Proof. Firstly we prove the sufficiency statement.

Let $\mathcal{M}$ be the original MDP, with reward function R, and let $\widetilde{\mathcal{M}}$ be the MDP obtained by adding the potential-based shaping reward

$$F(s, a, s') = \Phi(s') - \Phi(s).$$

Thus the reward function in $\widetilde{\mathcal{M}}$ is

$$R'(s, a, s') = R(s, a, s') + F(s, a, s') = R(s, a, s') + \Phi(s') - \Phi(s).$$

We prove that every optimal policy for $\widetilde{\mathcal{M}}$ is also optimal for M.

Let Q^*_M denote the optimal action-value function in M, it satisfies the Bellman optimality equation

$$Q^*_M(s, a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[R(s, a, s') + \max_{a' \in A} Q^*_M(s', a') \right].$$

Define

$$\widehat{Q}_{\widetilde{\mathcal{M}}}(s, a) := Q^*_M(s, a) - \Phi(s).$$

We will show that $\widehat{Q}_{M'}$ satisfies the Bellman optimality equation for $\widetilde{\mathcal{M}}$. Indeed,

$$\begin{aligned}
\widehat{Q}_{\widetilde{\mathcal{M}}}(s,a) &= Q^*_M(s,a) - \Phi(s) \\
&= \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a,s') + \max_{a'\in A} Q^*_M(s',a')\right] - \Phi(s) \\
&= \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a,s') - \Phi(s) + \max_{a'\in A} Q^*_M(s',a')\right] \\
&= \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a,s') + \Phi(s') - \Phi(s) + \max_{a'\in A}\big(Q^*_M(s',a') - \Phi(s')\big)\right] \\
&= \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R'(s,a,s') + \max_{a'\in A} \widehat{Q}_{\widetilde{\mathcal{M}}}(s',a')\right].
\end{aligned}$$

Therefore $\widehat{Q}_{\widetilde{\mathcal{M}}}$ satisfies the Bellman optimality equation for $\widetilde{\mathcal{M}}$.

Since the optimal action-value function is uniquely characterized by the Bellman optimality equation, it follows that

$$Q^*_{\widetilde{\mathcal{M}}}(s,a) = \widehat{Q}_{\widetilde{\mathcal{M}}}(s,a) = Q^*_M(s,a) - \Phi(s).$$

Hence, for every state s,

$$\begin{aligned}
\arg\max_{a\in A} Q^*_{\widetilde{\mathcal{M}}}(s,a) &= \arg\max_{a\in A}\big(Q^*_M(s,a) - \Phi(s)\big) \\
&= \arg\max_{a\in A} Q^*_M(s,a),
\end{aligned}$$

because $\Phi(s)$ does not depend on the action a.

Thus the set of optimal actions in each state is the same in M and $\widetilde{\mathcal{M}}$. Consequently, every optimal policy for $\widetilde{\mathcal{M}}$ is also an optimal policy for M.

Now we can prove the necessity statement.

Assume that F is not potential-based. We show that one can construct transition functions P and a reward function R such that no optimal policy in the shaped MDP $\widetilde{\mathcal{M}}$ is optimal in the original MDP $\mathcal{M}$. Throughout the proof we restrict attention to the case $|\mathcal{A}| = 2$, say $\mathcal{A} = \{a, a'\}$. The extension to larger action spaces is immediate by ignoring the remaining actions.

We consider two cases.

Case 1: F depends on actions. Suppose there exist $s \in \mathcal{S}\setminus\{s_0\}$, $s' \in \mathcal{S}$, and $a, a' \in \mathcal{A}$ such that

$$F(s,a,s') \neq F(s,a',s').$$

Without loss of generality, assume

$$F(s,a,s') > F(s,a',s'),$$

and define

$$\Delta := F(s, a, s') - F(s, a', s') > 0.$$

For simplicity, assume first that $s \neq s'$; the case $s = s'$ is analogous, up to minor modification needed to preserve properness.

Construct $\mathcal{M}$ as follows. From state s, both actions lead deterministically to s':

$$P(s' \mid s, a) = P(s' \mid s, a') = 1.$$

Let the rewards be

$$r(s, a, s') = 0, \qquad r(s, a', s') = \frac{\Delta}{2}.$$

All other transitions and rewards may be chosen so that the process is proper. The key step was in selecting the reward: we fixed one reward to 0, and we chose the other one greater than 0, but small enough to let the difference in F be significant enough to change the agent's behaviour.

Then, in the original MDP $\mathcal{M}$, action a' is strictly better than action a at state s, so

$$\pi^*_{\mathcal{M}}(s) = a'.$$

In the shaped MDP $\widetilde{\mathcal{M}}$, the reward is

$$\widetilde{r}(s, a, s') = r(s, a, s') + F(s, a, s'),$$

and similarly for a'. Hence

$$\widetilde{r}(s, a, s') = F(s, a, s'),$$

whereas

$$\widetilde{r}(s, a', s') = \frac{\Delta}{2} + F(s, a', s') = \frac{\Delta}{2} + F(s, a, s') - \Delta = F(s, a, s') - \frac{\Delta}{2}.$$

Therefore

$$\widetilde{r}(s, a, s') > \widetilde{r}(s, a', s'),$$

so in $\widetilde{\mathcal{M}}$ the unique optimal action at s is a:

$$\pi^*_{\widetilde{\mathcal{M}}}(s) = a.$$

Thus no optimal policy in $\widetilde{\mathcal{M}}$ is optimal in $\mathcal{M}$.

Case 2: F depends only on states. It remains to consider shaping functions of the form

$$F(s, a, s') = F(s, s'),$$

which do not depend on the action. Since constant offsets of the reward do not affect optimal policies, we may subtract a constant from F and assume without loss of generality that

$$F(\bar{s}, \bar{s}) = 0,$$

where $\bar{s}$ is the distinguished absorbing state.

Define

$$\Phi(s) := -F(s, \bar{s}).$$

Since F is not potential-based, there exist states $s_1, s_2 \in \mathcal{S}$ such that

$$\Phi(s_2) - \Phi(s_1) \neq F(s_1, s_2).$$

Equivalently,

$$F(s_1, s_2) + F(s_2, \bar{s}) - F(s_1, \bar{s}) \neq 0.$$

Let

$$\Delta := F(s_1, s_2) + F(s_2, \bar{s}) - F(s_1, \bar{s}),$$

so that $\Delta \neq 0$. We construct an MDP $\mathcal{M}$ as follows. From s_1, let

$$P(\bar{s} \mid s_1, a) = 1, \qquad P(s_2 \mid s_1, a') = 1.$$

From both s_2 and $\bar{s}$, both actions lead deterministically to $\bar{s}$. Set

$$r(s_1, a, \bar{s}) = \frac{\Delta}{2}, \qquad r(s, a, s') = 0 \quad \text{for all other transitions.}$$

By construction,

$$V^*_{\mathcal{M}}(\bar{s}) = V^*_{\widetilde{\mathcal{M}}}(\bar{s}) = 0.$$

In the original MDP $\mathcal{M}$, we have

$$Q^*_{\mathcal{M}}(s_1, a) = \frac{\Delta}{2}, \qquad Q^*_{\mathcal{M}}(s_1, a') = 0.$$

In the shaped MDP $\widetilde{\mathcal{M}}$, the corresponding action-values are

$$Q^*_{\widetilde{\mathcal{M}}}(s_1, a) = \frac{\Delta}{2} + F(s_1, \bar{s}),$$

and

$$Q^*_{\widetilde{\mathcal{M}}}(s_1, a') = F(s_1, s_2) + F(s_2, \bar{s}).$$

Using the definition of Δ,

$$F(s_1, s_2) + F(s_2, \bar{s}) = \Delta + F(s_1, \bar{s}),$$

and therefore

$$Q^*_{\widetilde{\mathcal{M}}}(s_1, a') = \Delta + F(s_1, \bar{s}).$$

Hence

$$Q^*_{\widetilde{\mathcal{M}}}(s_1, a') - Q^*_{\widetilde{\mathcal{M}}}(s_1, a) = \frac{\Delta}{2}.$$

Consequently,

$$\pi^*_{\mathcal{M}}(s_1) = \begin{cases} a, & \text{if } \Delta > 0, \\ a', & \text{if } \Delta < 0, \end{cases}$$

whereas

$$\pi^*_{\widetilde{\mathcal{M}}}(s_1) = \begin{cases} a', & \text{if } \Delta > 0, \\ a, & \text{if } \Delta < 0. \end{cases}$$

Thus the optimal action at s_1 is reversed by the shaping term. Therefore no optimal policy in $\widetilde{\mathcal{M}}$ is optimal in $\mathcal{M}$.

In both cases, if F is not potential-based, there exists an MDP $\mathcal{M}$ for which the shaping transformation changes the set of optimal policies. This proves necessity. □

An important remark when looking at the necessity condition is that it applies only when we do not possess specific knowledge of the transition or reward structure of the environment, since we might be able to exclude the specific $\mathcal{P}$ and R which cause problems with optimal policies. This constraint therefore does not apply, for example, to model-based reinforcement learning.

Remark 3.3.1. *In addition to the cycles' heuristic, another intuitive way to understand why potential-based shaping functions preserve optimal policies is to note that, if we build an MDP with reward function made only of differences of potential, so*

$$R(s, a, s') = \Phi(s') - \Phi(s),$$

than any policy is an optimal policy. To see this, we can look at the total return at time $t = 0$:

$$\mathbb{E}_\pi[G_0] = \mathbb{E}_\pi\left[\sum_{t=0}^{T} \Phi(s_{t+1}) - \Phi(s_t)\right] = \Phi(\bar{s}) - \mathbb{E}_\mu[\Phi(s_0)]\,.$$

Since the expected total return does not depend on the policy, any policy is optimal.

We have shown that potential-based shaping functions are the only way to modify the reward signal preserving the Markov property and keeping the optimal policies consistent, that is, the only way to construct a Markov *reward redistribution*. In the next section we will see the limitations of this approach: if we are willing to relax the requirement that the newly constructed reward signal should be Markov, we can obtain better theoretical results. Moreover, while potential-based shaping functions are extremely useful in practical applications, designing them requires knowledge of the environment, which is not always available.

3.4 Sequence Markov Decision Processes and Optimal Reward Redistribution

In this section, we investigate what an optimal reward redistribution is and the theory behind its construction, following [Arj+19].

Firstly, we define the expected sum of delayed rewards.

Definition 3.4.1. *For $1 \leq t \leq T$ and $0 \leq m \leq T - t$, the expected sum of delayed rewards at time $t-1$ in the interval $[\,t+1, t+m+1]$ is defined as*

$$\kappa(m, t-1) \coloneqq \mathbb{E}_\pi \left[\sum_{\tau=0}^{m} R_{t+1+\tau} \,\middle|\, s_{t-1}, a_{t-1} \right].$$

Moreover, for simplicity of notation, we define

$$\kappa(t) \coloneqq \kappa(T-t-1, t).$$

Intuitively, $\kappa(m, t)$ measures the amount of expected reward that is not given immediately at time t, but in the following m time steps. In addition, $\kappa(t)$ measures all the expected delayed reward. Therefore we can write the Bellman equations for Q-values as

$$q^\pi(s_t, a_t) = r(s_t, a_t) + \kappa(t).$$

To remove the problem of delayed rewards, we would like to have an environment where all reward is given immediately, in other words, on average the agent encounters no total additional reward, so $\kappa(t) = 0$. This motivates the following definition of optimal reward redistribution.

Definition 3.4.2. *A reward redistribution is **optimal** if $\kappa(t) = 0$, for all $0 \leq t \leq T-1$*

In the previous section, we saw that potential-based reward shaping is the only method to create a Markov reward redistribution, so it is natural to ask whether it is always an optimal reward redistribution. The next theorem shows that this is not the case, since, generally, an optimal reward redistribution is not Markov.

Theorem 3.4.1 (Optimal reward redistribution need not be Markov)**.** *In general, an optimal reward redistribution violates the assumption that the reward distribution is Markov.*

Proof. Consider an MDP $\mathcal{M}$ with redistributed reward $\widetilde{r}$ such that

$$r(s_T, a_T) \neq 0,$$

and suppose that there exist policies with different expected returns at time $t = 0$. If all redistributed reward were given at time $t = 0$, then all policies would have the same expected return at time $t = 0$, contradicting this assumption. Hence, not all redistributed reward can be assigned to time $t = 0$.

In vector notation, the Bellman equation for a fixed policy π is

$$q_t^\pi = r_t + P_{t\to t+1} q_{t+1}^\pi,$$

where $P_{t\to t+1}$ is the row-stochastic transition matrix whose entries are

$$P_{t\to t+1}\big((s_t, a_t), (s_{t+1}, a_{t+1})\big) = P(s_{t+1} \mid s_t, a_t)\, \pi(a_{t+1} \mid s_{t+1}).$$

An optimal reward redistribution requires that the expected future rewards vanish. Therefore,

$$P_{t\to t+1} \widetilde{q}_{t+1}^\pi = 0.$$

Moreover, by optimality of the redistribution we have

$$\widetilde{q}_{t+1}^\pi = \widetilde{r}_{t+1},$$

and hence

$$P_{t\to t+1} \widetilde{r}_{t+1} = 0,$$

where $\widetilde{r}_{t+1}$ denotes the vector with components

$$\widetilde{r}_{t+1}(s_{t+1}, a_{t+1}) = \widetilde{r}(s_{t+1}, a_{t+1}).$$

Now since the redistributed MDPs are return-equivalent, that $r(s_T, a_T) \neq 0$, and that not all redistributed reward is assigned to time $t = 0$. Then there exists some time $t + 1$ such that

$$\widetilde{r}_{t+1} \neq 0.$$

One can construct an MDP $\mathcal{M}$ such that the number of state-action pairs (s_t, a_t) is at least as large as the number of state-action pairs (s_{t+1}, a_{t+1}), and such that the transition matrix $P_{t\to t+1}$ has full rank. In that case,

$$P_{t\to t+1} \widetilde{r}_{t+1} = 0 \qquad \Longleftrightarrow \qquad \widetilde{r}_{t+1} = 0,$$

which contradicts $\widetilde{r}_{t+1} \neq 0$.

Thus, in general, it is impossible to simultaneously require the redistributed reward to be Markov and require the expected future return to be zero. Consequently, an optimal reward redistribution does not, in general, satisfy the Bellman equation. □

The above theorem suggests that we should look into reward redistribution that are not Markov. This motivates the definition of sequence-Markov decision processes, a generalization of Markov decision processes.

Definition 3.4.3. *A sequence-Markov decision process (SDP) is defined as a finite decision process with Markov transition probability, equipped with a Markov policy, but with a reward distribution that does not need to be Markov.*

Now we shall state the same results from Section 3.2, but for SDPs; we will omit the proof, as they are the same as in the MDP framework.

Definition 3.4.4 (Return-equivalent sequence-Markov decision processes)**.** *Two sequence-Markov decision processes $\widetilde{\mathcal{M}}$ and $\mathcal{M}$ are return-equivalent if they differ only in $\widetilde{r}(s,a)$ and $r(s,a)$ but have the same expected return $\widetilde{v}_0^\pi = v_0^\pi$ for each policy π. $\widetilde{\mathcal{M}}$ and $\mathcal{M}$ are strictly return-equivalent if they have the same expected return for every episode and for each policy π:*

$$\mathbb{E}_\pi\left[\widetilde{G}_0 \,\middle|\, s_0, a_0, \ldots, s_T, a_T\right] = \mathbb{E}_\pi[G_0 \,|\, s_0, a_0, \ldots, s_T, a_T]\,. \tag{3.3}$$

Proposition 3.4.1. *Strictly return-equivalent sequence-Markov decision processes are return equivalent.*

Proposition 3.4.2. *Return-equivalent sequence-Markov decision processes have the same optimal policy*

Definition 3.4.5 (Reward redistribution for SDP)**.** *A reward redistribution given an SDP $\mathcal{M}$ is a fixed procedure that redistributes, for each state-action sequence $s_0, a_0, \ldots, s_T, a_T$, the realization of the associated return variable*

$$G_0 = \sum_{t=0}^{T} R_{t+1}$$

or its expectation

$$\mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_T, a_T]$$

along the sequence. The redistribution creates a new SDP $\widetilde{\mathcal{M}}$ with redistributed reward $\widetilde{R}_{t+1}$ at time $(t+1)$ and return variable

$$\widetilde{G}_0 = \sum_{t=0}^{T} \widetilde{R}_{t+1}.$$

The redistribution procedure ensures for each sequence either $G_0 = \widetilde{G}_0$ or

$$\mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_T, a_T] = \mathbb{E}_\pi\left[\widetilde{G}_0 \mid s_0, a_0, \ldots, s_T, a_T\right]. \tag{3.4}$$

From now on when mentioning a reward redistribution we will intend it in this last sense, so we will assume, unless specified, that the new decision process is a general SDP, not necessarily an MDP. In this new sense a reward redistribution can be very general, since it can depend explicitly on the whole sequence. A useful subclass is causal reward redistributions, where the redistributed reward does not depend on future information.

Definition 3.4.6. *A reward redistribution is* ***causal*** *if, for the redistributed reward* $\widetilde{R}_{t+1}$*, the following holds*

$$\mathbb{E}\Big[\widetilde{R}_{t+1}\Big|s_0, a_0, \ldots, s_T, a_T\Big] = \mathbb{E}\Big[\widetilde{R}_{t+1}\Big|s_0, a_0, \ldots, s_t, a_t\Big]\,.$$

Moreover, a special case that will be useful to us is second order Markov reward redistributions.

Definition 3.4.7. *A (causal) reward redistribution is* ***second order Markov*** *if, for the redistributed reward* $\widetilde{R}_{t+1}$*, the following holds*

$$\mathbb{E}\Big[\widetilde{R}_{t+1}\Big|s_0, a_0, \ldots, s_T, a_T\Big] = \mathbb{E}\Big[\widetilde{R}_{t+1}\Big|s_0, a_0, \ldots, s_t, a_t\Big] = \mathbb{E}\Big[\widetilde{R}_{t+1}\Big|s_{t-1}, a_{t-1}, s_t, a_t\Big]\,.$$

Our goal now is to gain insight into what an optimal reward redistribution looks like. Firstly, we state a useful lemma.

Lemma 3.4.1 (Strict return equivalence preserves conditional expected returns)**.** *Let* $\widetilde{\mathcal{P}}$ *and* $\mathcal{P}$ *be two strictly return-equivalent SDPs. Then, for every state-action sub-sequence*

$$(s_0, a_0, \ldots, s_t, a_t), \qquad 0 \leq t \leq T,$$

we have

$$\mathbb{E}_\pi\Big[\widetilde{G}_0 \mid s_0, a_0, \ldots, s_t, a_t\Big] = \mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_t, a_t]\,.$$

Proof. Fix $0 \leq t \leq T$ and a state-action sub-sequence $(s_0, a_0, \ldots, s_t, a_t)$.

By the law of total expectation, by conditioning on the future state-action sequence, we get

$$\mathbb{E}_\pi\Big[\widetilde{G}_0 \mid s_0, a_0, \ldots, s_t, a_t\Big] = \sum_{s_{t+1}, a_{t+1}, \ldots, s_T, a_T} p_\pi(s_{t+1}, a_{t+1}, \ldots, s_T, a_T \mid s_t, a_t) \\ \mathbb{E}_\pi\Big[\widetilde{G}_0 \mid s_0, a_0, \ldots, s_T, a_T\Big]\,.$$

Since $\widetilde{\mathcal{P}}$ and $\mathcal{P}$ are strictly return-equivalent, the conditional expected return agrees on every complete state-action trajectory. Hence

$$\mathbb{E}_\pi\Big[\widetilde{G}_0 \mid s_0, a_0, \ldots, s_T, a_T\Big] = \mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_T, a_T]\,.$$

Substituting this identity into the previous display yields

$$\mathbb{E}_\pi\Big[\widetilde{G}_0 \mid s_0, a_0, \ldots, s_t, a_t\Big] = \sum_{s_{t+1}, a_{t+1}, \ldots, s_T, a_T} p_\pi(s_{t+1}, a_{t+1}, \ldots, s_T, a_T \mid s_t, a_t) \\ \mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_T, a_T]\,.$$

Applying the law of total expectation once more, and using the Markov property of the state-action process under π, the right-hand side marginalizes to

$$\mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_t, a_t]\,.$$

Therefore,

$$\mathbb{E}_\pi\Big[\widetilde{G}_0 \mid s_0, a_0, \ldots, s_t, a_t\Big] = \mathbb{E}_\pi[G_0 \mid s_0, a_0, \ldots, s_t, a_t]\,,$$

as we claimed. □

We can assume that the starting environment gives rewards only at the end of episodes. If this were not true, we could always transform the original MDP into one with this property by keeping track of the cumulative reward along trajectories and just assign it at the end of the episode. Although the idea is simple and intuitive, proving it rigorously is tedious and does not provide much insight, so we will omit the proof.

We have seen that the Markov property is too strict a condition to impose on the reward redistribution. The natural next candidate to consider is second-order Markov, the simplest non-Markov type of reward redistribution. The next theorem shows that this is indeed enough.

Theorem 3.4.2 (Optimal reward redistribution)**.** *Let $\mathcal{M}$ be a delayed-reward MDP in which the accumulated reward is given only at the end of the episode. Let $\widetilde{\mathcal{M}}$ be the SDP obtained from $\mathcal{M}$ by a reward redistribution that is second-order Markov. By definition of reward redistribution, $\widetilde{\mathcal{M}}$ is strictly return-equivalent to $\mathcal{M}$.*

Then the following two statements are equivalent:

(i) The reward redistribution is optimal, i.e.

$$\kappa(t) = 0.$$

(ii) For every $1 \le t \le T$,

$$\mathbb{E}\Big[\widetilde{R}_{t+1} \,\Big|\, s_{t-1}, a_{t-1}, s_t, a_t\Big] = q^\pi(s_t, a_t) - q^\pi(s_{t-1}, a_{t-1}).$$

Moreover, if the reward redistribution is optimal, then for every $1 \le t \le T$ and every $0 \le m \le T - t$,

$$\kappa(m, t-1) = 0.$$

Proof. We first prove that optimality implies the stated one-step reward representation. Assume that the reward redistribution is optimal, that is,

$$\kappa(t) = 0.$$

Since the redistributed reward $\widetilde{R}_{t+1}$ is second-order Markov, we write

$$h_t := \mathbb{E}\Big[\widetilde{R}_{t+1} \,\Big|\, s_{t-1}, a_{t-1}, s_t, a_t\Big].$$

By return-equivalence of $\mathcal{M}$ and $\widetilde{\mathcal{M}}$, for every state-action subsequence $(s_0, a_0, \ldots, s_t, a_t)$ with $0 \le t \le T$,

$$\mathbb{E}_\pi[G_0 \,|\, s_0, a_0, \ldots, s_t, a_t] = \mathbb{E}_\pi\Big[\widetilde{G}_0 \,\Big|\, s_0, a_0, \ldots, s_t, a_t\Big],$$

where

$$G_0 := R_{T+1}, \qquad \widetilde{G}_0 := \sum_{\tau=0}^{T} \widetilde{R}_{\tau+1}.$$

The Markov property of the original delayed-reward MDP $\mathcal{M}$ gives

$$\mathbb{E}_\pi\left[\sum_{\tau=0}^{T-t} R_{t+1+\tau} \,\middle|\, s_t, a_t\right] = \mathbb{E}_\pi\left[\sum_{\tau=0}^{T-t} R_{t+1+\tau} \,\middle|\, s_0, a_0, \ldots, s_t, a_t\right].$$

Since the original reward is delayed, the only nonzero original reward is the terminal reward, and hence

$$\sum_{\tau=0}^{T-t} R_{t+1+\tau} = R_{T+1}.$$

Moreover, the second-order Markov property of the redistributed process $\widetilde{\mathcal{M}}$ implies that the future redistributed reward from time $t+2$ onward is independent of the past subsequence $(s_0, a_0, \ldots, s_{t-1}, a_{t-1})$ conditional on (s_t, a_t):

$$\mathbb{E}_\pi\left[\sum_{\tau=0}^{T-t-1} \widetilde{R}_{t+2+\tau} \,\middle|\, s_t, a_t\right] = \mathbb{E}_\pi\left[\sum_{\tau=0}^{T-t-1} \widetilde{R}_{t+2+\tau} \,\middle|\, s_0, a_0, \ldots, s_t, a_t\right].$$

Using these identities, we obtain

$$\begin{aligned}
q^\pi(s_t, a_t) &= \mathbb{E}_\pi\left[\sum_{\tau=0}^{T-t} R_{t+1+\tau} \,\middle|\, s_t, a_t\right] \\
&= \mathbb{E}_\pi[R_{T+1} \,|\, s_0, a_0, \ldots, s_t, a_t] \\
&= \mathbb{E}_\pi[G_0 \,|\, s_0, a_0, \ldots, s_t, a_t] \\
\text{(Lemma 3.4.1)} \quad &= \mathbb{E}_\pi\Big[\widetilde{G}_0 \,\Big|\, s_0, a_0, \ldots, s_t, a_t\Big] \\
&= \mathbb{E}_\pi\left[\sum_{\tau=0}^{T} \widetilde{R}_{\tau+1} \,\middle|\, s_0, a_0, \ldots, s_t, a_t\right] \\
&= \mathbb{E}_\pi\left[\sum_{\tau=0}^{T-t-1} \widetilde{R}_{t+2+\tau} \,\middle|\, s_t, a_t\right] + \sum_{\tau=0}^{t} h_\tau \\
&= \kappa(T-t-1, t) + \sum_{\tau=0}^{t} h_\tau \\
&= \sum_{\tau=0}^{t} h_\tau. \qquad (3.5)
\end{aligned}$$

Therefore,

$$h_t = q^\pi(s_t, a_t) - q^\pi(s_{t-1}, a_{t-1}),$$

or equivalently,

$$\mathbb{E}\Big[\widetilde{R}_{t+1} \,\Big|\, s_{t-1}, a_{t-1}, s_t, a_t\Big] = q^\pi(s_t, a_t) - q^\pi(s_{t-1}, a_{t-1}).$$

Conversely, assume that

$$\mathbb{E}\Big[\widetilde{R}_{t+1} \,\Big|\, s_{t-1}, a_{t-1}, s_t, a_t\Big] = q^\pi(s_t, a_t) - q^\pi(s_{t-1}, a_{t-1}).$$

We show that this implies optimality. First consider $m = 0$. Since the original reward satisfies $r(s_{t-1}, a_{t-1}) = 0$ for $t < T$, the Bellman identity for the original delayed-reward MDP gives

$$q^\pi(s_{t-1}, a_{t-1}) = \sum_{s_t, a_t} p(s_t, a_t \mid s_{t-1}, a_{t-1}) q^\pi(s_t, a_t).$$

Hence, for $1 \le t \le T$,

$$\begin{aligned}
\kappa(0, t-1) &= \mathbb{E}_{s_t, a_t, \widetilde{R}_{t+1}}\Big[\widetilde{R}_{t+1} \,\Big|\, s_{t-1}, a_{t-1}\Big] \\
&= \mathbb{E}_{s_t, a_t}[q^\pi(s_t, a_t) - q^\pi(s_{t-1}, a_{t-1}) \,|\, s_{t-1}, a_{t-1}] \\
&= \sum_{s_t, a_t} p(s_t, a_t \mid s_{t-1}, a_{t-1})\Big(q^\pi(s_t, a_t) - q^\pi(s_{t-1}, a_{t-1})\Big) \\
&= q^\pi(s_{t-1}, a_{t-1}) - q^\pi(s_{t-1}, a_{t-1}) \\
&= 0.
\end{aligned} \tag{3.6}$$

Now let $1 \le t \le T$ and $1 \le m \le T - t$. Then

$$\begin{aligned}
\kappa(m, t-1) &= \mathbb{E}_\pi\left[\sum_{\tau=0}^{m} \widetilde{R}_{t+1+\tau} \,\middle|\, s_{t-1}, a_{t-1}\right] \\
&= \mathbb{E}_\pi\left[\sum_{\tau=0}^{m}\Big(q^\pi(s_{t+\tau}, a_{t+\tau}) - q^\pi(s_{t+\tau-1}, a_{t+\tau-1})\Big) \,\middle|\, s_{t-1}, a_{t-1}\right] \\
&= \mathbb{E}_\pi[q^\pi(s_{t+m}, a_{t+m}) - q^\pi(s_{t-1}, a_{t-1}) \,|\, s_{t-1}, a_{t-1}] \\
&= \mathbb{E}_\pi\left[\mathbb{E}_\pi\left[\sum_{\tau=t+m}^{T} R_{\tau+1} \,\middle|\, s_{t+m}, a_{t+m}\right] \,\middle|\, s_{t-1}, a_{t-1}\right] \\
&\quad - \mathbb{E}_\pi\left[\mathbb{E}_\pi\left[\sum_{\tau=t-1}^{T} R_{\tau+1} \,\middle|\, s_{t-1}, a_{t-1}\right] \,\middle|\, s_{t-1}, a_{t-1}\right] \\
&= \mathbb{E}_\pi[R_{T+1} \,|\, s_{t-1}, a_{t-1}] - \mathbb{E}_\pi[R_{T+1} \,|\, s_{t-1}, a_{t-1}] \\
&= 0.
\end{aligned} \tag{3.7}$$

Here we used that $R_{t+1} = 0$ for $t < T$, since the original reward is delayed until the end of the episode.

Choosing $m = T - t$ gives

$$\kappa(T - t, t - 1) = 0.$$

Equivalently, setting $t = \tau + 1$ yields

$$\kappa(\tau) = \kappa(T - \tau - 1, \tau) = 0,$$

which is the optimality condition. This proves the converse and also establishes the stronger statement

$$\kappa(m, t - 1) = 0, \qquad 1 \le t \le T, \quad 0 \le m \le T - t.$$

□

The next theorem gives us a way to express the Q-values of the original process $\mathcal{M}$ with the Q-values of $\widetilde{\mathcal{M}}$, which are much easier to estimate, since all expected reward is given immediately.

Theorem 3.4.3 (Q-values under an optimal reward redistribution)**.** *Assume the setup of Theorem 3.4.2, where $\mathcal{M}$ is the original delayed-reward MDP and $\widetilde{\mathcal{M}}$ is the second-order Markov reward redistribution. If the reward redistribution is optimal, then the Q-values of the redistributed SDP $\widetilde{\mathcal{M}}$ are given by*

$$\begin{aligned} \widetilde{q}^{\pi}(s_t, a_t) &= \widetilde{r}(s_t, a_t) \\ &= q^{\pi}(s_t, a_t) - \mathbb{E}_{s_{t-1}, a_{t-1}}[q^{\pi}(s_{t-1}, a_{t-1}) \,|\, s_t] \\ &= q^{\pi}(s_t, a_t) - \psi^{\pi}(s_t), \end{aligned} \tag{3.8}$$

where

$$\psi^{\pi}(s_t) := \mathbb{E}_{s_{t-1}, a_{t-1}}[q^{\pi}(s_{t-1}, a_{t-1}) \,|\, s_t]\,.$$

Moreover, the original delayed-reward MDP $\mathcal{M}$ and the redistributed SDP $\widetilde{\mathcal{M}}$ have the same advantage function.

Proof. The first equality comes directly from the definition of optimal reward redistribution:

$$\widetilde{q}^{\pi}(s_t, a_t) = \widetilde{r}(s_t, a_t) + \widetilde{\kappa}(t) = \widetilde{r}(s_t, a_t)$$

To get the second equality in the theorem's statement, we use the equivalence in Theorem 3.4.2. For $0 \le t \le T$, with s_{-1} and a_{-1} denoting formal initial symbols and with $q^{\pi}(s_{-1}, a_{-1}) = 0$, we have

$$\begin{aligned} \widetilde{r}(s_t, a_t) &= \mathbb{E}\Big[\widetilde{R}_{t+1} \,\Big|\, s_t, a_t\Big] \\ &= \mathbb{E}_{s_{t-1}, a_{t-1}}\Big[\mathbb{E}\Big[\widetilde{R}_{t+1} \,\Big|\, s_{t-1}, a_{t-1}, s_t, a_t\Big] \Big| s_t, a_t\Big] \qquad (3.9) \\ &= \mathbb{E}_{s_{t-1}, a_{t-1}}[q^{\pi}(s_t, a_t) - q^{\pi}(s_{t-1}, a_{t-1}) \,|\, s_t, a_t] \\ &= q^{\pi}(s_t, a_t) - \mathbb{E}_{s_{t-1}, a_{t-1}}[q^{\pi}(s_{t-1}, a_{t-1}) \,|\, s_t, a_t]\,. \qquad (3.10) \end{aligned}$$

It only remains to show that the conditional expectation depends only on s_t and not on a_t. By Bayes' rule

$$\begin{aligned} p(s_{t-1}, a_{t-1} \mid s_t, a_t) &= \frac{p(s_t, a_t \mid s_{t-1}, a_{t-1}) p(s_{t-1}, a_{t-1})}{p(s_t, a_t)} \\ &= \frac{p(s_t \mid s_{t-1}, a_{t-1}) \pi(a_t \mid s_t) p(s_{t-1}, a_{t-1})}{p(s_t) \pi(a_t \mid s_t)} \\ &= \frac{p(s_t \mid s_{t-1}, a_{t-1}) p(s_{t-1}, a_{t-1})}{p(s_t)} \\ &= p(s_{t-1}, a_{t-1} \mid s_t). \end{aligned} \tag{3.11}$$

Thus the posterior does not depend on a_t. Consequently,

$$\begin{aligned} \mathbb{E}_{s_{t-1}, a_{t-1}}[q^\pi(s_{t-1}, a_{t-1}) \,|\, s_t, a_t] &= \sum_{s_{t-1}, a_{t-1}} p(s_{t-1}, a_{t-1} \mid s_t, a_t) q^\pi(s_{t-1}, a_{t-1}) \\ &= \sum_{s_{t-1}, a_{t-1}} p(s_{t-1}, a_{t-1} \mid s_t) q^\pi(s_{t-1}, a_{t-1}) \\ &= \mathbb{E}_{s_{t-1}, a_{t-1}}[q^\pi(s_{t-1}, a_{t-1}) \,|\, s_t] \\ &= \psi^\pi(s_t). \end{aligned} \tag{3.12}$$

Therefore,

$$\widetilde{q}^\pi(s_t, a_t) = \widetilde{r}(s_t, a_t) = q^\pi(s_t, a_t) - \psi^\pi(s_t).$$

Finally, the value function of the redistributed SDP is

$$\begin{aligned} \widetilde{v}^\pi(s_t) &= \mathbb{E}_{a_t \sim \pi(\cdot|s_t)}[\widetilde{q}^\pi(s_t, a_t)] \\ &= \mathbb{E}_{a_t \sim \pi(\cdot|s_t)}[q^\pi(s_t, a_t) - \psi^\pi(s_t)] \\ &= v^\pi(s_t) - \psi^\pi(s_t). \end{aligned} \tag{3.13}$$

Hence

$$\begin{aligned} \widetilde{A}^\pi(s_t, a_t) &= \widetilde{q}^\pi(s_t, a_t) - \widetilde{v}^\pi(s_t) \\ &= \big(q^\pi(s_t, a_t) - \psi^\pi(s_t)\big) - \big(v^\pi(s_t) - \psi^\pi(s_t)\big) \\ &= q^\pi(s_t, a_t) - v^\pi(s_t) \\ &= A^\pi(s_t, a_t). \end{aligned} \tag{3.14}$$

Thus $\mathcal{M}$ and $\widetilde{\mathcal{M}}$ have the same advantage function. □

The results of this section show why reward redistribution provides a principled way to address delayed rewards. In general, an optimal redistribution cannot be required to remain Markov, which motivates the broader framework of sequence-Markov decision processes. Within this framework, second-order Markov redistributions are expressive enough to construct an optimal redistribution: the redistributed reward can be interpreted as the change in expected return caused by extending the observed state-action sequence by one step.

Consequently, all expected future reward is removed, while the advantage function, and therefore the policy improvement signal, is preserved.

This gives the theoretical basis for RUDDER. Instead of estimating delayed consequences through long temporal-difference backups, RUDDER attempts to learn the expected return of trajectory prefixes and uses changes in these predictions as redistributed rewards. In the next chapter, we will describe how this idea is implemented algorithmically.

3.5 Integrated Gradients

In this section, we introduce the *Integrated Gradients* attribution method and discuss how it can be applied in the reinforcement learning setting. The goal is to assign credit to individual actions by measuring how changes in those actions affect a model's prediction of the total return.

The core idea is the following. Suppose we have a differentiable function F that predicts the total return of an episode from the complete trajectory of states and actions. Then the derivatives of F with respect to the actions provide local information about how each action influences the predicted return. In the case of a one-dimensional continuous action space, a positive derivative

$$\partial_{a_t} F(s_0, a_0, \ldots, s_T, a_T) > 0$$

indicates that, locally, increasing the action a_t would increase the predicted total return. Conversely, a negative derivative indicates that decreasing a_t would locally increase the predicted return.

While ordinary gradients provide only local sensitivity information, Integrated Gradients extends this idea to a more global attribution. It does so by accumulating gradients along a path from a reference trajectory to the observed trajectory, thereby measuring how each action contributes to the change in predicted return between the two.

3.5.1 Axiomatic Introduction to integrated gradients

In this subsection we will mostly follow [STY17], the paper that introduced integrated gradients.

When introducing the concept of attribution methods and integrated gradients specifically, it is useful to first think of classification problems as a mental model, since it is conceptually simpler than reinforcement learning. We will therefore adopt this approach.

The problem we want to solve is attributing the predictions of a deep neural network to its input features, with the intention of understanding the input-output behaviour of the

network. This understanding is very beneficial for explaining what exactly the network learned, or for debugging purposes. In an image classification problem, for example, we might train a deep neural network to recognize the object the image depicts. When applied to this specific problem, attribution analysis can tell us which pixels in the image were more important in leading the network to its choice of label.

We start by formally defining an attribution.

Definition 3.5.1. *Suppose we have a function $F : \mathbb{R}^n \to [0,1]$ that represents a deep network, and an input $x = (x_1, \dots, x_n) \in \mathbb{R}^n$. An attribution of the prediction at input x relative to a baseline input x' is a vector $A(x, x'; F) = (a_1, \dots, a_n) \in \mathbb{R}^n$ where a_i is the contribution of x_i to the prediction $F(x)$.*

When the baseline or the referenced function is clear, we will omit them and simply write $A(x)$.

To understand the need for a baseline, we can make an analogy with humans' way of reasoning. When we want to assign blame to a possible cause of an event, we often employ counterfactual reasoning, considering the absence of such a cause as a baseline for comparing outcomes. In many, but not all, situations, an ideal baseline is an input feature with as little signal as possible; in image recognition, a common baseline is the black image.

We will use an axiomatic approach to introduce integrated gradients, presenting some properties that an attribution method should have, and then showing that the integrated gradients method satisfies them. In the original paper the authors also claimed necessity, so that path methods, a natural generalization of integrated gradients, are the only methods that satisfy the axioms we introduced. In [LHR22] it has been shown that this is not exactly true, as it holds only when some technical conditions are satisfied. This discussion is too detailed to fit within the scope of this thesis, so we will omit it.

In a linear prediction model of the form

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

a useful quantity to consider, to understand which variable is driving a prediction result, is the product $\widehat{\beta}_i x_i$ between the estimated coefficient and the feature values [Mol25, Chapter 6]. Since gradients are the analog of the model coefficients in the nonlinear case, this suggests considering the product between the gradients and the feature values as a starting point.

This first choice satisfies our first axiom, implementation invariance.

Definition 3.5.2 (Implementation invariance)**.** *An attribution method is implementation invariant if the attribution is not a function of model implementation, but solely of the mathematical mapping of the model's domain to the range.*

This axiom is useful to ensure that, if two different network architectures encode the same function, the attributions to the input data stay the same. The chain rule ensures that methods based on gradients satisfy this axiom: if a function f is expressed with two different architectures, for simplicity imagine

$$f = g_1 \circ h_1 = g_2 \circ h_2,$$

then, when we compute the derivatives with respect to the input feature using backpropagation, we get

$$\frac{\partial g_1}{\partial h_1} \cdot \frac{\partial h_1}{\partial x_i} = \frac{\partial f}{\partial x_i} = \frac{\partial g_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial x_i}.$$

The problem with simply choosing $x \cdot \nabla f$ as an attribution method is that it does not satisfy our second axiom: sensitivity.

An attribution method satisfies Sensitivity(a) if for every input and baseline that differ in one feature but have different predictions, then the differing feature should be given a non-zero attribution. Formally

Definition 3.5.3 (Sensitivity(a))**.** *Suppose x, x' vary in one component, so that $x_i \neq x'_i$ and $x_j = x'_j$ for every $j \neq i$. Then a method satisfies sensitivity(a) if $F(x) \neq F(x')$ implies $A_i(x, x'; F) \neq 0$.*

It is not hard to see why sensitivity(a) is a desirable property for an attribution method: if only one feature differs and the prediction changes, then clearly such feature was responsible for the change, so it cannot have zero attribution.

To show that the product of gradients and input features can break sensitivity(a), we consider a simple example. In Figure 3.1 we can see the function $f(x) = 1 - \text{ReLU}(1 - x)$, this is a one-variable, one-ReLU network. Suppose the baseline is $x' = 0$ and the input $x = 2$. Since there is only one feature, it is clear that x and x' vary only in one component. Moreover, $f(x) = 1$, while $f(x') = 0$; therefore, according to sensitivity(a), the attribution cannot be zero. However, we can see that f is constant for $x > 1$, so the gradient is zero.

We can gain a key insight from this example: the problem is that the gradient only considers the local change around x, and not the changes that occurred on the path from x' to x. This motivates the idea behind the definition of integrated gradients, to replace the gradient with its integral along the path from the baseline to the input.

Definition 3.5.4. *The integrated gradient along the i^{th} dimension for an input x and baseline x' is defined as follows.*

$$\text{IG}_i(x) \coloneqq (x_i - x'_i) \int_0^1 \partial_i F(x' + \alpha(x - x'))\, \mathrm{d}\alpha$$

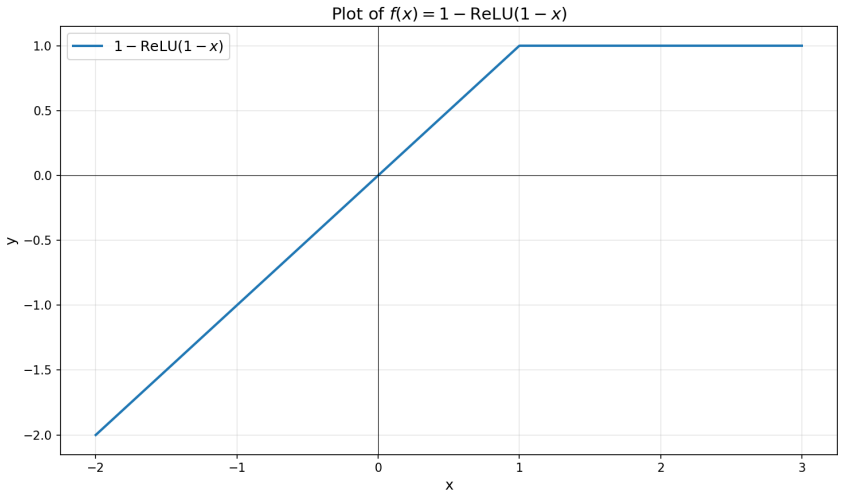


Figure 3.1: Visualization of the function $1 - \text{ReLU}(1 - x)$.

When applied to the previous example in Figure 3.1, we get

$$\text{IG}(2) = (2 - 0) \int_0^1 f'(2\alpha)\, d\alpha = 2 \int_0^{1/2} 1\, d\alpha = 1.$$

The integral of the gradient along the path from $x' = 0$ to $x = 2$ is able to capture the change in the function that occurs between 0 and 1.

To show that integrated gradients satisfies sensitivity(a), like the previous example seems to suggest, we will first prove that it is a *complete* attribution method, which in this setting implies sensitivity(a).

Definition 3.5.5 (Completeness)**.** *An attribution method satisfies the axiom of* completeness *if the attributions add up to the difference in prediction between x and the baseline x', formally*

$$\sum_{i=1}^{n} A_i(x, x'; F) = F(x) - F(x')$$

Proposition 3.5.1. *Integrated Gradients satisfies completeness, so*

$$\sum_{i=1}^{n} \text{IG}_i(x) = F(x) - F(x')$$

Proof. We can obtain the result by a simple application of the fundamental theorem of calculus □

Completeness is a desirable property for an attribution method, since it gives a clear meaning and scale to the value of attribution. It becomes a fundamental property when we do

not just want to understand which features are more significant, but we plan to use the numerical values for a task, as we shall see with reward redistribution.

Proposition 3.5.2. *If for an attribution method $A_j(x, x') = 0$ when $x_j = x'_j$, then completeness of the method implies sensitivity(a).*

Proof. Given x and x' such that $x_i \neq x'_i$ and $x_j = x'_j$ for every $j \neq i$, and $F(x) - F(x') \neq 0$, then

$$A_i(x, x') = \sum_{k=0}^{n} A_k(x, x') = F(x) - F(x') \neq 0.$$

□

In integrated gradients we multiply the integral by $(x_j - x'_j)$, then $A_j(x, x') = 0$ when $x_j = x'_j$, and since integrated gradients is complete, it satisfies sensitivity(a).

Integrated gradients is a special case of a more general class of methods called path methods. Path methods generalize the idea of integrating the gradient along the line segment from the baseline to the input, to any path.

Definition 3.5.6. *Formally, let $\gamma = (\gamma_1, \ldots, \gamma_n) : [0, 1] \to \mathbb{R}^n$ be a smooth function specifying a path in $\mathbb{R}^n$ from the baseline x' to the input x, i.e., $\gamma(0) = x'$ and $\gamma(1) = x$.*

Given a path function γ, path integrated gradients are obtained by integrating the gradients along the path $\gamma(\alpha)$ for $\alpha \in [0, 1]$. Formally, path integrated gradients along the i^{th} dimension for an input x is defined as follows.

$$\text{PIG}_i^{\gamma}(x) \coloneqq \int_{\alpha=0}^{1} \frac{\partial F(\gamma(\alpha))}{\partial \gamma_i(\alpha)} \frac{\partial \gamma_i(\alpha)}{\partial \alpha} \, d\alpha \tag{2}$$

where $\frac{\partial F(x)}{\partial x_i}$ is the gradient of F along the i^{th} dimension at x.

Integrated gradients is the path method for the path

$$\gamma(\alpha) = x' + \alpha(x - x')$$

Remark 3.5.1. *All path methods are implementation invariant, since they are defined using the underlying gradients. Moreover, they satisfy completeness and sensitivity(a). This can be proved using the same arguments as for integrated gradients.*

Now we can state some other axioms which are desirable for attribution methods.

Definition 3.5.7. *(i)* ***Sensitivity(b).*** *When the function F does not depend on a variable x_i, then the attribution $A_i(x, x'; F)$ is always zero.*

(ii) ***Linearity.*** *For every $\alpha, \beta \in \mathbb{R}$ and every function F, G than the attributions are linear in the functions, so*

$$A(x, x'; \alpha F + \beta G) = \alpha A(x, x'; F) + \beta A(x, x'; G).$$

(iii) ***Symmetry-Preserving****. For a given $x = (i, j)$, define x^* by swapping the values of i and j. A function is symmetric in i and j if, for all inputs x, $F(x) = F(x^*)$. Then the attribution method is symmetry preserving if, whenever $x_i = x_j$ and $x'_i = x'_j$, we have $A_i(x, x'; F) = A_j(x, x'; F)$.*

While sensitivity(a) makes sure that attribution is given to features that cause change in prediction, sensitivity(b), its natural complement, ensures that irrelevant features do not receive attribution. Since for such features $\partial_i F = 0$, all path methods satisfy sensitivity(b). The linearity of gradients and integrals guarantees that path methods satisfy linearity.

An example on why symmetry-preservation is a desirable property is the model

$$F(x_1 + x_2 + \cdots + x_n) = \text{Sigmoid}(x_1 + x_2 + \cdots + x_n).$$

Here all variables are symmetric, in other words, they all perform the same function. Hence, whenever, for example, $x_1 = x_2$, there is no reason why the attributions should vary.

It is understandable why integrated gradients is the canonical choice of path method, since it corresponds to the simplest path. However, the axiom of symmetry preservation gives us a more grounded reason to prefer the straight line path over all others, as the next theorem shows.

Theorem 3.5.1. *Integrated gradients is the unique path method that is symmetry-preserving.*

Proof. Firstly, we prove that if the path is a straight line, i.e. we use integrated gradients, then symmetry-preservation holds. Let, without loss of generality, x_1 and x_2 be two symmetric variables for a function $F\colon \mathbb{R}^n \longrightarrow \mathbb{R}$, x' a baseline with $x'_1 = x'_2$ and x an input with $x_1 = x_2$. The key observation is that, along the straight line path $\gamma(\alpha) = x' + \alpha(x - x')$, the first two variables will remain equal to each other, so $\gamma_1(\alpha) = \gamma_2(\alpha)$ for every α. Therefore, the derivatives of F with respect to the first and second variables along the path will remain equal. This implies

$$\text{IG}_1(x) = (x_1 - x'_1) \int_0^1 \partial_1 F(x' + \alpha(x - x'))\, \mathrm{d}\alpha = (x_2 - x'_2) \int_0^1 \partial_2 F(x' + \alpha(x - x'))\, \mathrm{d}\alpha = \text{IG}(x_2)$$

Now we shall prove that any other path method breaks symmetry-preservation. We choose $x' = (0, \ldots, 0)$ and $x = (1, \ldots, 1)$ as baseline and input, respectively.

Consider a non-straightline path

$$\gamma : [0,1] \to \mathbb{R}^n$$

from the baseline to the input. Since the path is not a straight line, there exists $t_0 \in [0,1]$ and two dimensions, which, again for simplicity, we choose to be $i = 1$ and $j = 2$, such that $\gamma_1(t_0) > \gamma_2(t_0)$.

Let (t_1, t_2) be the maximal open interval containing t_0 such that

$$\gamma_1(t) > \gamma_2(t) \qquad \text{for all } t \in (t_1, t_2).$$

Set

$$a = \gamma_1(t_1) = \gamma_2(t_1), \qquad b = \gamma_1(t_2) = \gamma_2(t_2).$$

Define a function $F : [0,1]^n \to \mathbb{R}$ by

$$F(x) = \begin{cases} 0, & \min(x_1, x_2) \le a, \\ (b-a)^2, & \max(x_1, x_2) \ge b, \\ (x_1 - a)(x_2 - a), & \text{otherwise.} \end{cases}$$

Now we compute the attributions of F at the input with respect to the baseline

The function F is symmetric in the coordinates x_i and x_j, so these two coordinates should receive identical attributions. However, along the path γ, the function F is constant outside the interval $[t_1, t_2]$, and therefore the attribution integrand is zero there. For $t \in (t_1, t_2)$, we have

$$\partial_1 F(\gamma(t)) = \gamma_2(t) - a, \qquad \partial_2 F(\gamma(t)) = \gamma_1(t) - a.$$

By construction of the interval (t_1, t_2),

$$\gamma_1(t) > \gamma_2(t) \qquad \text{for all } t \in (t_1, t_2),$$

and hence

$$\gamma_1(t) - a > \gamma_2(t) - a.$$

Thus the attribution integrand for x_j is strictly larger than the attribution integrand for x_1 throughout (t_1, t_2). Integrating over the path, it follows that x_2 receives a strictly larger attribution than x_1.

This contradicts the symmetry of F in x_1 and x_2, which requires the two coordinates to receive identical attributions. Therefore, the desired conclusion follows. □

This theorem justifies our decision to use only integrated gradients out of all path methods.

We can now make a last remark on the practical use of integrated gradients.

Remark 3.5.2. *One of the advantages of integrated gradients over more complex attribution methods is its simplicity of implementation. We only need a scheme to approximate the integral. Usually it is approximated by Riemann sums.*

$$\mathrm{IG}_i(x) \approx (x_i - x_i') \frac{1}{m} \sum_{k=1}^{m} \partial_i F\Big(x' + \frac{k}{m}(x - x')\Big)$$

The most computationally expensive task is to compute the gradients m times, which is very efficiently implemented in modern deep learning frameworks like PyTorch. Computing a good approximation of integrated gradients is therefore quite cheap computationally.

In this section we have shown, through an axiomatic approach, why integrated gradients is a promising attribution method.

3.5.2 Integrated gradients for reinforcement learning

In this section we study whether integrated gradients can be used to construct a reward redistribution from a return-prediction model. We assume that a model F receives an entire rollout as input and predicts the total return of the episode. We then ask whether attributing the prediction of F to individual actions yields redistributed rewards whose sum is equal to the original return. We show that this is true only if all relevant input coordinates are accounted for. We then discuss the additional difficulty that standard integrated gradients evaluate F on interpolated rollouts, which may not be valid trajectories of the environment.

Integrated gradients have been mentioned in the literature as a possible tool for credit assignment in reinforcement learning, most notably in [Arj+19]. However, to the best of our knowledge, the connection between integrated gradients and reward redistribution has not been made explicit in the form developed in this section. In particular, we clarify the role of completeness, show why action-only attributions are generally insufficient for return preservation, and highlight the off-manifold issue induced by standard straight-line interpolation. Moreover, as discussed in the next chapter, we are not aware of prior work that structures the resulting algorithm in the same way.

Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$ be an episodic Markov decision process with horizon T. A trajectory is denoted by

$$\tau = s_0, a_0, s_1, a_1, \ldots, s_T, a_T.$$

For simplicity, we write the it as a vector

$$x = (s_0, a_0, \ldots, s_T, a_T)$$

belonging to a suitable ambient Euclidean space.

We assume that F is a differentiable return-prediction model satisfying

$$F(x) = G_0.$$

In practice, F is learned from data and therefore only approximates G_0. The consequences of this approximation, and its role in the experimental implementation, are discussed in the next chapter.

We also choose a baseline rollout

$$x' = (s'_0, a'_0, \ldots, s'_T, a'_T)$$

such that

$$F(x') = 0.$$

Suppose that we try to define redistributed rewards by using only the action-coordinate attributions:

$$\widetilde{R}_t := \mathrm{IG}_{a_t}(x; x').$$

This construction is tempting because we often want credit to be assigned to actions. However, action-only attributions generally do not preserve returns.

The following proposition makes precise why action-only integrated gradients are not, in general, a reward redistribution.

Proposition 3.5.3 (Action-only integrated gradients are generally incomplete)**.** *Let*

$$x = (s_0, a_0, \ldots, s_T, a_T)$$

and assume that $F(x') = 0$*. If we define*

$$\widetilde{R}_t := \mathrm{IG}_{a_t}(x; x'),$$

then

$$\sum_{t=0}^{T} \widetilde{R}_t = G_0 - \sum_{t=0}^{T} \mathrm{IG}_{s_t}(x; x')$$

whenever $F(x) = G_0$*. Therefore, action-only integrated gradients preserve the return if and only if*

$$\sum_{t=0}^{T} \mathrm{IG}_{s_t}(x; x') = 0.$$

Proof. By the completeness of integrated gradients,

$$\sum_{t=0}^{T} \mathrm{IG}_{s_t}(x; x') + \sum_{t=0}^{T} \mathrm{IG}_{a_t}(x; x') = F(x) - F(x').$$

Using $F(x) = G_0$ and $F(x') = 0$, we obtain

$$\sum_{t=0}^{T} \mathrm{IG}_{a_t}(x; x') = G_0 - \sum_{t=0}^{T} \mathrm{IG}_{s_t}(x; x').$$

Since $\widetilde{R}_t = \mathrm{IG}_{a_t}(x; x')$, the result follows. □

Remark 3.5.3. *The failure in Proposition 3.5.3 is not a failure of integrated gradients. It is a consequence of discarding some of the attributions. Integrated gradients are complete over the input coordinates to which they are applied. If the model F depends on both states and actions, then omitting state attributions generally destroys completeness.*

A return-preserving construction is therefore obtained by assigning all coordinates to time steps.

Definition 3.5.8 (State-action integrated-gradient redistribution)**.** *Assume that the model input is ordered as*

$$x = (s_0, a_0, \ldots, s_T, a_T).$$

We define the state-action integrated-gradient redistribution by

$$\widetilde{R}_t := \mathrm{IG}_{s_t}(x; x') + \mathrm{IG}_{a_t}(x; x'), \qquad t = 0, \ldots, T.$$

Proposition 3.5.4 (Return preservation of state-action IG redistribution)**.** *Assume that $F(x) = G_0$ and $F(x') = 0$. Then the redistribution*

$$\widetilde{R}_t = \mathrm{IG}_{s_t}(x; x') + \mathrm{IG}_{a_t}(x; x')$$

satisfies

$$\sum_{t=0}^{T} \widetilde{R}_t = G_0.$$

Proof. The claim follows immediately from Proposition 3.5.1 after grouping the state and action coordinates belonging to the same time index. □

Depending on the intended temporal interpretation, we may also assign the state attributions differently. For example, if rewards are viewed as belonging to transitions, then it may be more natural to use

$$\widetilde{R}_t := \mathrm{IG}_{a_t}(x; x') + \mathrm{IG}_{s_{t+1}}(x; x'), \qquad t = 0, \ldots, T-1,$$

together with a separate initial-state term

$$\widetilde{R}_{-1} := \mathrm{IG}_{s_0}(x; x').$$

Then

$$\widetilde{R}_{-1} + \sum_{t=0}^{T-1} \widetilde{R}_t + \mathrm{IG}_{a_T}(x; x') = G_0,$$

provided all coordinates are accounted for. Thus the essential requirement is not a particular grouping scheme, but rather that no attribution mass be discarded.

Even though the completeness results in this section were stated for a baseline x' such that $F(x') = 0$, this is not really required. In fact, if

$$F(x') = b \neq 0,$$

then the redistributed return becomes

$$\sum_{t=0}^{T} \widetilde{R}_t = G_0 - b.$$

Subtracting a constant term clearly does not change the optimal policy, and b can be understood as a baseline in the sense of policy gradients, as introduced in Section 2.2. The choice of integrated gradients baselines is highly dependent on the specific environment, therefore we will leave its discussion to the next chapter, where we will describe the algorithm in detail and apply it.

Although the completeness identity is mathematically useful, standard integrated gradients introduce a separate conceptual issue. The attribution is computed along the straight-line path

$$x_\alpha = x' + \alpha(x - x'), \qquad \alpha \in [0, 1].$$

When x and x' are rollouts, the interpolated point x_α may not be a valid trajectory of the environment.

Definition 3.5.9 (Valid rollout)**.** *We say that*

$$\tau = (s_0, a_0, \ldots, s_T, a_T)$$

is a valid rollout if, for each t, the transition from s_t to s_{t+1} is compatible with the environment dynamics. In the deterministic case this means

$$s_{t+1} = P(s_t, a_t),$$

and in the stochastic case this means

$$s_{t+1} \in \operatorname{supp} P(\cdot \mid s_t, a_t).$$

Remark 3.5.4 (Straight-line interpolation can be off-manifold)**.** *Even if both x and x' are valid rollouts, the interpolated input*

$$x_\alpha = (s_0^\alpha, a_0^\alpha, \ldots, s_T^\alpha, a_T^\alpha)$$

need not satisfy

$$s_{t+1}^{\alpha} \in \operatorname{supp} P(\cdot \mid s_t^{\alpha}, a_t^{\alpha}).$$

Thus, standard integrated gradients generally evaluate F on off-manifold inputs. This is especially problematic when states are constrained physical states, image observations, or simulator states, or when actions are discrete and represented by one-hot vectors or embeddings.

This issue does not invalidate the completeness identity. If all coordinates are included, then

$$\sum_i \mathrm{IG}_i(x; x') = F(x) - F(x')$$

still holds. The problem is interpretability: the redistributed rewards may reflect the behavior of F on invalid rollouts rather than causal credit under the dynamics of the MDP. If F is trained only on valid trajectories, its off-manifold gradients may be arbitrary. Therefore, the resulting redistribution can be return-preserving but not causally meaningful.

In this thesis, we will study the standard integrated gradients method to determine if it can effectively redistribute rewards, even with off-manifold interpolation. Further research is needed to develop on-manifold methods and test whether they lead to an improvement in performance.

4 Gas Storage Experiments

In this chapter, we apply the theoretical frameworks discussed in the previous chapters to gas storage environments. We first formally introduce the structure of these environments, before describing in detail the RUDDER algorithm and the integrated gradients based reward redistribution method, drawing respectively on the theory presented in Section 3.4 and Section 3.5.2. We then apply RUDDER to a gas storage environment with a discrete action space, and compare the use of RUDDER and integrated gradients in a gas storage environment with a continuous action space.

We conclude that RUDDER fails to produce a useful redistributed reward signal in this setting. In contrast, integrated gradients yields a highly promising reward redistribution, although it suffers from stability issues.

To the best of our knowledge, the reward redistribution method based on integrated gradients and implemented using a simple multi-layer perceptron is original and has not previously appeared in published work.

4.1 The Gas Storage Problem

We formulate the gas storage problem in a form suitable for reinforcement learning. We consider a storage facility that can buy gas from the market, inject it into storage, keep it for later use, and withdraw it when market conditions are favorable. The purpose of the optimization problem is to find an operating policy that decides, at each time step, whether we should inject gas, withdraw gas, or do nothing.

We work on a finite time grid

$$t = 0, 1, \ldots, T,$$

where T is the final decision date. At each time t, we observe a one-dimensional gas price $P_t \in \mathbb{R}_+$ and the current inventory level X_t. The price P_t may be interpreted as a spot price or as any other one-dimensional price signal used in the model. In more advanced formulations, one may replace P_t by a vector of market variables, for example a forward curve, but this is not needed for the setting considered here.

The state of the system is therefore

$$S_t = (X_t, P_t, t). \tag{4.1}$$

We include the time index t in the state because the problem is finite-horizon and because gas storage decisions are typically seasonal.

The action A_t represents the amount of gas injected into or withdrawn from storage during the interval $[t, t+1]$. We use the sign convention

$$A_t > 0 \quad \text{for injection,} \qquad A_t < 0 \quad \text{for withdrawal,} \qquad A_t = 0 \quad \text{for no operation.}$$

The inventory evolves according to

$$X_{t+1} = X_t + A_t. \tag{4.2}$$

The storage facility has a finite capacity. We write

$$\underline{x} \leq X_t \leq \overline{x}, \tag{4.3}$$

where $\underline{x}$ and $\overline{x}$ are the minimum and maximum inventory levels. The minimum level will be zero in all experiments, although in real life application it may be higher due to technical constraints such as cushion gas. The maximum level is the physical storage capacity.

The action is also constrained by injection and withdrawal limits. Given the current inventory level x, we define the feasible action set by

$$\mathcal{A}(x) := \{a \in \mathbb{R} : -w_{\max} \leq a \leq i_{\max},\ \underline{x} \leq x + a \leq \overline{x}\}. \tag{4.4}$$

Here $i_{\max} \geq 0$ is the maximum amount that can be injected and $w_{\max} \geq 0$ is the maximum amount that can be withdrawn.

In the simplest case, the action space may be discretized as

$$A_t \in \{-\Delta x, 0, \Delta x\}, \tag{4.5}$$

where $-\Delta x$ denotes withdrawal, 0 denotes no operation, and Δx denotes injection. In a more accurate model, we may instead allow A_t to take any continuous value in the feasible interval $\mathcal{A}(X_t)$.

The reward is the cashflow generated by the storage decision. If we inject gas, we buy gas at the current price and therefore incur a cost. If we withdraw gas, we sell gas at the current price and therefore receive revenue.

With the sign convention above, the natural immediate reward is

$$r_t(X_t, P_t, A_t) = -P_t A_t \tag{4.6}$$

If $A_t > 0$, then $-P_t A_t$ is negative, because we are buying gas. If $A_t < 0$, then $-P_t A_t$ is positive, because we are selling gas.

At the final time T, we want the storage to finish near a target inventory x_{target}, therefore we add a terminal penalty:

$$\Phi(X_T) = -\lambda_{\text{term}} \left(X_T - x_{\text{target}}\right)^2, \tag{4.7}$$

where $\lambda_{\text{term}} \geq 0$ controls the strength of the penalty. We will choose zero as a target in all experiments, since we want no gas remaining in the storage at the end of the episode.

The objective is to maximize the expected discounted sum of rewards:

$$\sup_{\pi} \mathbb{E}^{\pi} \left[\sum_{t=0}^{T-1} \gamma^t r_t(X_t, P_t, A_t) + \gamma^T \Phi(X_T) \right], \tag{4.8}$$

subject to the inventory dynamics Equation (4.2) and the feasibility constraint

$$A_t \in \mathcal{A}(X_t).$$

The policy π is the operating rule that selects the storage action from the current state.

In the setting considered here, the Markov decision process is defined by:

$$S_t = (X_t, P_t, t), \qquad A_t \in \mathcal{A}(X_t), \qquad R_t = r_t(X_t, P_t, A_t).$$

The transition from S_t to S_{t+1} has two components. First, the inventory transition is deterministic:

$$X_{t+1} = X_t + A_t.$$

Second, the price evolves according to the chosen price model or according to historical/simulated data:

$$P_{t+1} \sim P(\cdot \mid P_t).$$

Thus the next state is

$$S_{t+1} = (X_t + A_t, P_{t+1}, t+1).$$

For one episode, corresponding for example to one storage year, the realized return is

$$G_0 = \sum_{t=0}^{T-1} \gamma^t R_t + \gamma^T \Phi(X_T). \tag{4.9}$$

The reinforcement learning objective is to find a parameterized policy π_θ that maximizes

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t r_t(X_t, P_t, A_t) + \gamma^T \Phi(X_T) \right]. \tag{4.10}$$

The policy takes the current price, inventory, and time as input and outputs an action. In a discrete action setting, the policy may output probabilities for withdrawal, no operation, and injection. In a continuous action setting, the policy may output an amount of gas to inject or withdraw, which must then be restricted to the feasible interval $\mathcal{A}(X_t)$.

Although a gas storage environment is not a delayed-rewards MDP as we defined in Chapter 3, it still suffers from delayed rewards. The key action to earn a profit is to inject when the price is low, but this action receives an immediate negative reward, while the payoff is collected only later, at the time of withdrawal. This motivates our study on the application of reward redistribution methods to gas storage environments.

4.2 Algorithms

In this section we shall outline the structure of the two algorithms of interest, which we will apply to gas storage environments: RUDDER and Integrated Gradients applied to PPO.

As a baseline, we train a policy using proximal policy optimization. We denote the parametrized policy by

$$\pi_\theta(a \mid s).$$

Given data generated by the current policy, PPO updates the policy by maximizing a clipped surrogate objective. Writing

$$\widehat{\rho}_t(\theta) = \frac{\pi_\theta(A_t \mid S_t)}{\pi_{\theta_{\text{old}}}(A_t \mid S_t)}, \tag{4.11}$$

the policy component of the PPO objective is

$$J^{\text{clip}}(\theta) = \mathbb{E}\left[\min\left\{\widehat{\rho}_t(\theta)\widehat{A}_t, \text{clip}\left(\widehat{\rho}_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\widehat{A}_t\right\}\right]. \tag{4.12}$$

The baseline PPO agent is trained directly on the original cashflow rewards in Equation (4.29). It therefore has to learn the relationship between purchases and later sales through its value estimates and advantage estimates.

The idea behind the two algorithms that we will present in this section is to find a reward signal that improves upon the original cashflow.

Remark 4.2.1. *Even though the results from Chapter 3 ensure that the optimal policy is preserved through reward redistribution, in practice, when predictors are approximated, this can break. This is especially relevant for RUDDER, since the prediction task is significantly more complicated.*

4.2.1 RUDDER

RUDDER is the algorithm based on the theory developed in Section 3.4.

We first generate an offline data set of trajectories. For each episode i, we record:

$$\tau^{(i)} = \left(S_0^{(i)}, A_0^{(i)}, R_1^{(i)}, \ldots, S_T^{(i)}, A_T^{(i)}, R_{T+1}^{(i)}\right) \tag{4.13}$$

and its realized return

$$G_0^{(i)} = \sum_{t=0}^{T} \gamma^t R_{t+1}^{(i)}. \tag{4.14}$$

We encode each state and action as a feature vector Z_t. The sequence presented to the return-prediction model is therefore

$$Z_{0:t} = (Z_0, \ldots, Z_t). \tag{4.15}$$

We train a long short-term memory (LSTM) with parameters ϕ to predict the total episode return from each trajectory prefix. Its prediction after observing the prefix up to time t is denoted by

$$\widehat{G}_{t+1}^{\phi} = f_\phi(Z_{0:t}). \tag{4.16}$$

In particular,

$$\widehat{G}_T^{\phi}$$

is the prediction formed after observing the complete state-action sequence.

The final prediction is especially important because it uses all the information in the trajectory. We therefore train the network using a loss that assigns a separate weight to the final prediction:

$$\mathcal{L}_{\text{RUDDER}}(\phi) = \lambda_{\text{final}} \frac{1}{N} \sum_{i=1}^{N} \left(\widehat{G}_T^{\phi,(i)} - G_0^{(i)} \right)^2 + \lambda_{\text{prefix}} \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(\widehat{G}_{t+1}^{\phi,(i)} - G_0^{(i)} \right)^2. \tag{4.17}$$

The first term measures the mean squared error of the full-trajectory prediction. The second term is the mean squared error of the intermediate prefix predictions, averaged across time. Choosing

$$\lambda_{\text{final}} > \lambda_{\text{prefix}}$$

places greater emphasis on accurately predicting the return from the complete sequence.

After training the return-prediction model, we redistribute the return using differences between consecutive prefix predictions, following the insight given by Theorem 3.4.2. If we consider the fully delayed rewards environment where all reward is given at the end, then

$$\widehat{G}_t^{\phi} \approx q^\pi(s_t, a_t)$$

The redistributed reward is then defined by taking differences between consecutive predictions

$$\widetilde{R}_t = \widehat{G}_t^{\phi} - \widehat{G}_{t-1}^{\phi}. \tag{4.18}$$

A large positive change in the predicted return assigns positive credit to the newly observed state-action pair, whereas a large negative change assigns negative credit.

We define an initial baseline prediction

$$\widehat{G}_0^\phi = 0.$$

The redistributed reward is

$$\widetilde{R}_{t+1} = \widehat{G}_{t+1}^\phi - \widehat{G}_t^\phi.$$

Therefore,

$$\sum_{t=0}^{T-1} \widetilde{R}_{t+1} = \widehat{G}_T^\phi - \widehat{G}_0^\phi = \widehat{G}_T^\phi.$$

An alternative is to add a terminal correction

$$\widetilde{R}_{T+1}^{\text{corr}} = \widetilde{R}_{T+1} + G - \widehat{G}_T^\phi, \tag{4.19}$$

which forces the redistributed rewards to reproduce the realized return exactly, up to the chosen initial baseline. However, when the return predictor is imperfect, this correction can place a large residual reward back at the end of the episode.

If the LSTM is capable of accurately predicting the expected final returns, then theorem Theorem 3.4.2 ensures that this reward redistribution is optimal, in the sense of Definition 3.4.1.

We then train a PPO model using $\widetilde{R}_t$ in place of the original cashflow reward R_t. The underlying state transitions and actions remain unchanged. This is the offline RUDDER algorithm. When this is not sufficient, in particular when the LSTM is unable to generalize from the dataset of training paths to the paths encountered during the PPO, we also have the option to train the LSTM online. In other words, we can collect the paths encountered during the PPO and perform gradient updates on them while the PPO is running.

The greatest challenge of this algorithm is the training of the LSTM. The model has to learn to predict the final expected return, given only a partial episode sequence; when the final returns have high variance, this can be a difficult task. Gas storage environments can have a high variance, since rollouts where gas is bought and never sold can lead to very negative rewards. Given this complexity, another significant factor to consider is the computational time. The offline training of the LSTM alone can take far more time than a PPO to run, and with online training it can reach prohibitive running times.

In the original paper [Arj+19], RUDDER in used only for discrete action spaces, but it extends naturally to continuous action spaces. However, in this setting, sampling trajectories and the prediction task of the LSTM become more complex.

In Algorithm 1 the key steps of the algorithm are summarized.

Algorithm 1 RUDDER

Require: Number of trajectories N, horizon T
Ensure: Trained policy π_θ

Generate an offline data set of N trajectories

$$\tau^{(i)} = \left(S_0^{(i)}, A_0^{(i)}, R_1^{(i)}, \ldots, S_T^{(i)}, A_T^{(i)}, R_{T+1}^{(i)}\right)$$

Compute the realized return of each trajectory

$$G^{(i)} = \sum_{t=0}^{T} \gamma^t R_{t+1}^{(i)}$$

Encode each state–action pair as a feature vector $Z_t^{(i)}$
Train the LSTM f_ϕ to predict $G^{(i)}$ from trajectory prefixes $Z_{0:t}^{(i)}$
for each trajectory used to train PPO **do**
    for $t = 0, \ldots, T-1$ **do**
        Compute the redistributed reward

$$\widetilde{R}_t = \widehat{G}_t^{\phi} - \widehat{G}_{t-1}^{\phi}$$

    end for
end for
Train PPO using $\widetilde{R}_t$ instead of R_t
if online LSTM training is used **then**
    Add the PPO trajectories to the LSTM training data
    Update the LSTM while PPO is training
end if
return π_θ

4.2.2 Integrated gradients for reward redistribution

The second algorithm that we will apply to gas storage environments uses integrated gradients to redistribute the reward signal, following Section 3.5.2. As in RUDDER, we generate an offline dataset of trajectories using one or more policies

$$\tau^{(i)} = \left(S_0^{(i)}, A_0^{(i)}, R_1^{(i)}, \ldots, S_T^{(i)}, A_T^{(i)}, R_{T+1}^{(i)}\right) \tag{4.20}$$

and its realized return

$$G_0^{(i)} = \sum_{t=0}^{T} \gamma^t R_{t+1}^{(i)}. \tag{4.21}$$

Then, we train a multilayer perceptron (MLP) f_θ to predict the final return, given the whole trajectory, so

$$\widehat{G}_T^{\phi} = f_\theta(s_0, a_0, \ldots, s_T, a_T).$$

As a training loss, we use a simple mean squared loss function. This task is much simpler than its counterpart in RUDDER, since the network only needs to predict the return from the whole episode, and not from partial episode sequences as well. This is why we can substitute the LSTM for a simpler MLP.

To redistribute the reward of a trajectory τ, we select

$$\widetilde{R}_{t+1} = \mathrm{IG}_{a_t}(\tau, \tau'; f_\theta),$$

where τ' is a baseline trajectory. The simplest choice of a baseline is to keep the same states as τ and set the action to zero. As discussed in Section 3.5.2, the interpolation can be off-manifold, but the mlp may still be able to find the correct dependence of f_θ on the actions. More advanced baselines tailored to the specifics of gas storage environments will be introduced in later sections.

As we have seen in Proposition 3.5.3, choosing only the integrated gradient with respect to the action, without considering the one with respect to states, can break completeness. However, with this choice of baseline where the state components of τ' are the same as those of τ, we get

$$\mathrm{IG}_{s_t}(\tau, \tau'; f_\theta) = 0.$$

Hence, completeness ensures that

$$\sum_{t=0}^{T} \widetilde{R}_{t+1} = \widehat{G}_T^{\phi} - f_\theta(\tau'). \tag{4.22}$$

Therefore integrated gradients generates a reward redistribution, and the optimal policies are preserved, in the ideal scenario where f_θ perfectly predicts the total return.

After training the mlp model, we use the redistributed reward obtained through integrated gradients as a replacement for the reward signal in a PPO model.

Since derivatives are needed, integrated gradients as a redistribution method is only suitable for continuous action spaces.

In Algorithm 2 the key steps of the algorithm are summarized.

Algorithm 2 Integrated Gradients for Reward Redistribution

Require: Number of trajectories N, horizon T
Ensure: Trained policy π_θ

Generate an offline data set of N trajectories

$$\tau^{(i)} = \left(S_0^{(i)}, A_0^{(i)}, R_1^{(i)}, \ldots, S_T^{(i)}, A_T^{(i)}, R_{T+1}^{(i)}\right)$$

Compute the realized return of each trajectory

$$G^{(i)} = \sum_{t=0}^{T} \gamma^t R_{t+1}^{(i)}$$

Train the MLP f_θ to predict $G^{(i)}$ from the complete trajectory $\tau^{(i)}$
for each trajectory τ used to train PPO **do**
    Construct a baseline trajectory τ' with the same states as τ and zero actions
    for $t = 0, \ldots, T-1$ **do**
        Compute the redistributed reward

$$\widetilde{R}_{t+1} = \mathrm{IG}_{a_t}(\tau, \tau'; f_\theta)$$

    end for
end for
Train PPO using $\widetilde{R}_t$ instead of R_t
return π_θ

4.2.3 A discharge trick for gas storage environments

The greatest discriminant for the success or failure of a policy in a gas storage environment is whether unsold gas is left at the end of the episode. When large quantities remain unsold, the episode will result in a significant loss. This phenomenon makes the delayed reward problem more severe: it is harder for the PPO model to learn that injecting can be profitable, when doing so can incur in seizable losses.

This can also impact the reward redistribution model. As final actions are much more important in determining the final return of the episode, compared to all the other ones, the reward is mostly redistributed to them, weakening the signal in the rest of the episode.

We can easily solve this problem, by constraining the policy to never keep more gas in the storage than it could sell by fully withdrawing in the remaining time steps of the episode, both during training and testing. For example, we want to ensure that

$$x_{T+1} \leq 0 \qquad \text{and } x_T \leq w_{\max},$$

where x_t indicates the storage level at time t in the episode, and $w_{\max}$ is the maximum

withdrawal amount. By induction we get

$$x_{t+1} \leq (T-t)w_{\max},$$

and since

$$x_{t+1} = x_t + a_t w_{\max},$$

the constraint on the action is

$$a_t \leq T - t - \frac{x_t}{w_{\max}} =: \widehat{a}_t. \tag{4.23}$$

Therefore, if an action $a_t > \widehat{a}_t$, then excess gas will inevitably remain unsold at the end of the episode. We can then use $\widehat{a}_t$ as the upper bound on the allowed actions. If an action exceeds $\widehat{a}_t$, the policy will simply choose $\widehat{a}_t$. When applying this during training, we make sure to disable gradient updates for the steps where $\widehat{a}_t$ is chosen instead of the network action. Because of the term $T - t$ in Equation (4.23), this correction will only matter in the last steps of the episode.

4.3 A first environment: discrete-action mean-reverting Markov chain

We first study a deliberately simple finite-state storage environment. The purpose of this experiment is to isolate the temporal credit-assignment problem arising from storage decisions while retaining an environment for which the transition mechanism and the optimal policy can be analysed explicitly. We compare a standard PPO agent with an agent trained using rewards redistributed by a RUDDER-style return-prediction model.

The purpose of this environment is to demonstrate that RUDDER is not well suited to gas storage environments. As shown below, the method does not achieve satisfactory results even in this simple setting, which suggests that it is unlikely to perform better in more complex environments.

Integrated gradients cannot be applied to this environment because the method requires taking derivatives with respect to the actions and is therefore only applicable to continuous action spaces.

4.3.1 Environment

Let $p_{\max} \in \mathbb{N}$ denote the maximum price level, let $C \in \mathbb{N}$ denote the storage capacity, and let $T \in \mathbb{N}$ denote the episode length. The state space is

$$\mathcal{S} = \{0, \ldots, C\} \times \{0, \ldots, p_{\max}\} \times \{0, \ldots, T\} \tag{4.24}$$

At time t, the state is written as

$$S_t = (X_t, P_t, t), \tag{4.25}$$

where P_t is the discrete market price and X_t is the current inventory.

The action space is

$$\mathcal{A} = \{-1, 0, 1\}, \tag{4.26}$$

where, keeping the same framework of the previous section, action 0 means that we do nothing, action -1 means that we sell one unit of gas, and action 1 means that we buy one unit of gas. The feasible action set depends on the inventory level. It is given by

$$\mathcal{A}(x) = \begin{cases} \{0, 1\}, & x = 0, \\ \{-1, 0, 1\}, & 0 < x < C, \\ \{-1, 0\}, & x = C. \end{cases} \tag{4.27}$$

Thus, we cannot sell when the storage is empty and we cannot buy when the storage is full.

For a feasible action, the inventory transition is deterministic:

$$X_{t+1} = X_t + A_t \tag{4.28}$$

If an infeasible action is selected, the inventory remains unchanged. In the implementation, such an action receives zero reward and is recorded as invalid.

The one-step reward is the cashflow generated at the current price, when the action is feasible:

$$R_t = -P_t A_t \tag{4.29}$$

Buying therefore produces an immediate negative reward, whereas selling produces an immediate positive reward. Doing nothing produces no cashflow.

In this first experiment, we do not impose a terminal liquidation condition or a terminal inventory penalty. It is important to notice, however, that keeping gas until the episode is over is highly suboptimal, since its value is lost when it is not sold. The return of an episode is therefore

$$G_0 = \sum_{t=0}^{T} \gamma^t R_{t+1}. \tag{4.30}$$

In the numerical experiments, we will take $\gamma = 1$.

The price follows a discrete mean-reverting Markov chain on

$$\{0, \dots, p_{\max}\}.$$

Let

$$\mu = \frac{p_{\max}}{2}, \qquad \beta = \frac{2}{p_{\max}} \tag{4.31}$$

For a price level $p \in \{0, \dots, p_{\max}\}$, we define the probabilities of an upward and a downward movement by

$$q_+(p) = \frac{1}{2}\left(1 - \beta(p - \mu)\right), \tag{4.32}$$

$$q_-(p) = \frac{1}{2}\left(1 + \beta(p - \mu)\right). \tag{4.33}$$

The price transition is then

$$P_{t+1} = \begin{cases} P_t + 1, & \text{with probability } q_+(P_t), \\ P_t - 1, & \text{with probability } q_-(P_t). \end{cases} \tag{4.34}$$

Since $q_+(p) + q_-(p) = 1$, this defines a nearest-neighbour random walk. Note that

$$q_+(p_{\max}) = \frac{1}{2}(1 - \beta\,\frac{p_{\max}}{2}) = 0, \quad q_-(0) = \frac{1}{2}\Big(1 + \beta\Big(-\frac{p_{\max}}{2}\Big)\Big) = 0,$$

so the price cannot drop below zero or rise above $p_{\max}$. The transition probabilities generate mean reversion towards μ. Indeed, for $p > \mu$, the probability of a downward movement is larger than the probability of an upward movement. Conversely, for $p < \mu$, an upward movement is more likely.

The initial inventory is fixed at

$$X_0 = 0, \tag{4.35}$$

whereas the initial price is sampled uniformly:

$$P_0 \sim \mathrm{Unif}\{0, \dots, p_{\max}\}. \tag{4.36}$$

Although the environment supplies an immediate cashflow after each action, the economic value of a purchase cannot be determined from the purchase reward alone. A profitable storage cycle consists of buying at some time t and selling at a later time $u > t$. Ignoring discounting, the combined reward of this cycle is

$$-P_t + P_u = P_u - P_t. \tag{4.37}$$

The buy action therefore receives a negative immediate reward even when it is essential for obtaining a positive future profit.

This creates a temporal credit-assignment problem. The learning algorithm must infer that an action with a negative immediate reward may be desirable because it enables a later sale. Moreover, the value of buying depends on the future evolution of the price, the remaining horizon, and the available storage capacity. We use this environment to investigate whether reward redistribution can make this relationship easier for a policy-gradient algorithm to learn.

In all experiments, the environment has the following parameters

$$C = 20, \qquad p_{\max} = 10, \qquad T = 200.$$

4.3.2 Results

We evaluate all learned policies using the original economic rewards in Equation (4.29), not the redistributed rewards. This distinction is essential: redistributed rewards are used only as a training signal, whereas performance is measured by the actual storage cashflows.

We compare the following policies:

(i) a random policy;

(ii) PPO trained on the original rewards;

(iii) PPO trained on the redistributed rewards;

(iv) an optimal dynamic-programming policy, when used as a finite-state benchmark.

For each method, we estimate the expected original return over a common collection of evaluation episodes. Using common random seeds reduces the variability of pairwise comparisons because the policies are evaluated under comparable price paths.

Since the state and action spaces are finite and the transition probabilities are known, we can compute the optimal value function by backward induction. With terminal value

$$V_T^*(p, x) = 0, \tag{4.38}$$

the recursion is

$$V_t^*(p, x) = \max_{a \in \mathcal{A}(x)} \left\{ r(p, x, a) + \gamma \sum_{p'=0}^{p_{\max}} P(p, p') V_{t+1}^*(p', x'(x, a)) \right\}, \tag{4.39}$$

where $x'(x, a)$ is the inventory obtained after applying action a. The resulting dynamic-programming policy provides a useful reference for determining whether poor performance is caused by the difficulty of the control problem or by the learning procedure.

Given the simplicity of the environment, a standard PPO converges to the optimal policy, given enough time. Our aim is to determine whether RUDDER-style redistribution can work in this environment, in other words, if the PPO using redistributed reward converges. Moreover, if it does, we are interested in whether the convergence speed improves.

This experiment therefore separates two questions. First, can the recurrent model infer the contribution of individual decisions to the total storage return? Second, conditional on obtaining a sufficiently accurate return decomposition, does PPO learn more effectively from the redistributed reward than from the original cashflow sequence?

The benchmark PPO algorithm was configured with the parameters in Table 4.1 and Table 4.2. We selected these parameter values based on preliminary experiments.

Table 4.1: Actor–critic network architecture used by the PPO agent. The actor and critic have identical hidden structures but independently learned parameters.

Property	**Actor**	**Critic**	**Details**
Function	Policy	Value estimator	Produces the action-distribution parameters and the scalar state-value estimate, respectively.
Input dimension	d_{obs}	d_{obs}	Current observation vector.
Hidden layers	$64, 64, 64$	$64, 64, 64$	Three fully connected layers.
Activation	SELU	SELU	Applied after each linear transformation.
Output dimension	d_{act}	1	One policy output per action dimension and one scalar value estimate.
Weight initialization	$\mathcal{N}(0, 0.1^2)$	$\mathcal{N}(0, 0.1^2)$	Biases are initialized to zero.
Initial policy std.	0.8	–	Initial exploration scale for continuous-action policy.
Parameter count	$64d_{\text{obs}} + 65d_{\text{act}}+8{,}384$	$64d_{\text{obs}} + 8{,}449$	Excludes separately stored or learned policy standard-deviation parameters.
	Combined parameter count		$\mathbf{128}\boldsymbol{d_{\text{obs}}} \mathbf{+ 65}\boldsymbol{d_{\text{act}}} \mathbf{+ 16{,}833}$

d_{obs} and d_{act} denote the observation and action dimensions.

Table 4.2: PPO optimization, sampling, and objective parameters.

Parameter	Value	Description
Sampling and optimization		
Optimizer	Adam	Optimizes the actor and critic parameters.
Number of updates	200	Number of data-collection and optimization cycles.
Episodes per update	50	Complete episodes collected before each update.
Batch size	200	Number of samples in each optimization batch.
Maximum gradient norm	0.7	Upper bound used for gradient-norm clipping.
Returns and advantages		
Discount factor γ	1.0	Corresponds to undiscounted episodic returns.
GAE parameter λ	0.98	Controls the bias–variance trade-off of generalized advantage estimation.
Advantage normalization	Enabled	Normalizes the estimated advantages before optimization.
PPO objective		
Clipping range ϵ	0.08	Restricts the policy probability ratio to [0.92, 1.08].
Value-loss coefficient c_v	0.7	Weight assigned to the critic loss.
Entropy coefficient c_{ent}	0.001	Encourages policy exploration.

The LSTM architecture is described in Table 4.3.

As a first experiment, we choose the most natural way to collect trajectories to train the LSTM: random trajectories. Formally, at all times t we sample the action independently from the state, with the policy

$$a_t \sim \text{Unif}(\{-1, 0, 1\}).$$

Not only is this simplest choice, but it also does not require any additional information on the environment or on the optimal policy. We use 10 000 training paths.

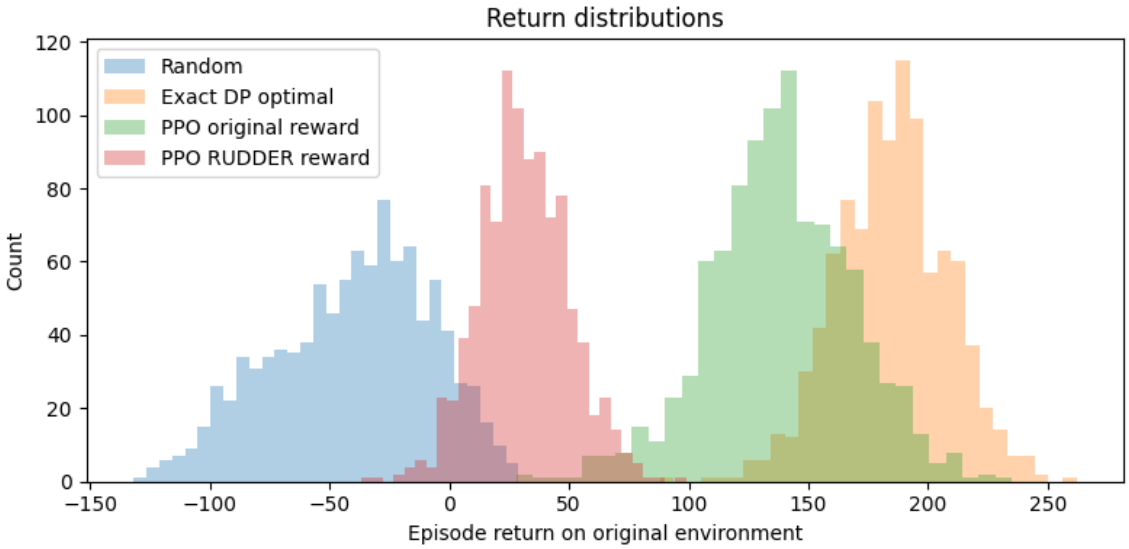


Figure 4.1: Distribution of 1000 episode returns in the original environment for a random policy, the exact dynamic-programming optimal policy, and PPO agents trained using the original and RUDDER-shaped rewards. Histograms are computed over evaluation episodes; higher returns indicate better performance

In Figure 4.1 we see that the PPO with redistributed rewards, after training for the same time as the standard PPO, fails to achieve an acceptable result. To understand why, we can look at Figure 4.2. The reward redistribution collapses to zero after the first 4 steps. Even in those steps, where the policy buys at low price, a profitable action, the reward is negative, although this is mostly due to the dataset mean being negative.

In this environment, therefore, offline training of the LSTM with random trajectories is not capable of obtaining a successful reward redistribution. Furthermore, using a reward redistribution obtained this way as the reward signal for a PPO destroys learning. Increasing the size of the training dataset did not significantly improve results.

In the second experiment, we improve the quality of the training dataset. Instead of collecting paths only according to a random policy, we also use an expert policy: the optimal policy obtained with dynamic programming. Like before, we build a dataset of

Table 4.3: Architecture and training configuration of the offline LSTM return-regression model. At each time step, the observation and action vectors are concatenated and processed by a unidirectional LSTM.

Component	Operation	Input size	Output size	Trainable parameters
Input	Concatenate observation and action at each time step	$T \times (d_{\text{obs}} + d_{\text{act}})$	$T \times D$	0
LSTM layer	Single-layer, unidirectional LSTM	$T \times D$	$T \times H$	$4H(D + H + 2)$
Prediction head	Time-distributed fully connected layer	$T \times H$	$T \times 1$	$H + 1$
Output	Squeeze final dimension	$T \times 1$	T	0
			Total trainable parameters	$\boldsymbol{4H(D + H + 2) + H + 1}$

Configuration	Value
Per-step input dimension	$D = d_{\text{obs}} + d_{\text{act}}$
Sequence length	T
Hidden-state dimension	$H = 64$
Number of recurrent layers	1
Recurrent direction	Unidirectional
Batch layout	Batch-first, with input shape (B, T, D)
Model output	One scalar return estimate per time step, with shape (B, T)
Loss function	As in Equation (4.17)
$\lambda_{final}, \lambda_{prefix}$	1, 0.5
Optimizer	Adam
Learning rate	3×10^{-3}
Number of epochs	10
Batch size	64
Gradient clipping	Maximum gradient norm of 1.0
Data shuffling	Enabled

Notation: B denotes the batch size, T the sequence length, d_{obs} the observation dimension, d_{act} the action dimension, $D = d_{\text{obs}} + d_{\text{act}}$, and $H = 64$ the LSTM hidden-state dimension.

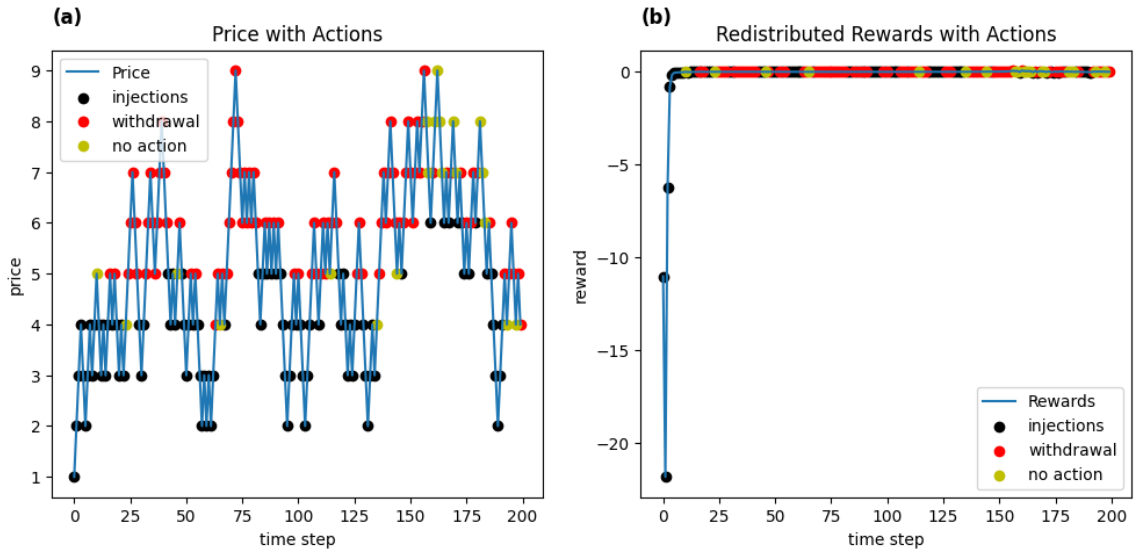


Figure 4.2: Rollout generated from the optimal policy computed with dynamic programming. The dots along the trajectories indicate the action at a time step. Black dots represent injections (+1), red dots withdrawals (-1) and yellow dots no action taken (0). (a) Price curve of the rollout with actions selected by the optimal policy. (b) Reward redistribution of the rollout, obtained with an LSTM trained on random trajectories, with actions selected from the optimal policy.

10 000 paths. We collect half of these with a random policy, and the other half with a noisy expert policy π_ε, given by

$$\pi_\varepsilon(s_t) = \begin{cases} \text{Unif}(\{-1, 0, 1\}), & \text{with probability } \varepsilon \\ \pi_{\text{opt}}(s_t), & \text{with probability } 1 - \varepsilon \end{cases}$$

As we can see in Figure 4.3, while the distribution of returns has improved, it is still far from a standard PPO. Looking at the reward redistribution in Figure 4.4, we observe progress compared to the fully random dataset: the redistributed reward signal does not completely flatten to 0, but exhibits some variation. However, it is still far from a desirable result. For example, the price curve presents clusters of low prices around time steps 65 and 100, where the optimal policy injects. These are key moments in the episode where the predicted final reward should have increased, therefore, we expect these actions to receive positive reward. Since this is not the case, we can conclude that the model is not capable of recognising a significant change in the expected final return. This explains the failure in the convergence of the PPO with this redistributed reward signal.

Using an online RUDDER algorithm leads to a better outcome, as is shown in Figure 4.5. Here, the PPO model with reward redistribution as the reward signal is able to converge. In Figure 4.6 we can observe the shape of the reward redistribution. In the cluster of actions around timestep 50, where the policy bought at low price, the reward redistribution model assigns a positive reward. Therefore, the model understood that those actions, which in

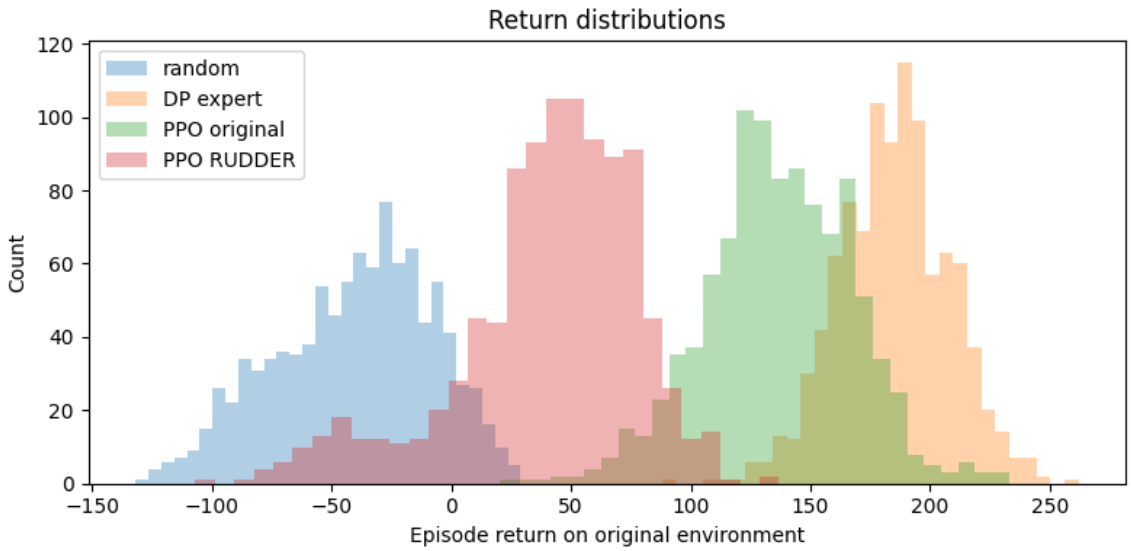


Figure 4.3: Distribution of 1000 episode returns in the original environment for a random policy, the exact dynamic-programming optimal policy, and PPO agents trained using the original and RUDDER-shaped rewards. Histograms are computed over evaluation episodes; higher returns indicate better performance

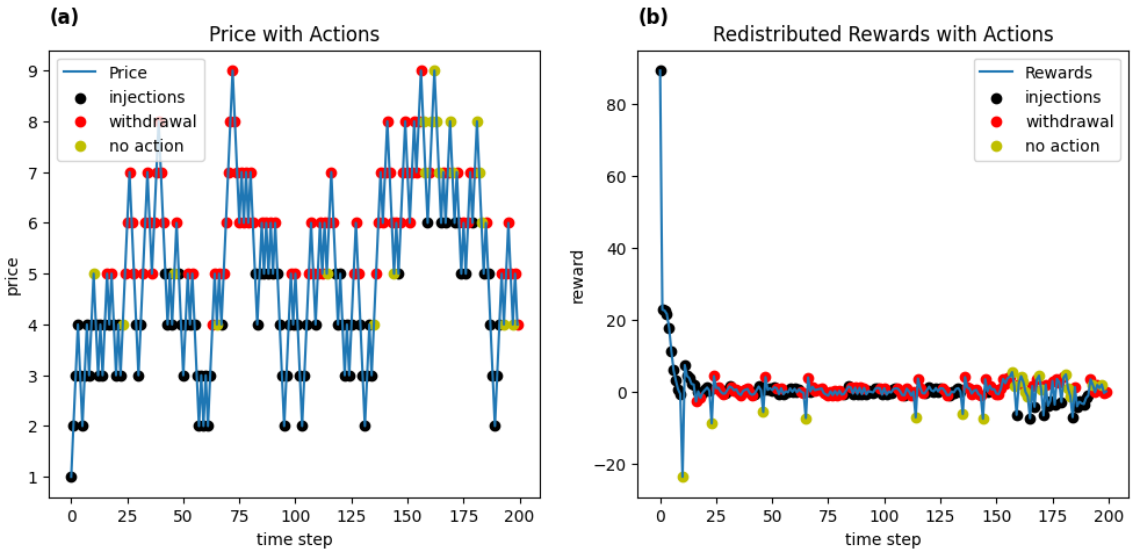


Figure 4.4: Rollout generated from the optimal policy computed with dynamic programming. The dots along the trajectories indicate the action at a time step. Black dots represent injections (+1), red dots withdrawals (-1) and yellow dots no action taken (0). (a) Price curve of the rollout with actions selected by the optimal policy. (b) Reward redistribution of the rollout, obtained with an LSTM trained on random and π_ε trajectories, with actions selected from the optimal policy.

the original setting received a negative reward, were responsible for high rewards later on, when the gas was sold.

Training the LSTM online is not ideal for two reasons. First, it is conceptually less compelling: rather than learning the general principle that injecting gas when prices are low is profitable, the model can identify this relationship only by fitting closely to the limited set of trajectories generated at a particular PPO iteration. As a result, the learned relationship is local to the current policy distribution rather than global. Second, online training is computationally very expensive. Although the convergence profile shown in Figure 4.5 appears compelling, the computational overhead of training the LSTM online is substantial: the PPO-RUDDER model requires more wall-clock time to reach 10 000 training steps than the standard PPO model requires to converge after 100 000 steps.

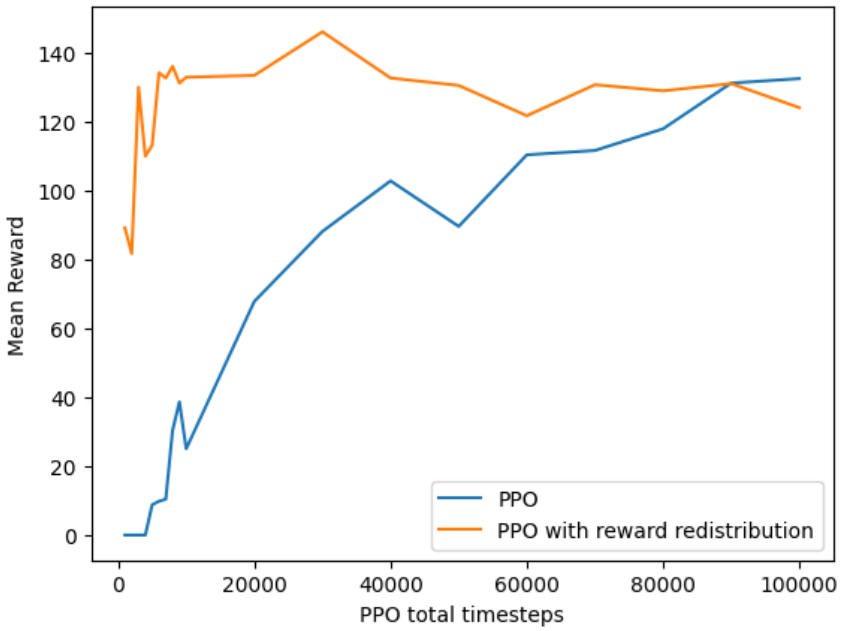
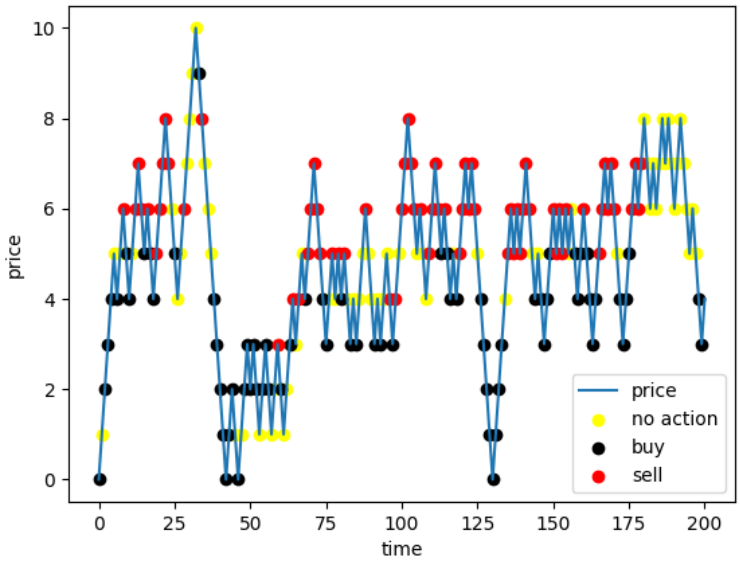


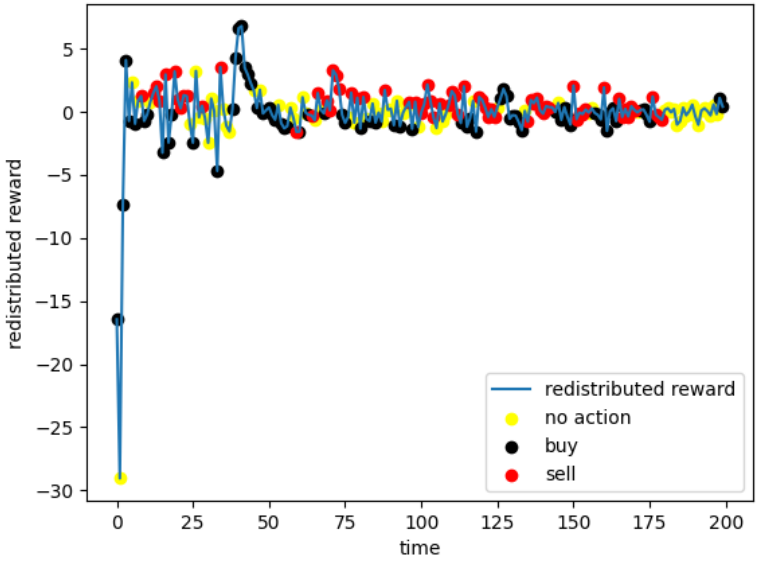
Figure 4.5: Convergence plots of a standard PPO algorithm and a PPO algorithm with reward redistribution obtained using RUDDER trained online on PPO paths. The models were tested each 1 000 gradients steps up to 10 000, and then each 10 000 timesteps. Testing consisted in an average over 100 seeds.

4.4 Continuous action: Gas Storage Deterministic

We now introduce a significant complication compared to the previous environment: continuous actions, specifically in the interval $[-1, 1]$. Since they are a significant challenge to reward redistribution, we consider an environment where the price curve is deterministic. If a reward redistribution method struggles in this simplified deterministic environment, this is strong evidence that it may be difficult to apply in more complex stochastic settings.



(a) Price curve of the rollout.



(b) Reward redistribution of the rollout, obtained with an lstm trained online on PPO paths.

Figure 4.6: Rollout generated from the PPO RUDDER policy at the end of training. The dots along the trajectories indicate the action at a time step. Black dots represent injections (+1), red dots withdrawals (-1) and yellow dots no action taken (0).

In all experiment will choose a time horizon $T = 365$, to simulate a trading year. The price is given by a deterministic sine function

$$P_t = \sin\left(2\pi\frac{t}{T}\right) + 2.0.$$

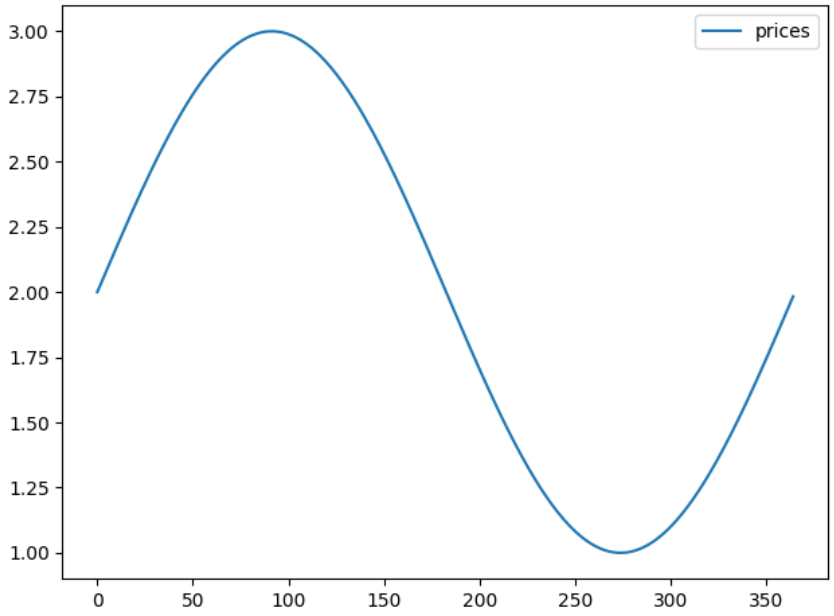


Figure 4.7: Price curve for Gas Storage Deterministic environment.

From Figure 4.7, the outline of optimal trading strategy is clear. The policy should buy at the beginning, then sell at the first peak around $t = 90$, followed by doing nothing until the price is low enough, then buy again, and sell at the end. Since, in the optimal policy, the reward for buying at low price is collected around 80 time steps later, we expect the environment to suffer from delayed rewards. The discharge trick introduced in Section 4.2.3 is promising in this environment, since the optimal policy will lead to gas being stored until the end, when it is sold.

In all experiments we will fix

$$i_{\max} = w_{\max} = 0.1, \qquad C = 20.$$

For these values, the optimal policy has a return of

$$G_0^{\text{opt}} = 5.8093$$

For the underlying PPO model, we use the parameters in Table 4.1 and Table 4.2.

4.4.1 RUDDER

The RUDDER model can be applied with the same exact setup as in Section 4.3, but continuous actions make its task much harder.

Firstly, we construct the training datasets. As in the discrete-action setting, the natural choice is a random dataset, in which actions are sampled uniformly. That is, at every time step t,

$$a_t \sim \text{Unif}([-1, 1]).$$

The second dataset, which we refer to as the expert dataset, contains a mixture of random and noisy-expert trajectories. We first compute an optimal policy π_{opt} using dynamic programming. For each noise level $\sigma \in \{0.1, 0.2, \dots, 0.9\}$, we then define the noisy-expert policy

$$\pi_\sigma(s_t) = \pi_{\text{opt}}(s_t) + \nu_t, \qquad \nu_t \sim \mathcal{N}(0, \sigma^2).$$

The expert dataset contains N trajectories and is divided into ten equally sized subsets. Specifically, $N/10$ trajectories are generated using the random policy, while, for each $\sigma \in \{0.1, 0.2, \dots, 0.9\}$, $N/10$ trajectories are generated using π_σ. Hence,

$$\mathcal{D}_{\text{expert}} = \mathcal{D}_{\text{rand}} \cup \bigcup_{\sigma \in \{0.1,\dots,0.9\}} \mathcal{D}_\sigma, \qquad |\mathcal{D}_{\text{rand}}| = |\mathcal{D}_\sigma| = \frac{N}{10},$$

so that

$$|\mathcal{D}_{\text{expert}}| = \frac{N}{10} + 9\frac{N}{10} = N.$$

In Figure 4.8 we can observe the reward redistribution obtained using the LSTM described in Table 4.3, trained offline on the random and expert datasets described above, with $N = 100\,000$, $1\,000\,000$ trajectories. It is immediately clear that training on random trajectories does not produce a useful signal: when the policy buys at low price, the reward is negative. Although this behaviour does not appear when the model is trained on the expert dataset, the shape of the redistributed reward is still not qualitatively satisfying. In the first batch of injecting action, while buying at the very start is better than buying later due to the lower price, the drop in the reward signal is too sudden. In the other clusters of actions it is even more transparent, as a signal is given only right at the beginning, when the action changes. In neither case, therefore, the LSTM is capable of finding a useful reward redistribution with offline training. This is not surprising, since it also failed in the simpler environment of Section 4.3 with a discrete action space.

As expected from the discussion above, when training the PPO model in Table 4.1 using this reward signal, the model completely fails to converge or produce any interesting result.

Continuous action spaces significantly complicate online training, as the model needs significantly more trajectories to understand the effect of the multitude of actions. Online

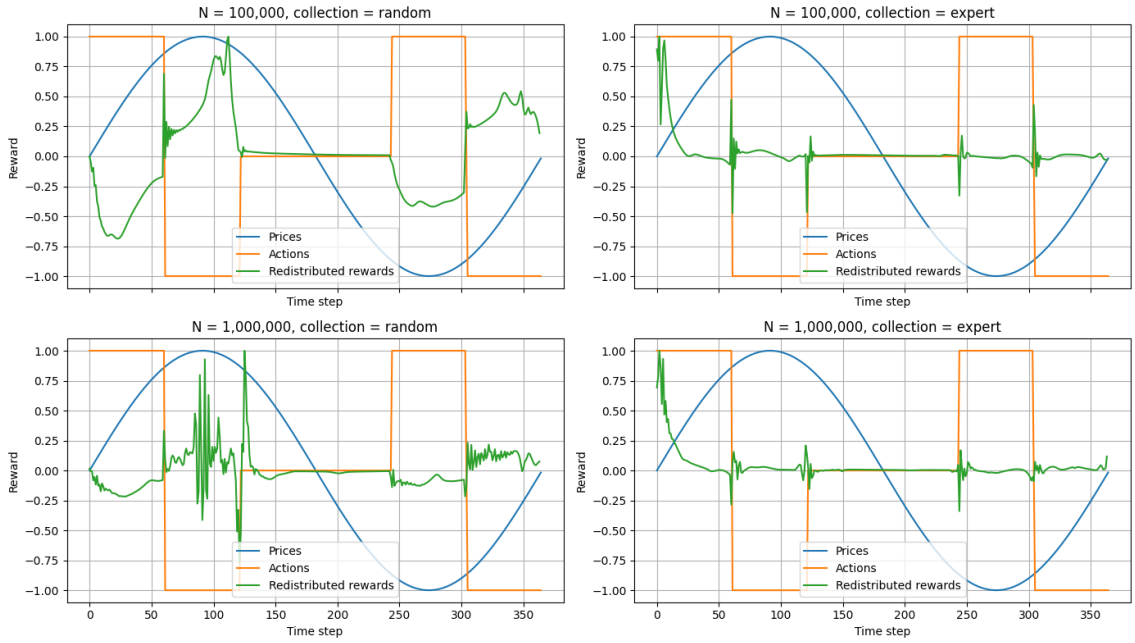


Figure 4.8: Offline RUDDER reward redistribution for an expert trajectory obtained through the dynamic programming optimal policy. The LSTM is trained on datasets with $N = 10\,000$, $100\,000$ trajectories, collected with random or expert policies. Actions and prices are shown as a reference. Prices are translated by -2 to ease visualization.

trajectories are also expensive to collect, since a full PPO iteration is needed. For this reason, running the same online RUDDER setup as for the discrete actions led to failure.

In [Arj+19, Section A4.2], the authors described adjustments made to the online RUDDER algorithm for difficult environments like Atari games. For brevity, we omit the details. We implemented all those modifications, but the model did not yield any noteworthy results, as it failed to converge,

These algorithms are computationally expensive because the LSTM is trained either on a large dataset in the offline setting or repeatedly over many iterations in the online setting. The training time was on the order of 1 hour for the dataset containing $100\,000$ paths and on the order of 10 hours for the dataset containing $1\,000\,000$ paths. The training time for a single online PPO RUDDER training run is on the order of 2 hours.

These results indicate that RUDDER may not be suitable for gas storage environments. Even in a simple environment with discrete actions, the LSTM was not able to find a global reward redistribution using offline training, and, to obtain convergence, we had to employ expensive online training. For continuous actions, even online training failed. More research is needed to understand the exact reasons. Based on Figure 4.8 and discussions in [Arj+19], we propose the following interpretation. The authors repeatedly emphasize that RUDDER is particularly effective in identifying abrupt changes in the action-value function $Q(s, a)$ associated with key actions. Because storage volumes and market prices evolve gradually,

the action-value function is expected to vary smoothly over time and across actions, rather than exhibiting abrupt jumps at a small number of decisive time steps. Consequently, there may be no distinct change point to which RUDDER can assign the delayed reward. Instead, an appropriate reward redistribution would have to allocate credit gradually across an entire sequence of mutually dependent actions. This mismatch between RUDDER's emphasis on identifying sharp changes in $Q(s, a)$ and the smooth structure of gas storage decisions may explain its poor performance in this setting. Figure 4.8, particularly the training on the expert dataset, shows how RUDDER attempts to find these jumps in the value function in between the change in actions.

4.4.2 Integrated Gradients

Since in this environment rewards are continuous, we can use integrated gradients to redistribute rewards. We use the datasets built in the previous section for training.

In Table 4.4 the architecture used for the reward redistribution MLP is described. We did not select specific learning rate, epochs, and training dataset, as we shall test different combinations of them.

4.4.2.1 Parameter tuning: dataset size and learning rate

Since in [Arj+19] the LSTM architecture for RUDDER is described in extensive detail, we used the authors' parameters for the model. However, our use of integrated gradients is not based on the literature, thus it is important that we carefully tune the MLP parameters.

The fully random dataset fails to converge and is therefore omitted from the discussion to avoid making the analysis unnecessarily extensive. We instead focus on the $\mathcal{D}_{\text{expert}}$ dataset.

Two important parameters to decide for the training of the neural network are learning rates and total number of training epochs. We study the performance on the network with four different values for the learning rate. As a training dataset we first choose the expert data set with $N_{train} = 1\,000\,000$ and then another smaller, independently generated expert dataset, with $N_{val} = 100\,000$. When one of the two datasets is used for training, the other is used for validation.

First, we study the behaviour of the larger dataset, with $1\,000\,000$ paths.

Figure 4.9 shows the evolution of the training and validation losses over 50 epochs for the four learning rates considered, while Table 4.5 summarizes the corresponding validation statistics. The training curves display a clear ordering with respect to the learning rate. The two smallest learning rates, 10^{-4} and 3×10^{-4}, produce the lowest training losses and

Table 4.4: Architecture and configuration of the offline return-regression MLP. The input consists of the flattened observation–action trajectory over T time steps.

Component	Operation	Input size	Output size	Trainable parameters
Input	Flatten and concatenate observations and actions	$T \times (d_{\text{obs}} + d_{\text{act}})$	D	0
Hidden layer 1	Fully connected + SELU	D	64	$64D + 64$
Hidden layer 2	Fully connected + SELU	64	64	$64^2 + 64 = 4{,}160$
Hidden layer 3	Fully connected + SELU	64	64	$64^2 + 64 = 4{,}160$
Output layer	Fully connected, linear output	64	1	$64 + 1 = 65$
		Total trainable parameters		$\mathbf{64D + 8{,}449}$

Configuration	Value
Input dimension	$D = T(d_{\text{obs}} + d_{\text{act}})$
Hidden-layer widths	[64, 64, 64]
Activation function	SELU after each hidden layer
Output	Scalar predicted trajectory return
Regression target	Sum of rewards over the trajectory, $G = \sum_{t=1}^{T} r_t$
Weight initialization	Orthogonal initialization
Loss function	Mean squared error (MSE)
Optimizer	Adam
Batch size	64
Gradient clipping	Maximum gradient norm of 1.0
Data-collection method	Expert policy with noise, using 10 levels

Notation: d_{obs} and d_{act} denote the dimensions of one observation and one action, respectively; T is the trajectory length; and $D = T(d_{\text{obs}} + d_{\text{act}})$ is the flattened network-input dimension.

continue to improve throughout the optimization. The learning rate 10^{-3} also decreases substantially, but it remains above the two smaller learning rates. By contrast, 3×10^{-3} reaches a much higher plateau, indicating that this step size is too large to obtain an accurate and stable solution.

Among the configurations considered, the learning rate 10^{-4} provides the best overall validation performance. It attains a minimum validation loss of 1.0169×10^{-5} at epoch 46, compared with 1.7958×10^{-5} for 3×10^{-4}, attained at the same epoch. Moreover, 10^{-4} also gives the smallest final validation loss, 1.2705×10^{-5}, and the smallest mean validation loss over the final five epochs, 2.0613×10^{-5}. Thus, unlike a comparison based only on an isolated minimum, all three validation criteria consistently favour the smallest learning rate.

This conclusion is also supported by the trajectories in Figure 4.9. The validation loss associated with 10^{-4} decreases rapidly during the initial epochs and subsequently remains mostly within the lowest-loss region, despite the fluctuations expected from the stochastic nature of the validation data. The curve for 3×10^{-4} also reaches small values, but it exhibits larger and more frequent upward fluctuations. Its final-five-epoch mean is approximately three times larger than that obtained with 10^{-4}. We therefore regard 10^{-4} as both the most accurate and the most reliable learning rate among those tested.

For 10^{-3}, the minimum validation loss, 2.2972×10^{-5}, is still relatively small and is attained at epoch 35. However, the validation curve is less stable than those corresponding to the two smaller learning rates. Its final validation loss increases to 1.3473×10^{-4}, while its mean over the last five epochs is 1.0982×10^{-4}. The training curve also remains systematically above those obtained with 10^{-4} and 3×10^{-4}. Hence, although this learning rate occasionally reaches a low validation loss, its overall performance is less consistent.

The learning rate 3×10^{-3} performs considerably worse. Its training loss decreases sharply during the first few epochs but then remains at a comparatively high level, of order 10^{-3}. The corresponding validation curve displays large oscillations and repeatedly reaches values of order 10^{-3} or higher. Its best validation loss is 1.3369×10^{-4}, attained at epoch 27, whereas its final validation loss is 1.3803×10^{-3}. The mean validation loss over the last five epochs is 1.0934×10^{-3}. These results indicate that the optimizer repeatedly overshoots the low-loss region and does not converge stably for this learning rate.

The training curves do not show evidence of classical overfitting for the smaller learning rates. In particular, the training loss continues to decrease, while the validation loss does not exhibit a sustained upward trend. Instead, the validation trajectories fluctuate around a generally decreasing lower envelope. Since the validation data consist of independently generated stochastic paths, part of this variability may be attributed to the sampling variation of the validation loss rather than to a systematic deterioration in generalization.

The best validation epoch for both 10^{-4} and 3×10^{-4} is epoch 46. Consequently, the present experiment does not provide strong evidence that substantially fewer than 50 epochs would always be sufficient. Nevertheless, Figure 4.9 shows that most of the reduction in

both training and validation loss occurs during the first 20–30 epochs, after which the improvements become smaller. For this reason, we will use 30 epochs during the PPO experiments.

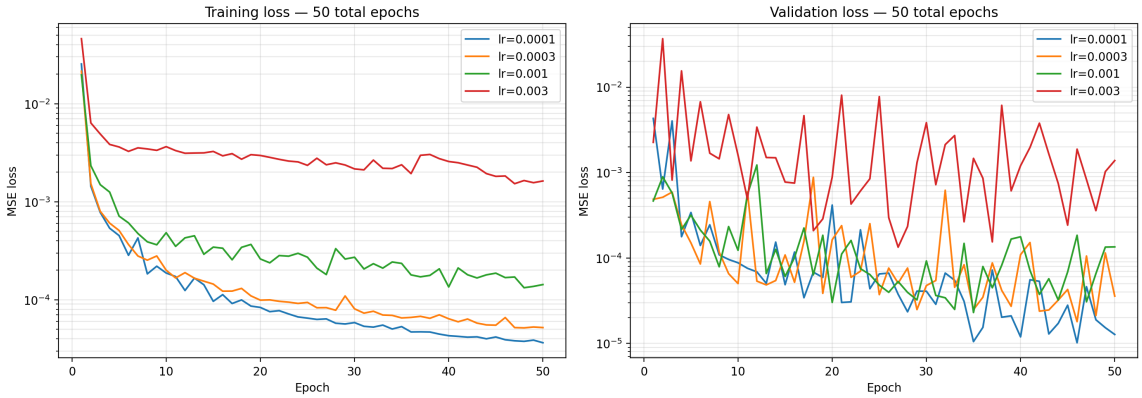


Figure 4.9: Training (left) and validation (right) mean squared error (MSE) over 50 epochs for four learning rates. The smaller learning rates, $\mathrm{lr} = 10^{-4}$ and $\mathrm{lr} = 3\times10^{-4}$, achieve the lowest and most stable losses, whereas $\mathrm{lr} = 10^{-3}$ converges to a higher training loss and exhibits greater validation variability. The largest learning rate, $\mathrm{lr} = 3 \times 10^{-3}$, results in unstable optimization and substantially higher losses. Both axes use a logarithmic scale for the loss.

Table 4.5: Validation performance for different learning rates. The best epoch is the epoch at which the minimum validation loss is attained. The final column reports the mean validation loss over the last five epochs.

Learning rate	Best validation loss	Best validation epoch	Final validation loss	Mean validation loss over final 5 epochs
1×10^{-4}	1.0169×10^{-5}	46	1.2705×10^{-5}	2.0613×10^{-5}
3×10^{-4}	1.7958×10^{-5}	46	3.5726×10^{-5}	5.8888×10^{-5}
1×10^{-3}	2.2972×10^{-5}	35	1.3473×10^{-4}	1.0982×10^{-4}
3×10^{-3}	1.3369×10^{-4}	27	1.3803×10^{-3}	1.0934×10^{-3}

We next repeat the learning-rate comparison for the smaller dataset with 100 000 trajectories.

The results are shown in Figure 4.10 and summarized in Table 4.6. As in the preceding experiment, the two smallest learning rates provide the best overall performance, whereas 3×10^{-3} leads to clearly inferior and less stable convergence.

The learning rate 3×10^{-4} attains the smallest validation loss, equal to 1.1806×10^{-4} at epoch 49. The learning rate 10^{-4} performs almost identically, with a minimum validation loss of 1.2752×10^{-4} at epoch 45. Their final validation losses and their averages over the final five epochs are also very close. Consequently, the present experiment does not reveal a practically significant difference between these two choices, although 3×10^{-4} is marginally better according to all three reported validation statistics.

The curves in Figure 4.10 nevertheless show that 10^{-4} produces the lowest training loss

near the end of the optimization, while 3×10^{-4} reaches a slightly smaller validation loss. This small discrepancy is consistent with ordinary sampling and optimization variability and does not indicate overfitting, since neither validation curve exhibits a sustained increase as the training loss decreases.

The learning rate 10^{-3} remains competitive at its best epoch, with a minimum validation loss of 1.5075×10^{-4}, but its validation curve is more irregular and its mean loss over the final five epochs is appreciably larger. The choice 3×10^{-3} again performs worst: its training loss remains at a much higher level, and its final validation loss exceeds 2×10^{-3}.

The best validation epochs for the two preferred learning rates occur late in training, at epochs 45 and 49. Since, as in the previous case, most of the loss reduction, especially the validation loss, has already occurred before epoch 30, we select this as the number of training epochs during the PPO experiments.

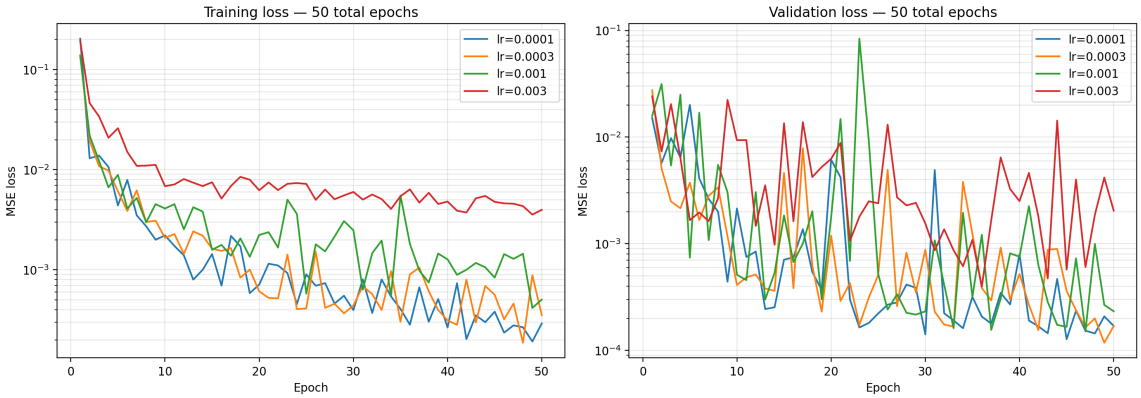


Figure 4.10: Training (left) and validation (right) mean squared error (MSE) over 50 epochs for four learning rates. The smaller learning rates, $\mathrm{lr} = 10^{-4}$ and $\mathrm{lr} = 3 \times 10^{-4}$, achieve the lowest and most stable losses, whereas $\mathrm{lr} = 10^{-3}$ converges to a higher training loss and exhibits greater validation variability. The largest learning rate, $\mathrm{lr} = 3 \times 10^{-3}$, results in unstable optimization and substantially higher losses. Both axes use a logarithmic scale for the loss.

Table 4.6: Validation performance for the network trained on 10^5 samples and validated on 10^6 samples. The best epoch is the epoch at which the minimum validation loss is attained. The final column reports the mean validation loss over the last five epochs.

Learning rate	Best validation loss	Best validation epoch	Final validation loss	Mean validation loss over final 5 epochs
1×10^{-4}	1.2752×10^{-4}	45	1.6974×10^{-4}	1.8227×10^{-4}
3×10^{-4}	1.1806×10^{-4}	49	1.6914×10^{-4}	1.7791×10^{-4}
1×10^{-3}	1.5075×10^{-4}	47	2.3293×10^{-4}	4.7567×10^{-4}
3×10^{-3}	3.9660×10^{-4}	36	2.0467×10^{-3}	2.5399×10^{-3}

4.4.2.2 Choosing the baseline

A key decision to make when using integrated gradients is the choice of baseline. As mentioned in Section 4.2.2, we choose baselines with the same states as the input trajectory, and only changes in the actions. The easiest choice is the baseline with all actions set to zero, the reinforcement learning equivalent of a black image in image recognition. However, as we shall see, this is not the best choice, since it assigns too much importance to the final selling actions. We will therefore study three types of baselines.

(i) Zero: all actions set to 0.

(ii) Sell last n: last n actions set to -0.9, all others to 0

(iii) Discharge Trick: If at any point the upper bound $\widehat{a}_t$ in Equation (4.23) is lower than 0, then set action t of the baseline to $\widehat{a}_t$, else set it to 0.

When training a PPO model with the discharge trick, the corresponding discharge-trick baseline assigns zero reward to any action a_t that violates Equation (4.23) and is consequently overwritten. In this case, the executed action coincides with the baseline action, $a_t = a'_t$, and therefore

$$\mathrm{IG}_{a_t}(\tau, \tau') = 0.$$

This prevents the final forced-sell actions from receiving credit. It does not adversely affect learning, because the actor and critic exclude these time steps from their gradient updates. However, this property also restricts the baseline to PPO models that themselves employ the discharge trick.

Sell last n is a simpler baseline, since it does not adapt to the trajectory and we can apply it to a standard PPO model, without discharge trick. Its idea is to balance the reward signal given to the last selling actions, without cancelling it completely.

In Figure 4.11 we compare the reward redistribution on an expert rollout obtained from the baselines described above, with values $n = 10, 50$ for the sell last n baseline. The MLP is trained on $\mathcal{D}_{\text{expert}}$ with $N = 100\,000$ trajectories and learning rate set to 10^{-4}. During the PPO, rewards are used to compute advantages, which are then normalized. Therefore, we are interested in the relative magnitude of reward received at a given step compared to the others. For this reason, we examine the normalized reward. In the Zero baseline, as anticipated in the previous paragraphs, the final actions in which residual gas is withdrawn dominate the reward redistribution. We observe that a baseline in which we sell only in the last 10 steps is not enough, since selling actions before those 10 steps still receive overwhelming credit. Extending selling in the baseline up to the last 50 actions, or using the discharge trick baseline, solves the problem. A qualitative analysis of these last

two redistributions in the bottom row shows that the redistributed reward behaves as we wished: the reward is high when injecting at low price or withdrawing at high price. This is a significant improvement compared to RUDDER, since we are able to obtain a reward redistribution which behaves as desired just with offline training.

A critique of the Sell last 50 baseline could be made when looking at Figure 4.11, as the magnitude of the reward in the last time steps seems to be too small. This could mean that -0.9 is too high. The problem with this argument is that the reward redistribution created by this baseline is highly dependent on the MLP parameters. As an example, in Figure 4.12 we can see the same experiment, but with the learning rate set to 10^{-3}. Here the Sell last 50 baseline gives very high rewards at the end of the episode. This exemplifies why artificially reducing the impact of the last action is a hard task. Conversely, working with the discharge trick gives a baseline that naturally solves the problem.

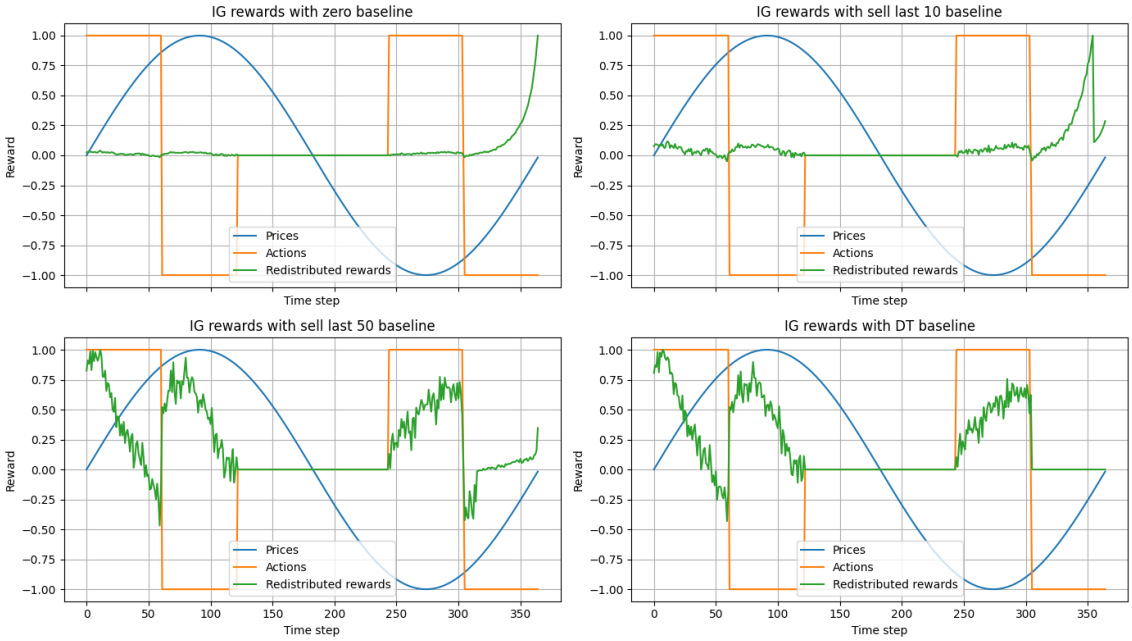


Figure 4.11: Comparison of the four reward redistributions obtained with integrated gradients and four different baselines: Zero, Sell last 10, Sell last 50, Discharge Trick. The rollout is generated using the optimal policy obtained from a dynamic program. Redistributed reward is normalized linearly, to set the highest value at 1. The MLP is trained with a 100 000 trajectories expert dataset and with 10^{-4} dataset.

4.4.2.3 Results

Now we present the experimental result from running PPO models with and without redistributing rewards. The PPO is configured with the parameters of Table 4.1 and Table 4.2. In all experiments we trained the reward redistribution MLP for 30 epochs. Each combination of dataset size and learning rate was tested on 10 random seeds. For each of these

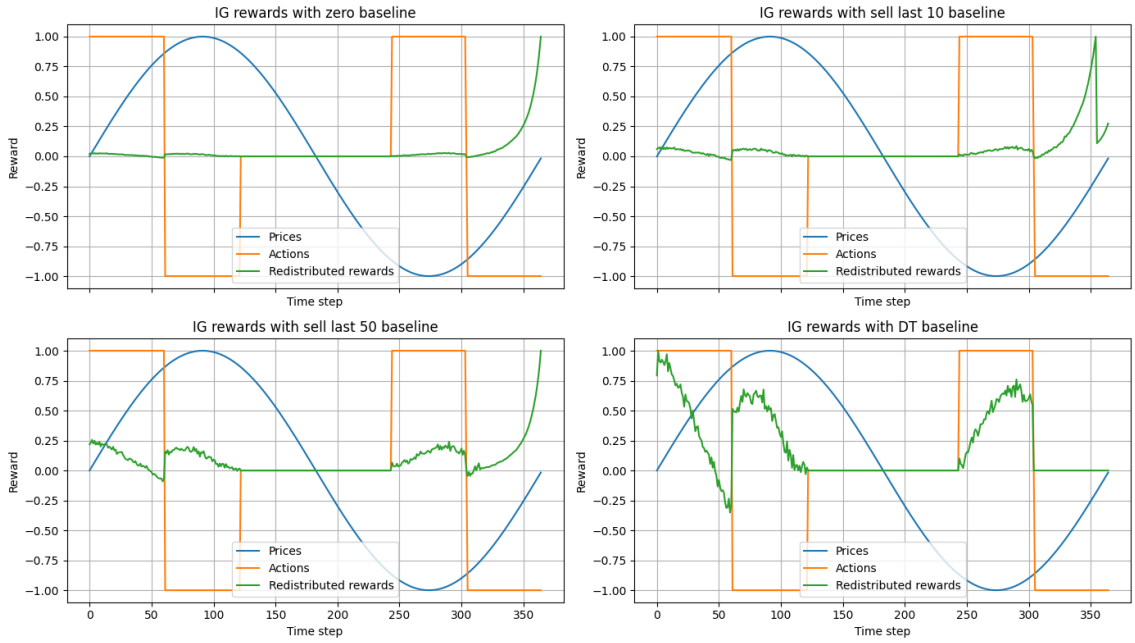


Figure 4.12: Comparison of the four reward redistributions obtained with integrated gradients and four different baselines: Zero, Sell last 10, Sell last 50, Discharge Trick. The rollout is generated using the optimal policy obtained from a dynamic program. Redistributed reward is normalized linearly, to set the highest value at 1. The MLP is trained with a 100 000 trajectories expert dataset and with 10^{-3} dataset.

seeds, both the MLP and the PPO model were trained from scratch. We discarded the learning rate 3×10^{-3} because of its poor performance in the return prediction task for both datasets.

Table 4.7 shows the results when no discharge trick is applied to the PPO model for expert datasets. We chose the Sell last 50 baseline, since, as Figure 4.11 showed, it is the best candidate when the Discharge Trick baseline is not available. We observe that no model with reward redistribution can reach a return greater than 5. Although this is an improvement over RUDDER, which struggled to get a positive return, it is still not a satisfactory result. The performance of the learning rate is in accordance with the findings of Figure 4.9 and Figure 4.10, with 1×10^{-4} and 3×10^{-4} showing more stable behaviour. It is noteworthy that the smaller 100 000 trajectory datasets outperform their larger counterparts.

Table 4.8 shows the results when the discharge trick is applied to the PPO model for expert datasets. We chose, naturally, the discharge trick baseline. These results are much more promising than the previous scenario, in which the discharge trick was not used. By looking at the best return column we see that the algorithm is able to reach high returns and converge for some seeds, except for the parameter configuration in the last row. Two concerns are the stability of the algorithm, since for many seeds it does not converge, as we can see by looking at the mean return, and the fact that, when it converges, it reaches values

Table 4.7: Results of PPO with IG reward redistribution across dataset sizes and learning rates. Datasets are $\mathcal{D}_{\text{expert}}$. No discharge trick is used and the IG baseline is Sell last 50. For each parameter configuration training is repeated across 10 random seeds. Results are ordered by mean return in descending order

Algorithm	Mean return	Best return	Worst return	Dataset size	Learning rate
PPO standard	5.594	5.690	5.071	–	–
PPO with IG	3.369	4.362	2.082	100,000	1×10^{-4}
PPO with IG	3.314	4.185	1.942	100,000	3×10^{-4}
PPO with IG	2.403	4.549	-0.517	100,000	1×10^{-3}
PPO with IG	2.323	4.239	0.193	1,000,000	3×10^{-4}
PPO with IG	2.215	3.468	0.193	1,000,000	1×10^{-4}
PPO with IG	1.920	3.921	0.056	1,000,000	1×10^{-3}

Table 4.8: Results of PPO with IG reward redistribution across dataset sizes and learning rates. Datasets are $\mathcal{D}_{\text{expert}}$. The discharge trick is used and the IG baseline is DT. For each parameter configuration training is repeated across 10 random seeds. Results are ordered by mean return in descending order.

Algorithm	Mean return	Best return	Worst return	Dataset size	Learning rate
PPO standard	5.705	5.781	5.421	–	–
PPO with IG	4.418	5.262	3.662	100,000	1×10^{-3}
PPO with IG	4.188	5.414	2.696	100,000	1×10^{-4}
PPO with IG	3.441	4.955	-3.873	100,000	3×10^{-4}
PPO with IG	3.273	5.361	1.006	1,000,000	1×10^{-4}
PPO with IG	2.828	5.331	0.711	1,000,000	3×10^{-4}
PPO with IG	0.559	4.178	-16.755	1,000,000	1×10^{-3}

Table 4.9: Configuration selected for the convergence plot.

Parameter	Selected value
Learning rate	10^{-3}
Training dataset size	10^5
Collection method	expert
Random seed	1233
IG baseline without DT	sell last 50

slightly lower than the standard PPO. The former requires more research to address, but it can be partly explained by the need for better hyperparameter tuning. The configuration in Table 4.1 and Table 4.2 was carefully selected after many experiments, and outside of those hyperparameters, the stability of the algorithm decreases substantially. The PPO model was therefore optimized to run with the original environment rewards, and there is no guarantee that the same configuration is optimal for the integrated gradients reward. In this thesis we have focused on optimizing the integrated gradients parameters, like dataset size and MLP learning rate, but further study is needed on the PPO parameters. For the latter concern, the explanation is twofold. First, the MLP has an error in predicting the total return. Secondly, and we believe more importantly, the off-manifold interpolation of integrated gradients increases this error. Finding an on-manifold path may be a worthwhile task to undertake, as it could improve both best performance and stability.

Even though the algorithm has problems, it is a significant improvement over RUDDER, as it can find a working reward redistribution and converge with just offline training, something that RUDDER was not capable of doing even in the discrete action space environment in Section 4.3. Moreover, training the MLP is far cheaper than the LSTM. For example, we took less than 30 seconds to train the MLP for 30 epochs over the dataset with 100 000 trajectories, while for the LSTM more than 1 hour was necessary. Replacing the LSTM with an MLP was possible since we simplified the training task, eliminating the need to predict the expected final return from incomplete rollouts.

Next we look at convergence plots of mean rewards during the training, to understand convergence speed. In Figure 4.13 we plot the evolution of the mean of episodes' returns during the PPO updates, using the parameters in Table 4.9.

Firstly, we observe that integrated gradients fails to converge when no discharge trick is used, while it succeeds with discharge trick. The reason lies in the problem of dealing with the last selling actions: using a PPO model with discharge trick gives the option to naturally handle credit assignment in the last steps by using the discharge trick baseline. In this way, the last actions receive no signal and integrated gradients can focus on the rest of the trajectory. If the underlying PPO model does not use the discharge trick, then there is no natural way to deal with the last steps. Although the Sell last 50 baseline produces a qualitatively acceptable reward, it is artificial, as there is no reason to choose -0.9, or any other specific value.

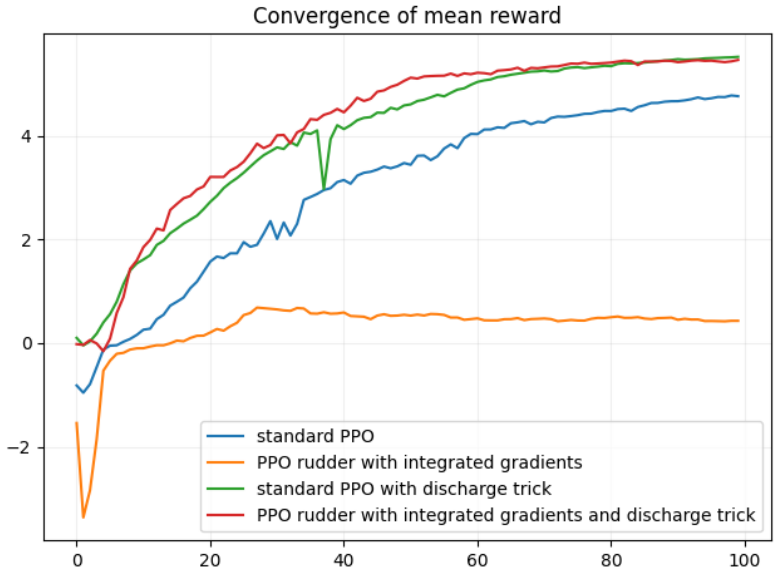


Figure 4.13: Evolution of mean rewards of rollouts during the first 100 PPO updates. Four variations of PPO models are shown: with or without the discharge trick and with or without reward redistribution through integrated gradients

Another interesting result is the magnitude of the improvement that the discharge trick produces. The PPO models that use the discharge trick, both with and without reward redistribution, have already converged by step 100, while the standard model requires other 100 steps.

Further research is needed to assess why, even though integrated gradients appears to successfully redistribute reward and assign credit to the earlier buying action, no significant speed-up in the convergence of the PPO is detected. A possible interpretation we propose is that the delay in collecting the reward for buying actions, of around 80 steps, is too short for the environment to suffer severely from delayed rewards. We showed that the most severe problem that impacts learning in gas storage environments is not the time delay in reward collection, but the possibility that the reward is not collected at all, when the gas is not sold. After the discharge trick solves this problem, it may be the case that no substantial further improvement can be gained from redistributing rewards. To test this hypothesis, insightful experiments could be conducted by extending the time window to values higher than $T = 365$, to extend the delay in collecting the reward.

5 Conclusion

In this thesis, we investigated the application of reward redistribution methods for reinforcement learning when applied to gas storage environments.

We found that RUDDER is not suited to this type of environment. Even in a simple environment with discrete actions, the long short-term memory was not able to find a global reward redistribution using offline training. Moreover, to obtain convergence, we had to employ expensive online training. For continuous actions, even online training failed. More research is needed to understand the exact reasons. In Section 3.2, we proposed the following interpretation: RUDDER is most effective when delayed rewards can be traced to abrupt changes in the action-value function. In gas storage, however, storage levels and market prices evolve gradually, so value changes are smoother and do not depend on single decisive steps. As a result, there may be no clear change point for RUDDER to assign credit to, forcing reward redistribution to spread credit across the trajectory. This mismatch likely explains the poor performance of RUDDER in this setting.

The algorithm we developed, which uses integrated gradients to redistribute the reward signal, performs significantly better than RUDDER. Using offline training alone, the new reward signal generated by the method satisfies the desired qualitative characteristics, and the method converges to high returns. To achieve this convergence, we employ the discharge trick described in Section 4.2.3, since it enables the use of a natural and powerful baseline.

Although the method produces a suitable reward redistribution, it does not significantly improve convergence speed compared to a standard PPO algorithm. Rather, the primary factor affecting convergence speed appears to be the use of the discharge trick. This result suggests two possible explanations: either a more effective reward redistribution method is required, or the discharge trick alone is sufficient to mitigate the delayed-reward problem in this class of environments. A final issue with the method is instability. This is caused both by the off-manifold interpolation in integrated gradients and by the need for more parameter tuning on the PPO parameters.

5.1 Future work

Several questions and directions arising from this thesis merit further investigation.

From a theoretical perspective, integrated gradients could be improved by developing an on-

manifold interpolation method between the baseline and the target trajectory, specifically designed for the manifold induced by the transition function of a reinforcement learning environment. The work of [Zah+24] could provide a useful starting point for this line of research.

In the algorithm design of integrated gradients, an important direction would be to identify an appropriate baseline for integrated gradients when the discharge trick is not used in the underlying reinforcement learning algorithm. As we noted in Section 4.4.2.2, this does not appear to be an easy task.

From an experimental perspective, two main directions are worth pursuing. First, the PPO parameters used in this thesis were tuned for the basic PPO model rather than for the model with reward redistribution based on integrated gradients. Tuning these parameters specifically for the proposed method could significantly improve algorithmic stability. Second, it would be useful to test gas storage environments with longer reward delays than those considered in this thesis. Such experiments could help determine whether reward redistribution can improve convergence speed in settings where the delayed-reward problem is more severe.

Bibliography

[Ach18] J. Achiam. *Spinning Up in Deep Reinforcement Learning*. https://spinningup.openai.com/. 2018.

[Arj+19] J. A. Arjona-Medina et al. 'RUDDER: Return Decomposition for Delayed Rewards'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019.

[Kro11] D. P. Kroese. *Monte Carlo Methods*. Lecture notes for the AMSI Summer School. 2011. URL: http://www.maths.uq.edu.au/~kroese.

[LHR22] D. D. Lundstrom, T. Huang and M. Razaviyayn. 'A rigorous study of integrated gradients method and extensions to internal neuron attributions'. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 14485–14508.

[Mni+15] V. Mnih et al. 'Human-level control through deep reinforcement learning'. In: *nature* 518.7540 (2015), pp. 529–533.

[Mol25] C. Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 3rd ed. 2025. ISBN: 978-3-911578-03-5. URL: https://christophm.github.io/interpretable-ml-book.

[NHR99] A. Y. Ng, D. Harada and S. J. Russell. 'Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping'. In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287. ISBN: 1558606122.

[RA98] J. Randløv and P. Alstrøm. 'Learning to Drive a Bicycle Using Reinforcement Learning and Shaping'. In: *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 463–471. ISBN: 1558605568.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. en. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. ISBN: 978-0-262-03924-6.

[Sch+15a] J. Schulman et al. 'High-dimensional continuous control using generalized advantage estimation'. In: *arXiv preprint arXiv:1506.02438* (2015).

[Sch+15b] J. Schulman et al. 'Trust region policy optimization'. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[Sch+17] J. Schulman et al. 'Proximal policy optimization algorithms'. In: *arXiv preprint arXiv:1707.06347* (2017).

[Sil+16] D. Silver et al. 'Mastering the game of Go with deep neural networks and tree search'. In: *nature* 529.7587 (2016), pp. 484–489.

[Sil+17] D. Silver et al. 'Mastering the game of go without human knowledge'. In: *nature* 550.7676 (2017), pp. 354–359.

[STY17] M. Sundararajan, A. Taly and Q. Yan. 'Axiomatic Attribution for Deep Networks'. In: *CoRR* abs/1703.01365 (2017). arXiv: 1703.01365. URL: http://arxiv.org/abs/1703.01365.

[Wil92] R. J. Williams. 'Simple statistical gradient-following algorithms for connectionist reinforcement learning'. In: *Machine learning* 8.3 (1992), pp. 229–256.

[Zah+24] E. Zaher et al. 'Manifold integrated gradients: riemannian geometry for feature attribution'. In: *arXiv preprint arXiv:2405.09800* (2024).