Combining Two Hindsight Approaches in Reinforcement Learning

Bence Szilágyi

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

2024

Abstract

In this project we introduce the concept of hindsight in reinforcement learning, and describe two recent approaches towards exploiting hindsight information. The first approach considered is Counterfactual Contribution Analysis (COCOA) [17], the second approach considered is the noise inference method of Counterfactual Credit Assignment (CCA) [16]. One of the primary aims of these approaches is to increase sample efficiency. We investigate whether combining them improves performance with regard to sample efficiency.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Bence Szilágyi*)

Acknowledgements

First, I would like to thank Dr. Michael Herrmann for his patient guidance and supervision, as well as my family for their continued and unconditional support.

I would like to acknowledge my core friend group at university, which has collectively christened itself “Andicam”. May the dream of the future Andicam Research Institute live on.

Thanks to Dani for bearing with me through my foolish ways, to Andi for being a generally amazing flatmate, and to Elias for the insightful discussions about RL.

To Wiki: gwop

Table of Contents

Chapter 1

Introduction

1.1 Motivation

We can all recall an experience of trying a new activity, and after a surprising outcome, asking ourselves the question: "**How** did I do that?". This question forms the basis of learning in hindsight. The use of hindsight in Reinforcement Learning (RL) has been investigated in papers such as Hindsight Experience Replay [3], and Hindsight Policy Gradients [23]. These approaches ask an agent to learn to reproduce situations it has experienced.

Another way humans often recontextualise our past experiences in hindsight is by asking a different question: "Did **I** do that?" When learning it is important to know to what extent our actions truly influenced certain outcomes. From the point of view of a given task, we can ask how lucky we were in a particular situation. Did we truly play well at the football game last Friday, or did we just happen to be on the same team as our friend who plays in the local league? One approach to separating luck from the skill of an agent is presented in [16]. They follow up their work in [15], where a similar, but even better performing approach is presented.

In this project we investigate a method proposed in [17] of combining the two aforementioned hindsight learning paradigms.

As one of the primary purposes of hindsight methods is to use the information available more efficiently, our comparisons between learning algorithms will focus on how many timesteps of interaction they require in order to converge. How much interaction with the environment a learning algorithm requires in order to converge is a fundamental question in RL. This property of an algorithm is frequently referred to as sample efficiency.

1.2 Objectives

The family of methods described in [17] will be referred to as COCOA Policy Gradient (PG) estimators, or COCOA algorithms from here onwards, as in [17]. The method

described in [16] will be referred to as CCA. We describe what a PG estimator is, and summarise the techniques utilised by COCOA and CCA in chapter 2.

The objectives of this project were as follows:

- Implement a COCOA PG estimator on top of a practical baseline RL algorithm.
- Implement a Linear Key-To-Door environment similar to the one described by [17], in which the credit assignment capabilities of the considered RL algorithms can be evaluated.
- Compare the resulting performance to the performance achieved by using a state of the art PG estimator. In particular, the COCOA approach was compared to Generalised Advantage Estimation (GAE) [25] used in a Proximal Policy Optimization (PPO) objective [24].
- Extend our COCOA implementation to incorporate hindsight information using similar methods to CCA. This extension will be referred to as CF-COCOA from here onwards. The idea behind CF-COCOA is first described in [17]. As will be explained in section 3.3.2.3, computing COCOA estimators corresponds to evaluating a query under a Do-intervention on the Structural Causal Model (SCM) corresponding to the underlying Partially Observable Markov Decision Process (POMDP) of the environment [17]. The "CF" in CF-COCOA stands for counterfactual, as computing CF-COCOA estimators corresponds to evaluating the same query under a counterfactual intervention (see section 3.3.2.4) on the same SCM [17].

The proposed objectives were completed. We find that our implementation of CF-COCOA does not significantly outperform our implementation of COCOA in our Linear Key-To-Door environment. However, we do find that the CF-COCOA hindsight classifier (see chapters 4 and 5) is capable of more accurately predicting the hindsight distribution in some environments (CartPole-v1 [1]).

We find that our implementations of both the COCOA and the CF-COCOA PG estimators result in algorithms highly sensitive to hyperparameters, and frequently subject to numerical instabilities. These issues are examined in chapter 6.

Chapter 2

Background

2.1 Overview

Reinforcement Learning (RL) poses two difficult challenges: exploration and credit assignment. In order to successfully explore an environment, we have to find courses of action which lead to favourable outcomes. In this sense exploration is synonymous with information gathering. Once given a set of experiences, we have to also separate the chaff from the wheat. Not all actions along a favourable trajectory contribute equally to the outcomes we want. We need to be able to identify which actions we should reinforce, and which actions we should avoid.

2.2 Credit assignment methods

Performing credit assignment involves estimating the effects of actions an agent takes on the outcomes it receives. Ultimately we are interested in increasing the expected return, by changing the policy our agent follows. In environments of computationally manageable size and complexity, classical RL approaches such as Q-Learning [26] can do well. Once we consider environments of greater complexity, classical approaches begin to struggle. For example, in infinite state spaces tabular Q-Learning may not be of much help. In such environments it is possible, and common, that exact state-action pairs are only ever visited once, so keeping track of values or action-values individually is infeasible.

2.2.1 Policy gradient estimators

Partly due to the aforementioned reasons, Deep RL has found significant success in recent years. Starting with Deep Q-Networks (DQN) [18], neural networks have become popular policy models. Neural networks have been trained to great effect with gradient ascent/descent methods in other learning paradigms such as supervised learning. Policy Gradient (PG) estimators [30] [27] attempt to bring this power to RL, enabling the use of well-established optimisers to tune our policy networks.

By today, PG methods have turned into one of the most popular and successful credit assignment methods. But despite their success, PG estimators come with the drawback that their variance can be quite high, which may lead to unstable learning [25] and brittle algorithms with high sensitivity to hyperparameters.

The most common PG estimators, such as REINFORCE [30] only take into account observations at or before the timestep the action was taken in. As gaining additional information (conditioning on more variables) generally reduces variance [8], when inferring the values of random variables, it is a reasonable idea to consider using the entire trajectory to derive PG estimates. This is in line with the intuitions outlined in the second paragraph of section 1.1.

Naively incorporating future information will generally lead to biased PG estimators [11]. This is due to the fact that the observed outcomes are sampled according to a given policy, so their distribution is affected by how the sampling policy chooses actions. Recent research has shown that it is possible to incorporate future information in a principled manner, such that we retain a lot of the benefits of hindsight learning, while ending up with PG estimators that are unbiased in theory, and only minimally biased (due to model error) in practice [11]. Their work relies on similar ideas to Hindsight Experience Replay (HER) [3] and Hindsight Policy Gradients (HPG) [23], which we will group together under the umbrella of goal-conditioning approaches.

The method of Hindsight Credit Assignment (HCA) [11] has significant drawbacks, as the importance sampling scheme used has been shown in many environments to introduce more variance than what may be acceptable due to the benefits gained from goal-conditioning [17]. Therefore it is difficult for the resulting PG estimator to be used in practice [28]. Nevertheless, the ideas presented are exciting, and have been built upon in more recent work, such as the previously mentioned CCA [16] and the more closely related COCOA [17]. To put HCA [11] into more context, we first describe HER [3] and HPG [23] in more detail in the next section. In the section following that we go into more detail about [11] and some further works they inspired.

2.3 Goal-conditioning

It can be useful to learn how to effectively navigate the state space in general [3]. Consider sparse reward environments, where the agent only receives a reward signal at the end of the episode. In these environments the final reward is also very frequently 0, with greater reward only being provided if the agent achieved some task. Sparse reward environments pose a difficulty where we may not encounter any reward signal in the majority of the episodes our agent plays. Such sparseness makes sample efficiency a high priority: when we do finally get some reward, we need to be able to learn how to get to states that provide this reward quickly.

By knowing about the underlying structure of the environment, we can learn more about a reward we receive: states from which we can get to where we received the reward should now also be considered more valuable. Hindsight Experience Replay (HER) [3] provides a way to learn about the underlying structure of an environment even without a reward signal by conditioning the policy on a goal, which can be any reasonable

representation of for example a state, a reward, or a future return. Trajectories are collected according to a goal determined by the task the agent is trying to complete.

Their main insight is that once stored experiences are collected in our replay buffer whilst following some original goal, we may recall them with the original goal replaced by some other goal that was actually achieved over the trajectory. While this does enable the agent to learn about the latent (underlying) structure of the environment, it also presents new questions.

How should we choose our goals? We face several issues. The obvious choice for our primary goal is to maximise the return. However, this requirement is generally very difficult to formulate such that it can be "understood" by common policy models. Thus we generally need to provide the agent with a specific return it should achieve [3]. The difficulty then comes from the fact that we may not know what the least upper bound is on the return, or there may not be an upper bound. How to select intermediate goals to learn from in hindsight is also not obvious. In practice, we can use heuristics to answer these questions and get good results [3].

In Hindsight Policy Gradients (HPG) [23] the ideas of HER [3] are extended to policy gradient methods by importance sampling. Importance sampling attempts to transform expectations taken over the distribution of outcomes under one policy to expectations taken over the distribution of outcomes under the policy we are improving. To achieve this, we re-weight experiences based on the ratio between the probabilities assigned to each action by the policy we are improving and the probabilities assigned to each action by the policy under which they were performed.

It is worth noting that importance sampling is used frequently in RL, however its most common purpose is to update a more recent policy by using trajectories sampled by older policies [26]. In HPG, importance sampling is used to update a policy conditioned on one goal, by using trajectories collected by a policy conditioned on another goal.

Goal-conditioning has also proven to be a useful idea in different contexts. Decision Transformer (DT) [5] and Trajectory Transformer [13] use similar ideas to gain good performance on offline RL problems. Both methods were developed concurrently, and although they operate on similar principles, the specific techniques used are quite different. DT models the next action based on a desired return and the previous state and action, so it is more similar to HER and HPG than Trajectory Transformer. The primary difference between DT and these older goal-conditioning methods are that DT is an offline RL method, while HER and HPG are both online. An advantage of DT is that it is able to leverage the power of modern sequence modelling approaches in the form of transformers.

As an aside, we note that there also exist ways to generalise DT to the online domain using iterative improvement approaches [31].

Trajectory Transformer learns in an autoregressive manner, similar to GPT, learning to predict the next states, actions and rewards. It does very well at predicting the dynamics of the environment, and can be used with planning approaches. Alternatively it can also be goal-conditioned by prepending the goal to the start of the considered sequences during both training and inference. Trajectory Transformer is also initially limited to

offline RL.

2.4 Future-Conditioning Hindsight Approaches

The essential idea of HCA [11] is to approximate how much each action contributes to reaching a given future state using an application of Bayes' rule. This enables the computation of a similar importance sampling PG estimator as the one used in HPG [23]. Although the idea of HCA is very exciting, the specific formulation provided in [11] is not very practical, as using individual states as hindsight "goals" (in HER/HPG terminology) yields a frequently high-variance importance sampling scheme [17], and their return-conditioned formulation also has significant drawbacks.

In the COCOA family of algorithms [17], individual rewards, or state encodings fully predictive of the rewards are proposed as future-conditioning targets. They show that the choice of individual rewards is in some sense optimal for variance reduction. However, in the case that very different conditions in the environment yield the same reward, or in a sparse reward setting, conditioning directly on the reward values becomes impractical. Conditioning on reward-predictive encodings has been suggested for this reason. The degree of variance reduction maintained depends on the degree to which these encodings group individual states together. As long as our encodings are coarse enough, the authors claim they will be helpful. They present this theory and more in very useful detail, but do not supply examples of their method working in more complex environments. Additionally, their work retains a limitation of [11], whereby their PG estimator will only be as good as the future-conditioned action distribution they can learn. Albeit they do show that their PG estimator is relatively robust to inaccuracies in the hindsight distribution for some environments they use, this is not quite enough to fully expect the same to hold in more complex environments.

Nevertheless, we see promise in their results, which is why we decided to use their work as the basis for this project. For simplicity, we only implement the reward-conditioned version of their algorithm, leaving conditioning on state-encodings for future work.

Chapter 3

Methods

3.1 Overview

In this chapter we briefly introduce terminology and notation from Reinforcement Learning (RL) and causality theory, then mention some connections between the two fields. In particular we explain how we can model Markov Decision Processes (MDPs), as well as Partially Observable MDPs (POMDPs) as Structural Causal Models (SCMs).

We are interested in SCMs because they will allow us to express the benefits we are hoping to gain from extending COCOA [17] to counterfactual interventions. In fact, both Do-interventions and counterfactual interventions will be defined as manipulations of SCMs.

3.2 Reinforcement Learning

We formulate RL as a machine learning problem, where an agent interacts with an environment that provides observations and numerical reward signals in response to the agent's actions. The agent's goal is to learn how to maximise the expected value of the sum of rewards it receives.

The process of interaction between the agent and the environment is resolved over discrete timesteps. At each timestep t the agent is given an observation o_t, responds with an action a_t, and receives a (possibly stochastic) reward r_t.

In this project we consider the episodic setting, where each interaction eventually terminates, producing a finite sequence of triples $(o_0, a_0, r_0), (o_1, a_1, r_1), \ldots, (o_T, a_T, r_T)$. Such a sequence is called a trajectory, and the interaction over which it was collected is called an episode. The sum of rewards within a trajectory τ is called the return $G(\tau) = \sum_{t=0}^{T} r_t$. Over several episodes the agent accumulates experience in the form of trajectories, which it may learn from in order to maximise the returns achieved in later episodes.

We define the agent's policy π as a map from either states s_t when acting in fully observable MDPs, or histories $(o_0, o_1, \ldots, o_t)$ when acting in POMDPs to probability

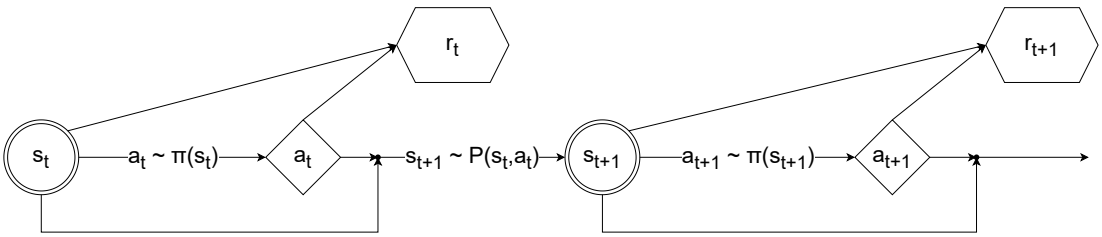

Figure 3.1: Markov Decision Process, with actions selected by policy π

distributions over possible actions at timestep t. The choice of input in the latter case is explained in section 3.2.1.2. At a given timestep the agent chooses an action by sampling from the policy distribution corrresponding to the current history.

3.2.1 Markov Decision Processes

The environment in an RL problem can be conveniently expressed as an MDP, which is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. Here $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}$ is a map that takes a state-action pair to a probability distribution over states, and $\mathcal{R}$ is a map that takes a state-action pair (s, a) to a reward that is immediately given to the agent upon reaching (s, a).

3.2.1.1 Acting in a fully observable MDP

In the simplest case with an MDP environment, the interaction at timestep t is resolved as follows (see also Fig. 3.1). We first sample a state $o_t := s_t \sim O(s_{t-1})$, (where we include a dummy state $s_{-1} \in S$). Then we sample an action from the policy $a_t \sim \pi(s_t)$. Finally the agent receives $r_t = \mathcal{R}(s_t, a_t)$.

Note that the distribution $\tau \sim \Gamma$ of trajectories that may arise as a result of an episode is fully determined by the MDP and the policy π. Since for a given problem we consider the environment fixed, we write $\Gamma(\pi)$ to denote that given an environment the sampling distribution of trajectories depends solely on the policy.

Also note that in our formulation the reward is a deterministic function of the trajectory. This does not lead to any loss of generality, as we will immediately introduce partially observable MDPs as environments, and will remain in that setting for the rest of this project. POMDPs eliminate any loss of generality due to deterministic reward, as from the point of view of the agent, stochasticity and partial observability are in a sense indistinguishable [19]. In other words any stochasticity in the environment (e.g. in the reward function) can be considered as a pre-determined variable of the hidden state.

It is also shown that agents can gain an advantage by modelling stochasticity in their environment as partial observability [19]. This is due to similar mechanisms to those described in section 3.3.2.4.

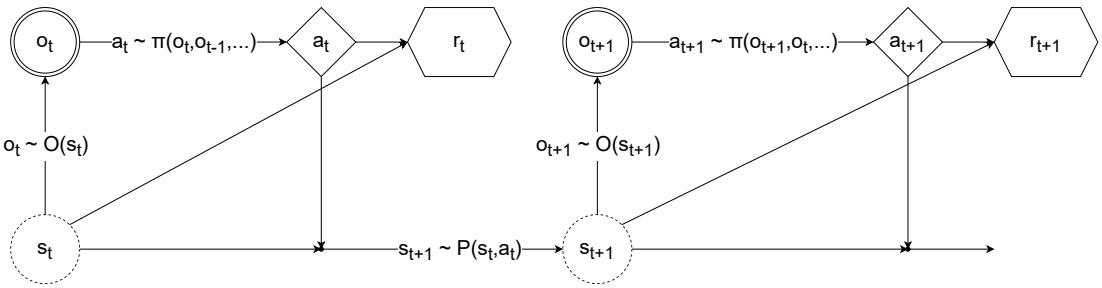

Figure 3.2: Partially Observable MDP, with actions selected by policy π

3.2.1.2 Acting in a partially observable MDP

In real-world problems it may be the case that we have an imperfect knowledge of the world around us. We can model this by separating observations from the underlying state of the MDP. After sampling a state s_t, we do not directly provide s_t to the agent, but instead sample an imperfect representation $o_t \sim O(s_t)$, where each state s_t has a corresponding probability distribution O over some set of observations.

Most fundamental RL algorithms only come with theoretical guarantees of convergence in the fully observable setting. Initially this presents a worrying picture for the partially observable case. However, it turns out that by providing the full history of observations to the agent at each timestep, a partially observable MDP M can converted to a fully observable MDP M' [32]. This comes with the tradeoff that generally M' will be more complex than M.

In practice, considering the full history at each timestep is computationally expensive, so modern RL algorithms learn a compact representation, essentially providing the agent with 'selective memory' to keep track of relevant information, e.g. as in [22].

3.3 Causality

As mentioned in Section 2.2, it is natural to pose the question of credit assignment in RL as a question about the effect of each action on the consequent outcomes. However, as [20] states, and as we have seen already in this chapter, when acting in MDPs the expected outcomes conditioned on a given action depend on the policy being followed. In particular, the expected return given a single action is thus tied to all future actions.

An example is given in [20] of an environment where it is clear that the true effect of an action is often far more limited in scope. Consider an MDP with a linear sequence of states. In each state we may choose from one of two actions: either we take the "upper" or the "lower" route to the next state. The upper route provides a reward of 1, whereas the lower route provides no reward. Although this is an extreme example where the effects of an action are limited to the immediate consequences, it highlights the need for a more fine-grained definition of the effect of an action than the associated expected return.

In particular we consider the Rubin-Neyman framework of causality [12] in this project.

3.3.1 Advantage as a causal effect

Continuing to follow [20], in this subsection we describe the connection they draw between the causality and RL literatures. This connection is not novel, as many recent papers in RL, for example [4] and [16] make use of ideas from the causality literature. However, as the work done in the present project, following [17], relies heavily on these ideas, a short introduction will serve us well.

In RL, we define the action-value function $Q_\pi(s_t, a_t)$ as the expected future return after performing action a_t in state s_t, and acting according to the policy π afterwards. The value function $V_\pi(s_t)$ is defined as the expected future return after a state s_t, when acting according to the policy π.

$$Q_\pi(s_t, a_t) = \mathbb{E}_{\tau\sim\Gamma(\pi)}[G(r_t + \sum_{t'>t} r_{t'}]$$
$$V_\pi(s_t) = \mathbb{E}_{\tau\sim\Gamma(\pi)}[G(\sum_{t'\geq t} r_{t'}]$$

The advantage $A_\pi(s_t, a_t)$ of action a_t at state s_t is then defined as the difference

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t).$$

3.3.2 Structural Causal Models

Structural Causal Models (SCMs), such as in Figures 3.3 and 3.4, are a tool of causality theory. They model dependencies between random (noise) variables and deterministic processes, called causal mechanisms, as directed graphs. We denote deterministic processes as square nodes in the graph, and stochastic nodes as circles. Dependencies are depicted as arrows between nodes. Importantly, in this project we make the assumption that the noise variables in an SCM should be independent. Any interaction between noise variables should thus come as a result of causal mechanisms.

3.3.2.1 Modelling MDPs as SCMs

See Figure 3.4 for how POMDPs can be modelled as SCMs. The construction is similar to the case of MDPs. Any stochastic transitions between states, observations, or actions are removed, and the randomness is pushed back as an input to deterministic functions giving rise to these values.

MDPs can be modelled as SCMs by replacing each state with a deterministic causal mechanism depending on the value of an independent noise variable, as well as the previous state-action pair. Any stochasticity in the policy is also modelled as a noise input. See Figure 3.3 for more details.

3.3.2.2 Modelling POMDPs as SCMs

Modelling POMDPs in this way essentially provides a reparameterisation of the environment where we may infer the values of the hidden noise variables based on our observations [4]. Combining such inference with learning to approximate the results of the causal mechanisms can lead to a more accurate representation of the hidden state.

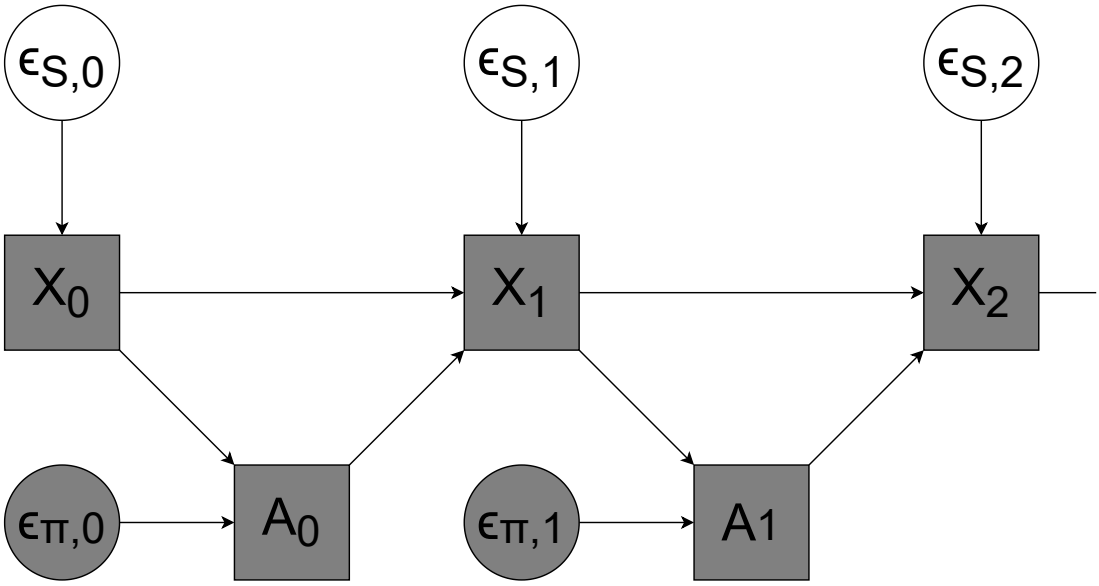

Figure 3.3: Modelling an MDP using an SCM, as in [16], values of grey nodes are obversed, values of white nodes are not observed. Square nodes are deterministic, circle nodes are stochastic. To simplify our diagram, the reward at time t is considered to be encoded in X_{t+1}.

Reducing the observed stochasticity in the environment in such a manner can lead to lower-variance estimates of values that depend on the underlying state.

3.3.2.3 Do-interventions

A Do-intervention can be performed on an SCM by cutting all incoming connections to a node X, and setting its value to y. We denote such an intervention by $\text{Do}(Y \to y)$. Given an SCM $\mathcal{M}$ the resulting SCM after a Do-intervention is denoted as $\mathcal{M}^{\text{Do}(X \to y)}$. In $\mathcal{M}^{\text{Do}(X \to y)}$ the distribution of all downstream variables are changed. The effect of an action can be interpreted as the result of a Do-intervention in our SCM. For example, to see the effect on the expected future return, which we know as the advantage of the given action, we can find the difference between the expected value in the SCM under $\text{Do}(A_t \to a_t)$ and the expected value without any intervention. The advantage function is thus recovered.

3.3.2.4 Counterfactual interventions

A counterfactual intervention is a generalisation of a Do-intervention, whereby we not only fix the value of a given node, but also infer the values of (some of) the hidden noise-variables. This is of interest to us as we hope to reduce the variance of the resulting outcome distributions due to the conditioning on additional information [8].

To put things into more familiar terms: if we model the environment of an RL agent as an SCM, we may be able to infer the noise variables of this SCM based on our observations. Incorporating this information into our PG estimate, we may then hope for lower variance. In the literature the noise distribution conditioned on information up

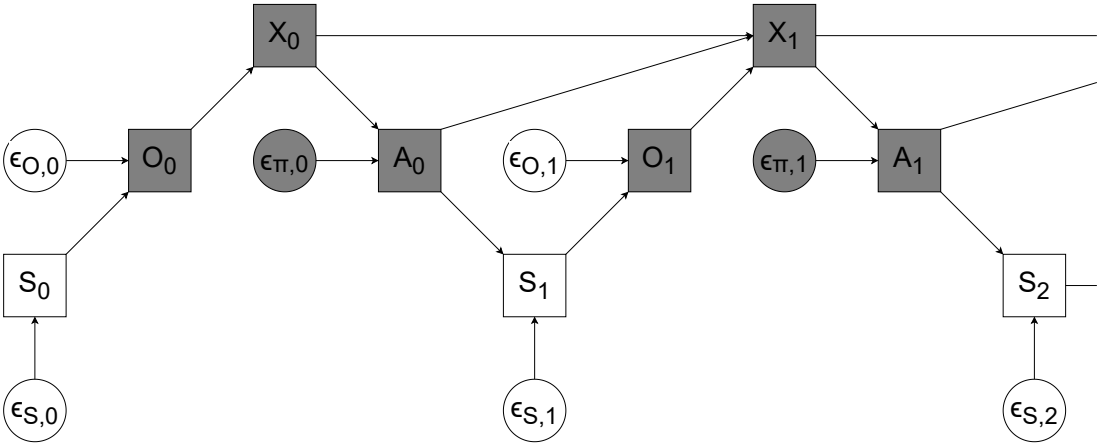

Figure 3.4: Modelling a POMDP using an SCM, as in [16], values of grey nodes are observed, values of white nodes are not observed. Square nodes are deterministic, circle nodes are stochastic. To simplify our diagram, the reward at time t is considered to be encoded in X_{t+1}.

to timestep t is frequently called the prior noise distribution, and the noise distribution conditioned on information from the entire trajectory is frequently called the posterior noise distribution [10] [29].

As a sidenote, based on this argument we can also see why there is such a close connection between the RL and Variational Inference literatures (e.g. see [21]).

In the following chapter we use the ideas we have introduced here in an attempt to turn them to our benefit, following the COCOA [17] and CCA [16] approaches.

Chapter 4

Design

4.1 Overview: why COCOA should see the future

In the previous chapter, we have become familiar with the theoretical idea of "denoising" outcome distributions. Here, following an idea proposed but not implemented by [17], we describe in short one possible way to go about this, with the goal of reducing the noise in COCOA PG estimators.

4.2 Who learns what: architecture and backpropagation

CF-COCOA builds on both COCOA [17] and CCA [16], and as such we end up with an architecture combining elements of both approaches. We aim to learn some summary statistic or encoding $\hat{\phi}_t$ of the future of a trajectory in relation to timestep t, such that $\hat{\phi}_t$ is independent of the action a_t at time t. This $\hat{\phi}_t$ can be thought of as a summary of the noise variables in an SCM representing the environment.

We learn the encoding $\hat{\phi}_t$ directly through backpropagation as we learn a hindsight action classifier $h_\theta(\hat{a}_t|\hat{s}_t, \hat{\phi}_t, r_t)$. As an aside, for the reminder of this project we use θ to represent the collection of all learned parameters of all our models.

Note that as described in [19], inferring the posterior noise distribution can also be thought of as marginalising over deterministic hidden states. Therefore thinking of $\hat{\phi}_t$ as a hidden state predictor would also be accurate.

We provide a schematic of our architecture in Figure 4.1. Note that our implementation builds on top of a baseline algorithm, Rec-PPO (short for Recurrent Proximal Policy Optimization) [22], as implemented by the SheepRL [6] library. This implementation has additional components, such as Multiencoders to ingest the raw observations provided to the agent. We will minimise mentions of these components in this project, as our focus is not on them.

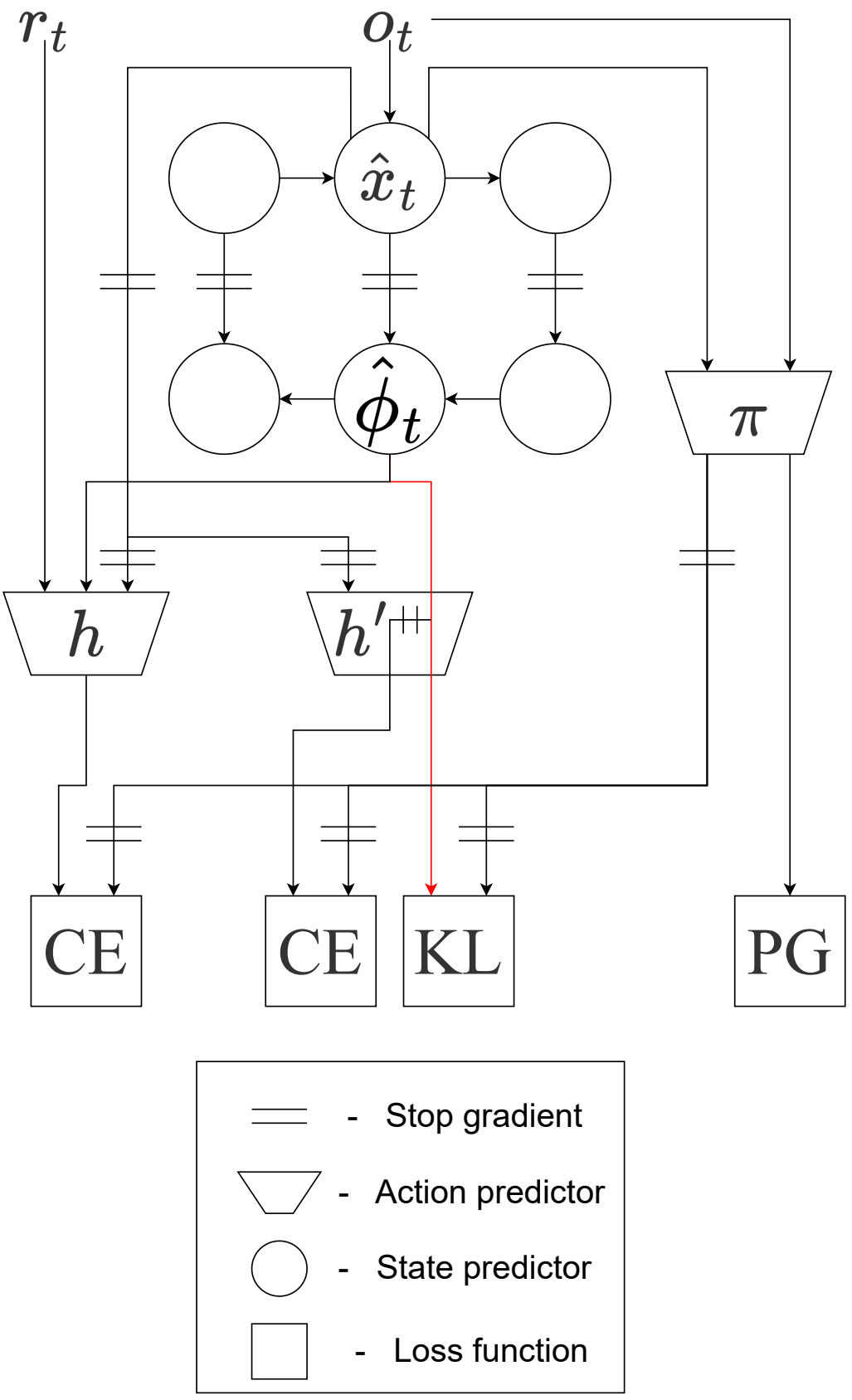

Figure 4.1: Data flow between our models. Our architecture is based on that presented in [16], combined with the ideas of [17]. The red arrow denotes that the KL-divergence is only used to train $\hat{\phi}_t$, and not h'_θ directly. The empty state predictor nodes represent the previous as well as the future states of the forward and backward RNNs.

4.3 Unbiasing our estimator by selective forgetting

In general, conditioning a PG baseline on the full trajectory can lead to biased PG estimators, as stated in [19] and [16]. In Corollary 1. of [16] the authors show how conditional independence given $\hat{s}_t$ between A_t, the random variable (RV) representing the action at time t, and Φ_t, the RV representing future information being incorporated in their PG estimator can lead to unbiasedness. In Theorem 2. the authors of CCA [16] elaborate on this, stating that such conditionally independent Φ_t essentially encodes inferred noise on the trajectory when the agent is considered to be acting in an SCM.

We build on these results according to the idea presented in [17]. Instead of requiring that $\hat{\phi}_t$ contain useful information for estimating the value function, we train it to contain useful information for approximating the hindsight distribution $h_\theta(a_t|\hat{s}_t, \hat{\phi}_t, r_t)$. As Φ_t represents the inferred noise on the trajectory, it is exactly what we need to extend COCOA [17] to a query of the outcome distribution after performing a counterfactual intervention.

As in [16], we also train an auxiliary hindsight classifier $h'_\theta(a_t|\hat{s}_t, \hat{\phi}_t)$, and minimise the KL-divergence

$$\sum_t \mathbb{KL}(\pi(A_t|S_t)||h'_\theta(A_t|S_t, \Phi_t).$$

This has the effect of regularising for independence between the two distributions. Intuitively, the KL-divergence represents a sort of distance between two probability distributions. If the two distributions are close to each other, i.e. the probabilities of actions at time t remain the same between them, then they are independent. Thus the KL-divergence should be minimised to achieve what we want.

Chapter 5

Implementation

5.1 Components

In this chapter we describe the learning algorithm our implementation follows. Afterwards, we provide details about how we implement the model used to encode the future information $\hat{\phi}_t$, as well as the primary and auxiliary hindsight classifiers h_θ, h'_θ.

5.1.1 Pseudocode: CF-COCOA in a recurrent-memory algorithm

Our algorithm combines steps from COCOA [17] and Rec-PPO [22], and proceeds as shown in Algorithm 1.

The contribution coefficients $w(\hat{s}_t, a_t, r_k)$ are calculated as follows [17]:

$$w(\hat{s}_t, a_t, r_k) = \frac{h_\theta(a_t|\hat{s}_t, \hat{\phi}_t, r_k)}{\phi_\theta(a_t|\hat{s}_t)} - 1.$$

In Algorithm 1, $p(a)$ refers to the one-hot "distribution" of factual actions taken during the sampled trajectory. We denote the weights (for the optimiser) of each loss by α_*. For example the coefficient of the policy gradient loss is denoted α_{PG}. The loss L^E_π is an entropy regularisation loss to promote exploration [24].

5.1.2 Backward RNN

Recurrent Neural Networks (RNNs) are frequently used to encode information in arbitrary-length sequences. They are initialised with a hidden state vector (commonly a zero vector), and process sequences elementwise, as at each timestep they are given the previous hidden state and the current element of the sequence and produce a new hidden state.

We use a backward RNN to propagate information from the end of the trajectory back to a timestep t_0. At each timestep t along the trajectory, the backward RNN is provided with the observation features $\hat{o}_t$ yielded by the Multiencoder, the hidden state $\hat{x}_t$ of the forward RNN, as well as its own previous hidden state, $\hat{\phi}_{t+1}$. The encodings

Algorithm 1 CF-COCOA on Rec-PPO

for I iterations **do**
  Collect T timesteps of experience acting according to policy π_θ
  Split the collected experience into trajectories
  Apply padding so the replay buffer can be stored as a tensor
  $T =$ length of the longest trajectory
  for K epochs **do** ▷ Train the hindsight classifiers
    for $t = 1$ to $T-1$ **do**
      $L^{CE}_{h'} = -\sum_{a\in\mathcal{A}} p(a) \log h'_\theta(a|\hat{s}_t, \hat{\phi}_t)$ ▷ $p(a)$ is the one-hot target
      $L^{KL}_{h'} = -\sum_{a\in\mathcal{A}} \pi_\theta(a|\hat{s}_t) \log \frac{\pi_\theta(a|\hat{s}_t)}{h'_\theta(a|\hat{s}_t, \hat{\phi}_t)}$
      Update the parameters θ along the gradient w.r.t. $\alpha_{CE,h'} L^{CE}_{h'} + \alpha_{KL} L^{KL}_{h'}$
      for $k = t+1$ to T **do**
        $L^{CE}_{h} = -\sum_{a\in\mathcal{A}} p(a) \log h_\theta(a_t|\hat{s}_t, \hat{\phi}_t, r_k)$ ▷ $p(a)$ is the one-hot target
        Update the parameters θ along the gradient w.r.t. $\alpha_{CE,h} L^{CE}_{h}$
      end for
    end for
  end for
  for $t = 1$ to $T-1$ **do** ▷ Compute the contribution coefficients
    for $a \in \mathcal{A}$ **do**
      $W_a \leftarrow \sum_{k=t+1}^{T} w(\hat{s}_t, a, r_k)$
    end for
  end for
  for N epochs **do** ▷ Train the policy
    $L^{PG}_{\pi} = \sum_{t\geq 0} (\log \pi(a_t|\hat{s}_t) r_t + \sum_{a\in\mathcal{A}} \log \pi(a|\hat{s}_t) W_a)$
    Update the parameters θ along the gradient w.r.t. $\alpha_{PG} L^{PG}_{\pi}$
  end for

$\hat{\phi}_t(\hat{o}_t, \hat{x}_t, \hat{\phi}_{t+1})$ produced by the backward RNN are fed to the primary hindsight classifier, $h_\theta(a_t|\hat{s}_t, \hat{\phi}_t, r_{t'})$ where t' is some timestep after t, as well as the auxiliary hindsight classifier $h'_\theta(a_t|\hat{s}_t, \hat{\phi}_t)$.

The backward RNN is trained by backpropagation through h_θ and h'_θ, in order to ensure that the embeddings $\hat{\phi}_t$ are independent from the action a_t conditional on $\hat{s}_t$, as well as to maximise the accuracy of the primary hindsight classifier h_θ. Importantly, gradients are not propagated back through $\hat{o}_t$ to the Multiencoder, nor through $\hat{x}_t$ to the forward RNN. We go into more detail about how h_θ and h'_θ are trained in subsections 5.1.3 and 5.1.4.

5.1.3 Hindsight classifier: a hypernetwork

We follow the original implementation of COCOA [17] in using hypernetworks as both of our hindsight classifiers. Multi-Layer Perceptron (MLP) hindsight classifiers have also been tried as part of the original implementation by [17], but they were found to perform relatively poorly [17]. The hypernetwork architecture they propose incorporates the action distribution of the acting policy, whose inductive bias is hypothesised as the reason behind the hypernetwork's stronger performance [17]. Proposition 5. of [17] shows that incorporating this additional information does not bias the resulting PG estimator.

5.1.3.1 What is a hypernetwork?

Let d_a denote the dimensionality of the action space. The hypernetwork we use is composed of the following layers:

- A linear layer with a ReLU activation. Takes the concatenation of $\hat{s}_t$, $\hat{\phi}_t$, and $r_{t'}$ as input. Its output is fed to:
  - A linear layer with a ReLU-g [17] activation function. Its outputs α are d_a-dimensional.
  - A linear layer with a ReLU-g [17] activation function. Its ouput G is a $d_a \cdot 2d_a$-dimensional vector, rolled into a matrix of shape $d_a \times 2d_a$.
- Finally the output of the hypernetwork is computed as

$$h_\theta(a_t|\hat{s}_t, \hat{\phi}_t, r_{t'}) = \log \pi(a_t|\hat{s}_t) + Gp,$$

  where p denotes the concatenation of $\log \pi(a_t|\hat{s}_t)$ and $\log(1 - \pi(a_t|\hat{s}_t))$.

The gated ReLU function ReLU-g is defined as in [17]:

$$\text{ReLU-g}(x) = \text{ReLU}(x_{0:n/2}) - \text{ReLU}(x_{n/2:n}).$$

As explained in chapter 6, the distributions learned by our hindsight classifiers can be rather sharp and unstable, partly due to weights and gradients with large absolute values. Sharp distributions introduce significant numerical instability to the computation of the contribution coefficients $w(\hat{s}_t, a_t, r_k)$. To combat these issues, we introduce layer norms before every activation function in our hypernetworks.

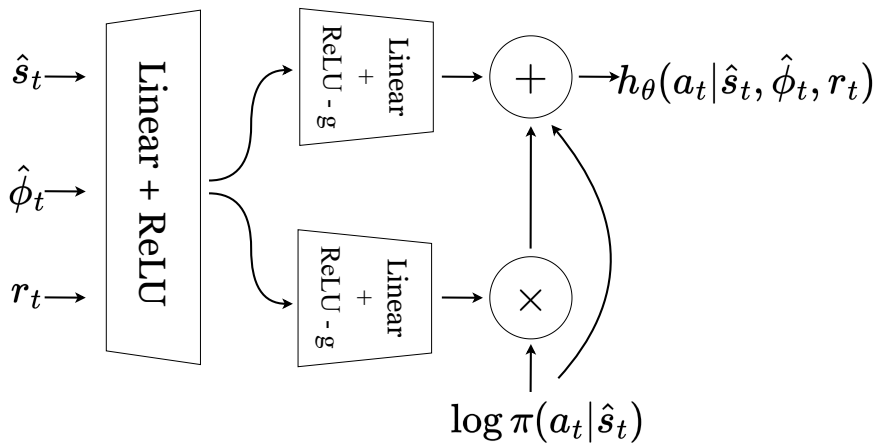

Figure 5.1: Schematic of our hypernetwork, adapted from [17].

5.1.3.2 Training the hindsight classifier

In order to provide useful information about the latest policy, during one iteration of the algorithm the hindsight classifier is trained immediately after each rollout, and immediately before the policy is trained using the contribution coefficients produced by the hindsight classifier. The supervised training is achieved through minimising the cross-entropy between the predictions of the hindsight classifier, and the actions sampled in the encountered trajectories. It is an important detail that the target distribution is that of the sampled actions, not directly the predictions of the policy. Our goal is precisely to learn how the action distribution predicted by the policy is affected by the additional future information.

In Algorithm 1, the cross-entropy loss for the hindsight classifier is denoted $L^{CE}_{h_\theta}$.

5.1.4 Auxiliary hindsight classifier: declaring independence

By the reasons outlined in Appendix I of [17], we wish to regularise the backward RNN to not encode information about the distribution of the action at time t. They suggest one approach to achieve this, inspired by CCA [16]. We follow their proposed approach.

Another hypernetwork is trained on a similar task as our primary hindsight classifier, however this time we do not condition on the future reward. The learning of this auxiliary hindsight classifier $h'_\theta(a_t|\hat{s}_t, \hat{\phi}_t)$ is achieved through a cross-entropy loss, as in our primary hindsight classifier. However, this time we do not backpropagate the cross-entropy loss through the backward RNN.

To achieve independence between $\hat{\phi}_t$ and a_t, we regularise the encodings of the backward RNN, by adding a KL-divergence loss between the auxiliary hindsight distribution $h'_\theta(a_t|\hat{s}_t, \hat{\phi}_t)$ and the action distribution induced by the policy $\phi_\theta(a_t|\hat{s}_t)$. The weights of our auxiliary classifier are frozen with respect to the KL-divergence, so only the parameters of the backward RNN are updated at this stage.

In Algorithm 1, the cross-entropy loss for the auxiliary hindsight classifier is denoted $L^{CE}_{h'_\theta}$, and the KL-divergence loss is denoted $L^{KL}_{h'_\theta}$.

5.1.5 Optimiser, loss weights, hyperparameters

In our experiments we use the ADAM optimiser [14], with a learning rate of $1e-3$ and $\varepsilon = 1e-4$. All other hyperparameters of the optimiser are as default in the PyTorch library.

We performed a manual hyperparameter search, and found a set of hyperparameters that work reasonably well in both the 10-step and 20-step long instances of our Linear Key-To-Door environment. The hyperparameters and full configurations for all experiments communicated in this report (see chapter 6) can be found in the supplementary materials.

We note that although the found hyperparameters lead to reasonable performance, the issue of over-sharpened policy distributions remains, which causes most runs to terminate early.

5.2 Techniques and caveats

Over the course of development, several technical difficulties were encountered. These difficulties include conceptual issues requiring careful consideration in order to ensure our implementation stays faithful to the theoretical requirements discussed. Issues of optimisation were also considered, ensuring the computational time and resources required to run our implementation do not render it unusable in practice.

5.2.1 Replay buffer and sampling

Our algorithm requires that entire trajectories be available to train the primary hindsight classifier, as we want to compute the contribution coefficients of actions for all future timesteps. Additionally, running the backward RNN from as far in the future as possible increases the available information, and further reduces noise.

By default, Rec-PPO [22] splits trajectories into sequences of a fixed length. This option can be disabled, but it is important to note that to ensure the correctness of our algorithm, it must be disabled when building on top of Rec-PPO [22].

As variable length sequences cannot be efficiently stored as tensors in the replay buffer, Rec-PPO [22] applies padding and stores a mask that is used in later computation to ensure the padding values do not effect results. The encodings of the backward RNN and the additional losses we compute have been implemented such that they also use the mask to disregard padding values.

5.2.2 Accounting for moving targets

The hindsight classifier is required to always approximate the hindsight distribution corresponding to the latest policy, with which the most recent rollouts were performed. As we are interspersing the updates of the policy with the updates of the hindsight

classifier, a situation with a moving target in the form of the policy is created. We accomodate for this difficulty after each rollout by first updating the hindsight classifiers for all epochs, and only then updating the policy, using the contribution coefficients computed using the updated hindsight classifier.

5.2.3 An optimisation: splitting epoch counts

We find that with our current implementation, training the hindsight classifiers takes a significant amount of computation time compared to the policy updates. This is likely due to the quadratic complexity of updating the hindsight coefficients at each timestep t with regard to all future timesteps k.

The lengthened training times were making it difficult to debug, as well as to run a reasonable number of experiments to see if our hindsight classifiers are able to learn the required distribution. To handle this difficulty, we used a different number of training epochs for training the hindsight classifiers than what we use to train the policy. We found that even using a single epoch of training after each rollout, the hindsight classifier converges reasonably well on CartPole-v1 [1].

The added hyperparameter of hindsight epochs had to be adjusted all the way back to 8 (the number of epochs used for policy learning in our experiments) for success in the Linear Key-To-Door environment we implemented.

Chapter 6

Evaluation and experiments

As the primary goal of our algorithms is to successfully assign credit to actions for the outcomes encountered, we choose to evaluate their performance in an environment that tests this exact capability. We implement a variant of the Linear Key-To-Door environment discussed in [17]. Several modifications to the original environment are made. Instead of listing the differences, we give more detail to our Linear Key-To-Door environment in the next section.

6.1 Our Linear Key-To-Door environment

See Figure 6.1 for a schematic of our Linear Key-To-Door environment.

The environment we chose to implement is in principle similar to the one presented in [17], but is more challenging. Instead of a distractor task, where between picking up the key and getting to the door the agent can pick up "apples" worth a relatively small reward each, we provide the agent with fully random rewards at each in-between step. This is inspired by Quantile Credit Assignment [15], a follow-up work to CCA [16], where it is demonstrated that such a fully random version of a key-to-door environment is more challenging than the version with a distractor task.

6.2 Evaluating our hindsight classifiers on CartPole-v1

Before any benchmarking of our PG estimators is reasonable, we should make sure that our hindsight classifiers are capable of learning the hindsight action distributions. We tested this by running instances of Rec-PPO on the CartPole-v1 task, and training hindsight classifiers on the action distribution of this agent. The cross-entropy losses of the hindsight classifiers in these experiments are presented in Figure 6.2.

It is noted that the cross-entropy of the auxiliary hindsight classifier with the policy is near identical to the "cross"-entropy of the policy with itself. This suggests that the auxiliary hindsight classifier has learned almost everything it can from the available information.

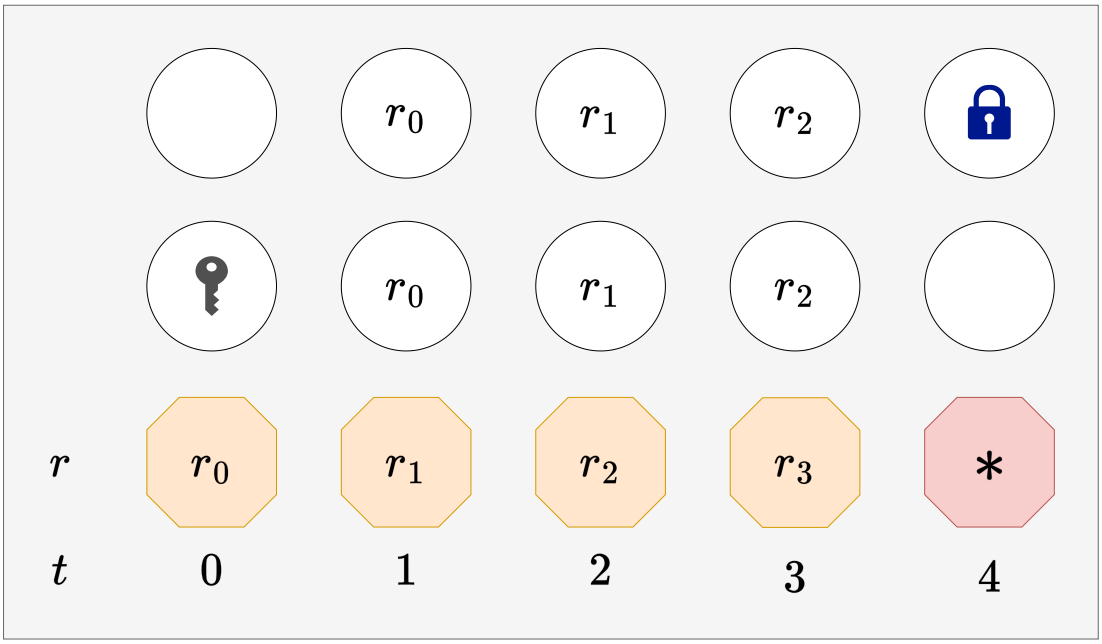

Figure 6.1: A schematic of our Linear Key-To-Door environment, with an episode-length of 5. Columns represent the individual timesteps, vertical pairs of circles represent the observations, and coloured hexagons represent the rewards. At each timestep the agent can choose from two actions. These correspond to interacting with the top or the bottom cell of the observation. At timestep 0 the agent may pick up the key by selecting the action corresponding to the cell the key (randomly) appears in. If the agent has picked up the key earlier, selecting the cell with the door in the last environment step leads to a reward of 10. In all other cases and timesteps the reward is randomly selected from $-1, 0$, and 1.

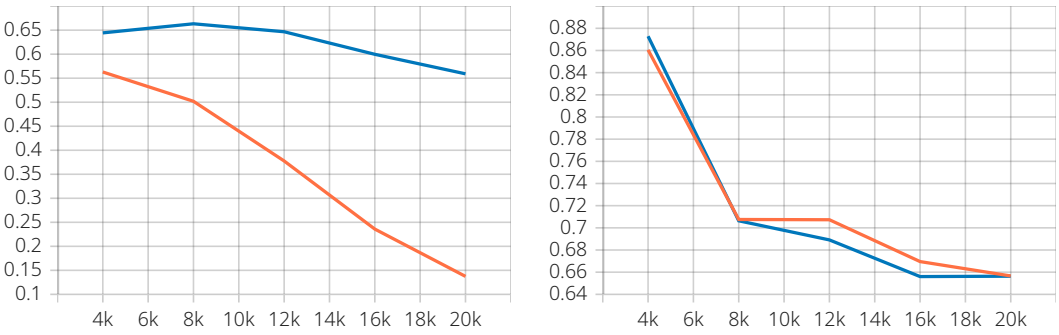

Figure 6.2: The cross-entropy losses of our primary (left) and auxiliary (right) hindsight classifiers on CartPole-v1 [1]. We note that the KL-divergence of the auxiliary hindsight classifier stayed below 0.05, indicating that the regularization of $\hat{\phi}_t$ was successful. The horizontal axis represents how many timesteps the agent has interacted with the environment. The blue curves correspond to COCOA (without backward RNN), the orange curves correspond to CF-COCOA (with backward RNN).

6.3 Evaluating the performance of our COCOA and CF-COCOA implementations on our Linear Key-To-Door environment

We find that due to their stability, the Rec-PPO baselines outperform both our COCOA and CF-COCOA implementations in the Linear Key-To-Door task (see Figures 6.3, 6.4, and 6.6). We do note that once their hindsight classifiers become sufficiently accurate, both the COCOA and the CF-COCOA agents show reasonable potential for rapid convergence, as even with our unstable implementations, under lucky seeds they occasionally do acquire higher sample efficiency than the baseline.

Additionally, the convergence of the hindsight classifiers may be expedited at the cost of extra computation time by increasing the number of epochs they are trained for during each iteration. Although such an approach can give fast results in a simple environment like ours, we expect it would lose efficacy in environments where the agent must have a basic level of competency in order to explore the parts of the state space most relevant to the task at hand. Thus we did not run extensive experiments to directly see the effects of varying the number of hindsight training epochs.

The COCOA and CF-COCOA agents learn to pick the key up as fast as, if not faster than, they learn to open the door, indicating that credit can successfully be assigned to actions over relatively long timescales, and with the presence of noise.

We ran experiments in both 10 and 20 timestep long instances of the Linear Key-To-Door environment, training each of the agents, and trying 5 distinct seeds for every combination of these parameters. Both the 10- and the 20-step experiments came back with similar results. All figures presented here thus correspond to experiments run in the 20 step long instance of the environment.

We note that as we vary the length of an episode, we often have to vary the hyperparameters of our COCOA and CF-COCOA algorithms in order to achieve as much stability in learning as possible. In particular, the coefficients of the entropy regularisation loss and the PG loss needed frequent adjustments between environment configurations.

We can see in Figure 6.5 that the primary failure mode of our COCOA and CF-COCOA implementation is that of an overly sharp action distribution induced by the policy, which leads to numerical issues when using the PyTorch library [2].

In the end we have found a set of hyperparameters that performed reasonably in both the 10 and 20 step Linear Key-To-Door instances. The results we present were acquired under these hyperparameters. As noted before, all hyperparameters and configurations for our experiments can be found in the supplementary materials.

As the rewards in our Linear Key-To-Door environment are purposely noisy, we present the fraction of times both the key and the door were chosen by our agents (see Figure 6.3 for COCOA and CF-COCOA), as well as the number of times the key was chosen regardless of whether or not the door was also chosen (see Figure 6.3 for COCOA and CF-COCOA). The results achieved by the Rec-PPO baseline agent are presented in Figure 6.6.

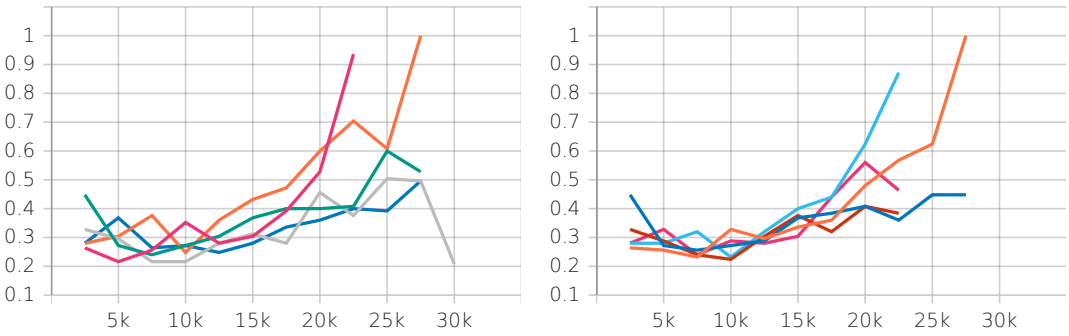

Figure 6.3: The average number of times both the key and the door were selected by the CF-COCOA agent (left), and by the COCOA agent (right). Each line represents the same experiment run under a distinct seed. The horizontal axis represents for how many timesteps the agent has interacted with the environment.

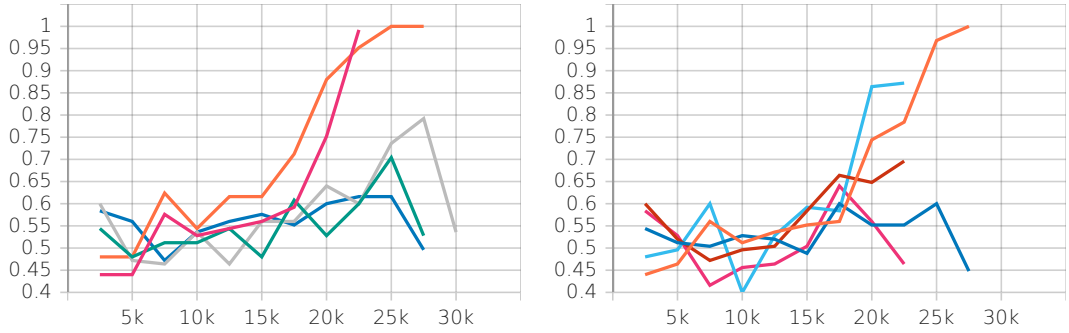

Figure 6.4: The average number of times the key was selected by the CF-COCOA agent (left), and by the COCOA agent (right). Each line represents the same experiment run under a distinct seed. The horizontal axis represents for how many timesteps the agent has interacted with the environment.

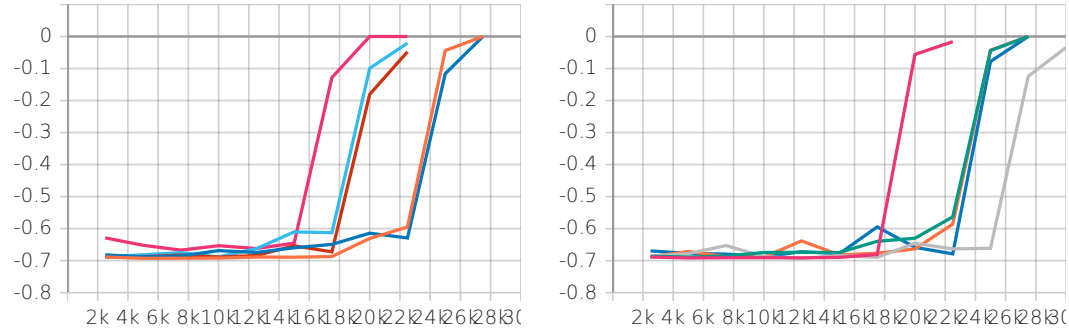

Figure 6.5: The entropy regularisation loss of the CF-COCOA agent (left), and of the COCOA agent (right). Each line represents the same experiment run under a distinct seed. The horizontal axis represents for how many timesteps the agent has interacted with the environment.

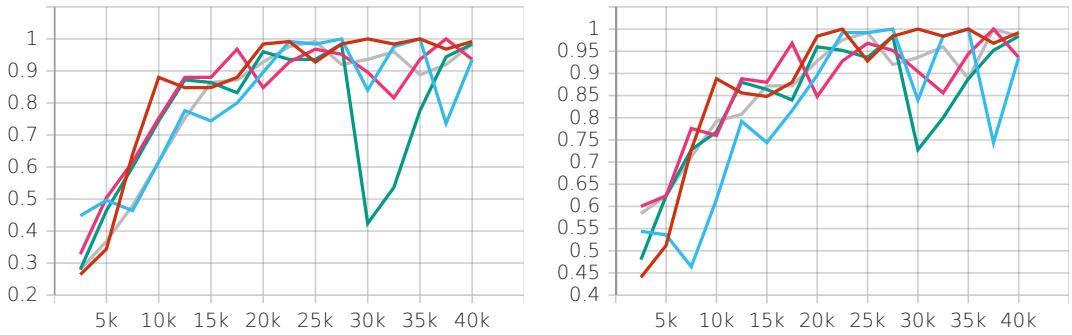

Figure 6.6: The average number of times both the key and the door were selected by the Rec-PPO agent (left), and the average number of times the key was selected by the Rec-PPO agent (right). Each line represents the same experiment run under a distinct seed. The horizontal axis represents for how many timesteps the agent has interacted with the environment.

Chapter 7

Limitations and Future Work

7.1 Limitations

We have already discussed the numerical instability of our implementations of COCOA and CF-COCOA in chapters 5 and 6. We discuss further limitations of our methods in this section.

7.1.1 Sampling trajectories to their end

Both learning and computing contribution coefficients requires that trajectories be sampled to their end. This poses two challenges.

In the case of environments where there may be large discrepancies between the lengths of trajectories it is possible that our experience buffer will be very memory inefficient.

In such environments updates at timesteps close to the end of the longer trajectories may also have significantly lower effective batch sizes. The data at these timesteps in the shorter trajectories will simply be padding. We calculate expected values (of the policy gradients for example) using Monte Carlo integration. Smaller batch sizes are known to increase the variance of Monte Carlo integration [7], thus reducing the quality of our estimates.

While significant, these limitations do not affect our algorithm's performance on the Linear Key-To-Door task, as all episodes are the same length in this environment.

7.1.2 Computational complexity of backward RNN

Although our backward RNN seems to be able to encode relevant information about the future of the trajectory, learning and computing these encodings using an LSTM can be rather computationally expensive. This effect is especially pronounced in the case of long trajectories. As suggested in the literature, using attention mechanisms, especially in the form of a Transformer encoder, may be a worthwhile direction for future research [11].

7.1.3 Large action spaces

As mentioned in [17] and [20], summing over all actions in large action spaces may be infeasible. In the extreme case of continuous actions it is in fact impossible. This currently limits our approach to action spaces of relatively small dimensionality. With some tradeoffs, this limitation can be removed if required. In [17] it is explained how independently sampling the policy distribution and averaging over the resulting samples provides an unbiased estimate of the sum over the action space. The tradeoff is that this introduces additional variance due to the sampling process.

7.2 Future Work

Our work aims to be a proof of concept, and thus we chose to forego use of some methods which may be important in practice. Some of these are outlined in this section.

7.2.1 Conditioning on state-encodings

As mentioned in section 2.4, we do not consider state-encodings as future-conditioning targets in the place of reward values. Although such encodings have been demonstrated to be beneficial [17], learning state-encodings that form useful future-conditioning targets introduces significant complexity. Thus, extending our methods to use such encodings is left for future work.

7.2.2 Multi-dimensional actions

We currently do not support environments with multi-dimensional action spaces. This is not a theoretical limitation however, and support for multi-dimension action spaces may be readily implemented in the future.

7.2.3 Z-forcing

Z-forcing, that is, forcing our encodings $\hat{\phi}_t$ to encode information about the latent state that we expect to be useful can significantly boost the performance of approaches based on inferring latent variables [29]. This is due to the fact that common mechanisms for learning these encodings often end up minimising the reliance of downstream models on $\hat{\phi}_t$ as a shortcut to higher performance (as at the early stages of learning $\hat{\phi}_t$ may be no more than random noise, and not the kind of noise it is trying to summarise).

State-forcing is one of the simplest Z-forcing methods, where we learn an auxiliary model to decode the forward state $\hat{s}_t$, or the observations o_t from the encoding $\hat{\phi}_t$, and backpropagate the relevant loss to the encoding model used to generate $\hat{\phi}_t$ [29]. State-forcing has been found to perform well on similar problems to ours, so is a promising approach for future work [29].

7.2.4 Stochastic modelling of latents

From the model-based RL literature it is known that deterministic models are frequently insufficient to model complex environments [9]. Although here we are not trying to model environment dynamics, it might nevertheless be an interesting direction for future research to see if stochastic embeddings $\hat{\phi}_t$ (or even stochastic $\hat{s}_t$) may provide any benefits to CF-COCOA.

Chapter 8

Conclusion

In this project, we started by clarifying the connections between Reinforcement Learning, causal theory, and hindsight.

Afterwards, based on suggestions for future work from [17], we implemented a Policy Gradient estimator combining the outcome-conditioned contribution coefficients of COCOA [17] with hindsight-based de-noising by the approach of CCA [16]. We refer to this combination algorithm as CF-COCOA.

We also designed and implemented a variant of Linear Key-To-Door environments, with the aim of presenting a relatively difficult long time-horizon credit assignment challenge at a small scale, enabling experimentation at low computational cost. To our knowledge this variant of Linear Key-To-Door has not been used before.

The implemented environment was then used to compare the performance of CF-COCOA to its ablation where we remove our attempt at de-noising (leaving us with a version of vanilla COCOA [17]), as well as to a good-performing baseline algorithm. The experiments performed were used to identify a crucial failure-mode of our implementations of COCOA and CF-COCOA. We also identify potential strengths of COCOA and CF-COCOA compared to the baseline of Rec-PPO.

Bibliography

[1] Gymnasium Documentation, . URL https://gymnasium.farama.org/environments/classic_control/cart_pole.html.

[2] PyTorch documentation — PyTorch 2.2 documentation, . URL https://pytorch.org/docs/stable/index.html.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay, February 2018. URL http://arxiv.org/abs/1707.01495. arXiv:1707.01495 [cs].

[4] Lars Buesing, Theophane Weber, Yori Zwols, Sebastien Racaniere, Arthur Guez, Jean-Baptiste Lespiau, and Nicolas Heess. Woulda, Coulda, Shoulda: Counterfactually-Guided Policy Search, November 2018. URL http://arxiv.org/abs/1811.06272. arXiv:1811.06272 [cs, stat].

[5] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling, June 2021. URL http://arxiv.org/abs/2106.01345. arXiv:2106.01345 [cs].

[6] EclecticSheep, Davide Angioni, Federico Belotti, Refik Can Malli, and Michele Milesi. SheepRL, 2023. URL https://github.com/Eclectic-Sheep/sheeprl/. original-date: 2023-05-16T19:33:30Z.

[7] Geof H Givens. Computational Statistics.

[8] Paul Glasserman. Stochastic Monotonicity and Conditional Monte Carlo for Likelihood Ratios. *Advances in Applied Probability*, 25(1):103–115, 1993. ISSN 0001-8678. doi: 10.2307/1427498. URL https://www.jstor.org/stable/1427498. Publisher: Applied Probability Trust.

[9] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning Latent Dynamics for Planning from Pixels, June 2019. URL http://arxiv.org/abs/1811.04551. arXiv:1811.04551 [cs, stat].

[10] Dongqi Han, Tadashi Kozuno, Xufang Luo, Zhaoyun Chen, Kenji Doya, Yuqing Yang, and Dongsheng Li. VARIATIONAL ORACLE GUIDING FOR REINFORCE- MENT LEARNING. 2022.

[11] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Greg Wayne, Satinder Singh, Doina Precup, and Remi Munos. Hindsight Credit Assignment, December 2019. URL http://arxiv.org/abs/1912.02503. arXiv:1912.02503 [cs, stat].

[12] Guido W. Imbens and Donald B. Rubin. Rubin Causal Model. In Steven N. Durlauf and Lawrence E. Blume, editors, *Microeconometrics*, pages 229–241. Palgrave Macmillan UK, London, 2010. ISBN 978-0-230-28081-6. doi: 10.1057/9780230280816_28. URL https://doi.org/10.1057/9780230280816_28.

[13] Michael Janner, Qiyang Li, and Sergey Levine. Offline Reinforcement Learning as One Big Sequence Modeling Problem, November 2021. URL http://arxiv.org/abs/2106.02039. arXiv:2106.02039 [cs].

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

[15] T. Mesnard, Wenqi Chen, Alaa Saade, Yunhao Tang, Mark Rowland, T. Weber, Clare Lyle, A. Gruslys, M. Vaĺko, Will Dabney, Georg Ostrovski, É Moulines, and R. Munos. Quantile Credit Assignment. 2023. URL https://www.semanticscholar.org/paper/Quantile-Credit-Assignment-Mesnard-Chen/73167084e0d103e6433a4558cc0fea247e899c41.

[16] Thomas Mesnard, Théophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Tom Stepleton, Nicolas Heess, Arthur Guez, Éric Moulines, Marcus Hutter, Lars Buesing, and Rémi Munos. Counterfactual Credit Assignment in Model-Free Reinforcement Learning, December 2021. URL http://arxiv.org/abs/2011.09464. arXiv:2011.09464 [cs].

[17] Alexander Meulemans, Simon Schug, Seijin Kobayashi, Nathaniel Daw, and Gregory Wayne. Would I have gotten that reward? Long-term credit assignment by counterfactual contribution analysis, October 2023. URL http://arxiv.org/abs/2306.16803. arXiv:2306.16803 [cs, stat].

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14236. URL https://www.nature.com/articles/nature14236.

[19] Chris Nota, Philip Thomas, and Bruno C. Da Silva. Posterior Value Functions: Hindsight Baselines for Policy Gradient Methods. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8238–8247. PMLR, July 2021. URL https://proceedings.mlr.press/v139/nota21a.html. ISSN: 2640-3498.

[20] Hsiao-Ru Pan, Nico Gürtler, Alexander Neitz, and Bernhard Schölkopf. Direct

Advantage Estimation, February 2023. URL http://arxiv.org/abs/2109.06093. arXiv:2109.06093 [cs].

[21] Paavo Parmas and Masashi Sugiyama. A unified view of likelihood ratio and reparameterization gradients, May 2021. URL http://arxiv.org/abs/2105.14900. arXiv:2105.14900 [cs, stat].

[22] Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Generalization, Mayhems and Limits in Recurrent Proximal Policy Optimization, May 2022. URL http://arxiv.org/abs/2205.11104. arXiv:2205.11104 [cs].

[23] Paulo Rauber, Avinash Ummadisingu, Filipe Mutz, and Juergen Schmidhuber. Hindsight policy gradients, February 2019. URL http://arxiv.org/abs/1711.06006. arXiv:1711.06006 [cs].

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. URL http://arxiv.org/abs/1707.06347. arXiv:1707.06347 [cs].

[25] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation, October 2018. URL http://arxiv.org/abs/1506.02438. arXiv:1506.02438 [cs].

[26] Richard S. Sutton and Andrew Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts London, England, second edition edition, 2020. ISBN 978-0-262-03924-6.

[27] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.

[28] Akash Velu, Skanda Vaidyanath, and Dilip Arumugam. Hindsight-DICE: Stable Credit Assignment for Deep Reinforcement Learning, August 2023. URL http://arxiv.org/abs/2307.11897. arXiv:2307.11897 [cs].

[29] David Venuto, Elaine Lau, Doina Precup, and Ofir Nachum. Policy Gradients Incorporating the Future, August 2021. URL http://arxiv.org/abs/2108.02096. arXiv:2108.02096 [cs].

[30] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. 1992.

[31] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online Decision Transformer, July 2022. URL http://arxiv.org/abs/2202.05607. arXiv:2202.05607 [cs].

[32] K.J Åström. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205,

February 1965. ISSN 0022247X. doi: 10.1016/0022-247X(65)90154-X. URL https://linkinghub.elsevier.com/retrieve/pii/0022247X6590154X.