Structural Causal Models for Reinforcement Learning

Vincent Luczkow, School of Computer Science

McGill University, Montreal

December, 2020

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Computer Science

Abstract

In this thesis we explore the applications of causal reasoning to reinforcement learning, specifically the use of causal models in model-based reinforcement learning. We examine possible approaches to this problem, and show that traditional approaches cannot learn minimal causal models of many environments. To avoid this problem we introduce an approach based on learning causal models at the time step level rather than the environment level. Finally, we implement and test this approach on some simple environments.

Abrégé

Dans cette thèse, nous explorons les applications du raisonnement causal à l'apprentissage par renforcement, en particulier l'utilisation des modèles causaux dans l'apprentissage par renforcement avec modèle. Nous examinons les méthodes courantes et montrons les limites d'apprentissage causaux à travers plusieurs environnements simulé. Pour résoudre ces défis, nous introduisons une algorithme qui produit un modèle causal adaptif à chaque pas de temps. Notre contribution est validé dans plusieurs environnements simple.

Acknowledgements

I would like to thank my supervisors Prakash Panangaden and Doina Precup for their support, feedback, and insightful comments throughout my research.

I would also like to thank Jordan Schneider and Jonathan Lebensold for many helpful conversations on the topic of my research.

Table of Contents

Abstract i

Abrégé ii

Acknowledgements iii

List of Figures vii

1 Introduction **1**

2 Background **3**

2.1 Reinforcement Learning 3

2.1.1 Model-Based Reinforcement Learning 4

2.2 Structural Causal Models 5

2.2.1 Three "Levels" of Causal Models 7

2.2.2 Graphs and Distributions 10

2.2.3 Causal Discovery 12

2.2.4 Time-Series Structural Causal Models 14

2.3 Previous Work 16

3 Concept **20**

3.1 Representing General MDPs as TSCMs 21

3.1.1 Modeling Actions 22

3.2 Vector Valued States 24

3.3 The Time-Variance Problem 25

3.4 Local Representations 26

3.5 Interventions . . . 29
3.5.1 Choice of Action as Intervention . . . 29
3.5.2 Actions and Local Representations . . . 30
3.5.3 External Interventions . . . 31
3.6 Counterfactuals . . . 32

4 Implementation and Experiments **34**
4.1 Local Representations as Neural Networks . . . 34
4.2 Environments . . . 35
4.3 Architecture . . . 36
4.3.1 Weighted Architecture . . . 37
4.3.2 Sampling Architecture . . . 38
4.4 Results . . . 38
4.4.1 Core Architecture . . . 38
4.4.2 Training with the True Graph . . . 41
4.4.3 Multi-Typed . . . 42
4.5 Alternate Training Algorithms . . . 43
4.5.1 Buffered Errors . . . 44
4.5.2 Iterated Learning . . . 45

5 Conclusion and Future Work **49**
5.1 Future Work . . . 50

List of Figures

3.1 Causal representation of a general MDP with a fixed policy. 23

3.2 Causal representation of a general MDP with an updating policy. The arrow from the rectangle to π_{t+1} represents the fact that π_{t+1} depends on everything in the rectangle. 23

3.3 A TSCM representation of an MDP with vector-valued states. For the sake of clarity we have left out some of the edges from state to action and reward. 24

3.4 An environment where TSCM discovery would return a fully connected graph, but each individual time step is fairly sparse. 26

3.5 Actions as interventions 29

4.1 Visualization of a single state of the hard-sphere environment. The lines are velocity vectors. 36

4.2 Left: state prediction loss and collision loss of the weighted architecture on the hard sphere gas environment. Right: The same for the sampling architecture. 39

4.3 Gradient norms at each batch, for each layer of g. 40

4.4 Training comparison between a simple fully connected model and the two causal models. 41

4.5 Performance of the state predictor when the true graph is given as input . . 42

4.6 Test loss comparison 42

4.7 Learning rate comparison. Each value is the mean training loss across the last 5 batches. 43

4.8 Test loss comparison 43
4.9 Performance of the weighted predictor with buffered errors. From left to right $p = 0.25, 0.5$, and 0.75. 44
4.10 Performance of the weighted predictor with buffered errors. From left to right $p = 0.25, 0.5$, and 0.75. 45
4.11 Performance of the weighted predictor with iterated learning. From left to right the batch sizes are 10 and 5, 15 and 1, and 30 and 1 (g batches and f batches respectively). 46
4.12 Performance of the weighted predictor with iterated learning. From left to right the batch sizes are 10 and 5, 15 and 1, and 30 and 1 (g batches and f batches respectively). 47
4.13 Norm of the gradient for each layer. Training signal is fairly rapidly lost. . . 47

Chapter 1

Introduction

There has been a flurry of work applying the theory of causality to machine learning, beginning with [1]. We are interested in the application of causality to reinforcement learning, and in particular to the problem of learning causal models for use in model-based reinforcement learning.

Model-based reinforcement learning involves learning an explicit function (the model) that predicts the next state. This has a great deal of intuitive appeal as an approach, but has seen relatively limited use due to the difficulty of learning a model efficiently.

Causal models (which we will formally define in 2.2) are in some sense the "smallest" possible models that still capture the entire dynamics of system. This suggests that they may be useful for building models in reinforcement learning, as they limit the complexity of the relationships that must be learned.

Furthermore, causal models contain information that non-causal models simply lack (again, see [1]). Roughly speaking, causal models can answer "what if" questions more effectively than other models. This has an obvious application to the core reinforcement learning problem of choosing among many possible actions, but might also prove useful for other tasks, e.g. credit assignment [2].

Lastly, causal models should be more "interpretable", in the sense that the information about which objects the model considers to be directly related would be transparently available.

Contributions

We examine a number of questions surrounding the "naive" application of concepts from causality to reinforcement learning, and show that in most cases these naive applications simply are not useful. Specifically:

We give an argument that standard causal learning algorithms and concepts cannot be directly adapted to the model-based reinforcement learning setting, and in particular cannot learn models that genuinely represent the underlying causal relationships.

To solve this problem, we introduce a novel algorithm for learning structural causal models that can be reasonably used in many reinforcement learning settings.

Finally, we implement a preliminary version of the learning algorithm, and test it on simple environments.

Chapter 2

Background

Before moving on to the main section, we give a very brief overview of model-based reinforcement learning.

2.1 Reinforcement Learning

(This section is drawn from [3]).

In reinforcement learning, we are interested in building agents that take actions in some environment, with the goal of maximizing a real-valued reward. We can formalize this with a Markov decision process:

Definition 2.1.1 (Markov Decision Process (MDP))**.**
A Markov decision process is a tuple (S, A, T, R, γ), where S is a set of states, A is a set of actions, $T : S \times A \to \mathcal{D}(S)$ is a function from state-action pairs to distributions over S, $R : S \times A \times S \to \mathbb{R}$ is the *reward* function, and $\gamma \in [0, 1]$ is the *discount factor*.

At each state s_t, an action is chosen by a *policy* $\pi : S \to A$, and then the next state is determined by the transition function:

$$s_{t+1} \sim T(s_t, \pi(s_t))$$

The reward for that time step is then determined:

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

The goal of much of reinforcement learning is to find a policy π that maximizes expected discounted reward:

$$\text{argmax}_\pi E\left[\sum_t \gamma^t r_t\right] = \text{argmax}_\pi E\left[\sum_t \gamma^t R(s_t, \pi(s_t), s_{t+1})\right]$$

MDPs can be extended to *partially observable Markov decision processes*, which serve to model environments where much of the state is hidden at any given time (e.g. robotic environments where the robot can see only a small fraction of the environment):

Definition 2.1.2 (Partially observable MDP (POMDP))**.**
A POMDP is an MDP with an additional components Ω and $O : S \to \Omega$. Here Ω is the space of possible observations. Rather than receiving the state as input, policies are now functions $\pi : \Omega \to A$. At each time step, an observation $o_t = O(s_t)$ is produced, and $a_t = \pi(o_t)$ [4].

We will not perform experimental work on POMDPs in this thesis, but they are important for understanding the use of counterfactuals in reinforcement learning (see 2.3 and 3.6).

2.1.1 Model-Based Reinforcement Learning

In model-based reinforcement learning (MBRL) paradigm, an explicit function (the "model") $f : S \times A \to S \times R$ from the current state and action to the next state and reward is learned [5]. This is in contrast to model-free learning, where a function determining the best action is learned directly.

This model can then be used as an input to a variety of *planning* algorithms, which produce a policy. Many such algorithms use the model to simulate sequences of possi-

ble states and actions (*rollouts*) to pick actions likely to lead to the highest reward. As an example, the Monte-Carlo tree search algorithm famously had significant success in application to the game of Go [6][1].

Model-based methods see less use in practice than model-free methods, although recent work has achieved impressive results [7]. This is mostly due to the difficulty of efficiently learning a model.

2.2 Structural Causal Models

For the purposes of this thesis, we focus exclusively on a particular technical definition of causality, namely the theory of *structural causal models* (SCMs) popularized by Judea Pearl [8]. We choose this framework not because of any particular devotion to its philosophical foundations, but because it lends itself well to practical implementations with neural networks.

In particular, the SCM framework represents causal relationships between two variables A and B as a *functional* relationship between them, which can in theory be represented by a neural network. Formally:

Definition 2.2.1 (Structural Causal Model)**.**
A structural causal model C is a tuple $(V, D, F, \mathcal{N})$, where

- V is a set of real random variables indexed by some set I
- D is a directed acyclic graph on V
- F is a set of functions also indexed by I
- $\mathcal{N}$ is a set of real random variables indexed by I

F is called the set of *update functions*, and $\mathcal{N}$ the set of *noise terms*.

[1] Go and other board games are notable in that a model is already known (the rules of the game), and so the very difficult model-learning step can be skipped.

Furthermore $\forall i \in I$:

$$v_i = f_i(\text{parents}(v_i), \nu_i) \tag{2.1}$$

where $v_i \in V$, $f_i \in F$, and $\nu_i \in \mathcal{N}$.

Note that the source nodes of D are just functions of noise terms, so in general we identify the source nodes with their corresponding noise terms and make their update function the identity function. In practice we will drop the indices when context allows it. We will also frequently write our variables as A, B, C, etc and write f_A, ν_A etc as the corresponding functions and noise terms.

The intuition for this definition is this: we are first defining (or restricting) "causation" to mean "is a function of". Given an SCM with DAG $A \to B$ where $f_B(A) = 2A$, this is expressing that B is caused by A.

The fact that each variable is a function of *only* its immediate parents encodes an independence assumption: if we condition a variable on its immediate causes, all other causes are screened off (in the sense of conditional independence).

The noise terms allow SCMs to capture underlying randomness: they allow a variable to be defined as a distribution determined by its parents, rather than a deterministic value.

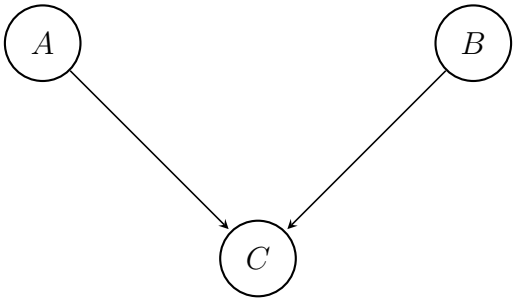
While SCMs form a very large class of models, it is certainly not adequate as a *full* formalization of causality in the physical universe (it is not, for instance, capable of capturing quantum systems [9]).

The DAG restriction is also significant, as it makes it impossible to represent systems with genuine mutual dependence between variables. Many such systems can be handled by treating the data as a time series (which we will elaborate on in 2.2.4, but if time series data is unavailable then a system with feedback/recurrence cannot be modeled (e.g. coupled harmonic oscillators).

However, both of these restrictions will end up being irrelevant to the particular application we are interested in: using SCMs to model Markov decision processes.

Example 2.2.1.1 (Linear Gaussian SCM)**.** A very common form of SCM is a *linear Gaussian SCM*. Here we restrict all functions f to be linear, and all noise terms ν to be drawn from Gaussian distributions (not necessarily the same Gaussian distribution)

Consider the system $A \sim \mathcal{N}(0,1)$, $B \sim \mathcal{N}(0,2)$, $C = A+2B$. This would be represented with the diagram:



It is important to note that unlike many diagrams in mathematics, arrows *do not* correspond with functions. Rather the entire set of edges into a node corresponds to a single function.

Example 2.2.1.2 (Deterministic SCM)**.** A system of two variables y and x and a function f such that $y = f(x)$ can be represented as an SCM by extending f to f', which accepts a noise term but ignores it: $f'(x, \nu) = f(x)$.

This construction can be extended to any number of variables and functions, so long as we do not violate the DAG condition.

2.2.1 Three "Levels" of Causal Models

We now introduce three different properties of causal models, which allow us to define exactly what is *causal* about them.

Definition 2.2.2 (Observational Distribution)**.**
The observational distribution of an SCM C is the joint distribution of its random variables V.

The observational distribution is also called the distribution entailed by C. It is written P_V^C.

Definition 2.2.3 (Observation)**.**
An observation is a sample from the observational distribution.

Intuitively, the observational level does not contain much causal information. Consider a deterministic model

$$A \xrightarrow{f} B \xrightarrow{g} C$$

where f and g are invertible. At the observational level, this is exactly equivalent to

$$C \xrightarrow{g^{-1}} B \xrightarrow{f^{-1}} A$$

In the next section we will formalize this with the notion of "Markov equivalence", but the key point here is that merely having an observational distribution is not always enough to distinguish the *direction* of causality.

Interventions

Definition 2.2.4 (Intervention)**.**
An intervention is a choice of variables $\mathcal{I} \subset V$, and a set of values for each of those variables. We will abuse notation and use $\mathcal{I}$ to refer to both the variables and the values.

Definition 2.2.5 (Interventional Distribution)**.**
An intervention $\mathcal{I}$ entails a new distribution for V, by creating a new SCM from C denoted $C; do(\mathcal{I})$.

This new model is created by:

- Removing all edges into $\mathcal{I}$
- Setting each variable v in $\mathcal{I}$ to the corresponding value in $\mathcal{I}$

We denote the entire interventional distribution as $P_V^{C;do\mathcal{I}}$

An intervention is meant to capture the concept of modifying the system "from the outside", by directly setting the values of some variables. In the social sciences context interventions chiefly represent randomized experiments.

Interventions are important from the causal perspective because they allow us to distinguish models with identical observational distributions. In the previous example, performing an intervention on B would remove the edge from A and from C respectively. In the former case B would be independent from A, while in the latter it would be independent from C. They also allow us to ask a certain kind of "what if" question, namely: what would the world look like if a particular thing *always* happened.

Finally, there is a notion of interventional equivalence:

Definition 2.2.6 (Interventional Equivalence)**.**
Two SCMs C_1 and C_2 are *interventionally equivalent* if their interventional distributions are identical under *all* interventions.

and the very important related theorem:

Theorem 2.2.1. *If C_1 and C_2 have the same observational distributions, this does* not *imply that they are interventionally equivalent.*

This last theorem is one of the main reasons why causal models are "more expressive" than simple probabilistic models.

Counterfactuals

Counterfactuals are the formalization of *after the fact* "what if" questions. They take an observation, perform an intervention ("what if this had happened instead"), and hold the noise terms constant ("but nothing else changed"). This distinguishes them from simple interventions, which ask about the *distribution* of outcomes after an intervention, not the change for a *specific* outcome. More formally:

Definition 2.2.7 (Counterfactual)**.**
Let $x \sim P_V^C$ with noise terms $N \sim \mathcal{N}$. Let I be an intervention on C. The corresponding counterfactual, written $P_V^{C|V=x;do(I)}$ is a sample from the interventional distribution $P_V^{C|do(I)}$ where the noise terms are set to N.

This definition can be generalized by holding only some noise variables fixed (which makes particular sense in scenarios involving time), or by allowing the intervention to be distributional.

As with interventions, equivalence at the lower layer does not imply counterfactual equivalence:

Theorem 2.2.2. *If C_1 and C_2 are interventionally equivalent, this does not imply that they have the same value for all counterfactuals.*

See for instance Example 6.19 in [10].

2.2.2 Graphs and Distributions

We now review some relationships between joint distributions on random variables and graphs on those random variables. While the definitions will all be phrased in these terms, they apply readily to SCMs through application to the observational distribution and the DAG of the SCM.

Definition 2.2.8 (Path blocked by a set)**.**
In a DAG D, a path $p = (v_1, \ldots, v_n)$ between nodes v_1 and v_n is *blocked by a set* S if there is a node v_k that satisfies one of the following:

- $v_k \in S$, and one of the following holds:

$$v_{k-1} \to v_k \to v_{k+1} \in p$$

$$v_{k-1} \leftarrow v_k \leftarrow v_{k+1} \in p$$

$$v_{k-1} \leftarrow v_k \to v_{k+1} \in p$$

- Neither v_k nor any of its descendants are in S, and:

$$v_{k-1} \to v_k \leftarrow v_{k+1} \in p$$

Definition 2.2.9 (d-separation)**.**
Two nodes v_1 and v_n are d-separated by a set S if *every* path between v_1 and v_n is blocked by S.

This is denoted $v_1 \perp\!\!\!\perp_D v_n | S$.

d-separation is essentially "graphical independence". The next two concepts define an equivalence between graphical independence and probabilistic independence.

Definition 2.2.10 (Global Markov property)**.**
A distribution P satisfies the global Markov property with respect to a DAG D if:

$$A \perp\!\!\!\perp_D B | C \implies A \perp\!\!\!\perp B | C$$

We say a distribution P is *Markovian* with respect to a graph D if it satisfies the global Markov property with respect to D. The set of distributions that are Markovian wrt D is written $M(D)$. Note that there are several alternatives to the global Markov property, which are usually equivalent to it [11]. For our purposes the global Markov property will suffice.

Theorem 2.2.3. *The observational distribution of an SCM is Markovian with respect to the graph of the SCM.*

The global Markov property is particularly important when learning an SCM from samples, as it provides us with constraints on the underlying graph.

Definition 2.2.11 (Faithfulness)**.**

A distribution P over random variables V is faithful to a graph D on V if:

$$A \perp\!\!\!\perp B|C \implies A \perp\!\!\!\perp_D B|C$$

Faithfulness is essentially the converse of the global Markov property. It is *not* guaranteed to hold for all SCMs. However models that do not satisfy this are fairly pathological[2], so it is common to assume faithfulness.

Definition 2.2.12 (Causal minimality)**.**

A distribution P is minimal with respect to G if it is Markovian with respect of G, but not with respect to any (proper) subgraph of G.

Causal minimality guarantees that there are no unnecessary edges. More formally:

Theorem 2.2.4. *P is minimal with respect to G iff there is no $v_i \in G$ such that*

$$v_i \perp\!\!\!\perp v_p|\mathit{parents}(v_i) - \{v_p\}$$

where v_p is one of the parents of v_i.

As an obvious corollary, a causally minimal SCM cannot have any functions f_i which are almost everywhere constant in one of the inputs. This is very important from a modeling perspective, as a causally minimal SCM representing an MDP is essentially the simplest possible SCM that fully captures the MDP[3][4].

2.2.3 Causal Discovery

One of the core problems in causality is that of *causal discovery*[5]. Given a dataset sampled from an unknown SCM C:

[2]Exceptions occur essentially when the causal effect of A on B happens to be cancelled out exactly, leaving them causally related but probabilistically independent.

[3]Under some additional conditions we can even show that this is unique.

[4]Of course if we are willing to merely *approximate* the MDP this is no longer the case.

[5]Also called "causal identification".

- Can we determine *a* graph that could be the causal graph of C?
- Is that graph unique?

The answer to the first question is yes (given some conditions), and the answer to the second is: almost if we only have observational data, and yes if we have sufficient interventional data.

Specifically, in the observational regime the graph can be determined up to its *Markov equivalence class*(which we will define shortly). All graphs in Markov equivalence class have the same undirected structure, but the direction of each edge can not always be determined.

Including interventional data allows us to determine the direction of the edges, essentially because the interventional distributions involve removing arrow *into* a variable but not coming from it, breaking the symmetry on the direction of the arrow.

The rest of this section will cover the core concepts of causal discovery, finishing with the results for observational and interventional causal discovery.

Discovery from Observations

Definition 2.2.13 (Markov equivalence)**.**
Two graphs D and D' are *Markov equivalent* if $M(D) = M(D')$. This is an equivalence relation, and the equivalence class is the *Markov equivalence class*.

We care about Markov equivalence classes because they are precisely what can be learned from purely observational data.

Theorem 2.2.5 (Markov equivalence and d-separation)**.** *Two graphs D and D' are Markov equivalent iff they have the same set of d-separations.*

Definition 2.2.14 (v-structure)**.**
A v-structure in a graph D is a subgraph $A \to C \leftarrow B$, where there is no edge between A and B in D.

This is sometimes called an "immorality".

Theorem 2.2.6 (Characterization of Markov equivalence classes)**.** *D and D' are Markov equivalent iff they are equal as undirected graphs, and they have the same v-structures. [12]*

Theorem 2.2.7 (Discovery from observations)**.** *Let C be an SCM with graph D, such that P_C is faithful with respect to D. Then given all the conditional independence relationships between variables in P_C, the Markov equivalence class of D can be recovered.*

Proof. This essentially follows from the faithfulness constraint. There are a variety of algorithms that perform the construction, e.g. [13] or [14] □

Of course this leaves the problem of actually acquiring the conditional independence relationships. There is no fully general conditional independence test [15], and so in practice assumptions must be made about the underlying distributions.

2.2.4 Time-Series Structural Causal Models

To model time-series data with SCMs, we extend them to *time-series structural causal models*. We will not ultimately use these to model MDPs, but they are an important stepping stone.

Definition 2.2.15 (Time-series structural causal model (TSCM))**.**
A TSCM is an SCM whose set of random variables is indexed by the index set I and by time indices $t \geq 0$ (the time indices). The set of t indices may not be finite.

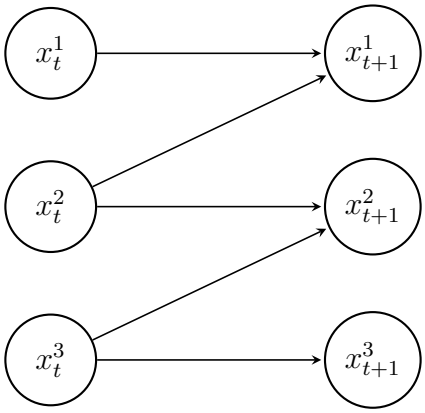
The TSCM meets the following additional constraints:

- For all edges $(v_{i,t}, v_{j,t'})$, $t \leq t'$. In other words, there are no arrows backwards in time.
- If there is an edge $(v_{i,t}, v_{j,t'})$, then there are also edges $(v_{i,t+k}, v_{j,t'+k})$ for all k that result in valid time indices.
- $f_{i,t} = f_{i,t'}$ for all valid time indices t and t'.

The equality constraints give a TSCM a repeating structure. When diagramming a TSCM, we draw only this repeating structure. As an example, consider a time series on 3 variables defined by:

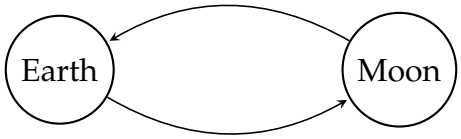
$$x_1^1, x_1^2, x_1^3 \sim \mathcal{N}(0, \sigma^2)$$
$$x_{t+1}^1 = x_t^1 + x_t^2$$
$$x_{t+1}^2 = x_t^2 * x_t^3$$
$$x_{t+1}^3 = e^{x_t^3}$$

The corresponding TSCM diagram is:

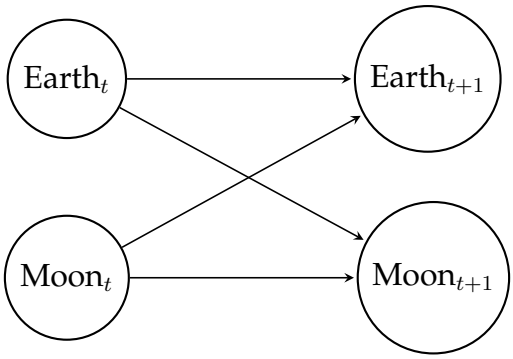


TSCMs allow for modeling many systems that would otherwise not be acyclic, since they separate a variable over time steps while preserving most of that variable's identity.

As an example, we could not model the orbits of the Earth and the Moon with a simple SCM, since we would end up with:



We can easily model it with a TSCM, however:



Causal Discovery for TSCMs

Identification for TSCMs is generally much easier (since the direction of most edges is already known).

Theorem 2.2.8 (Identification of TSCMs)**.** *If a TSCM does not contain any edges* $(v_{i,t}, v_{j,t})$*, then it is the only member of its Markov equivalence class.*

For a proof, see [16]. The edge restriction here is that there are no "instantaneous" edges: no edges between variables at the same time step. Since those edges do not have a direction enforced by time, the problem of directing the instantaneous edges reduces to normal causal discovery.

2.3 Previous Work

There has been a great deal of work applying deep learning to standard problems in causality, and to a lesser extent on using causal tools in deep learning.

Several authors have applied deep learning to score-based causal discovery, and have overall achieved promising results.

One approach to causal discovery is to treat it as an optimization problem over the space of DAGs. Some function is used to score a given DAG on how well it performs as a possible causal structure for a dataset, hence this approach is termed "score-based". Traditional score-based methods (e.g. [17]) are combinatorial optimization problems minimizing the

score subject to a DAG constraint on the graph.

NOTEARS [18] reformulates this combinatorial optimization problem as a continuous optimization problem. They develop a continuous criterion for acyclicity, specifically:

$$h(W) := \mathrm{tr}(e^{W \circ W}) - d$$

where $W \in \mathbb{R}^{d \times d}$ is the matrix, and $\circ$ is the Hadamard product. When $h(W) = 0$, W is acyclic. Using this, they give the continuous optimization problem:

$$\begin{aligned} &\min_W \ \mathrm{score}(W) \\ &\text{s.t. } h(W) = 0 \end{aligned}$$

which they solve with an augmented Lagrangian method.

DAG-GNN [19] builds off this work, minimizing the ELBO between the data distribution and the distribution learned by a graph based VAE. The use of a VAE allows them to model more complex distributions, and the graph-based architecture means the VAE satisfies the constraints imposed by W.

Gradient-Based Neural DAG Learning (GraN-DAG) [20] takes a similar approach, however they optimize MLE instead of ELBO, and the graph that is required to be acyclic is the connectivity graph of the actual neural network, i.e. the graph whose nodes are the input and output variables, and an edge exists between A and B if the A output is not constant in the B input.

Causal Discovery with Reinforcement Learning [21] takes a third approach, using reinforcement learning (an actor-critic algorithm, specifically) to optimize the Bayesian information criterion [22] (a common score for score-based causal discovery) combined with the acyclicity constraint from NOTEARS.

Overall, DAG-GNN outperforms NOTEARS on large causal networks, and the latter two papers generally outperform both of them on their selected problems. It should be noted, however, that comparisons between causal discovery algorithms are somewhat difficult.

There isn't a standard set of benchmark problems, and the assumptions being made can drastically change the problem (e.g. a model with Gaussian noise might be unidentifiable, while the same model with non-Gaussian noise would not be [10]). How each algorithm performs might depend heavily on the particular models under consideration, so firm conclusions are hard to draw.

Learning neural causal models from unknown interventions [23] works on the slightly different problem of causal discovery with interventional data. This problem is in general easier, although it does work with *unknown* interventions. They first learn a neural model of a causal system, then apply interventions to the system. A few samples are drawn from the interventional distribution, which are used to score the sampled graphs and update the graph predictor. This significantly outperforms the standard methods they compare against, frequently recovering the exact causal graph.

Conversely, there has been some work applying causality to machine learning, including to reinforcement learning.

The most important work for our purposes is *Woulda, Coulda, Shoulda: Counterfactually-Guided Policy Search* [24], as it provides a motivation for the use of specifically causal models in reinforcement learning. They show that actions can be evaluated as *counterfactual* rollouts, and proves that these are better estimators than naive rollouts in the partially observable case. It is important to note that they assume a known causal model of the environment, and do not learn one from scratch.

On a different track, [25] uses techniques from causal inference to improve generalization across environments for RL agents. Given a set of environments that share some underlying causal dynamics, they learn a representation of the state space which generalizes well across environments using the invariant causal prediction algorithm [26].

Relatedly, [27] draws a connection between transfer learning in contextual bandits and causal inference[6]. They show that the transfer learning problem is a causal inference prob-

[6]The prediction of an effect given some data, distinct from causal discovery which is our main focus.

lem (assuming some causal conditions on the bandits are met), and that information can be shared even when the causal effect is not fully identifiable. They go on to construct an upper confidence bound (UCB) method that makes use of this causal information, which has better asymptotic regret (in some cases) than standard UCB methods. They also experimentally show that their method outperforms standard bandit algorithms. Overall it's a promising result, even if it is restricted to contextual bandits and not full MDPs.

While not focused on causality, there has a fair amount of work on applications of deep learning to physics simulation, especially through graph neural networks. Our experiments are all on physical environments, and so this work is relevant.
A series of papers by Battaglia et. al. ([28], [29], [30]) use graph networks for simulating physical environments, and more generally argue for the use of graph networks to encode relationships in data. [28] focuses on relatively simple physical environments (e.g. bouncing balls), while [29] extends the work to more complex environments (in particular to MuJoCo environments, which are common environments in RL), and both achieve quite good results. It is notable for our purposes that both take the relationship graph as an input, rather than learning it from scratch.
Neural Relational Inference [31] is the most direct inspiration for the architecture we use in our experiments. It is similar to the previous papers, but a more end-to-end approach, using VAEs to learn both relationships and dynamics in physical environments. The encoder is composed of GNNs and learns a distribution over types of edges between nodes. Samples are drawn from a continuous approximation of this discrete distribution, and the decoder (another GNN) predicts the next state of the physical system. This architecture does quite well on the overall prediction task, and reasonably well at recovering the overall graph (although it should be noted that their environments are quite simple).

Chapter 3

Concept

In this chapter we explore the ways we might use causal models for modeling MDPs, settling on a particular approach that reaches (we believe) a reasonable trade off between representation power and learnability. We will go on to cover implementation details and experimental results in Chapter 4.

Our core guiding principles for what makes a "good" causal model will be:

- At all times the causal diagram should be explicitly known (not merely represented or encoded in some inaccessible way)
- It should be possible to compute interventions and counterfactuals with the model
- The causal model should be minimal: f_i should be non-constant over all $v_p \in \text{parents}(v_i)$

It will turn out that we cannot achieve the final property with a single SCM. This is because the causal relationships in an MDP vary over time. Rather than abandon minimality, we settle on a representation that encodes many different SCMs (while still satisfying the first two properties).

3.1 Representing General MDPs as TSCMs

We will start with the case of a general MDP, which can be represented as TSCMs. This representation is not useful (in that it is exactly as difficult to learn this TSCM as it is to learn a general MDP), but it will clarify the general technique.

Definition 3.1.1 (Representation of an MDP)**.**
An MDP $M = (S, A, T, R)$ is *represented by* an SCM C whose random variables are s_t, a_t, and r_t for all $t \geq 0$ if:

$$P^C(s'_1 = s) = T(s_1 = s)$$
$$P^C(s'_t = s_n | s'_{t-1} = s, a'_{t-1} = a) = T(s_t = s_n | S_{t-1} = s, a_{t-1} = a)$$
$$P^C(r'_t = r | s'_t = s_n, s'_{t-1} = s, a'_t = a) = T(r_t = r | s_t = s_n, s_{t-1} = s, a_t = a)$$

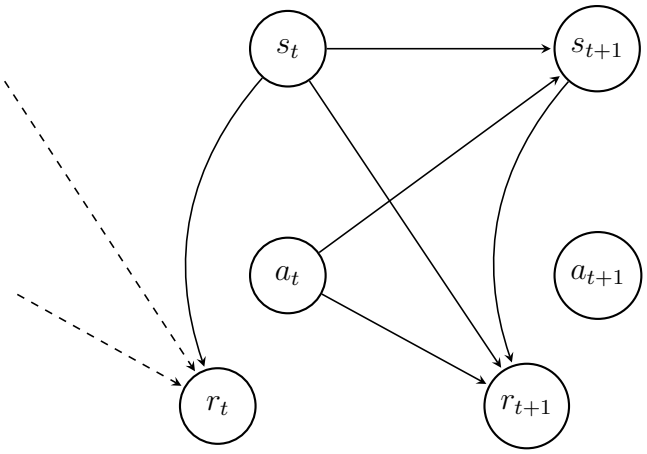
Note: we distinguish between s and s' because we are not saying the SCM and the MDP *share* random variables, only that their random variables have the same distributions.

Theorem 3.1.1 (Representation of MDPs by SCMs)**.** *Any MDP $M = (S, A, T, R)$ can be represented by a Markovian TSCM.*

Proof. We prove this by construction.

Let T be a Markovian TSCM whose variables are s', a', and r', defined on S, A, and $\mathrm{im}(R)$ respectively. Let $F_{s,a}$ be the cumulative distribution function of $T(s, a)$. Let ν_s and r_s both be distributed according to $U(0, 1)$.

Define the DAG structure on these variables by:



and define $f_s : S \times A \times [0,1] \to S$ by:

$$f_s(s, a, \nu_s) := F^{-1}_{s,a}(\nu_s) \tag{3.1}$$

Recall that $s_{t+1} = f_s(s'_t, a'_t, r'_t)$. Therefore:

$$\begin{aligned} P^C(s'_{t+1} = s_n | s'_t = s, a'_t = a) &= P^C(f_s(s, a, \nu_s) = s_n | s'_t = s, a'_t = a) \\ &= F^{-1}_{s,a}(\nu_s) \\ &= T(s, a) \end{aligned}$$

The construction for r is similar. We can easily include a discount factor γ by modifying the function f to $f_d = \gamma^t * f$ □

This is also a *minimal* representation of a general MDP. Specific MDPs may not require all arrows (e.g. if reward is constant), but most MDPs require every arrow. Therefore this is the simplest SCM representation of a general MDP.[1]

3.1.1 Modeling Actions

The above picture doesn't include the policy as part of the SCM: actions are treated as noise variables. Of course actions are (usually) not noise variables: they are produced by

[1] Assuming we split the MDP into this set of variables.

some policy $\pi : S \to A$, or rather by a sequence of policies which update at each time step:

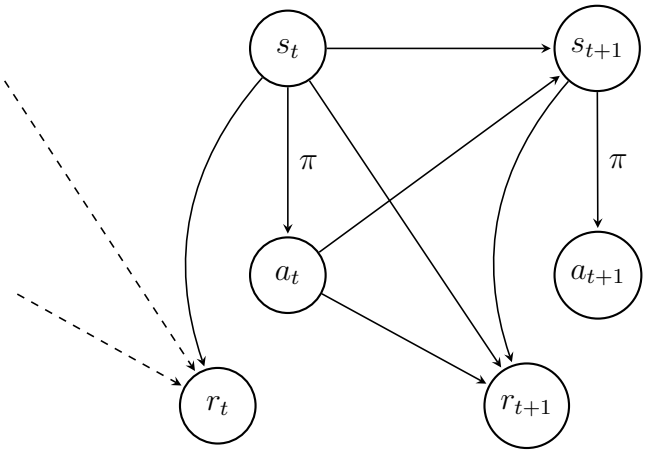


Figure 3.1: Causal representation of a general MDP with a fixed policy.

This is a causal representation for a general MDP with a *fixed* policy. If we wish to include a changing policy, we augment the graph with a policy variable π:

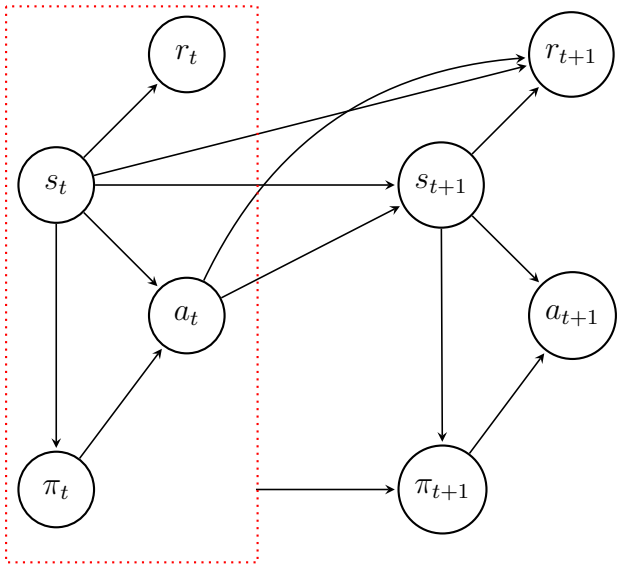


Figure 3.2: Causal representation of a general MDP with an updating policy. The arrow from the rectangle to π_{t+1} represents the fact that π_{t+1} depends on everything in the rectangle.

Since we are mostly interested in causal models of the *state* transition, we will usually leave π out of diagrams.

3.2 Vector Valued States

In practice, most RL environments have vector-valued states. Applying the general TSCM representation to an environment with states in $\mathbb{R}^n$ we obtain:

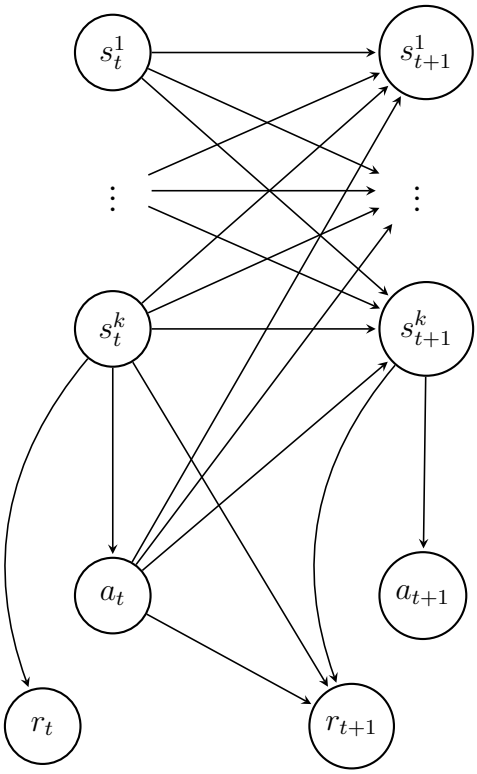


Figure 3.3: A TSCM representation of an MDP with vector-valued states. For the sake of clarity we have left out some of the edges from state to action and reward.

However, this representation is no longer necessarily minimal: it may be the case that v_t^i does not cause v_{t+1}^j. This is relatively common in practice: consider for instance the walls and floors of most video game environments. While they affect other objects in the environment, no other object affects them.

For the particular case of objects which *never* interact at any time step, we can use TSCM discovery to remove those arrows. This gives us a simple algorithm for generating a model:

1. Draw (potentially very many) samples X from M (using a random agent)[2]
2. Use any TSCM causal discovery algorithm to determine which elements of state do *not* have edges between them.
3. Initialize a neural network f as a model of the M. For every pair of states (v_i, v_j) that do not have an edge between them, set the parameters connecting them in f to 0.
4. Train f using the existing samples as a training set.[3]

3.3 The Time-Variance Problem

There is a significant problem with 3.2: the causal relationships in most environments change at every time step. 3.2 learns the smallest causal graph that contains *every* arrow that *ever* appears. For any particular time step there may be a much smaller causal graph that contains all the information required to predict the next state.

As an example, consider an environment consisting of n identical spheres bouncing in a closed space (i.e. a hard-sphere gas model). Two spheres are causally related at time step t if they will collide in that time step, in the sense that a function predicting the next state of either sphere must have the current state of both spheres as an input.[4]

Over a long enough period of time, every pair of spheres will interact, and so 3.2 will learn a complete bipartite causal graph and the resulting neural network model gains

[2]The use of a random agent here is suspect and could potentially cause problems because of distributional shift when we switch to using the model.

[3]We could of course sample again, but this is more sample-efficient.

[4]We will completely ignore the action space for the purposes of this example, as it makes no difference.

nothing from all this additional work. However at any particular time step the graph of interactions will frequently be empty and almost always be sparse[5].

We can see an example of this situation below:

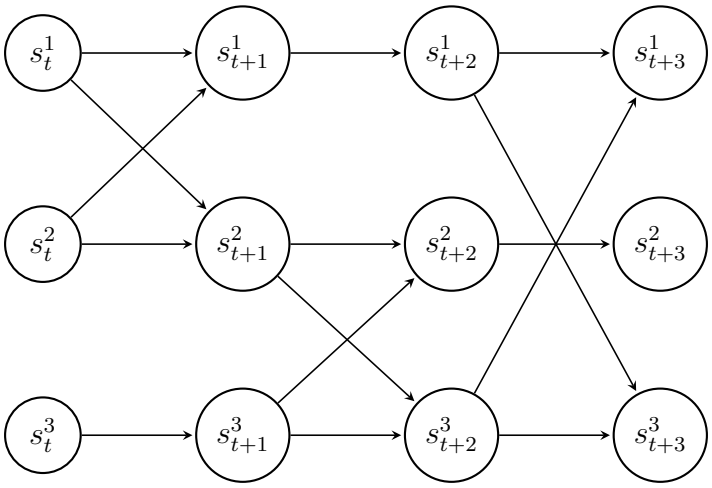


Figure 3.4: An environment where TSCM discovery would return a fully connected graph, but each individual time step is fairly sparse.

We cannot even try running the entire environment *many* times, because the causal graph at time step t will not be the same *across runs*.

This does not mean that 3.2 is useless. In environments where there are many non-interacting components it can still serve to reduce the complexity of the model that must be learned. However we would ideally have an architecture that allows us to take advantage of the local causal information as well as the global. We develop such an approach in the next section.

3.4 Local Representations

We must assume *some* connection between the diagrams of different states, or else the learning problem is completely hopeless. Drawing inspiration from physical environments, we will assume that the environment can be decomposed as two functions g :

[5]As the length of a time step tends to 0 we should in fact see at most 1 arrow per time step, assuming we are doing a pure rigid body simulation.

$S \times A \to \text{Bipartite}(k, k)$ and function $f : \text{Bipartite}(k, k) \times S \times A \to S$, where g produces the causal diagram at each time step, and f predicts the next state while taking the causal diagram into account.

We formalize this as a new notion of representation of an MDP:

Definition 3.4.1 (Local representation of an MDP)**.**
A *local representation* of an MDP with states in $\mathbb{R}^k$ is a pair

$$(g : S \times A \to \text{Bipartite}(k, k), f : \text{Bipartite}(k, k) \times S \times A \to \mathcal{D}(S))$$

that satisfies:

$$T(s_t, a_t) = f(g(s_t, a_t), s_t, a_t)$$

and $v_j \notin \text{parents}(v_i)$ in $g(s_t, a_t)$ implies the component function

$$f_i(g(s_t, a_t), s_t, a_t) := f(g(s_t, a_t), s_t, a_t)_i$$

is constant in v_j.

Essentially, f is a function for predicting the next state which respects the causal relationships encoded by g.

Remark. A TSCM representation is a local representation where g is constant.

We use this to define a new sense of minimality that more closely captures our intuition that we can do better than treat a box full of bouncing spheres as being fully-connected at every time step.

Definition 3.4.2 (Covering local representation of an MDP)**.**
Let g_t be a graph in the image of g. Let (S', A') be the preimage of g_t. A representation is *covering* if the function $f_i(g_t, -, -) : (S', A') \to S$ is *non*-constant in all $v_p \in \text{parents}(v_i)$.

Covering representations are closely related to minimal SCMs. However because we defined our non-constant functions in terms of the pre-image of a particular graph, this

will not quite work. For instance if g is constant then the pre-image is the set of all states and actions, so such a representation is covering as long as f is non-constant in all arguments.

We can fix this problem by selecting minimal covering representations under a subgraph ordering:

Definition 3.4.3 (Minimal local representation of an MDP)**.**
Let L_C be the set of covering local representations of an MDP M. Define a partial order on L_C by $l \leq l'$ iff $g(s,a) \subset g'(s,a)$ for all $s \in S, a \in A$.

A local representation is *minimal* if it is minimal with respect to this order.

Definition 3.4.4 (Minimal TSCM Representation)**.**
A TSCM representation C of an MDP M is minimal if it is minimal as a structural causal model.

Theorem 3.4.1. *There exist MDPs where the only minimal TSCM representation is fully connected, but the minimal local representation is not.*

Proof. The previously described hard-spheres gas model has a minimal local representation (the rules of rigid body motion) whose graphs are sparse, and can only be represented by a fully connected TSCM. □

This problem is not limited to the hard-sphere gas model. Most physics based environments (e.g. the MuJoCo environments [32]) will have fully-connected or almost-fully connected TSCM representations. Even very simple environments like Pong suffer from this problem: a minimal local representation would have an edge between the ball and a paddle only when they are about to collide, but a TSCM representation would have edges between them at all time steps.

3.5 Interventions

The discussion so far has focused entirely on the observational case. We move now to considering interventions. We will discuss several ways of defining interventions on MDPs, and most importantly we will show that in the standard setup where the agent's only interaction with the environment is through its choice of action, there are no interventions.

3.5.1 Choice of Action as Intervention

There is a somewhat natural intuition that actions are interventions on the environment. The agent or policy is outside the environment, making choices that affect the environment, so surely it makes sense to call these interventions? This intuition is visualized in 3.5.

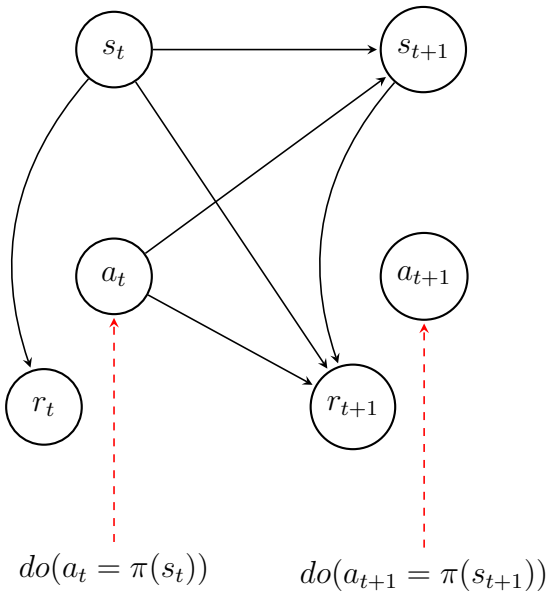


Figure 3.5: Actions as interventions

The problem with this view is that the s_t argument to $\pi(s_t)$ prevents it from being a proper intervention. An intervention must *remove* all arrows into the interventional variables, but in this case we've actually *added* an arrow from s_t to a_t (as in 3.1).

This problem is inescapable: *any* policy that depends on s_t will force s_t to be a parent of a_t, and thus choice of action is not an intervention.

It is not even clear how we would use "actions as interventions" if they were. Interventions are useful for asking "what if" questions (e.g. "what if all patients were treated in this way"). However under this view, the "what if" question is simply "what if we took this action instead". *This is just a normal rollout*. So we cannot use this definition of intervention to bring anything *new* to the planning problem in RL.

3.5.2 Actions and Local Representations

Within the local representation framework, this situation might be salvageable. Interventions are helpful for causal discovery because they *modify* the causal graph, this modification is apparent from sample data, and the original graph can be reconstructed as a result.

While different actions are still not properly *interventions* in the usual sense when applied to local representations, they *can* modify the graph, since g is a function of both state and action. Depending on the exact nature of g, this shift might be useful for learning a causal model.

Consider an extremely contrived example: the state is in $\mathbb{R}^n$. It is known that g is fully connected under the null action, and that under other actions some set of connections will be broken, but the exact mapping from actions to graphs. f_i is simply the sum of parents(v_t^i), and this is known.

In this case all steps where a particular action is taken form a TSCM (since the action is the sole determiner of the graph at a time step), and TSCM discovery can be used to find the map from actions to graphs.

Thus while actions are not strictly speaking *interventions*, they may still be usable for the causal model learning problem in a similar way.

Of course in this example we are starting with a great deal of information about g and f. It is not at all obvious how to apply this to the general case where g and f are

unknown, but it is at least possible that there are learning algorithms that could take this information into account.

Lastly, note that while actions bear some similarity to interventions for the purposes of *learning* a causal model, they are not helpful for the planning problem. Once g and f are known (or well-approximated), then asking the effect of an "intervention" is just performing a normal rollout, as it was under the previous view.

3.5.3 External Interventions

As an alternative, we can simply consider the set of all possible interventions on the SCM at time step t C_t. Of course practical RL environments do not usually *have* a mechanism for performing interventions at all, and we could consider many possible sets of interventions:

- Modifications to the low-level memory of the program (e.g. editing a register)
- Modifications to the memory *or the code* of the program
- Modifications to the high-level memory of the program (e.g. editing an attribute on a Python object)
- Modifications to the high-level memory of the program, while enforcing some invariants (e.g. disallowing interventions that produce technically possible but invalid game states)

We are mainly interested in how to use interventions once defined, and in what circumstances it makes sense to define them at all.

In many RL environments the standard for success is comparison to human learning. Defining a set of interventions on these environments does not make much sense, since this gives an RL algorithm access to tools human learners do not have and this invalidates the comparison.

However there are scenarios where it makes sense. [33] and [25] treat different *environments* as different interventional distributions of the same underlying causal model.

Something similar could certainly be done for the causal model learning case. Consider two MDPs M and M' where g and f are known to be drawn from one of several possible function classes. M also has a much smaller state space than M'. Then the classes could be learned on M, before generalizing to M'.

As a concrete example: consider the same hard-sphere gas model we have worked with. M is the environment where there are only five gas particles, and M' has 10000. But the underlying g and f are the same, so in theory they could be learned on M and then immediately generalized to M'.

3.6 Counterfactuals

Evaluating a counterfactual requires holding noise terms from a particular sample fixed. Thus the noise terms must be *known*. If they are not directly observed, then they must be inferred. This is certainly possible in theory (e.g. by performing MLE estimation of the noise variables, given a sample and the predictions of g and f), but this may be quite difficult in practice.

[24] demonstrates that given the ability to evaluate counterfactuals, they can be used to generate more accurate rollouts using the "actions as interventions" view. This is particularly true in the partially-observable case where noise terms represent unobserved variables. The shift from a normal rollout to a counterfactual rollout allows a planning algorithm to consider a rollout conditioned on particular values for the unobserved variables, producing an unbiased rollout.

Counterfactuals could also be used for the credit-assignment problem[6]. If we wish to determine which action was "responsible" for a particular result, then we can consider a (potentially very large) set of counterfactual actions, and assign credit to any actions

[6] In many ways counterfactuals were developed for exactly this problem

$a_t = a$ such that the result does not occur under counterfactual actions at those time steps.

Chapter 4

Implementation and Experiments

In this chapter we cover several variants of algorithms for learning local representations.

The core difficulty we encounter is that we assume the true causal graph is inaccessible at training time. Thus g must be trained by backpropagating the loss for f

All code is available at vluzko/causal_rl.

4.1 Local Representations as Neural Networks

Let (g, f) be a local representation. There are two difficulties: enforcing the constraint that f_i is constant in the non-parents of v_i, and making sure g produces a bipartite graph in a differentiable way,

We can accomplish this by making f a message passing network. A message passing network [34] is a graph neural network that takes a graph as input, and outputs a value for each node that depends only on the value of the node's neighbors[1].

Formally, the graph neural network computes the following:

$$\begin{aligned} m_{ij}^{t+1} &= \phi_{ij}(v_i^t, v_j^t) \\ v_i^{t+1} &= \psi_i\left(v_i^t, a_{j:v_j^t \in \text{parents}(v_i^{t+1})}(m_{ij}^{t+1})\right) \end{aligned}$$

[1] More complex architectures are possible, but for our purposes we will use this definition.

Here m_{ij} is the *message* from v^j to v^i. a is an *aggregation* function, meaning a function which accepts a list as an argument and which is invariant under permutations of the list. In practice a is usually the sum or mean of its arguments.

ϕ_{ij} and ψ_i are neural networks. For most of our experiments all ϕ and ψ networks are the same, but this is not necessary.

Thus v_i^{t+1} depends only on its parents in $g(s^t, a^t)$, assuming there is always an arrow from v_i^t to v_i^{t+1}.

g is constructed so that for all $i \neq j$, g produces a value

$$p_{ij}^{t+1} = P\left((v_i^t, v_j^{t+1}) \in g^{t+1}\right)$$

where g^{t+1} is the actual graph at time step $t+1$.

This is easy to accomplish by setting the output space of of g to be adjacency matrices on k variables, with all entries in $[0, 1]$ and the diagonal set to 1.

Treating all p_{ij} as independent, we have a distribution over bipartite(k, k), which we call D^{t+1}. We can use this distribution as an input to f in two ways.

4.2 Environments

The majority of our experiments are performed on a two-dimensional hard-sphere gas model (Figure 4.1). In several experiments we use shapes besides spheres. The reason for this choice is that it is easy to compute the g^t in this environment (we simply detect collisions). All of our experiments use an environment consisting of 15 particles, unless otherwise stated.

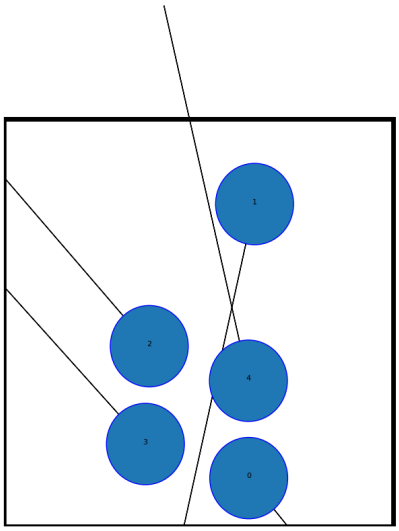


Figure 4.1: Visualization of a single state of the hard-sphere environment. The lines are velocity vectors.

The state space for this environment is the positions and velocities of each particle, making it a vector in $\mathbb{R}^{n\times 4}$, where n is the number of particles.

4.3 Architecture

We approximate g with a single fully-connected neural network with two hidden layers of width 32. g takes the full state vector as input, and outputs a vector in $\mathbb{R}^{(n-1)^2}$. The output vector is passed through a sigmoid activation to obtain the probabilities of all edges (v_i, v_j).

To represent f we implement a simplified version of the system described by 4.1. Instead of multiple separate networks $\psi_{i,j}$ and ϕ_i, there are only two networks ψ and ϕ.[2]. Both are implemented as fully connected networks with one hidden layer of width 16.

[2]We can make this simplification because all objects in the environment are identical.

ψ takes pairs of particle states (i.e. vectors in $\mathbb{R}^8$), and produces a vector in $\mathbb{R}^4$ representing the message from one particle to another, with the position and velocity components being treated as channels.

ϕ takes a particle state and the aggregate of all messages to that particle, and produces the predicted next state of the particle.

All networks use leaky ReLU activations and batch normalization between layers, and were trained with the Adam optimizer with learning rate 0.1.

For all experiments we use the mean-squared error loss for the prediction for time step t $\hat{s}^t$ to train phi and psi:

$$MSE(\hat{s}^t, s^t) := \frac{1}{n}\sum_{i=1}^{n}(s_i^t - \hat{s}_i^t)^2$$

We will also write MSE^t for the loss at time step t.

We also computed the L_2 loss for g:

$$L_2(\hat{g}^t, g^t) := \left[\sum_{i=1}^{n}(g^t - \hat{g}^t)_i^2\right]^{1/2}$$

At no point is this loss used to *train* g. We assume that knowledge of the true causal graph is generally unavailable, and so we cannot use it to compute a loss. This loss is *only* used as an after-the-fact assessment of how well g is learning.

To combine g and f, we took two different approaches:

4.3.1 Weighted Architecture

In the weighted architecture, we declare the graph to be complete, and treat D^{t+1} as a weight on each edge. This changes the graph network architecture 4.1 to be:

$$v_i^{t+1} = \psi\left(v_i^t, a_{j\in\text{parents}(v_i^{t+1})}(D_{ij}^{t+1}m_{ij}^{t+1})\right)$$

If a is summation, then this modified version produces the expectation of a under D^{t+1}.

This makes the composition of g and f differentiable, meaning we can use the state prediction error to train g as well as f. However this comes at the price of no longer completely forcing f to ignore certain connections.

4.3.2 Sampling Architecture

As an alternative, we can use the REINFORCE algorithm [35] with a sample from D^{t+1}, giving us an actual bipartite graph that can be used directly with 4.1. The actual loss for g is:

$$L_g^t := \ln(p_{D^{t+1}}(g^{t+1})) * L_f^t$$

where L_f is the loss for f. This has the benefit of giving an explicit causal graph, at the cost of no longer being able to backpropagate directly.

4.4 Results

As we will see, our architecture can learn f reasonably well, but learning g is very difficult. This is not surprising, given that g has no access to ground truth. Furthermore, the true causal graph is generally quite sparse (collisions are rare), so g tends to learn to predict an empty graph everywhere[3].

4.4.1 Core Architecture

First we compare the weighted and sampling architectures (Figure 4.2).

[3] If we do not regularize g, it tends to predicts a fully connected graph everywhere

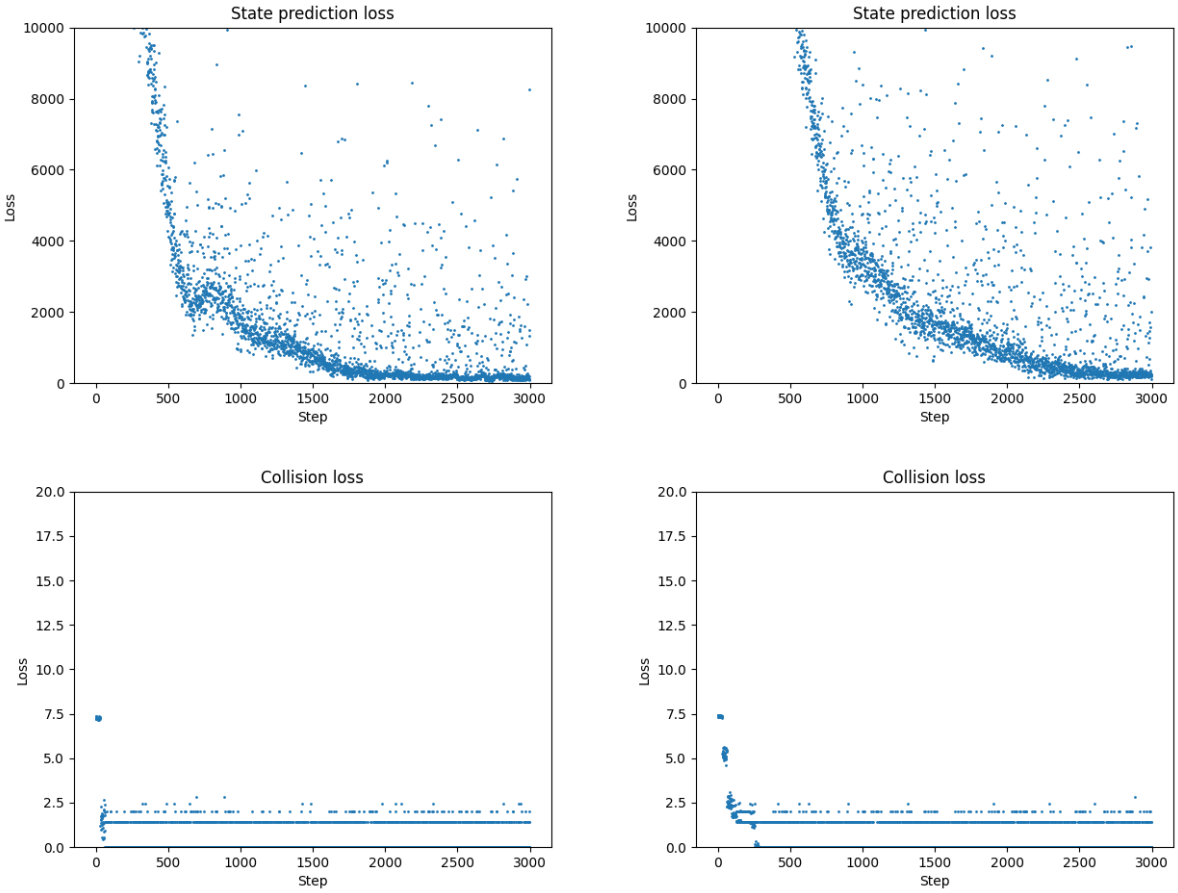


Figure 4.2: Left: state prediction loss and collision loss of the weighted architecture on the hard sphere gas environment. Right: The same for the sampling architecture.

As should be immediately apparent, the overall model learns to predict the next state reasonably well (with little difference between architectures), but the gradient of g drops to 0 fairly quickly (Figure 4.3). This is partially because the underlying graphs are in fact frequently sparse or empty, but (as we will see in 4.5.1), increasing the proportion of non-empty graphs does not suffice to solve this problem.

The fact that f still learns at all is mostly a fact about the environment. In the hard-sphere gas model the position of a particle will change very little across time-steps. The velocity will not change at all, except after a collision. Since collisions are relatively rare, simply reproducing the *current* state can produce reasonably good loss.

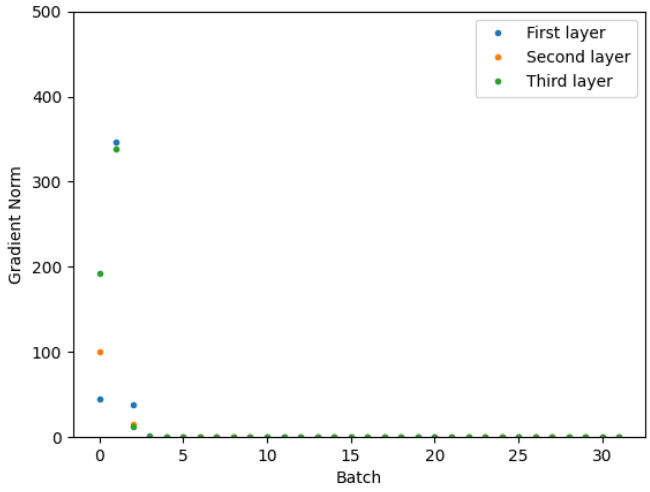


Figure 4.3: Gradient norms at each batch, for each layer of g.

Collisions are also the source of the high variance. The high loss states are mostly those where a collision occurs that g does not predict. For these states the position of the particle is mostly correctly predicted but the velocity prediction can be very wrong, to the extent that the incorrect velocity prediction for the two particles involved dominates the overall loss.

Plausibly it is the combination of the fact that velocity usually doesn't change, and occasionally changes quite drastically, that makes it so difficult for g to learn. If f quickly learns to simply reproduce the current velocity, then g's predictions become irrelevant.

As a baseline comparison, Figure 4.4 shows the results of training a fully connected model as a predictor:

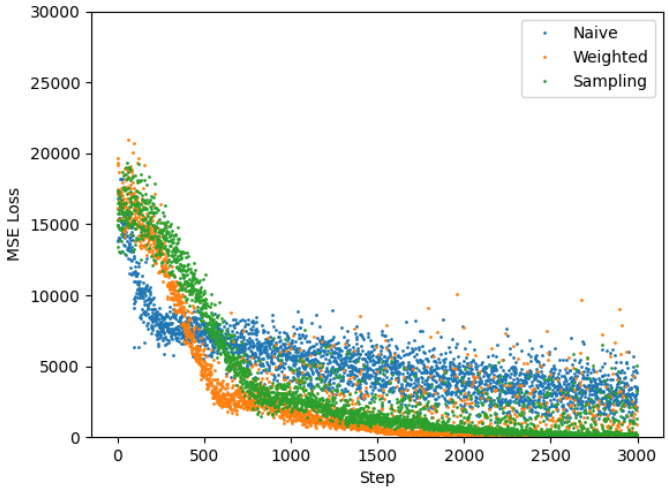


Figure 4.4: Training comparison between a simple fully connected model and the two causal models.

The local representation architecture clearly still helps with learning to some extent, most likely because it *does* successfully learn that *most* connections are unnecessary, which makes the prediction task much simpler.

4.4.2 Training with the True Graph

As a baseline, we train a predictor using the *true* graph rather than an estimate $\hat{g}$. This gives us an idea of what kind of performance can be expected if g is learned correctly (Figure 4.5).

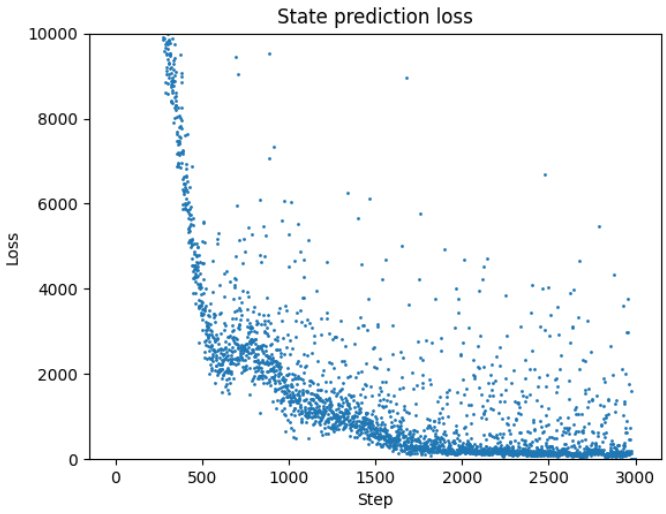


Figure 4.5: Performance of the state predictor when the true graph is given as input

Under these conditions, f achieves better test performance (4.6) and learns faster (4.7).

Model	Test Loss
Weighted predictor	643.65
Sampling predictor	645.76
With true graph	397.22

Figure 4.6: Test loss comparison

However, it does not perform *significantly* better. This is an unfortunate property of the hard-sphere environment: even without the true causal graph, the next state can be predicted quite well from the previous state, since most of the particles simply continue moving along their current trajectory.

4.4.3 Multi-Typed

Very few environments actually consist of totally uniform objects. Here we test performance when there are multiple types of object. We modify the hard-sphere gas environment to include different particle shapes (specifically squares, of different mass and size). In one environment the type of the object is passed as a one-hot encoded vector, in the other it is not. Performance improves in the latter environment (Figure 4.8).

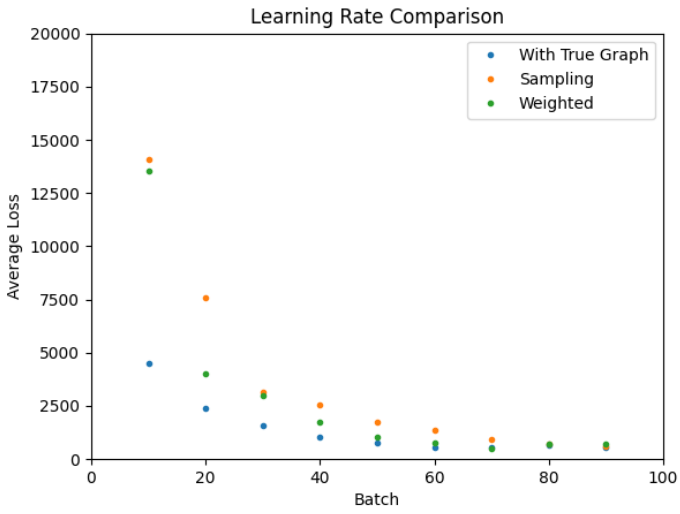


Figure 4.7: Learning rate comparison. Each value is the mean training loss across the last 5 batches.

Number of Particles	No Types Test Loss	With Types Test Loss
5	8521.40	5953.93
15	4002.44	3541.45
25	1723.79	854.11

Figure 4.8: Test loss comparison

This is a greatly simplified experiment. Most environments have a much richer set of object types (possibly equal to the number of objects), the types of those objects are not directly encoded in the observation space, and the number of types is initially unknown. Nevertheless it gives us some idea of how performance changes as we introduce more complexity.

4.5 Alternate Training Algorithms

To handle the difficulties in training g, we test several alternative training algorithms.

4.5.1 Buffered Errors

First we try building a buffer of bad states, defined as any state s^t where:

$$MSE^t > \frac{2}{b/2} \sum_{i=1}^{b/2} MSE^{t-i}$$

where b is the batch size. States are then resampled from this buffer with probability p, or a new state is generated with probability $1 - p$. Sampling does not begin until the buffer has reached a minimum size of 200 states.

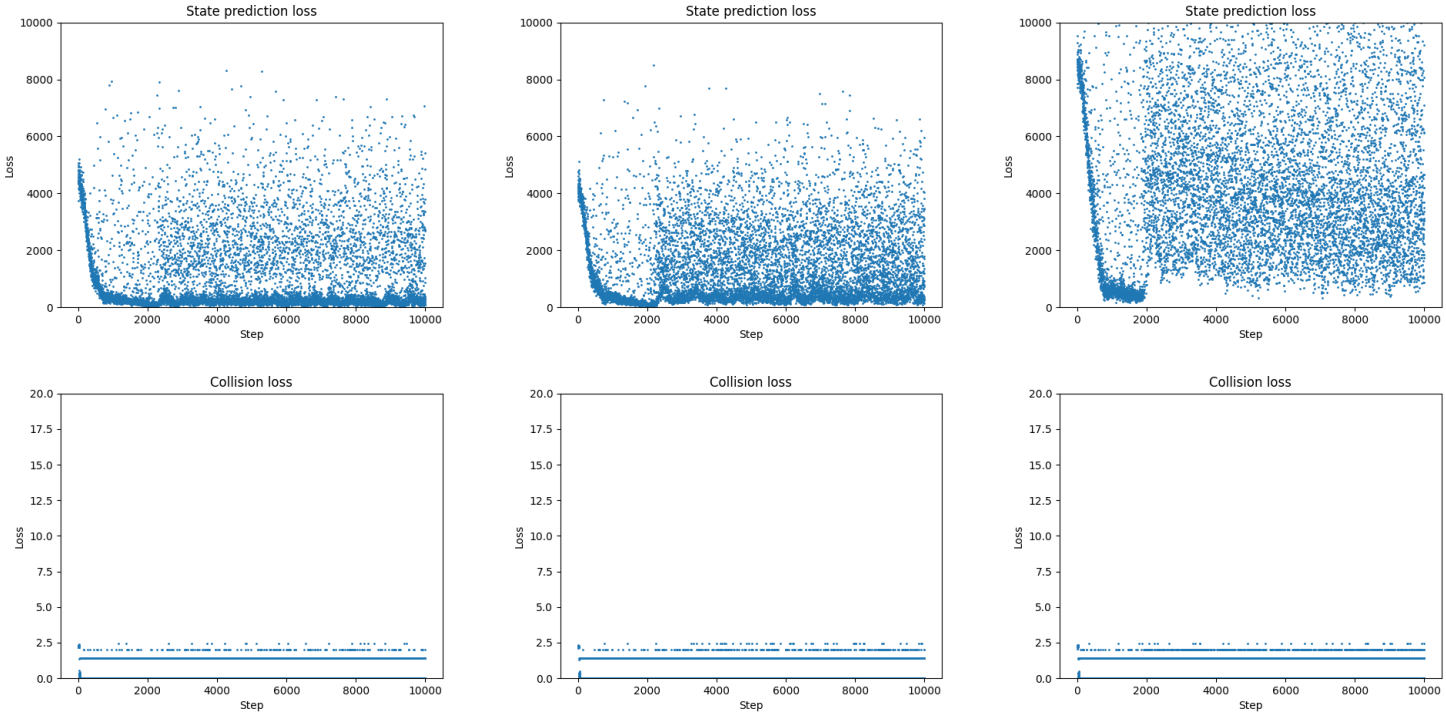


Figure 4.9: Performance of the weighted predictor with buffered errors. From left to right $p = 0.25, 0.5$, and 0.75.

As should be apparent from 4.9, the loss immediately becomes much worse once the buffer starts being used, with the drop in accuracy increasing with higher sampling probability. It may be that given how late the buffer starts being used, the networks are already close to a local optimum and cannot break out of it.

Preconstructing the Buffer

As a mitigation, we try instead building the buffer first (using the original unbuffered algorithm), and then train a second copy that uses the buffer from the start (4.10).

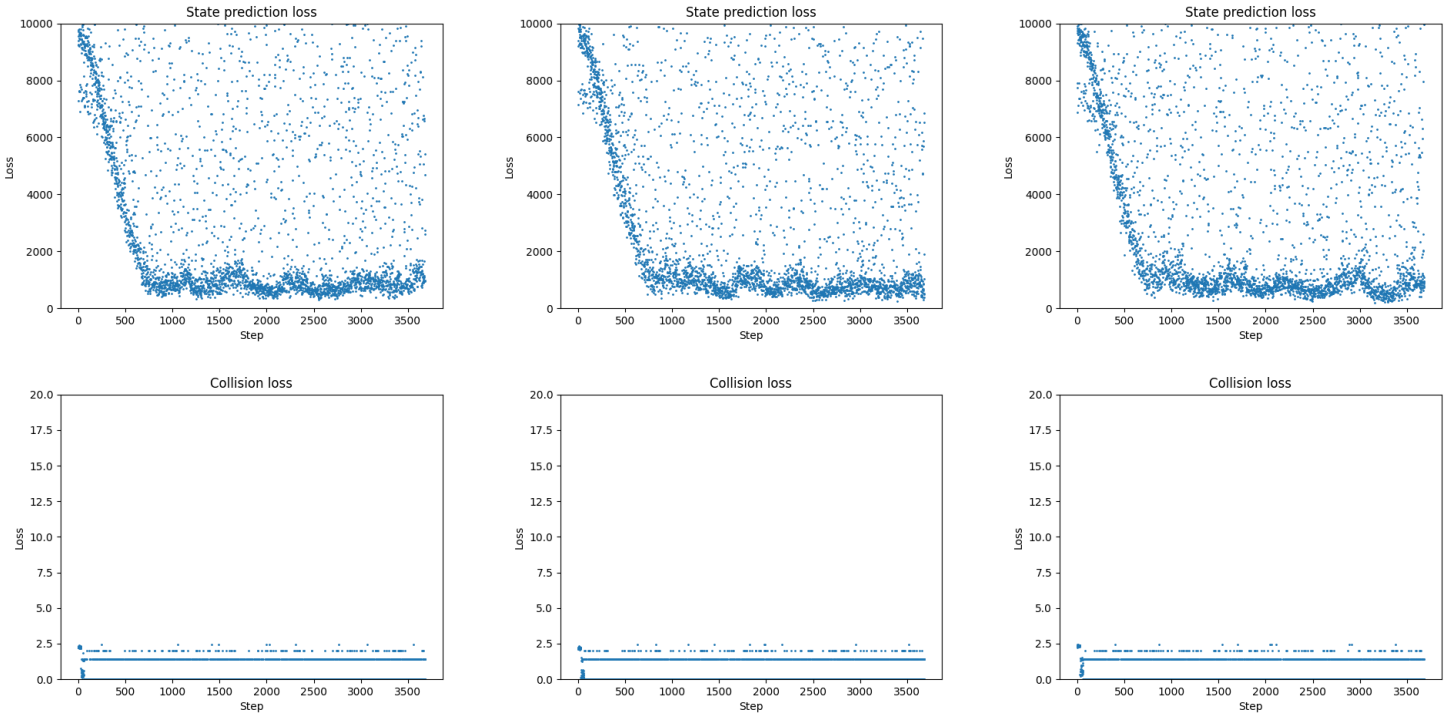


Figure 4.10: Performance of the weighted predictor with buffered errors. From left to right $p = 0.25, 0.5$, and 0.75.

With the preconstructed buffer the model does better overall, but it is clearly not sufficient to force g to learn.

4.5.2 Iterated Learning

As an alternative approach, we try switching between training f and g, the idea being to train g for some number of batches, then f for a different number of batches. In this way g is given a chance to learn before f learns to approximate well with just the current state. In this experiment we perform no L_1 regularization on g, as otherwise it quickly learns to predict a nearly empty graph (4.11).

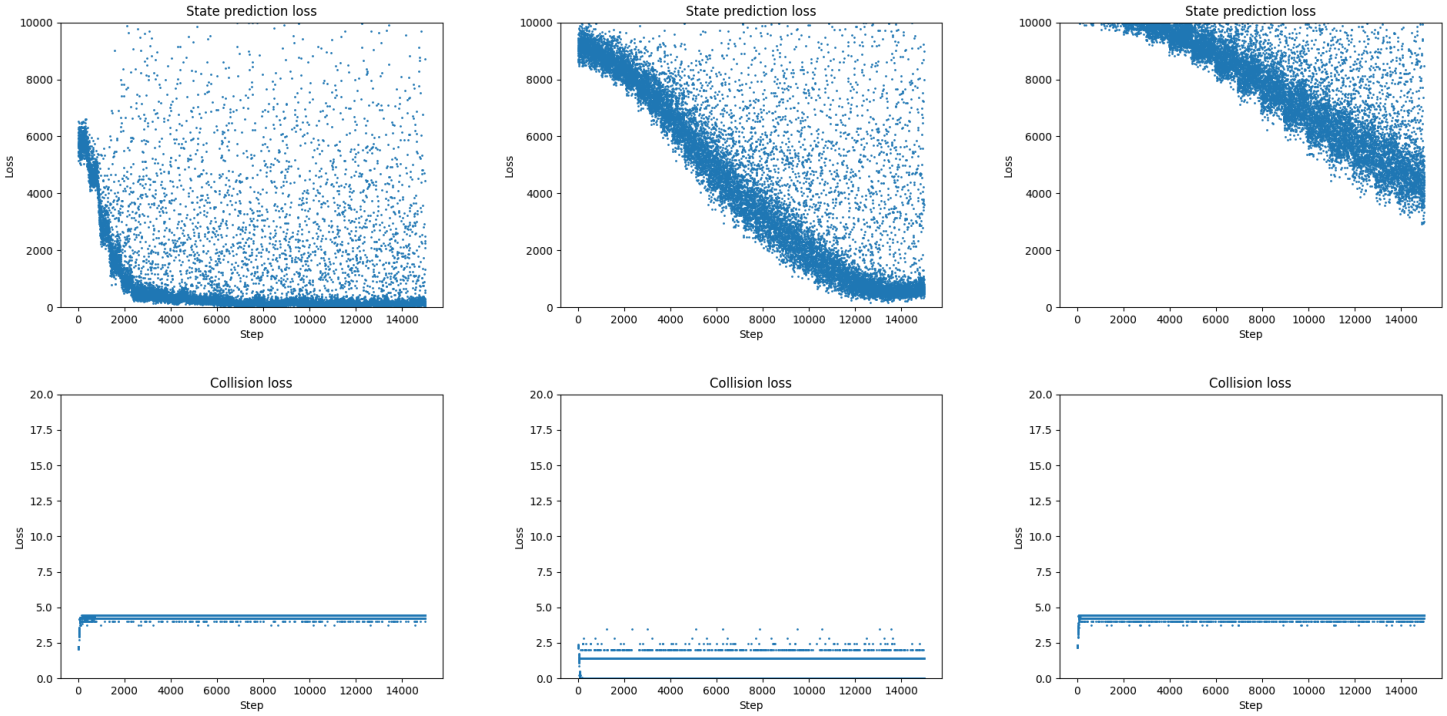


Figure 4.11: Performance of the weighted predictor with iterated learning. From left to right the batch sizes are 10 and 5, 15 and 1, and 30 and 1 (g batches and f batches respectively).

Of course g still suffers from sparse gradients. As a way around this we try combining the preconstructed buffer and the iterated approach (4.12).

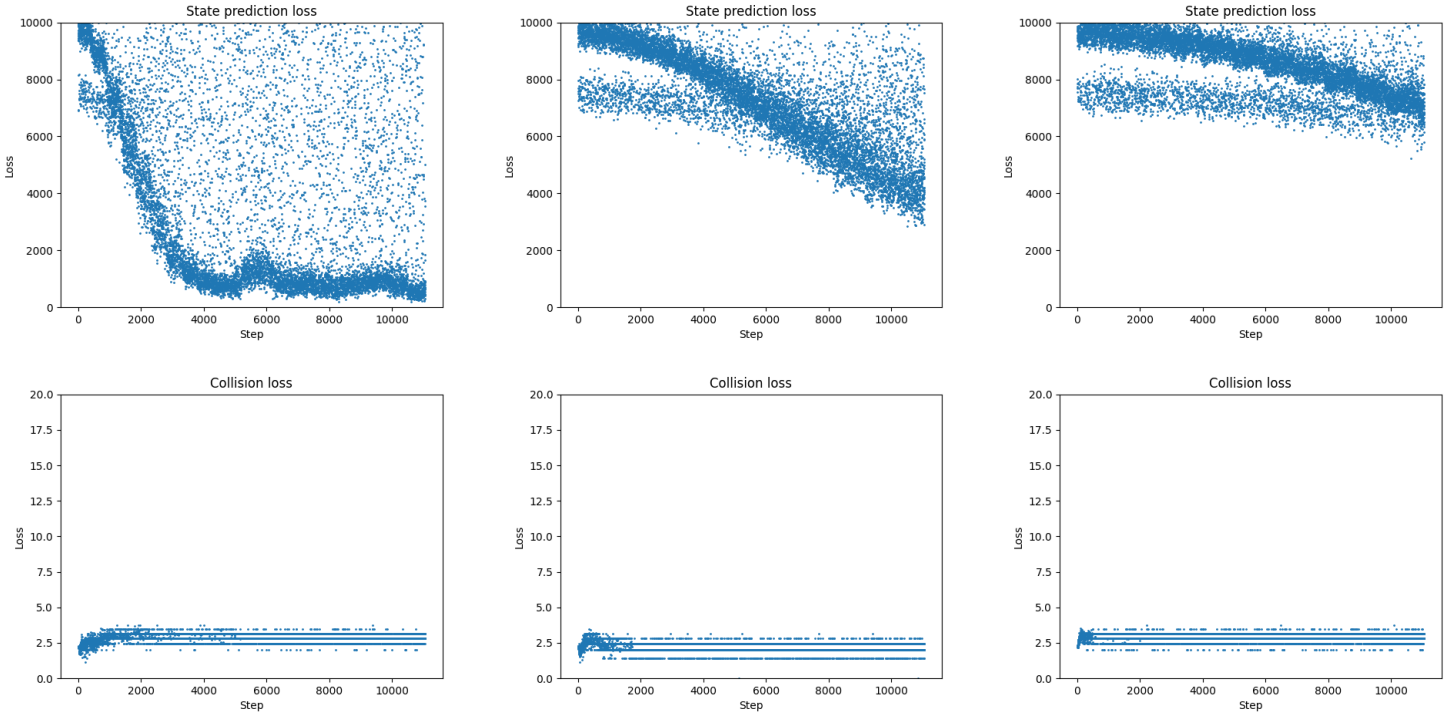


Figure 4.12: Performance of the weighted predictor with iterated learning. From left to right the batch sizes are 10 and 5, 15 and 1, and 30 and 1 (g batches and f batches respectively).

Unfortunately even in this experiment the gradients for g still vanish:

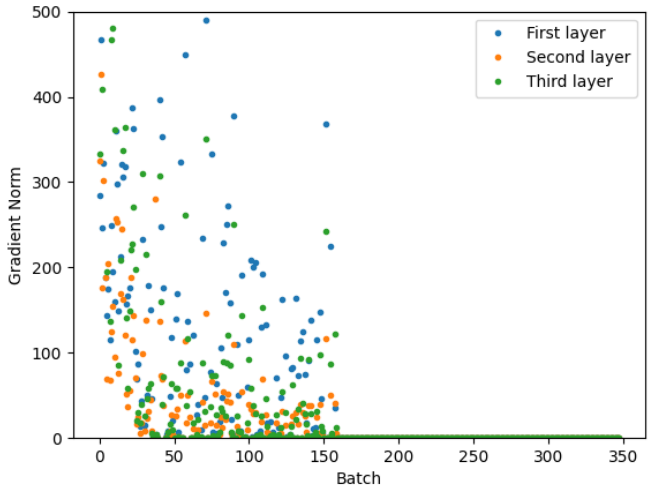


Figure 4.13: Norm of the gradient for each layer. Training signal is fairly rapidly lost.

This points to the fundamental problem being that the environment is too well predicted by extrapolating the current trajectories of each particle, and so very little optimization pressure is put on g.

It may be that this problem will disappear in other environments where the current trajectory is not sufficient to make accurate predictions. Testing this hypothesis would require environments where the full causal graph is known for each step. There are common RL environments where this graph can in theory be obtained (e.g. the collision graph in MuJoCo environments), but (to our knowledge) no environment makes it easily accessible without additional engineering effort, so that step must be performed first.

If it is *not* simply a problem with this environment, then it could of course be that the architectures tested here are flawed in some way, or that the problem is simply too difficult for current techniques.

Chapter 5

Conclusion and Future Work

We have shown that traditional causal discovery algorithms will not work on many reinforcement learning settings, or rather that they will work but will discover a "vacuous" causal graph which forces everything in the environment to be connected.

This problem is fundamentally caused by the time-variance of the underlying causal relationships. To solve this problem we introduced *local representations*, which consist of separate functions for predicting the causal graph at each time-step and for predicting the next state given the causal graph and current state. These local representations are in theory able to capture the dynamics of an MDP with a *minimum* number of causal relationships, unlike a more traditional approach using time-series structural causal models.

We developed an algorithm for learning local representations, using *only* knowledge of the states and not the true underlying causal graph. We implemented several variants of this algorithm, and tested them on a simple hard-sphere gas model. We find that while the overall architecture can learn to make accurate predictions the graph predictor has significant difficulty learning the causal graphs. This is most likely because the hard-sphere gas environment can be well-approximated even with an empty causal graph.

5.1 Future Work

There are several major steps required to bring this paradigm from toy environments to practical RL environments.

First, the issues with training the graph predictor must be resolved. Since this is plausibly caused by the simplicity of the current environment, more complex environments must be annotated with their true causal graphs so that the accuracy of the g approximation can be assessed. For instance, the MuJoCo environments could be annotated with their collision graphs.

If shifting to a different environment does not resolve the issues training g, then it may be that a different architecture that more closely couples g and f is required.

Second, to apply this work to environments with pixel-level data (or any state space that does not cleanly map to "objects" in the environment), some form of state representation must be learned, since directly predicting causal relationships between all pixels is probably too complex a problem for any learning to occur. This might be done with object detection algorithms, or perhaps by using the latent space of an autoencoder.

Lastly, the algorithm needs to be adapted to work in transfer learning settings. Many environments (e.g. all MuJoCo environments, many Atari environments) share some or all of their underlying causal dynamics. If the underlying dynamics can be learned on one environment, it should be possible to transfer to a novel environment with very little additional learning.

Bibliography

[1] J. Pearl. The seven tools of causal inference, with reflections on machine learning. *Communications of the ACM*, 62:54 – 60, 2019.

[2] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[3] R. Sutton and A. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 2005.

[4] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[5] T. M. Moerland, J. Broekens, and C. Jonker. Model-based reinforcement learning: A survey. *ArXiv*, abs/2006.16712, 2020.

[6] D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[7] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski.

Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020.

[8] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, USA, 2000.

[9] Jonathan Barrett, Robin Lorenz, and Ognyan Oreshkov. Quantum causal models, 2019.

[10] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. 2017.

[11] Steffen L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

[12] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '90, page 255–270, USA, 1990. Elsevier Science Inc.

[13] Judea Pearl and Thomas S. Verma. A theory of inferred causation. In Dag Prawitz, Brian Skyrms, and Dag Westerståhl, editors, *Logic, Methodology and Philosophy of Science IX*, volume 134 of *Studies in Logic and the Foundations of Mathematics*, pages 789–811. Elsevier, 1995.

[14] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1):62–72, 1991.

[15] Rajen Shah and Jonas Peters. The hardness of conditional independence testing and the generalised covariance measure. *Annals of Statistics*, 48, 04 2018.

[16] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. Causal inference on time series using restricted structural equation models. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[17] David Maxwell Chickering. Optimal structure identification with greedy search. *J. Mach. Learn. Res.*, 3(null):507–554, March 2003.

[18] Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. Dags with no tears: Continuous optimization for structure learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[19] Yue Yu, Jie Chen, Tian Gao, and Mo Yu. DAG-GNN: DAG structure learning with graph neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7154–7163. PMLR, 09–15 Jun 2019.

[20] Sébastien Lachapelle, Philippe Brouillard, Tristan Deleu, and Simon Lacoste-Julien. Gradient-based neural dag learning. In *International Conference on Learning Representations*, 2020.

[21] Shengyu Zhu, Ignavier Ng, and Zhitang Chen. Causal discovery with reinforcement learning. In *International Conference on Learning Representations*, 2020.

[22] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[23] N. Ke, Olexa Bilaniuk, Anirudh Goyal, S. Bauer, H. Larochelle, Chris Pal, and Yoshua Bengio. Learning neural causal models from unknown interventions. *ArXiv*, abs/1910.01075, 2019.

[24] Lars Buesing, Theophane Weber, Yori Zwols, Nicolas Heess, Sébastien Racanière, Arthur Guez, and Jean-Baptiste Lespiau. Woulda, coulda, shoulda: Counterfactually-guided policy search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[25] Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block MDPs. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11214–11224. PMLR, 13–18 Jul 2020.

[26] Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):947–1012, 2016.

[27] J. Zhang and E. Bareinboim. Transfer learning in multi-armed bandits: A causal approach. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1340–1346, Melbourne, Australia, Aug 2017. International Joint Conferences on Artificial Intelligence Organization.

[28] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[29] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4470–4479, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[30] Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victo-

ria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018.

[31] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2688–2697, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[32] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[33] Elan Rosenfeld, Pradeep Kumar Ravikumar, and Andrej Risteski. The risks of invariant risk minimization. In *International Conference on Learning Representations*, 2021.

[34] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[35] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.