When is Transfer Learning Possible?

My Phan **Kianté Brantley** [*1] **Stephanie Milani** [*2] **Soroush Mehri** [*3] **Gokul Swamy** [*2] **Geoffrey J. Gordon** [2]

Abstract

We present a general framework for transfer learning that is flexible enough to capture transfer in supervised, reinforcement, and imitation learning. Our framework enables new insights into the fundamental question of *when* we can successfully transfer learned information across problems. We model the learner as interacting with a sequence of problem instances, or *environments*, each of which is generated from a common structural causal model (SCM) by choosing the SCM's parameters from restricted sets. We derive a procedure that can propagate restrictions on SCM parameters through the SCM's graph structure to other parameters that we are trying to learn. The propagated restrictions then enable more efficient learning (i.e., transfer). By analyzing the procedure, we are able to challenge widely-held beliefs about transfer learning. First, we show that having *sparse* changes across environments is neither necessary nor sufficient for transfer. Second, we show an example where the common heuristic of *freezing* a layer in a network causes poor transfer performance. We then use our procedure to select a more refined set of parameters to freeze, leading to successful transfer learning.

1. Introduction

Transfer learning is a well-studied problem with many variations such as covariate shift, representation learning, imitation learning, and finding invariant predictors. In this work, we present a framework that unifies multiple transfer learning problems, and can be used broadly for analyzing transfer learning problems. As a result, we challenge commonly-held assumptions about transfer, such as when sparse mechanism shift and layer freezing can be helpful. Our framework also helps us design models and decide which components to transfer.

We model the transfer learning problem by viewing the *world* as a structural causal model (SCM, Pearl et al., 2000) with unknown constraints on its parameters. Learning about these constraints is what enables transfer. Within a world, the learner interacts with a sequence of *environments*. In each environment, the parameters of the SCM are chosen from the constrained domains, and the data is generated from the SCM with the chosen parameters. That is, each environment can be seen as a conditional intervention on a base SCM that represents the world. From each environment's data, we learn its parameters, and thus gain information about the unknown constraints. This view can be used to describe covariate shift, representation learning, and transfer learning in supervised learning, reinforcement learning, and imitation learning (see Section 3).

Therefore the environments' generation from the world follows the Independent Causal Mechanisms (ICM) Principle (Schölkopf et al., 2021), where the model is composed of independent modules. Within each module, we can have constraints on its parameters (called local constraints). We show how constraints on parameters across the model (called global constraints) can be computed from local constraints, allowing us to learn faster in new environments.

Contributions. In this work, we make the following contributions. First, we present a novel formalism for transfer learning that helps unify previously distinct areas of research, including covariate shift, representation, few-shot learning, and zero-shot learning, in which *environments* are generated from *worlds* by choosing local parameters of an SCM from constrained domains (Section 3). Second, we show how the constraints of global distributions can be computed from the constraints of local distributions, which enables us to analyze if transfer is feasible and derive a meta-algorithm to perform transfer (Section 4). Finally, we present case studies to illustrate how our work may be used to analyze and even challenge widely-held beliefs about transfer learning. We show that sparse mechanism shifts are neither necessary nor sufficient for transfer, and we show that freezing a layer of a network may either succeed or fail at transfer (Section 5).

[*]Equal contribution [1]Cornell University [2]Carnegie Mellon University [3]Elementera AI. Correspondence to: My Phan <mail.myphan@gmail.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

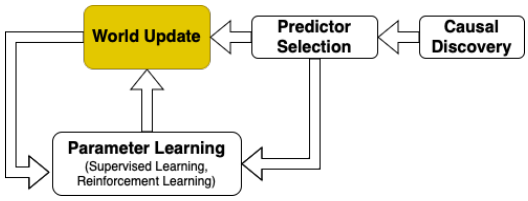


Figure 1. Relationships among problems in Section 3.1. We focus on transfer learning through the world update problem.

2. Motivation

Transfer learning means using knowledge from previous environments to speed up learning in a similar but different environment. For example, an agent can be trained to classify cows using photos where cows are often associated with pasture, and tested on photos where cows are often associated with sand (Beery et al., 2018). A robot may be trained to walk on Earth, then asked to perform on Mars where the gravity constant is different. Using our formalism, a world can consist of a set of possible associations between cows and the background, or a set of possible gravity constants, and environments correspond to specific values of the association or the gravity constant. Parameters learnt from previous environments can therefore be used as data points to learn about the constrained domain specified by the world. We call this the world update problem (Figure 1) and provide more details in Section 3.1. Formally, we define our **transfer learning** problem as the study of speedups we can get by *learning constraints* from previous environments to *reduce the hypothesis class* in the current environment — ruling out out possible parameter values, so that we need less data or less computation for learning.

3. Problem Formalism

To that end, define a *world* as an SCM with a set of constrained domains for its parameters. We generate an *environment* from the world by choosing parameters from their domains. The learner starts out with some but not all information about the world: it might know something about the structure and/or the constraints. It then sees data from a sequence of environments. Parameters learnt in these environments become data points to estimate the domains, so that we can speed up learning in new environments.

For one example, the learner might know that some parameters of a model are constant in all environments but might not know their shared value. After seeing enough data in one environment, the learner can transfer these parameters to the next (parameter freezing). In this example, the transfer strategy is clear. It is not obvious in general how to learn unknown constraints for transfer; we discuss this question in App. A, and give a more complex example next.

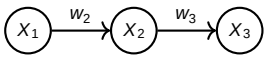


Figure 2. Causal graph for Example 3.1

Example 3.1. In the SCM of Figure 2, $X_3 = w_3 X_2 \ 2 \ \mathrm{R}^1; X_2 = w_2 X_1 \ 2 \ \mathrm{R}^2;$ and $X_1 \ 2 \ \mathrm{R}^3$ is drawn from a Gaussian distribution with mean w_1. In each environment, the parameters $w_1; w_2;$ and w_3 are selected from sets $W_1; W_2;$, and W_3. Here, W_1 is the set of all 2 1 matrices, $W_2 = f[1; 2; 3; 4; 5; 6]g$ is a singleton 2 3 matrix (same in all environments), and W_1 is R^3. Suppose the learner aims to predict X_3 from X_1 when X_2 is unobserved; it knows w_2 is constant in all environments but doesn't know its value. Starting from a conditional linear model, $X_3 = w_{3j1} X_1$, the optimal (unknown) parameter value is $w_{3j1} = w_3 w_2$ in each environment. Since w_2 is the same in all environments, w_{3j1} is always in the row space of w_2. Using examples of w_{3j1} learnt from previous environments, the learner can discover this row space, then use it as the hypothesis class for w_{3j1} in new environments to expedite learning.

We now present the formal definition of worlds and environments, extending Buesing et al. (2019). Upper-case letters denote random variables, lower-case letters their values, and bold-faced letters denote tuples.

Definition 3.1 (World)**.** Let $X = (X_1; : : : ; X_n) \ 2 \ _1 \ _n$ be n variables where $_i$ denotes the domain of X_i. Let $O \ X$ denote the observable variables and $Z \ X$ denotes the unobservable ones. Let $U = (U_1; : : : ; U_n)$ where U_i is a vector of d_i independent uniform noise variables.[1] Let f_w be a parameterized function class and $P_i \ fX_1; : : : ; X_{i \ 1}g$ be the set of possible causal parents of X_i, so that $X_i = f_{w_i}(P_i; U_i); \ 1 \ i \ n$. (We call w_i an *atomic parameter*, to distinguish from aggregated parameters like w_{3j1} in Ex. 3.1.) Let D_i be the unconstrained domain of w_i and $W_i \ D_i$ be the constrained domain of values for w_i. (When relevant, we call these *atomic* domains.) The *world* is the set of all possible SCMs obtained by selecting atomic parameters from their constrained domains: $w_i \ 2 \ W_i; \ 8i; 1 \ i \ n$.

The learner sees a sequence of k environments $fE_e g_{e=1}^k$.

Definition 3.2 (Environment)**.** To make an *environment* E_e, we divide the observable variables O into ones the learner cannot intervene on (states) $S^e \ O$ and ones the learner can intervene on (actions) $A^e \ O$. We then select the parameter $w_i{}^e$ from W_i for all i (1 i n).[2]

[1] U_i provides randomness for X_i. Using inverse CDF transform sampling, joint distributions can be expressed as SCMs (Buesing et al., 2019).

[2] This can be interpreted as conditional interventions, described in (Correa & Bareinboim, 2020).

In each environment, the learner learns about conditional distributions in order to make predictions. We leave the exact prediction task unspecified; instead we focus on the conditionals, and assume that discovering these will lead to task success. We represent conditional with parameter vectors:

Definition 3.3 (Description)**.** Let I and K be tuples of indices, and write X_{I} and X_{K} for the corresponding tuples of variables. The learner represents the conditional distribution of X_{I} given X_{K}, $P^{\mathsf{I}|\mathsf{K}}_{w_{\mathsf{I}|\mathsf{K}}}(\,)$, using a vector of parameters $w_{\mathsf{I}|\mathsf{K}}$.[3] Many parameter vectors might be equivalent in a given environment E_e, either because they yield identical conditional distributions, or because E_e's distribution over X_{K} puts no weight on places where they differ (see Example 3.2 below). We write $P^{\mathsf{I}|\mathsf{K}}_{w} = P^{\mathsf{I}|\mathsf{K}}_{w'}$, and we say that all of these vectors are *descriptions* of the same conditional distribution in E_e. We use a bar to denote a set of equivalent descriptions: e.g., $\bar{w}$ is the set of descriptions that are equivalent to w. If the conditional is for a single variable X_i given its parents, we call it an *atomic* description. For brevity, we sometimes write the conditional distribution as $X_{\mathsf{I}}|X_{\mathsf{K}}$ or $\mathsf{I}|\mathsf{K}$, leaving the parameters implicit.

After learning descriptions of some conditional distributions, the learner can update its estimates of the corresponding constrained domains. It can use these estimates to try to transfer to future environments. We discuss this process in more detail below (Sec. 4); there we address issues like how to reconcile descriptions learnt in different environments, and how to reason about the way different constrained domains relate to one another. To this end, we will give a toolbox of basic operations that can combine descriptions to build new ones step by step. These operations let us compute general domains from atomic ones: that is, $w_{\mathsf{I}|\mathsf{K}}$ lies in a domain $W_{\mathsf{I}|\mathsf{K}} \subseteq D_{\mathsf{I}|\mathsf{K}}$ that we can compute from the domains W_i.

As mentioned above, whether two conditional distributions for $X_{\mathsf{I}}|X_{\mathsf{K}}$ are distinct can depend on the domain of X_{K}. The following example illustrates this:

Example 3.2 (Covariate Shift). We consider the SCM $X_2 = w_2 X_1$ where $w_2 = (1; 1)$. Let D_1, the unrestricted domain of w_1, be $\{u; v\}$. If $w_1 = u$, X_1 is uniformly distributed on the line segment from $(0; 0)$ to $(1; 2)$, and if $w_1 = v$, X_1 is normally distributed on R^2. Therefore in Environment 1 when $w_1 = u$, $\bar{w}_2 = \{w \mid w(1; 2)^T = 3\}$ and in Environment 2 when $w_1 = v$, $\bar{w}_2 = \{(1; 1)\}$. Note that $\bar{w}_2$ depends on w_1 even when w_2 is chosen independently from w_1. From Environment 1, the agent might learn $\hat{W}_2 = (2; 0.5)$, which is a description of w_2 and incorrectly transfers it to the next environment.

[3]For example, with discrete variables, $w_{\mathsf{I}|\mathsf{K}}$ could be the conditional probability table; or with Gaussian variables, $w_{\mathsf{I}|\mathsf{K}}$ could be the best linear predictor.

In Section 3.2 and Section 4.3, for simplicity, we assume that each w results in a unique distribution (Assumption 3.4); for example, this happens when the joint distribution is Gaussian. We discuss the case when the assumption does not hold in Section 4.4.

Assumption 3.4 (Unique Description)**.** For any $w_1; \ldots; w_n \in D_1; \ldots; D_n$, for any two sets of disjoint indexes $\mathsf{I}; \mathsf{K}$, $\forall w', w \in D_{\mathsf{I}|\mathsf{K}}$, if $P^{\mathsf{I}|\mathsf{K}}_{w}(\cdot) = P^{\mathsf{I}|\mathsf{K}}_{w'}(\cdot)$ then $w = w'$.

3.1. Landscape around Transfer Learning

In this section, we differentiate transfer learning from related problems (Figure 1) and discuss how it relates to common concepts. Let X_{Q} be some query variables that we wish to estimate: for example, the label in supervised learning or the expected return (Q function) in reinforcement learning. To help predict X_{Q}, the learner can solve several problems:

Causal discovery: Learn the graphical structure of the underlying SCM.

Predictor selection: Choose evidence variables X_{E} to estimate $X_{\mathsf{Q}}|X_{\mathsf{E}}$.[4]

Parameter learning: Given a hypothesis set $\hat{W}_{\mathsf{Q}|\mathsf{E}}$, learn $w^e_{\mathsf{Q}|\mathsf{E}} \in \hat{W}_{\mathsf{Q}|\mathsf{E}}$.

World update: $\hat{W}^0_{\mathsf{Q}|\mathsf{E}}$ is the initial knowledge of the learner about $w_{\mathsf{Q}|\mathsf{E}}$ before any environment. Given estimated values of $w^1_{\mathsf{Q}|\mathsf{E}}; \ldots; w^e_{\mathsf{Q}|\mathsf{E}}$ learnt from previous environments $E_1; \ldots; E_e$, compute $\hat{W}^e_{\mathsf{Q}|\mathsf{E}}$ as an estimate of $W_{\mathsf{Q}|\mathsf{E}}$ to be used in parameter learning.

In this work, we focus on the world update problem, but the other problems are also important. The causal discovery problem is widely studied in the causal inference literature. Predictor selection is studied by works that focus on finding the invariant predictor (Zhang et al., 2020; Peters et al., 2015; Arjovsky et al., 2019). (Note that our framework is applicable even when there is no invariant predictor (see the Complex Colored MNIST example in Section 5.2.2).) Supervised learning and reinforcement learning methods focus on parameter learning.

Independent Causal Mechanism (ICM) principle and Sparse Mechanism Shift (SMS) hypothesis. The ICM principle (Schölkopf et al., 2021) states that the causal generative model consists of modules that do not inform or influence each other. The SMS hypothesis (Schölkopf et al., 2021) states that changes affect the causal factorization locally in-

[4]Evidence variables can be chosen to be either effects or causes of the query variables Q. For example, medical tests (which measure effects) are often performed to diagnose an illness (the cause). We discuss in Example 5.3 how choosing the evidence variables can be a trade-off between complexity and achievable accuracy.

stead of affecting all factors simultaneously. Our framework follows the ICM principle, but does not assume the SMS hypothesis. In Section 5.1 we show that the SMS hypothesis is not enough to guarantee transfer.

Few-shot and Zero-shot Learning. Zero-shot learning happens when the constrained domain learnt from previous environments contains only one point (or a small neighborhood), and therefore can be used directly in the new environment. Few-shot learning happens when the learnt domain has low complexity, and therefore it takes a small number of samples to learn the parameter.

Covariate Shift. This is a special case of our setting where the model is $X \to Y$ and the conditional distribution $P(Y \mid X)$ is the same in all environments. In equations, $Y = f_{w_Y}(X; U_Y)$; $X = f_{w_X}(U_X)$, where w_Y is the same in all environments and w_X changes in different environments.

Representation Learning. Representation learning is a special case of our setting where X is the evidence and there exists a representation variable Z. Either $Z = f_{w_Z}(X; U_Z)$ and w_Z is the same in all environments or $X = f_{w_X}(Z; U_X)$ where w_X is the same in all environments.

Transfer in Supervised Learning. There is no action variable.

Time series. We have an SCM that consists of repeated copies of a base structure, one per time step; parameters and their domains are often shared across time steps. Examples include hidden Markov models and dynamic Bayes nets.

Transfer in Reinforcement Learning. There are action variables. There may be multiple time steps. The setting includes Markov decision processes (MDPs, Puterman, 1990) when there are no hidden variables. When there are hidden variables the setting includes partially-observable MDPs (POMDPs, Kaelbling et al., 1998).

Imitation Learning. Action variables change from non-intervenable in a previous environment to intervenable in the current environment.

3.2. Feasibility of Transfer Learning

In order for transfer to be feasible, our world updates must somehow reduce the difficulty of learning in new environments. Depending on the setting, we can measure this difficulty in different ways: e.g., estimates of the number of samples needed or of the computational resources required. For example, it is common to bound the required number of samples in terms of a complexity measure such as VC dimension in supervised learning (Vapnik et al., 1994) or Bellman-Eluder dimension in reinforcement learning (Jin et al., 2021). Transfer is feasible when our updated domain $\mathcal{W}^e_{Q|E}$ improves on the original one $\mathcal{W}^0_{Q|E}$ according to the appropriate complexity measure.

We formalize this idea below. For simplicity, we assume unique generators (Assumption 3.4), and we assume **realizability**, i.e., $W_{Q|E} \in \mathcal{W}^0_{Q|E}$. We believe these assumptions can be lifted, but we leave this direction to future work.

Definition 3.5 (Transfer feasibility of the world)**.** Given a complexity measure $\mathrm{comp} : 2^{D_{Q|E}} \to \mathbb{R}$ we say that the world is transfer feasible if $\mathrm{comp}(W_{Q|E}) < \mathrm{comp}(\mathcal{W}^0_{Q|E})$.

In a transfer-feasible world, our initial knowledge of the domains is meaningfully weaker than what we might eventually learn. This doesn't guarantee that learning will actually happen; for that, we need a transfer algorithm.

Definition 3.6 (Transfer feasibility of an algorithm)**.** Let $\mathcal{W}^e_{I|K} \subseteq D_{Q|E}$ be the estimate of $W_{Q|E}$ after environments $E_1, \ldots, E_e$ output by algorithm A. Given a complexity measure $\mathrm{comp} : 2^{D_{Q|E}} \to \mathbb{R}$, if $W_{Q|E} \in \mathcal{W}^e_{Q|E}$ and $\mathrm{comp}(\mathcal{W}^e_{Q|E}) < \mathrm{comp}(\mathcal{W}^0_{Q|E})$ then A is called transfer feasible after e environments.

Zero-shot and few-shot learning are examples of transfer feasibility: they happen when $\mathcal{W}^e_{Q|E}$ is extremely simple, so that parameter learning is unnecessary (zero-shot) or takes only a small number of samples (few-shot).

4. Main Results

Following the ICM principle, the generator $w_i{}^e$'s are chosen independently from their domains W_i. In this section we show that the generator of interest, $w^e_{Q|E}$, is also chosen from a constrained domain $W_{Q|E}$ which can be computed from W_i. We present a procedure to describe how to compute the constrained domain $W_{Q|E}$ from atomic constrained domains W_i (Sections 4.1 to 4.3). Therefore, given some knowledge about the atomic constrained domains W_i's, we present a meta-algorithm to compute the constrained domain $W_{Q|E}$ (Section 4.4) for the world update problem. The meta-algorithm helps to design the architecture of the model and identify the component to transfer, which we demonstrate in Section 5.

4.1. Operations and Generators

In this section we define operations to compute a description of the distribution $Q|E$ from the atomic parameters w_i of the SCM. Using the operations, we define the computation procedure in Section 4.2.

First, we assume there exist operations to compute the product, sum-out and conditional $\cdot$, $\sum$ and $\int$ on the descriptions of the distributions. Using these operations, we show in Theorem 4.2 that we can compute a description of $Q|E$ from descriptions of X_i given its parents. Second, we show a description of X_i given its parents that only depends on w_i. Therefore, we can compute a description of $Q|E$ from

the atomic parameters w_i. In Section 4.3 we show how to use this computation to compute the constraint on this description of $\mathsf{Q}|\mathsf{E}$ from the atomic constraints W_i.

First we define the description of the distribution of X_i given its parents that only depends on w_i. We call this the generator.

Definition 4.1 (Generator)**.** Let $_i$ and ^p_i denote the domains of X_i and P_i (the parents of X_i) defined by the world. We assume that there exists a function $P^i_w(:;:) : {}_i \times {}^p_i \to \mathsf{R}$ parameterized by w such that for any $p \in {}^p_i$, $P_w(:;p) : {}_i \to \mathsf{R}$ represents the distribution of $X_i = f_w(p; U_i)$.[5] Then $P^i_{w_i}(:;p) : {}_i \to \mathsf{R}$ represents the distribution of $X_i = f_{w_i}(p; U_i)$ defined by the SCM. We call $P^i_{w_i}(:;:)$ the canonical (atomic) conditional distribution of X_i given its parents, and w_i its (atomic) generator.

Operation	Explanation
$\lvert\{w\}_{\mathsf{I};\mathsf{J}\|\mathsf{K}}\rvert = \lvert\{u\}_{\mathsf{I}\|\mathsf{J};\mathsf{K}}\rvert \lvert\{v\}_{\mathsf{J}\|\mathsf{K}}\rvert$	Computing the product $(\mathsf{I};\mathsf{J}\|\mathsf{K})$
$\lvert\{w\}_{\mathsf{I}\|\mathsf{K}}\rvert = {}_{\mathsf{J}} \lvert\{u\}_{\mathsf{I};\mathsf{J}\|\mathsf{K}}\rvert$	Summing out J
$\lvert\{w\}_{\mathsf{I}\|\mathsf{J};\mathsf{K}}\rvert = {}_{\mathsf{J}} \lvert\{u\}_{\mathsf{I};\mathsf{J}\|\mathsf{K}}\rvert$	Conditioning on J
$\lvert\{w\}_{\mathsf{I}\|\mathsf{K}}\rvert = \lvert\{u\}_{\mathsf{I}\|\mathsf{J};\mathsf{K}}\rvert \lvert\{v\}_{\mathsf{J}\|\mathsf{K}}\rvert$	Computing $\mathsf{I}\|\mathsf{K}$ from $\mathsf{I}\|\mathsf{J};\mathsf{K}$ and $\mathsf{J}\|\mathsf{K}$

Table 1. Operations. $\lvert\{w\}_{\mathsf{I}\|\mathsf{K}}\rvert$ is equivalent to $(w;\mathsf{I}\|\mathsf{K})$.

Let the expression $(u;\mathsf{I}\|\mathsf{K})$, where $u \in D_{\mathsf{I}\|\mathsf{K}}$, denote that u is a description of $\mathsf{I}\|\mathsf{K}$, and return u.

- Given $(u;\mathsf{I}\|\mathsf{J};\mathsf{K})$ where $u \in D_{\mathsf{I}\|\mathsf{J};\mathsf{K}}$ and $(v;\mathsf{J}\|\mathsf{K})$ where $v \in D_{\mathsf{J}\|\mathsf{K}}$, returns $(w;\mathsf{I};\mathsf{J}\|\mathsf{K})$ where $\mathsf{I};\mathsf{J}\|\mathsf{K}$ specifies the distribution computes. Assuming that u is a description of the distribution $\mathsf{I}\|\mathsf{J};\mathsf{K}$ and v is a description of the distribution $\mathsf{J}\|\mathsf{K}$, returns a description w of $\mathsf{I};\mathsf{J}\|\mathsf{K}$. If u is the generator of $\mathsf{I}\|\mathsf{J};\mathsf{K}$ and v is the generator of $\mathsf{J}\|\mathsf{K}$ then we call w the generator of $\mathsf{I};\mathsf{J}\|\mathsf{K}$.
- Given $(u;\mathsf{I};\mathsf{J}\|\mathsf{K})$ where $u \in D_{\mathsf{I};\mathsf{J}\|\mathsf{K}}$, ${}_{\mathsf{J}}$ returns $(w;\mathsf{I}\|\mathsf{K})$ where $\mathsf{I}\|\mathsf{K}$ specifies the distribution ${}_{\mathsf{J}}$ computes. Assuming that u is a description of the distribution $\mathsf{I};\mathsf{J}\|\mathsf{K}$, ${}_{\mathsf{J}}$ returns a description of w of $\mathsf{I}\|\mathsf{K}$. If u is the generator of $\mathsf{I};\mathsf{J}\|\mathsf{K}$ then we call w the generator of $\mathsf{I}\|\mathsf{K}$.

  Example: If $w_{\mathsf{I};\mathsf{J}\|\mathsf{K}}$ is a probability table then ${}_{\mathsf{J}}$ creates the probability table $w_{\mathsf{I}\|\mathsf{K}}$ by summing over the appropriate entries in $w_{\mathsf{I};\mathsf{J}\|\mathsf{K}}$.
- Given $(u;\mathsf{I};\mathsf{J}\|\mathsf{K})$ where $u \in D_{\mathsf{I};\mathsf{J}\|\mathsf{K}}$, ${}_{\mathsf{J}}$ returns $(w;\mathsf{I}\|\mathsf{J};\mathsf{K})$ where $\mathsf{I}\|\mathsf{J};\mathsf{K}$ specifies the distribution ${}_{\mathsf{J}}$ computes. Assuming that u is a description of the distribution $\mathsf{I};\mathsf{J}\|\mathsf{K}$, ${}_{\mathsf{J}}$ returns a description of w of $\mathsf{I}\|\mathsf{J};\mathsf{K}$. If u is the generator of $\mathsf{I};\mathsf{J}\|\mathsf{K}$ then we call w the generator of $\mathsf{I}\|\mathsf{J};\mathsf{K}$.

  Example: Let $(X;Y) \sim \mathit{Norm}(\mu_{X;Y}; \Sigma_{X;Y})$ where $\mu_{X;Y} = \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}$ and $\Sigma_{X;Y} = \begin{bmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{bmatrix}^{-1}$. Then $w_{X;Y} = (\mu_{X;Y}; \Sigma_{X;Y})$ and $w_{Y|X} = w_{Y;X} = \mu_Y - \Sigma_{YY}^{-1}\Sigma_{YX}(x - \mu_X); \Sigma_{YY}^{-1}$ (Bishop, 2006).

For convenience, we define the operation from and :

$$(u;\mathsf{I}\|\mathsf{J};\mathsf{K}) \quad (v;\mathsf{J}\|\mathsf{K}) := {}_{\mathsf{J}}((u;\mathsf{I}\|\mathsf{J};\mathsf{K}) \quad (v;\mathsf{J}\|\mathsf{K})) \quad (1)$$
$$= (w;\mathsf{I}\|\mathsf{K}): \quad (2)$$

Given $(u;\mathsf{I}\|\mathsf{J};\mathsf{K})$ where $u \in D_{\mathsf{I}\|\mathsf{J};\mathsf{K}}$ and $(v;\mathsf{J}\|\mathsf{K})$ where $v \in D_{\mathsf{J}\|\mathsf{K}}$, returns $(w;\mathsf{I}\|\mathsf{K})$ where $\mathsf{I}\|\mathsf{K}$ specifies the distribution computes. Assuming that u is a description of the distribution $\mathsf{I}\|\mathsf{J};\mathsf{K}$ and v is a description of the distribution $\mathsf{J}\|\mathsf{K}$, returns a description w of $\mathsf{I}\|\mathsf{K}$. If u is the generator of $\mathsf{I}\|\mathsf{J};\mathsf{K}$ and v is the generator of $\mathsf{J}\|\mathsf{K}$ then we call w the generator of $\mathsf{I}\|\mathsf{K}$.

We summarize the operations in Table 1. To simplify notations, we omit the indexes $\mathsf{I};\mathsf{J}$ and K from the formula and write $u \quad v$ instead of $(u;\mathsf{I}\|\mathsf{J};\mathsf{K}) \quad (v;\mathsf{J}\|\mathsf{K})$ whenever the context is clear. Similarly we write ${}_{\mathsf{J}}u$, ${}_{\mathsf{J}}u$ and $u \quad v$ without the indexes. We reuse $w_{\mathsf{I}|\mathsf{K}}$ from Definition 3.3 to denote the generator of $\mathsf{I}\|\mathsf{K}$[6].

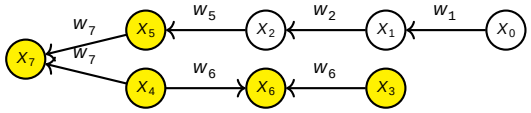


Figure 3. The causal graph and the scope of $w_{7|6,2}$ in yellow. The formula for $w_{7|6,2}$ only uses w_i of variables in the scope.

4.2. Scopes and Computation Trees

The result in this section does not depend on whether Assumption 3.4 holds. We present a theorem to compute the generator $w_{\mathsf{Q}|\mathsf{E}}$ from atomic generators w_i's using the defined operations. Using the computation procedure, we can compute the constraint of this description of $\mathsf{Q}|\mathsf{E}$ from the constraints W_i of the atomic parameters in Section 4.3.

[5] For example, with a linear-Gaussian SCM, $P^i_w(.,\boldsymbol{p})$ could be the Gaussian pdf and w could be the linear coefficient.

[6] Note that w_i denotes the generator of X_i given its parents while w_i denotes the generator marginal of X_i.

Naively we can compute $w_{Q|E}$ from the joint distribution $w_{Q;E}$, which could be calculated from the set of all w_i using variable elimination. However we derive a smaller set of variables w_i from which $w_{Q|E}$ can be computed, called the scope of $w_{Q|E}$ and denoted Scope$(Q|E)$. We illustrate the scope in Figure 3 and Figure 4. Our method recursively computes $w_{Q|E}$ by building a computation tree top-down. It starts with $w_{Q|E}$ as the root, recursively chooses a node in the tree and decomposes it into its children using the operations until all of the leaf nodes are of the form w_i. Let $w_{\text{Scope}(Q|E)} := \{w_i | i \in \text{Scope}(Q|E)\}$.

Variable elimination is a special case of our method if we perform $w_{Q|E} = \sum_E w_{Q;E}$ in the first step and then compute the joint $w_{Q;E}$ from the atomic generators using $\otimes$ and $\oplus$ in the remaining steps. We compare our method with variable elimination in Figure 4 and Appendix E.

Theorem 4.2 (Top-down computation tree, informal)**.** *$w_{Q|E}$ can be calculated from w_i's in the scope of $w_{Q|E}$ using $\oplus$; $\otimes$ and $\sum$ so that each w_i appear at most once in reverse topological order. We denote the resulting formula $g_{Q|E}(w_{Scope(Q|E)})$, also called a* computation tree *of $w_{Q|E}$*[7]*.*

For clarity, we show some examples of the computation trees in Figure 4. We also show other examples (including the computation tree for Q-learning) in App. E.

4.3. Computing Global Constraints from Local Constraints

Given the computation tree, we can compute the set $W_{Q|E}$ that $w_{Q|E}$ is chosen from. Let $W_{\text{Scope}(Q|E)} = \{W_i | w_i \in w_{\text{Scope}(Q|E)}\}$ be the corresponding constrained domain of $w_{\text{Scope}(Q|E)}$. From Theorem 4.2, $w_{Q|E}$ is computed by selecting $w_{\text{Scope}(Q|E)}$ and computing $g_{Q|E}(w_{\text{Scope}(Q|E)})$. Therefore $W_{Q|E}$ is the set of all possible $w_{Q|E}$ computed by selecting $w_{\text{Scope}(Q|E)}$ from $W_{\text{Scope}(Q|E)}$ and then applying the computation tree.

We define a notation for the set of function values over a set of inputs: for any set S and a function f such that S is in the domain of f, $f(S) := \{f(s) | s \in S\}$. Another approach to compute $g_{Q|E}(W_{\text{Scope}(Q|E)})$ in the case of supervised learning is discussed in Appendix E.4.

Corollary 4.3. *$W_{Q|E}$ can be computed from $W_{Scope(Q|E)}$ $\{W_1, \ldots, W_n\}$: $W_{Q|E} = g_{Q|E}(W_{Scope(Q|E)})$. Therefore the transfer feasibility of the world can be determined from $W_1, \ldots, W_n$ by computing comp$(W_{Q|E})$*[8]*.*

When the learner does not know the exact formula, we introduce the following properties to analyze how the local constraints can propagate throughout the network. We illustrate in Sec. 5 how to use the properties to derive constraints on the quantity we would like to estimate.

Property 4.3.1 (Propagatable). Let D_1, D_2, and $D_{1\ 2}$ be the unconstrained domain of w_1, w_2, and $w_1 \quad w_2$ respectively where $: D_1 \quad D_2 \to D_{1\ 2}$ be an operation. We define how a constraint can propagate to the left and to the right. Let $F \quad D_2 \cup D_{1\ 2}$. An operation $$ is called F-left-propagatable if for any $w_2 \in F$, $w_1 \quad w_2 \in F$. Now, let $F \quad D_1 \cup D_{1\ 2}$. An operation $$ is called F-right-propagatable if for any $w_1 \in F$, $w_1 \quad w_2 \in F$.

4.4. Transfer Process

Now we are equipped to derive a process to learn $\hat{W}^e_{Q|E}$ from $\hat{w}^1_{Q|E}, \ldots, \hat{w}^e_{Q|E}$ learnt from multiple environments $E_1, \ldots, E_e$. Let $\hat{W}_i$'s for some $i; 1 \quad i \quad n$ be some local information the learner knows about W_i (if the learner does not know anything about W_i it can assume $\hat{W}_i = D_i$). Let $\hat{\text{Scope}}(Q|E)$ be an estimation of Scope$(Q|E)$ such that Scope$(Q|E) \quad \hat{\text{Scope}}(Q|E)$. Without any knowledge we can set $\hat{\text{Scope}}(Q|E)$ to be all variables. Let $\hat{g}_{Q|E}$ be an approximation of $g_{Q|E}$ obtained by knowledge or by the propagation properties such that $g_{Q|E}(W_{\text{Scope}(Q|E)}) \in \hat{g}_{Q|E}(W_{\hat{\text{Scope}}(Q|E)})$ for any input W. For example, if all operations are F-right propagatable and the left-most term is in F, $\hat{g}_{Q|E}$ can output F. We call a set W parameterized by a parameter if W is determined by . For example, if W is the set of all w such that $f(w) = 0$ for a known class of function f, then W is parameterized by . Algorithm 1 captures our procedure to compute $\hat{W}^e_{Q|E}$. In App. A, we describe an algorithm to learn the constraint from the approximations of the parameters in previous environments.

If Assumption 3.4 does not hold, Algorithm 1 needs to be modified. Consider the case in Example 3.2, when $w_2 = (1;1)$ but $\overline{w}_2 = \{w : w^T(1;2) = 3\}$, and w_2 is the same in all environments. From Environment 1, the agent might learn $\hat{w}_2 = (2;0.5)$, which is a description of w_2 and incorrectly transfers it to the next environment. Instead, for all environments e, the agents need to learn the set of all possible descriptions in $\hat{W}^e_{Q|E}$, therefore in this case $\hat{w}^i_{Q|E}$ does not denote just a point but a set of all possible, equivalent descriptions. In Example 3.2, it would be equivalent to learn $\overline{w}_2$ from Environment 1. The learner than uses the set of all possible descriptions $\hat{w}^i_{Q|E}$ learned from previous environment to learn the set of such that $\forall \quad 2 : \forall i = 1; \quad ; e; \exists w^i \in \hat{w}^i_{Q|E}$ such that $f(w^i) = 0$. Let $\hat{W}^e_{Q|E} = [_2 \{w : f(w) = 0\}$. Then the agent will learn the set of all possible descriptions $\hat{w}^{e+1}_{Q|E}$ in $\hat{W}^e_{Q|E}$.

[7] There can be multiple computation trees depending on the choice the learner made.

[8] The second statement of Collorary 4.3 assumes Assumption 3.4 holds while the first statement does not depend on Assumption 3.4.

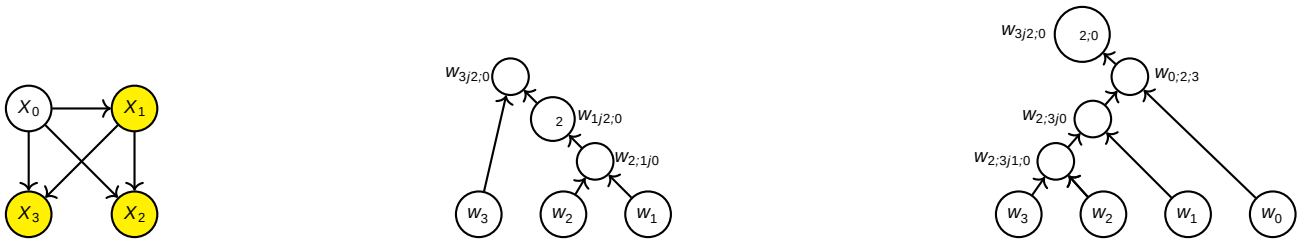


Figure 4. Causal graph and the computation trees for $w_{3j0,2}$. For clarity, we do not draw the uniform noise variables. **(Left)**: Causal graph and the scope of $w_{3j2,0}$ in yellow. **(Middle)** A computation tree of $w_{3j2,0}$ constructed from Thm. 4.2, which is equivalent to calculating $p(X_3 j X_2, X_0) = \sum_{x_1} p(X_3 j X_0, X_1 = x_1) p(X_1 = x_1 j X_2, X_0)$; $p(X_1 = x_1 j X_2, X_0) = \frac{p(X_1 = x_1, X_2 j X_0)}{\sum_{x_1^0} p(X_1 = x_1^0, X_2 j X_0)}$ and $p(X_1 = x_1, X_2 j X_0) = p(X_2 j X_0, x_1) p(x_1 j X_0)$ for discrete distributions. Variable elimination cannot construct this computation tree because conditioning ( ) is performed to compute $p(X_1 = x_1 j X_2, X_0)$ before sum-out and product ( ). w_3 goes through only 1 operation. **(Right)** A computation tree of $w_{3j2,0}$ constructed by naive variable elimination on the joint distribution $w_{0,2,3}$. Each leaf goes through at least 2 operations (at least one to construct the joint distribution and then the conditional).

Algorithm 1 Meta-Algorithm for Transfer

Input: Estimated constrained domain $\hat{\mathcal{W}}_1, \ldots, \hat{\mathcal{W}}_n$; parameters learnt from previous environments $\hat{w}^1_{Qj\mathsf{E}}, \ldots, \hat{w}^e_{Qj\mathsf{E}}$; estimated Scope$(Qj\mathsf{E})$; estimated computation tree $\hat{g}_{Qj\mathsf{E}}$ to compute $w_{Qj\mathsf{E}}$ from w_i's

Output: $\hat{\mathcal{W}}^e_{Qj\mathsf{E}}$, an estimation of $\mathcal{W}_{Qj\mathsf{E}}$

Calculate $\hat{\mathcal{W}}^e_{Qj\mathsf{E}} = \hat{g}_{Qj\mathsf{E}}(\mathcal{W}_{\text{Scope}(Qj\mathsf{E})})$ parameterized by unknown parameter ;

Learn using $\hat{w}^1_{Qj\mathsf{E}}, \ldots, \hat{w}^e_{Qj\mathsf{E}}$ and compute $\hat{\mathcal{W}}^e_{Qj\mathsf{E}}$ from .

5. Case Studies

We now demonstrate how our framework can be used to analyze widely-held beliefs about transfer learning.

5.1. Sparse Mechanism Shift

The Sparse Mechanism Shift (SMS) Hypothesis states that changes are often sparse and local in a causal graph (Schölkopf et al., 2021; Lopez et al., 2023; Lachapelle et al., 2022; Bereket & Karaletsos, 2024). We demonstrate that SMS does not always imply transfer feasibility and vice versa.

Example 5.1 (The Chain Problem). Assume X_0 is normally distributed on $\mathbb{R}^2$. For $i = 1, \ldots, 100$, $X_i = f_{w_i}(X_{i-1})$. $X_1, \ldots, X_{99}$ are unobserved. Consider the linear model where $f_{w_i}(X_{i-1}) = w_i X_{i-1}$. The learner observes $X = X_0$ and $Y = X_{100}$ and would like to predict Y from X.

The learner fits a linear model $Y = wX$ to predict Y from X, where w is any 2 2 matrix, making $\mathcal{W}^0_{100j0}$ the set of all 2 2 matrices. The complexity measure comp is the number of parameters that need to be learned, with $\text{comp}(\mathcal{W}^0_{100j0}) = 4$. App. F shows that if w_i ($82 \quad i \quad 99$) is the same in all environments (sparse changes), transfer learning may not be feasible. However, if only w_1 is the same in all environments (non-sparse changes), transfer may be feasible, reducing $\text{comp}(\mathcal{W}^1_{100j0})$ — the dimension of the hypothesis class after environment 1 — to 2.

5.2. Freezing Layers of Neural Networks

Freezing a layer of a neural network learnt from previous environments is standard in transfer learning (Yosinski et al., 2014; Long et al., 2015; Kirichenko et al., 2023; Tajbakhsh et al., 2016; Guo et al., 2019). Our framework helps to explain when this practice works (Section 5.2.1). In the next few subsections, we show that this standard practice may be ill-advised: performance can depend strongly on exactly which parameters we freeze.

5.2.1. The Chain Problems

We sketch an example where freezing layers of a NN is correct for transfer learning – see Appendix F for more details. We use the result in (Rolnick & Kording, 2020), where it was shown that if the Linear Region Assumption is satisfied, ReLU networks can be identified up to permutation and scaling.

Example 5.2 (Freezing layers for neural networks). Modify the SCM in Example 5.1 so that f_{w_i} is a single ReLU layer with weights w_i. Assume that in all environments E_e, the Linear Region Assumption (Rolnick & Kording, 2020) is satisfied and that X_0's distribution satisfies that the description set of w^e_{100j0} consists of only equivalent neural networks up to permutation and scaling. is stacking layers: for $i > j > k$, $w_{ijk} = w_{ijj} \quad w_{jjk} = (w_{ijj}, w_{jjk})$. Therefore the generator $w_{100j0} = (w_{100}, \ldots, w_1)$. For any i, if w_i is the same in all environments, $\mathcal{W}_{100j0}$ is the set of $(w_{100}, \ldots, w_1)$ up to permutation and scaling with the same w_i in all environments. Thus, we can learn layer w_i from the first environment and freeze it in later environments.

We show in App. F.2 that even with slight modifications from the above example, no longer corresponds to an entire layer, and therefore freezing a layer results in poor performance. In the next section, we present a more complex example to explore this phenomenon.

5.2.2. The Complex Colored MNIST Problem

We now show when freezing an entire layer of the network leads to poor transfer, while freezing a subset of the layer leads to successful transfer. This example also demonstrates our approach where we treat transfer learning as a new problem separated from finding an invariant predictor which are often pursued in previous works (Zhang et al., 2020; Peters et al., 2015; Arjovsky et al., 2019). It also demonstrates a case of *representation learning*, where the distribution of the image X given its parents is assumed to be the same in all environments.

Example 5.3 (Complex Colored MNIST). This binary classification problem is based on MNIST: instances are colorized images of digits, and the colors are spuriously correlated with labels. In the original dataset (Arjovsky et al., 2020), binary labels are based on the digit with probability 0.25 and flipped otherwise. In 2 environments, the image is colored with a color that correlated with the binary label with probability 0.1 and 0.9 respectively. We modify the environments to make it more difficult. In E_1, the correlation between digit identity and label is 0.1, and the correlation between color and label is 0.2. In E_2, the correlation is 1 between digit and label, and 0.4 between color and label.

We illustrate the trade-off between achievable accuracy and sample complexity for choosing different predictors (Section 3.1) using the original Colored MNIST problem. Predicting the label using the digit in Environment 1 allows zero-shot learning in Environment 2 with a maximum accuracy of 0.75. Predicting the label using the color in Environment 2 requires samples from Environment 2 but enables a maximum theoretical accuracy of 0.9. Therefore, predictor choice affects the achievable accuracy and sample complexity in the new environment. In the Complex Colored MNIST problem, the label distribution given the digit changes across environments, making previous methods (Zhang et al., 2020; Peters et al., 2015; Arjovsky et al., 2019) that identify the invariant predictors for zero-shot learning ineffective.

Transfer Algorithm. Let (X) denote the flattened vector of X. We first derive the model for $W^1_{Y|X}$. We assume that (1) the parents P_X of the image X are discrete random variables (2) conditioning on $P_X = p$, (X) is in the exponential family $p(x|p) = \frac{1}{s}h(\frac{1}{s}x)\, g(\ _p)\exp(\frac{1}{s}\ ^T_p x)$ where the scale parameter s does not depend on p (Section 4 in (Bishop, 2006)); (3) X is d-separated from Y given P_X, (4) the distribution $X|P_X$ is unchanged in all environments.[9]

Via Theorem 4.2 we compute $P(Y|X)$ (Appendix F.3):

$$P(Y|X) = \frac{\sum_p \exp(\frac{1}{s}\ ^T_p(X) + \log g(\ _p) + \log P(p;Y))}{\sum_y \sum_p \exp(\frac{1}{s}\ ^T_p(X) + \log g(\ _p) + \log P(p;y))} \tag{3}$$

where $\sum_p$ denotes the summation over all possible values p of the parents of X. The blue terms are affected by w_X while the black terms are not. Eq. 3 can be implemented as a 1-layer network, but in this implementation, freezing a layer is the same as freezing the entire model. This would lead to poor performance because the label in Environment 2 is almost the opposite of Environment 1. So, to allow more flexibility in freezing and give the competing model where the entire layer is frozen a better advantage, we derive a 2-layer implementation of Eq. 3. Rearranging the terms we get the following 2-layer network: the first layer implements

$$P(p|X) = \frac{\exp(\frac{1}{s}\ ^T_p(X) + \log g(\ _p) + \log P(p))}{\sum_p \exp(\frac{1}{s}\ ^T_p(X) + \log g(\ _p) + \log P(p))};$$

and the second layer implements

$$P(Y|X) = \sum_p P(p|X)\log P(Y|p):$$

To learn, we make two additional assumptions: (5) for any choice of the parameters in the constrained domain, the description set of the any network in the constrained domain is the set of all equivalent weights in which the linear coefficients differ only by permutation (the biases could differ by other means) and (6) the optima of the loss function are in the description set.

Given these assumptions, our method recommends transferring by freezing the linear coefficient in the first layer: from (6), minimizing the loss will result in a description. From (5), similar to the discussion in Section F.2, we can freeze the first layer's linear coefficient to transfer. This reflects the fact that the linear coefficient represents $P(X|P_X)$, which is the same in all environments. We compare to other transfer strategies that might seem reasonable without our derivations, such as freezing the entire first layer.

Transfer Feasibility of the Algorithm. Let the complexity measure comp be the number of parameters to be learned. Assuming that $n_p = 20$, without transfer learning, comp $= 47040 + 20 + 40$; with transfer learning, comp $= 20 + 40$. The details are provided in Appendix F.3.

Experiment. Fig. 5 compares transfer performance of several methods: learning the model in the new environment without transfer, freezing the first layer of the two-layer

[9] As an example, a detailed hypothetical model that satisfies our assumptions is given in Appendix F.3.

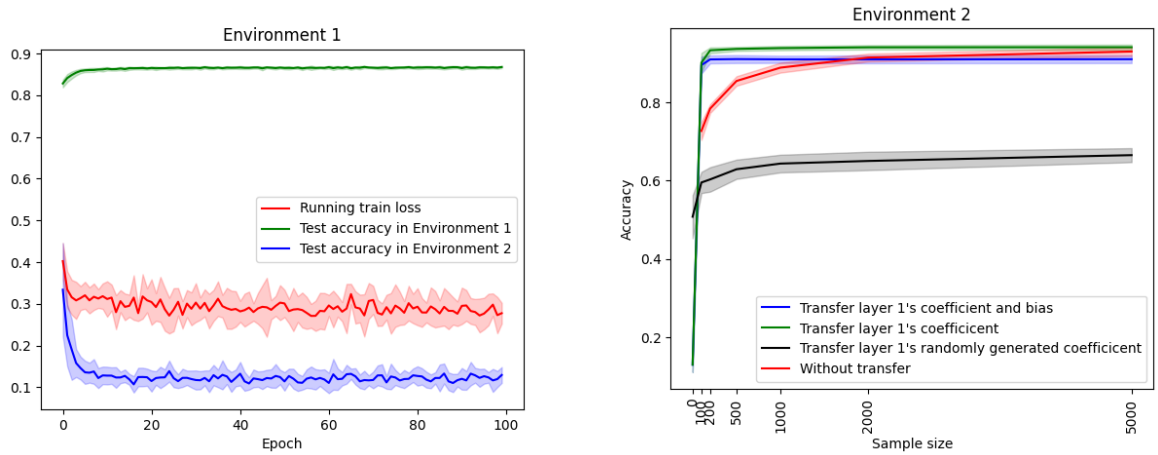


Figure 5. The Complex Colored MNIST problem, Ex. 5.3. **(Left)**: Zero-shot transfer performance of model trained in Env. 1. Accuracy in Env. 2 is poor because the labels in Env. 1 and Env. 2 are very different. **(Right)** Performance of various transfer methods as they train on Env. 2. Transferring the first layer's linear coefficient results in the best performance.

model, and freezing just the linear coefficient of the first layer. App. F.3 gives implementation details using Ray Tune (Liaw et al., 2018). We show that the strategy recommended by our derivation (freezing just the linear coefficient of the first layer) performs best. We also compare to a random "transfer" strategy (freezing the first layer's linear coefficient to a random value instead of a learned one) to provide a baseline. Note that when the sample size available in Env. 2 grows larger than in the plot (5000) it might be possible that the model without transfer might outperform the model that transfers layer 1's coefficient due to limited samples in the first environment, failure of the model to find the true coefficient in the first environment or other factors.

5.3. Reinforcement Learning

We now illustrate how our framework applies to reinforcement learning with a simple example. Suppose the dynamics $P(s_{t+1}|s_t; a_t)$ remain constant across environments, and the expected reward is $E[R_t|s_t; a_t] = w^T \quad (s_t; a_t)$, where is constant across environments and w varies (Barreto et al., 2017). Suppose that states and actions are discrete, so that the dynamics are a table of size $n_s \quad n_s n_a$ and is a table of size $d \quad n_s n_a$, where $n_s; n_a$ and d are the number of states, actions and dimension of . Observe that the Q function of any policy in any environment can be written as $Q \quad (s; a) = E[R_i|s; a] + \quad E[V(s_{t+1}|s; a)]$. Considered as a vector of length $n_s n_a$ indexed by $(s; a)$, the first term on the RHS is in the row space of the table, and the second term is in the row space of the dynamics table. So Q is in the span of these two row spaces.

Suppose there are $n_a = 4$ actions and $n_s = 5$ states, and has dimension $d = 2$. Then Q (of dimension 20) is in a subspace of dimension $n_s + d = 7$. Once we evaluate enough policies in different environments, we can a basis for this subspace. In a new environment, we can constrain $Q(s; a)$ to be in this subspace, potentially saving both data and computation: e.g., we can estimate the environment's w from observations (saving data compared to estimating the vector $E[R(s; a)]$), then run value iteration or Q iteration entirely in the subspace (saving data by not estimating the dynamics, and computation by working in the subspace). Barreto et al. (2017) compute an improved policy but not the optimal policy in the new environment. Brantley et al. (2021) compute the optimal policy, but do not address learning.

6. Conclusion

Our paper provides a framework for transfer learning that can guide how to perform transfer and analyze if common heuristics such as freezing a layer may or may not succeed. Our framework can be used to model a wide range of transfer scenarios including those in supervised learning, imitation learning and reinforcement learning. It can also help users design models that enable transfer or select components to transfer.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. Invariant risk minimization, 2019. URL https://arxiv.org/abs/1907.02893.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. Invariant risk minimization, 2020.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H., and Silver, D. Successor features for transfer in reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 4058–4068, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Beery, S., Van Horn, G., and Perona, P. Recognition in terra incognita. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Bereket, M. and Karaletsos, T. Modelling cellular perturbations with the sparse additive mechanism shift variational autoencoder. *Advances in Neural Information Processing Systems*, 36, 2024.

Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Brantley, K., Mehri, S., and Gordon, G. J. Successor feature sets: Generalizing successor representations across policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11774–11781, 2021.

Buesing, L., Weber, T., Zwols, Y., Heess, N., Racaniere, S., Guez, A., and Lespiau, J.-B. Woulda, coulda, shoulda: Counterfactually-guided policy search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJG0voC9YQ.

Chen, Y. and Bühlmann, P. Domain adaptation under structural causal models. *The Journal of Machine Learning Research*, 22(1):11856–11935, 2021.

Correa, J. and Bareinboim, E. A calculus for stochastic interventions:causal effect identification and surrogate experiments. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):10093–10100, Apr. 2020. doi: 10.1609/aaai.v34i06.6567. URL https://ojs.aaai.org/index.php/AAAI/article/view/6567.

Dietterich, T. G. Hierarchical reinforcement learning with the MAXQ value function decomposition, 1999. URL https://arxiv.org/abs/cs/9905014.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. Domain-adversarial training of neural networks, 2016.

Gulrajani, I. and Lopez-Paz, D. In search of lost domain generalization. *CoRR*, abs/2007.01434, 2020. URL https://arxiv.org/abs/2007.01434.

Guo, Y., Shi, H., Kumar, A., Grauman, K., Rosing, T., and Feris, R. Spottune: Transfer learning through adaptive fine-tuning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4800–4809, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society. doi: 10.1109/CVPR.2019.00494. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00494.

Huang, B., Feng, F., Lu, C., Magliacane, S., and Zhang, K. Adarl: What, where, and how to adapt in transfer reinforcement learning, 2021. URL https://arxiv.org/abs/2107.02729.

Jin, C., Liu, Q., and Miryoosefi, S. Bellman eluder dimension: New rich classes of rl problems, and sample-efficient algorithms, 2021.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Kirichenko, P., Izmailov, P., and Wilson, A. G. Last layer re-training is sufficient for robustness to spurious correlations. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Zb6c8A-Fghk.

Lachapelle, S., Rodriguez, P., Sharma, Y., Everett, K. E., Le Priol, R., Lacoste, A., and Lacoste-Julien, S. Disentanglement via mechanism sparsity regularization: A new principle for nonlinear ica. In *Conference on Causal Learning and Reasoning*, pp. 428–484. PMLR, 2022.

Li, X., Grandvalet, Y., and Davoine, F. Explicit inductive bias for transfer learning with convolutional networks. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2825–2834. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/li18a.html.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

Long, M., Cao, Y., Wang, J., and Jordan, M. I. Learning transferable features with deep adaptation networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 97–105. JMLR.org, 2015.

Lopez, R., Tagasovska, N., Ra, S., Cho, K., Pritchard, J., and Regev, A. Learning causal representations of single cells via sparse mechanism shift modeling. In *2nd Conference on Causal Learning and Reasoning*, 2023. URL https://openreview.net/forum?id=IOWJsPJ2xGd.

Magliacane, S., van Ommen, T., Claassen, T., Bongers, S., Versteeg, P., and Mooij, J. M. Causal transfer learning. *CoRR*, abs/1707.06422, 2017. URL http://arxiv.org/abs/1707.06422.

Myung, S., Huh, I., Jang, W., Choe, J. M., Ryu, J., Kim, D., Kim, K.-E., and Jeong, C. PAC-net: A model pruning approach to inductive transfer learning. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16240–16252. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/myung22a.html.

Pearl, J. and Bareinboim, E. Transportability of causal and statistical relations: A formal approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pp. 247–254, 2011.

Pearl, J. et al. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 19(2), 2000.

Peters, J., Bühlmann, P., and Meinshausen, N. Causal inference using invariant prediction: identification and confidence intervals, 2015.

Puterman, M. L. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Rolnick, D. and Kording, K. Reverse-engineering deep ReLU networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8178–8187. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/rolnick20a.html.

Saengkyongam, S., Thams, N., Peters, J., and Pfister, N. Invariant policy learning: A causal perspective. *CoRR*, abs/2106.00808, 2021. URL https://arxiv.org/abs/2106.00808.

Schölkopf, B., Locatello, F., Bauer, S., Ke, N. R., Kalchbrenner, N., Goyal, A., and Bengio, Y. Toward causal representation learning. *Proceedings of the IEEE*, 109(5): 612–634, 2021.

Shen, K., Jones, R., Kumar, A., Xie, S. M., HaoChen, J. Z., Ma, T., and Liang, P. Connect, not collapse: Explaining contrastive learning for unsupervised domain adaptation, 2022.

Sohn, K., Berthelot, D., Li, C.-L., Zhang, Z., Carlini, N., Cubuk, E. D., Kurakin, A., Zhang, H., and Raffel, C. Fixmatch: Simplifying semi-supervised learning with consistency and confidence, 2020.

Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., and Liang, J. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35:1299–1312, 2016. URL https://api.semanticscholar.org/CorpusID:32710.

Vapnik, V., Levin, E., and Le Cun, Y. Measuring the vc-dimension of a learning machine. *Neural computation*, 6 (5):851–876, 1994.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/375c71349b295fbe2dcdca9206f20a06-Paper.pdf.

Zhang, A., Lyle, C., Sodhani, S., Filos, A., Kwiatkowska, M., Pineau, J., Gal, Y., and Precup, D. Invariant causal prediction for block MDPs. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11214–11224. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/zhang20t.html.

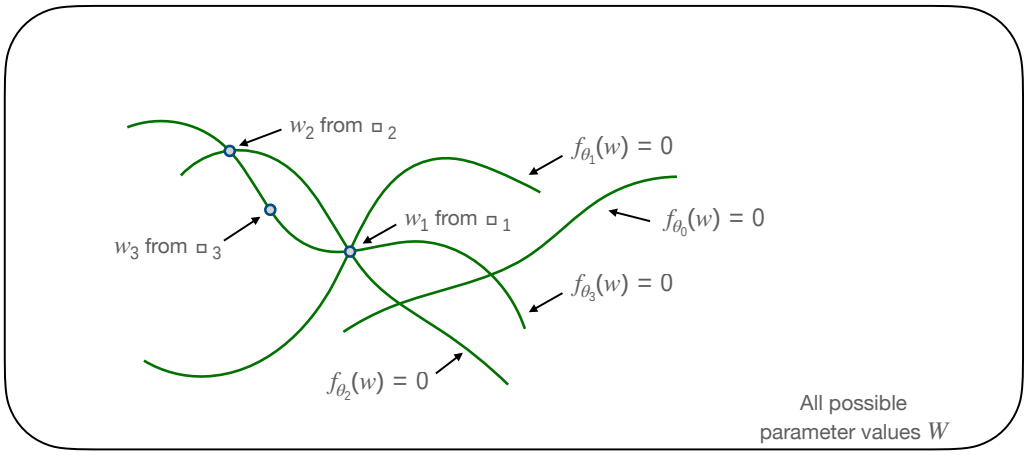


Figure 6. Constraint learning problem without noise. Each marker represents a possible environment parameter (a value of w), and each teal curve represents a possible constraint (a value of θ).

Supplementary Material: When is Transfer Learning Possible?

In Section A we describe an algorithm to learn the constraint using w^e's from previous environment. In Section B we discuss related works. In Section C we discuss future works. In Section D we discuss more details of the setting related to reinforcement learning. In Section E we provide details for Section 4 including the proof of Theorem 4.2 and more examples of computation trees including Q-Learning. In Section F we provide details for the case studies (Section 5) including the details of the Colored MNIST experiments. In Section G we discuss the limitation of our paper.

A. Constraint Learning

In this section we describe a simple method for learning constraints from examples. This is an important prerequisite for transfer learning: the tools we describe in the main paper can help us propagate constraints, so that we know how a constraint on one parameter of our model might imply a constraint on other parameters, but we have to combine this sort of propagation with a way to extract constraints from observed data.

There are many ways that one might hope to improve on the constraint learning method we describe here. The exact method of constraint learning is not central to the current paper; all that is important here is that some learning algorithm proposes constraints that fit into our tools for transfer. So, our intent is not to be state-of-the-art, but merely to demonstrate that this kind of learning is possible. We leave it to future work to refine and optimize our learning method.

We have two main goals in designing our constraint learning method. First, we'd like to be agnostic to how the environments are generated: for example, we don't want to assume that they are sampled i.i.d. from a fixed distribution over the set of allowed environments. Second, we'd like to maximize the benefits we see from transfer: we'd like to learn helpful constraints that narrow down the set of parameters that we have to learn from, but we'd like to avoid costly mistakes where incorrect constraints limit performance in new environments.

We'll build up the constraint learning problem in several steps, introducing new issues at each step. The first version of our learning problem is illustrated in Figure 6. We are trying to learn some parameter w that takes a different (unknown) value w_i in each new environment E_i. We assume a family of possible constraints that might apply to w: $f(w) = 0$, where is an (unknown) parameter vector. If we want to enforce multiple constraints on w, we can have f return a vector; in this case contains the parameters of all constraints, concatenated together.

As we gain experience in a sequence of environments $E_1; E_2; \ldots$, we can hope to learn a constraint that describes the feasible values of $w_1; w_2; \ldots$. For example, after we see data from E_1, we can learn an estimate of w_1, which lets us narrow down the possible values of . In the figure, $_0$ is inconsistent with w_1, since w_1 does not lie on the curve $f_0(w) = 0$. On the other hand, $_1$, $_2$, and $_3$ are all consistent with w_1.

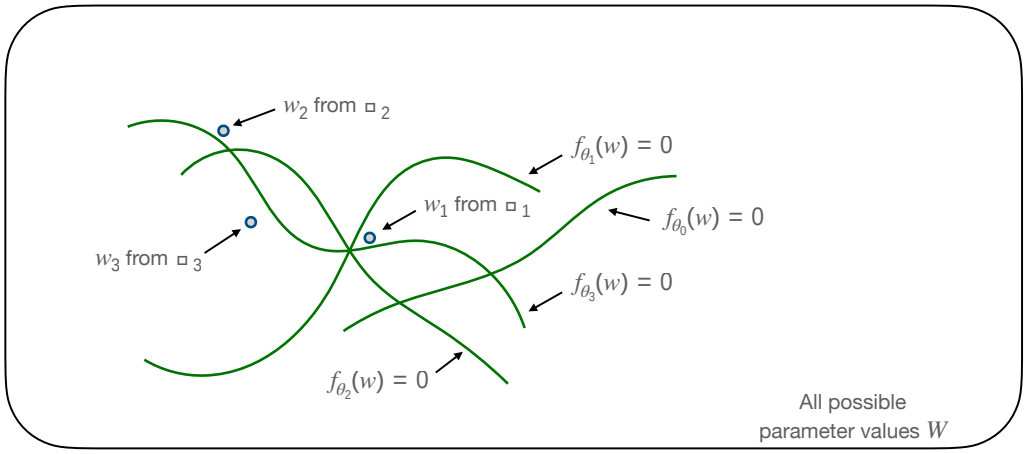


Figure 7. Constraint learning problem with noise.

As we see additional environments, we continue to narrow down the possible values of . If we next learn w_2 by seeing data from environment E_2, we can rule out $_1$: the constraint $f_{\ 1}(w) = 0$ passes through w_1 but not w_2. If we then learn w_3 by visiting environment E_3, we rule out $_2$, leaving $f_{\ 3}(w) = 0$ as the only one of the illustrated constraints that is still consistent. If we assume that $_3$ is the correct constraint, then in E_4 we can use this information to learn w_4 faster: the set $f_{\ 3}(w) = 0$ has lower dimension than the full parameter space W, so we will need fewer samples to distinguish the value of w_4 that is the best fit within the constraint.

If we leave aside computational considerations, we can write a very simple algorithm that learns the correct constraint in our setup so far: we just keep track of which values of are still consistent after seeing $w_1; w_2; \ldots$. When only a single value of remains consistent, we are done.

Of course, this algorithm will not work very well in practice, since we have ignored an important aspect of constraint learning: due to estimation error, we will not get a perfect estimate of w_i from any finite amount of data in E_i. So, the picture will look more like Figure 7: the true constraint $f_{\ 3}(w) = 0$ will not pass exactly through our estimates $w_1; w_2; w_3$, but will only pass close to them.

To handle estimation errors like this, we propose a simple no-regret algorithm. We start from a set of possible constraint parameters. Before seeing each environment E_i, we propose a hypothesis $_i \ 2$ as described below. Then we find out w_i, and we get a penalty

$$P_i(\ _i) = kf_{\ _i}(w_i)k^2$$

This penalty is zero if and only if w satisfies the constraint $f\ (w_i) = 0$. If we fail to satisfy the constraint, then $P_i(\ )$ is strictly positive, with a larger value if we are farther from satisfying the constraint.

Our goal is to get a small total penalty $P(\ ) = \sum_i P_i(\ )$. To accomplish this, we can choose $_i$ using any no-regret algorithm — that is, any algorithm where the regret

$$\text{regret} = \max_{2} \left[\sum_i P_i(\ _i) \quad P(\ ) \right]$$

grows sublinearly in T.

Depending on the sort of guarantees we want, we can pick our no-regret algorithm one of two ways. The first possibility is a learner like online gradient descent. Depending on the form of f , OGD gives us different guarantees: if we have linear constraints, then $P_i(\ )$ will be convex, and we can get convergence to the regret of the global optimum. On the other hand, for general f , the functions $P_i(\ )$ may not be convex, in which case we might get stuck in local optima. Either way, OGD

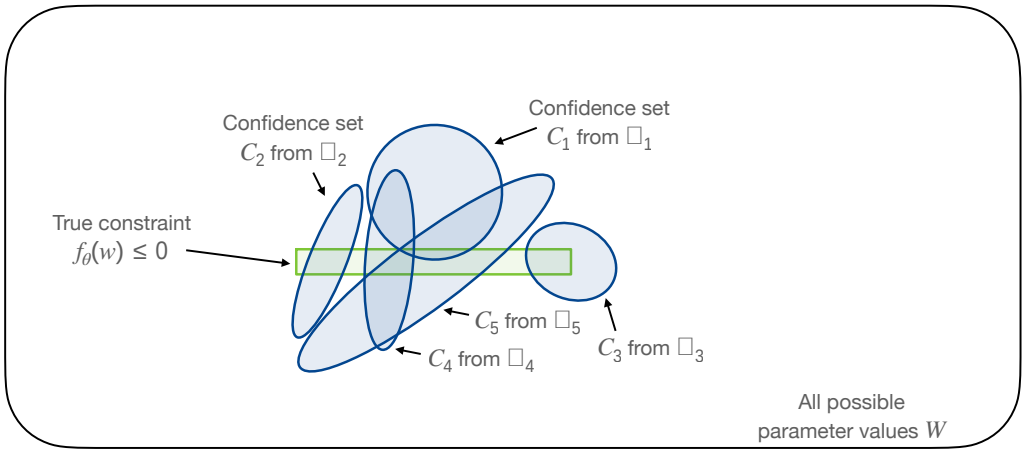


Figure 8. Constraint learning problem with inequality constraints and confidence sets.

is computationally cheap: the main cost is computing the gradient or subgradient of the penalty $P_i(\ )$.

The second possibility is to choose a no-regret algorithm that has strong guarantees in the face of non-convex learning problems. One such approach is to tile the space of constraints with an -net, and run a no-regret method such as multiplicative weights to pick among all of the points in the net. The size of the -net will be exponential in the dimension of , so this approach will typically only be computationally feasible for low-dimensional constraint sets.[10] On the other hand, we will get a much stronger guarantee: for MW, after T environments, we will propose at least one parameter $_i$ whose penalty is at most

$$P(\ _i) \quad P(\ ) + O(\sqrt{\frac{}{T}} \ln j \quad j)$$

where is the best (post-hoc) parameter value in . Therefore, the average penalty $P(\ _i)=T$ for the best proposal will approach the optimal value $P(\ )=T$ as $T \ ! \ 1$.

No matter how we choose our no-regret learner, as our estimate $_i$ improves, we become more and more likely to see a benefit from transfer. But early in the learning process we may see *negative* transfer: if we enforce a poorly-estimated constraint, it can *reduce* our performance at estimating w_i in some environment E_i. To guard against negative transfer, we can calculate both the best constrained w_i (the one that minimizes loss subject to $f_{\ i}(w_i) = 0$) as well as the best unconstrained w_i (the one that minimizes loss over the entire parameter space W). We can then choose between the constrained and unconstrained estimates: e.g., we can run an additional no-regret learner inside each new environment to predict w_i nearly as well as the better of the two estimates.

At this point we have a constraint learning method that can work well in practice. But, there are two additional wrinkles that can affect performance. So, for the final version of our constraint learning problem, we'll make two changes to our setup.

First, instead of just equality constraints of the form $f\ (w) = 0$, we'll allow inequality constraints like $f\ (w) \quad 0$. This makes it easier for us to represent a wider variety of constraint shapes. Second, we will let our learning algorithm give us more information: instead of just a point estimate w_i, we will let it return a confidence set C_i, with the intention that C_i covers the correct value of w_i with high probability. (We can still handle point estimates by treating them as singleton sets.)

With these changes, the picture looks like Figure 8. The biggest change here is that there is a more complicated relationship between the confidence sets and the constraint set. First, there will likely be parameter values in C_i that don't satisfy the constraint, due to limited data in environment E_i. And second, there will likely be parameter values that satisfy the constraint but fall outside of C_i, since the constraint describes possible parameter values for *all* environments, not just E_i.

To handle this situation, we can make some simple changes to our no-regret algorithm. First, the penalty for proposing $_i$

[10] If there is structure that we can take advantage of in the set of constraints , we might be able to get the best of both worlds: computational efficiency and strong guarantees. We leave this sort of potential extension to future work.

becomes the penalty for the best value of w that falls within C_i. For an equality constraint, this is

$$P_i(\ ) = \min_{w \in C_i} k f\ (w) k^2$$

Second, if some coordinates of f correspond to inequality constraints, we threshold so that negative values of f are not penalized. If we write for the thresholding function, so that

$$_j(z) = \begin{array}{ll} z_j & \text{if } z_j \quad 0 \text{ or coordinate } j \text{ is an equality constraint} \\ 0 & \text{otherwise} \end{array}$$

then the penalty is

$$P_i(\ ) = \min_{w \in C_i} k\ \ (f\ (w)) k^2$$

Finally, inequality constraints require a bit of additional care: different inequalities can be stronger or weaker, and therefore more or less desirable to learn. If we ignore this issue, we will tend to learn very weak constraints: in the limit, the trivial constraint that allows all of W will get zero penalty no matter what C_i is. (This issue is also present with some kinds of equality constraints, since we can equivalently incorporate into f and treat every constraint as an equality; but essentially every family of inequality constraints needs to take account of this problem.)

Ideally we would directly measure the strength of transfer that each constraint provides: tighter constraints will get higher $P_i(\ )$ but have the potential for stronger transfer. Unfortunately, this sort of measurement could be expensive; so, we leave it to future work to explore this avenue. Instead, here we suggest modifying $P_i(\ )$ by adding an extra penalty that measures the weakness of the constraint that corresponds to : e.g., we could use a penalty proportional to the volume covered by the constraint. The scaling of such a penalty adds a hyperparameter; we hope that future work might show a way to eliminate the need for such a hyperparameter, or at least tune it at low cost.

To summarize, our final learning method is to run a no-regret algorithm over the set of possible constraint parameters. For each environment E_i, this outer-loop algorithm predicts a constraint $_i$. Given $_i$, we run two learners for w_i: one that enforces the predicted constraint, and one that is unconstrained. We switch between the two estimates of w_i using an inner-loop no-regret algorithm. We then use the unconstrained estimate of w_i to find a confidence set C_i, and pass this confidence set back to our outer-loop no-regret algorithm. To calculate this algorithm's loss, we find the most favorable value of w within C_i: the value that minimizes the penalty $k\ \ (f\ _i(w)) k^2$. Optionally, if different values of correspond to tighter or looser constraints, we add an additional penalty to the outer algorithm's loss, in order to favor tighter constraints.

B. Related Works

There are a number of previous works that implicitly follow a special case of our setting and that approach our problem differently (e.g., with qualitatively different assumptions or goals). Huang et al. (2021) is a special case of our setting. This work assumes that a small set of variables is intervened on and therefore represents the changes in different environments. Therefore, they freeze the rest of the model learnt from previous environments, and only learn the changes in the new environment. Li et al. (2018); Myung et al. (2022) consider the case where there is limited data in the target domain but ample data in the source domain. Both papers assume we know what to transfer across environments, while our framework allows us to explicitly determine that from expert knowledge or the data.

MaxQ (Dietterich, 1999) approaches transfer differently: it is concerned with both *computational* and *statistical* benefits. Their setting for statistical benefits is reminiscent of ours: they propose a way to constrain the parameters of a policy in a new environment by freezing part of a "MAXQ hierarchy." This same strategy also provides computational benefits, since decision-theoretic planning can be extremely expensive if we have to start from scratch. Our setup could address this benefit if we had a practical measure of the computational complexity of RL in different hypothesis classes, but such a measure is a subject for future research.

Deep Learning Heuristics for Transfer. Several heuristics for learning representations for domain adaptation have been proposed in the literature, including adversarial training (Ganin et al., 2016), self-supervised techniques (Sohn et al., 2020),

and contrastive learning (Shen et al., 2022). In contrast to these works, we focus on devising a framework via a causal perspective. However, it would be very interesting to apply our framework to these ideas.

Learning Invariant Features. Various works have explored the idea of learning "invariant" representations that use information available in all environments with the hope that this will lead to better transfer to downstream tasks (Arjovsky et al., 2020; Peters et al., 2015; Saengkyongam et al., 2021; Zhang et al., 2020; Magliacane et al., 2017). In our framework, taking advantage of an invariance is similar to choosing predictors E such that $W_{Q|\mathsf{E}}$ contains only one point, thus allowing zero-shot learning. As we discuss in Section 3.1 and Section 5.3, we consider the world update problem, which is different from choosing predictors.

Causal Transfer Learning. Our work builds on prior studies of transfer learning from a causal modeling perspective, without assuming causal faithfulness or sufficiency. Unlike Chen & Bühlmann (2021), who assume a linear SCM setting, our framework does not. Pearl & Bareinboim (2011) use selection variables S to model environmental differences, a special case of our framework. They allow any distribution selection, not constraining the variable given its parents. Our framework provides more details about when transfer learning is possible. Consider the causal graph among variables $X; Y; Z; S$ with edges $Z \to X; Z \to Y; X \to Y$ and $S \to Y$. This example is a case of Figure 4(b) in Pearl & Bareinboim (2011) where it was determined that transfer learning of $P(Y|do(X))$ is not possible (Example 6 in Pearl & Bareinboim (2011)). Variable S here could be interpreted as determining the selection of the distribution of Y given its parents X and Z in different environments. However if $P(Y|X; Z)$ was selected from a constrained set, transfer learning of $P(Y|do(X))$ might still be feasible, which can be analyzed in an extension of our framework.

C. Future Works

We describe a more general approach to learn $W_{Q|\mathsf{E}}$.

We assume that the agent might have some expert knowledge about some $W_{\mathsf{I}|\mathsf{K}}$ even though I or K might not be observed. Thus, the agent wants to learn $W_{Q|\mathsf{E}}$ from the set $\{W_{\mathsf{I}|\mathsf{K}}\}$ for some $\mathsf{I}; \mathsf{K}$ where it has some knowledge about $W_{\mathsf{I}|\mathsf{K}}$. In this paper we only consider a special case where the agent has knowledge about some atomic domains $\{W_i\}$, and using Algorithm 1 we express $W_{Q|\mathsf{E}}$ as a formula of W_i. In the general case of learning $W_{Q|\mathsf{E}}$ from the set $\{W_{\mathsf{I}|\mathsf{K}}\}$, the problem can be divided into 2 steps:

1. *Choosing the terms.* Choosing the sets of $(\mathsf{I}; \mathsf{K})$'s to compute $W_{Q|\mathsf{E}}$ from $W_{\mathsf{I}|\mathsf{K}}$'s.
2. *Calculation.* Compute $W_{Q|\mathsf{E}}$ from the set of chosen $W_{\mathsf{I}|\mathsf{K}}$'s.

While it might be intuitive to always choose W_i's to compute $W_{Q|\mathsf{E}}$, we show an example where it is not the best way. Consider the graph $X \to Y \to Z$ where both $Y|X$ and $Z|Y$ are poorly conditioned and therefore estimating them yields higher errors while $Z|X$ is the identity. Therefore if the goal is to compute $Z|X$, it is better to compute it directly rather than from $Z|Y$ and $Y|X$.

Once the terms $w_{\mathsf{I}|\mathsf{K}}$'s are chosen, there are possible different approaches to calculate the target $w_{Q|\mathsf{E}}$. One approach is to break both $w_{\mathsf{I}|\mathsf{K}}$'s and $w_{Q|\mathsf{E}}$ into formulae of the atomic term w_i's using Theorem E.3 and then to reason about how the constraint on $W_{\mathsf{I}|\mathsf{K}}$'s implies constraint on the atomic W_i, and then how the constraint on the atomic W_i's implies constraint on $W_{Q|\mathsf{E}}$, converting the problem to the special case we already addressed in this paper. Another approach is to directly express $w_{Q|\mathsf{E}}$ as a formula of the chosen term $w_{\mathsf{I}|\mathsf{K}}$'s.

We leave the extension to the future works.

Model Selection. If the agent has no expert knowledge to select the hypothesis class of function g that describes $W_{\mathsf{I}|\mathsf{K}}$, or that the expert knowledge is not specific enough, selecting the hypothesis class of g using the expert knowledge can result in a large over-estimate of $W_{\mathsf{I}|\mathsf{K}}$. In this case the agent can employ model selection strategies to select the hypothesis class from a nested set of hypothesis class using the incoming data w^e. If the agent first assumes that w^e is in a large set but later notices that all incoming w^e's lie in a small subset, the agent can narrow down the hypothesis class. The expert knowledge, if exists, place an upper bound on the model selection strategy. For example, suppose that $w_{100|0} = w_{100} \quad w_1$ is unchanged in all environments but the agent's expert knowledge is only that w_1 is unchanged in all environments, using which it infers that $w^e_{100|0}$ have the same row space in all environment. Then the agent can search through a nested hypothesis class for $W_{100|0}$ with the largest class being the set of matrices with the same row space with unknown bases and the smallest class

being the set of an unknown single matrix. After the observing that all of the incoming data $w^1_{100j0}; \quad ; w^e_{100j0}$ are the same matrix the agent can pick the smallest hypothesis class and learn the value of the unknown matrix. If the agent has no expert knowledge at all, it has to search through a nested hypothesis class with the largest class being the set of all matrices and the smallest class being the set of a single matrix.

We leave this extension to future works.

D. Details about Section 3

The SCM and the ordering generate a causal graph. Let T be the time horizon. Let be an ordering of the n variables X. At time $t = 0$, each variable is initialized with an initial value X^0. At time t, at step $i; 1 \quad i \quad n$, the SCM generates the variables $X^{t}{}_{(i)}$. Let $P_i^{t \ 1;t} \quad X^{t \ 1} [X^t$ denote the set of parents of X_i^t. Then we have: $X_i^t = f_{w_i^t}(P_i^{t \ 1;t}; U_i^t)$ where $w_i^t = w_i$.

For examples, consider the following SCM:

$$X_1 = f_{w_1}(U_1) \tag{4}$$

$$X_2 = f_{w_2}(X_1; U_2) \tag{5}$$

If the ordering $= (1; 2)$, then the parents of X_2^1 is $P_2^{t \ 1;t} = fX_1^1 g$ and $X_2^1 = f_{w_2^1}(X_1^0; U_2^1)$ since X_1^1 is generated before X_2^1. If the ordering $= (2; 1)$, then the parents of X_2^1 is $P_2^{t \ 1;t} = fX_1^0 g$ and $X_2^1 = f_{w_2^1}(X_1^0; U_2^1)$ since X_1^0 is generated before and X_1^1 is generated after X_2^1.

The SCM and the ordering induces a causal graph. The causal graph is defined such that there are directed edges from each node in the set of parents $P_i^{t \ 1;t}$ of X_i^t to X_i^t (even when W_i contains the value of w_i such that X_i^t does not depend on its parents). We provide an example of the graph for Q Learning in Figure 10b. The causal graph induces a partial order a partial order on the variables X_i^t's: for any $A; B \ 2 \ fX_i^t; 1 \quad t \quad T; 1 \quad i \quad ng; A \quad B$ if there exists a directed path from A to B. To simplify the notations, in the main text we remove the superscript t by re-indexing the variables according to such that $X_i \quad X_j$ if $i \quad j$.

E. Details about Section 4

E.1. Computation Trees

In this section we provide the proof of Theorem 4.2, which is restated by Theorem E.3.

d-separation between variables (denotes $?_d$) and the partial order is defined on the causal graph (Section D. For any 2 sets X and Y, X Y if $X \quad Y \ 8X \ 2 \ X; Y \ 2 \ Y$. For any two variables $X \ 6= Y$, X is called an ancestor of Y and Y a descendant of X if there is a directed path from X to Y in the causal graph. Let the *ancestor set* $A(I)$ of I be the set of all ancestors of variables in I, excluding variables in I. Let $A^0(I) := A(I) [I$. The *parent set* of I is the set of all parents of variables in I, excluding variables in I.

A set J is called a *sufficient ancestor set* of a set I if J $A(I)$, J I and $I ?_d A(J) j J$. The empty set is considered a sufficient ancestor set of any other set. Conditioning on a sufficient ancestor set J of I, we can compute I from the atomic parameters without needing the ancestors of J (Theorem E.3). Given $I; K$ where $I \setminus K = ;$, a set J is called a *sufficient ancestor set* of $I \ j \ K$ if J K and J is the sufficient ancestor set of $I [(KnJ)$.

We show first that there is a maximal sufficient ancestor set of $I \ j \ K$.

Lemma E.1. *There exists a maximal sufficient ancestor set* J *of* $I \ j \ K$ *such that* J *is a sufficient ancestor set of* $I \ j \ K$ *and* J *contains any other sufficient ancestor set of* $I \ j \ K$.

Proof. We will show that if J_1 and J_2 are sufficient ancestor sets of $I \ j \ K$ then $J := J_1 [J_2$ is also a sufficient ancestor set of $I \ j \ K$. Since $J_1 \quad K; J_2 \quad K$, we have J K.

By definition, $I [KnJ_i$ is d-separated from $A(J_i)$ given J_i for $i = 1; 2$. Therefore $I [KnJ$ is d-separated from $A(J_i)$ given J for $i = 1; 2$ by weak union by moving $J_1 n J_2$ from $I [KnJ_2$ to J_2 (or $J_2 n J_1$ from $I [KnJ_1$ to J_1). Therefore $I [KnJ$ is d-separated from $A(J_1) [A(J_2)$ given J by composition. Since $A(J) = A(J_1 [J_2) \quad A(J_1) [A(J_2)$, $I [KnJ$ is d-separated from $A(J)$ given J by decomposition. By definition, J is a sufficient ancestor set of $I \ j \ K$. □

Therefore the scope of $I \mid K$ is defined by computing the scope of $I \cup (K \cap J) \mid J$ where $J \subseteq K$ is the maximal sufficient ancestor set of $I \mid K$.

Let K^0 be an independent set of $I \mid K$ if $K^0 \subseteq K$ and $I \perp_d K^0 \mid K \cap K^0$. There exists a set K^0 such that K^0 is an independent set of $I \mid K$ and K^0 contains any other independent set of $I \mid K$, called the *maximal independent set* of $I \mid K$. Such a maximal set exists because of the intersection property of d-separation: If $A \perp_d B \mid C \cup D$ and $A \perp_d C \mid B \cup D$ then $A \perp_d B \cup C \mid D$. Suppose there exist $K_1 \subseteq K$ and $K_2 \subseteq K$ such that $I \perp_d K_1 \mid K \cap K_1$ and $I \perp_d K_2 \mid K \cap K_2$. Then $I \perp_d K_2 \cap K_1 \mid K \cap (K_1 \cup K_2)$ by weak union.

Definition E.2 (Scope of $w_{I|K}$)**.** Let I and K be 2 disjoint sets. Let $K^0 \subseteq K$ be the maximal independent set of $I \mid K$. Let $J \subseteq K \cap K^0$ be the maximal sufficient ancestor set of $I \mid K \cap K^0$. Let $I' := I \cup (K \cap K^0) \cap J$. Then $\text{Scope}(I \mid K) := A^0(I') \cap A^0(J)$.

We show that $w_{Q|E}$ could be computed from variables in $\text{Scope}(Q \mid E)$ using the operations $;$, $\cdot$, and $\int$, where w_s^t's appear in reverse topological order. Let $w_{\text{Scope}(Q|E)} := \{w_i^t \mid (i,t) \in \text{Scope}(Q \mid E)\}$ and $\mathcal{W}_{\text{Scope}(Q|E)}$ be the corresponding constrained domain of $w_{\text{Scope}(Q|E)}$.

Theorem E.3 (Top-down computation tree)**.** *Given $Q \setminus E = \emptyset$, we initiate the formula with $w_{Q|E}$. For each term of the form $w_{I|K}$ in the formula, the agent repeatedly performing the following steps. First it removes the maximal independent set K^0 of $I \mid K$ from K. To make the notations simple, we re-use K afterwards to denote the set $K \cap K^0$ after removing K^0. The agent then chooses one of the following computations.*

- *Choose $J \subset I$, s.t. $J \subseteq I \cap J$ and $J \cup K \cap J^0$ is a sufficient ancestor set of $I \cap J$ where J^0 is the set of variables d-separated from $I \cap J$ given $(J \cup K) \cap J^0$. Then calculate the product:*

$$w_{I|K} = w_{I \cap J \mid (J \cup K) \cap J^0} \cdot w_{J|K}. \tag{6}$$

- *Choose $J \subseteq \text{Scope}(I \mid K)$ such that $I \setminus J = \emptyset$, $K \setminus J = \emptyset$; and sum out J from the joint distribution:*

$$w_{I|K} = \int_J w_{I,J|K}. \tag{7}$$

- *Let $J \subseteq K$ be the maximal sufficient ancestor set of $I \mid K$. If $J \neq K$, the learner then compute $I \mid K$ from the joint $I, K \cap J \mid J$ by conditioning on $K \cap J$ in addition to J:*

$$w_{I|K} = \int_{K \cap J} w_{I,K \cap J \mid J} \tag{8}$$

These options are executed until no such non-empty J satisfying either condition exists.

All variables in the formula are $w_i^t \in \text{Scope}(Q \mid E)$ in reverse topological order and each w_i^t appear at most once in the formula. We denote the resulting formula $g_{Q|E}(w_{\text{Scope}(Q|E)})$, also called a computation tree[11] *of $w_{Q|E}$.*

Proof. We show that removing $K^0 \subseteq K$ from K if $I \perp_d K^0 \mid K \cap K^0$ does not expand the scope, i.e. $\text{Scope}(I \mid K \cap K^0) \subseteq \text{Scope}(I \mid K)$. Let $P \subseteq K$ and $P^0 \subseteq K \cap K^0$ be the maximal sufficient ancestor set of $I \mid K$ and $I \mid K \cap K^0$. Since $I \cup (K \cap P) \perp_d A(P) \mid P$, we have $I \cup (K \cap K^0 \cap P) \perp_d A(P) \mid P$ by decomposition. Therefore P is a sufficient ancestor set of $I \mid K \cap K^0$, and $P \supseteq P^0$. Then we have $\underbrace{A^0(I \cup (K \cap K^0) \cap P^0) \cap A^0(P^0)}_{\text{Scope}(I \mid K \cap K^0)} \subseteq \underbrace{A^0(I \cup K \cap P) \cap A^0(P)}_{\text{Scope}(I \mid K)}$ because $I \cup ((K \cap K^0) \cap P^0) \subseteq I \cup (K \cap P)$ and $P^0 \supseteq P$.

We first show that if the agent terminates, then all the terms are in the form of w_i^t in reverse topological order. Since the agent cannot execute the $\cdot$ operations, the term $w_{I|K}$ satisfies the condition that K is a the maximal sufficient ancestor set of I. Therefore $\text{Scope}(I \mid K) = A^0(I) \cap A^0(K)$. Since the agent cannot execute $\int$, $A^0(I) \cap A^0(K) = I$, and therefore the parent set of I is a subset of $A^0(K)$. Since K is a sufficient ancestor set of I and therefore $I \perp_d A(K) \mid K$, the parent set of I is a subset of K. If I has more than 1 element, the agent can choose J to be the smallest element topologically to execute the $\int$ operation because $J \cup K$ will contain the parent set of $I \cap J$ and therefore $J \cup K$ will d-separated $I \cap J$ from $A(J \cup K)$. Therefore I has only one element denoted (i,t) and K contains all of its parents, and $w_{I|K}$ can be substituted by w_i^t. Since the $\int$ operation ensures that $J \subseteq I \cap J$, w_i^t comes before $w_j^{t'}$ if $(j,t') \in (i,t)$, and therefore the terms are in reverse topological order.

[11] There can be multiple computation trees depending on the choice the learner made.

We show now that for all operations ; and the scopes of the terms on the right-handed side are subset of the scope of the term on the left-hand side, and therefore all of the terms w_i^t 's in the formula are in the scope of $w_{\mathsf{Q}j\mathsf{E}}$. We also show that in the operation the scopes of the 2 terms in the right-handed side are disjoint, and therefore each term w_i^t appears at most once.

- First, consider the case when $w_{\mathsf{I}j\mathsf{K}} = w_{\mathsf{I}n\mathsf{J}j\mathsf{J};\mathsf{K}}$ $w_{\mathsf{J}j\mathsf{K}}$. Let P K denote the maximal sufficient ancestor set of I j K and P^0 K denote the maximal sufficient ancestor set of J j K. Since J [(KnP) ?$_d$ A(P) j P by decomposition, P is a sufficient ancestor set of J j K. Therefore P P^0. We have $\underbrace{A^0(\mathsf{J} [(\mathsf{K}n\{\mathsf{Z}\mathsf{P}^0))}_{\text{Scope}(\mathsf{J}j\mathsf{K})}nA^0(\mathsf{P}^0)$ $\underbrace{A^0(\mathsf{I} [(\mathsf{K}n\{\mathsf{Z}\mathsf{P}))}_{\text{Scope}(\mathsf{I}j\mathsf{K})}nA^0(\mathsf{P})$.

  We will now show that $\underbrace{A^0(\mathsf{I}n\mathsf{J})nA^0\{\mathsf{Z}(\mathsf{J} [\mathsf{K}n\mathsf{J}^0)}_{\text{Scope}(\mathsf{I}n\mathsf{J}j\mathsf{J} [\mathsf{K}n\mathsf{J}^0)}$ $A^0(\mathsf{I}n\mathsf{J})nA^0(\mathsf{J} [\mathsf{K})$. We will show that for any mutually disjoint sets X; Y; Z, if X A(Y) and Y ?$_d$ Z j X then A^0(Y)nA^0(X) A^0(Y)nA^0(X [Z). To do so, we show that any node N 2 A^0(Y)nA^0(X) is not in A^0(X [Z). Since A^0(X) A^0(Y) (because X A(Y)) and N 2 A^0(Y)nA^0(X), we have N 2 A^0(X). Suppose N is in A^0(X [Z). Then there must exists Z^0 2 Z such that N 2 $A^0(Z^0)$ and there is a directed path from N to Z^0 not going through X. Since N 2 A^0(Y)nA^0(X), there is a directed path from N to Y not going through X. Therefore conditioning on X, there is a Bayes' ball path from Y to Z^0 through N. This is a contradiction, therefore N 2 A^0(X [Z). We have $\underbrace{A^0(\mathsf{I}n\mathsf{J})n\{\mathsf{Z}A^0(\mathsf{J} [\mathsf{K})}_{\text{A superset of Scope}(\mathsf{I}n\mathsf{J}j\mathsf{J} [\mathsf{K}n\mathsf{J}^0)}$ $\underbrace{A^0(\mathsf{I} [(\mathsf{K}n\{\mathsf{Z}\mathsf{P}))}_{\text{Scope}(\mathsf{I}j\mathsf{K})}nA^0(\mathsf{P})$ because InJ I [KnP and J [K P, and therefore Scope(InJ j J [KnJ^0) Scope(I j K).

  We also have Scope(InJ j J [KnJ^0) \ Scope(J j K) = ; because A^0(J [K), a superset of the second scope, is removed from the first scope. Therefore all leaves appear at most once in the formula.

- Second, consider the case when $w_{\mathsf{I}j\mathsf{K}}$ = $_{\mathsf{J}}w_{\mathsf{I};\mathsf{J}j\mathsf{K}}$. Let P denote the maximal sufficient ancestor set of I j K. We will first show that P is a sufficient ancestor set of I; J j K. Let I^0 := I [(KnP). By the definition of sufficient ancestor set, I^0 is d-separated from A(P) given P. Since J $A^0(\mathsf{I}^0)nA^0$(P), there are directed paths from J to I^0 not through P. Suppose there are Bayes ball's paths from A(P) to J given P. Then there are Bayes ball's paths from A(P) to I given P by combining the Bayes ball's paths from A(P) to J given P and the directed path from J to I not through P. Therefore A(P) is not d-separated from I given P, a contradiction. Therefore there is no Bayes' ball path from A(P) to J given P, so J is d-separated from A(P) given P. Therefore I^0 [J = I [J [(KnP) is also d-separated from A(P) given P, which implies P is a sufficient ancestor set of I; J j K.

  Let P^{00} be the maximal sufficient ancestor set of I; J j K, then we have P P^{00}. Therefore A^0(I [J [(KnP^{00})) A^0(I [J [(KnP)) = $A^0(\mathsf{I}^0 [\mathsf{J})$. Since J $A^0(\mathsf{I}^0)$; $A^0(\mathsf{I}^0 [\mathsf{J}) = A^0(\mathsf{I}^0)$. Therefore A^0(I [J [(KnP^{00})) $A^0(\mathsf{I}^0)$, and we have $\underbrace{A^0(\mathsf{I} [\mathsf{J} [(\mathsf{K}n\{\mathsf{Z}\mathsf{P}^{00}))}_{\text{Scope}(\mathsf{I};\mathsf{J}j\mathsf{K})}nA^0(\mathsf{P}^{00})$ $\underbrace{A^0(\mathsf{I}^0))n\{\mathsf{Z}A^0(\mathsf{P})}_{\text{Scope}(\mathsf{I}j\mathsf{K})}$.

- Finally, when $w_{\mathsf{I}j\mathsf{K}}$ = $_{\mathsf{K}n\mathsf{J}}w_{\mathsf{I};\mathsf{K}n\mathsf{J}j\mathsf{J}}$, the scope of the term on the right-handed side is the scope of the term on the left-handed side by definition.

Therefore all of the terms w_i^t 's in the formula are in the scope of $w_{\mathsf{Q}j\mathsf{E}}$, and each term appear at most once. □

E.2. Comparisons with Variable Elimination

There are several differences: (1) Variable elimination computes the tree from the bottom-up by starting with the leaves w_i 's and choosing several nodes to be combine to create their parent; (2) Variable elimination does not explicitly describe the intermediate nodes; (3) Variable elimination does not describe a smaller scope and (4) In variable elimination, each leaf goes through at least 2 operations: at least 1 to build the joint distribution and the conditional operation while in our method it is possible for a leaf to go through only one operation (w_3 in Figure 4), which could make it easier to analyze if w_3 is the variable of interest.

E.3. Details about Examples in Section 4.1

We provide another example of causal graph, scope and computation tree in Figure 9, Example E.1 and Example E.2.

(a) Causal graph with the scope of w_{3j0} in yellow.

(b) A computation tree for w_{3j0}.

Figure 9. Causal graphs and their computation trees. For clarity, we do not draw the uniform noise variables.

Example E.1 (Q Learning). Let $A_t; S_t; R_t$, and G_t denote the action, state, reward, and return at time t. The $Q(s; a)$ function is the expected return at time t given the state at $t = 0$ parameterized by $w_{G_t j S_0; A_0}$. The SCM for Q-Learning is the following:

$$A = f_{w_A}(U_A) \tag{9}$$
$$S = f_{w_S}(A; S; U_S) \tag{10}$$
$$R = f_{w_R}(A; S; U_R) \tag{11}$$
$$G = f_{w_G}(R; G) \tag{12}$$

and the ordering to generate the variables $= (S; A; R; G)$.

$A_t; S_t; R_t$ and G_t are interpreted as the action, state, reward and return at time t. The $Q(s; a)$ function is the expected return at time G_t given S_0 and A_0. The causal graph is shown in Figure 10a and the below computation tree for $w_{G_2 j A_0; S_0}$ is shown in Figure 10b. For any variable X_i, since $w_i^t = w_i$ for any t, we can compress all nodes w_i^t into a single node w_i.

$$w_{G_2 j S_0; A_0} = w_{G_2 j G_1; R_1; A_1; S_1} \left(w_{G_1; R_1; A_1; S_1 j G_0; R_0; A_0; S_0} \left(w_{G_0; R_0 j S_0; A_0} \right) \right) \tag{13}$$
$$= (w_G \ (w_R \ (w_S \ w_A))) \ (w_G \ (w_R \ (w_S \ w_A))) \ (w_G \ w_R)) \tag{14}$$

Example E.2 (Figure 3). Scope$(7\ j\ 6; 3) = \{3; 4; 5; 6; 7\}$. We show below a computation tree with variables in the scope:

$$w_{7j6;2} = {}_6 w_{7;6j2} \tag{15}$$
$$= {}_6(w_{7;6j5;4;3;2} \ w_{5;4;3j2}) \tag{16}$$
$$= {}_6((w_7 \ w_6) \ (w_5 \ w_4 \ w_3)) \tag{17}$$

E.4. Details about Section 4.3

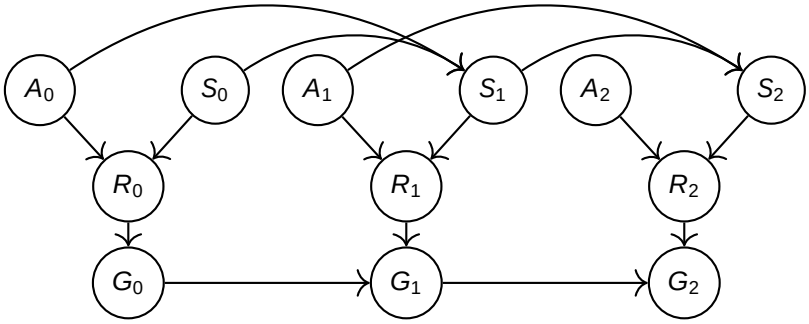
If each atomic parameter w_i appear at most once after the substitution of w_i^t by w_i as in the case of supervised learning, the formula will be simple, and constrained domain can be computed iteratively: if $w_{Y j X} = (w_3 \ w_2) \ w_1$ where denote some operations, then we can first compute the constrained domain W_{32} of $w_3 \ w_2$, and then compute $W_{Y j X} = W_{32} \ W_1$ where $W_{32} \ W_1$ is define to be the set $\{u \ v \ j \ u \in W_{32}; v \in W_1\}$ since W_{32} are independently chosen according to the Independent Causal Mechanism. If $w_{Y j X} = (w_1 \ w_2) \ w_1$ where w_1 appears twice, we cannot compute $W_{Y j X} = W_{12} \ W_1$ because W_{12} and W_1 are not chosen independently. Instead we need to define $W_{Y j X} = \{u \ v \ u \ j \ u \in W_1; v \in W_2\}$.

F. Details about Section 5

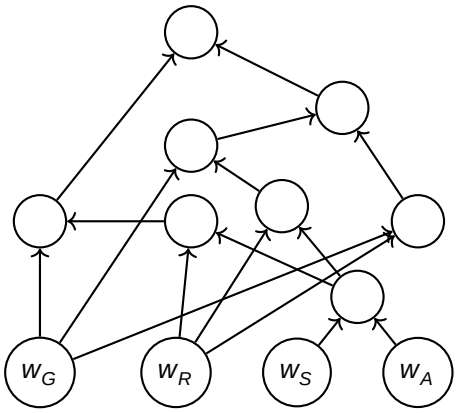
F.1. Details about Example 5.1

Sparse mechanism shifts do not imply transfer feasibility. Consider the first scenario where $\forall 2 \ i \ 99; w_i$ is the same matrix satisfying $\prod_{i=2}^{99} w_i$ has rank 2 in all environments, while w_1 and w_{100} are re-selected from all possible $2 \ 2$ matrices in each new environment. Since only w_1 and w_{100} change in different environments, the changes are sparse. We show below that the constraints on $w_2; \cdots; w_{99}$ do not result in any constraint on $w^e_{100/0}$ by showing the following property of .

Lemma F.1. *Let $M^{m \ n}$ denote the set of all matrices with dimension $m \ n$. Let w be a $2 \ 2$ matrix of rank 2. Then: $\{AwB \ j \ A \in M^{2 \ 2}; B \in M^{2 \ 2}\} = M^{2 \ 2}$.*



(a) The variables created by the SCM of Example E.1 described in Eq. 9 and the ordering $\lambda = (S, A, R, G)$. G_t is the return at time t. For clarity, we do not draw the uniform noise variables.



(b) A computation graph of $w_{G_2 | A_0, S_0}$ for Example E.1. For all variables X, we compress all nodes w_X^t into a single node w_X.

Figure 10. The causal graph and a computation tree for Q-Learning in Example E.1.

Since w_{99} w_2 has rank 2, fAw_{99} $w_2 B\ j\ A\ 2\ M^{2\ 2}; B\ 2\ M^{2\ 2}g = M^{2\ 2}$. The constraint that $w_{99}; : : : ; w_2$ is the same matrix in all environments does not enforce any constraint on w^e_{100j0}.

Transfer feasibility does not imply sparse mechanism shifts. Consider the second scenario in Example 5.1 where in each environment, $8 2\ i\ 100; w_i$ is selected from the set of all $2\ 2$ matrices while w_1 is the same matrix of rank 1. Since w_1 is the right-most term in the computation graph, we analyze the left-propagatable property for . Matrix multiplication is F-left-propagatable where F is the set of all matrices with rows in the row space of the right term w_1. Therefore, w^e_{100j0} is in the set of all matrices with rows in the row space of w_1. Since w_1 has rank 1, in Algorithm 1 is the $1\ 2$ basis vector of the row space, which can be learnt from w^1_{100j1} of Environment 1.

Transfer feasibility of the algorithm. Without transfer learning, comp = 4 since the matrix w_{100j} is $2\ 2$. With our transfer learning procedure, the learner needs to learn a $2\ 1$ vector to compute w_{100j0} after knowing the $1\ 2$ basis of its row space, therefore comp = 2.

Representation Learning. The distribution of $X_1\ j\ X_0$ where X_0 is the predictor could be consider the representation. In the second scenario, the representation is the same across the environments, and transfer feasibility is analyzed by the left-propagatable property of the operation since the representation is the right-most term.

Proof of Lemma F.1. Given $w\ 2\ M^{2\ 2}$ of rank 2, we will show that for any $C\ 2\ M^{2\ 2}$, there exist $A; B\ 2\ M^{2\ 2}$ such that:

$$AwB = C: \tag{18}$$

Let $w = U\ V^T$ and $C = U^{0\ 0} V^{0T}$ be the SVD decompositions of w and C. Since $rank(C)$ $rank(w)$, there exists a

diagonal matrix D such that $^{0} = \quad D$. Let $A = U^{0}U^{T}$ and $B = VDV^{0T}$. Then:

$$AwB = (U^{0}U^{T})(U \quad V^{T})(VDV^{0T}) \tag{19}$$
$$= U^{0} \quad DV^{0T} \tag{20}$$
$$= U^{0} \quad {}^{0}V^{0T} \tag{21}$$
$$= C \tag{22}$$

□

F.2. Details about Example 5.2

We derive a computation graph:

$$w_{100j0} = w_{100} \quad w_{99} \qquad w_{1} \tag{23}$$

where the computations are performed from left to right.

For neural network. is stacking layer: for $i > j > k$, the generator $w_{ijk} = w_{ijj} \quad w_{jjk} = (w_{ijj}; w_{jjk})$. Therefore $w_{100j0} = (w_{100}; \quad ; w_{1})$

(Rolnick & Kording, 2020) show that under the Linear Regions Assumption, except for a measure-zero set of networks, ReLU networks can be recovered up to permutations and rescaling if the domain of the input is the entire multidimensional real space. The description set w can be obtained from w by replacing $w_{k}^{(:i)}; w_{k+1}^{(i:)}$ by $cw_{k}^{(:i)}; \frac{1}{c}w_{k+1}^{(i:)}$ and replacing $w_{k}; w_{k+1}$ by a permutation of the rows of w_{k} and the columns of w_{k+1}. Under the Linear Region Assumption, in all environments, from Eq. 23, the description set $\overline{w}_{100j0} = f((w_{100}; \quad ; w_{1}); \; ; c) \; j \quad 2 \quad {}_{100j0}; c \; 2 \; C_{100j0}g$ where ${}_{100j0}$ and C_{100j0} are sets of permutations and scaling applying to $(w_{100}; \quad ; w_{1})$ to generate the descriptions.

For any $i; 1 \quad i \quad 100$, if w_{i} is the same in all environments, then for any description $u = (u_{1}; \quad ; u_{100})$ of w_{100j0}, u_{i} can be obtained from w_{i} by permutation or rescaling, and therefore is layer i of a description of w_{100j0}^{k} in another environment E_{k}. Therefore we can learn layer u_{i} of a description from the first environment and freeze it in later environments. We do not use the exact algorithm described at the end of Section 4.4 (when Assumption 3.4 does not hold) because this discussion is shorter, but in both the algorithm in Section 4.4 and this discussion, we still need to learn or infer all possible descriptions from previous environments in order to transfer correctly.

We now show below that even with slight modifications from the above example, no longer corresponds to an entire layer, and therefore freezing a layer learnt from previous environments lead to sub-optimal performance.

Example F.1. Consider the following SCM: $X_{0}; X_{1}; X_{2} \; 2 \; \mathrm{R}^{2}$ and $X_{i} = f_{w_{i}}(X_{i} \quad _{1}) + \quad _{i}; i \; 2 \; f1; 2g$ where $_{i}$'s are independent noise satisfying $E[\, _{i}] = 0$ and $f_{w_{i}}$ is a 1-layer ReLU neural network where w_{i} denotes the weight. In all environments, $w_{2} = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$ is the same and w_{1} can be selected from all 2 2 matrices. Suppose the leaner wants to transfer knowledge of w_{2j0}, which is the weight of a 2-layer ReLU network, from Environment 1 to Environment 2. Consider the following cases:

1. In environment E_{1}, ${w_{1}}^{1} = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 0 \\ \hline \end{array}$ and $X_{0} \; 2 \; \mathrm{R}^{2}$ is a multivariate Gaussian on R^{2}.
2. In environment E_{1}, ${w_{1}}^{1} = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$ and $X_{0} \; 2 \; \mathrm{R}^{2}$ is a multivariate Gaussian on the line $f[1; 1]^{T} cjc \; 2 \; Rg$.

In Case 1, in Environment 1, the following 2-layer weights are both descriptions of $w_{2j0} = (w_{1}; w_{2})$: $\begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 0 \\ \hline \end{array}; \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$ and $\begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 0 \\ \hline \end{array}; \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 6 \\ \hline \end{array}$. In Case 2, in Environment 1, the following 2-layer weights are both descriptions of $w_{2j0} = (w_{2}; w_{1})$ on the domain of X_{0}: $\begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}; \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$ and $\begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}; \begin{array}{|c|c|}\hline 0.3 & 0.7 \\ \hline 0.5 & 0.5 \\ \hline \end{array}$. Therefore in Case 1 and Case 2 the

agent cannot freeze w_2 learnt from Environment 1 because it may freeze an incorrect value such as $\begin{bmatrix} 1 & 0 \\ 0 & 6 \end{bmatrix}$ in Case 1 and $\begin{bmatrix} 0.3 & 0.7 \\ 0.5 & 0.5 \end{bmatrix}$ in Case 2. This is because in Case 1 and Case 2, the constrained domain that can be learnt from Environment 1 is no longer the set of multi-layer networks with a common layer w_2. Instead, the agent needs to learn the set of all possible descriptions from Environment 1.

F.3. Details about Section 5.2.2

In this section we provide details for the case study of the Colored MNIST problem in Section 5.2.2.

F.3.1. Derivation of the Architecture

We apply Theorem E.3 to derive the following architecture:

$$P(Y \mid X) = \frac{P(X;Y)}{\sum_y P(y;X)} \tag{24}$$

$$= \frac{\sum_p P(X;p;Y)}{\sum_y \sum_p P(X;p;y)} \tag{25}$$

$$= \frac{\sum_p P(X \mid p) P(p;Y)}{\sum_y \sum_p P(X \mid p) P(p;y)} \tag{26}$$

$$= \frac{\sum_p \exp\left(\frac{1}{s} \frac{T}{p}(X) + \log g(\ _p) + \log P(p;Y)\right)}{\sum_y \sum_p \exp\left(\frac{1}{s} \frac{T}{p}(X) + \log g(\ _p) + \log P(p;y)\right)} \tag{27}$$

F.3.2. Data Generation

Dataset and the pre-processing code to generate multiple environments of colored images from the original MNIST dataset from Gulrajani & Lopez-Paz (2020)[12] were used with image size $s = 28$ and number of color channels $k = 2$. We add one channel to make $k = 3$. Environment 1 and 2 has 60000 and 10000 samples respectively. We divide Environments 1 to train, val and test set with ratio 0.8; 0.1; 0.1. In Environment 2, we use 5000 samples for training and validation and 5000 samples for testing. Among the 5000 samples for training and validation, we gradually take the first 100; 200; 500; 1000; 2000 and 5000 samples as the number of samples available to the agent. For each available sample size 100; 200; 500; 1000; 2000 and 5000, we divide the train and val sets with ratio $7:3$.

F.3.3. Transfer feasibility of the algorithm

To learn $W^1_{Y|X}$, we need to learn the linear coefficient of (x). There are n_p coefficients $2\ R^{n\ n\ k}$ for each value of p. There are $n_y \quad n_p$ biases $\text{bias}_{y;p}\ 2\ R$ for each value of y and p. Since we do not know n_p, we use $n_p = 20$ as an upper bound. The number of cells for is $n_p \ n \ n \ k = 20 \ 28 \ 28 \ 3 = 47040$. The number of cells for the bias of the first layer is $n_p = 20$. The number of cells of the weight of the second layer is $n_p \ n_y = 20 \ 2 = 40$. Let the complexity measure comp be the dimension of the parameters needed to learn. Then, without transfer learning, comp $= 47040 + 20 + 40$; with transfer learning, comp $= 20 + 40$.

F.3.4. Implementation

In all of our training procedures, we use stochastic gradient descent and the negative log likelihood loss. We repeat for 10 times the following procedure and plot the average and the standard deviation values in the plot. First we create Environments 1 and 2 as described above. We then train a model on the train set of Environment 1 and plot its accuracy the test set of Environment 1 and Environment 2 in Figure 5. Next we perform transfer from Environment 1 to Environment 2 in Figure 5. Each line is an average over 10 runs with the shaded region denoted the standard deviation. We plot the accuracy on the test set of Env. 2.

The weight in the second layer is $P(y \mid p)$ and therefore needs to satisfy $\sum_y P(y \mid p) = 1$. We satisfy this requirement by placing a softmax function on the weight.

[12] URL: https://github.com/facebookresearch/DomainBed/blob/main/domainbed/datasets.py

The linear coefficients of the first layer and second are initialized uniformly from $(0, 1)$. The bias of the first layer is initialized uniformly from $(-\frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}})$ where k is the number of incoming features.

F.3.5. Hyperparameters Tuning

We use Ray Tune to tune the hyperparameters. We sample the parameters from the configuration range in the table below, run the training process until the number of epochs is 100 or the standard deviation of the validation loss of the last 10 epoch is at most 0.01. We select the hyperparameter with the lowest validation loss, and retrain the model with the selected hyperparameter on the combined train and validation set for 100 epochs. We use the retrain model to predict the test set and to transfer. Due to limited computation resource, we use less resource to tune our method by using a smaller hyperparameter range (and therefore a smaller number of sampled hyperparameters) and use more resource to tune competing methods by using a larger range and more samples.

	Train in Env. 1	Train in Env.3 (No Transfer)	Transfer Coefficient	Transfer Layer, Transfer Random Coefficient
Learning rate	loguniform(0.01, 5)			
Momentum	0.9			
Weight decay	0.0001	loguniform(0.00001, 0.01)	0.0001	loguniform(0.00001, 0.01)
n_p	20	20 or 50 uniformly		
Batch size	512			
Number of sampled hyperparameters	8	100	8	50

F.3.6. Plot Legend

Without transfer. The model is train from scratch in Env. 2 the available samples $100, 200, 500, 1000, 2000, 5000$ successively.

Transfer coefficient. We freeze the first layer's coefficient learnt from Env. 1 and retrain the first layer's bias and the second layer in Env. 2 with the available samples $0, 100, 200, 500, 1000, 2000, 5000$ successively.

Transfer layer. We freeze the first layer's coefficient and bias learnt from Env. 1 and retrain the second layer in Env. 2 with the available samples $0, 100, 200, 500, 1000, 2000, 5000$ successively.

Transfer random coefficient. We generate the first layer's coefficient randomly, freeze it and retrain the first layer's bias and the second layer in Env. 2 with the available samples $0, 100, 200, 500, 1000, 2000, 5000$ successively.

F.3.7. Compute

The experiments that produce the figures in this paper were performed on a personal laptop.

F.3.8. Hypothetical Model Satisfying the Assumptions

We present a hypothetical model that satisfies the 4 assumption stated in Section 5.2.2. Note that this model is only for illustration and to show that the 4 assumptions could be reasonably satisfied. We did not use this model in our analysis or experiment.

Example F.2 (A hypothetical model for Colored MNIST). We present a hypothetical generative model that satisfied all of our assumptions (Figure 11). Let $X \in \mathbb{R}^{s \times s \times k}$ denote the image, $D \in \{0, \ldots, 9\}$ denote the digit, $Y \in \{0, 1\}$ denote the binary digit label, $C \in \{0, 1\}$ denote the color, $M \in \mathbb{R}^{s \times s \times n}$ denote the digit matrix and $N \in \mathbb{R}^{k \times k}$ denote the color matrix. The binary digit Y is constructed by flipping D with probability 0.25. The color C is constructed by flipping Y with probability $0.1, 0.2,$ and 0.9 respectively in environments $1, 2,$ and 3. Each image x is a $s \times s \times k$ tensor where s is the image size and k is the number of color channels. For each digit d, there is a digit matrix $m_d \in \mathbb{R}^{s \times s \times k}$ which draws the digit in red while other color channels of m_d are 0. For each color c, there is an associated color matrix $n_c \in \mathbb{R}^{k \times k}$. The image x is constructed by first computing $m_d \times n_c$ where $\times$ denotes multiplication where each $s \times k$ slice of m_d is multiplied with n_c. Then independent standard Gaussian noise is added to each cell.

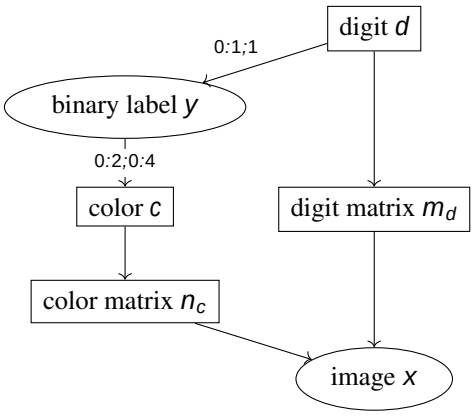


Figure 11. A hypothetical generative model of Complex Colored MNIST dataset. Ellipse variables are observable while square variables are not observable.

G. Limitations

Our work assumes realizability, a common assumption which does not always hold in practice. Nevertheless, experiments on the Colored MNIST example show that our method still works. Relaxation of the assumption is a topic for future works.

Our model does not include the case in reinforcement learning where the reward R_t at time t is a function of both the previous state $S_{t\ \ 1}$ and the current state S_t.