Causality without Estimands: from Causal Estimation to Black-Box Introspection

Alvaro Parafita

UNIVERSITAT DE BARCELONA

PHD IN MATHEMATICS AND COMPUTER SCIENCE

Causality without Estimands: from Causal Estimation to Black-Box Introspection

Author:
Álvaro PARAFITA

Advisor:
Dr. Jordi VITRIÀ

A thesis submitted

in the

Facultat de Matemàtiques i Informàtica

December 16th, 2022

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica
Departament de Matemàtiques i Informàtica

**Causality without Estimands:
from Causal Estimation
to Black-Box Introspection**

by Álvaro PARAFITA

After years of obscurity, Causality is here to stay: from medicine, to social sciences, to econometrics, to artificial intelligence, many fields have started integrating the findings of the field of Causal Inference. However, the essential tool for any application based on causality, Causal Query Estimation, has typically been limited by techniques defined around estimands, formulas that translate causal expressions into observational terms that can be computed with passive data. This reliance on an estimand for each and every query results in ad hoc models, hard to apply to different contexts with more nuanced causal connections and requiring the training of one model for each new query. Here we propose Deep Causal Graphs (DCGs), a general estimand-agnostic framework capable of answering any *identifiable* causal query using one single model; trained only once per dataset with the same general procedure, it can adapt to many kinds of causal queries through three simple operations. Despite being an abstract framework, DCGs can leverage the expressive power of Deep Neural Networks and Normalizing Flows, allowing to model complex real-world distributions. We showcase the estimation capabilities of DCGs in comparison with the state of the art in Causal Query Estimation, and provide applications to the fields of Black-Box Interpretability, Explainability and Fairness.

Acknowledgements

This has been a long and arduous journey. I would never be here without the support of so many people. This is for each and every one of you.

To Jordi, for his invaluable advice, for helping me keep every setback in perspective, for his many words of encouragement.

To Edu, for always being there, for the many, *many* times I needed someone to talk to, for patiently listening me unravel every tiny problem I was facing.

To my parents, for their unwavering support, for always reminding me to take it easy, for helping me focus on the finish line.

To my sister, because no matter the distance, no matter how busy life can get, things will always be the same.

To my niece, because her uncle knows she will surpass us all.

To Carlos, my office buddy, for countless talks, for all the coffee breaks, for always encouraging me.

To Pablo, Axel, Guillem, Pere, Paula and Mariona, for making me part of the group, even if I'm not always there.

To my college friends, for the journey we started together, for being such an amazing family.

To my friends at home, for all these years.

This work has been partially funded by projects RTI2018-095232-B-C21 (MINECO/FEDER, UE), 2017SGR1742 (Generalitat de Catalunya) and an APIF scholarship from Universitat de Barcelona.

Contents

Notation

We use uppercase letters (X) for random variables (r. v. s) and the corresponding lowercase letters (x) for values of these variables. We denote sets of variables with boldface type (e.g., $\mathbf{X} = (X_1, \ldots X_K)$), again with corresponding lowercase letters for values of this set ($\mathbf{x} = (x_1, \ldots, x_K)$). We follow this rule unless the symbol itself already conveys a set of variables (e.g., Pa_X for the set of Markov parents of X), therefore with no possible ambiguity.

We refer to the distribution of a r. v. X as $P(X)$. We denote a sampled value from any of these r. v. s by $x \sim P(X)$. We often operate with discrete and continuous variables indistinctly; whenever we use the term $P(x)$ we might be referring to the Probability Mass Function (PMF) for the discrete case or the Probability Density Function (PDF) for the continuous case. In the same way, we avoid decompressing expectations unless necessary: given an arbitrary function f of $\mathbf{X}$, $E_{\mathbf{X}}[f(\mathbf{X})] = \sum_{\mathbf{x}} f(\mathbf{x}) \cdot P(\mathbf{x})$ for the discrete case, or $E_{\mathbf{X}}[f(\mathbf{X})] = \int f(\mathbf{x})P(\mathbf{x})\partial\mathbf{x}$ for the continuous case.

We denote "X is independent of Y in distribution P" by $(X \perp\!\!\!\perp Y)_P$. We also denote "X is independent of Y conditional on Z in distribution P" by $(X \perp\!\!\!\perp Y \mid Z)_P$. We denote non-independence with the $\not\perp\!\!\!\perp$ symbol in the same way. We omit some of these terms for simplicity (e.g., $X \perp\!\!\!\perp Y$) provided it does not lead to ambiguity.

Part I

Preliminaries

Chapter 1

Introduction and Background

The notion of cause and effect is fundamental to our understanding of the real world; ice cream sales correlate with jellyfish stings (both increase during summer), but a ban on ice cream could hardly stop jellyfishes. This discrepancy between the patterns that we *observe* and the results of our *actions* is essential: **without causal knowledge we are mere spectators of the world**, unable to understand its inner workings, enact effective change, explain which factors were responsible for a specific outcome or imagine potential scenarios resulting from alternative decisions.

The field of statistics has traditionally stayed in the realm of observations, powerless in the measurement of causal effects unless by performing **randomized experiments**. These consist of dividing a set of individuals in two groups at random and assigning a certain action/treatment to each subgroup, to then compare the outcomes of both. This could be applied, for instance, to measure the impact of large-scale advertisement campaigns on sales, test the effects of smoking on the development of lung cancer, or determine the influence of new pedagogical strategies on eventual career success. However, randomized experiments are not always feasible, as is the case in these examples, due to economic, ethical or timing concerns.

Causal Inference is the field that studies how to circumvent this problem: only using *observational* data, not subject to randomization, it allows us to measure causal effects. Even so, the standard approach for Causal Estimation (CE), **estimand-based** methods, results in ad hoc models that cannot extrapolate to other datasets with different causal relationships, and often require training a new model every time we want to answer a different query on the same dataset. Contrary to this perspective, **estimand-agnostic** approaches train a model of the *observational distribution* that acts as a proxy of the underlying mechanism that generated the data; this model needs to be trained only once and can answer any *identifiable* queries reliably. However, this latter approach has seldom been studied, primarily because of the difficulty of defining a good model of the target distribution satisfying every causal requirement while still flexible enough to answer the desired causal queries.

This dissertation is focused on the definition of a general estimand-agnostic CE framework, **Deep Causal Graphs**, that can leverage the expressive modelling capabilities of Neural Networks and Normalizing Flows while still providing a flexible and comprehensive estimation toolkit for all kinds of causal queries. We will contrast its capabilities against other estimand-agnostic approaches and measure its performance in comparison with the state of the art in Causal Query Estimation.

Finally, we will also illustrate the connection between CE and **Machine Learning Interpretability, Explainability and Fairness**: since the examination of black-boxes often requires to answer many causal queries (e.g., what is the effect of each input variable on the outcome, or how would the outcome have changed had we intervened on a certain input), estimand-based techniques would force us to train as many different models; in contrast, estimand-agnostic frameworks allow us to ask as many questions as needed with just a single trained model, and therefore are essential for this kind of application.

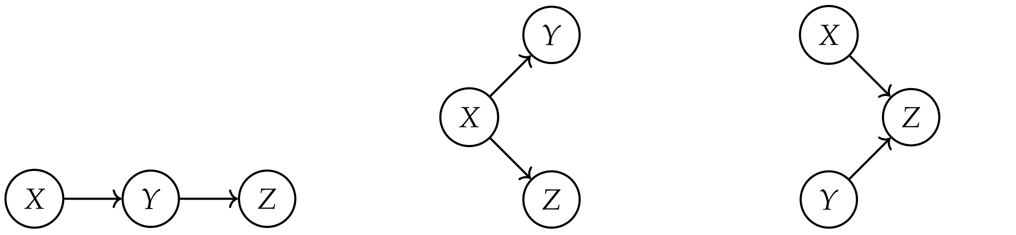
1.1 Estimand-Based Causal Estimation

In order to explain the shortcomings of the estimand-based CE approach, we need to define the tools of Causal Inference, particularly from Pearl's perspective using Causal Graphs [1]; for an overview from the Potential Outcomes perspective, please refer to [2]. In the following subsections, we will establish the required graph terminology, define *d-separability* and show its importance for identifying independencies in complex probability distributions, explain the concept of *intervention* and its effect on Bayesian Networks, define *identifiability* and the rules of do-calculus, to finally describe the estimand-based approach, relevant works on the subject and its assumptions and limitations.

1.1.1 Graph Terminology

A **graph** is a tuple $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ with $\mathbf{V} = \{V_1, \ldots, V_K\}$ a set of **nodes** and $\mathbf{E}$ a set of **edges**, pairs of connected nodes in $\mathbf{V}$. Any edge between variables $V_k, V_l \in V$ can be *directed* ($V_k \to V_l$ or $V_k \leftarrow V_l$), *bidirected* ($V_k \leftrightarrow V_l$) or *undirected* ($V_k — V_l$). A graph consisting only of directed edges is called a **directed graph**. A **path** is an ordered collection of node-pairs (V_k, V_l) where every pair is connected by an edge $e \in \mathbf{E}$ finishing on a node that is the beginning of the next pair (e.g., $((A, B), (B, D), (D, C))$). A path that begins and finishes on the same node is a **cycle**. A path where all edges are directed and point in the same direction (e.g., $A \to B \to C$) is a **directed path**. A directed graph with no cycles is called a **Directed Acyclic Graph** (DAG).

Consider a DAG. Given a directed edge $X \to Y$, X is a **parent** of Y and Y a **child** of X. An **ancestor** of Y is any node $X \in \mathbf{V}$ such that there exists a directed path from X to Y (including Y). A **descendant** of X is any node $Y \in \mathbf{V}$ such that X is an ancestor of Y. We denote the set of parents, children, ancestors and descendants of X as Pa_X, Ch_X, An_X and De_X, respectively. We use the notation $Pa_k := Pa_{V_k}$ for any $V_k \in \mathbf{V}$ without loss of generality. A graph $\mathcal{G}$ can have its set of nodes $\mathbf{V}$ defined in an arbitrary order; a **topological order** of a DAG $\mathcal{G}$ is any ordering $\mathbf{V} = \{V_1, \ldots, V_K\}$ such that $\forall V_l$, $\forall V_k \in An(V_l)$ then $k \leq l$. A graph $\mathcal{G}$ is **complete** when all pairs of nodes are connected by an edge. Given a set of nodes $\mathbf{V}$ in an arbitrary order, we can create a complete DAG $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ by defining $\mathbf{E} := \{V_k \to V_l \mid k < l\}$, which results in $\{V_1, \ldots, V_k\}$ being the topological order of the graph.



(A) **Chain**: $X \to Y \to Z$. (B) **Fork**: $Y \leftarrow X \to Z$. (C) **Collider**: $X \to Z \leftarrow Y$.

FIGURE 1.1: Three-node path patterns: chains, forks and colliders.

Consider the three graphs in Fig. 1.1; each graph exemplifies a kind of pattern for paths with three nodes. Graph 1.1a shows a **chain** $X \to Y \to Z$, where every node points to the next one. Graph 1.1b contains a **fork** $Y \leftarrow X \to Z$, with both edges emerging from the fork-node X. Graph 1.1c describes a **collider** $X \to Z \leftarrow Y$, with both arrows converging on the collider-node Z. We will discuss the importance of these structures in the following subsection.

1.1.2 Bayesian Networks and d-separability

Given a probability distribution P on a set of random variables (r.v.) $\mathcal{V}$ in an arbitrary order $\mathcal{V} := \{V_1, \ldots, V_K\}$, with $V_{<k} := \{V_l \mid l < k\}$ denoting the set of predecessors of any variable $V_k \in \mathcal{V}$, we define the **Markovian Parents** of V_k, denoted by Pa_k, as the minimal subset $Pa_k \subseteq V_{<k}$ such that V_k is *independent* of $V_{<k} \setminus Pa_k$ conditioned on Pa_k: $(V_k \perp\!\!\!\perp V_{<k} \setminus Pa_k \mid Pa_k)_P$. Given the Markovian Parents of every variable $V_k \in \mathcal{V}$, we can factorize the **joint distribution** P of these variables:

$$P(\mathcal{V}) = P(V_1, \ldots, V_K) = \prod_{k=1..K} P(V_k \mid V_{<k}) = \prod_{k=1..K} P(V_k \mid Pa_k). \tag{1.1}$$

Given a DAG $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ where every node V_k represents a r. v. and Pa_k denotes the parents of V_k in $\mathcal{G}$, and given a joint probability distribution $P(V_1, \ldots, V_K)$ on these variables, we say that P is **Markov relative** to $\mathcal{G}$ if P admits a factorization based on Markov Parents through the parent sets Pa_k in $\mathcal{G}$. We define such a graph as a **Bayesian Network** for distribution P.

Theorem 1.2.6 in [1] states that a necessary and sufficient condition for a probability distribution P to be Markov relative to a DAG $\mathcal{G}$ is that every variable V_k must be independent of its predecessors $V_{<k}$ in some topological order of $\mathcal{G}$ conditioned on its parents Pa_k. Theorem 1.2.7 proposes an equivalent condition not requiring a given order: every variable must be independent of all its nondescendants conditional on its parents in $\mathcal{G}$. This property is essential, since constructing a graph $\mathcal{G}$ that is Markov relative to a distribution P of interest allows us to determine independencies in its distribution through a graphical criterion: d-**separability**.

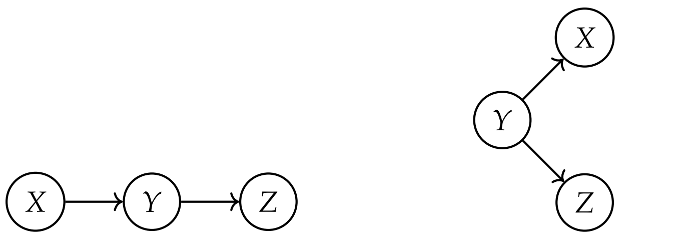
A path p is d**-separated** (blocked) by a set of nodes $\mathbf{Z} \subset \mathbf{V}$ (possibly empty) if and only if any of the following conditions is true:

- p contains a *chain* $A \rightarrow B \rightarrow C$ such that $B \in \mathbf{Z}$.
- p contains a *fork* $A \leftarrow B \rightarrow C$ such that $B \in \mathbf{Z}$.
- p contains a *collider* $A \rightarrow B \leftarrow C$ such that $\mathbf{Z}$ does not contain any descendant of B (including B), $De_B \cap \mathbf{Z} = \varnothing$.

Given disjoint sets $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subset \mathbf{V}$, we say that $\mathbf{Z}$ d**-separates X from Y** in $\mathcal{G}$, denoted by $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})_{\mathcal{G}}$, if $\mathbf{Z}$ d-separates every path p from a node $X \in \mathbf{X}$ to a node $Y \in \mathbf{Y}$. Theorem 1.2.4 in [1] states that if sets $\mathbf{X}$ and $\mathbf{Y}$ are d-separated by $\mathbf{Z}$ in a DAG $\mathcal{G}$, then $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})_P$ in every distribution P Markov relative to $\mathcal{G}$. Moreover, if X and Y are not d-separated by Z, then $(X \not\perp\!\!\!\perp Y \mid Z)_P$ for at least one distribution P Markov relative to $\mathcal{G}$; not only that, but *almost all* such distributions show this dependency.

This reflects the importance of d-separability, as it allows us to determine independencies in probability distributions P Markov relative to a DAG $\mathcal{G}$. Given a distribution P and an arbitrary order of its nodes $\mathcal{V} = \{V_1, \ldots, V_K\}$, we can study its independencies by determining the Markov Parents Pa_k of every variable V_k and then defining the corresponding DAG $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, with $\mathbf{V}$ representing every variable in $\mathcal{V}$ and $\mathbf{E}$ defined by the set of Markov Parents, $\mathbf{E} := \{V_l \rightarrow V_k \mid \forall k, \forall V_l \in Pa_k\}$. Note that the resulting graph depends on the initial node ordering.

Something that must be taken into consideration is that the same distribution P can be Markov relative to two different graphs $\mathcal{G}_1, \mathcal{G}_2$, in which case they are said to be **observationally equivalent**. Consider Fig. 1.2, consisting of two graphs with the same variables but different topological order: $\mathcal{V}_1 = \{X, Y, Z\}$, $\mathcal{V}_2 = \{Y, X, Z\}$. Using d-separability, the only independence relationship (conditional or unconditional) that can be inferred from both graphs is $(X \perp\!\!\!\perp Z \mid Y)$, which makes them observationally equivalent. However, were we to perform causal estimation with these graphs, we would arrive at different conclusions; therefore, it is imperative to address this ambiguity before any causal analysis. This is what we explore in the next subsection.



(A) $\mathcal{G}_1$: original ordering. (B) $\mathcal{G}_2$: alternative ordering.

FIGURE 1.2: Two observationally-equivalent graphs.

1.1.3 Interventions and Causal Bayesian Networks

Consider the following two variables: room temperature T and thermometer reading R. Under normal circumstances, T and R output identical results except for small calibration errors, so $T \not\perp\!\!\!\perp R$. Therefore, there are three possible Bayesian Networks describing them, contained in Fig. 1.3: $\mathcal{G} := (T \to R)$, $\mathcal{G}' := (T \leftarrow R)$, or $\mathcal{G}'' := (T \leftarrow Z \to R)$, with Z being an unmeasured variable connecting them. Note that **all three graphs are equally valid from an observational standpoint**, since $T \not\perp\!\!\!\perp R$ in all of them.

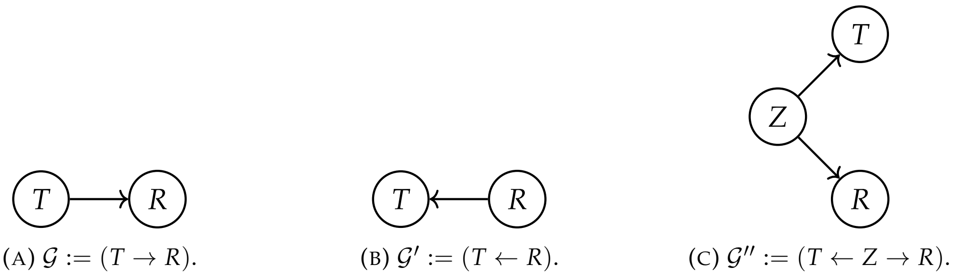

(A) $\mathcal{G} := (T \to R)$. (B) $\mathcal{G}' := (T \leftarrow R)$. (C) $\mathcal{G}'' := (T \leftarrow Z \to R)$.

FIGURE 1.3: Observationally equivalent Bayesian Networks for the Thermometer example.

The key concept here is **intervention**: if we were to alter one of these variables, change its value to a predetermined one irrespective of what it would result in naturally, would we see an effect on the other variables? Let us say that we *intervene* on R by putting the thermometer in the freezer at a certain temperature r. We denote this by $do(R = r)$, and we are now interested in the distribution of T subject to this intervention; it remains unaltered as, naturally, putting a thermometer in the freezer would not alter room temperature. As such, $P(T \mid do(R = r)) = P(T)$. The same cannot be said about the opposite case: if we intervene on T by turning on the room heating to a certain temperature t, we would observe a change in the thermometer's distribution $P(R \mid do(T = t)) = P(R \mid T = t)$.

This asymmetry, where intervening one variable affects the other but not the other way around, provides information about the independencies of the **intervened distributions and graphs**. Given a variable X intervened with value x, the intervened distribution is denoted by P_x and the intervened graph by $\mathcal{G}_x$[1].

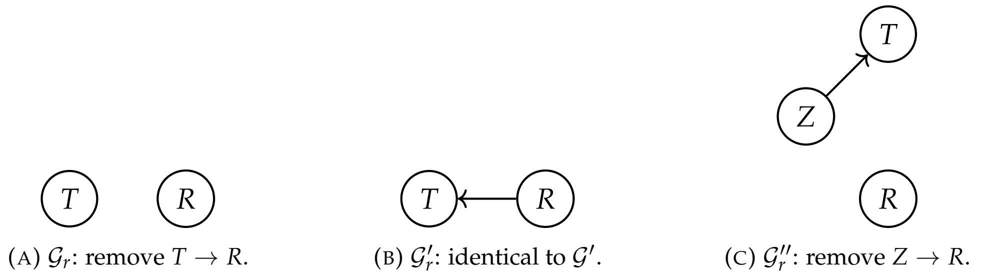
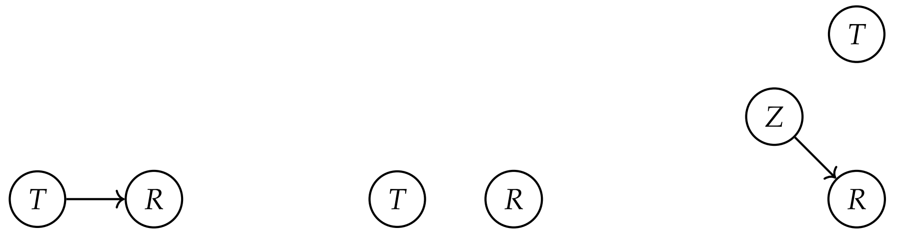

(A) $\mathcal{G}_r$: remove $T \to R$. (B) $\mathcal{G}'_r$: identical to $\mathcal{G}'$. (C) $\mathcal{G}''_r$: remove $Z \to R$.

FIGURE 1.4: Intervention $do(R = r)$ on the graphs from figure 1.3.

[1]We normally write P_x or $\mathcal{G}_x$ instead of $P_{do(X=x)}$ or $\mathcal{G}_{do(X=x)}$ for economy of notation, unless it leads to ambiguity.



(A) $\mathcal{G}_t$: identical to $\mathcal{G}$. (B) $\mathcal{G}'_t$: remove $T \leftarrow R$. (C) $\mathcal{G}''_t$: remove $Z \rightarrow T$.

FIGURE 1.5: Intervention $do(T = t)$ on the graphs from figure 1.3.

Let us determine the shape of all three possible graphs $\mathcal{G}, \mathcal{G}', \mathcal{G}''$ subject to interventions $do(R = r)$ on the thermometer and $do(T = t)$ on room temperature. In general, when intervening a variable X, the natural process with which we sample values from it is altered by one that just samples a single value, x, irrespective of what the other variables in the graph say. Such an intervention *mutilates* the graph by removing any edges that point towards the intervened variable (the reasoning being that its parents do not affect the intervened variable's value, as it is now fixed by something else); figures 1.4 and 1.5 show the resulting intervened graphs. From each of these, we can infer independence relationships in the intervened distributions, which lets us discern which is the correct Bayesian Network in this case. $\mathcal{G}'_r$ tells us that $(T \not\perp\!\!\!\perp R)_{\mathcal{G}'_r}$, which we know is not true; this discards $\mathcal{G}'$. $\mathcal{G}''_t$ proves that $(T \perp\!\!\!\perp R)_{\mathcal{G}''_t}$, which again is not true since $P(R \mid do(T = t)) = P(R \mid T = t) \neq P(R)$. However, $\mathcal{G}_r$ entails $(T \perp\!\!\!\perp R)_{\mathcal{G}_r}$ and $\mathcal{G}_t$ entails $(T \not\perp\!\!\!\perp R)_{\mathcal{G}_t}$; therefore, P, P_r and P_t are Markov relative to $\mathcal{G}$, $\mathcal{G}_r$ and $\mathcal{G}_t$, respectively.

This example illustrates what a Bayesian Network needs to properly represent the effect of interventions. Consider a set of variables $\mathcal{V}$ and a distribution $P(\mathcal{V})$. We define a **Causal Bayesian Network** (CBN) compatible with P to be any DAG $\mathcal{G}$ such that, for all subsets $\varnothing \subseteq \mathbf{X} \subseteq \mathcal{V}$ and interventions $do(\mathbf{X} = \mathbf{x})$:

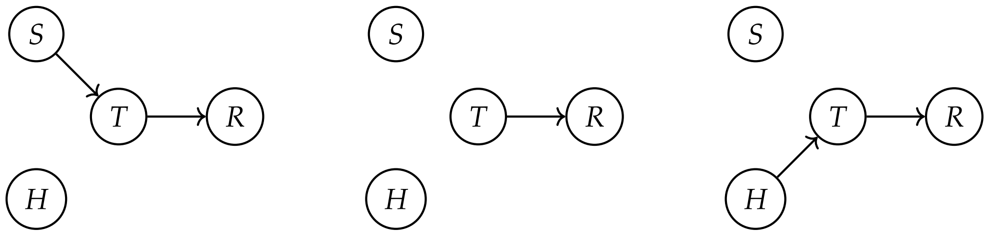
1. $P_{\mathbf{x}}$ is Markov relative to $\mathcal{G}_{\mathbf{x}}$.

2. $P_{\mathbf{x}}(v_k) = 1$ for all $V_k \in \mathbf{X}$ whenever v_k is consistent with $\mathbf{X} = \mathbf{x}$.

3. $P_{\mathbf{x}}(v_k \mid pa_k) = P(v_k \mid pa_k)$ for all $V_k \notin \mathbf{X}$ whenever pa_k is consistent with $\mathbf{X} = \mathbf{x}$.

This criterion allows us to filter any Bayesian Networks that do not follow the interventional structure underlying P. From the resulting graph, we can ascertain independences between the variables in the intervened distributions, which is essential for Causal Estimation.

One last important aspect is that interventions need not be **constant**, such as $do(\mathbf{X} = \mathbf{x})$. Consider the example in figure 1.6; season S affects room temperature T, which in turn affects thermometer reading R, and we have an additional variable H, people at home, that tells us if there is anyone present. Let us assume the effect of people on temperature negligible. We can now study an intervention $do(T = t)$, constant, produced by a thermostat that enables the room heating to bring temperature up to a certain stable t; that results in graph 1.6b. However, we could also have a *smart* thermostat provided with a sensor that sets temperature to t when people are present in the room, $H = 1$, or reduces it to t' to save energy when not, $H = 0$. We denote this behaviour by a deterministic function, $f(H) := t \cdot H + t' \cdot (1 - H)$, and we intervene variable T with this function f, resulting in graph 1.6c. Note how in this case the intervention still removes the edge $S \to T$ but creates a new edge $H \to T$ given that H acts a non-negligible input of f. Therefore, one must consider interventions not as simple value assignments, but as **replacements of the sampling process** that generates values for these r. v. s.

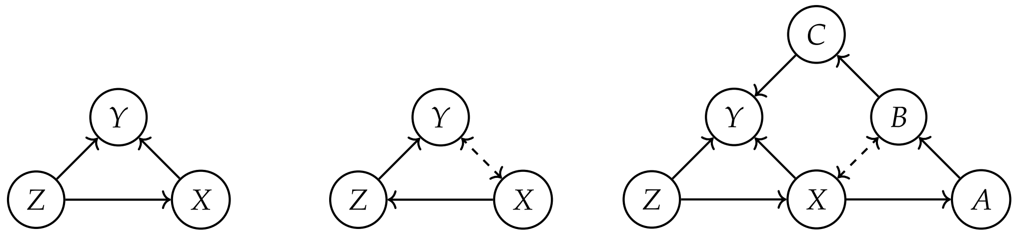


(A) Natural process. (B) Constant Thermostat: $do(T = t)$. (C) Smart Thermostat: $do(T = f(H))$.

FIGURE 1.6: Thermostat example. Variables: season S, room temperature T, thermometer reading R and people at home H.

1.1.4 Identifiability and *do*-calculus

We will now explicit the connection between Causal Estimation and CBNs. Our main object of study is the **query**, a probabilistic expression $\mathcal{Q}$ defined on a distribution P compatible with a CBN $\mathcal{G}$. We will illustrate how the estimation of the **causal effect of** X **on** Y, the query $\mathcal{Q} := P(Y \mid do(X = x)) = P_x(Y)$, varies depending on which CBN is compatible with the distribution P; see Fig. 1.7. Note the use of dashed bidirectional arrows between X and Y in graph 1.7b, which denotes the existence of a third variable that hasn't been measured, a **latent variable**, $U_{\{X,Y\}}$, that acts as a fork $X \leftarrow U_{\{X,Y\}} \to Y$ for the other two. We omit these variables in the graph representation for economy of notation, instead denoting them by dashed bidirectional edges, but the graph remains a DAG, as they translate into directed edges.



(A) $\mathcal{G}_1$: back-door example. (B) $\mathcal{G}_2$: front-door example. (C) $\mathcal{G}_3$: do-calculus example.

FIGURE 1.7: Estimand derivation examples.

Note that $do(X = x)$ embeds $\mathcal{Q}$ into P_x, which in general is different from P. This means that $\mathcal{Q}$ cannot be estimated directly from P unless we perform a random experiment to obtain data from P_x. The common alternative is to transform this query into an **estimand**, an equivalent formula consisting only of observational terms, which can be derived from the observational P. Causal Inference [1, Chapter 3] provides tools to achieve this, which we will explain in this subsection.

Theorem 1 (Back-Door Adjustment)
*Given a probability distribution P compatible with a CBN $\mathcal{G}$ and three disjoint sets of variables $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \subset \mathcal{V}$, we say that $\boldsymbol{Z}$ satisfies the **Back-Door Criterion** relative to $\boldsymbol{X}$ and $\boldsymbol{Y}$ if, $\forall X \in \boldsymbol{X}, \forall Y \in \boldsymbol{Y}$:*

1. *No node in $\boldsymbol{Z}$ is a descendant of X.*
2. *$\boldsymbol{Z}$ d-separates every **back-door path** from X to Y (i.e., a path between X and Y that contains an arrow into X, $X \leftarrow \cdots$).*

If that is the case,

$$P_{\boldsymbol{x}}(\boldsymbol{Y}) = \mathbb{E}_{\boldsymbol{Z}}\left[P(\boldsymbol{Y} \mid \boldsymbol{x}, \boldsymbol{Z})\right]. \tag{1.2}$$

This formula allows us to estimate $\mathcal{Q}$ for the graph $\mathcal{G}_1$ in Fig. 1.7a. Since Z is not a descendant of X and the only back-door path between X and Y ($X \leftarrow Z \rightarrow Y$) is d-separated by Z (fork), Z satisfies the Back-Door Criterion. As a result, $\mathcal{Q}$, a causal term, can be transformed into observational terms, $\mathcal{Q} := P_x(Y) = \mathbb{E}_Z\left[P(Y \mid x, Z)\right]$, which in turn can be estimated using P. For example, consider a dataset $\mathcal{D} = (\mathbf{v}^{(i)})_{i=1..N}$ consisting of N i. i. d. samples $\mathbf{v}^{(i)} = (z^{(i)}, x^{(i)}, y^{(i)})$. The expectation can be estimated with Monte Carlo, using these N samples, while $P(Y \mid x, Z)$ can be modelled with any Density Estimation method. Let us say, for simplicity, that Y is a Bernoulli r. v. and that we model this term using Logistic Regression trained with the dataset $\mathcal{D}$, with X and Z as inputs and Y as the output. Then,

$$\mathcal{Q}_{\mathcal{G}_1} = \mathbb{E}_Z\left[P(Y \mid x, Z)\right] \approx \frac{1}{N}\sum_{i=1}^{N} P(Y \mid x, z^{(i)}).$$

Theorem 2 (Front-Door Adjustment)
*Given a probability distribution P compatible with a CBN $\mathcal{G}$ and three disjoint sets of variables $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \subset \mathcal{V}$, we say that $\boldsymbol{Z}$ satisfies the **Front-Door Criterion** relative to $\boldsymbol{X}$ and $\boldsymbol{Y}$ if, $\forall X \in \boldsymbol{X}, \forall Y \in \boldsymbol{Y}$,*

1. *$\boldsymbol{Z}$ d-separates all directed paths from X to Y.*
2. *All back-door paths from X to $\boldsymbol{Z}$ are blocked.*
3. *All back-door paths from $\boldsymbol{Z}$ to Y are blocked by X.*

If this criterion is satisfied and $P(\boldsymbol{X}, \boldsymbol{Z}) > 0$, then:

$$P_{\boldsymbol{x}}(\boldsymbol{Y}) = \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{x}}\left[\mathbb{E}_{\boldsymbol{X}}\left[P(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{Z})\right]\right]. \tag{1.3}$$

We can now estimate $\mathcal{Q}$ for $\mathcal{G}_2$ in Fig. 1.7b. Note that we cannot use the Back-Door Criterion on $U_{\{X,Y\}}$ because it is an unmeasured variable, so we cannot condition on it. However, Z satisfies the Front-Door Criterion since: 1) Z blocks the only directed path from X to Y (chain), 2) the only back-door path from X to Z ($X \leftarrow U_{\{X,Y\}} \rightarrow Y \leftarrow Z$) is blocked by Y (collider) and 3) the only back-door path from Z to Y ($Z \leftarrow X \leftarrow U_{\{X,Y\}} \rightarrow Y$) is blocked by X (chain). As a result, we can apply the Front-Door formula, which results in an estimand for $\mathcal{Q}$ that can again be estimated with Density Estimation methods and Monte Carlo.

Finally, we can introduce the last tool at our disposal: ***do*-calculus**, a set of three basic inference rules to transform interventional queries. Let us define some notation to simplify the following statements. Given a CBN described by a graph $\mathcal{G}$ with Markov relative probability P and disjoint subsets of variables $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subset \mathcal{V}$, we use $\mathcal{G}_{\overline{\mathbf{X}}}$ to refer to the removal of incoming arrows to nodes $X \in \mathbf{X}$ ($\cdot \rightarrow X$) and $\mathcal{G}_{\underline{\mathbf{X}}}$ to the removal of outgoing arrows from nodes $X \in \mathbf{X}$ ($X \rightarrow \cdot$); hence, $\mathcal{G}_{\overline{\mathbf{X}}\underline{\mathbf{Z}}}$ represents graph $\mathcal{G}$ without all incoming arrows to $\mathbf{X}$ and all outgoing arrows from $\mathbf{Z}$.

Theorem 3 (*do*-calculus)
For any disjoint subsets of variables $\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{W} \subset \mathcal{V}$ ($\mathbf{X}$ and $\mathbf{W}$ possibly empty):

1. *Insertion/deletion of observations:*

$$P_x(\mathbf{Y} \mid \mathbf{Z}, \mathbf{W}) = P_x(\mathbf{Y} \mid \mathbf{W}) \quad \textit{if } (\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}, \mathbf{W})_{\mathcal{G}_{\overline{\mathbf{X}}}}. \tag{1.4}$$

2. *Exchange of interventions/observations:*

$$P_{x,z}(\mathbf{Y} \mid \mathbf{W}) = P_x(\mathbf{Y} \mid z, \mathbf{W}) \quad \textit{if } (\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}, \mathbf{W})_{\mathcal{G}_{\overline{\mathbf{X}}\underline{\mathbf{Z}}}}. \tag{1.5}$$

3. *Insertion/deletion of interventions:*

$$P_{x,z}(\mathbf{Y} \mid \mathbf{W}) = P_x(\mathbf{Y} \mid \mathbf{W}) \quad \textit{if } (\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}, \mathbf{W})_{\mathcal{G}_{\overline{\mathbf{X} \cup \mathbf{Z}(\mathbf{W})}}}, \tag{1.6}$$

where $\mathbf{Z}(\mathbf{W}) := \mathbf{Z} \setminus An_{\mathcal{G}_{\overline{\mathbf{X}}}}(\mathbf{W})$, the set of nodes in $\mathbf{Z}$ that are not ancestors of $\mathbf{W}$ (including $\mathbf{W}$) in graph $\mathcal{G}_{\overline{\mathbf{X}}}$.

This set of rules (R1-3) constitutes the base of the so-called ***do*-calculus**. Let us derive an estimand of $\mathcal{Q}$ for graph $\mathcal{G}_3$ in Fig. 1.7c.

$$\begin{aligned}
P_x(Y) = &\mathbb{E}_{Z,C|do(X=x)}\left[P_x(Y \mid Z, C)\right] = \\
&\mathbb{E}_{Z,C|do(X=x)}\left[P(Y \mid x, Z, C)\right] = \\
&\mathbb{E}_{Z,C}\left[P(Y \mid x, Z, C) \cdot \frac{P_x(Z,C)}{P(Z,C)}\right] = \\
&\mathbb{E}_{Z,C}\left[\frac{P(Y \mid x, Z, C)}{P(Z,C)} \cdot \mathbb{E}_{A|x}\left[\mathbb{E}_X\left[P(Z, C \mid X, A)\right]\right]\right].
\end{aligned}$$

The first step comes from the marginalization of (Z, C) on the distribution P_x. Next, we can transform P_x into $P(\cdot \mid x)$ using R2, since $(Y \perp\!\!\!\perp X \mid Z, C)_{\mathcal{G}_{\underline{X}}}$. Then, we use **importance sampling**[2] to transform the expectation on $P_x(Z, C)$ to $P(Z, C)$, and finally we derive $P_x(Z, C)$ into the new expectation at the last line by the Front-Door Adjustment Formula, since A satisfies the criterion.

The sequential application of the rules of probability and *do*-calculus transforms causal queries that operate on interventional distributions P_x into formulas that only require the observational distribution P; this process is called **identification**, and plays a central role in Causal Query Estimation, both for estimand-based and estimand-agnostic approaches. Once we derive an estimand, we can estimate the query with an i. i. d. dataset as before.

We can also prove the Back-Door and Front-Door Criterions with *do*-calculus.

Proof of Theorem 1.
Given conditions C1-2 from the Back-Door Criterion on a distribution P with three disjoint sets $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subset \mathcal{V}$,

$$P_{\mathbf{x}}(\mathbf{Y}) = \mathbb{E}_{\mathbf{Z}|do(\mathbf{X}=\mathbf{x})}\left[P_{\mathbf{x}}(\mathbf{Y} \mid \mathbf{Z})\right] = \mathbb{E}_{\mathbf{Z}|do(\mathbf{X}=\mathbf{x})}\left[P(\mathbf{Y} \mid \mathbf{x}, \mathbf{Z})\right] = \mathbb{E}_{\mathbf{Z}}\left[P(\mathbf{Y} \mid \mathbf{x}, \mathbf{Z})\right].$$

We marginalize $P_{\mathbf{x}}(\mathbf{Z})$ on the first step. Then we can use R2 with $(\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z})_{\mathcal{G}_{\underline{\mathbf{X}}}}$ so that $P_{\mathbf{x}}(\mathbf{Y} \mid \mathbf{Z}) = P(\mathbf{Y} \mid \mathbf{x}, \mathbf{Z})$, since any path between $\mathbf{X}$ and $\mathbf{Y}$ in $\mathcal{G}_{\underline{\mathbf{X}}}$ must be a back-door path, which we know are blocked (C2). Finally, $(\mathbf{Z} \perp\!\!\!\perp \mathbf{X})_{\mathcal{G}_{\overline{\mathbf{X}}}}$ and by R3, $P_{\mathbf{x}}(\mathbf{Z}) = P(\mathbf{Z})$, because any path from $\mathbf{X}$ to $\mathbf{Z}$ in $\mathcal{G}_{\overline{\mathbf{X}}}$ must either contain a collider (therefore blocked) or be a directed path, which is impossible (C1, $\mathbf{Z} \notin De(\mathbf{X})$). □

Proof of Theorem 2.
Given conditions C1-3 from the Front-Door Criterion on a distribution P with three disjoint sets $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subset \mathcal{V}$,

$$P_{\mathbf{x}}(\mathbf{Y}) = \mathbb{E}_{\mathbf{Z}|do(\mathbf{X}=\mathbf{x})}\left[P_{\mathbf{x}}(\mathbf{Y} \mid \mathbf{Z})\right] = \mathbb{E}_{\mathbf{Z}|\mathbf{x}}\left[P_{\mathbf{x}}(\mathbf{Y} \mid \mathbf{Z})\right].$$

Again, we marginalize $P_{\mathbf{x}}(\mathbf{Z})$ on the first equality. Then, by R2, knowing that $(\mathbf{Z} \perp\!\!\!\perp \mathbf{X})_{\mathcal{G}_{\underline{\mathbf{X}}}}$ ($\mathcal{G}_{\underline{\mathbf{X}}}$ can only contain back-door paths from X to Z, all blocked by C2), then $P_{\mathbf{x}}(\mathbf{Z}) = P(\mathbf{Z} \mid \mathbf{x})$, which lets us change the expectation. The $P_{\mathbf{x}}(\mathbf{Y} \mid \mathbf{Z})$ term requires more steps.

$$P_{\mathbf{x}}(\mathbf{Y} \mid \mathbf{Z}) = P_{\mathbf{x},\mathbf{Z}}(\mathbf{Y}) = P_{\mathbf{Z}}(\mathbf{Y}) = \mathbb{E}_{\mathbf{X}|do(\mathbf{Z}=\mathbf{Z})}\left[P_{\mathbf{Z}}(\mathbf{Y} \mid \mathbf{X})\right] = \mathbb{E}_{\mathbf{X}}\left[P(\mathbf{Y} \mid \mathbf{X}, \mathbf{Z})\right].$$

Note that the uppercase $\mathbf{Z}$ here is a value that comes from the marginalization in the previous equation. Firstly, to transform $P_{\mathbf{x}}(\mathbf{Y} \mid \mathbf{Z})$ into $P_{\mathbf{x},\mathbf{Z}}(\mathbf{Y})$, we apply R2 with $(\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X})_{\mathcal{G}_{\overline{\mathbf{X}}\underline{\mathbf{Z}}}}$, since we only have back-door paths from $\mathbf{Z}$ to $\mathbf{Y}$ in $\mathcal{G}_{\overline{\mathbf{X}}\underline{\mathbf{Z}}}$, which

[2]**Importance sampling**. For any pair of disjoint sets of r. v. s $\mathbf{X}, \mathbf{Z}$ and an arbitrary function $f(\mathbf{X})$, $\mathbb{E}_{\mathbf{X}|\mathbf{z}}\left[f(\mathbf{X})\right] = \sum_{\mathbf{x}} f(\mathbf{x}) P(\mathbf{x} \mid \mathbf{z}) = \sum_{\mathbf{x}} f(\mathbf{x}) \frac{P(\mathbf{x})}{P(\mathbf{x})} P(\mathbf{x} \mid \mathbf{z}) = \mathbb{E}_{\mathbf{X}}\left[f(\mathbf{X}) \frac{P(\mathbf{X}|\mathbf{z})}{P(\mathbf{X})}\right]$. The continuous case is analogous. We use this technique throughout this work to derive expectations on conditional distributions.

are blocked by $\mathbf{X}$ (C3). Secondly, we can apply R3 with $(\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z})_{\mathcal{G}_{\overline{\mathbf{ZX}}}}$ to remove $do(\mathbf{X} = \mathbf{x})$, because we only have front-door paths from $\mathbf{X}$ to $\mathbf{Y}$ in $\mathcal{G}_{\overline{\mathbf{ZX}}}$ and these are either directed, which are controlled by $\mathbf{Z}$ (C1), or have a collider of which $\mathbf{Z}$ cannot be a descendant (the incoming edges of $\mathbf{Z}$ are removed in $\mathcal{G}_{\overline{\mathbf{ZX}}}$). Next, we marginalize by $P_{\mathbf{Z}}(\mathbf{X})$, and we solve the last equality in two steps. $P_{\mathbf{Z}}(\mathbf{X}) = P(\mathbf{X})$ by R3, since $(\mathbf{X} \perp\!\!\!\perp \mathbf{Z})_{\mathcal{G}_{\overline{\mathbf{Z}}}}$ due to C2 blocking any back-door paths from $\mathbf{X}$ to $\mathbf{Z}$ and any front-door paths being either directed, which is not possible in $\mathcal{G}_{\overline{\mathbf{Z}}}$, or having an uncontrolled collider, which blocks the path. Finally, $P_{\mathbf{Z}}(\mathbf{Y} \mid \mathbf{X}) = P(\mathbf{Y} \mid \mathbf{Z}, \mathbf{X})$ by R2, since $(\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X})_{\mathcal{G}_{\underline{\mathbf{Z}}}}$ because all back-door paths from $\mathbf{Z}$ to $\mathbf{Y}$ are blocked by $\mathbf{X}$ (C3). Putting it all together, we arrive at formula 1.3, which proves the theorem. □

It can be quite difficult to derive an estimand by hand for complex CBNs $\mathcal{G}$, as it usually involves a set of several steps using the rules of *do*-calculus, with no guidance on which approach would help us transform the causal terms into observational ones. However, there are three **algorithms** that automatically derive estimands for particular types of queries, if they exist, or demonstrate that such an estimand cannot exist otherwise: [3] for $\mathcal{Q} := P_{\mathbf{x}}(\mathbf{Y})$, [4] for $\mathcal{Q} := P_{\mathbf{x}}(\mathbf{Y} \mid \mathbf{Z})$ or [5] for *counterfactual* queries (see section 1.2.2). These algorithms can be quite complex to apply manually, but fortunately there are implementations for the first two in R and Python, with [6] and [7], respectively.

We now have all the required tools for the estimation of Causal Queries. Given an i. i. d. dataset $\mathcal{D}$ following a distribution P compatible to a CBN $\mathcal{G}$, and assuming we know that graph structure, **estimand-based** approaches consist of transforming the target query $\mathcal{Q}$ into an estimand through *do*-calculus, and once that estimand has been found, dataset statistics are computed and Density Estimation models are trained to answer each of the terms in the estimand. With this, we can finally compute an estimate of the causal query only using observational data.

1.1.5 Related Work

In the following, we will summarize recent estimand-based approaches, specifically the ones that have measured their performance against the most standard datasets on causal estimation, which we will cover in chapter 6. These methods operate with the Potential Outcomes Perspective [2] and three sets of variables: $\mathbf{X}$, the confounding variables; T, the treatment (usually binary); and Y, the target variable, for which there are two potential outcomes Y_0 or Y_1 depending on which treatment $t = 0$ or $t = 1$ is applied. Additionally, they follow the common assumptions of *Consistency* (i.e., the potential outcome Y_t given the observed treatment t is equivalent to the observed outcome Y) and *Strong Ignorability* (i.e., conditioning on $\mathbf{X}$, the factual and counterfactual outcomes of Y are independent of the observed treatment t). Every one of the following methods focuses on estimating the Individual Treatment Effect (ITE), $\mathcal{Q} := \mathbb{E}\left[Y_1 - Y_0 \mid \mathbf{x}^{(i)}\right]$ for a particular individual/sample i in our i. i. d. dataset $\mathcal{D}$. Given the previous two assumptions,

$\mathcal{Q} = \mathbb{E}\left[Y \mid \mathbf{x}^{(i)}, T = 1\right] - \mathbb{E}\left[Y \mid \mathbf{x}^{(i)}, T = 0\right]$, with both terms estimated by a model $f(\mathbf{x}, t) \approx \mathbb{E}\left[Y \mid \mathbf{X} = \mathbf{x}, T = t\right]$ to be trained.

The immediate approach is to apply a standard Machine Learning Regressor or Classifier (depending on which kind of r. v. Y is). However, $(X \not\perp\!\!\!\perp T)$, so certain clusters of X are more likely to receive treatment than others, making the estimation of this function a problem akin to domain adaptation; that is the approach taken by most of the following techniques. Balancing Neural Network (BNN) [8] learns a representation $h(\mathbf{X})$ used by a prediction network $f(h(\mathbf{X}), t)$, forcing these representations to adapt to both treatments. Treatment-agnostic Representation Network (TARNet) [9] goes a step further by adding an Integral Probability Metric (e.g., Maximum Mean Discrepancy or Wasserstein Distance) on the representations of both groups as regularization for the model training, in order to make these representations independent of the treatment. Adaptively similarity-preserved representation learning for Causal Effect estimation (ACE) [10] adopts the previous ideas and adds further regularization to ensure that similar units in terms of $\mathbf{X}$ have a stronger impact on the eventual prediction of their peers. Subspace Learning Based Counterfactual Inference (SCI) [11] includes an additional subspace to the representation layer that takes into account both $\mathbf{X}$ and T, so as to exploit treatment-agnostic and treatment-aware representations. Finally, Causal Optimal Transport (CausalOT) [12] employs Optimal Transport Theory in order to alleviate the issues derived from limited overlapping between the treated and control subpopulations.

Every method listed here is focused on a particular query for a particular kind of graph that fulfills the given assumptions. This makes these methods difficult to apply to different settings without extensive adjustments, demonstrating the ad hoc nature of estimand-based frameworks. In the following section, we discuss the **limitations** of CE, and particularly of estimand-based approaches.

1.1.6 Assumptions and Limitations

First and foremost, to perform any kind of causal estimation, **we need to know the corresponding CBN $\mathcal{G}$ for a distribution** P, but, in general, observational data alone cannot distinguish between different causal scenarios: multiple graphs can be compatible to an observational distribution P despite not being compatible with an interventional $P_{\mathbf{x}}$. Therefore, finding an appropriate graph for P's distribution is a prerequisite for any kind of estimation, with or without an estimand.

Secondly, some queries, applied to certain graphs $\mathcal{G}$, might not be **identifiable** (i.e., no estimand exists). When that is the case, we need to make some adjustments to our data collection process to ensure that the query becomes identifiable: for example, we can measure additional variables that block certain paths in the graph, allowing us to transform one of the causal terms into an observational one; alternatively, we can perform a random experiment, either for the query itself, or for one of the derived sub-terms in the estimand that we cannot identify otherwise. On the other hand, some CE approaches include additional assumptions on the data distribution P (e.g., functional restrictions to the sampling process of its r. v. s, such as linear equations), which simplifies the identification process and makes the query identifiable. While these restrictions extend the applicability of the techniques to other graphs, their assumptions cannot apply to any data distribution. Nevertheless, both the estimand-based and estimand-agnostic approaches suffer from this problem, because if the query is not identifiable, neither of them can provide a reliable answer to the query without some adjustments.

Note that every new estimand for a given causal query requires a careful design of the different Density Estimation models needed to estimate its terms, and certain graph-query combinations may result in **complex estimands**, which are harder to employ. As an example, consider $\mathcal{Q} := P_x(Y)$ on graph 1.7c, discussed in section 1.1.4; its estimand requires the computation of at least three different probability terms and three nested expectations, one of them conditional. Consequently, estimand-based techniques can get significantly more complex depending on the graph structure, and result in highly specific models that are hard to adapt to any other settings.

On the other hand, the exact same query can result in vastly different estimands depending on the underlying graph structure, as we saw in the aforementioned example of Fig. 1.7; in other words, estimand-based approaches lead to **completely different models to answer the same query**. In contrast, estimand-agnostic approaches learn a model of the observational distribution following the causal structure imposed by the CBN, resulting in a proxy of the underlying data generating process, with which we can answer any queries pertaining to that system, as long as they are identifiable in the graph. These queries can be estimated through general procedures applicable to arbitrary graphs, and so, the same query, despite resulting in completely different estimands, can be resolved with the same technique by using estimand-agnostic approaches.

Finally, since estimand-based models are designed specifically for a particular dataset and query, it is possible to optimize the estimand's density models to deliver better estimation performance on that particular case. In contrast, estimand-agnostic approaches learn a model of the data distribution and then answer any query from that single model, but that makes it harder to optimize specifically for a particular expression. This **trade-off between specificity** (estimand-based) **and flexibility** (estimand-agnostic) is specially relevant. For sensitive estimations, where the priority is to minimize estimation error, estimand-based techniques might be preferable. On the other hand, certain applications such as Black-Box Introspection (discussed in section 1.3) benefit from more flexibility in their application, in order to interrogate the system with many different queries using a single model.

1.2 Estimand-Agnostic Causal Estimation

We devote this section to the discussion of estimand-agnostic approaches. We begin by introducing Structural Causal Models (SCMs), the main concept behind this framework, followed by the definition of counterfactuals in the context of SCMs, using the Parallel Worlds Graph framework. We continue by discussing SCM training and the role of identifiability in SCM-based estimation, list the desiderata for estimand-agnostic approaches and evaluate the state of the art with these requirements, and finish by explaining their limitations.

1.2.1 Structural Causal Models

We define a **Structural Causal Model** as the tuple $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{U}, \mathcal{P}, \mathcal{F})$ with:

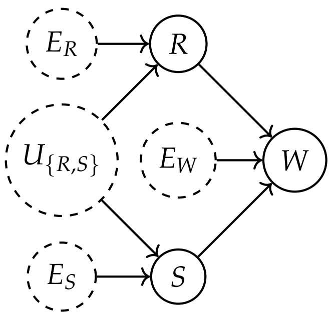
- $\mathcal{V} := \{V_1, \ldots, V_K\}$ **measured variables**, for which we have observed samples in our dataset.
- $\mathcal{E} := \{E_1, \ldots, E_K\}$ **exogenous noise signals**, one for each V_k, which provide stochasticity to the otherwise deterministic functions $f_k \in \mathcal{F}$.
- $\mathcal{U} \subseteq \{U_{\{k,l\}} \mid k, l = 1..K, k \neq l\}$ **latent confounder variables**, $U_{\{k,l\}} := U_{\{V_k, V_l\}}$, that correlate pairs[3] of measured variables V_k, V_l. Can be empty.
- $\mathcal{P}(\mathcal{E}, \mathcal{U})$ **prior distribution** for both sets of latent variables. $\mathcal{E}$ and $\mathcal{U}$ are mutually and internally independent: $\mathcal{P}(\mathcal{E}, \mathcal{U}) = \prod_{E \in \mathcal{E}} \mathcal{P}(E) \cdot \prod_{U \in \mathcal{U}} \mathcal{P}(U)$.
- $\mathcal{F} := \{f_k \mid V_k := f_k(Pa_k, U_{\{k,\cdot\}}, E_k)\}_{k=1..K}$ **functional assignments** describing the relationship between each variable $V_k \in \mathcal{V}$ and its corresponding E_k, its confounders $U_{\{k,\cdot\}}$ and its measured parents $Pa_k \subset \mathcal{V} \setminus \{V_k\}$.

Let us denote $Pa'_k := Pa_k \cup U_{\{k,\cdot\}}$, the set of V_k's Markov parents[4]. The relationships between the inputs in each f_k and the resulting variable V_k define a directed **graph structure** $\mathcal{G}_\mathcal{M} = (\mathbf{V}, \mathbf{E})$ with nodes $\mathbf{V} := \mathcal{V} \cup \mathcal{E} \cup \mathcal{U}$ (every measured and latent variable) and edges $\mathbf{E}$ connecting every input-output pair in $\mathcal{F}$, $\mathbf{E} := \{X \to V_k \mid \forall k, \forall X \in Pa'_k \cup \{E_k\}\}$. In this work, we focus on SCMs $\mathcal{M}$ defined by **Directed Acyclic Graphs** (DAGs) $\mathcal{G}_\mathcal{M}$ with $\mathcal{V}$ listed in topological order.

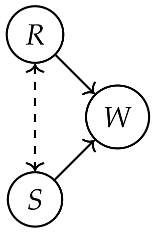
By way of example, let us define an SCM $\mathcal{M}$ with variables Rain (R, "Has it rained in the last 24 hours?"), Sprinkler (S, "Did the user activate the sprinkler yesterday?") and Wet (W, "Is the grass wet?"). There is a confounder between Rain and

[3] This definition of SCMs limits its structure to latent confounders that are root nodes with exactly two children. Tian *et al.* [13] show that for arbitrary latent confounders (e.g., non-root latent confounders in between measured variables, or root confounders with more than two descendants) we can *project* them onto a new set of confounders that follows our restriction. This projection preserves the set of d-independences between measured variables; therefore, it does not preclude the set of queries we can compute from the model, nor their validity.

[4]Note that in presence of latent confounders ($\mathcal{U} \neq \varnothing$), $Pa_k \subseteq \mathcal{V}$ are not always the Markov parents of V_k, since $U_{\{k,\cdot\}}$ create dependencies between some variables in $V_{<k}$, but Pa'_k always are.



(A) Explicit graph.



(B) Implicit graph.

FIGURE 1.8: Sprinkler example. We normally omit the variables in $\mathcal{E}$ (they are considered implicit) and the latent confounders $\mathcal{U}$ are represented by dashed bi-directional edges between both affected variables. The explicit, unfolded representation (A) is usually summarized by means of the implicit graph (B)

Sprinkler that has not been measured (and hence is latent), called Weather ($U_{\{Z,X\}}$), which affects both rain and the probability of activating the sprinkler. The resulting graph $\mathcal{G}$ is the one in Fig. 1.8a; consequently, $\mathcal{V} := \{R, S, W\}$, $\mathcal{E} := \{E_R, E_S, E_W\}$, $\mathcal{U} := \{U_{\{R,S\}}\}$. We normally omit $\mathcal{E}$ from the graph, since every observable variable implicitly has its own, and any latent confounders $U \in \mathcal{U}$ are represented by bi-directed dashed edges between both affected nodes; Fig. 1.8b shows the corresponding **implicit graph**.

An SCM describes a **sampling procedure** through its latent priors $\mathcal{P}$ and its functional relationships $\mathcal{F}$: we take a sample $(\varepsilon, u) \sim \mathcal{P}(\mathcal{E}, \mathcal{U})$ and then progressively apply each function $f_k \in \mathcal{F}$ to generate a new value for V_k. Since $\mathcal{V}$ follows a topological order of the graph, applying each function f_k in order guarantees that we have a value for each of its inputs $(pa_k, u_{\{k,\cdot\}}, \varepsilon_k)$, and can deterministically generate a new value v_k for V_k. In other words, $\mathcal{F}$ is a function from $(\mathcal{E}, \mathcal{U})$ to $\mathcal{V}$; the result is $\mathcal{M}$'s **observational distribution** $\mathcal{P}(\mathcal{V})$.

Due to the SCM's structure, Pa'_k are the Markov Parents of V_k in $\mathcal{P}(\mathcal{V})$, since $(V_k \perp\!\!\!\perp V_{<k} \setminus Pa'_k \mid Pa'_k)$. Consequently, given $\mathcal{V}$ in topological order,

$$P(\mathcal{V}, \mathcal{U}) = \prod_{k=1..K} P(V_k \mid V_{<k}, \mathcal{U})\, P(\mathcal{U}) = \prod_{k=1..K} P(V_k \mid Pa'_k) \prod_{U \in \mathcal{U}} P(U). \tag{1.7}$$

Interventions $do(\mathbf{X} = \mathbf{x})$ result in **intervened SCMs** $\mathcal{M}_{\mathbf{x}}$[5] where, for every $X \in \mathbf{X}$, the corresponding functionals f_X are replaced by the assignment $X := x$. This alters the graph $\mathcal{G}_{\mathcal{M}}$ to $\mathcal{G}_{\mathcal{M}_{\mathbf{x}}}$, where any incoming edges to $\mathbf{X}$ are removed, and results in a new distribution originating from the intervened sampling process, $P_{\mathbf{x}}$.

Continuing with the example from graph 1.8b, we define a set of functions $\mathcal{F}$ that describe a possible sampling mechanism of $\mathcal{M}$. Let us denote $U := U_{\{R,S\}}$ for brevity in this example. Given priors $P(U)$, $P(E_R)$, $P(E_S)$, $P(E_W)$ all following a

[5]For general interventions $do(\mathbf{X} = f(\mathbf{x}))$, we denote the resulting model and graph by $\mathcal{M}_f$ and $\mathcal{G}_{\mathcal{M}_f}$, respectively. Graph $\mathcal{G}_{\mathcal{M}_f}$ replaces any edges leading to the intervened nodes with the new input-output relationships of f.

uniform distribution $\mathcal{U}(0,1)$, we can define an SCM with parametric functions $\mathcal{F}_\Theta$:

$$\begin{cases} r := f_R(u, \varepsilon_R) = \varepsilon_R \leq u \cdot \theta_{R,1} + (1-u) \cdot \theta_{R,0} \\ s := f_S(u, \varepsilon_S) = \varepsilon_S \leq u \cdot \theta_{S,1} + (1-u) \cdot \theta_{S,0} \\ w := f_W(r, s, \varepsilon_W) = \varepsilon_W \leq rs \cdot \theta_{W,1,1} + r(1-s) \cdot \theta_{W,1,0} + \\ \qquad\qquad\qquad\qquad\qquad (1-r)s \cdot \theta_{W,0,1} + (1-r)(1-s) \cdot \theta_{W,0,0} \end{cases}$$

with every binary r, s, w functioning both as a logical value and a $0-1$ value, and every $\theta \in (0,1)$. Let Θ be the set of parameters θ; every configuration of Θ results in a different distribution for $P(\mathcal{V})$, so it is natural to define that distribution as relative to Θ: $P_\Theta(\mathcal{V})$. Note that every functional set $\mathcal{F}_\Theta$ requires a certain domain for its parameters; this is relevant when defining the architecture of our models.

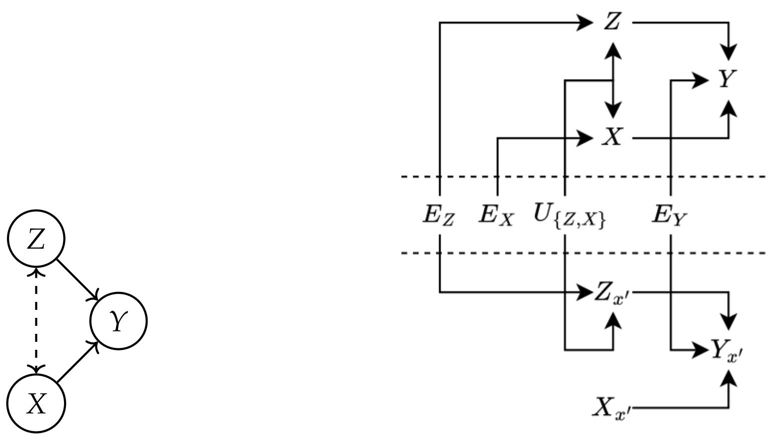
1.2.2 Counterfactuals and the Parallel Worlds Graph

A **counterfactual** is the hypothetical result that an intervention may have on an individual for whom we have already observed a different *factual* outcome. For example, we measured a certain blood sugar level on a patient who was not treated and we want to know what the blood sugar would have been had they taken the treatment. This **parallel** world where certain variables are intervened upon and consequently result in a different outcome is what we call the **counterfactual world**.

Returning to the Sprinkler example with graph 1.8b, let us describe a certain sample $\mathbf{v} = (r, s, w)$, where we observe that the grass is not wet ($w = 0$), even though it did rain yesterday ($r = 1$), but the sprinkler had not been turned on ($s = 0$). Knowing this factual observation gives us insight about the latent variables (maybe it was a particularly hot day after the rain and the water had evaporated since); we extract that information by computing $P(\mathcal{E}, \mathcal{U} \mid \mathbf{v})$, a process called **abduction**. We then study what effect a new **intervention**, turning the Sprinkler on, would have had on the eventual outcome variable, Wet, knowing the *posterior* state of the latent variables thanks to the *factual* observations. This results in a **prediction** of that *counterfactual* outcome, which lets us answer the query "Would the grass be wet had we turned the sprinkler on, knowing that it is dry now, and that it did rain yesterday but we did not turn the sprinkler on?". The three-step process described above, abduction-intervention-prediction, is the counterfactual process defined by Pearl [1], which lets us consider this kind of hypothetical causal query.

Counterfactuals are essential for **explainability** applications [14]: knowing the effects that certain interventions would have had in contrast with the factual outcome we observe allows us to study the effect of these variables for particular individuals. "How would my salary change had I been a man?" is an example of a counterfactual query, where the interest is not on applying the intervention, but in finding the reasons behind a certain outcome, or even the **fairness** of a decision [15].

For the previous example, the counterfactual query consists of estimating an intervention on the modified SCM $\mathcal{M}_{do(S=s')|\mathbf{v}} := (\mathcal{V}, \mathcal{E}, \mathcal{U}, \mathcal{P}(\mathcal{E}, \mathcal{U} \mid \mathbf{v}), \mathcal{F}_{s'})$ with its



(A) Original graph. (B) Parallel worlds graph.

FIGURE 1.9: Parallel World Graph example. Based on the original graph (left), we construct the Parallel Worlds Graph (right) with two subgraphs, the factual (up) and the counterfactual (down), linked by all latent variables. Any edges towards $X_{x'}$ are removed due to the intervention. To better convey the "parallel world" concept, here we do not remove $X_{x'}$ or fuse Z and $Z_{x'}$.

latent priors conditioned on the observed variables $\mathbf{v}$ and its functions $\mathcal{F}$ affected by $do(S = s')$. However, the current notation is ambiguous, as the variables in $\mathcal{V}$ are duplicated between the factual and counterfactual model, and it is essential to distinguish them in order to derive the proper expressions.

An alternative notation is the so-called **Parallel Worlds Graph** [5]; we extend the original SCM $\mathcal{M}$ with a second one, where every variable in $\mathcal{V}$ is replicated (normally denoted by a subscript with the intervention value, e.g., $Y_{x'}$) but with every variable in $\mathcal{E}$ and $\mathcal{U}$ shared between the two. Consider graph 1.9b; E_Y points towards Y and $Y_{x'}$. Normally, for constant interventions $do(\mathbf{X} = \mathbf{x})$, we omit any intervened variable in the counterfactual graph, as they become constant variables; also, any duplicated variables with the exact same parents and distribution should be fused together as one ($Z = Z_{x'}$, since both share $U_{\{Z,X\}}$ as the only parent and f_Z as their functional assignment, therefore having the exact same distribution). However, in this particular figure, we do not omit $X_{x'}$ nor fuse Z and $Z_{x'}$ for clarity of exposition.

When we talk about counterfactuals, we normally condition on some factual outcome $\mathbf{v} \sim P(\mathcal{V})$ and query the variables in the counterfactual world subject to intervention $do(\mathbf{X} = \mathbf{x'})$. Therefore, to bridge the gap between both worlds, we must compute the posterior for every latent variable in $\mathcal{E}$ and $\mathcal{U}$; in the Rain example, given the factual information and an intervention $do(S = s')$, E_W's prior changes, which affects the counterfactual variable $W_{s'}$. Note that $R_{s'}$ is not affected by the intervention, since R is not a descendant of S. We do not need to abduct E_S either since the intervention already provides a constant value for the intervened variable.

1.2.3 SCM-based Estimation

Given an i. i. d. dataset $\mathcal{D} = (v^{(i)})_{i=1..N}$ sampled from a certain distribution P with associated CBN $\mathcal{G}$, if the graph is a DAG, it can be assumed that an underlying SCM $\mathcal{M}$ with graph $\mathcal{G}_{\mathcal{M}} = \mathcal{G}$ generated this dataset. Consider the case where we know the shape of that graph $\mathcal{G}$. We can define our own proxy SCM $\mathcal{M}'$ following that same graph, and given a flexible implementation of $\mathcal{F}_\Theta$ with appropriate priors $P_{\mathcal{M}'}(\mathcal{E},\mathcal{U})$ (both potentially different from the ones in $\mathcal{M}$), we can train this model by finding the appropriate parameters θ such that the resulting distribution $P_\theta(\mathcal{V})$ is a good enough approximation of the original $P(\mathcal{V})$. With this **proxy SCM**, it should be possible to estimate causal queries as if we had the original underlying SCM $\mathcal{M}$.

There are three obstacles with this approach: how to define appropriate functions $\mathcal{F}_\Theta$ and priors $P_{\mathcal{M}'}(\mathcal{E},\mathcal{U})$ that can model $P(\mathcal{V})$ (**model architecture**); how to train the model to find the right values θ so that $P_\theta(\mathcal{V}) \approx P(\mathcal{V})$ (**model training**); and whether the resulting proxy SCM $\mathcal{M}'$ would result in unbiased estimations of the queries (**identifiability**).

Regarding **model architecture**, we take advantage of the fact that neural networks are universal approximators; for every node $V_k \in \mathcal{V}$, we can define $f'_k \in \mathcal{F}_{\mathcal{M}'}$ as an expressive enough neural network, and train the whole DCG to model $P(\mathcal{V})$. We will elaborate on this topic when we define our Deep Causal Unit implementations in chapter 3.

In order to **train** an SCM $\mathcal{M}'$ to adjust to a certain observational distribution $P(\mathcal{V})$ using an i. i. d. dataset $\mathcal{D}$, assuming a flexible enough architecture for $\mathcal{F}'$, we can use Maximum Likelihood Estimation. Given a sample $\mathbf{v} = (v_1, \dots, v_K) \in \mathcal{D}$, we can compute its log-likelihood as:

$$\log P_\theta(\mathbf{v}) = \log \mathbb{E}_{\mathcal{U}}\left[P_\theta(\mathbf{v} \mid \mathcal{U})\right] = \log \mathbb{E}_{\mathcal{U}}\left[\prod_{k=1..K} P_\theta(v_k \mid pa'_k)\right]. \tag{1.8}$$

Therefore, training consists of finding the set of parameters $\widehat{\theta}$ such that:

$$\begin{aligned} \widehat{\theta} &:= \arg\max_\theta \prod_{i=1..N} P_\theta(\mathbf{v}^{(i)}) = \arg\max_\theta \sum_{i=1..N} \log P_\theta(\mathbf{v}^{(i)}) = \\ &\arg\max_\theta \sum_{i=1..N} \log \mathbb{E}_{\mathcal{U}}\left[\exp \sum_{k=1..K} \log P_\theta({v_k}^{(i)} \mid {pa'_k}^{(i)})\right] \approx \\ &\arg\max_\theta \sum_{i=1..N} \log \sum_{j=1..M} \exp \sum_{k=1..K} \log P_\theta({v_k}^{(i)} \mid {pa'_k}^{(i,j)}). \end{aligned} \tag{1.9}$$

We use Monte Carlo to approximate the expectation, taking $N \cdot M$ i. i. d. samples $\mathbf{u}^{(i,j)} \sim P(\mathcal{U})$ that give value to the Markov parents ${pa'_k}^{(i,j)}$ for every pair $({v_{<k}}^{(i)}, \mathbf{u}^{(i,j)})$. We add exp log to the inside of the expectation so that we can employ the log-sum-exp trick[6] for numerical stability. Finally, note that if $\mathcal{U} = \varnothing$ (there are no latent

[6]**Log-sum-exp trick**. Given a set of values $\mathbf{x} := (x^{(i)})_{i=1..N}$ and a constant $C := \max_i x_i$, we can compute the LogSumExp (LSE) of $\mathbf{x}$ as: $\mathrm{LSE}(\mathbf{x}) := \log \sum_{i=1..N} \exp x^{(i)} = C - C + \log \sum_{i=1..N} \exp x^{(i)} =$

confounders) this expectation is not required and simplifies the expression to:

$$\widehat{\theta} := \arg\max_{\theta} \sum_{i=1..N} \log P_\theta(\mathbf{v}^{(i)}) = \arg\max_{\theta} \sum_{i=1..N} \sum_{k=1..K} \log P_\theta({v_k}^{(i)} \mid {pa'_k}^{(i)}). \quad (1.10)$$

Finally, we evaluate the topic of **identifiability** *w.r.t.* proxy SCM estimation. We operate on a finite i. i. d. dataset $\mathcal{D}$ coming from an underlying SCM $\mathcal{M}$ with graph $\mathcal{G}_\mathcal{M}$. We do not know what its functional assignments $\mathcal{F}$ nor latent priors $P(\mathcal{E},\mathcal{U})$ are, but we do assume to know its graph structure $\mathcal{G}_\mathcal{M}$, either through domain expertise, experimental testing or using causal discovery algorithms [16]. We also assume *positivity*, i.e., its observational distribution $P(\mathcal{V}) > 0$ throughout its domain.

Consider a query $Q(.)$, whose result depends on which SCM it is applied (e.g., $Q(\mathcal{M}) := \mathbb{E}\left[Y \mid do(X = x)\right]$). We want to find an estimator for $Q(\mathcal{M})$ by **defining a proxy SCM $\mathcal{M}'$ with equivalent graph $\mathcal{G}_{\mathcal{M}'} = \mathcal{G}_\mathcal{M}$ and observational distribution** $P_{\mathcal{M}'}(\mathcal{V}) = P_\mathcal{M}(\mathcal{V})$; we will answer the query with $\mathcal{M}'$ as if it were the underlying $\mathcal{M}$. Can we use $Q(\mathcal{M}')$ as an estimator of $Q(\mathcal{M})$ if the functions in $\mathcal{F}_{\mathcal{M}'}$ or the priors $P_{\mathcal{M}'}(\mathcal{E},\mathcal{U})$ are not the same as the ones in $\mathcal{M}$? Yes, but only if the query is *identifiable*.

Let us define the class of models $\mathbb{M}(\mathcal{M})$ consisting of all SCMs $\mathcal{M}'$ such that $\mathcal{G}_{\mathcal{M}'} = \mathcal{G}_\mathcal{M}$ and $P_{\mathcal{M}'}(\mathcal{V}) = P_\mathcal{M}(\mathcal{V})$. We say that a query Q is **identifiable** in $\mathbb{M}(\mathcal{M})$ if $\forall \mathcal{M}' \in \mathbb{M}(\mathcal{M}),\ Q(\mathcal{M}') = Q(\mathcal{M})$. Identifiability can be **proven by finding an estimand** for Q [1]: if it exists, then both $\mathcal{M}$ and $\mathcal{M}'$ would output the same result for that estimand, since they share the same causal structure (hence the estimand applies for both models) and observational distribution $P(\mathcal{V})$ (hence each term in the formula returns the same results). Additionally, as discussed at the end of section 1.1.4, **queries can be proven to be identifiable by automatic algorithms**, which either provide an estimand, proving identifiability, or prove that it cannot exist, proving non-identifiability. Note that we only need to prove the existence of an estimand; if it exists, the query is identifiable, we can discard the estimand and estimate the query directly with our proxy SCM.

In conclusion, given a proxy SCM $\mathcal{M}'$ with the same causal structure as $\mathcal{G}_\mathcal{M}$ and the same distribution $P_\mathcal{M}(\mathcal{V})$[7], if a certain query is *identifiable*, we can estimate it with $\mathcal{M}'$ as if we were using the original underlying $\mathcal{M}$. Note that our model need not have the same functions $\mathcal{F}$ nor latent priors $P(\mathcal{E}), P(\mathcal{U})$, only that $P_{\mathcal{M}'}(\mathcal{V}) = P_\mathcal{M}(\mathcal{V})$. More importantly, we do not need the estimand that proved the query's identifiability, as the SCM itself can estimate the query using its own estimation procedures (see section 4.1).

As a final note, Xia *et al.* [17] propose an alternative technique for identifiability. Given that flexible-enough Deep-Learning-powered SCM models can encompass

$C + \log(e^{-C} \cdot \sum_{i=1..N} \exp x^{(i)}) = C + \log \sum_{i=1..N} \exp(x^{(i)} - C)$. This ensures that the largest exponentiated value is 1, stabilizing the computation of the LSE. We use this technique throughout this work.

[7]Note that we can never expect to achieve a perfect match between our SCM's $P_{\mathcal{M}'}(\mathcal{V})$ and the real underlying distribution $P_\mathcal{M}(\mathcal{V})$. This miscalibration has an effect on the eventual estimations we perform with the model; an analysis on this topic is left for future work.

many different SCMs all with the same observational distribution $P(\mathcal{V})$ but possibly different interventional distributions $P_{\mathbf{x}}(\mathcal{V})$, if a certain query were not identifiable, by initializing the same SCM model with different parameters and running its training procedure, each of these models would output different results for that query. As a result, tests can be defined to determine if the differences in their estimations are significant or not, proving non-identifiability in the latter case. We do not cover this technique, focusing instead on non-parametric identification of queries for their subsequent estimation using our trained SCMs, but this is a promising research topic for complex queries for which we cannot derive an estimand nor prove its non-existence.

1.2.4 Related Work

SCM-based approaches have been studied previously, beginning with Wright's [18] first SCM, which consisted of linear equations. However, their modelling capabilities have typically been limited by simplistic architectures or restrictive distributions (e.g., Bernoulli, Normal). Since a proxy SCM needs to model the observational distribution of the underlying CBN in order to correctly estimate causal queries, this was not a feasible approach for real-world data until very recently, when the latest advances in density estimation provided by Deep Learning strategies were adapted to the field of Causal Query Estimation. Some of these works employ Generative Adversarial Networks (GANs), with CausalGAN [19] as an example, or Variational Autoencoders (VAEs), such as CEVAE [20] or VACA [21]. Adopting a different perspective focused on modelling each node's distribution through a network function, we can mention two parallel works that share some aspects with our techniques. Pawlowski *et al.* [22] proposed Deep SCMs (DSCMs), powered with Normalizing Flows for continuous multidimensional variables, applied to image generation. Xia *et al.* [17] followed a similar approach and proposed Neural Causal Models (NCMs), but their work is concerned with theoretical aspects of proxy-SCM estimation and is currently limited to discrete datasets in the examples they provide.

Although these works suggest several avenues for tackling the problem of proxy SCM estimation, we find that they lack several desirable aspects for such a system. Let us define the following **desiderata for a practitioner-ready, Deep Learning powered estimand-agnostic framework**:

1. **Explicit Likelihood**: being able to estimate likelihood queries (e.g., $P(Y \mid Z)$), with or without interventions or conditioning terms. In addition to estimating likelihoods of values, this also helps with conditional sampling, as we will see in section 4.1.

2. **Latent Confounders**: accounting for the existence of latent confounders, without restricting[8] the kinds of identifiable queries that can be estimated.

[8]Compiling two confounded variables into a single multidimensional variable lets us model them as part of a graph, but prevents us from intervening on only one of them.

TABLE 1.1: Desiderata for an estimand-agnostic CE framework.

METHOD	EXPLICIT LK.	LATENT CF.	COUNTERF.	EXPRESSIVENESS	SCALABILITY	GENERALITY
CAUSALGAN [19]	✗	✗	✗	✓	✗	✗
CEVAE [20]	✗	✓	✗	✗	✗	✗
VACA [21]	✗	✗	✓	✓	✓	✓
DSCM [22]	✓	✗	✓	✓	✗	✓
NCM [17]	✗	✓	✗	✗	✗	✓
DCN [23]	✓	✗	✗	✗	✗	✓
DCG	✓	✓	✓	✓	✓	✓

3. **Counterfactuals**: allowing counterfactual estimation, not only purely interventional queries. This means enabling abduction using factual variables to propagate the abducted noise signals to the counterfactual graph.

4. **Expressiveness**: providing expressive implementations capable of modelling complex real-world data. Finding the appropriate probability distribution for each and every node can be time-consuming (not scalable) or even unfeasible (real-world data need not fit within any of these families), so a method adopting this approach would not fulfill the requirement.

5. **Scalability**: instead of defining a different network for each node, defining a single network for all variables at the same time. This limits the number of trainable parameters in the model for graphs with a large number of variables, therefore mitigating overfitting.

6. **Generality**: the method can follow the structure of arbitrary causal graphs, and its training and estimation procedures should be immediately applicable. Most methods derive an expression for each graph and query they want to estimate, instead of defining general procedures that can adapt to each situation.

With this in mind, we proposed **Deep Causal Graphs** (DCGs), a general, modular, estimand-agnostic framework, which fulfills all of the above points. In order to present our contributions and explain the differences with previous and parallel works, we provide Table 1.1, which lists the items in the desiderata covered by each method. Together with our proposal (DCG) we include Distributional Causal Nodes (DCN) [23], our first contribution to the problem, in which we introduced the most basic implementation of our approach, later discussed in chapter 2.

It is worth noting that only DSCMs provide a method for computing the explicit **likelihood** of a sample (given an invertible-explicit implementation for its nodes). **Latent Confounders** are only covered in CEVAE (the latent space) and NCMs, whereas VACA proposes collapsing variables affected by the same latent confounder as a multidimensional, heterogenous node, which limits the kinds of queries the model can answer. **Counterfactuals** are only discussed in VACA and DSCM; the NCM paper talks about the third rung on Pearl's Ladder of Causation [24], the counterfactual level, but focuses only on purely interventional queries

$P(Y \mid do(X))$ in the examples given. Regarding **expressiveness**, we say that CEVAE and NCM are not expressive enough in the implementations that they provide, since they require the assumption of a certain probability distribution in modelling each node rather than a more flexible alternative, such as the Normalizing Flows used in our own approach or DSCMs. With respect to **scalability**, every method except for VACA requires defining a separate network for each node, which leads to overfitting on larger graphs. Finally, in relation to **generality**, we say that CausalGAN and CEVAE are not general: the former requires the definition of a *discriminator* and two *labeller* networks for the GAN node, which is not clear how to extend when modelling more than one node with GANs; the latter defines a specific architecture for the particular kind of graph the authors work with, with no indication of how that structure would change with other kinds of graphs. As for our initial proposal, DCNs, we did not cover either the latent confounder case or counterfactual estimation, focusing instead on an implementation very similar to the one later found in NCM, which limited expressiveness. We did not use our new Graphical Conditioner either (see section 4.2), which affected scalability.

As stated above, our approach, DCG, fulfills every requirement in the desiderata. Furthermore, none of the previous methods provide **an algorithmic solution for estimating general queries**, leaving the derivation to the reader, which is not trivial in some cases. We devote section 4.1 to this problem, covering observational/interventional/counterfactual sampling, likelihood and expectation queries, all with or without a conditioning term. We also provide an **open-source library** with all of our implementations ready for practitioners to use and researchers to extend.

1.2.5 Limitations

We finish this section discussing the limitations of estimand-agnostic approaches.

The identifiability results discussed in section 1.2.3 state that we can estimate identifiable queries as long as our SCMs use the same graph $\mathcal{G}$ and model the same distribution $P(\mathcal{V})$ as the underlying SCM that generated our data. However, this adjustment to the distribution is rarely exact, as our models are trained to converge towards an approximation of $P(\mathcal{V})$; this mismatch at the distribution level can result in **miscalibrations** at the estimation level.

There are two factors which can result in these miscalibrations. Proxy SCM approaches need to model the whole graph, node by node, before any estimation is carried out; this means that if a certain node has a **complex distribution** (e.g., outliers, density discontinuities) and has not been modelled properly, it can affect the eventual estimations. Additionally, if the DAG $\mathcal{G}$ has a high **depth** (the length of the longest path from roots to leaves) this could have a stronger effect when sampling from each node, as samples from an imperfect adjustment can affect all its descendants' samples.

On the other hand, the fact that our model's training objective is to estimate $P(\mathcal{V})$, and query estimations come as a result of a later procedure, means that they

cannot be optimized to answer specific queries, in contrast with estimand-based techniques that do optimize for each and every query. However, as stated before, SCMs can estimate many (identifiable) queries with the same model; in this sense, there is a trade-off between the flexibility of estimand-agnostic models and the specificity of estimand-based techniques, which can prove beneficial depending on the use case.

1.3 Black-Box Introspection

Ever since the publication of the General Data Protection Regulation (GDPR) [25] by the European Commission, concerns about algorithmic decision-making and the black-box nature of most Machine Learning models have fostered interest on **Interpretability, Explainability and Fairness** techniques. These try to elucidate the internal processes by which algorithms perform decisions, identify the effect that each input variable has on a particular outcome, or discern whether the resulting decision is fair *w.r.t.* some *sensitive* variables, respectively. There is no consensus on the different concepts and categorizations concerning these topics, so we will describe our own interpretation as a starting point.

Given a certain model $Y = f(\mathbf{X})$ to inspect, we can approach it from two perspectives: assuming that f is a **black-box** (i.e., we know nothing about the internal processes with which an output y is computed and we can only examine the model through its input-output routine $\mathbf{x} \xrightarrow{f} y$) or that it is a **white-box** (i.e., we know the exact architecture of the model —e.g., the hidden layers in a Neural Network— and can examine and tweak its parameters as much as needed). We work on the former, in order to provide techniques applicable to any kind of model f, irrespective of its internal architecture.

Orthogonal to the black- and white-box axis, we can explore three different questions. **Interpretability** deals with inspecting the general rules by which the system operates at the population level, discussing its input-output routine for any input. **Explainability**, on the other hand, works at the individual level, explaining why a certain input has been given an output, and what could change in the input variables to affect that outcome. Finally, **Fairness** considers the effect that certain *protected variables* (e.g., gender, ethnicity, age) have on the outcome, as it would be deemed unfair to favour one demographic group over the other basing the decision on these factors. Related to the previous two points, Fairness can be discussed at the population level (Interpretability) or at the individual level (Explainability), but it can also be considered as an objective to attain, training a new model towards a fairer distribution. We refer to these three topics as the more general **Black-Box Introspection**, since their subsequent application on a black-box of interest allows us to progressively examine the process so as to discern its inner workings.

These topics have been studied in myriad ways, such as gradient-based methods [26] (estimating the effects of individual input dimensions by measuring their gradients *w.r.t.* the point of study) or local surrogate models such as LIME [27] (training simpler *interpretable* models of the original predictor in the region of the given input so as to explain the function in that local region), both estimating feature importance by considering **perturbations** on the input variables, which might not always be realistic.

Instead, we focus on **Counterfactual Reasoning**, understood as contrastive explanations between the factual, observed outcome, and an hypothetical (not always causal) *counterfactual* outcome subject to input variable changes. Numerous works discuss the topic of contrastive/counterfactual explanations. Wachter *et al*. [28] define counterfactuals as data points similar to the object of study with certain strategic interventions in its input dimensions in order to change the predictor's outcome; these changes do not take into account the underlying causal structure, so the resulting input perturbations might not agree with the effects of such an intervention. Mothilal *et al*. [29] employ a similar approach, but also considering the *feasibility* of the counterfactual action (whether the individual can effectively impose the intervention) and the diversity of the resulting counterfactual examples. Hendricks *et al*. [30] describe counterfactuals as samples including some information that was missing in the original input, resulting in a prediction change; their technique identifies these traits and expresses them through natural language explanations. Goyal *et al*. [31] operate on visual explainability, identifying *distractor* images and regions within these images so that, when replacing the original image with these regions, the model results in different predictions, thereby signaling the visual features responsible for the prediction. Finally, Guidotti *et al*. [32] adopt the local surrogate approach, by training a local interpretable model in the original point's region, and then generate explanations as decision rules describing the factual decision, along with counterfactuals showcasing which changes would result in a different outcome.

Note that all of the previous examples operate from a purely **observational perspective**, modifying the variables of study by looking for similar data points, disregarding any causal effects that these alterations may have on some other input variables, which eventually affects the outcome. Alternatively, we can consider the implications of these changes through a **causal lens**: comparing two outcomes subject to a change in input (intervention) and the downstream effects of this change. This is the approach we adopt to define our black-box introspection tools.

Finally, note that Interpretability, Explainability or Fairness are not meant to answer one single query on the distribution of interest. For example, in a study about gender discrimination on salary assignment, we are not only interested in the effect of gender of salary (which can be answered at the population level or at the individual level), but also how gender affects the other input factors in the salary black-box decision system, and how each of them affects salary in turn. Effectively, we will need to answer many different questions as we explore the system, something akin to a **dialogue between user and system** through these introspection tools, which results in many different queries to answer on the same distribution. As a result, one can see why estimand-based CE approaches are not suited for this application, as every new query would require graph- and query-specific estimands and models. In contrast, estimand-agnostic techniques result in a single model capable of answering any of the required queries. Additionally, our proposed framework Deep Causal Graphs provides general procedures, applicable to any kind of graph, to answer many of these queries; hence, it is specially suited to this use case.

1.4 Contributions and Thesis Outline

We finish this chapter listing the contributions of this dissertation.

- Álvaro Parafita and Jordi Vitrià. Explaining visual models by causal attribution. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 4167–4175. Seoul, Korea, 2019. IEEE.

- Álvaro Parafita and Jordi Vitrià. Deep causal graphs for causal inference, black-box explainability and fairness. *Artificial Intelligence Research and Development*, 339:415–424, 2021.

- Álvaro Parafita and Jordi Vitrià. Estimand-agnostic causal query estimation with Deep Causal Graphs. *IEEE Access*, 10:71370–71386, 2022.

- Álvaro Parafita and Jordi Vitrià. A unified framework for causal analysis, explainability and fairness. Submitted.

- Martí Pedemonte, Jordi Vitrià, and Álvaro Parafita. Algorithmic causal effect identification with causaleffect. *arXiv preprint* arXiv:2107.04632, 2021.

The initial goal in our research was to bridge the gap between Machine Learning Introspection and Causal Query Estimation. Given an image classifier, we provided an explainability tool based on causality with the introduction of Distributional Causal Graphs [23]. We discuss this first approach in chapter 2 as an illustrative example of the general approach.

We then expanded the original proposal to a general framework called Deep Causal Graphs [33], including an implementation based on Normalizing Flows to improve its modelling performance, and finished with an Explainability and Fairness study using these techniques.

We completed the previous contribution with even more implementations of the framework to increase its applicability to different problems, and defined a set of general estimation procedures applicable to any graph and many kinds of causal queries; this work constitutes the most comprehensive and complete exposition of DCGs [34]. We cover these topics in chapters 3 and 4, respectively. Along with this work, we also developed an open-source software library[9] that implements this general framework, ready for practitioners to use and researchers to extend. We discuss these libraries in chapter 5, cover the performance of DCGs in CE benchmarks in chapter 6 and their application to complex causal queries in chapter 7.

Finally, we explored the application of DCGs to the field of Black-Box Introspection with a complete study [35] that reflects the necessity of estimand-agnostic techniques for explainability tools; we show this study also in chapter 7. Additionally, we also collaborated in the implementation of identifiability algorithms for purely-interventional and conditioned interventional causal effect queries [7].

[9]See `https://github.com/aparafita/dcg` for the `dcg` library and the code for the experiments discussed throughout this work.

Part II

The Deep Causal Graph Framework

Chapter 2

Distributional Causal Graphs

This chapter exemplifies our DCG framework in its simplest form, before we explore it in depth in chapters 3 and 4. This is our first contribution [23] to the topic, focused on determining feature importance in an image classifier. We will describe the goals of this approach and the proposed solution in the following sections.

2.1 Motivation

Consider a classifier $y := f(X)$, X being a three-dimensional matrix of pixel values (RGB colours) and y being the logits of a binary classification (i.e., with the threshold at 0, $f(X) \geq 0$ predicts class 1, while $f(X) < 0$ predicts class 0). Our objective is to **explain which factors motivated the classifier to decide for a certain class**. Most methods in the literature explore this problem from a pixel attribution (saliency maps) perspective (e.g., [36, 37, 38, 39]), meaning, by creating a bi-dimensional matrix with values indicating the importance of the corresponding pixel in the eventual prediction. We take issue with this approach because the pixels themselves only serve to locate the position of the salient features, and it is the researcher, with their preexisting knowledge, who can interpret these results in order to explain a certain prediction. Being that the case, it is difficult to determine if the explanation is derived purely from the available data or from subjective interpretations after the fact.

With this in mind, we focus on an alternative approach. Given certain *latent factors*[1] that describe the image in question, if we know how these factors affect one another causally, can we **explain the classifier's prediction based on these feature effects**? For example, given a dog/cat predictor, we could define latent factors such as pose, iris shape, time of day, background, etc. Note that some features can affect others (time affects which background the picture was taken in, and both affect the animal's action and pose), so we can describe these relationships with an SCM. All of these latent factors affect what is shown in the final picture, which constitutes a multivariate node X (an image/matrix of dimension $width \times height \times 3$) with every latent factor pointing at it. Finally, we also represent the classifier as another node Y, with only the picture X as a parent and with the classifier itself f as its generative

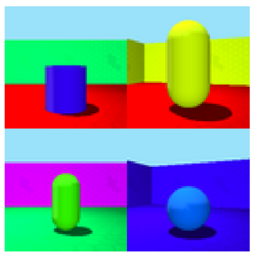
[1] In this chapter we preserve the original nomenclature and name the features that describe the picture as *latent factors*, not in the sense used in chapter 1, but as underlying features of the image.

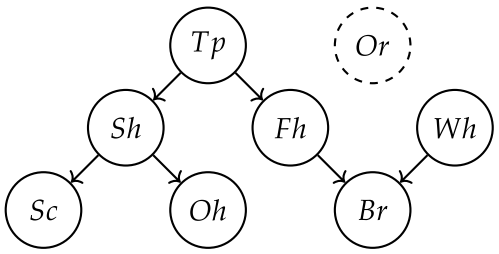
function f_Y. If we aggregate every latent factor as a set of variables $\mathbf{Z}$, the resulting graph is $\mathbf{Z} \to X \to Y$.

If we want to **study the effect of each factor $Z \in \mathbf{Z}$ on the classifier**, we can intervene on Z with different values, resulting in some changes to its descendants, which configure a new picture that will be later passed through the classifier to obtain the new predictions y; we average these outcomes to estimate the effect of each intervened value on the classifier.

2.2 3D Shapes Dataset

The main difficulty of this approach is how to sample new images, which can be achieved with an image generation model; we discuss this in more detail in section 2.5. In this chapter, however, we operate with a **synthetic dataset** that allows us to bypass this step, so as to focus on the latent factor approach. We use the 3D Shapes Dataset [40]; see Fig. 2.1a for some sample images. These pictures consist of an object with different possible shapes, colours and sizes, sitting in a room with a certain floor and wall hue and different camera orientations. There is an image for every possible configuration of these six factors in the dataset. We also create two additional factors: *brightness*, the average brightness of the picture, that we artificially impose when sampling; and *type*, a binary value not visible in the image that will be the target variable for the classifier. We create an **artificial causal graph** relating these eight factors, so that the resulting factor configurations determine which pictures are sampled more frequently; given that there is a picture for every configuration in the dataset, we can "generate" an image by taking the corresponding picture from the dataset, thereby bypassing the necessity of an image generator. See the proposed graph in Fig. 2.1b, with the following nodes: Type (Tp), Orientation (Or), Shape (Sh), Floor Hue (Fh), Wall Hue (Wh), Scale (Sc), Object Hue (Oh) and Brightness (Br). Note that *type* affects almost every factor, so even if it is not visible in the final picture, its nature can be discerned from the remaining visual factors.


(A) Sample images.



(B) 3D-Shapes Example Graph. *Or* is considered a latent node, not measured.

FIGURE 2.1: 3D Shapes example. The left image shows four sample pictures from the dataset. The right image shows the artificial causal graph describing the relationships between its latent factors.

We add three more complications to this synthetic problem. We filter out any samples that result in a Brightness (Br) level outside of $[0.4, 0.6]$; as a result, the dataset contains selection bias, in the sense that by conditioning on Br, we are opening a path between Fh and Wh that was closed before by the unconditioned collider Br. We perform this filtering during the sampling process, but we do not take it into account in our SCM, using the graph in 2.1b instead; consequently, we can use the difference between the data generating process and the estimating SCM to test the resilience of our technique to **graph mismatch**. Additionally, when specifying the functional assignments of our SCM, we will not enforce this domain restriction to the Br node, using $[0, 1]$ as its domain instead, to also test our technique against **distribution mismatch**. On the other hand, the Orientation (Or) node will not be measured (therefore it is a latent node, although disconnected to the rest of the nodes) so as to have 15 different pictures with the same factors, adding some **stochasticity** to the image generation part of the graph.

2.3 Node Modelling

Consider a given DAG $\mathcal{G}$, consisting of variables/nodes $\mathcal{V} := \{V_1, \dots, V_K\}$ with a certain distribution $P(\mathcal{V})$ from which samples $\mathcal{D} := (\mathbf{v}^{(i)})_{i=1..N}$ are generated. We define an SCM $\mathcal{M}_\Theta$ with the same variables $\mathcal{V}$ and graph structure $\mathcal{G}$ depending on some parameters $\Theta = (\Theta_1, \cdots, \Theta_K)$ (one set per node) that need to be trained in order to model the resulting distribution $P_\Theta(\mathcal{V})$. In this method, we assume there are no latent confounders ($\mathcal{U} = \varnothing$).

Given any node $X \in \mathcal{V}$ with Markov parents Pa_X, we want to model the node's distribution conditioned on its parents, $P(X \mid Pa_X)$. **Distributional Causal Nodes (DCNs) assume that $P(X \mid Pa_X)$ belongs to a certain probability distribution family** (e.g., Normal, Exponential, Bernoulli, Categorical), each with some parameters Θ_X (e.g., $\Theta_X = (\mu_X, \sigma_X)$). Note that since the distribution depends on the values of Pa_X, we can define Θ_X as a function of its parents: $\theta_X = \Theta_X(pa_X)$. This is the process by which we model every variable X: looking at its shape, we determine which distribution fits the data and then define a feed-forward Neural Network f_X that takes as input pa_X and returns a value for the distribution's parameters θ_X.

Now, given the parameters θ_X for a node X, we want to be able to sample from the distribution $P(X \mid \theta_X)$. Depending on the chosen node's distribution, we need to define an appropriate sampling process; this method suggests using a reparametrization formula [41] as the sampling operation. For example, consider graph $\mathcal{G} := (Z \to X \to Y)$, with $P(X \mid Z)$ fitting a univariate Normal Distribution $\mathcal{N}(\mu, \sigma)$. We can specify E_X's prior (the exogenous noise signal distribution, from which we add stochasticity to the node's sampling process) to be a standard univariate Normal Distribution $\mathcal{N}(0, 1)$, and set the sampling operation as $x \leftarrow \sigma \cdot \varepsilon_X + \mu$, with the parameters coming from the parameters network: $(\mu, \sigma) = \theta_X = \Theta_X(pa_X)$. We can replicate this approach for many other distributions (e.g., Beta, Poisson); we will provide more details about DCNs and alternative parametrizations in section 3.2.

Finally, we need to compute $\log P(X \mid \theta_X = \Theta_X(pa_X))$ in order to train the parameter's network: since we know the node's distribution family, we can simply use the corresponding Probability Mass Function or Probability Density Function. As a result, we can compute the joint **log-likelihood** of a full sample $\mathbf{v} \sim P(\mathcal{V})$ as:

$$\log P(\mathbf{v}) = \log \prod_{k=1}^{K} P(v_k \mid pa_k) = \sum_{k=1}^{K} \log P(v_k \mid \theta_k = \Theta_k(pa_k)). \tag{2.1}$$

Therefore, we can train the overall SCM consisting of these DCN modules with **Maximum Likelihood Estimation**, as detailed in section 1.2.3.

2.4 Visual Explainability

We now describe how to measure latent factor importance with our SCM model. DCNs as they were at this point had no general mechanism to perform *abduction*, an essential step for counterfactual estimation. For this reason, this work [23] focused on an alternative query to explain the classifier's predictions.

Given a particular image X described by factors $\mathbf{Z}$, of type $Tp = 0$ but classified incorrectly ($f(X) > 0$), we want to **identify the factor responsible for the wrong prediction**. For any factor $V_k \in \mathcal{V}$, we consider interventions v_k for every possible value of the variable (every factor has a finite set of values in this dataset, despite some of them representing a continuous variable) and plot the average logits and their confidence intervals when we generate the intervened images and pass them through the classifier. However, in order to estimate this query using the information observed in $\mathbf{Z}$, we would need a counterfactual query that takes into account the factual information; what we do instead is intervene every non-descendant of the intervened variable $V_{k'} \in \mathcal{V} \setminus De(V_k)$ with the observed value in $\mathbf{Z}$ (the same as what happens in a counterfactual, where any non-descendant would not be affected by the intervention, thus resulting in the same value as the factual configuration), and every descendant of V_k is sampled anew taking into account its new parents. This ignores the information contained in the factual values of every descendant of V_k, but it is unavoidable given that no abduction mechanism was provided at this point; please refer to the following chapters where we detail the abduction operation and provide proper counterfactual estimation techniques.

See Fig. 2.2 for a visualization of these effects. This particular image was predicted as class 1 despite being class 0, so we are looking for whichever variable reduces the classifier's logit to negative values when intervened. The only variable capable of this is Floor Hue, that was 0.3 originally and set the classifier right at the decision boundary, but changes the prediction for values bigger than 0.4. In other words, this kind of query/visualization allows us to attribute decisions to individual factors through **contrastive explanations** and therefore interpret a black-box.

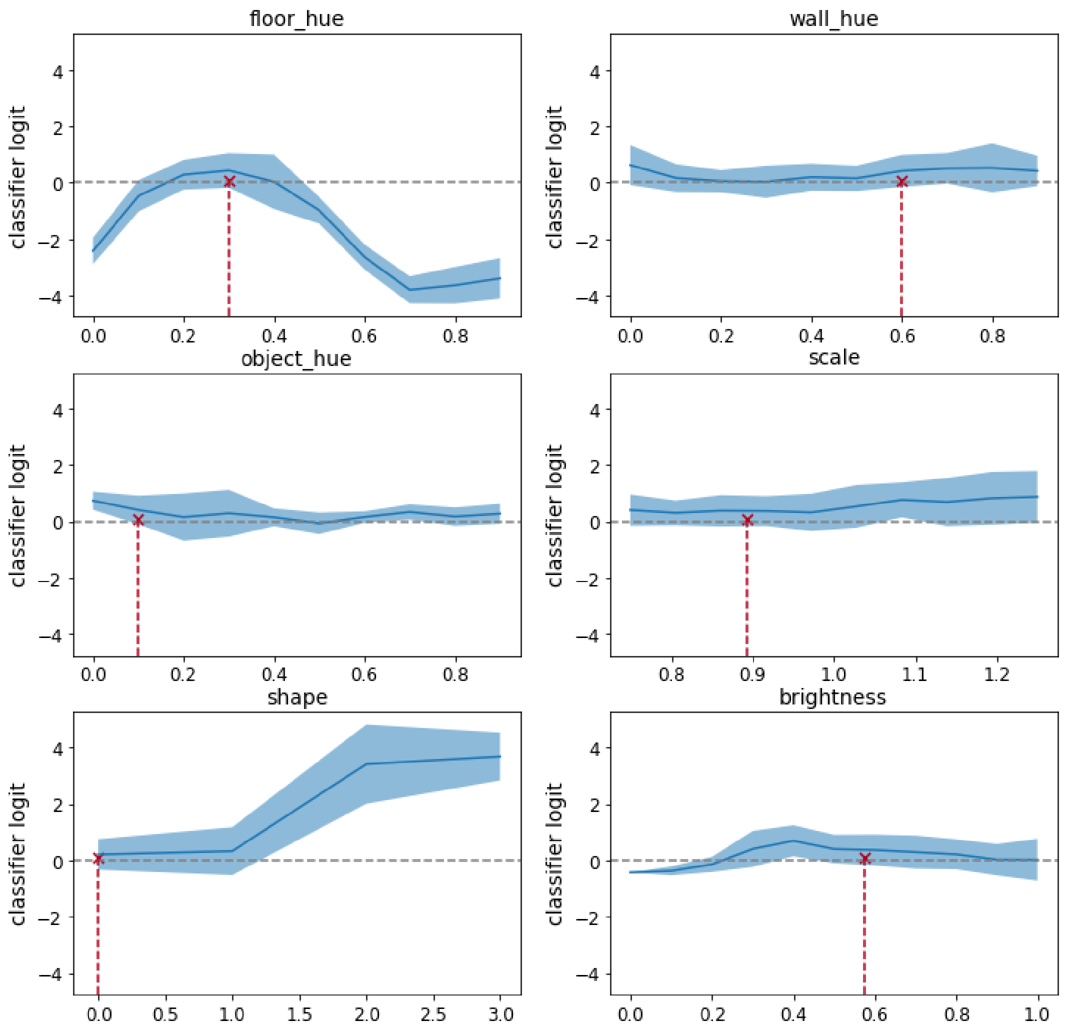


FIGURE 2.2: 3D Shapes Intervention Effects. Every plot describes the effect of intervening a particular variable with every possible value, showing the expected logit with a blue line and the confidence interval of this expectation with a blue area. Finally, the red cross shows the *factual* value for the variable. The dashed horizontal line represents the decision boundary at $y = 0$; positive values result in class 1, while negative values are class 0.

2.5 Causal Image Generator

An important part of this technique is the model responsible for the **generation of *counterfactual* images** according to a given latent factor configuration. One possible approach for such a model is a Conditional Generative Adversarial Network (Conditional-GAN), that creates *realistic* images (fitting the data distribution) based on some conditioning values (the latent factors $\mathbf{Z}$). However, there are some aspects of the image X that cannot be encoded only through its latent factor description, defined by the exogenous signal E_X, so they must be taken into account when computing counterfactuals of that particular image (in other words, some sort of abduction mechanism). This has been attempted by adding the original, factual image as an additional input of the model, along with some reconstruction loss during training that ensures that the factual and counterfactual images are *similar enough*. Fader Networks [42] and AttGAN [43] are two examples of this approach. However, this setup results in a number of difficulties.

First and foremost, a Causal Image Generator needs to be able to **generate realistic images given *any* configuration of the latent factors**, even if it is not likely in the observational distribution. As an example, consider a cisnormative gender classifier working on portraits, with latent factors gender $\rightarrow$ moustache. Observationally, "woman with moustache" would be found very unlikely, but it could be a plausible configuration given an intervention on the latent factor *moustache*; however, if we train the model to generate images only fitting the observational, non-intervened distribution, we can find multiple artifacts "adding masculinity" to the resulting image just to make it feasible in the original distribution. This is specially important in an adversarial training context, as it requires a secondary network, the Discriminator, that distinguishes between realistic and unrealistic images; such a Discriminator would discard "woman with moustache" due to it being unrealistic in the context of the training dataset, thereby encouraging the Generator to avoid this kind of image.

On the other hand, if we use some kind of **reconstruction loss** to ensure that the generator respects the factual image information when computing its counterfactual, we need to define this loss **according to the causal graph** of the factors. Since we are asking for a different factor configuration subject to an intervention, the counterfactual image needs to differ from the factual one in the intervened factor and its descendants, but nowhere else. Despite this, common reconstruction losses usually work at the pixel level, such as an L_2 difference between both images, which disregards this aspect. This could be avoided with a Generator that actually allows for abduction, sampling from the exogenous noise signal distribution conditioned on the factual values, but this had not been studied yet at the time of this experiment.

2.6 Conclusion

The field of Causal Query Estimation was —and still is— dominated by estimand-based approaches, while the proxy-SCM approach had not been sufficiently explored at the time of this work. DCNs provided a promising direction for further research, given its many **flexible implementations** (adjusting for distributions other than Bernoulli or Categorical variables) and the fact that proxy SCMs allow for **interventions on continuous variables** seamlessly (while the rest of the literature is commonly restricted to binary treatments); not only that, but its **estimand-agnostic** nature extended their applicability to many other graphs and queries not supported by most estimand-based techniques.

On the other hand, we also showcased a potential **application** of this framework to **Visual Explainability**, given an appropriate Causal Image Generator; we identified a number of challenges for these generative models that are still open for future work. However, given the potential of DCNs as an alternative to the estimand-based approach, irrespective of its application to Visual Explainability, we opted to focus on this topic instead, as there were many different problems to solve before DCNs could become a complete, general and flexible framework for Causal Estimation. We will cover these challenges in the following chapters.

Chapter 3

Deep Causal Unit

In this chapter we describe the main building block in our framework, Deep Causal Units (DCUs), with which we model the distribution of a node while also providing every required functionality for the eventual estimation of causal queries. We begin with the definition of the DCU to then explore four different implementations of this abstract specification, each with its own use cases.

3.1 DCU Specification

A DCG is an SCM where every variable/node is modelled by a submodule, called the **Deep Causal Unit** (DCU). For any $X \in \mathcal{V}$, each DCU can be understood as a subnetwork with trainable parameters Θ_X that models the distribution of its own node conditioned on its Markov parents, $P(X \mid Pa'_X)$, while providing functionality for three distinct operations:

1. **Sample**: sampling from $P(X \mid pa'_X)$ by taking a value $\epsilon_X \sim P(E_X)$ and passing it through the function f_X modelled by the node.
2. **Loglk**: computing the log-likelihood $\log P(x \mid pa'_X)$ corresponding to the random variable X that results from using f_X as the sampling operation. This operation must be **differentiable** *w.r.t.* the distribution's parameters Θ_X, as it will employed to compute the training loss for the overall DCG.
3. **Abduct**: sampling from the posterior $P(E_X \mid x, pa'_X)$. This is required for counterfactual estimation.

This definition of the DCU requires that every node in the graph defines its own subnetwork, the **Conditioner**, which would not scale when the number of variables is too high. Alternatively, we can employ the Graphical Conditioner, discussed in section 4.2, which encompasses every node's network into a single network, thereby bypassing the problem. For clarity of explanation, we will proceed as if we defined a specific network for each node; when we explain the **Graphical Conditioner**, we will see how this is simply an abstraction for a single all-encompassing network.

This abstract definition allows DCG graphs to include different implementations of DCU nodes for every variable and still use the same training and estimation procedures. By accessing these three operations, the actual structure inside the node is irrelevant for the graph. In the following sections, we will cover four possible implementations of this specification; as long as they can execute these three operations, they are DCUs and can be integrated within the overall DCG framework.

3.2 Distributional Causal Node

The most basic implementation for DCUs is the **Distributional Causal Node** (DCN), previously described in section 2.3. We assume that a random variable $P(X \mid Pa'_X)$ behaves like a certain parametric distribution family (e.g., the Exponential distribution) with parameters Θ_X. These parameters come as a function of its parents Pa'_X ($\theta_X := \Theta_X(pa'_X)$), which we model with a feed-forward network and pertinent activation functions for each parameter depending on its domain (e.g., a softplus function for $\sigma > 0$, or a softmax for $(p_k)_{k=1..K}$ so that $\sum_{k=1}^{K} p_k = 1$); this network is the DCN's **Conditioner**.

Distributional Causal Nodes are valid DCUs given the following implementations for the three DCU operations:

- **Sample**. We define a certain prior for $P(E_X)$ independent to Θ_X and a deterministic sampling function f_X that transforms samples $\varepsilon_X \sim E_X$ into samples $x \sim P(X \mid pa'_X)$ with $x := f_X(\varepsilon_X, \theta_X = \Theta_X(pa'_X))$. This can be done using the reparametrization trick [41], as explained in chapter 2, or using **inverse transform sampling** if possible: $\varepsilon_X \sim \mathcal{U}(0,1), x := PPF_{\theta_X}(\varepsilon_X)$, with PPF_{θ_X} the Percentile Point Function of $P_{\theta_X}(X) = P(X \mid \theta_X = \Theta_X(pa'_X)) = P(X \mid pa'_X)$. Note that the sampling operation need not be differentiable *w.r.t.* Θ_X.

- **Loglk**. Since we know the distribution family, we can use the log-likelihood formula corresponding to that family.

- **Abduct**. We need to sample from $P(E_X \mid x, pa'_X)$. If the sampling operation is invertible, this distribution is constant and we just invert the formula; alternatively, we need a different strategy, discussed in the following paragraphs.

Table 3.1 contains the DCN implementation of several distribution families. As an example of a more involved abduction process, consider the Categorical distribution (of which the Bernoulli distribution is a special case) with K levels. For sampling, we need K i. i. d. Gumbel values to use the Gumbel-argmax trick [44, 45], which allows samples to be generated differentiably *w.r.t.* parameters p; this, however, results in a non-injective sample function. Nonetheless, abduction is still possible: given the observed category k' ($x = k'$) and pa'_X, from which we compute $\theta_X = \Theta_X(pa'_X) = (p_k)_k$, we sample $g_{k'} \sim \mathcal{G}(0,1)$ and based on this value, we sample the remaining $K-1$ values from $\mathcal{G}(\log p_k, 1)$ truncated by the previous $g_{k'}$. Finally, we transform these g values back to a $\mathcal{G}(0,1)$ so as to decouple them from the parameters $\log p$. See [44] and [46] for more details.

TABLE 3.1: DCN specification for several continuous and discrete distributions: Normal $\mathcal{N}$, Exponential Exp, Asymmetric Laplace ALD, Beta $\mathcal{B}$, Truncated Distribution $X(\theta_X)$ to interval (a,b), Categorical Cat and $Poisson$. These distributions are already implemented in our library; the purpose of this table is to illustrate how they are defined as DCNs.

$Distr(\Theta_X)$	E_X PRIOR	SAMPLE: $x \sim P(X \mid \Theta_X(pa'_X))$	LOGLK: $\log p(x \mid \Theta_X(pa'_X))$	ABDUCT: $\varepsilon \sim P(E_X \mid x, \Theta_X(pa'_X))$
$\mathcal{N}(\mu,\sigma)$	$\mathcal{N}(0,1)$	$x \leftarrow \sigma \cdot \varepsilon + \mu$	$-\frac{1}{2}(\log 2\pi\sigma^2 + \frac{(x-\mu)^2}{\sigma^2})$	$\varepsilon \leftarrow \frac{x-\mu}{\sigma}$
$Exp(\lambda)$	$\mathcal{U}(0,1)$	$x \leftarrow -\frac{\log \varepsilon}{\lambda}$	$\log \lambda - \lambda x$	$\varepsilon \leftarrow exp\{-\lambda x\}$
$ALD(\mu,\lambda,\kappa)$	$\mathcal{U}(0,1)$	$\varepsilon \leftarrow \varepsilon(\kappa + \kappa^{-1}) - \kappa$ $s \leftarrow sign(\varepsilon)$ $x \leftarrow \mu - \frac{1}{\lambda s \kappa^s} \log(1 - \varepsilon s \kappa^s)$	$s \leftarrow sign(x-\mu);$ $\log \frac{\lambda}{\kappa+\kappa^{-1}} - (x-\mu)\lambda s \kappa^s$	$s \leftarrow sign(x-\mu)$ $\varepsilon \leftarrow (1 - e^{-(x-\mu)\lambda s \kappa^s}) s \kappa^{-s}$ $\varepsilon \leftarrow \frac{\varepsilon+\kappa}{\kappa+\kappa^{-1}}$
$\mathcal{B}(\alpha,\beta)$	$\mathcal{U}(0,1)$	$x \leftarrow PPF(\varepsilon,\alpha,\beta)$	$\log PDF(x,\alpha,\beta)$	$\varepsilon \leftarrow CDF(x,\alpha,\beta)$
$X(\theta_X \mid X \in (a,b))$ $\varepsilon_a := CDF(a,\theta_X)$ $\varepsilon_b := CDF(b,\theta_X)$	$\mathcal{U}(0,1)$	$\varepsilon \leftarrow \varepsilon(\varepsilon_b - \varepsilon_a) + \varepsilon_a$ $x \leftarrow PPF(\varepsilon,\theta_X)$	$\log \frac{PDF(x,\theta_X)}{\varepsilon_b - \varepsilon_a}$	$\varepsilon \leftarrow CDF(x,\theta_X)$ $\varepsilon \leftarrow \frac{\varepsilon-\varepsilon_a}{\varepsilon_b-\varepsilon_a}$
$Cat(p_1,\cdots,p_K)$	$\mathcal{G}(0,1)^K$	$x \leftarrow argmax_k(\log p_k + \epsilon_k)$	$\sum_k x_k \cdot \log p_k$	$k' \leftarrow argmax_k\, x_k$ $\varepsilon_{k'} \sim \mathcal{G}(0,1);$ $\forall k \neq k',\ \varepsilon_k \sim \mathcal{G}(\log p_k, 1)$ $\forall k \neq k',\ \varepsilon_k \leftarrow -\log(e^{-\varepsilon_k} + e^{-\varepsilon_{k'}})$ $\forall k,\ \varepsilon_k \leftarrow \varepsilon_k - \log p_k$
$Poisson(\lambda)$	$\mathcal{U}(0,1)$	$x \leftarrow min\{n \mid e^{\lambda}\varepsilon < \sum_{k=0}^{n+1} \frac{\lambda^k}{k!}\}$	$x \log \lambda - \lambda - \sum_{k=1}^{x} \log k$	$\varepsilon \sim \mathcal{U}(e^{-\lambda} \sum_{k=0}^{x} \frac{\lambda^k}{k!}, e^{-\lambda} \sum_{k=0}^{x+1} \frac{\lambda^k}{k!})$

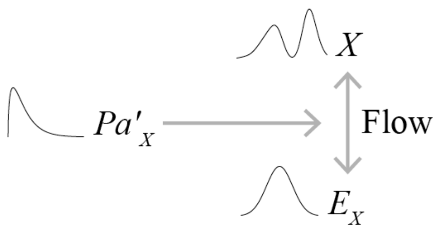


FIGURE 3.1: NCF architecture. The flow's invertible function allows us to transform between X and E_X in either direction, conditioned on the parents Pa'_X. Sampling moves from E_X to X; abduction moves from X to E_X. The log-likelihood can be computed from the flow's architecture.

In the case of the Beta distribution, we cannot find a reparametrization formula that allows for abduction. However, **inverse transform sampling** covers these two operations and allows the distribution to be applied to a DCN, as long as we can compute its log-likelihood differentiably *w.r.t.* Θ_X and we have algorithms for its Cumulative Distribution Function (CDF) and PPF, one inverse of the other. With these, we can transform from our X to $\mathcal{U}(0,1)$ and back. This general strategy is applicable to a number of other distributions, and even allows for distributions X truncated to intervals (a, b) (possibly infinite) as long as its CDF is differentiable *w.r.t.* Θ_X: we compute the CDF of its extremes $\varepsilon_a, \varepsilon_b$ and use them on all three steps of the DCU, as shown in the table.

In summary, DCNs are general, expressive DCU implementations that encompass a wide array of distributions. None of the remaining DCUs discussed in this chapter work for **discrete distributions**, so DCNs are the *de facto* DCU in these cases. However, for the continuous case, the requirement to specify a certain distribution family for every single variable does not scale to graphs with many nodes; it can also be too restrictive, as a known family might not fit real world data. To avoid these scalability and expressiveness problems, we propose an alternative DCU implementation in the following subsection. However, when dealing with simpler distributions or a small number of training samples, DCNs are still a good option.

As a final note, if we wanted to use a **linear SCM** embedded in the DCG framework, this would be possible with DCNs, by forcing every node's Conditioner network to be a simple Linear layer with appropriate activations. This means that every procedure described in section 4.1 is also applicable to the linear case. However, we must be sure that such a restrictive architecture is capable of modelling $P(\mathcal{V})$; otherwise, its estimations would not be reliable.

3.3 Normalizing Causal Flow

A different strategy for continuous distributions is the use of Conditional Normalizing Flows (see [47] for an extensive survey on the topic), density estimation methods based on defining an invertible function between two random variables X and E, given a conditioning Z, $E = f(X \mid Z)$. For our purposes, $E := E_X$ and $Z := Pa'_X$. We define **Normalizing Causal Flows** (NCF) —see Fig. 3.1— as a flow-based DCU implementation:

- **Sample**. By sampling a value $\epsilon_X \sim P(E_X)$ (with $P(E_X)$ predetermined by the flow), we compute the corresponding x sample with $x := f_X^{-1}(\varepsilon_X \mid pa'_X)$.
- **Loglk**. Given the Jacobian J_{f_X} of f_X,

$$\log P_X(x \mid pa'_X) = \log P_{E_X}(f_X(x \mid pa'_X)) + \log |\det \mathrm{J}_{f_X}(x \mid pa'_X)|. \tag{3.1}$$

  If $\mathrm{J}_{f_X}(x \mid pa'_X)$ is a triangular matrix, we only need its diagonal terms, with:

$$\log |\det \mathrm{J}_{f_X}(x \mid pa'_X)| = \log |\prod_{j=1}^{D} \frac{\partial f_{X,j}(x \mid pa'_X)}{\partial x_j}| = \sum_{j=1}^{D} \log |\frac{\partial f_{X,j}(x \mid pa'_X)}{\partial x_j}|. \tag{3.2}$$

- **Abduct**. $P(E_X \mid x, pa'_X)$ is a constant r. v. with its values computed deterministically through f_X: $\varepsilon_X = f_X(x \mid pa'_X)$.

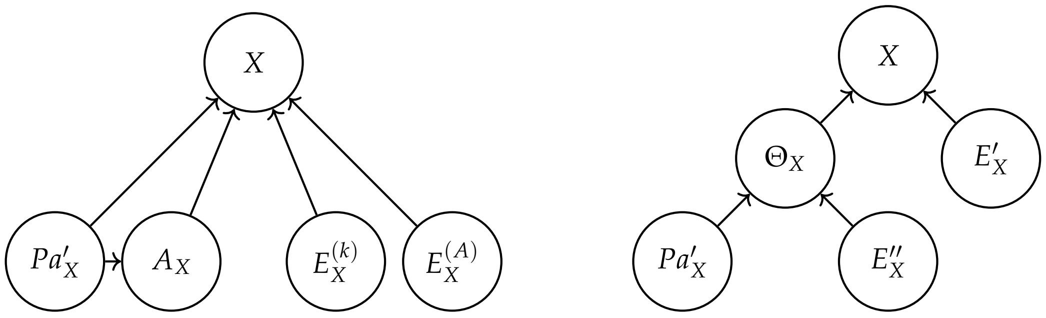
An essential aspect of Normalizing Flows is that they are *composable*, meaning, they can be defined as the composition of a finite number of subflows. Consider a flow $f_X := f_X{}^{(K)} \circ \cdots \circ f_X{}^{(1)}$; its Jacobian is the matrix product of the subflows' Jacobians, therefore $\log |\det \mathrm{J}_{f_X}(x)| = \sum_{k=1..K} \log |\det \mathrm{J}_{f_X{}^{(k)}}(x)|$. In other words, we can stack a number of flow operations to progressively transform from our data distribution $P(X)$ to its exogenous noise signal prior $P(E_X)$, usually $\mathcal{N}(0,1)$.

Regarding the flow operation f_X, it is normally defined as the conjunction of a **Transformer** (the actual function that transforms between X and E_X, which depends on some parameters Θ_X) and a **Conditioner** (a network that takes the conditioning values pa'_X as input and outputs θ_X). For example, an Affine Transformer defines two parameters per dimension $\theta_k := (\mu_k, \sigma_k)$ so that $\varepsilon_k := \sigma_k \cdot x_k + \mu_k$. Given such a Transformer operation defined dimension-wise with parameters Θ_k, an Autoregressive (Conditional) Conditioner defines an arbitrary network h_k for every dimension so that $\theta_k := h_k(x_{<k}, pa'_X)$ with $x_{<k}$ the dimensions in x before the k-th dimension. With such a structure, the Jacobian is triangular and its log-likelihood is feasible to compute no matter the number of dimensions. In particular, **for unidimensional variables**, the Conditioner just takes into account the parents' values; as a result, **we can stack multiple Transformer layers powered by a single Conditioner** that encompasses them all, providing values for the parameters of every layer.

This implementation of the DCU imposes **no restrictions on the Transformer operation**, and **for unidimensional variables** (as is the case for most nodes in a causal graph) **the Conditioner need not have any particular architecture**. As such, we can use any Transformer architecture from the literature and it will work like any other DCU in the overall DCG framework; this allows us to leverage any advances in the field for our models, which is essential in properly modelling the desired $P(\mathcal{V})$.

For multi-dimensional variables, the Conditioner might require special restrictions (e.g., an autoregressive structure, coupling layers, etc.). If that is the case, we can isolate the Conditioner for this variable as a separate network, leaving the shared Graphical Conditioner for the rest of variables in $\mathcal{V}$; as a result, we can use highly specialized networks for complex nodes, while leaving the general Conditioner, which is less prone to overfitting, for simpler nodes.

3.4 Mixture DCU



(A) Mixture DCU. (B) Compound DCU.

FIGURE 3.2: Mixture and Compound DCU subgraphs.

An interesting finding is that **mixtures of *any* DCU implementation are themselves valid DCUs**. Let us consider a node X, with parents Pa'_X, and assume we have K DCU implementations $(X^{(k)})_{k=1..K}$ for X (not necessarily homogeneous), each with its own $E_X^{(k)}$ and likelihood function $f_X^{(k)}$. Let us define an additional exogenous noise signal A_X modelled by a Categorical distribution with weights/probabilities dependent on Pa'_X: $w_X = W_X(pa'_X) = ({w_X}^{(k)})_{k=1..K}$, with $\sum_k {w_X}^{(k)} = 1$. In practice, we extend the node's parameters to include these weights: $\Theta'_X := (\Theta_X, W_X)$. Alternatively we can assume $(A_X \perp\!\!\!\perp Pa'_X)$, in which case the parameters are learnt in isolation without the Conditioner network. Fig. 3.2a shows the subgraph of a Mixture DCU. We can now define their K-mixture:

- **Sample**. Generate $a_X \in \{1, \ldots, K\} \sim P(A_X \mid pa'_X)$; this can be done with Gumbel sampling, as in the Categorical-DCN implementation. Then, we sample from the a_X-th DCU component $P(X^{(a_X)} \mid pa'_X)$ as usual.

- **Loglk**. We only need the likelihood of a mixture:

$$\begin{aligned} \log f(x \mid pa'_X) = & \log \sum_{k=1..K} {W_X}^{(k)}(pa'_X) \cdot f_X^{(k)}(x \mid pa'_X) = \\ & \log \sum_{k=1..K} \exp\left(\log {W_X}^{(k)}(pa'_X) + \log f_X^{(k)}(x \mid pa'_X)\right). \end{aligned} \tag{3.3}$$

We add exp log inside the summation in order to use the log-sum-exp trick, for numerical stability.

- **Abduct**. With a Mixture Node, our exogenous variables are A_X and $(E_X^{(k)})_{k=1..K}$, so we need to sample from $P(A_X, (E_X^{(k)})_k \mid x, pa'_X)$, but we do not know which component $X^{(k)}$ generated x. However,

$$\begin{aligned} &P(A_X, (E_X^{(k)})_k \mid x, pa'_X) = \\ &\quad P(A_X \mid x, pa'_X)\, P((E_X^{(k)})_k \mid x, pa'_X, A_X) = \\ &\quad P(A_X \mid x, pa'_X) \prod_k P(E_X^{(k)} \mid x, pa'_X, A_X) = \\ &\quad P(A_X \mid x, pa'_X)\, P(E_X^{(A_X)} \mid x, pa'_X, A_X) \prod_{k \neq A_X} P(E_X^{(k)}). \end{aligned} \tag{3.4}$$

  We split the first conditional term in two, and then split $P((E_X^{(k)})_k \mid x, pa'_X, A_X)$ into individual terms given that $\forall k \neq k',\ (E_X^{(k)} \perp\!\!\!\perp E_X^{(k')} \mid x, pa'_X, A_X)$ (since A_X cuts every $E_X^{(k)}, k \neq A_X$, from X, thereby removing any possible path between them). Finally $((E_X^{(k)})_{k \neq A_X} \perp\!\!\!\perp X, Pa'_X \mid A_X)$ by the same reasoning, and then we simplify $P(E_X^{(k)} \mid A_X)$ to $P(E_X^{(k)})$ since $((E_X^{(k)})_{k \neq A_X} \perp\!\!\!\perp A_X)$.

  Now, in order to sample from this distribution, we can sample term by term. Given a value a_X for A_X, we can independently sample from every $E_X^{(k)}$, $k \neq a_X$ and only abduct from $P(E_X^{(a_X)} \mid x, pa'_X, a_X)$. The term $P(A_X \mid x, pa'_X)$ is solved by **conditional sampling**: generate N i. i. d. samples $a_X \sim P(A_X \mid pa'_X)$ and then use weights $s(\log P(x \mid pa'_X, a_X))$ to subsample, with s the softmax operation. Refer to section 4.1.3 for an explanation of conditional sampling.

Mixture Nodes can be **used to empower more restrictive DCUs** (DCNs, in particular, benefit from this). By way of example, we can define Gaussian Mixtures with this technique using the simple Gaussian-DCN implementation. Additionally, it is possible to create mixtures from models trained with different splits in a **Cross Validation** setup; this helps in datasets with a limited number of training samples, as the validation set in one split can also be employed when training the rest. We will elaborate on this point in chapter 6.

3.5 Compound DCU

A natural extension to Mixture DCUs is the Compound DCU. A Compound distribution is a parametrical distribution X dependent on parameters Θ_X that are themselves random variables with prior $P(\Theta_X)$. As a result, X is an **uncountable mixture**, $P(X) = \mathbb{E}_{\Theta_X}[P(X \mid \Theta_X)]$, with every possible value $\theta_X \sim P(\Theta_X)$ describing a different component with a certain likelihood of being selected. Compound DCUs generalize the work in [48], which proposed a similar implementation for uncountable mixtures of Asymmetric Laplace Distributions.

We will assume the components of this mixture to be homogeneous. Each subcomponent is defined by the same set of parameters Θ_X, which in turn are computed with a network that takes its parents' values as input, $\theta_X = \Theta_X(pa'_X)$. However, to model the uncountable mixture with a single network, what we do instead is extend the corresponding exogenous signal E_X with a second source of stochasticity, $E_X := (E'_X, E''_X)$, where E'_X follows the usual role in the DCU implementation and E''_X is not used for the sampling operation, but rather employed in computing the parameters $\theta_X = \Theta_X(pa'_X, \varepsilon''_X)$. Given any arbitrary prior for E''_X (e.g., $E''_X \sim \mathcal{N}(0,1)$), using it in the Conditioner network as an additional input adds a source of **stochasticity to the parameters' computation**, resulting in an **uncountable mixture of DCUs**.

The simplest way to implement a Compound DCU for a certain node X is to create an additional latent variable E''_X; this latent only affects the corresponding X (it is not a confounder) through the Conditioner's network, adding it as an input. See Fig. 3.2b for the subgraph of a Compound DCU; Θ_X is a r. v. with a deterministic computation given values for Pa'_X and E''_X. The result is a DCU, since:

- **Sample**. We sample $\varepsilon''_X \sim P(E''_X)$, compute the parameters $\theta_X = \Theta_X(pa'_X, \varepsilon''_X)$ and then apply the DCU's sample operation for X as usual.

- **Loglk**. We compute log-likelihoods with X's loglk operation as usual, but marginalizing over E''_X:

$$\begin{aligned}\log f(x \mid pa'_X) = &\log \mathbb{E}_{E''_X \mid pa'_X}\left[f(x \mid pa'_X, \varepsilon''_X)\right] = \\ &\log \mathbb{E}_{E''_X}\left[\exp \log f(x \mid \theta_X = \Theta_X(pa'_X, \varepsilon''_X))\right].\end{aligned} \tag{3.5}$$

  We can simplify the expectation since $(E''_X \perp\!\!\!\perp Pa'_X)$, with X as a collider. We also use the log-sum-exp trick for numerical stability.

- **Abduct**. As with the Mixture DCU, the downside to this method is that we cannot find the ε''_X that generated θ_X and the corresponding x. However:

$$\begin{aligned}P(E'_X, E''_X \mid x, pa'_X) = &P(E''_X \mid x, pa'_X)\, P(E'_X \mid x, pa'_X, E''_X) = \\ &P(E''_X \mid x, pa'_X)\, P(E'_X \mid x, \theta_X = \Theta_X(pa'_X, E''_X)).\end{aligned} \tag{3.6}$$

  The first term is covered by conditional sampling, generating N i. i. d. samples $\varepsilon''_X \sim P(E''_X \mid pa'_X) = P(E''_X)$ since $(E''_X \perp\!\!\!\perp Pa'_X)$, using $s(\log P(x \mid pa'_X, \varepsilon''_X))$ as weights. The second term comes directly from the internal DCU abduction, as conditioning on pa'_X and ε''_X gives us $\theta_X = \Theta_X(pa'_X, \varepsilon''_X)$.

This technique allows us to implement more **flexible forms of simpler DCU implementations** with ease: with just one component, we can implement uncountable mixtures of components by moving the stochasticity of the mixture to this additional noise signal, introduced as a new input to the parameter's network. See Fig. 3.3 for

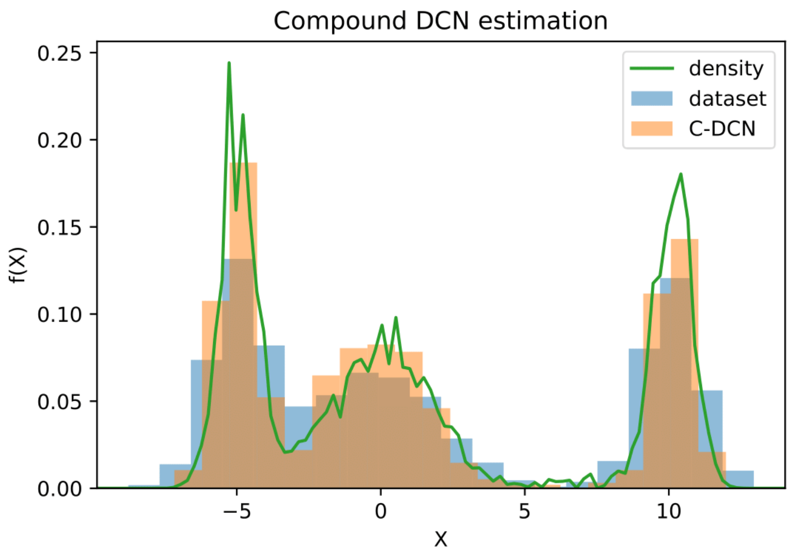


FIGURE 3.3: Compound DCN density estimation example. This figure shows the histograms of the training data and a sample extracted from the model, in addition to the estimated density. Given data following a mixture of three Normal components, a Compound Normal-DCN models their mixture only using a secondary noise signal.

a density estimation example using a Compound Normal-DCN. Note that this technique can be applied to any other kinds of DCU, as it makes **no assumptions about their internal structure**. In the example in section 7.1 we will also see the potential of Compound DCUs, especially for training in small datasets.

Chapter 4

Deep Causal Graphs

In this chapter we will show how DCUs fit in the overall Deep Causal Graph framework, detail its estimation procedures and explain how to aggregate every DCU's Conditioner network into a single Graphical Conditioner.

4.1 Estimation Procedures

Let us define some notation needed for the following equations. When referring to a subset of variables $\mathbf{V} \subset \mathcal{V}$ or $\mathbf{E} \subseteq \mathcal{E}$, let $\mathbf{V}^c := \mathcal{V} \setminus \mathbf{V}$, $\mathbf{E}^c := \mathcal{E} \setminus \mathbf{E}$. Let us also denote $\mathcal{E}_{\mathbf{V}} := \{E_V \in \mathcal{E} \mid V \in \mathbf{V}\}$ (then $\mathcal{E}^c_{\mathbf{V}} := \mathcal{E} \setminus \mathcal{E}_{\mathbf{V}}$). We will operate with the Parallel Worlds Graph model [5] described in section 1.2.2, where we replicate all variables in $\mathcal{V}$ to the new counterfactual world $\mathcal{V}_{\mathbf{x}}$ subject to intervention $do(\mathbf{X} = \mathbf{x})$, with only $\mathcal{E}$ and $\mathcal{U}$ being shared. This allows us to distinguish between expressions like $P(Y_{\mathbf{x}} \mid Z)$ and $P(Y_{\mathbf{x}} \mid Z_{\mathbf{x}})$ (pre- and post-intervention conditionals).

4.1.1 Observational and Interventional Queries

First and foremost, we define the **training objective**. As discussed in section 1.2.3, SCMs and therefore DCGs can be trained using Maximum Likelihood Estimation: if there are no latent confounders ($\mathcal{U} = \varnothing$), then,

$$\log P(\mathcal{V}) = \sum_{V \in \mathcal{V}} \log P(V \mid Pa'_V); \tag{4.1}$$

otherwise,

$$\log P(\mathcal{V}) = \log \mathbb{E}_{\mathcal{U}} \left[P(\mathcal{V} \mid \mathcal{U}) \right] = \log \mathbb{E}_{\mathcal{U}} \left[\exp \sum_{V \in \mathcal{V}} \log P(V \mid Pa'_V) \right]. \tag{4.2}$$

This expectation can be approximated by sampling N i. i. d. values from our SCM's prior $P(\mathcal{U})$. The use of logarithms helps with numerical stability, and we also use the log-sum-exp trick for the latter case. Each of the terms $\log P(V \mid Pa'_V)$ can be estimated with the corresponding DCU *loglk* operation, which is required to be differentiable *w.r.t.* the network's parameters. This allows us to optimize the model $\mathcal{M}$, with a view to maximizing the average log-likelihood of an i. i. d. dataset $\mathcal{D}$ generated with the underlying distribution $P(\mathcal{V})$ that we wish to model.

Secondly, we develop the **sampling routine** to generate samples $\mathbf{v} \sim P(\mathcal{V})$. To this end, we sample values $\boldsymbol{\varepsilon} \sim P(\mathcal{E})$, $\mathbf{u} \sim P(\mathcal{U})$ from their respective priors, and, for each node $V \in \mathcal{V}$, following a topological order of the graph, we use its *sample* operation, passing its parents' values (which may include subsets from $\mathbf{u}$) and its exogenous noise signal $\boldsymbol{\varepsilon}_V$. This generates values $\mathbf{v}$, which follow the DCG's observational distribution $P(\mathcal{V})$. Additionally, to sample from $P(\mathcal{V}_{\mathbf{x}})$ ($\mathcal{V}$ in $\mathcal{M}_{\mathbf{x}}$, subject to $do(\mathbf{X} = \mathbf{x})$), we employ the previous procedure on the intervened model $\mathcal{M}_{\mathbf{x}}$, replacing each f_X by $X := x$ for all $X \in \mathbf{X}$.

Next, we study how to **estimate log-likelihoods** of subsets $\mathbf{V} \subset \mathcal{V}$. Note that:

$$\log P(\mathbf{V}) = \log \mathbb{E}_{\mathcal{E}^c_{\mathbf{V}},\mathcal{U}} \left[P(\mathbf{V} \mid \mathcal{E}^c_{\mathbf{V}}, \mathcal{U}) \right] = \log \mathbb{E}_{\mathcal{E}^c_{\mathbf{V}},\mathcal{U}} \left[\exp \sum_{V \in \mathbf{V}} \log P(V \mid Pa'_V) \right] \tag{4.3}$$

Each term comes from the DCU's *loglk* operation and its parent values result from applying the previous sampling procedure to fill any variables in $\mathbf{V}^c$. We can also compute conditional queries, simply by realizing that $P(\mathbf{V} \mid \mathbf{z}) = \frac{P(\mathbf{V},\mathbf{z})}{P(\mathbf{z})}$, both of these terms being computable with the previous procedure. Then, in the presence of interventions $do(\mathbf{X} = \mathbf{x})$, we can simply consider the intervened model $M_{\mathbf{x}}$ to answer the aforementioned kinds of queries, either $P(\mathbf{V}_{\mathbf{x}})$ or $P(\mathbf{V}_{\mathbf{x}} \mid \mathbf{z}_{\mathbf{x}})$.

Finally, let us consider **expectation queries** $\mathbb{E}_{\mathbf{V}} [f(\mathbf{V})]$ for arbitrary functions f. These can be estimated using Monte Carlo by taking N i. i. d. samples from $\mathbf{V}$ with the methods detailed above and then averaging the resulting samples $(f(\mathbf{v}^{(i)}))_{i=1..N}$ to estimate the expectation. For conditional queries, in the form $\mathbb{E}_{\mathbf{V}|\mathbf{z}} [f(\mathbf{V})]$, we can use importance sampling along with the **softmax trick**:

Theorem 4 (Softmax trick)
*Given disjoint sets $\boldsymbol{V}, \boldsymbol{Z} \subset \mathcal{V}$, a sample $\boldsymbol{z} \sim P(\boldsymbol{Z})$, and N i. i. d. unconditional samples $\boldsymbol{v} := (\boldsymbol{v}^{(i)})_{i=1..N} \sim P(\boldsymbol{V})$, we can approximate expectations over conditional distributions using the so-called **softmax trick**:*

$$\mathbb{E}_{\boldsymbol{V}|\boldsymbol{z}} [f(\boldsymbol{V})] \approx \sum_{i=1..N} f(\boldsymbol{v}^{(i)}) \cdot s(\log P(\boldsymbol{z} \mid \boldsymbol{v}))^{(i)}, \tag{4.4}$$

with s the softmax operation applied on the set $(\log P(\boldsymbol{z} \mid \boldsymbol{v}^{(i)}))_{i=1..N}$.

Proof of Theorem 4.
We apply importance sampling to add a weighting term, that we transform as such:

$$\mathbb{E}_{\mathbf{V}|\mathbf{z}} [f(\mathbf{V})] = \mathbb{E}_{\mathbf{V}} \left[f(\mathbf{V}) \frac{P(\mathbf{V} \mid \mathbf{z})}{P(\mathbf{V})} \right] = \mathbb{E}_{\mathbf{V}} \left[f(\mathbf{V}) \frac{P(\mathbf{z} \mid \mathbf{V})}{P(\mathbf{z})} \right].$$

We can approximate the expectation and the denominator ($P(\mathbf{z}) = \mathbb{E}_{\mathbf{V}}\left[P(\mathbf{z} \mid \mathbf{V})\right]$) using Monte Carlo by taking N i. i. d. samples $\mathbf{v} := (\mathbf{v}^{(i)})_{i=1..N} \sim P(\mathbf{V})$ so that

$$
\begin{aligned}
\mathbb{E}_{\mathbf{V}|\mathbf{z}}\left[f(\mathbf{V})\right] \approx \frac{1}{N} \sum_{i=1..N} f(\mathbf{v}^{(i)}) \cdot \frac{P(\mathbf{z} \mid \mathbf{v}^{(i)})}{\frac{1}{N}\sum_{i=1..N} P(\mathbf{z} \mid \mathbf{v}^{(i)})} &= \\
\sum_{i=1..N} f(\mathbf{v}^{(i)}) \cdot \frac{\exp \log P(\mathbf{z} \mid \mathbf{v}^{(i)})}{\sum_{i=1..N} \exp \log P(\mathbf{z} \mid \mathbf{v}^{(i)})} &= \\
\sum_{i=1..N} f(\mathbf{v}^{(i)}) \cdot s(\log P(\mathbf{z} \mid \mathbf{v}))^{(i)}.
\end{aligned}
$$

□

Lastly, any of these queries subject to an intervention $do(\mathbf{X} = \mathbf{x})$ (if conditional, only if the conditioning $\mathbf{Z}$ is post-interventional, $\mathbf{Z}_{\mathbf{x}}$) are treated as previously, considering the intervened model $\mathcal{M}_{\mathbf{x}}$.

4.1.2 Counterfactual Queries

In this section, we will cover **counterfactual expectations**. Although we employ the 3-step process (abduction, intervention, prediction) described by Pearl in [1], we derive the formula to point out where abduction should take place and how DCU operations help perform the desired estimation. We begin with the simplest counterfactual query (no "missing" variables) and extend it to the general case.

Given a DAG $\mathcal{G}$ with no latent confounders ($\mathcal{U} = \varnothing$), disjoint sets $\mathbf{V}, \mathbf{X} \subset \mathcal{V}$, a sample $v \sim P(\mathcal{V})$ and an intervention $do(\mathbf{X} = \mathbf{x})$, let us consider the query $\mathbb{E}_{\mathbf{V}_{\mathbf{x}}|v}\left[f(\mathbf{V}_{\mathbf{x}})\right]$. In simpler terms, we want to perform an expectation of a function of some variables $\mathbf{V}$ subject to an intervention $do(\mathbf{X} = \mathbf{x})$ and conditioned on every factual variable in $\mathcal{V}$. In order to sample from $\mathbf{V}_{\mathbf{x}}$, we need values for $\mathcal{E}^c_{\mathbf{X}}$ (no need for the exogenous variables corresponding to $\mathbf{X}$, since they attain their value directly from the intervention), with which we can apply the sampling mechanism of the graph to obtain values deterministically for $\mathbf{V}_{\mathbf{x}}$. Therefore the expectation over $P(\mathbf{V}_{\mathbf{x}} \mid v)$ is equivalent to an expectation over $P(\mathcal{E}^c_{\mathbf{X}} \mid v)$:

$$
\mathbb{E}_{\mathbf{V}_{\mathbf{x}}|v}\left[f(\mathbf{V}_{\mathbf{x}})\right] = \mathbb{E}_{\mathcal{E}^c_{\mathbf{X}}|v}\left[f(\mathbf{V}_{\mathbf{x}}(\mathcal{E}^c_{\mathbf{X}}))\right]. \tag{4.5}
$$

We can approximate this expectation with Monte Carlo using M i. i. d. samples from $P(\mathcal{E}^c_{\mathbf{X}} \mid v)$, which results from applying the DCU *abduct* operation on every node independently:

$$
\begin{aligned}
P(\mathcal{E}^c_{\mathbf{X}} \mid v) = \prod_{k=1..K,\ V_k \notin \mathbf{X}} P(E_k \mid v, \mathcal{E}_{<k} \cap \mathcal{E}^c_{\mathbf{X}}) &= \\
\prod_{k=1..K,\ V_k \notin \mathbf{X}} P(E_k \mid v_k, pa'_k, \mathcal{E}_{<k} \cap \mathcal{E}^c_{\mathbf{X}}) &= \\
\prod_{k=1..K,\ V_k \notin \mathbf{X}} P(E_k \mid v_k, pa'_k).
\end{aligned} \tag{4.6}
$$

Note that $\forall k \neq l, V_l \notin Pa'_k$, $(E_k \perp\!\!\!\perp V_l \mid V_k, Pa'_k, \mathcal{E}_{<k} \cap \mathcal{E}^c_{\mathbf{X}})$, since every path connecting them (if one exists) is either $E_k \to V_k \to \dots$, with V_k as a chain, or $E_k \to V_k \leftarrow Pa_k \overset{?}{—} \cdots$, with Pa'_k as a chain or a fork; hence, either V_k or Pa'_k blocks these paths, which allows us to remove every other $V_l \in \mathcal{V}$ from the conditional. Secondly, $\forall k \neq l$, $(E_k \perp\!\!\!\perp E_l \mid V_k, Pa'_k)$ by the same reasoning; therefore, we can simplify the conditional to (v_k, pa'_k). What results is precisely the DCU *abduct* operation.

In summary, $\mathbb{E}_{\mathbf{V}_{\mathbf{x}}|v}\left[f(\mathbf{V}_{\mathbf{x}})\right]$ can be approximated with Monte Carlo by abducting M i. i. d. samples independently node by node in $\mathcal{V} \setminus \mathbf{X}$; then we can use these latent samples to generate values for $\mathcal{V}_{\mathbf{x}} \setminus \mathbf{X}_{\mathbf{x}}$ (every remaining value in $\mathcal{M}_{\mathbf{x}}$) and pass them through f before finally averaging them to obtain an estimation.

Let us now consider the query $\mathbb{E}_{\mathbf{V}_{\mathbf{x}}|\mathbf{z}}\left[f(\mathbf{V}_{\mathbf{x}})\right]$ with a sample $\mathbf{z} \sim P(\mathbf{Z})$ from $\mathbf{Z} \subset \mathcal{V}$, and a graph $\mathcal{G}$ that may contain latent confounders; if $\mathcal{U} \cup (\mathcal{V} \setminus \mathbf{Z}) \neq \varnothing$, there is "missing" information and the estimator becomes more complex. Since the DCU *abduct* operation requires a value for every parent of the node and the node itself, some nodes will be missing information; hence, we need to marginalize over $\mathcal{E}^c_{\mathbf{Z}}$ and $\mathcal{U}$ conditioned on $\mathbf{z}$, given that these two variables and $\mathbf{Z}$ allow us to obtain values for the remaining $\mathbf{Z}^c$ deterministically:

$$\begin{aligned}
\mathbb{E}_{\mathbf{V}_{\mathbf{x}}|\mathbf{z}}\left[f(\mathbf{V}_{\mathbf{x}})\right] = \mathbb{E}_{\mathcal{E}^c_{\mathbf{Z}},\mathcal{U}|\mathbf{z}}&\left[\mathbb{E}_{\mathbf{V}_{\mathbf{x}}|\mathbf{z},\mathcal{E}^c_{\mathbf{Z}},\mathcal{U}}\left[f(\mathbf{V}_{\mathbf{x}})\right]\right] = \\
&\mathbb{E}_{\mathcal{E}^c_{\mathbf{Z}},\mathcal{U}}\left[\mathbb{E}_{\mathbf{V}_{\mathbf{x}}|\mathbf{z},\mathcal{E}^c_{\mathbf{Z}},\mathcal{U}}\left[f(\mathbf{V}_{\mathbf{x}})\right]\frac{P(\mathbf{z} \mid \mathcal{E}^c_{\mathbf{Z}},\mathcal{U})}{P(\mathbf{z})}\right] \approx \\
&\sum_{i=1}^{N}\mathbb{E}_{\mathcal{E}_{\mathbf{Z}\setminus\mathbf{X}}|\mathbf{z},{\varepsilon^c_{\mathbf{Z}}}^{(i)},\mathbf{u}^{(i)}}\left[f(V_{\mathbf{x}}({\varepsilon^c_{\mathbf{Z}}}^{(i)},\mathbf{u}^{(i)},{\mathcal{E}_{\mathbf{Z}\setminus\mathbf{X}}}^{(i)}))\right] s(\log P(\mathbf{z} \mid \varepsilon^c_{\mathbf{Z}},\mathbf{u}))^{(i)}.
\end{aligned} \tag{4.7}$$

We use importance sampling and the softmax trick to sample unconditionally. On the other hand, the internal expectation can be estimated with the previous procedure, by abducting every node independently with the available information and sampling in the intervened model; this generates values for $\mathcal{V}_{\mathbf{x}} \setminus \mathbf{X}_{\mathbf{x}}$ as before, which we use to finally answer our query.

Further queries could be answered by means of the three DCU operations, using similar derivations. Evidently, each estimation procedure results in different estimators with more or less variance, but the fact that we can train a single model for an arbitrary graph and employ it for any of these (identifiable) queries using a general estimator is, in our view, more powerful for the end-user than the variety of ad-hoc models present in the estimand-based literature. Moreover, our library already provides utilities for all of these procedures, so that practitioners can apply them to their problems directly.

4.1.3 Conditional Sampling

The remaining operations concern the procedure by which we can sample from observational, interventional and counterfactual distributions when there are some conditioning values. Let us consider an observational distribution $P(\mathbf{V} \mid \mathbf{z})$, from which we want to generate N i. i. d. samples $(\mathbf{v}^{(i)})_{i=1..N}$. Since $P(\mathbf{V} \mid \mathbf{z}) = P(\mathbf{V})\frac{P(\mathbf{V}|\mathbf{z})}{P(\mathbf{V})}$, we can:

1. Generate M samples $(\mathbf{v}^{(i,j)})_{j=1..M} \sim P(\mathbf{V})$ from unconditioned $\mathbf{V}$.
2. Choose one of these M samples by weighted sampling, with *unnormalized* weights $\widetilde{w}^{(i,j)} := \frac{P(\mathbf{v}^{(i,j)}|\mathbf{z})}{P(\mathbf{v}^{(i,j)})}$.
3. Repeat N times, to generate each $v^{(i)}$.

The corresponding normalized weights $w^{(i,j)}$ result in the softmax operation:

$$w^{(i,j)} := \frac{\widetilde{w}^{(i,j)}}{\sum_{j=1..M} \widetilde{w}^{(i,j)}} = \frac{\exp \log \widetilde{w}^{(i,j)}}{\sum_{j=1..M} \exp \log \widetilde{w}^{(i,j)}} = s(\log \widetilde{w}^{(i,\cdot)})^{(j)}. \tag{4.8}$$

Note that $\log \widetilde{w}^{(i,j)} = \log P(\mathbf{v}^{(i,j)} \mid \mathbf{z}) - \log P(\mathbf{v}^{(i,j)})$, and both terms can be obtained through the above procedures.

One downside of this technique is that it requires M unconditional samples to generate each conditional sample. In order to mitigate this problem, we can generate M samples once, and then take N subsamples with replacement. Both alternatives are valid procedures, provided M is big enough. Another potential point of failure happens if the conditioning term $\mathbf{Z} = \mathbf{z}$ is highly unlikely for most of the sampled values $\mathbf{v}$. In these cases, a bigger value for M is also required.

This solves the **observational case** $P(\mathbf{V} \mid \mathbf{z})$. **Interventional conditioned distributions** $P(\mathbf{V}_\mathbf{x} \mid \mathbf{z}_\mathbf{x})$ (subject to intervention $do(\mathbf{X} = \mathbf{x})$) are handled in the same way, but in the intervened SCM $\mathcal{M}_\mathbf{x}$. Finally, for **counterfactual distributions** $P(\mathbf{V}_\mathbf{x} \mid \mathbf{z})$, note that we can focus on sampling from every latent variable (conditioned on $\mathbf{z}$), and then follow the deterministic functions in $\mathcal{F}$. Let us split $\mathcal{E} = \mathcal{E}_\mathbf{Z} \cup \mathcal{E}_\mathbf{Z}^c$. Then:

$$\begin{aligned} P(\mathcal{U}, \mathcal{E}_\mathbf{Z}, \mathcal{E}_\mathbf{Z}^c \mid \mathbf{z}) &= P(\mathcal{U}, \mathcal{E}_\mathbf{Z}^c \mid \mathbf{z}) \cdot P(\mathcal{E}_\mathbf{Z} \mid \mathbf{z}, \mathcal{U}, \mathcal{E}_\mathbf{Z}^c) = \\ & P(\mathcal{U}, \mathcal{E}_\mathbf{Z}^c \mid \mathbf{z}) \cdot \prod_{k:\, V_k \in \mathbf{Z}} P(E_k \mid \mathbf{z}, \mathcal{U}, \mathcal{E}_\mathbf{Z}^c, \mathcal{E}_{<k} \cap \mathcal{E}_\mathbf{Z}) = \\ & P(\mathcal{U}, \mathcal{E}_\mathbf{Z}^c) \cdot \frac{P(\mathcal{U}, \mathcal{E}_\mathbf{Z}^c \mid \mathbf{z})}{P(\mathcal{U}, \mathcal{E}_\mathbf{Z}^c)} \cdot \prod_{k:\, V_k \in \mathbf{Z}} P(E_k \mid v_k, pa'_k). \end{aligned} \tag{4.9}$$

We transform the first term as before, in order to bypass the conditional with weighted sampling. The second term can be decomposed following the topological order of the graph, focusing on each $E_k \in \mathcal{E}_\mathbf{Z}$ individually. The long conditional can be simplified, since $(E_k \perp\!\!\!\perp \mathcal{U} \setminus Pa'_k, \mathcal{E}_k^c \mid V_k, Pa'_k)$ (either V_k acts as a chain or Pa'_k acts as a chain or a fork); therefore, we can simplify the conditional to the corresponding v_k value (given by $\mathbf{z}$) and its parents pa'_k (computed deterministically from $\mathbf{z}, \mathcal{U}$ and the required noise in $\mathcal{E}_{<k}$). Finally, each of these terms can be sampled independently using the DCU's *abduct* operation.

To summarize, in order to generate N i. i. d. samples $(\mathbf{v_x}^{(i)})_{i=1..N} \sim P(\mathbf{V_x} \mid \mathbf{z})$:

1. Generate M values $((\mathbf{u}^{(i,j)}, {\varepsilon^c_{\mathbf{Z}}}^{(i,j)}))_{j=1..M} \sim P(\mathcal{U}, \mathcal{E}^c_{\mathbf{Z}})$ from the unconditioned priors $P(\mathcal{U}, \mathcal{E}^c_{\mathbf{Z}})$, mutually and internally independent.

2. Choose one of these M samples by weighted sampling, with *unnormalized* weights $\widetilde{w}^{(i,j)} := \frac{P(\mathbf{u}^{(i,j)}, {\varepsilon^c_{\mathbf{Z}}}^{(i,j)} | \mathbf{z})}{P(\mathbf{u}^{(i,j)}, {\varepsilon^c_{\mathbf{Z}}}^{(i,j)})}$. Again, the corresponding normalized weights result from the softmax operation: $w^{(i,j)} := s(\log \widetilde{w}^{(i,\cdot)})^{(j)}$.

3. Repeat N times, to generate each pair $(\mathbf{u}^{(i)}, {\varepsilon^c_{\mathbf{Z}}}^{(i)})_{i=1..N}$.

4. Abduct every node $V_k \in \mathbf{Z}$ to generate values $\{{\varepsilon_{\mathbf{Z}}}^{(i)}\}_{i=1..N}$, using the given v_k (in $\mathbf{z}$) and its parents' values, that come either from $\mathbf{z}$ or from the deterministic application of $\mathcal{F}$.

5. Apply the deterministic functions in $\mathcal{F}_{\mathbf{x}}$ (subject to intervention $do(\mathbf{X} = \mathbf{x})$) to obtain the desired counterfactual samples $({v_{\mathbf{x}}}^{(i)})_{i=1..N}$.

Again, we can avoid sampling $N \times M$ unconditioned samples by subsampling with replacement, provided M is big enough. This procedure allows us to inspect counterfactual distribution shapes, not only expectations of counterfactuals, which we will explore in chapter 5.3.

4.2 Graphical Conditioner

We finish this chapter describing the technique that allows us to **encompass all DCU subnetworks into a single network, the Conditioner**. This is based on [49], which proposes a Conditioner for Normalizing Flows that respects any independencies described by a DAG. Our Conditioner, however, has an additional requirement, as it models every node's parameters Θ_X, which are heterogeneous between nodes.

Let us consider a DCG $\mathcal{M}$ with variables $\mathcal{V}$. For simplicity of notation, we denote $\mathcal{V}' := \mathcal{V} \cup \mathcal{U}$, $K := |\mathcal{V}|$, $K' := |\mathcal{V}'| = |\mathcal{V}| + |\mathcal{U}|$, with $\mathcal{V} = (V_1, \ldots, V_K)$ in a topological order and $\mathcal{U} = \{U_{K+1}, U_{K'}\}$ in an arbitrary fixed order. Each node $V_k \in \mathcal{V}$ depends on parameters Θ_k; let us define $\widehat{\Theta} := (\widehat{\Theta}_1, \ldots, \widehat{\Theta}_D)$, the concatenation of all Θ_k in order and $D := |\widehat{\Theta}|$, the total dimensionality of the concatenated parameters. Since $P(V_1, \ldots, V_K) = \mathbb{E}_{\mathcal{U}}\left[\prod_{k=1..K} P(V_k \mid Pa'_k)\right]$ and each term depends on $\theta_k = \Theta_k(Pa'_k)$, we define a **masking matrix** A with shape $K' \times D$, where each term $a_{k,d}$ is an indicator for whether the k-th variable (either from $\mathcal{V}$ or $\mathcal{U}$) is an input for $\widehat{\Theta}_d$[1]. Given the

[1]Normally, every single parameter inside a node depends on the whole set of Markov Parents for that node; therefore, every one of their masking matrix columns is identical. Nonetheless, we provide a general definition to include the case where certain parameters are dependent on a subset of their node's parents; this does not affect performance, since identical columns are processed simultaneously, as we will see in the following.

$$\begin{tikzcd}
Z \arrow[dr] \arrow[dd, leftrightarrow, dashed] & \\
& Y \\
X \arrow[ur] &
\end{tikzcd}$$

(A) Example graph.

$$\begin{array}{cc|c|c|c|}
 & & p_Z & p_X & p_Y \\
\hline
\mathcal{V} & Z & 0 & 0 & 1 \\
\hline
 & X & 0 & 0 & 1 \\
\hline
 & Y & 0 & 0 & 0 \\
\hline
\mathcal{U} & U_{\{Z,X\}} & 1 & 1 & 0 \\
\hline
\end{array}$$

(B) Masking matrix.

FIGURE 4.1: Graphical Conditioner Masking Matrix example.

graph in Fig. 4.1a, the corresponding masking matrix is the one found in Fig. 4.1b. Assuming that every variable is binary, we model them with Bernoulli-DCNs, each with a single p parameter; hence, $\widehat{\Theta} = (p_Z, p_X, p_Y)$.

We can define an arbitrary neural network h with K' inputs and D outputs that, given values for $\mathcal{V}$ and $\mathcal{U}$, returns values for parameters $\widehat{\Theta}$. Note that the resulting parameters still need an activation function pass to transform them to their appropriate domain; each DCU implementation takes care of this step internally.

If we used this network directly, $\widehat{\Theta} = h(\mathbf{v}, \mathbf{u})$, we would not be respecting the independencies defined by our graph $\mathcal{G}$. However, when computing a parameter Θ_d corresponding to a certain node V_k, we can multiply the concatenated vector $(\mathbf{v}, \mathbf{u})$ by $A_{\cdot,d}$, so that any variable not a parent of V_k will be masked. Naturally, any parameters with identical columns in A (even from different nodes) can be computed simultaneously with only one mask, optimizing the procedure substantially. We will elaborate on this point in the following.

We first exemplify the sampling and log-likelihood procedure using a shared Graphical Conditioner. These procedures are already implemented in our software library, but we showcase them here to illustrate how to appropriately use the Conditioner in our estimations. In order to **sample**, i.e., generate a value $\mathbf{v} \sim P(\mathcal{V})$, we start by sampling from all latent variables in $(\mathcal{E}, \mathcal{U})$, and then follow the topological order of the graph to process each node sequentially, as usual. Given a node $V_k \in \mathcal{V}$ and the values $v_{<k}$ sampled beforehand, we employ the Graphical Conditioner h to compute the values θ_k corresponding to that node:

1. Pass $(\mathbf{v}, \mathbf{u})$ to h (zero-padding any variables not sampled yet), masking the input using the corresponding mask $A_{\cdot,J_k}$, with J_k being the set of columns corresponding to parameters Θ_k.

2. Slice the resulting vector to columns J_k to obtain the desired values θ_k.

3. Pass θ_k along with pa'_k and ε_k to the DCU *sample* operation for V_k. The resulting sample v_k can then be employed to sample the node's descendants.

The **log-likehood** operation is equivalent, but we do not need to follow a topological order of the graph, as we already have values for all its nodes.

An important **optimization** of this procedure is that we can compute several sets of parameters at once (in a way, fusing their columns in the matrix A), thereby reducing the number of passes required for the Conditioner. Consider a certain topological order of the graph, and divide all nodes into levels based on their depths; given a level of nodes, any nodes with exactly the same Markov parents (therefore, the same mask for their parameters) can be resolved in a single pass of the network. On the other hand, we can replicate the input matrix for each of the masks required for the current level, mask them, concatenate them in the same input matrix and compute all of them at once with a single pass. Moreover, since the log-likelihood operation does not need to consider depth levels separately, this effectively reduces the passes through the Conditioner to a single pass. Note that the alternative, not using a Graphical Conditioner and having separate networks for every node, still results in multiples passes, one per each node subnetwork; therefore, a Graphical Conditioner can optimize these computations substantially, depending on the shape of the graph.

In practice, we can define **any network** h **with arbitrary architecture** to compute our parameters $\widehat{\Theta}$. Not only that, but the use of a **single network**, instead of individual networks for each node, **reduces model complexity, overfitting risk, memory requirements and training and estimation times** significantly. Nevertheless, if a particular DCU implementation requires a more complex Conditioner architecture (e.g., multivariate distributions modelled with an NCF), it is always possible to model it individually, restricting the shared Graphical Conditioner to the rest of the nodes in the graph, thus allowing for specialized modelling in complex nodes while reducing model weight and overfitting risk for simpler distributions.

Chapter 5

DCG Software

This chapter is devoted to the **implementation of the DCG framework** through three custom libraries: `torch-misc`, `flow` and `dcg`, all of them based on the PyTorch Deep Learning library [50]. We will illustrate their functionalities through a synthetic example, covering DCG training modelled with discrete DCNs and continuous NCFs, then estimating many causal queries of observational, interventional and counterfactual nature. The code corresponding to this example can be found as the tutorial notebook in the `dcg` library Github page.

5.1 Salary Dataset

We begin by describing the Salary dataset, a synthetic data generating process designed to mimic gender biases in the workplace, with which we can ask causal queries to ascertain fairness *w.r.t.* salary assignment. We specify the causal graph in Fig. 5.1 with several paths through which gender affects salary indirectly, modelled after common societal biases: gender affects both work field and seniority, which in turn affects salary; gender and age are correlated, having a downstream effect on salary through education level or seniority (mothers, who are generally older, may stop working, which leaves older workers more likely to be males; we model this with a latent confounder named `stay-at-home`).

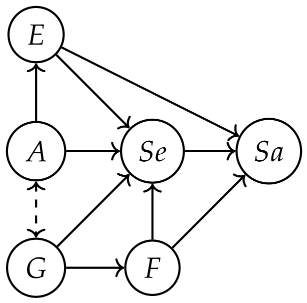


FIGURE 5.1: Salary Dataset Causal Graph. Nodes: age (A), education (E), gender (G), field (F), seniority (Se), salary (Sa).

These variables correspond to three kinds of distribution: gender is a Bernoulli variable (assuming binary gender as a simplification), field is a Categorical variable with three levels, and every other variable is a continuous non-negative variable.

Therefore, we will model these variables with a Bernoulli-DCN, a Categorical-DCN and NCFs, respectively.

5.2 DCG Training

We start by creating the DCG model that will describe this dataset. For that, we need to import the main DCG class, all three types of DCU and an additional one for latent nodes (which we will model after a $\mathcal{N}(0,1)$), all classes needed for the definition of a Flow Transformer and some additional imports:

```
# DCG class:
from dcg.graph import CausalGraph

# DCU classes:
from dcg.distributional.discrete import Bernoulli, Categorical
from dcg.flow import NCF
from dcg.latents import Normal as Latent

# NCF-related imports:
from flow.flow import Sequential as SequentialFlow, inv_flow
from flow.transformer import Affine, RQ_Spline
from flow.modules import Scaler, Normalizer, Softplus
from flow.prior import Normal as NormalPrior

# Other imports:
import torch
from torch import nn
from functools import partial
```

FIGURE 5.2: General imports.

First and foremost, we will use a shared **Graphical Conditioner**, so we need to define the architecture for its network. However, we do not create it directly, but specify a function that receives dimensions and returns the corresponding network, so that the `dcg` library itself creates it with the appropriate inputs. This is what we cover in section 5.2.1. Secondly, for NCF nodes, we need to define their **Transformer flow**, which will receive its parameters from the Shared Conditioner; we discuss their architecture in section 5.2.2. Then, we **create and train a DCG model**, passing it the dataset's graph structure and both creation functions, covered in section 5.2.3. After training, we finish by inspecting the **adjustment** of the model to the observational distribution. This is an iterative process, as the imperfections found in this latter phase inform us of possible architectural changes and hyperparameter calibration that help in defining a better model for $P(\mathcal{V})$. Once we are satisfied with the adjustment, we can proceed to **query estimation** in section 5.3.

5.2.1 Graphical Conditioner

The **Conditioner** network can follow any architecture, as long as its input accepts `input_dim` dimensions and returns `output_dim` dimensions. We will define a function f(input_dim, output_dim, init=None) that creates a PyTorch feed-forward network; see Fig. 5.3. We use the `init` parameter to specify initialization values for the last bias layer, its values provided by the `dcg` library itself taking into account each node's DCU. Any activation functions required to adjust the domain of the resulting parameters are applied by the DCU classes themselves.

```
def net_f(input_dim, output_dim, init=None):
    net = nn.Sequential(
        nn.BatchNorm1d(input_dim, affine=False),
        nn.Linear(input_dim, 256),
        nn.ReLU(),
        nn.Dropout(),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.Dropout(),
        nn.Linear(128, output_dim)
    )

    if init is not None:
        net[-1].bias.data = init

    return net
```

FIGURE 5.3: Graphical Conditioner network definition.

5.2.2 NCF Transformer

In order to use flows with the `dcg` library, we define functions `f(dim, cond_dim=0)` that create the desired flows. `dim` is the dimension of the variable to model, while `cond_dim` is the dimension of the conditioning tensor (the sum of dimensions of the node's parents). Since the Graphical Conditioner already provides values for the flow's parameters, we can ignore `cond_dim`, as we do not need a Conditioner.

We can define a **sequence of flow-Transformers** with `flow.flow.Sequential`, that we imported as `SequentialFlow`. We pass our sequence of flows as arguments to its constructor and end it with two keyword arguments: `dim`, to inform the dimension of each of these blocks, and `prior`, to specify the prior distribution of E_X (usually $\mathcal{N}(0, 1)$, imported as `NormalPrior`). See Fig. 5.4 for this code.

The flow is defined to transform from X to E_X, so we need to take into account the domain of X before we carry out any additional steps. In this example, all our continuous variables are non-negative, so we use the inverse of a Softplus as the first step, to transform back to all reals: for that, we use the function `inv_flow`, which returns the inverse of the flow passed as argument, in this case a Softplus. We'll precede it with a `Scaler` flow that learns an appropriate scale parameter to mitigate numerical instability. Afterwards, we use a `Normalizer` to transform to mean 0 and standard deviation 1, followed by five blocks of Spline [51] transformers (with 8 splits in the $(-3, 3)$ interval) and an Affine flow ($\varepsilon = \sigma x + \mu$) after each spline.

```
def flow_f(dim, cond_dim=0):
    return SequentialFlow(
        Scaler,
        inv_flow(Softplus),
        Normalizer,
        *sum((
            [partial(RQ_Spline, K=8, A=-3., B=3.), Affine]
            for _ in range(5)
        ), []),

        dim=dim, prior=NormalPrior
    )
```

FIGURE 5.4: Flow-Transformer definition.

5.2.3 DCG Creation and Training

The final step is to **define the causal structure** of the DCG and couple it with both functions to create the model. We define the graph with a special formatting, listing every variable with a DCU-alias, its dimensionality and a whitespace-separated list of parents. We then link each DCU-alias with the corresponding DCU class and pass both creation functions. Finally, we need to **warm-start** the model; given an `Xtrain` PyTorch Tensor with N training samples for every measurable variable ($\mathcal{V}$) in the graph, we pass it to the `warm_start` method. See Fig. 5.5.

Next we **train** the created DCG model. We will use several utility functions from `dcg.training` (inherited from `torch_misc.training`), which we will describe first.

The `train` function trains a PyTorch module (our DCG) using Stochastic Gradient Descent with Early Stopping given training and validation data in the form of PyTorch DataLoader instances (`torch.utils.data.DataLoader`). The list of parameters for this class is the following:

- `graph`: graph to train.
- `train_loader`: DataLoader for the training set.
- `val_loader`: DataLoader for the validation set.
- `loss_f`: loss function `f(module, batch)` to train. Returns the loss of every sample in the given batch individually.
- `optimizer`: optimizer to use, inheriting from `torch.optim.Optimizer`. Defaults to AdamW[52].

```
# Define the graph structure
definition = '''
stay_at_home lat 1
gender bern 1 stay_at_home
field cat 3 gender
age cont 1 stay_at_home
education cont 1 age
seniority cont 1 age education gender field
salary cont 1 education seniority field
'''

# Define the NCF DCU with the flow_f network
NCF_trnf = partial(NCF, flow_f=flow_f)

# Parse definition str and create the DCG model
parsed_definition = CausalGraph.parse_definition(
    definition,

    lat=Latent,
    bern=Bernoulli,
    cat=Categorical,
    cont=NCF_trnf
)

graph = CausalGraph.from_definition(
    parsed_definition,
    net_f=net_f # Shared Conditioner
)

# Don't forget to warm start and move to GPU
graph = graph.warm_start(Xtrain).cuda()
```

FIGURE 5.5: DCG definition and creation.

- `optimizer_kwargs`: keyword arguments to pass to the optimizer (e.g., learning rate `lr`, L_2 regularization `weight_decay`).
- `scheduler`: learning rate scheduler to use, from `torch.optim.lr_scheduler`.
- `scheduler_kwargs`: keyword arguments to pass to the scheduler.
- `n_epochs`: maximum number of epochs to train.
- `patience`: maximum number of epochs without improvement.
- `gradient_clipping`: maximum magnitude of the gradient.
- `callback`: callback function to call at every step of training.
- `use_tqdm`: use the `tqdm`[1] library to track training progress.

[1]TQDM Python library: https://github.com/tqdm/tqdm

This function returns:

- `train_losses`: tuples `(epoch, loss)` with the average training loss per epoch.
- `val_losses`: tuples `(epoch, loss)` with the average validation loss per epoch.

We can use `dcg.training.loss_f` to create the loss function to pass to the `train` procedure. We also use `dcg.training.data_loader` to create a DataLoader from `torch.utils.data.TensorDataset` instances containing our training and validation tensors. Finally, `dcg.training.plot_losses` allows us to plot both loss curves to monitor training, passing it the result of the `train` function directly. See Fig. 5.6 for the training call. Note that we can use a different training procedure if we so desire: `graph.nll` returns the negative log-likelihood of every inputted sample, so we can use it for our training loss directly.

```
from dcg.training import *
# This import includes: train, data_loader, TensorDataset,
# loss_f, plot_losses, test_loglk, among others.

train_losses, val_losses = train(
    graph,
    data_loader(
        TensorDataset(Xtrain),
        batch_size=256, drop_last=True
    ),
    data_loader(
        TensorDataset(Xval),
        batch_size=256, drop_last=False
    ),
    loss_f=loss_f(ex_n=100),
    optimizer_kwargs=dict(lr=1e-3, weight_decay=1e-2),
    patience=100
)
```

FIGURE 5.6: DCG training.

We can inspect the **adjustment** of our trained model by various means. First, by using the training loss on a held-out test split of the dataset; this metric can be used to compare between several trained models. The `dcg.training.test_loglk` function accepts a trained DCG model and a Tensor with the test data and returns the average test log-likelihood (higher is better). On the other hand, we can also visually inspect the model's adjustment to the probability distribution by generating N samples from the trained `graph` calling `graph.sample`, which returns a Tensor with N samples of the model's distribution, and plotting these samples in a scatterplot-histogram matrix. See Fig. 5.7; we can see how the marginal distribution of every continuous variable matches with the one resulting from DCG samples, and the scatterplots depict the same pair-wise relationships between these variables. This kind of plots are mere sanity checks, but allow us to find discrepancies in our DCG adjustments, together with the test metric.

5.3 DCG Estimation

Here we describe the main DCG methods we will use throughout the next sections.

- `sample(self, n, target_node, interventions, return_all)`: generates n **samples** from the distribution. If we only want to obtain samples from a certain node, we pass the node's name to `target_node`. If we want to sample from the intervened distribution, we can pass any desired interventions to the `interventions` parameter, as a dictionary. The method will return a Tensor with all variables or only the requested target variable. If `return_all` is `True`, it will also return a dictionary with every node's value (including latent confounders and exogenous noise signals).

- `loglk(self, x, target_node, interventions, cond, ex_n)`: computes the **log-likelihood** of the given samples `x`: $\log P(\mathbf{x})$. If we pass a `target_node`, it will only compute $\log P(Y)$ where Y is the `target_node`. If we pass a conditioning dictionary `Z = { node: value }` with the `cond` parameter, it returns $\log P(\mathbf{x} \mid \mathbf{Z})$. Finally, we can pass any interventions we desire, if we want to compute log-likelihoods in the intervened model. `ex_n` determines how many samples can be generated for any missing variable in equation 4.3.

- `cond_exp(self, x, f, ex_n)`: returns **conditional expectations** $\mathbb{E}_{\mathcal{V}|\mathbf{x}}[f(\mathcal{V})]$, where $\mathcal{V}$ are all observable variables in the graph, f is the provided function (or the identity if `None`), and $\mathbf{x}$ are the conditioning terms, defined by the passed dictionary `x = { node: value }`. If we need to compute additional samples, `ex_n` determines how many will be sampled, as in equation 4.4.

- `counterfactual(self, x, target_node, interventions, ex_n, f, agg)`: computes the **counterfactual expectation** $\mathbb{E}_{\mathcal{V}_\mathbf{t}|\mathbf{x}}[f(\mathcal{V}_\mathbf{t})]$ where $\mathbf{x}$ are the factual, observed variables (either a dictionary or a Tensor with all measured variables), f is a function that is applied to the counterfactual variables (the identity if f is `None`) and $\mathbf{t}$ are the provided interventions. If `target_node` is specified, the method only computes the counterfactual `target_node` and passes it to f, instead of every other variable. `ex_n` determines how many missing and/or abducted samples will be generated for equation 4.7. Finally, if `agg` is `False`, the method doesn't compute the aggregated result, but returns the **counterfactual samples** along with the associated unnormalized weights (use the softmax operation to normalize them), indexing terms, etc., that can be used to perform other kinds of queries (counterfactual sampling, for example); see the documentation for more details.

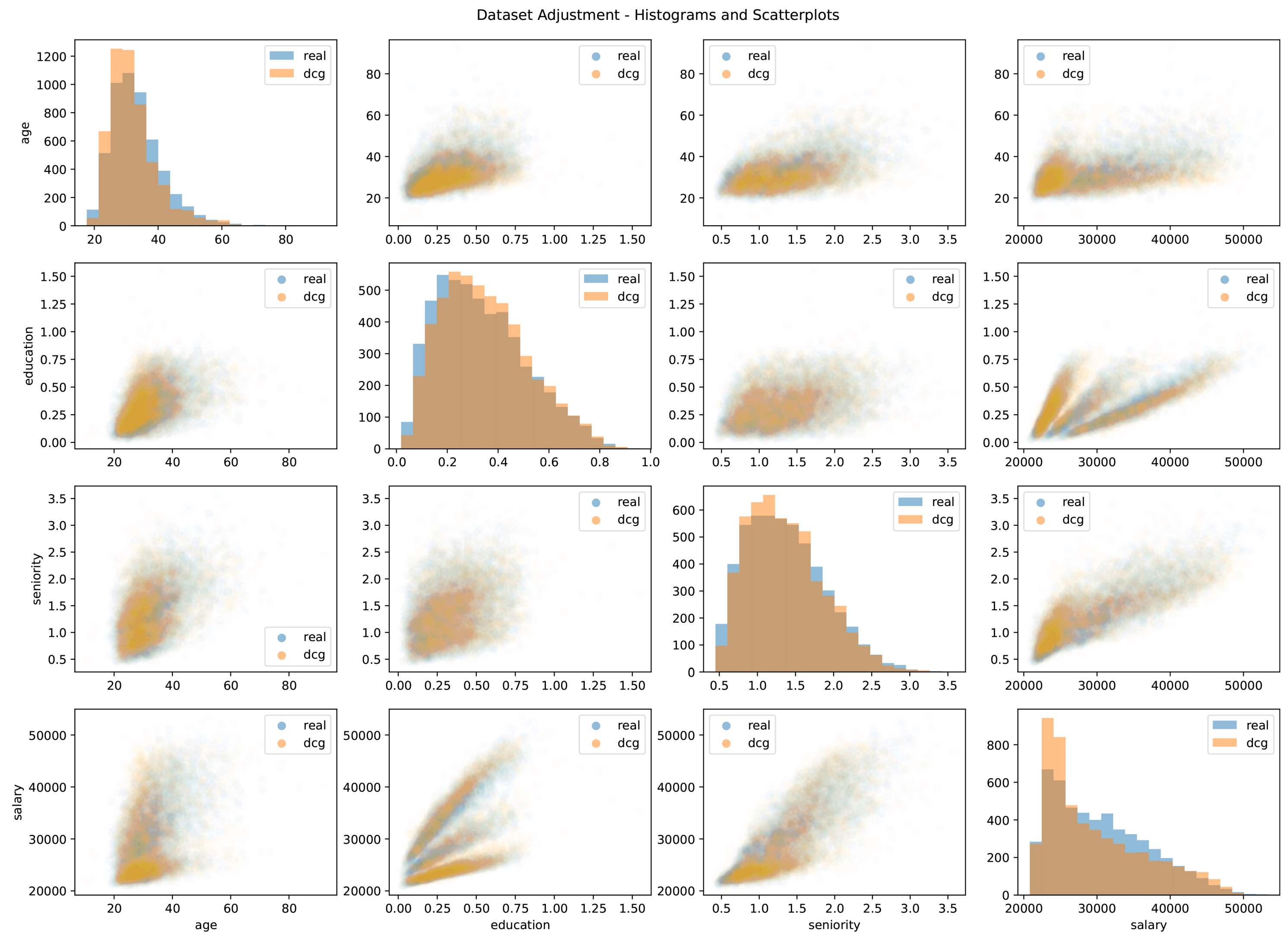


FIGURE 5.7: Scatterplot-Histogram matrix comparing the dataset's distribution (real) and the DCG learnt distribution (dcg).

Most of these functions accept different formatting for its inputs. Values can be passed either as a `torch.Tensor` (in which case it is a matrix with every measured variable in the graph, in the order specified in the definition) or as a dictionary `{ node: value }`. Note that you can always pass dictionaries; full Tensor matrices are only accepted in the `loglk` and `counterfactual` operation `x` parameter. For a dictionary input, its keys can either be node names (strings) or the actual nodes in the graph (accessed with `graph[name]`); its values can either be single values (booleans, integers, floats), NumPy arrays for single dimensional variables or matrices for multi-dimensional variables, or `torch.Tensor`.

We will now exemplify these methods while computing queries of interest.

5.3.1 Observational and Interventional Queries

Let us start with the causal effect of variables X on salary Y, $\mathbb{E}\left[Y \mid do(X = x)\right]$, and compare it with the conditional query $\mathbb{E}\left[Y \mid X = x\right]$. X will be either a Bernoulli variable, a Categorical variable, or a continuous variable; the procedure will essentially be the same. See Fig. 5.8 for the gender-on-salary query: the results for the interventional query are \$35137.79 and \$26399.20 for men and women, respectively, while the observational query returns \$36474.39 and \$25918.47. Note that the observational query overestimates the effect of males on salary, most probably as a result of the latent confounder, which the interventional mechanism ignores because any incoming arrows are discarded when intervening. The Categorical example is analogous, simply intervening with Id_D, the identity matrix of size D, one for each level of the Categorical variable; this results in D samples, one pointing to each level.

Finally, for continuous variables, we define K equidistant points in $[0, 1]$ and compute the corresponding quantiles from the intervened/conditioned variable using training data directly; these are our intervened values, which we can use to sample N values each from the target variable, with which to estimate the interventional/observational effects. See Fig. 5.9 for the corresponding code. We plot both curves for every variable; see Fig. 5.10.

5.3.2 Likelihood Queries

The `loglk` operation allows us to compute many kinds of density queries. Next, we compute the **density** of salary in three cases: unconditioned, conditioning on gender and intervening on gender, all using the same DCG method. See Fig. 5.11 for the required code. We plot the resulting curves along with the corresponding histograms to test their adjustment; Fig. 5.12 shows these estimations. As we can see, every density curve fits its corresponding histogram, which shows that indeed DCGs learn these densities, even the interventional curve, despite only ever training with observational unconditioned data.

```
with torch.no_grad(): # no need to compute gradients
    # Interventional effect
    sample = graph.sample(
        N * 2, target_node='salary',
         interventions={ 'gender': [True, False] }
    )
    # We asked for 2N samples,
    # N for gender=True (male), N for gender=False (female).
    # We aggregate each one separately.
    intv = sample.view(N, 2).mean(0)

    # Observational effect (conditional)
    # We condition on gender for the expectation on salary.
    # We pass the conditioning terms first,
    # then specify f to compute E[f(V) | G=g].
    # If f is a str, f filters V to the specified node.
    obs = graph.cond_exp(
         { 'gender': [True, False] },
         f='salary', ex_n=N
    )
```

FIGURE 5.8: Bernoulli intervention/conditional estimation.

5.3.3 Counterfactual Queries

For counterfactual queries, we focus on individuals for whom we have observed some of their variables, and we want to explore the counterfactual world where a certain variable is intervened. We will operate on this space from the perspective of **fairness**: "what would my salary be had I been a man?". Given a woman with average salary, we will study the (average) counterfactual salary had she been a man; we will also sample from her counterfactual distribution to study its shape.

See Fig. 5.13 for the associated code; the observed salary is $27101.51 while the (average) counterfactual salary is $37676.84. We also inspect the **histogram of counterfactual samples** in Fig. 5.14a. This distribution has some multi-modality, probably due to the effect of each of the three different fields; we test this hypothesis by also conditioning on the first field, with the resulting histogram in Fig. 5.14b, which removes this multi-modality as expected.

This simple example shows the importance of **counterfactual sampling**, since only estimating the expectation would not give us information about the multi-modality in the underlying distribution. However, thanks to these counterfactual samples, we can inspect its shape.

```
K = 20 # generate K values for the intervening variable

for v in ['age', 'education', 'seniority']:
    # df_train is a pd.DataFrame of our training data
    # Generate quantiles from 0 to 1
    intv = df_train[v].quantile(np.linspace(0, 1, K)).values

    with torch.no_grad():
        # Interventional effect
        intv_sample = graph.sample(
            N * K, target_node='salary',
             interventions={v: intv}
        )
        # sample is a Tensor with N * 20 samples, aggregate N
        intv_sample = intv_sample.view(N, K).mean(0)

        # Conditional effect
        # graph.cond_exp needs more memory than graph.sample
        # so ex_n=N needs a lot of memory
        # if we pass every generated quantile.
        # We compute them one by one for this reason.
        cond_sample = np.array([
            graph.cond_exp({v: x}, 'salary', ex_n=N).item()
            for x in intv
        ])

    # Plot both curves
    # ...
```

FIGURE 5.9: Continuous intervention/conditional estimation.

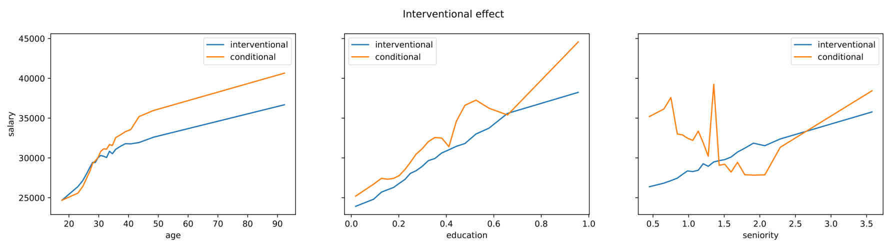
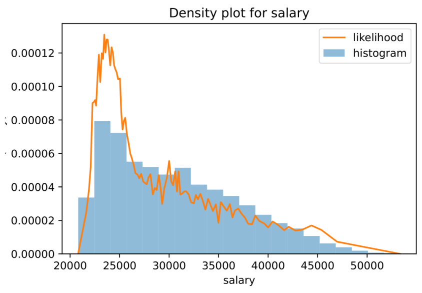


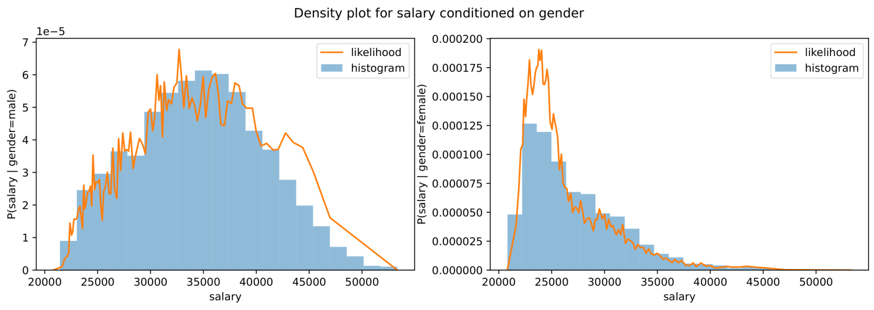
FIGURE 5.10: Continuous intervention/conditional estimation.

```
N = 1000
K = 100

# Compute loglk for a spread of quantiles
qs = df_train.salary.quantile(np.linspace(0, 1, K)).values

with torch.no_grad():
    # Unconditioned observational
    uncond_loglk = graph.loglk(
        { 'salary': qs }, target_node='salary', ex_n=N
    )

    # Conditioned observational
    cond_loglk = [
        graph.loglk(
            { 'salary': qs }, target_node='salary',
            cond={ 'gender': v }, ex_n=N
        )
        for v in [True, False]
    ]

    # Intervened
    intv_loglk = [
        graph.loglk(
            { 'salary': qs }, target_node='salary',
            interventions={ 'gender': v }, ex_n=N
        )
        for v in [True, False]
    ]

    # Plot curves
    # ...
```

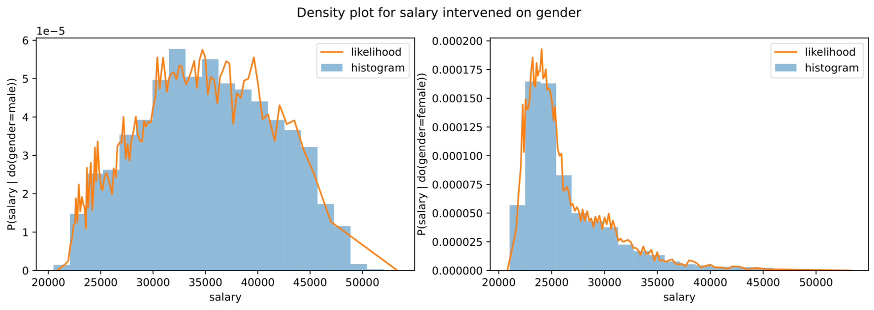
FIGURE 5.11: Likelihood estimation.



(A) Observational density.



(B) Conditional density.

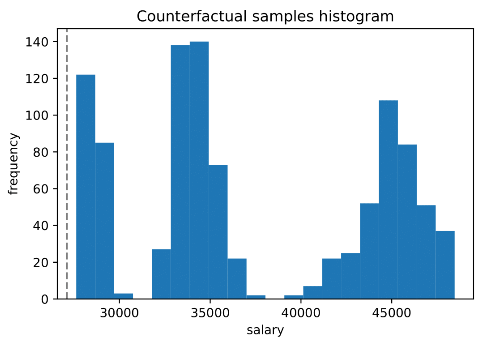


(C) Interventional density.

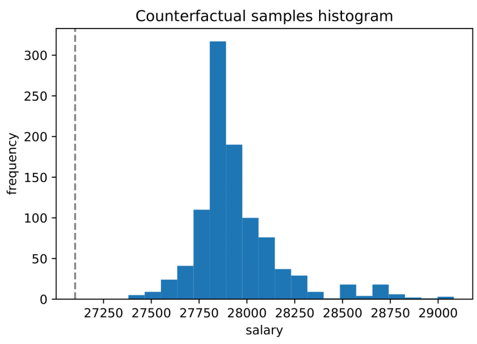
FIGURE 5.12: Density estimation.

```
avg_salary = df_train[~df_train.gender].salary.mean()

with torch.no_grad():
    # Counterfactual Expectation
    cf = graph.counterfactual(
        { 'gender': False, 'salary': avg_salary },
        target_node='salary',
        interventions={ 'gender': True },
    )

    # Counterfactual Sampling
    x, w, _, _, _ = graph.counterfactual(
        { 'gender': False, 'salary': avg_salary },
        target_node='salary',
        interventions={ 'gender': True },
        agg=False,
        ex_n=10000
    )

    # Normalize weights and compute its log
    log_w = torch.nn.functional.log_softmax(w, 0)

    from dcg.sampling import weighted_sampling
    x = weighted_sampling(1000, x, log_w=log_w)
```

FIGURE 5.13: Counterfactual estimation and sampling.



(A) Counterfactual samples.



(B) Counterfactual samples with factual field=A.

FIGURE 5.14: Counterfactual sampling.

Part III

Experiments and Applications

Chapter 6

Estimation Benchmark

This chapter verifies DCG's estimation capabilities for causal queries. The main difficulty in measuring **estimation performance** in a causal setting is that we can never measure both the factual and counterfactual outcome: for every individual/sample, we only have access to one intervention-outcome pair, but never the counterfactual outcome under a different intervention. It is for this reason that we cannot measure error metrics either. As a result, most estimation experiments opt for **semi-synthetic datasets**: real datasets, normally from a randomized experiment, but subject to an artificial sampling process that follows a known Causal Graph. Consequently, we can test our methods against certain causal metrics based on real-world data, even if the underlying causal mechanism is synthetic in nature.

We employ the **IHDP-Jobs benchmark** [9], a well-established estimation benchmark in the Potential Outcomes literature. It consists of two semi-synthetic datasets: the Infant Health Development Program Dataset (IHDP) [53], with a continuous outcome Y, and the Jobs Dataset [54], with a discrete outcome. Working on a graph $(X \to T, Y;\ T \to Y)$, the objective query is the **Individual Treatment Effect** (ITE), $Q := \mathbb{E}\left[Y_1 - Y_0 \mid X\right]$, subject to a binary intervention on T. Along with it, we can also estimate the **Average Treatment Effect** (ATE), $Q := \mathbb{E}\left[Y_1 - Y_0\right]$, considering that $\mathbb{E}\left[Y_t\right] = \mathbb{E}_X\left[\mathbb{E}\left[Y_t \mid X\right]\right]$. Please refer to the `dcg` library Github page for the code of this experiment.

Since both datasets are semi-synthetic, we do have access to these terms and so we can compute the following metrics proposed in [9]: e_{ATE}, Mean Absolute Error (MAE) in the ATE; $\sqrt{e_{PEHE}}$, Root Mean Squared Error (RMSE) in the ITE; e_{ATT}, MAE in the ATE for the Treated ($T = 1$); and R_{pol}, policy risk, also related to ITE. We follow the benchmark's experimental setup: same train, validation and test splits, with estimations performed on 1,000 replications of the IHDP Dataset and ten replications of the Jobs Dataset, which allows for confidence intervals to be obtained for each metric (± 1.96 standard deviations).

We compare our results against many different **Potential Outcome strategies**: simple and bi-headed Linear Regression (LR_1, LR_2), Causal Effect Variational Autoencoder (CEVAE, the only method in the Desiderata table discussed in section 1.2.4 that offered results for this benchmark) [20], Balancing Neural Networks (BNN)

[8], Treatment-Agnostic Representation Network (TAR) [9], Counterfactual Regression (CFR) [9], Adaptively similarity-preserved representation learning for Causal Effect estimation (ACE)[1] [10], Subspace Learning Based Counterfactual Inference (SCI) [11] and Causal Optimal Transport (CausalOT)[2] [12]. Except for CEVAE and our own technique, all methods are estimand-based and therefore specialized in the estimation of these particular queries. These methods constitute the state-of-the-art in estimand-based Causal Query Estimation.

6.1 Experiment Setup

For this particular benchmark, we compare against methods that learn $\mathbb{E}\left[Y_t \mid X\right]$, with T binary and X confounder covariates (25 dimensions for IHDP, 17 for Jobs). Normally, we would model the whole graph node by node, but here we are only interested in the distribution $P(Y \mid X, T)$ (modelled by Y's DCU), as the estimation procedure for each metric does not require modelling either T or X ($X \to T$, therefore intervening T cannot affect X). What we can do instead is define what the `dcg` library calls an *Input DCU*, a placeholder node that accepts i. i. d. values from the dataset. We can employ the training values for these variables as "samples" when performing the usual estimations so that we avoid modelling those nodes, as we do not need to perform either *loglk* or *abduct* with them.

We employ this simplification for two reasons. Firstly, the IHDP experiment consists of 1,000 replications, and a model must be trained for each of them. Not only that, but we employ a 5-Fold Cross-Validation (CV) strategy (detailed below), requiring 5,000 models. Were we to train the whole 27-variable graph, training times for the whole benchmark would be prohibitively long. Additionally, modelling errors in the variables of X would propagate to the outcomes in Y. This would be unfair on our model, which is designed to be graph- and query-agnostic, when comparing against highly specialized models designed for one singular graph and query. We therefore decided on this simplified strategy for the benchmark. This means that we cannot evaluate the effect of error propagation on our estimations, and such an experiment remains for future research.

Regarding the CV strategy, since the number of samples in the IHDP dataset is quite limited (672 training samples) and our flows are highly flexible models, we train five different models instead — one for each CV split of the original training and validation subsets — and join them all together in a single model with a **Mixture DCU**, discussed in section 3.4. We fix their weights as a constant (independent of Pa'_Y), the softmax of their average validation log-likelihood. This mixture allows us to **use the available data effectively**, as the data normally employed for validation can also be used for training in every other submodel. In the following section, we

[1]ACE provide results for e_{PEHE} on IHDP, but not for $\sqrt{e_{PEHE}}$, which makes it impossible to compute when averaging the errors for all replications.

[2]CausalOT provides results for both metrics on IHDP, but they do not specify if these metrics come from the train or test split.

refer to this mixture as Mixture-DCG, whereas Single-DCG represents the single best model in validation from the five splits.

With respect to the DCG architecture, each dataset requires a different type of DCU model, as the IHDP outcome is continuous and the Jobs outcome is binary. For the former, we employ an NCF with Rational Quadratic Flows [51] as in chapter 5.2.2's example, whereas for the latter, we only require a Bernoulli DCN. Both methods use the same Conditioner architecture, consisting of a bi-headed Network, following the example of [9]. Finally, we use the same training procedure as before, with slight hyperparameter adjustments depending on the dataset.

Note that no method in the benchmark uses the factual outcome of Y, denoted by Y_f, in their estimation, only the covariates X and the treatment T. However, our DCGs can employ this information if available, using counterfactual estimation. We compute the resulting metrics without Y_f (DCG) and with Y_f (DCG*) to evaluate how our method responds when we have **post-facto information about the treatment outcome** (e.g., "Would we have recovered had we administered the treatment?"). Table 6.1 presents all of these results; we highlight every method that improves on the rest (except DCG*), and also DCG* if it is the best among them.

6.2 Discussion

DCGs achieve the best performance on e_{ATE} for the IHDP dataset, and second place on e_{ATT} test for the Jobs dataset, improved on only by CEVAE, which is the best model for this query. Note, however, that the train split attains better results when using far simpler methods (LR_1), which suggests that overfitting might be damaging the performance of the rest of the models. On the other hand, CFR [9] is a clear improvement on ITE metrics ($\sqrt{e_{PEHE}}$ and R_{pol}) except for R_{pol} on the training set, whereas DCGs achieve third place for the IHDP dataset, close to TAR. Note that the DCGs we implement for this experiment are essentially equivalent to TAR [9] in all regards (the exact same Conditioner architecture) except that **instead of estimating the expected treatment outcome, they model the actual distribution** through Normalizing Flows, from which we later estimate the expectation. We believe that the added complexity in modelling the distribution (an additional functionality missing from the other methods) might account for this slight drop in performance. CFR, on the other hand, uses balancing regularization between both treatment distributions to improve its results, something that we omitted for this experiment in benefit of simple, general models for arbitrary query estimation. Regarding Single-DCG and Mixture-DCG, the mixture accomplishes better results overall, **demonstrating the applicability of the CV-Mixture technique on small datasets**. Finally, DCG* results in better performance than DCG as expected, since **the addition of the factual outcome provides more information** to draw from; this is a functionality that the other methods do not provide directly.

TABLE 6.1: Metrics on the IHDP-Jobs potential outcomes benchmark. Lower is better.

	IHDP				JOBS			
	e_{ATE}		$\sqrt{e_{PEHE}}$		e_{ATT}		R_{pol}	
	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST
LR$_1$	$.73 \pm .04$	$.94 \pm .06$	$5.8 \pm .3$	$5.8 \pm .3$	$\mathbf{.01 \pm .00}$	$.08 \pm .04$	$.22 \pm .00$	$.23 \pm .02$
LR$_2$	$.14 \pm .01$	$.31 \pm .02$	$2.4 \pm .1$	$2.5 \pm .1$	$.01 \pm .01$	$.08 \pm .03$	$.21 \pm .00$	$.24 \pm .01$
CEVAE [20]	$.34 \pm .01$	$.46 \pm .02$	$2.7 \pm .1$	$2.6 \pm .1$	$.02 \pm .01$	$\mathbf{.03 \pm .01}$	$\mathbf{.15 \pm .00}$	$.26 \pm .00$
BNN [8]	$.37 \pm .03$	$.42 \pm .03$	$2.2 \pm .1$	$2.1 \pm .1$	$.04 \pm .01$	$.09 \pm .04$	$.20 \pm .01$	$.24 \pm .02$
TAR [9]	$.26 \pm .01$	$.28 \pm .01$	$.88 \pm .02$	$.95 \pm .02$	$.05 \pm .02$	$.11 \pm .04$	$.17 \pm .01$	$\mathbf{.21 \pm .01}$
CFR [9]	$.25 \pm .01$	$.27 \pm .01$	$\mathbf{.71 \pm .02}$	$\mathbf{.76 \pm .02}$	$.04 \pm .01$	$.09 \pm .03$	$.17 \pm .01$	$\mathbf{.21 \pm .01}$
ACE [10]	-	-	-*	-*	-	-	$.22 \pm .01$	$.22 \pm .01$
SCI [11]	-	-	-	-	-	-	$.21 \pm .01$	$.23 \pm .01$
CAUSALOT [12]	-*	-*	-*	-*	-	-	$.20 \pm .01$	$.21 \pm .03$
SINGLE-DCG	$.19 \pm .02$	$.22 \pm .02$	$1.0 \pm .08$	$1.0 \pm .08$	$.08 \pm .02$	$.08 \pm .02$	$.24 \pm .01$	$.24 \pm .03$
MIXTURE-DCG	$\mathbf{.12 \pm .01}$	$\mathbf{.17 \pm .01}$	$.93 \pm .07$	$.96 \pm .08$	$.07 \pm .01$	$.07 \pm .02$	$.23 \pm .01$	$.25 \pm .05$
SINGLE-DCG*	$.16 \pm .01$	$.19 \pm .01$	$.94 \pm .07$	$.96 \pm .09$	$.09 \pm .02$	$.07 \pm .02$	$.19 \pm .01$	$.19 \pm .04$
MIXTURE-DCG*	$\mathbf{.12 \pm .01}$	$\mathbf{.15 \pm .01}$	$.85 \pm .06$	$.87 \pm .08$	$.08 \pm .02$	$.07 \pm .02$	$\mathbf{.10 \pm .02}$	$\mathbf{.10 \pm .03}$

In conclusion, DCGs achieve **competitive results against state-of-the-art models** in potential outcome estimation, **even though they do not use ad-hoc estimands** of the query at hand. Rather, they employ general training procedures valid for arbitrary graphs that result in models capable of answering any identifiable query, again with general procedures valid for any other graph. Also note that our model architecture can be interchanged for any other, providing high modularity that could even obtain better results than those achieved here. We expect future work to provide extensions for our technique with more powerful model architectures.

6.3 Modelling Error vs. Estimation Error

Section 1.2.3 explains why the correct estimation of an identifiable Causal Query is viable using our techniques if the model follows the same Causal Structure $\mathcal{G}$ and the same observational distribution $P(\mathcal{V})$ as the underlying data generating process. Although Neural Networks are universal approximators, in practice we cannot realistically achieve the exact same distribution, only **approximate** it. Further work on this subject should study the effect of this discrepancy, and the variance of our estimations resulting from **error propagation** across every node in the graph.

Regarding the former problem, here we perform a minor experiment as a **sanity check**: will the estimation metrics in the previous benchmark improve throughout training, while getting closer to the underlying distribution $P(\mathcal{V})$? We study this by means of the scatterplot in Fig. 6.1, which relates the average negative log-likelihood (training loss) on the test dataset with every metric in the former experiment (including the estimations with Y_f, denoted with a star (*)). Although this does not prove that diminishing modelling errors entail better estimations, it does corroborate the hypothesis, at least for this experiment.

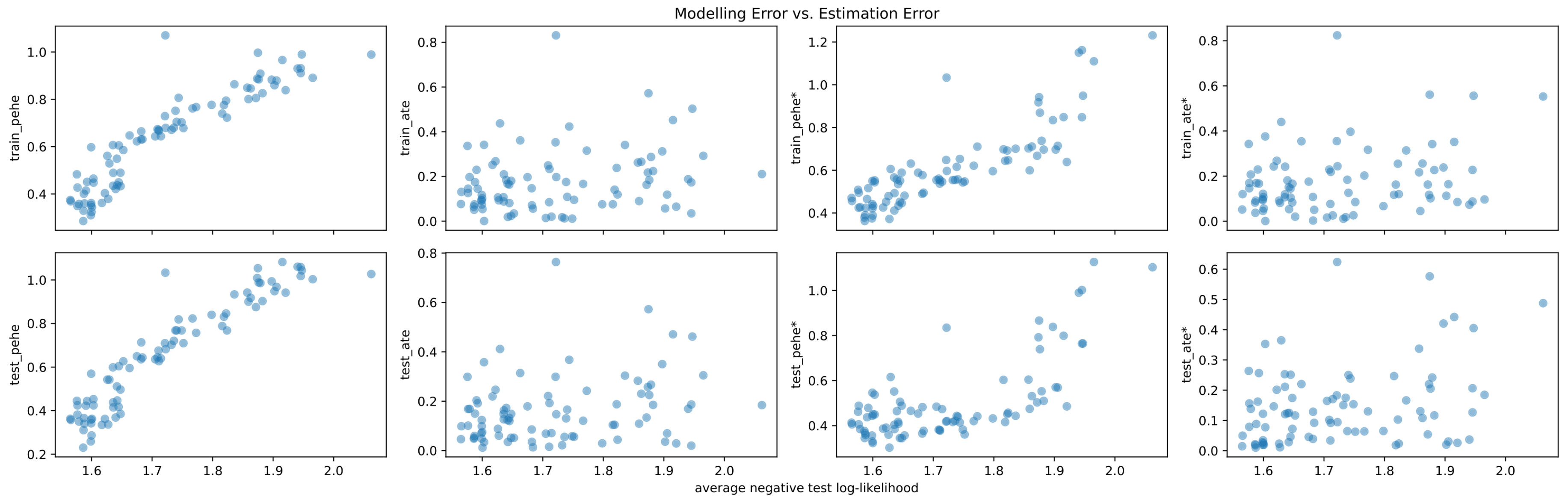


FIGURE 6.1: Average negative test log-likelihood vs. estimation errors throughout training (lower is better) for the IHDP benchmark, first replication, first CV split. As expected, lower modelling error leads to lower estimation error, especially in PEHE-related metrics.

Chapter 7

Applications

In this chapter we showcase our techniques on two **real-world datasets**; the first experiment deals with **Causal Query Estimation**, as a means of understanding an underlying mechanism, while the second one covers **Black-Box Introspection and Counterfactual Fairness**. Please refer to the `dcg` library Github page for the preprocessing and experimental code of these examples.

7.1 Bike Sharing Dataset

We start our exposition with the **Bike Sharing Dataset** [55], which contains the number of bikes used on a Bike Rental service in Washington, D.C., USA on a daily basis between 2011 and 2012 (731 samples), along with weather data for each day. Our aim here is to study the effect of *temperature* (T) on *bike rental* (Y), assuming that the underlying DGP follows the causal graph shown in Fig. 7.1. Here, *season* and *weather* are Categorical variables (4 and 3 levels, respectively); *working day* is a Bernoulli variable; *temperature*, *humidity*, *windspeed* and *feeling temperature* are continuous variables normalized to the $(0, 1)$ interval, and *bikes* is the target variable, a count of rented bikes on a given day (which we assume to be continuous for ease of modelling). We define a latent confounder between *humidity* and *wind speed* to reflect weather factors that affect both but are not captured by the Categorical variable *weather*. We do

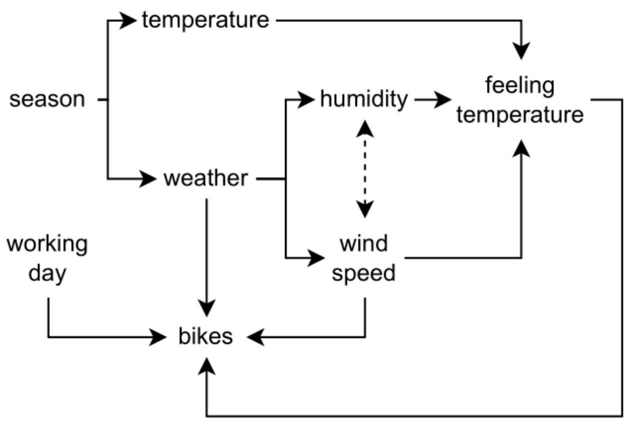


FIGURE 7.1: Bike Sharing Dataset, proposed Causal Graph.

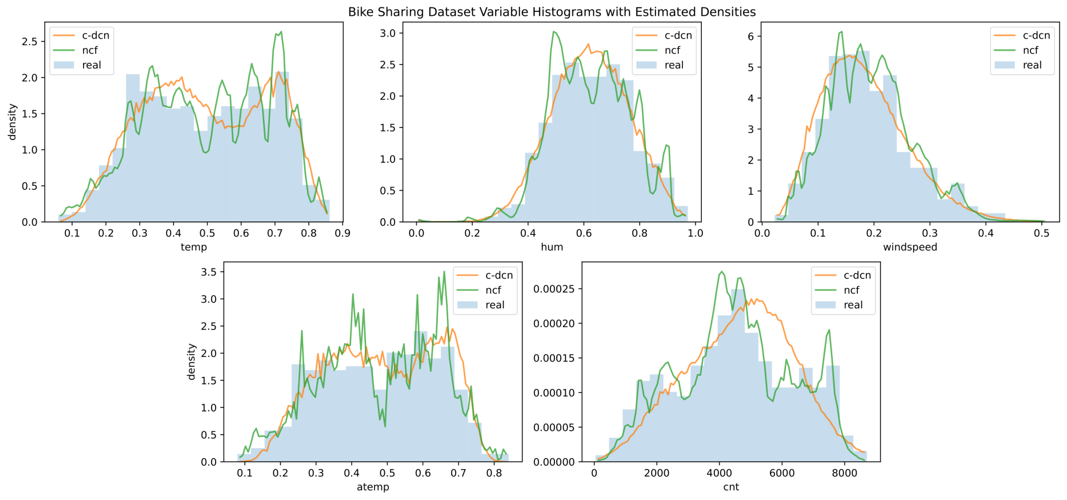


FIGURE 7.2: Bike Sharing Dataset, variable histograms (blue) with DCG densities: Compound-DCN and NCF.

not wish to make any claims regarding the real underlying graph; henceforth, we assume this graph to be valid only for illustrative purposes.

7.1.1 Data Adjustment

First we need to decide which **DCU implementation** will be employed for each node. To illustrate the potential of the different DCUs mentioned in this work, we train two different models, one consisting of Compound-DCNs and one with NCFs.

For the former case, since *temperature*, *humidity*, *wind speed* and *feeling temperature* are all bounded to the $(0, 1)$ interval, we employ the Beta Distribution for their variables. As for *bikes*, the counting variable, we use the Normal Distribution (ND). Note that despite modelling these nodes with a Beta or a Normal distribution, they only follow them when conditioned by all its parent values; when we marginalize those conditioning terms, the result is a mixture of their base distributions, which is far more expressive than first assumed. Not only that, but the Compound DCU allows for further expressivity in their modelling.

Regarding the NCF implementation, we use the same Transformer as in Fig. 5.4, except for the two beginning layers: *bikes* operates in non-negative reals, but we do not restrict its domain; however, every other variable is restricted to the $(0, 1)$ interval, so we start with a Logit flow that transforms it back to all reals.

Fig. 7.2 shows the **dataset histograms** for every continuous variable and the corresponding marginalized **likelihood curves**, evaluated by each DCG. Although both alternatives are quite similar, C-DCNs struggle with *cnt* (count of rented bikes), possibly due to its multi-modality, whereas NCFs can be as flexible as required given a powerful flow architecture. **The choice of DCU depends on the complexity of the data and the number of samples** (the smaller the dataset, the more overfitting introduced by NCFs; in that case, DCNs or C-DCNs are preferable). In the following section, we will evaluate three identifiable queries with the NCF model.

7.1.2 Causal Query Estimation

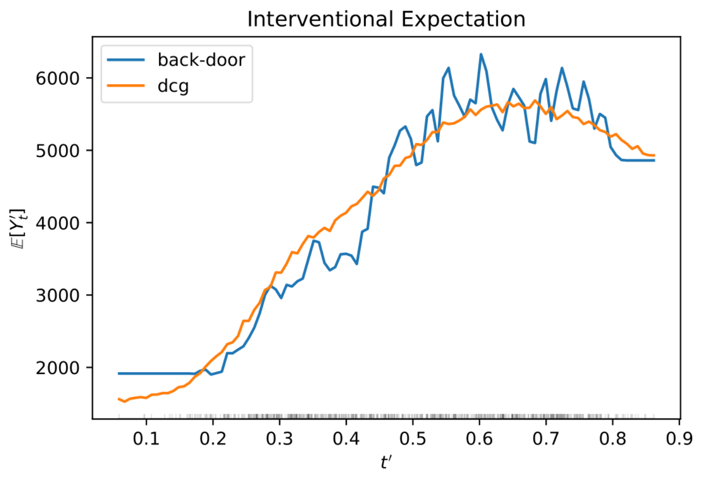


FIGURE 7.3: Bike Sharing Dataset, interventional effect of *temperature* t on *rentals* Y_t, estimated with DCGs and the back-door formula for comparison.

We start with an **interventional query**: the average number of rented bikes when intervening *temperature* by a certain value t. Fig. 7.3 plots the effect (y-axis) of each intervention $do(T = t)$ (x-axis) computed using our DCG model, but also with the *back-door* formula $\mathbb{E}[Y_t] = \mathbb{E}_S \left[\mathbb{E}_{Y|T=t,S}[Y]\right]$ for comparison. For the latter, we use Random Forests to model each term needed to compute the query. As can be seen from the figure, there are no significant differences between the two curves, which shows that, despite following a completely different strategy to the back-door approach, DCGs still manage to answer the query reliably.

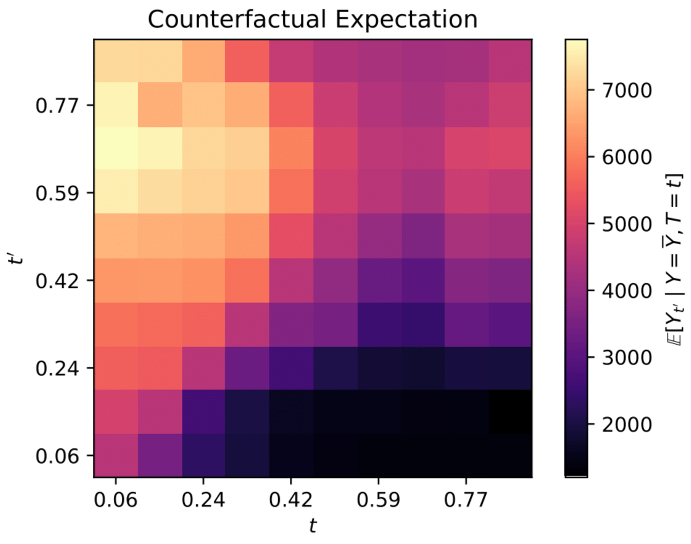


FIGURE 7.4: Bike Sharing Dataset, counterfactual effects of intervened *temperature* t' on *rentals* when we have observed an average number of *rentals* and different values of *temperature* t.

Next, we study a **counterfactual query**: the expected number of rented *bikes* intervened by *temperature* with value $do(T = t')$, when we have already observed a different *temperature* $T = t$ and the rented *bikes* are observed to be their average number ($Y = \overline{Y}$). Fig. 7.4 shows a heatmap of this counterfactual quantity (colour is the counterfactual value) when we have observed a certain t value (x-axis) and intervened on *temperature* with a different t' (y-axis). From the previous query, we know that Y and interventions on T are directly correlated, which tells us that days with higher *temperatures* should expect above-average number of *bikes*, whereas days with lower *temperatures* expect below-average counts. However, a factual average number was observed in both cases; this means that higher values of t start from an average Y, and can only go down as the intervention reduces *temperature*. Conversely, lower values of t start from an average count and go to the highest values as t' increases. In layman's terms, if today was a cold day but we still had an average number of *rentals*, this must mean that it was a busier day than usual, and increasing the *temperature* can only result in higher *rentals*.

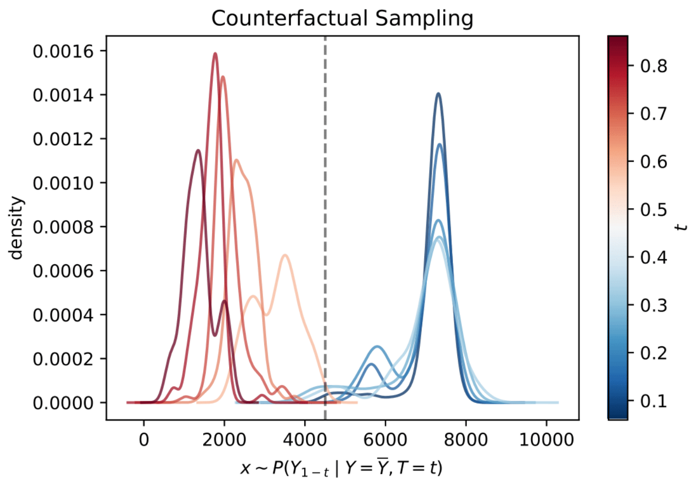


FIGURE 7.5: Bike Sharing Dataset, KDE curves resulting from counterfactual sampling, where we observe *temperature* t and an average number of *rentals*, but we intervene on *temperature* $1 - t$.

Finally, Fig. 7.5 showcases an example of **counterfactual sampling**. Having again observed an average count of *rentals* ($Y = \widetilde{Y}$) and a certain *temperature* t, we intervene by inverting the *temperature* $do(T = 1 - t)$ (remember that T is normalized to $(0, 1)$) and obtain samples from the counterfactual variable Y_{1-t}. We use Kernel Density Estimation (KDE) to plot the density curves from these samples, so we can see all curves at once. t is coded by colour, with lower *temperatures* in blue and higher *temperatures* in red. As we can see, observing low *temperatures* t (blue) means intervening with high *temperatures*, which increases the number of *rentals*, resulting in above-average values (past the dashed vertical line, the average number of *rentals*). Higher *temperatures* (red), on the other hand, are intervened on with lower *temperatures*, which can only result in lower counts.

As stated before, the fact that we can obtain these counterfactual samples instead of only expectations of these distributions allows for extra flexibility in our causal studies; since we are not only restricted to expectation studies, we can, for example, look for multi-modalities or asymmetries in their distributions. This is a significant advantage of DCGs in comparison to estimand-based approaches, and even some other estimand-agnostic models that do not provide abduction mechanisms.

7.2 City of Phoenix Employee Compensation Dataset

Our final experiment explores the **City of Phoenix Employee Compensation Dataset** [56] for the year of 2016, which lists the salaries of public employees along with several variables describing the person and their work situation. We will study this dataset in three steps: firstly, we will analyze the data itself with **interventional and counterfactual queries**, in order to understand the hiring process, especially with regard to salary assignment and gender; secondly, we will measure the **Counterfactual Fairness** of salary towards gender and train a black-box predictor to be counterfactually fairer; finally, we will inspect this black box with **Causal Introspection** techniques, from Feature Importance to Counterfactual Explainability. In the end, we will also measure the effect that this predictor would have on contract termination, so as to evaluate the impact of the new policy.

This dataset consists of 15,026 entries and 11 variables, including *employee name*, *hire date*, *salary*, etc. We perform several **preprocessing** steps:

- We compute the *seniority* of an employee by subtracting *hire date* from *termination date* (or the last day of the year if not terminated).
- We define *salary* as the sum of the regular, overtime and other pay columns.
- We aggregate every *department* and *benefits* values (both categorical) that account for less than 5% of the data as a new category: "Other".
- We determine the employee's *gender* from their name using an automated predictor service [57]. While there is a degree of uncertainty in this prediction of gender, it is the only way for us to exemplify a study of fairness on this dataset.
- We remove any *part-time* employees so as to study the effects of unfairness for full-time positions only. The part-time group constitutes a 10% of the dataset and has a substantially higher ratio of females (47%) than the full-time group (29%), with much lower salaries. While this constitutes a source of unfairness, it is hard to study this subgroup as, given the variability of its salaries, it requires a higher number samples and additional variables to which we do not have access (e.g., number of weekly hours or hiring period length).

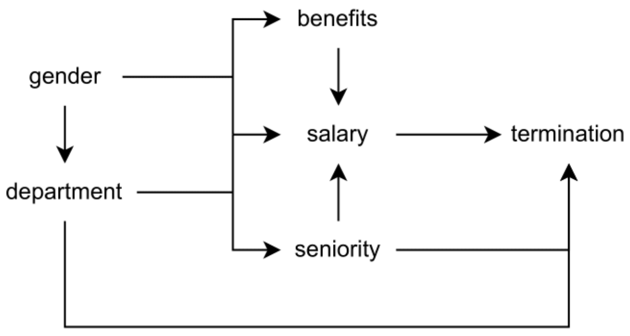


FIGURE 7.6: Phoenix Salary Dataset, causal graph.

- We remove any employees with a *salary* above the 99% percentile; these samples are extreme values belonging primarily to males (87%), which also explains a considerable source of unfairness. However, if we included these values, it would bias the numerical estimations, so we omit them.

As a result of this preprocessing step, the dataset consists of 13,139 samples with 6 variables: *gender* (binary), *department*, *benefits* (type of indirect work pay perceived by the employee), *salary*, *seniority* and *termination* (whether the employee's contract has been terminated during the year). Descriptive statistics of the resulting dataset are included in Table 7.1. We divide the dataset in three random subsets -train, validation and test- with 70%, 15% and 15% of the samples, respectively; only training and validation will be used for the graph and predictor training.

See Fig. 7.6 for the proposed **causal graph**. We operate on the assumption that *gender* affects the choice of *department* due to societal pressure, while both affect *benefits*, *seniority* and *salary*. On the other hand, *department*, *seniority* and *salary* affect the eventual *termination* of the contract.

We acknowledge several **limitations** in the choice of variables and graph. ***Gender*** is a flawed variable since we do not have access to it directly, forcing us to extrapolate it from the employee's name; more so, since the gender predictor only considers binary gender, it does not account for non-binary genders. ***Department* and *benefits*** are aggregated when their values account for less than 5% of the dataset, as stated before, which semantically is not always appropriate for these categories. *Salary* assignment and contract *termination* depend on the necessities of each *department* and the allocated budget, which would create a **feedback loop** in this Causal Graph for other to-be-hired employees, something that we cannot account for with DCGs at the moment. We also omit any **latent confounders**, as we do not see any relevant relationship in this structure that would require them; nonetheless, were there any, DCGs can model them without limitations to their capabilities, as long as the queries to be estimated are identifiable. We choose this Causal Graph attaining to the limitations present in the data and our own judgement of how these variables interact with one another. However, as with every other causal study, the graph is subject to revision, and subsequently the results derived from it.

	Dataset Mean	Dataset S.E.	DCG Mean	DCG S.E.
P(Gender=Male)	0.709	0.004	0.706	0.004
P(Termination)	0.057	0.002	0.053	0.002
Department[Aviation]	0.063	0.002	0.061	0.002
Department[Fire Dep.]	0.151	0.003	0.154	0.003
Department[Police Dep.]	0.304	0.004	0.289	0.004
Department[Public Works]	0.074	0.002	0.073	0.002
Department[Water Services]	0.104	0.003	0.098	0.003
Department[Other]	0.305	0.004	0.325	0.004
Benefits[1]	0.075	0.002	0.079	0.002
Benefits[2]	0.124	0.003	0.118	0.003
Benefits[3]	0.175	0.003	0.176	0.003
Benefits[4]	0.193	0.003	0.179	0.003
Benefits[5]	0.122	0.003	0.125	0.003
Benefits[7]	0.244	0.004	0.264	0.004
Benefits[Other]	0.067	0.002	0.059	0.002
Seniority (days)	4,799	27.7	4,728	27.561
Salary ($)	66,660	263.244	65,220	267.531

TABLE 7.1: Phoenix Salary Dataset, dataset statistics per variable vs. estimated statistics from DCG's samples.

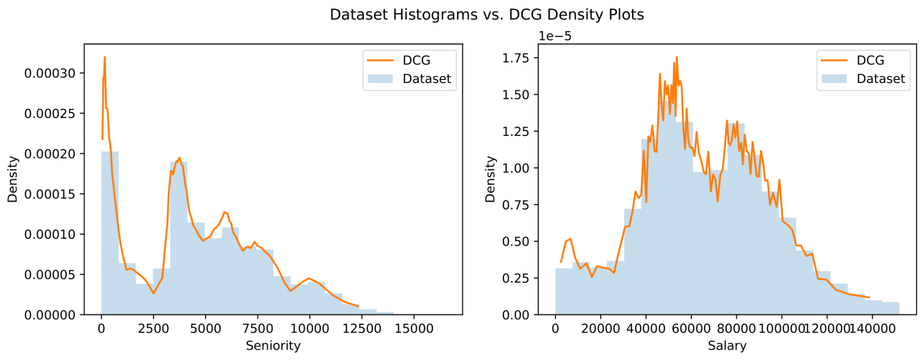


FIGURE 7.7: Phoenix Salary Dataset, continuous variable histograms vs. estimated densities using DCGs.

7.2.1 Data Adjustment

Before we perform a Causal Analysis, we need to train the DCG model to adjust to the dataset's distribution. We begin by demonstrating the adjustment of the model to the target distribution. Table 7.1 lists the mean and standard error (S.E.) for every variable in the dataset across $N = 13,139$ samples, along with the estimated means and S.E. computed from N DCG-generated samples. Additionally, we plot the histograms of the two continuous variables, seniority and salary, together with their DCG-estimated densities; see Fig. 7.7. Both of these results show a good adjustment to the underlying distribution.

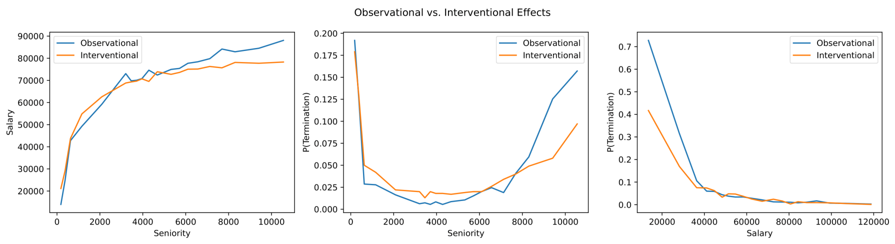


FIGURE 7.8: Phoenix Salary Dataset, observational vs. interventional effects for several continuous variables.

7.2.2 Causal Analysis

Interventional effects

We start by measuring **interventional causal effects** between pairs of variables, the first of which would be the effect of intervening *gender* on every other variable. We compute the effect on the target variable Y (*seniority*, *salary* or $P(termination)$) when intervening *gender* to males ($do(G = 1)$) or to females ($do(G = 0)$), with G, *gender*, as an indicator of male. We compare the difference in expectation between the intervened population of males against females, $\mathbb{E}[Y_{G=1}] - \mathbb{E}[Y_{G=0}]$; the results are 290 days, \$11,734 and -1% (1% less likely of being terminated if male), respectively. Note that especially the first two results denote a certain bias against women, which will reappear throughout the next estimations.

We will not compute the effect of categorical variables (*department* and *benefits*) since their levels do not provide enough meaningful information. However, if we were interested in them, we would simply need to intervene with every possible level of the variable, as with the Bernoulli case, and compare between them.

Finally, for continuous variables X, let us define K evenly-spaced terms between 0 and 1 (extremes excluded), $u := (u_i)_{i=1..K}$, $u_i := \frac{i}{K+1}$, and compute the corresponding u-quantiles for the intervening variable, $(q_i)_{i=1..K}$, $q_i := CDF^{-1}(u_i)$. We can then estimate the observational and interventional effects, $\overline{y}_i := \mathbb{E}[Y \mid X = q_i]$ and $\widehat{y}_i := \mathbb{E}[Y \mid do(X = q_i)]$, respectively, both using DCG procedures. We plot the pairs $(q_i, \overline{y}_i)$ and $(q_i, \widehat{y}_i)$ in Fig. 7.8, to compare the effects of conditioning and intervening X on Y. The ticks on the x-axis represent distribution density on the intervening variable.

When interpreting these effects, we see several trends that are consistent with our intuition: *salary* and *seniority* are directly proportional, with a slower slope as *seniority* increases; *seniority* reduces the likelihood of *termination* at first, but after a certain point around 8,000 days (more than 20 years), we see the trend increase again, likely due to retirement; regarding the last plot, we see that *salary* and the probability of *termination* are inversely proportional, with it being much more likely on lower *salaries* (less than \$40,000), either because these individuals look for alternative job offers, or because management finds it easier to let them go. This latter trend in particular is important, as it will inform some of the counterfactual analyses carried out later.

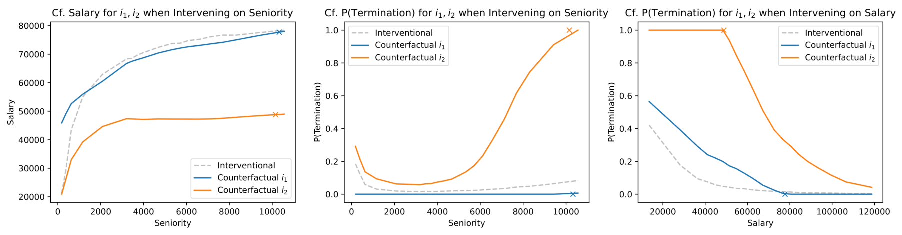


FIGURE 7.9: Phoenix Salary Dataset, counterfactual effects for individuals i_1 and i_2. The interventional effect for the whole population is included for comparison. The coloured x-marks show the individuals' observed values for both axes/variables.

Finally, we see no remarkable differences between the interventional and observational effects; while they are technically not the same, in practice they can be quite similar in some datasets. Nevertheless, one should never take observational estimations as proxies for causal queries, as they can considerably diverge depending on the problem.

Counterfactual effects

The former computations all relate to interventional expectations, queries that describe the behaviour of the whole population when submitted to a certain intervention. Now, we focus on **counterfactual queries**, that can be understood as focused on an individual of whom we know some *factual* information; we want to estimate what effect a certain intervention would have had on this particular person. For the remainder of the experiment, we will consider two individuals, $i_1 = 7,071$ and $i_2 = 5,352$ (their 0-indexed position in the dataset), chosen according to certain requirements. Both are women with *salaries* between the 25th and 75th quantiles, one whose contract has not been *terminated*, the other that has, respectively. With these restrictions, we choose the woman with the highest gain in *salary* had they been males; in other words, we compute the counterfactual expectation of *salary* given an intervention of *gender* for every individual in these subsets and select the one with the highest difference between the counterfactual and factual *salaries*. We are interested in these individuals so as to measure how they are affected by the **unfairness** of the *salary* mechanism, but also to see how the other variables affect that decision.

In terms of the **counterfactual *gender* effect**, i_1 has a factual *salary* of \$77,740 and a counterfactual *salary* of \$128,524 (a differential of \$50,785), while i_2 has a factual *salary* of \$48,817 and a counterfactual *salary* of \$93,728 (a differential of \$44,911). We are **not restricted to binary interventions**; we can also compute the counterfactual effects with the aforementioned intervening quantiles $(q_i)_i$ and plot them as before on Fig. 7.9; here we also include the previous interventional effects to compare how each individual differs from the population as a whole. The x-marks represent the observed (factual) data for these individuals along the two axes.

As we can see, there is a remarkable difference between both counterfactuals when intervening on each continuous variable. i_1 is a more prototypical individual *w.r.t.* the whole population (the interventional curve) for all three plots, while i_2 behaves quite differently: even though *seniority* does have an effect on her *salary*, it is still remarkably lower than the average trend, and the likelihood of *termination* is also quite high. Informed by the factual information that this individual did indeed terminate their employment, the counterfactual reflects this with higher likelihoods of *termination* and gives us additional information: the individual was more likely to terminate the contract as *seniority* increased, and the factual value of *seniority* was indeed quite high, which suggests that this person might have been close to retirement. On the other hand, *salary* also had an effect on this decision, as the plots tells us that if this person had had a higher *salary*, the likelihood of *termination* would have progressively decreased, but lower *salaries* would not have changed that decision, as expected. In summary, these counterfactual queries help us ascertain **the causes behind a certain decision, focused on an individual instead of the population as a whole**. This relates directly to Black-Box Introspection, understanding the natural processes through which this dataset was created as a black box.

7.2.3 Counterfactual Fairness

In this section we will measure the unfairness inherent in the observed salary assignment mechanism. Consider Counterfactual Fairness [15], defined as the stipulation that the factual and counterfactual distributions remain identical when intervening on a protected variable (in our case, *gender*). In order to measure the deviation from this objective, for a Bernoulli intervened variable, we define a metric called **Counterfactual Unfairness** (CU) of degree k, CU_k, as:

$$CU_k := \mathbb{E}_{\mathcal{V}}\left[\mathbb{E}_{\mathcal{E},\mathcal{U}|\mathcal{V}}\left[|Y_{1-x}(\mathcal{E},\mathcal{U}) - Y(\mathcal{E},\mathcal{U})|^k\right]\right], \tag{7.1}$$

where the expectation is across all measured variables $\mathcal{V}$ in the Causal Graph, $\mathcal{E}$ and $\mathcal{U}$ are the SCM's latent variables, X is the intervened variable with x the factual value and $1 - x$ the intervened value, and Y, the target variable. Note that Y acts as a deterministic function with inputs $\mathcal{E}$ and $\mathcal{U}$, given that the deterministic functions in the SCM provide a sampling procedure that transforms stochastic samples from $\mathcal{E},\mathcal{U}$ into samples for every variable in $\mathcal{V}$. As an example, CU of degree 1 estimates the average absolute difference between the factual result of Y and its counterfactual value Y_{1-x} where we intervene variable X by taking the opposite value $1 - x$. In our use case, X is the indicator for male, so we measure the average absolute difference between the factual outcome (the individual's *gender*) and the counterfactual outcome (intervened with the opposite *gender*). If the system was Counterfactually Fair, then $CU_k = 0 \ \forall k$, so we can consider CU as a loss metric representing unfairness.

Both expectations are estimated using Monte Carlo. The first one, by averaging across every sample in the dataset $\mathcal{D} = \{\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N)}\}$, described by variables $\mathcal{V}$;

the second one, by abducting the exogenous noise signals $\mathcal{E}$ and latent confounders $\mathcal{U}$ from each factual configuration $\mathbf{v}$. In any case, the `dcg` library takes care of this process to compute the desired expectation. The result on this dataset is $CU_1 = \$16,723$, which lets us gauge how much of an effect *gender* has on *salary* assignment.

CU can also be employed as a regularization term, and that allows us to **train fairer predictors** using an adequate weighting value. Let us assume we want to replace the original salary assignment policy by **a black-box predictor of salary** (with $\mathbf{V} := \{gender, department, benefits, seniority\}$ as input variables) **that mitigates this unfairness** in its estimations; this will be called f, the fairness-aware predictor. Additionally, we will train another model, the fairness-unaware predictor f_0, which serves as a point of comparison in how the fairness-aware training of f diverges from the estimations of f_0. Both models are defined with the same generic Neural Network architecture and training procedure, with the only difference that f also has CU_2 as a regularization term, directing the learning process towards fairer decisions. Please refer to the associated code for more details.

In this case, we want to **balance predictive performance with fairness** (the fairest predictor possible would simply assign a constant *salary* to every individual); we train f with five possible values for the regularization weight hyperparameter λ and compare the **regression metrics** resulting from each value (including $\lambda = 0$, which represents the fairness-unaware predictor f_0). See Table 7.2 and Fig. 7.10 for the relation between the regularization weight and every regression metric. We compute the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) on the estimation of *salary*, the R^2 regression coefficient and also the Spearman correlation between the original and predicted salaries, amongst the whole population and for each demographic group in isolation. We use the Spearman correlation as a way to measure how much the ordering of the predicted salaries differs from the original ones. We also include two fairness metrics, CU_1 and $\sqrt{CU_2}$.

Note that adding this regularization term to the network training makes the predictor learn from a different distribution than the one that created the dataset, one that is fairer; because of that, it is natural for some metrics to worsen, since we are indeed changing the real *salary* for a different one to make it fairer. It is for this reason that we also study the Spearman correlation between males and females; it allows us to measure the *ordering* in which they are assigned salaries, and we would not expect differences in ranking inside each group when we are training the model to reduce the gender gap. Therefore, this metric should be more stable than the others to measure predictive performance. With this in mind, we select $\lambda = 1$ as the regularization weight, as it achieves a **considerable decrease in unfairness** (more than half the f_0's and the dataset's values) while still **preserving good results for the predictive metrics** in comparison with f_0 (specially on Spearman correlations).

TABLE 7.2: Phoenix Salary Dataset, metrics for each CU_2-regularized predictor. λ is the regularizing weight and $\lambda = 0$ denotes the fairness-unaware model f_0. Lower is better for RMSE, MAE, CU_1 and $\sqrt{CU_2}$; higher is better for R^2 and every Spearman Correlation (Sp.) metric (Global, Female and Male, respectively). We highlight the rows pertaining to f_0 ($\lambda = 0$), the fairness-unaware predictor, and the chosen fairer predictor f ($\lambda = 1$).

λ	RMSE (↓)	MAE (↓)	R^2 (↑)	Sp. (↑)	Sp. F. (↑)	Sp. M. (↑)	CU_1 (↓)	$\sqrt{CU_2}$ (↓)
0.0	**20,383**	**15,218**	**0.53**	**0.76**	**0.66**	**0.78**	**16,867**	**22,067**
0.1	20,382	15,063	0.53	0.75	0.67	0.78	14,838	20,116
0.5	21,724	16,426	0.47	0.73	0.62	0.77	9,402	13,266
1.0	**23,282**	**17,949**	**0.39**	**0.71**	**0.60**	**0.76**	**6,994**	**9,952**
5.0	30,863	24,435	-0.07	0.65	0.55	0.70	2,925	4,472
10.0	40,148	33,192	-0.81	0.60	0.52	0.65	1,866	2,822

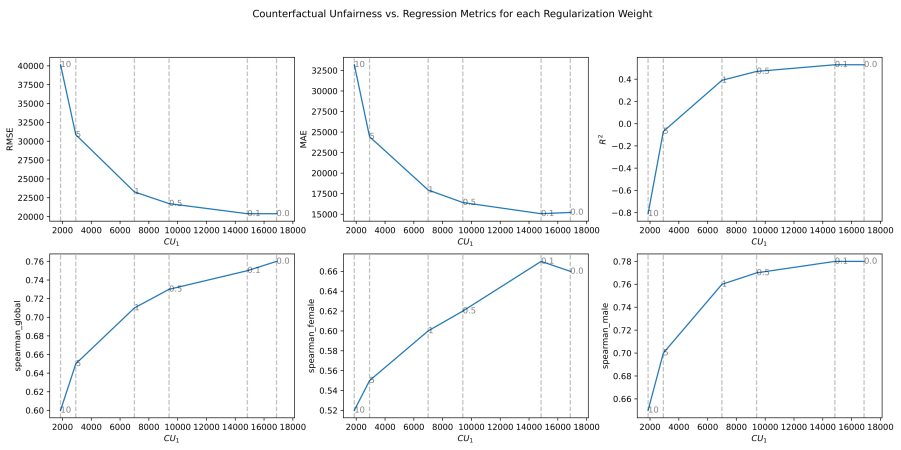


FIGURE 7.10: Phoenix Salary Dataset, counterfactual unfairness vs. regression metrics for every regularization weight λ. Lower is better for RMSE, MAE and CU_1; higher is better for the rest.

7.2.4 Black-Box Introspection

In this final section, we want to explain the *salaries* predicted by these new policies, so we will perform **Black-Box Introspection** with the two models trained before; this can be carried out at the population level, with interventional effects, or at the individual level, with counterfactual effects. Note that we can consider a modified Structural Causal Model $\mathcal{M}_{f_0}$ and $\mathcal{M}_f$ where we replace the original *salary* node (representing the variable measured in the dataset) with the Machine Learning models. In this new graph, the altered *salary* variable would have as parents every input variable in the predictor, as their values affect its outcome; in our case study, that is every parent of the original variable. Black-Box Introspection can be carried out by studying the causal effects of these altered SCMs.

We start with the **interventional effects** of *gender* and *seniority* on the predictor's *salary*. The fairness-unaware predictor f_0 results in a difference of \$10,419 (close

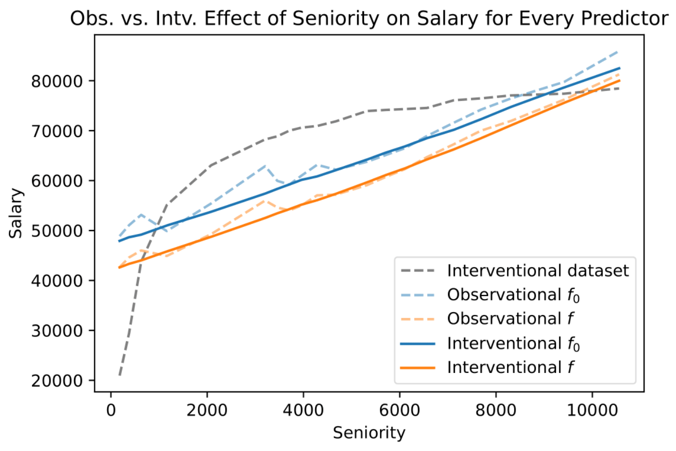


FIGURE 7.11: Phoenix Salary Dataset, observational vs. interventional effect of *seniority* on each predictor's salary. We include the interventional effect of the original process for comparison.

to the original *gender* effect on the natural *salary* mechanism), while f reduces it to $2,163, a remarkable improvement. Regarding the effect of *seniority* on *salary*, see Fig. 7.11; here we compare the observational and interventional effects as before, but for each predictor. We include the dataset's interventional effect to compare against the predictor curves. We can see a linear effect, something that does not fit the natural process, which had a decelerating curve; this discrepancy happens both for f and f_0, which might mean that the input variables had not enough informative data to model that trend more faithfully. In any case, from the perspective of Introspection, we can see the difference in effect that seniority has between both predictors, with f leading to lower salaries overall (compensating for the higher men salaries) despite having the same slope.

Regarding **counterfactual effects**, we can measure the effect of *gender* on each predictor for both individuals i_1 and i_2; in other words, what their predicted *salaries* would be had they been males. We compute the differential between the counterfactual and factual estimates for both predictors and individuals; for i_1, the differentials are $39,609 and $6,343 for f_0 and f, respectively; for i_2, $39724,74 and $6,351. As expected, we see a remarkable reduction in gender gap for these individuals.

Finally, we would like to ascertain if the new policy could have a negative effect on contract *termination*. In order to do that, we need to intervene *salary* (Y) with the f-predicted *salary* for each individual, and then sample from the *termination* (T) variable as usual, so as to estimate its ratio: $\mathbb{E}_{\mathcal{V}}\left[T \mid do(Y = f(\mathcal{V}))\right]$. We can compare the *termination* rate from the dataset (a simple average) and the termination rate resulting from the policy change: 5.7% and 2.5%, respectively. In this case, the fairness-aware predictor also results in less than half *terminations*, showing that fairer salaries, in this case, would actually discourage contract termination.

Conclusion

Motivation

Over the last few years, **Causal Theory** has progressively been adopted in many different fields to study the effect of *interventions*, to understand the causes behind certain decisions or to consider alternative outcomes depending on our actions. Examples of such studies can be found in medicine (determine the recovery rate when subject to a certain treatment), social sciences (measure the impact of policies) or criminology (uncovering the main causes behind recidivism), to name a few. This kind of questions can sometimes be answered with randomized experiments, but depending on the use case it may be expensive, unethical or ultimately unfeasible. Causal Theory provides tools to circumvent this problem by employing passively-obtained (observational) datasets and some form of **estimator**.

The main object of study is the **causal query**, a probabilistic expression describing the behaviour of random variables subject to interventions. This results in expressions that are purely observational, interventional or counterfactual in nature; all can be answered through the lens of causality, given a causal graph describing these variables and the relationships between them. This graph consists of asymmetrical cause-effect relationships between pairs of variables, and allows us to determine if a certain causal query can be estimated or not (**identifiability**) and which form of *estimand* we need to answer it.

An **estimand** is a formula transforming causal terms into observational terms, resulting in probabilistic expressions that can be computed from observational data. This estimand can be derived through automatic algorithms, if it exists, or determined not to exist, in which case the query is said to be non-identifiable and therefore unable to be estimated unless we impose additional restrictions on the assumed underlying process that governs the data. Given an identifiable query and the corresponding estimand, the **estimand-based** approach to Causal Query Estimation employs Machine Learning tools to model some of the observational terms in the estimand, and computes an answer to the formula with these estimated terms.

This is the default approach to the problem, but it comes with a number of **shortcomings**. Firstly, each estimand depends on the query and the causal graph in which it is defined, sometimes resulting in non-trivial expressions, requiring a complex case-by-case definition of the estimator procedure to answer the query. Consequently, the literature on estimand-based causal estimation presents practitioners with a wide range of different approaches, each specialized on a particular kind of graph and query; this hyper-specificity makes these methods extremely **ad hoc**, hard to apply to more complex scenarios. Secondly, were we to ask a second query on the same dataset, the resulting estimand might require a completely different model to answer it, forcing practitioners to start from scratch. Therefore, every new query, despite it concerning the same dataset, can result in training new models; use cases that require an iterative interrogation of the causal mechanism, answering many different queries as we study the system, suffer from this shortcoming especially.

Alternatively, we can employ **Structural Causal Models** (SCMs) to learn the observational distribution, but subject to certain structural constraints imposed by the dataset's causal graph. These models can be used as proxies of the underlying mechanism that generated this data, as long as they follow the same causal graph and learn the observational distribution $P(\mathcal{V})$. Given estimation procedures defined directly on these models, we can define estimators that completely circumvent estimands. We call this kind of techniques **estimand-agnostic** approaches, because they do not follow an estimand to perform estimation; rather, they learn from the target distribution once and then can estimate any causal query, provided it is identifiable and we have appropriate estimation procedures to estimate it.

The main difficulty concerning these approaches is how to design models capable of modelling real world distributions while following the desired causal structure, but also providing estimation procedures for many kinds of causal queries. The structural requirements forbid us to employ most density estimation methods, as they usually learn the joint probability distribution with no independence restrictions between its dimensions. On the other hand, even if we manage to mirror the target distribution perfectly along with the structural restrictions, we still need this model to provide estimation procedures so as to answer the desired causal queries, with counterfactuals being the primary example of difficult queries to solve. The estimand-agnostic perspective has been **hindered** by these difficulties until recent years, when the advances in Deep Learning were adopted to the field of Causal Query Estimation.

The current literature, however, has a number of shortcomings that we mean to address with our technique. We identify the **desiderata** for a **practitioner-ready, Deep-Learning-powered estimand-agnostic framework**: provide tools to compute likelihoods of its variables; account for the existence of latent confounders in the graph; allow for counterfactual estimation through the abduction mechanism; provide expressive modelling to account for complex real-world distributions; be scalable to the number of nodes in the graph; and be applicable to arbitrary causal graphs in its training and estimation procedures. We identify several estimand-agnostic approaches proposed in recent years, determine its shortcomings with respect to this desiderata and provide a **novel approach** that encompasses each of these requirements.

Contributions

We propose **Deep Causal Graphs** (DCGs), an SCM-based abstract framework that allows for estimand-agnostic estimation of many kinds of identifiable causal queries (including counterfactuals), all through general procedures applicable to any causal graph that can be represented with a Directed Acyclic Graph, even in the presence of latent confounders. It consists of submodules called **Deep Causal Units** (DCUs), one for each node in the graph, that are meant to model their conditional distribution while still providing functionality for three distinct operations: sampling, computing likelihoods and abducting noise signals. When a differentiable model accounts for these three operations, it can be employed as a DCU inside a DCG.

The DCU is an abstract specification, but it admits many **expressive implementations based on Deep Learning** models. We present **Distributional Causal Nodes**, representing probability distributions through a neural network that outputs its parameters, and **Normalizing Causal Flows**, employing Conditional Normalizing Flows as a node modelling mechanism for continuous variables. These two implementations can be further extended through the use of **Mixture DCUs**, modelling mixtures of DCU submodules as a new type of DCU, and **Compound DCUs**, which allow for unmeasurable mixtures of DCUs through the use of an additional noise signal. We propose these four implementations for the DCU specification, allowing to model complex real-world distributions through the use of Neural Networks. We demonstrate the expressiveness of these models through several experiments described in this work.

Once we have defined our DCU implementations, we can discuss the overall DCG model. We derive the **training mechanism** for the overall model and several **estimation procedures**, all based on the three DCU operations, applicable to many kinds of **observational, interventional and counterfactual queries**. These procedures are **defined for arbitrary graphs**, as long as their corresponding queries are identifiable in the causal graph of study, which allows for an easy application in many different settings once the model has been trained, in contrast with the ad hoc nature of estimand-based techniques. Additionally, we define the **shared Graphical Conditioner**, a simple technique that allows any arbitrary feed-forward Neural Network to be employed as the sole network for all DCU nodes in a graph, thereby guaranteeing scalability of the model in terms of memory requirements and training times with respect to the number of nodes in the graph.

Nonetheless, these contributions come with some **limitations**, all **shared with other estimand-agnostic approaches**. Firstly, the main assumption is that the underlying causal graph that generated the dataset of study is known, which is not always the case. For this reason, any causal estimations are contingent on the adequacy of the graph. More so, the graph is assumed to be a Directed Acyclic Graph; any cyclic causal relationships between variables are discarded in the current framework, which forbids its use on use cases with this requirement. Secondly, the exact estimation of an identifiable causal query through an SCM is only ensured if the model represents the exact same observational distribution $P(\mathcal{V})$ that describes the dataset. Given that stochastic training can only aim to approximate the target distribution, the mismatch between the learned and real distributions can have an effect on our estimations; furthermore, in graphs with high depth, this mismatch could result in compounding errors affecting the eventual quality of our estimators. Finally, the fact that estimand-agnostic techniques are not trained with one particular query in mind means that these models cannot be optimized towards more precise estimators of that particular query, while estimand-based techniques can do so by their ad hoc nature. These shortcomings are left for further research, to extend the applicability of our techniques.

Finally, despite the complexity of the framework and all its subsystems and procedures, we created a **complete open-source software library** based on the PyTorch Deep Learning library that allows for the use of each of these techniques by practitioners while being flexible enough for researchers to extend with further implementations of the abstract DCU and DCG specification. The goal of this library is the democratization of our techniques, as a way to promote the use and further study of estimand-agnostic approaches. We also include a complete use case in this work describing the application of the library to an example dataset, from DCG creation and training to causal query estimation employing DCG procedures.

Experiments and Applications

In order to **demonstrate the capabilities of our technique**, we perform several **experiments** with synthetic, semi-synthetic and real-world datasets. We test its modelling performance by learning the observational distribution of these datasets and comparing the dataset's histograms with DCG samples and density curves estimated with DCGs. By choosing the appropriate DCU architecture and training parameters, we observe good adjustment to these distributions, specially through the use of Normalizing Causal Flows or Compound Distributional Causal Nodes, depending on dataset size and distribution complexity.

We evaluate **estimation performance** with the semi-synthetic IHDP-Jobs potential outcomes estimation benchmark. Despite using an estimand-agnostic approach, and therefore unable to optimize the model to specifically obtain the best performance for a particular query, **DCGs achieve competitive results with the state of the art**. Moreover, given the highly modular nature of DCGs, alternative implementations of DCUs could be employed to improve these results further.

We finish this work with two case studies on real world datasets, with which we showcase the **breadth of causal queries that can be estimated using a single DCG model**. We also demonstrate the **connection between Causal Query Estimation and Black-Box introspection**: no matter if the black-box is a Machine Learning model or the data itself, we can interrogate the system with several causal queries so as to understand the underlying mechanism governing it, determining the causal effects of its variables and contrasting scenarios with interventional and counterfactual queries at the population and individual levels, respectively.

Given this connection, the applicability of Causal Estimation and DCGs in particular to **Machine Learning Interpretability, Explainability and Fairness** is clear. We exemplify this application with the last experiment, in a setting detailing gender biases in the workplace. Given a mechanism to **measure Counterfactual Fairness**, we can even **train black-box Machine Learning predictors towards counterfactually fairer distributions**.

In conclusion, we propose **Deep Causal Graphs**, a general, modular, estimand-agnostic framework for Causal Query Estimation that allows for the estimation of many kinds of causal queries, all with the training of a single model capable of learning complex real-world distributions through its implementations leveraging the modelling capabilities of Neural Networks and Normalizing Flows. We detail many possible implementations of the base specification, each useful in different settings, along with several estimation procedures that cover a wide range of queries. We include a complete software library implementing these techniques and provide several experiments showcasing the framework's capabilities. We finish with an application of the technique to Black-Box Introspection, covering Machine Learning Interpretability, Explainability and Counterfactual Fairness.

Bibliography

[1] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, second edition, 2009.

[2] Donald B Rubin. Causal inference using potential outcomes: design, modeling, decisions. *Journal of the American Statistical Association*, 100(469):322–331, 2005.

[3] Ilya Shpitser and Judea Pearl. Identification of joint interventional distributions in recursive semi-Markovian causal models. In *Proceedings of 21st National Conference on Artificial Intelligence (AAAI)*, pages 1219–1226, Boston, MA, USA, 2006.

[4] Ilya Shpitser and Judea Pearl. Identification of conditional interventional distributions. In *Proceedings of the 22th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 437–444, Cambridge, MA, USA, 2006.

[5] Ilya Shpitser and Judea Pearl. What counterfactuals can be tested. In *Proceedings of the 23th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 352–359, Vancouver, Canada, 2007.

[6] Santtu Tikka and Juha Karvanen. Identifying causal effects with the R package causaleffect. *Journal of Statistical Software*, 76(12), 2017.

[7] Martí Pedemonte, Jordi Vitrià, and Álvaro Parafita. Algorithmic causal effect identification with causaleffect. *arXiv preprint arXiv:2107.04632*, 2021.

[8] Fredrik Johansson, Uri Shalit, and David Sontag. Learning representations for counterfactual inference. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 3020–3029, New York, NY, USA, 2016. PMLR.

[9] Uri Shalit, Fredrik D Johansson, and David Sontag. Estimating individual treatment effect: generalization bounds and algorithms. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3076–3085, Sydney, Australia, 2017. PMLR.

[10] Liuyi Yao, Sheng Li, Yaliang Li, Mengdi Huai, Jing Gao, and Aidong Zhang. ACE: adaptively similarity-preserved representation learning for individual treatment effect estimation. In *Proceedings of the 2019 IEEE International Conference on Data Mining (ICDM)*, pages 1432–1437, Beijing, China, 2019. IEEE.

[11] Liuyi Yao, Yaliang Li, Sheng Li, Mengdi Huai, Jing Gao, and Aidong Zhang. SCI: subspace learning based counterfactual inference for individual treatment effect estimation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, volume 30, pages 3583–3587, Gold Coast, Australia, 2021.

[12] Qian Li, Zhichao Wang, Shaowu Liu, Gang Li, and Guandong Xu. Causal optimal transport for treatment effect estimation. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[13] Jin Tian and Judea Pearl. On the testable implications of causal models with hidden variables. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 519–527, Edmonton, Canada, 2002.

[14] Sahil Verma, John Dickerson, and Keegan Hines. Counterfactual explanations for machine learning: a review. *arXiv preprint arXiv:2010.10596*, 2020.

[15] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, Long Beach, CA, USA, 2017.

[16] Peter Spirtes and Kun Zhang. Causal discovery and inference: concepts and recent methodological advances. In *Applied informatics*, volume 3. SpringerOpen, 2016.

[17] Kevin Xia, Kai-Zhan Lee, Yoshua Bengio, and Elias Bareinboim. The causal-neural connection: expressiveness, learnability, and inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 10823–10836, 2021.

[18] Sewall Wright. Correlation and causation. *Journal of Agricultural Research*, 1921.

[19] Murat Kocaoglu, Christopher Snyder, Alexandros G. Dimakis, and Sriram Vishwanath. CausalGAN: learning causal implicit generative models with adversarial training. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, Vancouver, Canada, 2018.

[20] Christos Louizos, Uri Shalit, Joris M Mooij, David Sontag, Richard Zemel, and Max Welling. Causal effect inference with deep latent-variable models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, Long Beach, CA, USA, 2017.

[21] Pablo Sánchez-Martın, Miriam Rateike, and Isabel Valera. VACA: designing Variational Graph Autoencoders for causal queries. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, volume 36, 2022.

[22] Nick Pawlowski, Daniel Coelho de Castro, and Ben Glocker. Deep Structural Causal Models for tractable counterfactual inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

[23] Álvaro Parafita and Jordi Vitrià. Explaining visual models by causal attribution. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 4167–4175, Seoul, Korea, 2019. IEEE.

[24] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Penguin Books, 2019.

[25] European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), 2016.

[26] Marco Ancona, Enea Ceolini, Cengiz Oztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for Deep Neural Networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[27] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 1135–1144, San Francisco, CA, USA, 2016.

[28] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: automated decisions and the GDPR. *Harvard Journal of Law & Technology*, 31:841, 2017.

[29] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. Explaining Machine Learning classifiers through diverse counterfactual explanations. In *Proceedings of the 3rd ACM Conference on Fairness, Accountability, and Transparency (FAT*)*, pages 607–617, Barcelona, Spain, 2020.

[30] Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. Generating counterfactual explanations with Natural Language. In *ICML Workshop on Human Interpretability in Machine Learning*, pages 95–98, Stockholm, Sweden, 2018. PMLR.

[31] Yash Goyal, Ziyan Wu, Jan Ernst, Dhruv Batra, Devi Parikh, and Stefan Lee. Counterfactual visual explanations. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2376–2384, Long Beach, CA, USA, 2019. PMLR.

[32] Riccardo Guidotti, Anna Monreale, Fosca Giannotti, Dino Pedreschi, Salvatore Ruggieri, and Franco Turini. Factual and counterfactual explanations for black box decision making. *IEEE Intelligent Systems*, 34(6):14–23, 2019.

[33] Álvaro Parafita and Jordi Vitrià. Deep Causal Graphs for causal inference, black-box explainability and fairness. *Artificial Intelligence Research and Development*, 339:415–424, 2021.

[34] Álvaro Parafita and Jordi Vitrià. Estimand-agnostic causal query estimation with Deep Causal Graphs. *IEEE Access*, 10:71370–71386, 2022.

[35] Álvaro Parafita and Jordi Vitrià. A unified framework for causal analysis, explainability and fairness. *Submitted*.

[36] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding Convolutional Networks. In *Proceedings of the 13th European Conference on Computer Vision (ECCV)*, pages 818–833, Zurich, Switzerland, 2014.

[37] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3319–3328, Sydney, Australia, 2017. PMLR.

[38] Marco Ancona, Enea Ceolini, Cengiz Oztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for Deep Neural Networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, Vancouver, Canada, 2018.

[39] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA, 2019.

[40] Chris Burgess and Hyunjik Kim. 3D Shapes Dataset. https://github.com/deepmind/3dshapes-dataset/, 2018.

[41] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, Banff, Canada, 2014.

[42] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'Aurelio Ranzato. Fader Networks: Manipulating images by sliding attributes. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5967–5976. Long Beach, CA, USA, 2017.

[43] Z. He, W. Zuo, M. Kan, S. Shan, S. Shan, and X. Chen. AttGAN: facial attribute editing by only changing what you want. *IEEE Transactions on Image Processing*, 2019.

[44] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, Montréal, Canada, 2014.

[45] Iris AM Huijben, Wouter Kool, Max Benedikt Paulus, and Ruud JG Van Sloun. A review of the Gumbel-max trick and its extensions for discrete stochasticity in Machine Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[46] Chris J Maddison and Danny Tarlow. Gumbel machinery. `https://cmaddis.github.io/gumbel-machinery`, Jan 2017.

[47] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57), 2021.

[48] Axel Brando, Jose A Rodriguez, Jordi Vitria, and Alberto Rubio Muñoz. Modelling heterogeneous distributions with an uncountable mixture of Asymmetric Laplacians. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, Vancouver, Canada, 2019.

[49] Antoine Wehenkel and Gilles Louppe. Graphical Normalizing Flows. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AIStats)*, pages 37–45. PMLR, 2021.

[50] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 8024–8035. Vancouver, Canada, 2019.

[51] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural Spline Flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 7511–7522, Vancouver, Canada, 2019.

[52] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[53] Jennifer L Hill. Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20:217–240, 2011.

[54] Robert J LaLonde. Evaluating the econometric evaluations of training programs with experimental data. *The American Economic Review*, pages 604–620, 1986.

[55] Hadi Fanaee-T and Joao Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, 2013.

[56] Employee compensation. City of Phoenix Open Data. https://www.phoenixopendata.com/dataset/employee-compensation. Accessed on May 2022.

[57] Genderize. Demografix ApS. https://genderize.io/. Accessed on May 2022.