Architectures and learning algorithms for data-driven decision making

by

Jason Siyanda Hartford

B.Sc., University of Witwatersrand, 2008

M.EconSc., University of Witwatersrand, 2011

M.Sc., University of British Columbia, 2016

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

April 2021

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Architectures and learning algorithms for data-driven decision making

submitted by **Jason Siyanda Hartford** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Computer Science**.

Examining Committee:

Kevin Leyton-Brown, Department of Computer Science, University of British Columbia
Supervisor

Frank Wood, Department of Computer Science, University of British Columbia
Supervisory Committee Member

Mark Schmidt, Department of Computer Science, University of British Columbia
Supervisory Committee Member

Jeff Clune, Department of Computer Science, University of British Columbia
University Examiner

Alexandre Bouchard-Côté, Department of Statistics, University of British Columbia
University Examiner

Russ Greiner, Department of Computing Science, University of Alberta
External Examiner

Abstract

To design good policy, we need accurate models of how the decision makers that operate within a given system will respond to policy changes. For example, an economist reasoning about the design of an auction needs a model of human behavior in order to predict how changes to the auction design will be reflected in outcomes; or a doctor deciding on treatments needs a model of people's health responses under different treatments to select the best treatment policy. We would like to leverage the accuracy of modern deep learning approaches to estimate these models, but this setting brings two non-standard challenges. First, decision problems often involve reasoning over sets of items, so we need deep networks that reflect this structure. The first part of this thesis develops a deep network layer that reflects this structural assumption, and shows that the resulting layer is maximally expressive among parameter tying schemes. We then evaluate deep network architectures composed of these layers on a variety of decision problems from human decision making in a game theory setting, to algorithmic decision making on propositional satisfiability problems. The second challenge is that predicting the effect of policy changes involves reasoning about shifts in distribution: any policy change will, by definition, change the conditions under which decision makers operate. This violates the standard machine learning assumption that models will be evaluated under the same conditions as those under which they were trained (the "independent and identically distributed" data assumption). The second part of this thesis shows how we can train deep networks that make valid predictions of the results of such policy interventions, by adapting the classical causal inference method of instrumental variables. Finally, we develop methods that are robust to some violations of the instrumental variable assumptions in settings with multiple instrumental variables.

Lay Summary

A key role of applied science and engineering is using past data to recommend good future policies. For example, epidemiologists recommend lifestyle policies by experimentally studying their health outcomes, economists recommend social policy by inferring their effects from natural experiments, and data scientists build personalized recommendations by analyzing past behavior. In all of these applications, more accurate predictions facilitate better policy decisions. This thesis develops methods that adapt recent advances in machine learning to make more accurate predictions in these settings. It has two parts. First we develop deep network architectures for modeling decision behavior that involves reasoning over sets of items—a common structural property of data from a variety of decision problems from game theory to recommender systems. Second, we leverage ideas from causal inference to develop the learning algorithms that ensure the predictions remain valid after the policy changes are implemented.

Preface

The work presented in this thesis is based on work that was published or is under review.

- Chapter 2 appeared at ICML 2018 in,

  Jason Hartford, Devon R Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. "Deep Models of Interactions Across Sets." In Proceedings of the 35th International Conference on Machine Learning 2018

  It was joint work with Devon Graham, Siamak Ravanbakhsh and Kevin Leyton-Brown. Devon and I were joint first authors on the paper. Devon and Siamak worked on the theoretical results showing that proposed layer is maximally expressive, all four of us jointly developed the methods for sampling, regularization and combining the layers into deep network architectures, and Devon and I wrote the code and performed the experiments.

- Chapter 3 appeared at NeurIPS 2016 in,

  Jason Hartford, James R. Wright, Kevin Leyton-Brown. "Deep Learning for Predicting Human Strategic Behavior." In the Advances in Neural Information Processing Systems 2016

  It was joint work with James Wright and Kevin Leyton-Brown. I developed the deep network architecture with James and Kevin, and I wrote the code, conducted the experiments and wrote the first draft of the paper. This work was conducted during my Masters, but is included in this thesis because it provides a useful application for exchangeable models. It is a very similar ar-

chitecture to the model developed in Chapter 2, which provides the theoretical argument for the approach taken.

- Chapter 4 appeared at AAAI 2020 in,

  Chris Cameron, Rex Chen, Jason Hartford, Kevin Leyton-Brown. "Predicting Propositional Satisfiability via End-to-End Learning." In the Proceedings of the AAAI Conference on Artificial Intelligence 2020

  It is joint work with Chris Cameron, Rex Chen and Kevin Leyton-Brown. I adapted the architecture from Chapter 2 for use in the SAT domain, wrote the first version of the code, and developed the encoding of the SAT problems as exchangeable matrices with Chris Cameron. Chris, Rex and I all ran experiments and we jointly wrote the paper.

- Chapter 5 appeared at ICML 2017 in,

  Jason Hartford, Greg Lewis, Kevin Leyton-Brown, Matt Taddy. "Deep IV: A Flexible Approach for Counterfactual Prediction." In the Proceedings of the 34th International Conference on Machine Learning 2017

  It was joint work with Matt Taddy, Greg Lewis and Kevin Leyton-Brown. Matt and Greg identified non-parametric instrumental variable estimation as likely to benefit from machine learning-based methods; I proposed the deep learning-based solution that appears in the paper and we jointly worked on the details of the method. I wrote all the code, conducted the experiments and all four of us wrote the paper.

- Chapter 6 has been accepted to appear at ICML 2021 in,

  Jason Hartford, Victor Veitch, Dhanya Sridhar, Kevin Leyton-Brown. "Valid Causal Inference with (Some) Invalid Instruments." In the Proceedings of the 38th International Conference on Machine Learning 2021

  Machine Learning, 2021. It is joint work with Victor Veitch, Dhanya Sridhar and Kevin Leyton-Brown. Under Kevin's supervision, I proposed the method, wrote the code, ran the experiments and wrote the first draft of the paper. Victor and I worked jointly on the theory and all four of us helped write the final version of the paper.

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I want to thank my advisor Kevin Leyton-Brown. It is hard to overstate Kevin's influence on my approach to research, the presentation of my work, and how to think about what to problems work on. If this thesis has any lasting impact, it will be because Kevin continually pushed me to ask what makes the problem important? What gives me a comparative advantage in tackling it? And how can we present the key ideas clearly?

This work would not have been possible without collaborations which shaped the two parts of this thesis. Matt Taddy and Greg Lewis hosted me at Microsoft Research as I was starting my PhD and the work that came out of this collaboration both changed my research direction—before working with Matt and Greg I barely understood the basics of causal inference, and never imagined working on it—and deepened my appreciation for interdisciplinary work. I find that most interesting developments in machine learning are strongly influenced by collaborations across disciplines, and because of its interdisciplinary nature, Microsoft Research was an ideal place to experience this phenomenon first hand.

The first part of this thesis builds on a collaboration with Siamak Ravanbakhsh and Devon Graham who developed the theoretical foundations for the equivariant layers. This collaboration made me realize that the methods that I had first worked on in the context of behavioral game theory were far more broadly applicable than I had initially realized, and ultimately gave me the confidence to apply these ideas in SAT.

My time at UBC would not have been nearly as enjoyable without GTDT: Chris Cameron, Neil Newman, Heddy Zarkoob, Taylor Lundy, Greg D'Eon, Devon Graham, James Wright, Rex Chen and all the undergrads and interns who have joined

the group over the years. Thank you for keeping long lab meetings entertaining and for exploring British Columbia's backcountry with me.

To my supervisory committee Frank Wood and Mark Schmidt, thank you for guidance and the push to get this thesis out into the world. A special thanks to Mark for his teaching, both in class and in reading groups. Mark has an unusual ability to present machine learning material rigorously without letting the rigor get in the way of clarity, and his reading groups have equipped me with a breadth of machine learning knowledge that goes far beyond the hot current trends. Mark's machine learning group has also been my unofficial second home at UBC. Fred Kunstner, Aaron Mishkin, Cathy Meng, Wilder Lavington, Betty Shea, Sharan Vaswani and Reza Babanezhad, thank you for all for all the coffees, technical debates, general rants about machine learning, and good company over the last few years.

To my three examiners, Jeff Clune, Bouchard-Côté and Russ Greiner, thank you for spending so much time on my thesis; I really enjoyed my defense and I felt like I learned a lot about my own work through the process. A special thanks to Russ who clearly went far beyond what was expected from an external examiner: he highlighted many places where technical presentation could be improved and found an embarrassing number of typos! This thesis is significantly improved as a result.

Thank you to Benjamin Bloem-Reddy for our conversations about the relationship between the two parts of this thesis. That work is still ongoing, but it helped shape the introduction to my thesis.

And finally, thank you to my to family, Nicole, Lyra and Rae. When Nicole and I left South Africa to come to Canada, we thought that we would only be gone for a two year Masters program; we still have some boxes of our stuff in our parents houses, waiting for us to return. Six and a half years later, we are now married with two wonderful children and are planning to spend the next two years in Montreal for my post-doc... and who knows what that next step will turn into. Nicole, thank you for supporting me on this adventure, for getting me out of the office and into the mountains, and embracing the nomadic lifestyle that academia necessitates. Lyra and Rae, thank you for being the stars that light up my world. And last but certainly not least, thank you to my mother Louise Almon for flying all the way to Canada during a pandemic to help us handle the birth of a newborn, a toddler and a thesis deadline; I am not sure I would have finished without the help.

For Nicole, Lyra, and Rae.

Chapter 1

Introduction

A key role of applied science and engineering is recommending good policies to decision makers such as users of a system or individuals seeking expert advice. For example, epidemiologists may study the effect of body mass index (BMI) on heart disease [Holmes et al., 2014] so that they can recommend lifestyle "policies" for people that lead to good health outcomes; economists study the downstream effects of incarceration on economic and social outcomes [Frandsen et al., 2019] so that they can recommend more humane and effective sentencing policies to judges; and data scientists aim to predict users' preferences to parameterize the policies that drive product recommendations and search ranking algorithms in order to ensure that relevant items are shown. Because the quality of these policy recommendations is bounded by the accuracy of whatever model is used to predict the effect of different potential policies, we would like to leverage the massive advances that deep learning has made in building more accurate predictive models. At first glance, this appears to be a direct application of existing supervised learning ideas: assuming we have sufficient historical data on responses to past policy changes, the task of learning a function that maps from past observations to outcomes is well-understood. But, modeling decision making requires enforcing appropriate invariances in order to predict the effect of new policies and not just report the effects of policies under past regimes.

To illustrate, consider the toy preference learning problem shown in Figure 1.1. We are given five examples where a user was presented with a choice set of three

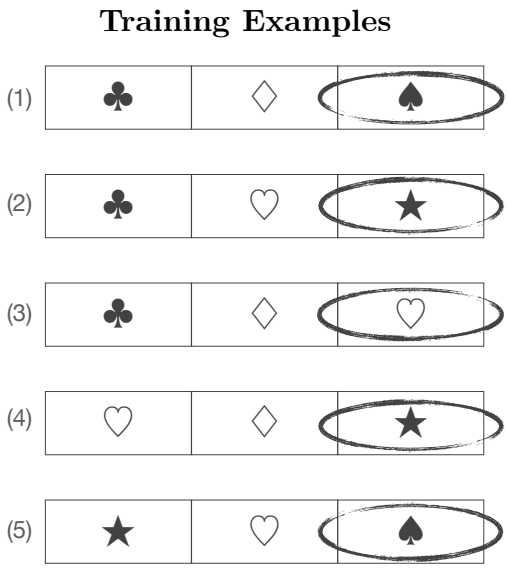

Figure 1.1: A toy preference learning example. From the items selected in each of the five choice sets, we can infer that ♠, ⋆ and ♡ are preferred over ♣ and ♢ because the latter two are never selected. Further, ♠ is selected over ⋆, and ⋆ is selected over ♡, so if we assume transitivity, any function that enforces this ordering, ♠ ≻ ⋆ ≻ ♡ ≻ ♣ ∼ ♢, would be consistent with this user's choices. Alternatively, a shortcut that simply selects the third item in each set would also correctly predict the user's choices under this ordering, but it would fail if the choice sets were reordered.

shapes and they circled their preferred shaped; our task is to learn a mapping between from the choice set to the selected item[1]. When laid out in this configuration, there are at least two mappings that solve this problem. The first involves understanding the user's underlying preference ordering: from the items selected in the example choice sets, we can see that ♠ ≻ ⋆ ≻ ♡ ≻ ♣ ∼ ♢ (see the figure caption for details). Alternatively, we could also notice that in each example that we were shown, the selected item was in the last position and so a *shortcut* solution [Geirhos et al., 2020] for this layout is to ignore the input entirely and just "solve" the problem by

[1]For the purpose of this example, we assume that the user's preferences are only over the shapes themselves, and not over their layout.

always reporting the last position as our solution. As long as we are only ever tested on examples that have this relationship between the order and preferences—perhaps because the data was collected under an existing layout algorithm that induced this dependence—both approaches to solving the problem are "correct" in so far as they give the correct output, and will even generalize to new examples collected under the same regime. But this second solution is brittle: if instead of a choice set of three shapes we are given four, it is not obvious how to apply the shortcut—should we select the third position or the last position?—and of course, the shortcut fails for any layout where the target is not in the third / last position.

This example highlights two problems with the shortcut solution. First, it relies on a spurious relationship between the choice set and target solutions. The correlation between the target solutions and their order is a function of a particular layout, rather than the result of the causal mapping that results from the user's preferences. And second, the shortcut solution is tightly coupled to the particular size of the input. When one changes the number of candidate outputs, the shortcut does not unambiguously map onto the new output size.

These problems are accentuated if we take a naive machine learning approach to solving this task. The simplest encoding would flatten the three shapes into a vector of fixed size and use a standard multilayer perceptron to predict the target item for each example. We could then minimize a loss on the five examples[2] using gradient descent. As before, if we found weight configurations that encode either solution we would get zero loss and hence, would solve the problem from the perspective of an empirical risk minimizer. However, whereas before we might have been indifferent between the two strategies to solving the problem, in this machine learning setup the shortcut solution would be preferred because it is easier for gradient descent

[2]With only five examples, we would have a significant overfitting problem, but the same argument applies to an infinite data regime.

to learn,[3] and a deep network of this form would not even be able to take the four shape problem as input because it only operates on fixed sized inputs.

Of course, this is just a toy problem,[4] but these problems with the shortcut solution illustrate two desiderata for learned models.

1. We want our hypothesis class to reflect the known invariances inherent in the data: the intended mapping in our toy problem was a set-valued function, so any permutation in the input should leave the output unchanged up to a corresponding permutation. At a minimum, any candidate solution should reflect this known structure.

2. We want models that remain valid under interventions that break the dependence between the content of the input and the policy under which the data was collected. In the toy example, the content of the input is the particular shapes that were used in each choice set, while the layout algorithm determines the policy under which the data was collected; we want a model that makes valid predictions even if the layout is chosen independently of the content such that the target no longer always appears in a predictable position.

It turns out that in this particular problem, addressing the first desideratum fixes both failures (because set-valued functions cannot depend on ordering by construction), but in general either problem may occur. As such, this thesis addresses the two desiderata in two parts. In Part 1, we develop a deep network architecture that is equivariant[5] with respect to a set of permutations of the input. This allows us to model problems that involve reasoning over sets of inputs, which we evaluate across a variety of applications from human decision making in recommender

[3]The shortcut solution can be represented in a multilayer perceptron by setting all the weights to zero except the bias term in the output layer. When parameters are initialized near zero, this solution essentially involves optimizing only one parameter, and is "simple" in that most of the weights are zero. We know from recent work [Belkin et al., 2019] on the implicit bias of stochastic gradient descent in overparameterized models, if the minimizer of a loss function is not unique, gradient descent will choose the "simplest" solution (under the appropriate norm). More complex examples of the phenomenon of "shortcut learning" are surveyed in Geirhos et al. [2020].

[4]Though examples of layout-induced confounding do occur in real data; for example, see the sponsored search experiment in Section 5.5.2.

[5]A function, f is *invariant* with respect to a set of transformations $\mathcal{G}$, if for all $g \in \mathcal{G}$, $f(x) = f(g(x))$; that is, its output is unchanged for all transformations in $\mathcal{G}$. A function is *equivariant*, if the function's output is constant up to a corresponding transformation of the output, so $g_y(f(x)) = f(g_x(x))$.

systems and game theoretic settings, to algorithmic decision making in Boolean satisfiability problems. Part 2 addresses the second desideratum by developing methods for performing causal inference using deep networks. The dependence between layout and content is an example of confounding that was induced by the layout algorithm. By design, causal inference approaches are robust to these confounding effects by aiming to estimate the effect of "interventions" on the input that break these dependencies.

1.1 Deep networks for permutation equivariant data

If there are known invariances in the data, we would like to enforce them through the choice of deep network architecture such that they are reflected regardless of the training procedure. By far the most successful example of this is the convolution architecture [LeCun et al., 1989] which enforces equivariance with respect to translations in images. This idea has been extended and generalized across a large range of transformations from 2D and 3D rotations [Cohen et al., 2018], or to more general classes of transformations defined by actions of discrete and compact groups [Shawe-Taylor, 1989; Wood and Shawe-Taylor, 1996; Cohen and Welling, 2016; Ravanbakhsh et al., 2017; Kondor and Trivedi, 2018; Cohen et al., 2019]. The unifying idea behind all of these contributions is that, if the symmetries in the data can be expressed in terms of the actions of a group, there exists an appropriately defined convolution operation that enforces this invariance. Given some known symmetry, the challenge is to characterize the associated convolution operation.

Chapter 2 discusses my work in developing a deep network architecture for matrices of data that are equivariant with respect to permutations of their rows or columns. Data of this form is particularly important to decision making because it is prevalent in settings where decision makers reason over the relationships between discrete sets. For example, in game theory, decision makers have to decide which of a set of actions to play, while accounting for the fact that their payoffs are a function of both their actions and the associated set of their opponent's actions. The standard representation for these problems is the so-called *normal form*: a bi-matrix of payoffs that is equivariant with respect to permutations of rows and columns because they correspond to relabelings of the actions that leave the game

strategically equivalent. Chapter 3 will show an application of this architecture for the behavioral game theory problem of predicting the distribution of actions people will take in a given a two player normal-form game.

The same permutation invariant structure is present in the standard representation of logical formulae that is used by propositional satisfiability solvers. Every logical formula has an equivalent equivalent representation as a conjunction—a logical AND—over clauses, each of which consists only of a disjunction—a logical OR—over a set of variables [see Russell and Norvig, 2010, Chapter 7]. There is a natural representation of these formulae as a matrix indexed by clauses and variables, and because both the conjunction and disjunction operations are commutative, this matrix is permutation invariant. Chapter 4 discusses my work that uses the architecture from Chapter 2 for predicting the satisfiability of logical formulae.

1.2 Causal inference

For any learning problem whose goal is to guide decision making, we will typically have to consider the relationship between three sets of variables: a 'treatment' or decision variable, t, that records what action was taken in each observation, a context variable; x, that records the observed covariates or state variables that guided the choice of action; and a response variable y that is a function of both the treatment and the context. This set up is true of both reinforcement learning and causal inference, but while in reinforcement learning you have both the ability to experiment, and knowledge of the policy that was used for each observation, in causal inference you have neither. The goal is still to predict the effect to taking actions by setting the treatment variable, which is complicated by the fact that, because both the treatment and response variables share common causes, there may be marginal dependence between the two variables even if the treatment has no effect on the response. For example, if a medical treatment is expensive, rich people may be more likely to decide to take it because they can afford it. But, they will also have better health outcomes independently of the effect of the treatment because they have access to better nutrition, exercise regimes, and so forth. These differences will bias naive estimation procedures that ignore this wealth effect.

There are a large class of methods that address this problem [for textbook treatments see Angrist and Pischke, 2008; Pearl, 2009; Imbens and Rubin, 2015; Hernán and Robins, 2020], all of which require that we make assumptions about the relationship between any observed and latent variables that affect both the treatment and response. The popular approaches to solving causal inference problems can be broadly categorized by whether or not you assume that x includes all common causes that may confound the relationship between t and y. In the "unconfounded" setting where x includes all confounders, if it is the case that for each x, every treatment value has some probability of being selected (the "overlap" assumption), we can estimate the causal effect of taking actions. Overlap ensures that, given a large enough dataset, we will eventually try every treatment—we cannot expect to say something about the relationship between treatments and responses for treatments we never try—and unconfoundedness ensures that any conditional relationship (given x) that we observe is not driven by some unobserved factor.

In this thesis we focus on the second class of methods, that do not assume unconfoundedness but instead rely on the existence of an "instrumental variable" to identify the causal effect [Wright, 1928; Reiersøl, 1945; Angrist and Pischke, 2008]. An instrumental variable, z, is a variable that is assumed to affect the response only through its effect on the treatment ($z \to t \to y$). Under this assumption, the difference between the instrument's respective effect on the treatment, ($z \to t$), and the response, ($z \to y$), can be attributed to the causal relationship between the treatment and response ($t \to y$), and as a result, we can identify its effect. Chapter 5 discusses my work that uses instrumental variables with deep networks to estimate to the effect of interventions in the presence of unobserved confounding factors. The instrumental variable assumptions are powerful but restrictive so in Chapter 6 develops an approach that lets us apply instrumental variable estimation more broadly by showing how we can leverage multiple candidate instruments to perform valid inference even if only a subset of the candidates are valid.

Part I

Permutation Equivariance: Models and Applications

Chapter 2

Deep Networks for Exchangeable Arrays

In this chapter we develop a deep network layer that is equivariant under permutations of rows and columns of input matrices (with extensions to higher order tensors) and we show that the layer is maximally expressive among all possible parameter tying schemes for achieving equivariance. Empirically, we show that this architecture gives strong generalization performance both within distribution and in tasks that require transfer to new distributions without retraining.

This chapter presents the layer in the context of matrix completion for recommender systems—arguably the canonical machine learning task for exchangeable arrays—and Chapter 3 and Chapter 4 will discuss its applications to behavioral game theory and propositional satisfiability respectively.

2.1 Introduction

A canonical problem where we need to learn a model of a decision maker's behavior is the problem of *matrix completion* for recommender systems. We are given a sparse matrix of data of the form shown in Figure 2.1 (left), and our task is to fill in the blanks: we have to predict what rating each user would have given to each movie so that we can recommend unrated movies that we expect the users would like.

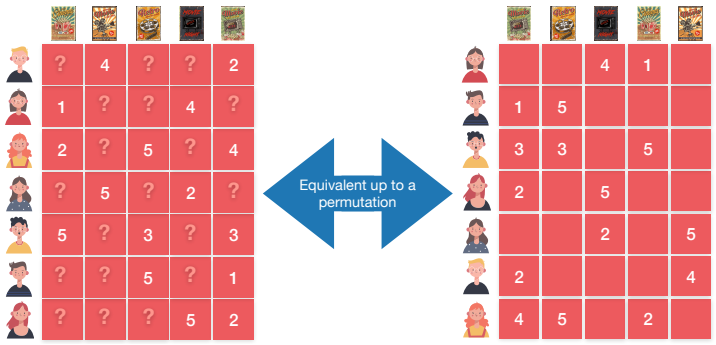

Figure 2.1: Example ratings matrix indexed by users on the rows and movies on the columns. Our task is to fill in the blanks indicated with a '?' on the left. Notice that the matrix on the left and the matrix on the right are equivalent up to a permutation of the rows and columns.

This data has a structural property that we can leverage: any row- or column-wise permutation of this sparse matrix only changes how the matrix is represented on the page (Figure 2.1 (right)), but leaves the underlying meaning of the data—the relationship between each user's ratings and the corresponding movies—unchanged. Matrices with this property are known as *exchangeable matrices*, and as we will see in later chapters, they can represent the input to a variety of decision problems where decision makers have to reason over sets of items.

Exchangeability has a long history in machine learning and statistics. de Finetti's theorem [De Finetti, 1930] states that exchangeable sequences are the product of a latent variable model. Extensions of this theorem characterize distributions over other exchangeable structures, from matrices to graphs; see Orbanz and Roy [2015] for a detailed treatment. In machine learning, a variety of frameworks formalize exchangeability in data, from plate notation to statistical relational models [Getoor and Taskar, 2007; Raedt et al., 2016]. For example, in a simple mixture model, the mixture components are exchangeable. Topic models [Blei, 2012] assume several types of interrelated exchangeabilities; e.g., in Latent Dirichlet Allocation, all the words, topics and documents are exchangeable. When dealing with exchangeable arrays (or tensors), a common approach is tensor factorization. In particular, one thread of work leverages tensor decomposition for inference in latent variable mod-

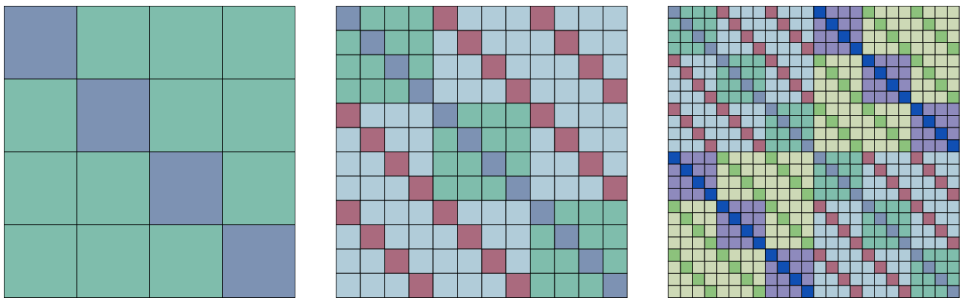

Figure 2.2: Structure of our parameter matrix for the 1D (left), 2D (centre), and 3D (right) input arrays. The parameter-sharing patterns for the weight matrix of the higher dimensional arrays can be produced via the *Kronecker product* of the weight matrix for the 1D array (i.e., vector).

els [Anandkumar et al., 2014]. However, in addition to having limited expressive power, tensor factorization requires retraining models for each new input.

Our goal in this chapter is to design learning algorithms that respect this property by being *permutation equivariant*. A function is permutation equivariant if its output is invariant to row- or column-permutations of the input, up to a corresponding permutation of the output. How might we achieve this? One approach for a given $N \times M$ matrix would be to augment the input with all $M! \times N!$ permutations of rows and columns. However, this is computationally wasteful and becomes infeasible for all but the smallest N and M. Instead, we show that there exists a simple parameter-sharing scheme that constrains the weights of a standard feed-forward layer such that a deep model composed of these layers can represent only permutation equivariant functions. The result is analogous to the idea of a convolution layer: a lower-dimensional effective parameter space that enforces a desired equivariance property. Indeed, parameter-sharing is a generic and efficient approach for achieving equivariance in deep models [Ravanbakhsh et al., 2017]. Additionally, we prove that our parameter-sharing scheme is maximally expressive among all possible parameter-sharing schemes.

When the exchangeable matrix models the interaction between the members of the same group, one could further constrain permutations to be identical across both rows and columns. An example of such a *jointly exchangeable matrix* [Orbanz and Roy, 2015], modeling the interaction of the nodes in a graph, is the adjacency matrix

deployed by graph convolutions. Our approach reduces to a graph convolution over edges in the special case of 2D arrays with this additional parameter-sharing constraint, but also applies to arbitrary matrices and higher dimensional arrays. Our model for exchangeable matrices can also be seen as a generalization of the *deep sets* architecture [Zaheer et al., 2017], by regarding a set as a one-dimensional exchangeable array.

In Section 2.2, we begin by introducing our exchangeable matrix layer which uses a parameter-sharing approach to enforce permutation equivariance. In Section 2.3.2, we study two architectures for matrix completion that use an exchangeable matrix layer. In particular the factorized autoencoding model provides a powerful alternative to commonly used matrix factorization methods; notably, it does not require retraining to be evaluated on previously unseen data. Our results also generalize higher-dimensional tensors, which we show in Section 2.5 and we conclude with further analysis and discussion in Section 2.5.

2.2 Exchangeable Matrix Layer

Let $X \in \mathbb{R}^{N \times M}$ be our exchangeable input matrix. We use $\text{vec}(X) : \mathbb{R}^{N \times M} \to \mathbb{R}^{NM}$ to denote its vectorized form of X and $\text{vec}^{-1}(x) : \mathbb{R}^{NM} \to \mathbb{R}^{N \times M}$ to denote the inverse of the vectorization that reshapes a vector of length NM into an $N \times M$ matrix[1]—i.e., $\text{vec}^{-1}(\text{vec}(X)) = X$. Consider a fully connected layer $Y := \text{vec}^{-1}(\sigma(W\,\text{vec}(X)))$ where σ is an element-wise nonlinear function such as sigmoid, $W \in \mathbb{R}^{NM \times NM}$, and $Y \in \mathbb{R}^{N \times M}$ is the output matrix. Exchangeability of X motivates the following property: suppose we permute the rows and columns X using two arbitrary permutation matrices $G^{(N)} \in \{0,1\}^{N \times N}$ and $G^{(M)} \in \{0,1\}^{M \times M}$ to get $X' := G^{(N)} X G^{(M)}$. permutation equivariance requires the new output matrix $Y' := \text{vec}^{-1}(\sigma(W\,\text{vec}(X')))$ to experience the same permutation of rows and columns—that is, we require $Y' = G^{(N)} Y G^{(M)}$.

[1] We should write $\text{vec}^{-1}(x, N, M)$, since the inverse operation depends on the size of the original matrix; we suppress this dependence for notational clarity

[2] This definition is simplified to ease exposition; the full definition (see Section 2.5) adds the additional constraint that the layer *not* be equivariant wrt any other permutation of the elements of X. Otherwise, a trivial layer with a constant weight matrix $W_{n,m} = c$ would also satisfy the stated equivariance property.

Definition 2.2.1 (exchangeable matrix layer, simplified[2])**.** Given $X \in \mathbb{R}^{N\times M}$, a fully connected layer $\sigma(W\,\mathrm{vec}(X))$ with $W \in \mathbb{R}^{NM\times NM}$ is called an exchangeable matrix layer if, for all permutation matrices $G^{(N)} \in \{0,1\}^{N\times N}$ and $G^{(M)} \in \{0,1\}^{M\times M}$, permutation of the rows and columns results in the same permutations of the output:

$$\begin{aligned}\mathrm{vec}^{-1}(\sigma(W\,\mathrm{vec}(G^{(N)}XG^{(M)}))) = \\ G^{(N)}\,\mathrm{vec}^{-1}(\sigma(W\,\mathrm{vec}(X)))G^{(M)}.\end{aligned} \tag{2.1}$$

This requirement heavily constrains the weight matrix W: indeed, its number of effective degrees of freedom cannot even grow with N and M. Instead, the resulting layer is forced to have the following, simple form:

$$W_{(n,m),(n',m')} := \begin{cases} w_1 & n = n', m = m' \\ w_2 & n = n', m \neq m' \\ w_3 & n \neq n', m = m' \\ w_4 & n \neq n', m \neq m'. \end{cases} \tag{2.2}$$

For each output element $Y_{n,m}$, we have the following parameters: one connecting it to its counterpart $X_{n,m}$; one each connecting it to the inputs of the same row and the inputs of the same column; and one shared by all the other connections. We also include a bias parameter; see Figure 2.2 for a visual illustration of this parameter matrix. Theorem 1 formalizes the requirement on our parameter matrix. All proofs are deferred to Appendix A.

Theorem 1. *Given a strictly monotonic function σ, a neural layer $\sigma(W\,\mathrm{vec}(X))$ is an **exchangeable matrix layer** iff the elements of the parameter matrix W are tied together such that the resulting fully connected layer simplifies to*

$$\begin{aligned}Y = \sigma\Bigg(w'_1 X + \frac{w'_2}{N}(\mathbf{1}_N\mathbf{1}_N^\mathsf{T}X) + \frac{w'_3}{M}(X\mathbf{1}_M\mathbf{1}_M^\mathsf{T}) \\ + \frac{w'_4}{NM}(\mathbf{1}_N\mathbf{1}_N^\mathsf{T}X\mathbf{1}_M\mathbf{1}_M^\mathsf{T}) + w'_5\mathbf{1}_N\mathbf{1}_M^\mathsf{T}\Bigg)\end{aligned} \tag{2.3}$$

where $\mathbf{1}_N = \underbrace{[1,\ldots,1]^\mathsf{T}}_{\text{length } N}$ *and* $w'_1,\ldots,w'_5 \in \mathbb{R}$.

This theorem shows that the parameter sharing scheme in Equation 2.2 produces a layer that is equvariant to exactly those permutations we desire, and moreover, it is optimal in the sense that any layer having fewer ties in its parameters (i.e., more parameters) would fail to be equivariant, and hence the layer is maximally expressive among all possible parameter sharing schemes.

This parameter sharing can be implemented efficiently by summing or averaging elements across rows and columns; more generally, permutation equivariance is preserved by any commutative pooling operation. Moreover, stacking multiple layers with the same equivariance properties preserves equivariance [Ravanbakhsh et al., 2017]. This allows us to build *deep* permutation equivariant models.

Multiple Input–Output Channels Equation 2.3 describes the layer as though it has single input and output matrices. However, as with convolution, we may have K input and O output channels. We use superscripts $X^{\langle k \rangle}$ and $Y^{\langle o \rangle}$ to denote such channels. Cross-channel interactions are fully connected—that is, we have five unique parameters $w_1^{\langle k,o \rangle}, \ldots, w_5^{\langle k,o \rangle}$ for *each combination* of input–output channels; note that the bias parameter w_5 does not depend on the input. Similar to convolution, the number of channels provides a tuning nob for the expressive power of the model. In this setting, the scalar output $Y_{n,m}^{\langle o \rangle}$ is given as

$$Y_{n,m}^{\langle o \rangle} = \sigma\Bigg(\sum_{k=1}^{K}\Bigg(w_1^{\langle k,o \rangle} X_{n,m}^{\langle k \rangle} + \frac{w_2^{\langle k,o \rangle}}{N}\Big(\sum_{n'} X_{n',m}^{\langle k \rangle}\Big) + \frac{w_3^{\langle k,o \rangle}}{M}\Big(\sum_{m'} X_{n,m'}^{\langle k \rangle}\Big) + \frac{w_4^{\langle k,o \rangle}}{NM}\Big(\sum_{n',m'} X_{n',m'}^{\langle k \rangle}\Big) + w_5^{\langle o \rangle}\Bigg)\Bigg) \tag{2.4}$$

Input Features for Rows and Columns In some applications, in addition to the matrix $X \in \mathbb{R}^{N \times M \times K}$, where K is the number of input channels/features, we may have additional features for rows $R \in \mathbb{R}^{N \times K'}$ and/or columns $C \in \mathbb{R}^{M \times K''}$. We can preserve permutation equivariance by broadcasting these row/column features over the whole matrix and treating them as additional input channels.

Jointly Exchangeable Matrices For jointly exchangeable matrices, such as adjacency matrix, Equation 2.1 is constrained to have $N = M$ and $G^{(N)} = G^{(M)}$. This will constrain the corresponding parameter-sharing so that $w_2 = w_3$ in Equation 2.2.

2.2.1 Sparse Inputs

Real-world arrays are often extremely sparse. Indeed, matrix and tensor completion is only useful with missing entries. Fortunately, the equivariance properties of Theorem 1 continue to hold when we only consider the nonzero (observed) entries. For sparse matrices, we continue to use the same parameter-sharing scheme across rows and columns, with the only difference being that we limit the model to observed entries. We now make this precise.

Let $X \in \mathbb{R}^{N \times M \times K}$ be a sparse exchangeable array with K channels, where *all* the channels for each row-column pair $\langle n, m \rangle$ are either fully observed over all K channels or completely missing. Let $\mathbb{I}$ identify the set of such non-zero indices. Let $\mathbb{R}_n = \{m \mid (n, m) \in \mathbb{I}\}$ be the non-zero entries in the n^{th} row of X, and let $\mathbb{C}_m$ be the non-zero entries of its m^{th} column. For this sparse matrix, the terms in the layer of Equation 2.4 are adapted as one would expect:

$$
\begin{aligned}
\frac{w_2^{\langle k,o \rangle}}{N} \sum_{n'} X_{n',m}^{\langle k \rangle} \quad &\to \quad \frac{w_2^{\langle k,o \rangle}}{|\mathbb{C}_m|} \sum_{n' \in \mathbb{C}_m} X_{n',m}^{\langle k \rangle} \\
\frac{w_3^{\langle k,o \rangle}}{M} \sum_{m'} X_{n,m'}^{\langle k \rangle} \quad &\to \quad \frac{w_3^{\langle k,o \rangle}}{|\mathbb{R}_n|} \sum_{m' \in \mathbb{R}_n} X_{n,m'}^{\langle k \rangle} \\
\frac{w_4^{\langle k,o \rangle}}{NM} \sum_{n',m'} X_{n',m'}^{\langle k \rangle} \quad &\to \quad \frac{w_4^{\langle k,o \rangle}}{|\mathbb{I}|} \sum_{(n',m') \in \mathbb{I}} X_{n',m'}^{\langle k \rangle}
\end{aligned}
$$

2.3 Matrix Factorization and Completion

Recommender systems are very widely applied, with many modern applications suggesting new items (e.g., movies, friends, restaurants, etc.) to users based on previous ratings of other items. The core underlying problem is naturally posed as a matrix completion task: each row corresponds to a user and each column

corresponds to an item; the matrix has a value for each rating given to an item by a given user; the goal is to fill in the missing values of this matrix.

In Subsection 2.3.1 we review two types of analysis in dealing with exchangeable matrices. Subsection 2.3.2 introduces two architectures: a self-supervised model—a simple composition of exchangeable matrix layers—that is trained to produce randomly removed entries of the observed matrix during the training; and a factorized model that uses an auto-encoding nonlinear factorization scheme. While there are innumerable methods for (nonlinear) matrix factorization and completion, both of these models are the first to generalize to inductive settings while achieving competitive performance in transductive settings. Subsection 2.3.3 introduces two subsampling techniques for large sparse matrices followed by a literature review in Subsection 2.3.4.

2.3.1 Inductive and Transductive Analysis

In matrix completion, during training we are given a sparse input matrix X_{tr} with observed entries $\mathbb{I}_{\text{tr}} = \{(n,m)\}$. At test time, we are given X_{ts} with observed entries $\mathbb{I}_{\text{ts}} = \{(n',m')\}$, and we are interested in predicting (some of) the missing entries of X_{ts}, expressed through $\mathbb{I}'_{\text{ts}}$. In the transductive or *matrix interpolation* setting, $\mathbb{I}_{\text{tr}}$ and $\mathbb{I}_{\text{ts}}$ have overlapping rows and/or columns—that is, for every rating in the test set, we have seen one or more training rating from the same user or movie such that at least one of the following is true: $\{m \mid (n,m) \in \mathbb{I}\} \cap \{m' \mid (n',m') \in \mathbb{I}'\} \neq \emptyset$ or $\{n \mid (n,m) \in \mathbb{I}\} \cap \{n' \mid (n',m') \in \mathbb{I}'\} \neq \emptyset$. In fact, often X_{tr} and X_{ts} are identical. Conversely, in the inductive or *matrix extrapolation* setting, we are interested in making predictions about completely unseen entries: the training and test row/column indices are completely disjoint. We will even consider cases where X_{tr} and X_{ts} are completely different datasets—e.g., movie-rating vs music-rating. The same distinction applies in matrix factorization. Training a model to factorize a particular given matrix is transductive, while factorizing unseen matrices *after* training is inductive.

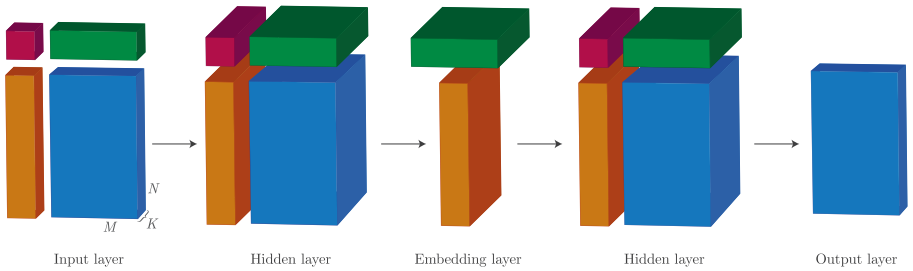

Figure 2.3: Factorized exchangeable autoencoder. The encoder maps from the input tensor to an embedding layer of row / column factors via one or more hidden layers. The decoder attempts to reconstruct the input using the factors via one or more hidden layers.

2.3.2 Architectures

Self-supervised Exchangeable Model When the task is matrix completion, we can construct a permutation equivariant deep network, $f_{ss} : \mathbb{R}^{N\times M\times K} \to \mathbb{R}^{N\times M\times K}$, by composing multiple exchangeable matrix layers. Given the matrix X with observed entries $\mathbb{I}$, we divide $\mathbb{I} = \mathbb{I}_{\text{in}} \cup \mathbb{I}_{\text{pr}}$ into disjoint input and prediction entries. We then train $f_{ss}(X_{\text{in}})$ to predict the prediction entries X_{pr}.

Factorized Exchangeable Autoencoder (FEA) Our factorized autoencoder is composed of an encoder and a decoder. The encoder $f_{\text{enc}} : \mathbb{R}^{N\times M\times K} \to \mathbb{R}^{K_N\times N} \times \mathbb{R}^{K_M\times M}$ maps the (sparse) input matrix $X \in \mathbb{R}^{N\times M\times K}$ to a row-factor $Z_N \in \mathbb{R}^{K_N\times N}$ and a column-factor $Z_M \in \mathbb{R}^{K_M\times M}$. To do so, the encoder uses a composition of exchangeable matrix layers. The output of the final layer $Y^l \in \mathbb{R}^{N\times M\times K^l}$ is pooled across rows and columns (and optionally passed through a feed-forward layer) to produce latent factors Z_N and Z_M. The decoder $g_{\text{dec}} : \mathbb{R}^{K_N\times N} \times \mathbb{R}^{K_M\times M} \to \mathbb{R}^{N\times M\times K}$ also uses a composition of exchangeable matrix layers, and reconstructs the input matrix X from the factors. The optimization objective is to minimize reconstruction error $\ell(X, g_{\text{dec}}(f_{enc}(X)))$; similar to classical autoencoders.

This procedure is also analogous to classical matrix factorization, with an important distinction: once trained, we can factorize unseen matrices without

performing any optimization. Note that the same architecture trivially extends to tensor factorization, where we use an exchangeable tensor layer (see Section 2.5).

Channel Dropout Both the factorized autoencoder and self-supervised exchangeable model are flexible enough to make regularization important for good generalization performance. Dropout [Srivastava et al., 2014] can be extended to apply to exchangeable matrix layers by noticing that each channel in an exchangeable matrix layer is analogous to a single unit in a standard feed-forward network. We therefore randomly drop out whole channels during training (as opposed to dropping out individual elements). This procedure is equivalent to the *SpatialDropout* technique used in convolutional networks [Tompson et al., 2015].

2.3.3 Subsampling in Large Matrices

A key practical challenge arising from our approach is that our models are designed to take the whole data matrix X as input and will make different (and typically worse) predictions if given only a subset of the data matrix. As datasets grow, the model and input data become too large to fit within fixed-size GPU memory. This is problematic both during training and at evaluation time because our models rely on aggregating shared representations across data points to make their predictions. To address this, we explore two subsampling procedures.

Uniform sampling The simplest approach is to sample sub-matrices of X by uniformly sampling from its (typically sparse) elements. This procedure is computationally cheap, but has the potential to limit the performance of the model because the relationships between the elements of X are sparsified.

Conditional sampling Rather than sparsifying interactions between all set members, we can pick a subset of rows and columns and maintain all their interactions; see Figure 2.4. We designed this procedure to ensure that each element $(n, m) \in \mathbb{I}$ has the same marginal probability of being sampled as it would have had under the uniform sampling scheme. To achieve this, we first sample a subset of rows $\mathbb{N}' \subseteq \mathbb{N} = \{1, \ldots, N\}$ from the marginal $P(n) := \frac{|\mathbb{R}_n|}{|\mathbb{I}|}$, followed by subsampling of

columns using the marginal distribution over the columns, within the selected rows: $P(m \mid \mathbb{N}') = \frac{|\{(m,n) \in \mathbb{I} | n \in \mathbb{N}'\}|}{|\{(m',n) \in \mathbb{I} | n \in \mathbb{N}'\}|}$. This gives us a set of columns $\mathbb{M}' \subseteq \mathbb{M}$. We consider any observation within $\mathbb{N}' \times \mathbb{M}'$ as our subsample: $\mathbb{I}_{\text{sample}} := \{(n,m) \in \mathbb{I} \mid n \in \mathbb{N}', m \in \mathbb{M}'\}$. This sampling procedure is more expensive than uniform sampling, as we have to calculate conditional distributions for each set of samples.

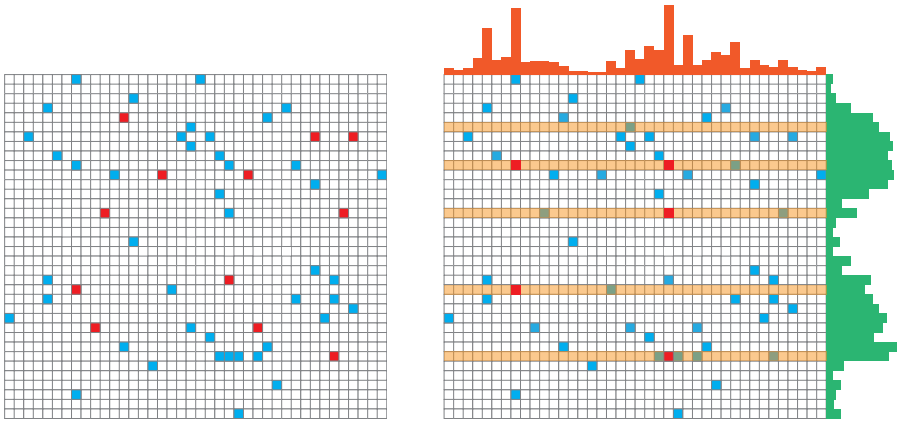

Figure 2.4: Uniform sampling (left) selects samples (red) uniformly from the non-zero indices of the the matrix X while conditional sampling (right) first samples a set of rows (shown in orange) from the row marginal distribution (green) and then selects sample from the resulting column conditional distribution.

2.3.4 Related Literature

The literature in matrix factorization and completion is vast. The classical approach to solving the matrix completion problem is to find some low rank (or sparse) approximation that minimizes a reconstruction loss for the observed ratings [see e.g., Mnih and Salakhutdinov, 2008; Candès and Recht, 2009; Koren et al., 2009]. Because these procedures learn embeddings for each user and item to make predictions, they are transductive, meaning they can only make predictions about users and items observed during training. To our knowledge this is also true for all recent deep factorization and other collaborative filtering techniques [e.g., Salakhutdinov et al., 2007; Dziugaite and Roy, 2015; Li et al., 2015; Sedhain et al., 2015; Wang

et al., 2015; Zheng et al., 2016; Deng et al., 2017]. An exception is a recent work by Yang et al. [2016] that extends factorization-style approaches to the inductive setting (where predictions can be made on unseen users and items). However their method relies on additional side information to represent users and items. By contrast, our approach is able to make inductive completions on rating matrices that may differ from that which was observed during training without using any side information (though our approach can easily incorporate side information).

Matrix completion may also be posed as predicting edge weights in bipartite graphs [Berg et al., 2017; Monti et al., 2017]. This approach builds on recent work applying convolutional neural networks to graph-structured data [Scarselli et al., 2009; Bruna et al., 2014; Duvenaud et al., 2015; Defferrard et al., 2016; Kipf and Welling, 2016; Hamilton et al., 2017a]. As we saw, parameter sharing in graph convolution is closely related to parameter sharing in exchangeable matrices, and indeed it is a special case where $w_2 = w_3$ in Equation 2.2. The key difference is that our exchangeable matrix layer is exactly equivariant to all permutations of rows and columns, while the graph convolution layer is equivariant to all permutations within the automorphism group of the graph [Ravanbakhsh et al., 2017]. In the original graph convolution paper and its extensions, a number of variations have been proposed for how to aggregate information across nodes in a graph. For example, node representations may be averaged with their neighbors [Kipf and Welling, 2016] or concatenated with their neighbors [Hamilton et al., 2017a]. These variations are equivalent to choices about which of the terms of Equation 2.4 to include in our layer, though we are not aware of any prior work that includes all five terms. This lets us reinterpret these aggregation techniques as constraining a

In another related work, Monti et al. [2017] pose the problem of matrix completion as one of deep learning on graph-structured data. Weighted, undirected nearest neighbour graphs are constructed over the space of user and item features. For example, the column graph representing users could come from information in a social network, and the row graph representing items could be based on item similarities. Two architectures are considered. The first acts on the full data matrix X, and the second acts on a factorized version of the form $X = WH^T$, where W and H are $N \times R$ and $M \times R$ matrices with $R \ll \min(N, M)$. This form reduces the learning complexity from $O(NM)$ for the full model to $O(N + M)$ for the factorized model.

Dataset	Users	Items	Ratings	Sparsity
MovieLens 100K	943	1682	100,000	6.30%
MovieLens 1M	6040	3706	1,000,209	4.47%
Flixster	3000	3000	26173	0.291%
Douban	3000	3000	136891	1.521%
Yahoo Music	3000	3000	5335	0.059%

Table 2.1: Number of users, items and ratings for the data sets we used in our experiments. For MovieLens we used the standard MovieLens data sets [Harper and Konstan, 2015], while for Flixster, Douban and Yahoo Music we used the 3000×3000 submatrix presented by Monti et al. [2017] so that we could compare to the results given in their work.

Both architectures consist of two components, first a (Multi-) Graph convolutional network is applied to the rows and columns of the input data. For each element of the input matrix, this module outputs a q-dimensional vector of spacial features. This is fed to a standard LSTM [Hochreiter and Schmidhuber, 1997] to extract temporal features.

2.4 Empirical Results

For reproducibility we have released Tensorflow and Pytorch implementations of our model.[3] Subsection 2.4.1 reports experimental results in the standard transductive (matrix interpolation) setting. However, more interesting results are reported in Subsection 2.4.2, where we test a trained deep model on a completely different dataset. Because of the compute costs associated with these experiments, error bars are not included; in practice we found performance was consistent across random seeds for any particular hyperparameter configuration.

2.4.1 Transductive Setting (Matrix Interpolation)

Here, we demonstrate that exploiting the permutation equivariant structure of the exchangeable matrices allows us to achieve results competitive with state-of-the-

[3]Tensorflow: https://github.com/mravanba/deep_exchangeable_tensors. Pytorch: https://github.com/jhartford/AutoEncSets.

art, while maintaining a constant number of parameters. Note that the number of parameters in all the competing methods grow with N and/or M.

In Table 2.2 we report that the self-supervised exchangeable model is able to achieve state of the art performance on MovieLens-100K dataset. For MovieLens-1M dataset, we cannot fit the whole dataset into GPU memory for training and therefore use conditional subsampling. Our results on this dataset are summarized in Table 2.3. On this larger dataset both models gave comparatively weaker performance than what we observed on the smaller ML-100k dataset and in the extrapolation results. We suspect this is largely due to memory constraints: there is a trade-off between the size of the model (in terms of number of layers and units per layer) and the batch size one can train. We found that both larger batches and deeper models tended to offer better performance, but on these larger datasets it is not currently possible to have both. The results for the factorized exchangeable autoencoder architecture are similar and reported in the same table.

Table 2.2: MovieLens 100K

Model	**ML-100K**
MC [Candès and Recht, 2009]	0.973
GMC [Kalofolias et al., 2014]	0.996
GRALS [Rao et al., 2015]	0.945
sRGCNN [Monti et al., 2017]	0.929
Factorized EAE (ours)	0.920
GC-MC [Berg et al., 2017]	**0.910**
Self-Supervised Model (ours)	**0.910**

Table 2.3: MovieLens 1 million

Model	**ML-1M**
PMF [Mnih and Salakhutdinov, 2008]	0.883
Self-Supervised Model (ours)	0.863
Factorized EAE (ours)	0.860
I-RBM [Salakhutdinov et al., 2007]	0.854
BiasMF [Koren et al., 2009]	0.845
NNMF [Dziugaite and Roy, 2015]	0.843
LLORMA-Local [Lee et al., 2013]	0.833
GC-MC [Berg et al., 2017]	0.832
I-AUTOREC [Sedhain et al., 2015]	0.831
CF-NADE [Zheng et al., 2016]	**0.829**

(Left) Comparison of RMSE scores for the MovieLens-100k dataset, based on the canonical 80/20 training/test split. *(Right)* Comparison of RMSE scores for the MovieLens-1M dataset on random 90/10 training/test split. Baseline numbers are taken from [Berg et al., 2017].

2.4.2 Inductive Setting (Matrix Extrapolation)

Because our model does not rely on any per-user or per-movie parameters, it should be able to generalize to new users and movies that were not present during training. We tested this by training an exchangeable factorized autoencoder on the MovieLens-100k dataset and then evaluating it on a subsample of data from the MovieLens-1M dataset. At test time, the model was shown a portion of the new ratings and then made to make predictions on the remaining ratings.

Figure 2.5 summarizes the results where we vary the amount of data shown to the model from 5% of the new ratings up to 95% and compare against k-nearest neighbors approaches. Our model consistently outperforms the baselines in this task and performance degrades gracefully as we reduce the amount of data observed.

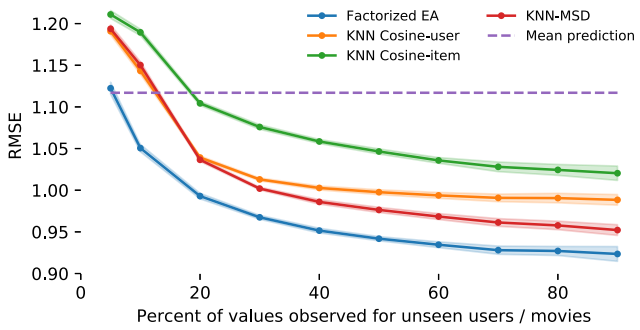

Figure 2.5: Evaluation of our model's ability to generalize. We trained on ML-100k and evaluated on a random subsets of the ML-1M data. At evaluation time, p% of the ML-1M data was treated as *observed* and the model was required to complete the remaining $(1-p)$% (p varied from 5% to 95%). The model outperforms nearest-neighbour approaches for all values of p and degrades gracefully to the mean prediction in the small data case.

Extrapolation to new datasets Perhaps most surprisingly, we were able to achieve competitive results when training and testing on completely disjoint datasets. For this experiment we stress-tested our model's inductive ability by testing how it generalizes to new datasets *without retraining*. We used a Factorized Exchangeable Autoencoder that was trained to make predictions on the MovieLens-100k dataset

and tasked it with making predictions on the Flixster, Douban and YahooMusic datasets. We then evaluated its performance against models trained for each of these individual datasets. All the datasets involve rating prediction tasks, so they share some similar semantics with MovieLens, but they have different user bases and (in the case of YahooMusic) deal with music not movie ratings, so we may expect some change in the rating distributions and user-item interactions. Furthermore, the Flixster ratings are in 0.5 point increments from $1-5$ and YahooMusic allows ratings from $1-100$, while Douban and MovieLens ratings are on $1-5$ scale. To account for the different rating scales, we simply binned the inputs to our model to a $1-5$ range and, where applicable, linearly re-scaled the output before comparing it to the true rating[4]. Despite all of this, Table 2.4 shows that our model achieves very competitive results with models that were trained for the task.

For comparison, we also include the performance of a Factorized EAE trained on the respective datasets. This improves performance of our model over previous state of the art results on the Flixster and YahooMusic datasets and gives very similar performance to Berg et al. [2017]'s GC-MC model on the Douban dataset.

2.4.3 Comparison of sampling procedures

We evaluated the effect of the subsampling the input matrix X on performance for the MovieLens-100k dataset using techniques described in Section 2.3.3. The results are summarized in Figure 2.6. The two key findings are: I) even with large batch sizes of 20 000 examples, performance for both sampling methods is diminished compared to the full batch case. We suspect that our models' weaker results on the larger ML-1M dataset may be partially attributed to the need to subsample. II) the conditional sampling method was able to recover some of the diminished performance. We believe it is likely that more sophisticated sampling schemes that explicitly take into account the dependence between hidden nodes will lead to better performance but we leave that to future work.

[4]Because of this binning procedure, our model received input data that is considerably coarser grained than what was used for the comparison models.

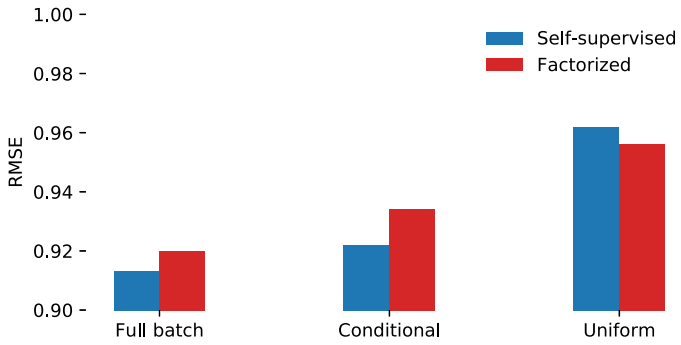

Figure 2.6: Performance difference between sampling methods on the ML-100k. The two sampling methods use mini-batches of size 20 000, while the full batch method used all 75 000 training examples. Note that the y-axis does not begin at 0.

Model	**Flixster**	**Douban**	**YahooMusic**	**Netflix**
GRALS [Rao et al., 2015]	1.313	0.833	38.0	-
sRGCNN [Monti et al., 2017]	1.179	0.801	22.4	-
GC-MC [Berg et al., 2017]	0.941	**0.734**	20.5	-
Factorize EAE (ours)	**0.908**	0.738	**20.0**	-
Factorize EAE (trained on ML100k)	0.987	0.766	23.3	0.918
Netflix Challenge Baseline	-	-	-	0.951
PMF [Mnih and Salakhutdinov, 2008]	-	-	-	0.897
LLORMA-Local [Lee et al., 2013]	-	-	-	0.834
I-AUTOREC [Sedhain et al., 2015]	-	-	-	0.823
CF-NADE [Zheng et al., 2016]	-	-	-	**0.803**

Table 2.4: Evaluation of our model's ability to generalize across datasets. We trained a factorized model on ML100k and then evaluated it on four new datasets. Results for the smaller datasets are from [Berg et al., 2017].

2.5 Extension to Tensors

In this section we generalize the exchangeable matrix layer to higher-dimensional arrays (tensors) and formalize its optimal qualities. Suppose $X \in \mathbb{R}^{N_1 \times \ldots \times N_D}$ is our D-dimensional data tensor, and $\mathrm{vec}(X)$ its vectorized form. We index $\mathrm{vec}(X)$ by tuples $(n_1, n_2, \ldots, n_D)$, corresponding to the original dimensions of X. The precise method of vectorization is irrelevant, provided it is used consistently. Let $\mathcal{N} = \prod_i N_i$ and let $\underline{n} = (n_i, n_{-i})$ be an element of $N_1 \times \ldots \times N_D$ such that n_i is the value of the i-th

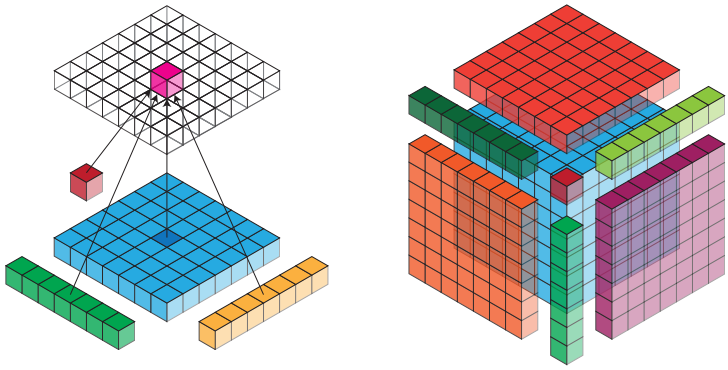

Figure 2.7: Pooling structure implied by the tied weights for matrices (left) and 3D tensors (right). The pink cube highlights one element of the output. It is calculated as a function of the corresponding element from the input (dark blue), pooled aggregations over the rows and columns of the input (green and yellow), and pooled aggregation over the whole input matrix (red). In the tensor case (right), we pool over all sub-tensors (orange and purple sub-matrices, green sub-vectors and red scalar). For clarity, the output connections are not shown in the tensor case.

entry of $\underline{n}$, and n_{-i} the values of the remaining entries (where it is understood that the ordering of the elements of $\underline{n}$ is unchanged). We seek a layer that is equivariant only to certain, meaningful, permutations of vec(X). This motivates our definition of an exchangeable tensor layer in a manner that is completely analogous to Definition 2.2.1 for matrices.

For any positive integer N, let $\mathcal{S}^{(N)}$ denote the symmetric group of all permutations of N objects. Then $\mathcal{S}^{(N_1)} \times ... \times \mathcal{S}^{(N_D)}$ refers to the product group of all permutations of N_1 through N_D objects, while $\mathcal{S}^{(\mathcal{N})}$ refers to the group of all permutations of $\mathcal{N} = \prod_i N_i$ objects. So $\mathcal{S}^{(N_1)} \times ... \times \mathcal{S}^{(N_D)}$ is a proper subgroup of $\mathcal{S}^{(\mathcal{N})}$ having $\prod_i (N_i!)$ members, compared to $(\prod_i N_i)!$ members in $\mathcal{S}^{(\mathcal{N})}$. We want a layer that is equivariant to *only* those permutations in $\mathcal{S}^{(N_1)} \times ... \times \mathcal{S}^{(N_D)}$, but not to any other member of $\mathcal{S}^{(\mathcal{N})}$.

Definition 2.5.1 (exchangeable tensor layer)**.** Let $g^{(\mathcal{N})} \in \mathcal{S}^{(N_1)} \times \mathcal{S}^{(N_2)} \times ... \times \mathcal{S}^{(N_D)}$ and $G^{(\mathcal{N})}$ be the corresponding permutation matrix. Then the neural layer $\text{vec}^{-1}(\sigma(W\,\text{vec}(X)))$ with $X \in \mathbb{R}^{N_1 \times ... \times N_D}$ and $W \in \mathbb{R}^{\mathcal{N} \times \mathcal{N}}$ is an exchangeable tensor layer if permuting the elements of the input along any set of axes results in the same

permutation of the output tensor:

$$G^{(\mathcal{N})}\sigma(W\,\mathrm{vec}(X)) = \sigma(WG^{(\mathcal{N})}\,\mathrm{vec}(X)) \quad \forall X, \tag{2.5}$$

and moreover for any other permutation of the elements X (i.e., permutations that are not along axes), there exists an input X for which this equality does not hold.

The following theorem asserts that a simple parameter tying scheme achieves the desired permutation equivariance for tensor-structured data.

Theorem 2. *The layer $Y = \mathrm{vec}^{-1}(\sigma(W\,\mathrm{vec}(X)))$, where σ is any strictly monotonic, element-wise nonlinearity (e.g., sigmoid), is an exchangeable tensor layer iff the parameter matrix $W \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ is defined as follows.*

For each $\mathbb{S} \subseteq [D] = \{1,\ldots,D\}$, define a distinct parameter $w_{\mathbb{S}} \in \mathbb{R}$, and tie the entries of W as follows

$$W_{\underline{n},\underline{n}'} := w_{\mathbb{S}} \quad s.t. \quad n_i = n'_i \iff i \in \mathbb{S}. \tag{2.6}$$

That is, the $(\underline{n},\underline{n}')$-th element of W is uniquely determined by the set of indices at which n and n' are equal.

In the special case that $X \in \mathbb{R}^{N_1\times N_2}$ is a matrix, this gives the formulation of W described in Section 2.2. Theorem 2 says that a layer constructed in this manner is equivariant with respect to *only* those permutations of $\mathcal{N}$ objects that correspond to permutations of sub-tensors along the D dimensions of the input. The proof is in the Appendix. Equivariance with respect to permutations in $\mathcal{S}^{(N_1)}\times\mathcal{S}^{(N_2)}\times\ldots\times\mathcal{S}^{(N_D)}$ follows as a simple corollary of Theorem 2.1 in [Ravanbakhsh et al., 2017]. Non-equivariance with respect to other permutations follows from the following Propositions.

Proposition 3. *Let $g^{(\mathcal{N})} \in \mathcal{S}^{(\mathcal{N})}$ be an "illegal" permutation of elements of the tensor X – that is $g^{(\mathcal{N})} \notin \mathcal{S}^{(N_1)}\times\mathcal{S}^{(N_2)}\times\ldots\times\mathcal{S}^{(N_D)}$. Then there exists a dimension $i \in [D]$ such that, for some $n_i, n'_i, n_{-i}, n'_{-i}$:*

$$g^{(\mathcal{N})}((n_i, n_{-i})) = (n'_i, n_{-i}), \quad but$$
$$g^{(\mathcal{N})}((n_i, n'_{-i})) \neq (n'_i, n'_{-i}).$$

If we consider the sub-tensor of X obtained by fixing the value of the i-th dimension to n_i, we expect a "legal" permutation to move this whole subtensor to some n'_i (it could additionally shuffle the elements within this subtensor.) This Proposition asserts that an "illegal" permutation $g^{(\mathcal{N})} \notin \mathcal{S}^{(N_1)} \times \mathcal{S}^{(N_2)} \times ... \times \mathcal{S}^{(N_D)}$ is guaranteed to violate this constraint for some dimension/subtensor combination. The next proposition asserts that if we can identify such inconsistency in permutation of sub-tensors then we can identify and entry in $G^{(\mathcal{N})}W$ that will differ from $WG^{(\mathcal{N})}$, and therefore for some input tensor X, the equivariance property is violated – i.e., $G^{(\mathcal{N})}\sigma(W\,\mathrm{vec}(X)) \neq \sigma(WG^{(\mathcal{N})}\,\mathrm{vec}(X))$.

Proposition 4. *Let $g^{(\mathcal{N})} \in \mathcal{S}^{(\mathcal{N})}$ with $G^{(\mathcal{N})} \in \{0,1\}^{\mathcal{N}\times\mathcal{N}}$ the corresponding permutation matrix. Suppose $g^{(\mathcal{N})} \notin \mathcal{S}^{(N_1)} \times \mathcal{S}^{(N_2)} \times ... \times \mathcal{S}^{(N_D)}$, and let $W \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ be as defined above. If there exists an $i \in [D]$, and some $n_i, n'_i, n_{-i}, n'_{-i}$ such that*

$$g^{(\mathcal{N})}((n_i, n_{-i})) = (n'_i, n_{-i}), \quad \textit{and}$$
$$g^{(\mathcal{N})}((n_i, n'_{-i})) \neq (n'_i, n'_{-i}),$$

then

$$(G^{(\mathcal{N})}W)_{(n'_i,n'_{-i}),(n_i,n_{-i})} \neq (WG^{(\mathcal{N})})_{(n'_i,n'_{-i}),(n_i,n_{-i})}$$

Proposition 4 makes explicit a particular element at which the products $G^{(N)}W$ and $WG^{(N)}$ will differ, provided $g^{(\mathcal{N})}$ is not of the desired form.

Theorem 2 shows that this parameter sharing scheme produces a layer that is equvariant to exactly those permutations we desire, and moreover, it is optimal in the sense that any layer having fewer ties in its parameters (i.e., more parameters) would fail to be equivariant.

Discussion

This chapter considered the problem of predicting relationships between the elements of two or more distinct sets of objects, where the data can also be expressed as an exchangeable matrix or tensor. We introduced a weight tying scheme enabling the application of deep models to this type of data. We proved that our scheme always produces permutation equivariant models and that no increase in model

expressiveness is possible without violating this guarantee. Experimentally, we showed that our models achieve state-of-the-art or competitive performance on widely studied matrix completion benchmarks. Notably, our models achieved this performance despite having a number of parameters independent of the size of the matrix to be completed, unlike *all* other approaches that offer strong performance. Also unlike these other approaches, our models can achieve competitive results for the problem of *matrix extrapolation*: asking an already-trained model to complete a new matrix involving new objects that were unobserved at training time. Finally, we observed that our methods were surprisingly strong on various transfer learning tasks: extrapolating from MovieLens ratings to Flixter, Douban, and YahooMusic ratings. All of these contained different user populations and different distributions over the objects being rated; indeed, in the Yahoo Music dataset, the underlying objects were not even of the same kind as those rated in training data.

Chapter 3

Deep Learning for Predicting Human Strategic Behavior

In this chapter we develop a deep network architecture for predicting the actions of human strategic decision makers in normal-form games. This work preceded my work on deep networks for exchangeable arrays from Chapter 2, but the architecture was designed to capture the same invariances and because permutation invariance significantly constrains the number of free parameters in a network (see Section 2.2), the resulting architecture was very similar to what was presented in Chapter 2. In particular, the network consisted of two components: "feature layers" which are exactly the exchangeable matrix layers presented in Chapter 2, and "action response layers" which were designed to model iterative reasoning from the behavioral game theory literature.

3.1 Introduction

Game theory provides a powerful framework for the design and analysis of multiagent systems that involve strategic interactions [see, e.g., Shoham and Leyton-Brown, 2008]. Prominent examples of such systems include search engines, which use advertising auctions to generate a significant portion of their revenues and rely on game theoretic reasoning to analyze and optimize these mechanisms [Edelman et al., 2007; Varian, 2007]; spectrum auctions, which rely on game theoretic analysis

to carefully design the "rules of the game" in order to coordinate the reallocation of valuable radio spectrum [Milgrom and Segal, 2014]; and security systems, which analyze the allocation of security personnel as a game between rational adversaries in order to optimize their use of scarce resources [Tambe, 2011]. In such applications, system designers optimize their choices with respect to assumptions about the preferences, beliefs and capabilities of human players [Parkes and Wellman, 2015]. A standard game theoretic approach is to assume that players are perfectly rational expected utility maximizers and indeed, that they have common knowledge of this. In some applications, such as the high-stakes spectrum auctions just mentioned, this assumption is probably reasonable, as participants are typically large companies that hire consultants to optimize their decision making. In other scenarios that allow less time for planning or involve less sophisticated participants, the perfect rationality assumption may lead to suboptimal system designs. For example, Yang et al. [2013] were able to improve the performance of systems that defend against adversaries in security games by relaxing the perfect rationality assumption. Of course, relaxing this assumption means finding something else to replace it with: an accurate model of boundedly rational human behavior.

The behavioral game theory literature has developed a wide range of models for predicting human behavior in strategic settings by incorporating cognitive biases and limitations derived from observations of play and insights from cognitive psychology [Camerer, 2003]. Like much previous work, we study the unrepeated, simultaneous-move setting, for two reasons. First, the setting is conceptually straightforward: games can be represented in a so-called "normal form", simply by listing the utilities to each player for each combination of their actions (e.g., see Figure 3.1). Second, the setting is surprisingly general: auctions, security systems, and many other interactions can be modeled naturally as normal form games. The most successful predictive models for this setting combine notions of iterative reasoning and noisy best response [Wright and Leyton-Brown, 2010] and use hand-crafted features to model the behavior of non-strategic players [Wright and Leyton-Brown, 2014].

The success of deep learning has demonstrated that feature engineering can be dispensed with, by fitting flexible models that are capable of learning novel representations, but a key feature in successful deep models is the use of careful design choices to encode "*basic* domain knowledge of the input, in particular its

topological structure... to *learn* better features" [Bengio et al., 2013, emphasis original]. For example, feed-forward neural nets can, in principle, represent the same functions as convolution networks, but the latter tend to be more effective in vision applications because they encode the prior that low-level features should be derived from the pixels within a small neighborhood and that predictions should be invariant to small input translations. Our work seeks to do the same for the behavioral game theory setting, identifying novel architectures that extend deep learning to predicting behavior in strategic scenarios encoded as two player, normal-form games.

A key property required of such a model is the ability to generalize across game size: a model must be able to take as input an $m \times n$ bimatrix game (i.e., two $m \times n$ matrices encoding the payoffs of players 1 and 2 respectively) and output an m-dimensional probability distribution over player 1's actions, for arbitrary values of n and m, including values that did not appear in training data. In contrast, existing deep models typically assume either a fixed-dimensional input or an arbitrary-length sequence of fixed-dimensional inputs, in both cases with a fixed-dimensional output. To achieve this, we leverage knowledge that permuting rows and columns in the input (i.e., changing the order in which actions are presented to the players) does not change the strategic properties of the game, so it should not change a model's output beyond a corresponding permutation. In Section 3.3, we present an architecture that operates on matrices using scalar weights to capture invariance to changes in the size of the input matrices and to permutations of its rows and columns. In Section 3.4 we evaluate our model's ability to predict distributions of play given normal form descriptions of games on a dataset of experimental data from a variety of experiments, and find that our feature-free deep learning model significantly exceeds the performance of the current state-of-the-art model, which has access to hand-tuned features based on expert knowledge [Wright and Leyton-Brown, 2014].

3.2 Related Work

Prediction in normal form games. The task of predicting actions in normal form games has been studied mostly in the behavioral game theory literature. Such models

	L	C	R
T	**10,10**	**3, 5**	**18, 8**
M	**5, 3**	**20,20**	**0, 25**
B	**8, 18**	**25, 0**	**15, 15**

Figure 3.1: An example 3×3 normal form game. The row player chooses from actions $\{T, M, B\}$ and the column player chooses from actions $\{R, C, L\}$. If the row player played action T and column player played action C, their resulting payoffs would be 3 and 5 respectively. Given such a matrix as input we aim to predict a distribution over the row player's choice of actions defined by the observed frequency of actions shown on the right.

tend to have few parameters and to aim to describe previously identified cognitive processes. Two key ideas are the relaxation of best response to "quantal response" and the notion of "limited iterative strategic reasoning". Models that assume *quantal response* assume that players select actions with probability increasing in expected utility instead of always selecting the action with the largest expected utility [McKelvey and Palfrey, 1995]. This is expressed formally by assuming that players select actions, a_i, with probability, s_i, given by the *logistic quantal response function* $s_i(a_i) = \frac{\exp(\lambda u_i(a_i, s_{-i}))}{\sum_{a_i'} \exp(\lambda u_i(a_i', s_{-i}))}$. This function is equivalent to the familiar softmax function with an additional scalar sharpness parameter λ that allows the function to output the best response as $\lambda \to \infty$ and the uniform distribution as $\lambda \to 0$. This relaxation is motivated by the behavioral notion that if two actions have similar expected utility then they will also have similar probability of being chosen.

Iterative strategic reasoning means that players perform a bounded number of steps of reasoning in deciding on their actions, rather than always converging to fixed points as in classical game theory. Models incorporating this idea typically assume that every agent has an integer *level*. Non-strategic, "level-0" players choose actions uniformly at random; level-k players best respond to the level-$(k-1)$ players [Costa-Gomes et al., 2001] or to a mixture of levels between level-0 and level-

$(k-1)$ [Camerer et al., 2004]. The two ideas can be combined, allowing players to quantally respond to lower level players [Stahl and Wilson, 1994; Wright and Leyton-Brown, 2012]. Because iterative reasoning models are defined recursively starting from a base-case of level-0 behavior, their performance can be improved by better modeling the non-strategic level-0 players. Wright and Leyton-Brown [2014] combine quantal response and bounded steps of reasoning with a model of non-strategic behavior based on hand-crafted game theoretic features. To the best of our knowledge, this is the current state-of-the-art model.

Deep learning. Deep learning has demonstrated much recent success in solving supervised learning problems in vision, speech and natural language processing [see, e.g., LeCun et al., 2015; Schmidhuber, 2015]. By contrast, there have been relatively few applications of deep learning to multiagent settings. Notable exceptions are Clark and Storkey [2015] and the policy network used in Silver et al. [2016]'s work in predicting the actions of human players in Go. Their approach is similar in spirit to ours: they map from a description of the Go board at every move to the choices made by human players, while we perform the same mapping from a normal form game. The setting differs in that Go is a single, sequential, zero-sum game with a far larger, but fixed, action space, which requires an architecture tailored for pattern recognition on the Go board. In contrast, we focus on constructing an architecture that generalizes across general-sum, normal form games.

3.3 Modeling Human Strategic Behavior with Deep Networks

A natural starting point in applying deep networks to a new domain is testing the performance of a regular feed-forward neural network. To apply such a model to a normal form game, we need to flatten the utility values into a single vector of length $mn + nm$ and learn a function that maps to the m-simplex output via multiple hidden layers. Feed-forward networks cannot handle size-invariant inputs, but we can temporarily set that problem aside by restricting ourselves to games with a fixed input size. We experimented with that approach and found that feed-forward networks often generalized poorly as the network overfitted the training data (see

Section B.2 of the supplementary material for experimental evidence). One way of combating overfitting is to encourage invariance through data augmentation: for example, one may augment a dataset of images by rotating, shifting and scaling the images slightly. In games, a natural simplifying assumption is that players are indifferent to the order in which actions are presented, implying invariance to permutations of the payoff matrix.[1] Incorporating this assumption by randomly permuting rows or columns of the payoff matrix at every epoch of training dramatically improved the generalization performance of a feed-forward network in our experiments, but the network is still limited to games of the size that it was trained on.

Our approach is to enforce this invariance in the model architecture rather than through data augmentation. We then add further flexibility by incorporating iterative response ideas inspired by behavioral game theory models. The result is a model that is flexible enough to represent all the models surveyed in Wright and Leyton-Brown [2012] and Wright and Leyton-Brown [2014]—and a huge space of novel models as well—and which can be identified automatically. The model is also invariant to the size of the input payoff matrix, differentiable end to end and trainable using standard gradient-based optimization.

The model has two parts: *feature layers* and *action response layers*; see Figure 4.2 for a graphical overview. The feature layers take the row and column player's normalized utility matrices $\mathbf{U}^{(r)}$ and $\mathbf{U}^{(c)} \in \mathbb{R}^{m \times n}$ as input, where the row player has m actions and the column player has n actions. The feature layers consist of multiple levels of *hidden matrix units*, $\mathbf{H}^{(r)}_{i,j} \in \mathbb{R}^{m \times n}$, each of which calculates a weighted sum of the units below and applies a non-linear activation function. Each layer of hidden units is followed by *pooling units*, which output aggregated versions of the hidden matrices to be used by the following layer. After multiple layers, the matrices are aggregated to vectors and normalized to points on the $m-$simplex, $\Delta^m := \{x \in \mathbb{R}^m : \sum_i x_i = 1, x_i \geq 0\}$, to define a distribution over actions, $\mathbf{f}^{(r)}_i \in \Delta^m$ in *softmax* units. We refer to these distributions as *features* because they encode higher-level representations of the input matrices that may be combined to construct the output distribution.

[1] We thus ignore salience effects that could arise from action ordering.

Iterative strategic reasoning is an important phenomenon in human decision making; we thus want to allow our models the option of incorporating such reasoning. To do so, we compute features for the column player in the same manner by applying the feature layers to the transpose of the input matrices, which outputs $\mathbf{f}_i^{(c)} \in \Delta^n$. Each action response layer for a given player then takes the opposite player's preceding action response layers as input and uses them to construct distributions over the respective players' outputs. The final output $\mathbf{y} \in \Delta^m$ is a weighted sum of all action response layers' outputs.

3.3.1 Equivariant Hidden Units

We build a model that ties parameters in our network by encoding the assumption that players reason about each action identically. This assumption implies that the row player applies the same function to each row of a given game's utility matrices. Thus, in a normal form game represented by the utility matrices $\mathbf{U}^{(r)}$ and $\mathbf{U}^{(c)}$, the weights associated with each row of $\mathbf{U}^{(r)}$ and $\mathbf{U}^{(c)}$ must be the same. Similarly, the corresponding assumption about the column player implies that the weights associated with each column of $\mathbf{U}^{(r)}$ and $\mathbf{U}^{(c)}$ must also be the same. We can satisfy both assumptions by applying a single scalar weight to each of the utility matrices, computing $w_r\mathbf{U}^{(r)} + w_c\mathbf{U}^{(c)}$. This idea can be generalized as in a standard feed-forward network to allow us to fit more complex functions. A hidden matrix unit taking all the preceding hidden matrix units as input can be calculated as

$$\mathbf{H}_{l,i} = \phi\left(\sum_j w_{l,i,j}\mathbf{H}_{l-1,j} + b_{l,i}\right) \quad \mathbf{H}_{l,i} \in \mathbb{R}^{m\times n},$$

where $\mathbf{H}_{l,i}$ is the i^{th} hidden unit matrix for layer l, $w_{l,i,j}$ is the j^{th} scalar weight, $b_{l,i}$ is a scalar bias variable, and ϕ is a non-linear activation function applied element-wise. Notice that, as in a traditional feed-forward neural network, the output of each hidden unit is simply a nonlinear transformation of the weighted sum of the preceding layer's hidden units. Our architecture differs by maintaining a matrix at each hidden unit instead of a scalar. So while in a traditional feed-forward network each hidden unit maps the previous layer's vector of outputs into a scalar output, in

our architecture each hidden unit maps a tensor of outputs from the previous layer into a matrix output.

Tying weights in this way reduces the number of parameters in our network by a factor of nm, offering two benefits. First, it reduces the degree to which the network is able to overfit; second and more importantly, it makes the model invariant to the size of the input matrices. To see this, notice that each hidden unit maps from a tensor containing the k output matrices of the preceding layer in $\mathbb{R}^{k\times m\times n}$ to a matrix in $\mathbb{R}^{m\times n}$ using k weights. Thus our number of parameters in each layer depends on the number of hidden units in the preceding layer, but not on the sizes of the input and output matrices. This allows the model to generalize to input sizes that do not appear in training data.

3.3.2 Pooling units

A limitation of the weight tying used in our hidden matrix units is that it forces independence between the elements of their matrices, preventing the network from learning functions that compare the values of related elements (see Figure 3.2 (left)). Recall that each element of the matrices in our model corresponds to an outcome in a normal form game. We would like our model to be able to compare is the set of payoffs associated with each of the players' actions that led to that outcome. This corresponds to the row and column of each matrix associated with the particular element.

This observation motivates our pooling units, which allow information sharing by outputting aggregated versions of their input matrix that may be used by later layers in the network to learn to compare the values of a particular cell in a matrix and its row- or column-wise aggregates.

$$\mathbf{H} \to \{\mathbf{H}_c, \mathbf{H}_r\} = \left\{ \begin{pmatrix} \max_i h_{i,1} & \max_i h_{i,2} & \dots \\ \max_i h_{i,1} & \max_i h_{i,2} & \dots \\ \vdots & \vdots & \\ \max_i h_{i,1} & \max_i h_{i,2} & \end{pmatrix}, \begin{pmatrix} \max_j h_{1,j} & \max_j h_{1,j} & \dots \\ \max_j h_{2,j} & \max_j h_{2,j} & \dots \\ \vdots & \vdots & \\ \max_j h_{m,j} & \max_j h_{m,j} & \dots \end{pmatrix} \right\} \tag{3.1}$$

A pooling unit takes a matrix as input and outputs two matrices constructed from row- and column-preserving pooling operations respectively. A pooling operation

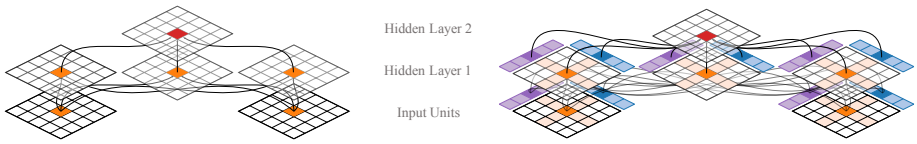

Figure 3.2: *Left:* Without pooling units, each element of every hidden matrix unit depends only on the corresponding elements in the units from the layer below; e.g., the middle element highlighted in red depends only on the value of the elements of the matrices highlighted in orange. *Right:* With pooling units at each layer in the network, each element of every hidden matrix unit depends both on the corresponding elements in the units below *and* the pooled quantity from each row and column. E.g., the light blue and purple blocks represent the row and column-wise aggregates corresponding to their adjacent matrices. The dark blue and purple blocks show which of these values the red element depends on. Thus, the red element depends on both the dark- and light-shaded orange cells.

could be any permutation invariant function that maps from $\mathbb{R}^n \to \mathbb{R}$. We use the *max* function because it is necessary to represent known behavioral functions (see Section 4 of the supplementary material for details) and offered the best empirical performance of the functions we tested. Equation (3.1) shows an example of a pooling layer with *max* functions for some arbitrary matrix $\mathbf{H}$. The first of the two outputs, $\mathbf{H}_c$, is column-preserving in that it selects the maximum value in each column of $\mathbf{H}$ and then stacks the resulting vector n-dimensional vector m times such that the dimensionality of $\mathbf{H}$ and $\mathbf{H}_c$ are the same. Similarly, the row-preserving output constructs a vector of the max elements in each column and stacks the resulting m-dimensional vector n times such that $\mathbf{H}_r$ and $\mathbf{H}$ have the same dimensionality. We stack the vectors that result from the pooling operation in this fashion so that the hidden units from the next layer in the network may take $\mathbf{H}, \mathbf{H}_c$ and $\mathbf{H}_r$ as input. This allows these later hidden units to learn functions where each element of their output is a function both of the corresponding element from the matrices below as well as their row and column-preserving maximums (see Figure 3.2 (right)).

3.3.3 Output distribution

Our model predicts a distribution over the row player's actions. In order to do this, we need to map from the hidden matrices in the final layer, $\mathbf{H}_{L,i} \in R^{m \times n}$, of the network onto a point on the m-simplex, Δ^m. We achieve this mapping by applying a row-preserving sum to each of the final layer hidden matrices $\mathbf{H}_{L,i}$ (i.e. we sum uniformly over the columns of the matrix as described above) and then applying a softmax function to convert each of the resulting vectors $\mathbf{h}_i$ into normalized distributions. This produces k features $\mathbf{f}_i$, each of which is a distribution over the row player's m actions:

$$\mathbf{f}_i = \text{softmax}\left(\mathbf{h}^{(i)}\right) \quad \text{where } \mathbf{h}_j^{(i)} = \sum_{k=1}^{n} h_{j,k}^{(i)} \text{ for all } j \in \{1,...,m\}, h_{j,k}^{(i)} \in \mathbf{H}^{(i)} \quad i \in \{1,...,k\}.$$

We can then produce the output of our features, $\mathbf{ar}_0$, using a weighted sum of the individual features, $\mathbf{ar}_0 = \sum_{i=1}^{k} w_i \mathbf{f}_i$, where we optimize w_i under simplex constraints, $w_i \geq 0$, $\sum_i w_i = 1$. Because each $\mathbf{f}_i$ is a distribution and our weights w_i are points on the simplex, the output of the feature layers is a mixture of distributions.

3.3.4 Action Response Layers

The feature layers described above are sufficient to meet our objective of mapping from the input payoff matrices to a distribution over the row player's actions. However, this architecture is not capable of explicitly representing iterative strategic reasoning, which the behavioral game theory literature has identified as an important modeling ingredient. We incorporate this ingredient using action response layers: the first player can respond to the second's beliefs, the second can respond to this response by the first player, and so on to some finite depth. The proportion of players in the population who iterate at each depth is a parameter of the model; thus, our architecture is also able to learn not to perform iterative reasoning.

More formally, we begin by denoting the output of the feature layers as $\mathbf{ar}_0^{(r)} = \sum_{i=1}^{k} w_{0i}^{(r)} \mathbf{f}_i^{(r)}$, where we now include an index (r) to refer to the output of *row* player's action response layer $\mathbf{ar}_0^{(r)} \in \Delta^m$. Similarly, by applying the feature layers to a transposed version of the input matrices, the model also outputs a corresponding $\mathbf{ar}_0^{(c)} \in \Delta^n$ for the column player which expresses the row player's beliefs about

which actions the column player will choose. Each action response layer composes its output by calculating the expected value of an internal representation of utility with respect to its belief distribution over the opposition actions. For this internal representation of utility, we chose a weighted sum of the final layer of the hidden layers, $\sum_i w_i \mathbf{H}_{L,i}$, because each $\mathbf{H}_{L,i}$ is already some non-linear transformation of the original payoff matrix, and so this allows the model to express utility as a transformation of the original payoffs. Given the matrix that results from this sum, we can compute expected utility with respect to the vector of beliefs about the opposition's choice of actions, $\mathbf{ar}_j^{(c)}$, by simply taking the dot product of the weighted sum and beliefs. When we iterate this process of responding to beliefs about one's opposition more than once, higher-level players will respond to beliefs, $\mathbf{ar}_i$, for all i less than their level and then output a weighted combination of these responses using some weights, $v_{l,i}$. Putting this together, the l^{th} action response layer for the *row* player (r) is defined as

$$\mathbf{ar}_l^{(r)} = \text{softmax}\left(\lambda_l\left(\sum_{j=0}^{l-1} v_{l,j}^{(r)}\left(\sum_{i=1}^{k} w_{l,i}^{(r)}\mathbf{H}_{L,i}^{(r)}\right)\cdot \mathbf{ar}_j^{(c)}\right)\right), \quad \mathbf{ar}_l^{(r)} \in \Delta^m, l \in \{1, ..., K\},$$

where l indexes the action response layer, λ_l is a scalar sharpness parameter that allows us to sharpen the resulting distribution, $w_{l,i}^{(r)}$ and $v_{l,j}^{(r)}$ are scalar weights, $\mathbf{H}_{L,i}$ are the *row* player's k hidden units from the final hidden layer L, $\mathbf{ar}_j^{(c)}$ is the output of the *column* player's j^{th} action response layer, and K is the total number of action response layers. We constrain $w_{li}^{(r)}$ and $v_{lj}^{(r)}$ to the simplex and use λ_l to sharpen the output distribution so that we can optimize the sharpness of the distribution and relative weighting of its terms independently. We build up the column player's action response layer, $\mathbf{ar}_l^{(c)}$, similarly, using the column player's internal utility representation, $\mathbf{H}_{L,i}^{(c)}$, responding to the row player's action response layers, $\mathbf{ar}_l^{(r)}$. These layers are not used in the final output directly but are relied upon by subsequent action response layers of the row player.

Output Our model's final output is a weighted sum of the outputs of the action response layers. This output needs to be a valid distribution over actions. Because each of the action response layers also outputs a distribution over actions, we can

achieve this requirement by constraining these weights to the simplex, thereby ensuring that the output is just a mixture of distributions. The model's output is thus $\mathbf{y} = \sum_{j=1}^{K} w_j \mathbf{ar}_j^{(r)}$, where $\mathbf{y}$ and $\mathbf{ar}_j^{(r)} \in \Delta^m$, and $w_j \in \Delta^K$.

3.3.5 Representational generality of our architecture

Our work aims to extend existing models in behavioral game theory via deep learning, not to propose an orthogonal approach. Thus, we must demonstrate that our representation is rich enough to capture models and features that have proven important in that literature. We omit the details here for space reasons (see the supplementary material, Section B.1), but summarize our findings. Overall, our architecture can express the *quantal cognitive hierarchy* [Wright and Leyton-Brown, 2014] and *quantal level-k* [Stahl and Wilson, 1994] models and as their sharpness tends to infinity, their best-response equivalents *cognitive hierarchy* [Camerer et al., 2004] and *level-k* [Costa-Gomes et al., 2001]. Using feature layers we can also encode all the behavioral features used in Wright and Leyton-Brown [2014]. However, our architecture is not universal; notably, it is unable to express certain features that are likely to be useful, such as identification of dominated strategies.

3.4 Experiments

Experimental Setup We used a dataset combining observations from 9 human-subject experimental studies conducted by behavioral economists in which subjects were paid to select actions in normal-form games. Their payment depended on the subject's actions and the actions of their unseen opposition who chose an action simultaneously.We are interested in the model's ability to predict the distribution over the row player's action, rather than just its accuracy in predicting the most likely action. As a result, we fit models to maximize the likelihood of training data $\mathbb{P}(\mathcal{D}|\theta)$ (where θ are the parameters of the model and $\mathcal{D}$ is our dataset) and evaluate them in terms of negative log-likelihood on the test set.

All the models presented in the experimental section were optimized using Adam [Kingma and Ba, 2014] with an initial learning rate of 0.0002, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The models were all regularized using Dropout [Srivastava

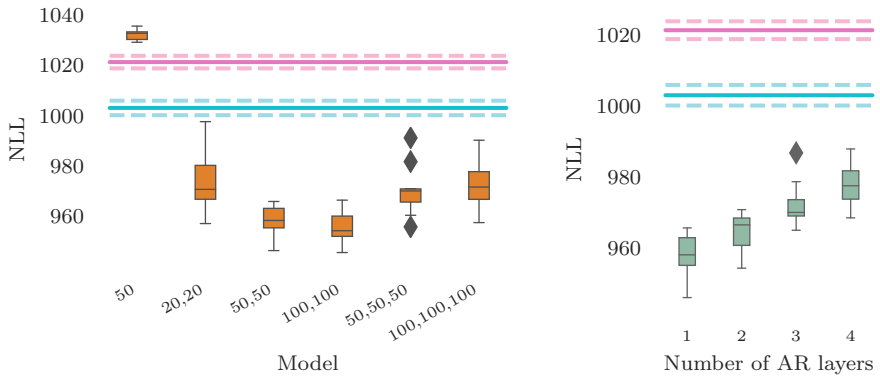

Figure 3.3: Negative Log Likelihood Performance (smaller is better). The error bars represent 95% confidence intervals across 10 rounds of 10-fold cross-validation. We compare various models built using our architecture to QCH Uniform (pink line) and QCH Linear4 (blue line).

et al., 2014] with drop probability of 0.2 and L_1 regularization with parameter 0.01. They were all trained until there was no training set improvement up to a maximum of 25 000 epochs and the parameters from the iteration with the best training set performance was returned. Our architecture imposes simplex constraints on the mixture weight parameters. Fortunately, simplex constraints fall within the class of *simple constraints* that can be efficiently optimized using the projected gradient algorithm [Goldstein, 1964]. The algorithm modifies standard SGD by projecting the relevant parameters onto the constraint set after each gradient update.

Experimental Results Figure 3.3 (left) shows a performance comparison between a model built using our deep learning architecture with only a single action response layer (i.e. no iterative reasoning; details below) and the previous state of the art, quantal cognitive hierarchy (QCH) with hand-crafted features (shown as a blue line); for reference we also include the best feature-free model, QCH with a uniform model of level-0 behavior (shown as a pink line). We refer to an instantiation of our model with L hidden layers and K action response layers as an $L+K$ layer network.

All instantiations of our model with 3 or more layers significantly[2] improved on both alternatives and thus represents a new state of the art. Notably, the magnitude of the improvement was considerably larger than that of adding hand-crafted features to the original QCH model.

Figure 3.3 (left) considers the effect of varying the number of hidden units and layers on performance using a single action response layer. Perhaps unsurprisingly, we found that a two layer network with only a single hidden layer of 50 units performed poorly on both training and test data. Adding a second hidden layer resulted in test set performance that improved on the previous state of the art. For these three layer networks (denoted (20, 20), (50, 50) and (100, 100)), performance improved with more units per layer, but there were diminishing returns to increasing the number of units per layer beyond 50. The four-layer networks (denoted (50, 50, 50) and (100, 100, 100)) offered further improvements in training set performance but test set performance diminished as the networks were able to overfit the data. Thus, our final network contained two layers of 50 hidden units and pooling units.

Our next set of experiments committed to this configuration for feature layers and investigated configurations of action-response layers, varying their number between one and four (i.e., from no iterative reasoning up to three levels of iterative reasoning; see Figure 3.3 (right)). The networks with more than one action-response layer showed signs of overfitting: performance on the training set improved steadily as we added AR layers but test set performance suffered. Thus, our final network used only one action-response layer. We nevertheless remain committed to an architecture that can capture iterative strategic reasoning; we intend to investigate more effective methods of regularizing the parameters of action-response layers in future work.

3.5 Discussion and Conclusions

To design systems that efficiently interact with human players, we need an accurate model of boundedly rational behavior. We present an architecture for learning such models that significantly improves upon state-of-the-art performance without

[2] A one-sided Wilcoxon test to check whether each instantiation is less than QCH Linear4 gave p-values < 0.001 for all the instantiations.

needing hand-tuned features developed by domain experts. Interestingly, while the full architecture can include action response layers to explicitly incorporate the iterative reasoning process modeled by level-k-style models, our best performing model did not need them to set a new performance benchmark. This best performing architecture is equivalent to a composition of the exchangeable matrix layers that we developed in Chapter 2. This is already a very flexible function approximator, so it is likely that the additional flexibility provided by modelling iterative reasoning explicitly was unnecessary to achieve good predictive performance on these benchmarks. Because both components of the architecture—the feature layers and the action response layers—are sufficiently flexible to model the mapping from games to behaviour, we have an identifiability problem where either component could explain the data. To separate these effects, we would need carefully designed experiments that separate the contribution of the "strategic" iterative reasoning modeled by the action response layers and the "nonstrategic" play. Additionally, some natural future directions, are to extend our architecture beyond two-player, unrepeated games to games with more than two players, as well as to richer interaction environments, such as games in which the same players interact repeatedly and games of imperfect information.

Chapter 4

Predicting Propositional Satisfiability via End-to-End Learning

This chapter changes focus from predicting human decision making to predicting algorithmic decision making. It is motivated by the question of whether the function from raw input to satisfiability status is even learnable; *a priori* it is not obvious that NP-complete problems should have sufficient structure that we can learn models that generalize across different sized inputs. In this work we evaluated the architecture developed in Chapter 2 on SAT problems encoded in conjunctive normal form (CNF). The CNF encoding is also an exchangeable matrix so it shares the same structure as the game theory task in Chapter 3, but here we are predicting the outcome of a deterministic algorithmic decision.

4.1 Introduction

NP-complete combinatorial problems are pervasive in many domains, such as planning, scheduling, and networking. The propositional satisfiability (SAT) problem is among the most widely studied of these; indeed, it was the first to be proven NP-complete. It is also used in many applications (e.g., model checking [Clarke et al., 2001] and radio spectrum reallocation [Fréchette et al., 2016]) due both to

its encoding flexibility and the availability of many high-performance solvers. The SAT problem asks whether a given Boolean expression evaluates to true. SAT is typically represented in conjunctive normal form (CNF)–as a conjunction (AND) over clauses, each of which is a disjunction (OR) over Boolean variables, which may optionally be negated. The conjunction and disjunction operators are commutative, so permuting their arguments does not change a problem instance. The two-layer logical structure of CNF is simple to reason about and is used by most SAT solvers.

Over the past two decades, multiple papers have shown that machine learning can make accurate instance-specific predictions about properties of SAT problems (e.g., algorithm runtime prediction [Hutter et al., 2014], algorithm selection [Xu et al., 2008], and satisfiability prediction [Xu et al., 2012]). Perhaps the key drawback of this work is its reliance on computationally expensive hand-engineered features. The computation of most of these features requires between linear and cubic time in the size of the input. It is difficult to assess whether less computationally expensive features would yield similar results, to determine whether important features are missing, and to translate a modeling approach to a new domain. Learning representations from raw problem descriptions via neural networks is a promising approach for addressing these obstacles. While there has been recent work in this direction [Evans et al., 2018; Selsam et al., 2019] that is very interesting from a machine learning perspective—it is not at all obvious that it should be possible to learn a solver from data—this work has tended to focus on problems that are trivial from a combinatorial optimization perspective (e.g., < 1 second to solve by modern SAT solvers).

One of the most widely studied distributions of SAT instances is uniform random 3-SAT at the solubility phase transition [Cheeseman et al., 1991; Mitchell et al., 1992]. These problems are easy to generate, but are very challenging to solve in practice. Indeed, empirical runtimes for high-performance complete solvers have been shown to scale exponentially with problem size on these instances [Mu and Hoos, 2015]. Holding the number of variables fixed, the probability that a randomly generated formula will be satisfiable approaches 100% as the number of clauses shrinks (most problems are underconstrained) and approaches 0% as the number of clauses grows (most problems are overconstrained). For intermediate numbers of variables, this probability does not vary gradually, but instead undergoes

a sharp phase transition at a critical point (a clauses-to-variables ratio of about 4.26) where the probability that a formula will be satisfiable is exactly 50%. Using hand-engineered features, past work [Sandholm, 1996; Xu et al., 2012] showed that an instance's satisfiability status can be predicted with accuracy higher than that of random guessing. In particular, Xu et al. built the models that we believe represent the current state of the art, and they also investigated whether models could generalize across problem sizes so as to claim to identify "asymptotic" behavior. They thus limited their models to features that preserve their meanings across problem scales (e.g., graph-theoretic properties of the constraint structure and the proximity to integrality of the solution to the linear programming relaxation of SAT, rather than e.g., the solution quality attained by a simple local search procedure in a fixed amount of time). They demonstrated that random forest models achieve classification accuracies of at least 70%, even when trained on tiny (100 variable) problems and tested on the largest problems they could solve (600 variables).

In our work, we investigate the use of end-to-end deep networks for this problem. Combinatorial problems are highly structured; changing a single variable can easily flip a formula from satisfiable to unsatisfiable. We thus believe (and will later show experimentally) that success in this domain requires identifying model architectures that capture the correct invariances. Specifically, we encode raw CNF SAT problems as permutation-invariant sparse matrices, where rows represent clauses, columns represent variables, and matrix entries represent whether or not a variable is negated. The deep network architectures we explore are invariant to column-wise and row-wise permutations of the input matrix, which produce logically equivalent problems.

We evaluate two different architectural approaches. The first is to compose a constant number of network layers, each with its own trainable parameters; we use the *exchangeable architecture* that was presented in Chapter 2, because it was shown to be maximally expressive and thus generalizes all other such parameter sharing approaches. The second alternative is to use *message passing* networks, which repeatedly apply the same layer for any desired number of steps; we use the neural message passing implementation of Selsam et al. [2019], as it captures the correct invariances. Selsam et al.'s approach was already applied to SAT, but the focus of their work was on the much harder task of decoding a satisfiable solution.

For that reason, their work only applied the architecture to small problems, which are trivial for modern SAT solvers.

We evaluated both of these neural network approaches on uniform-random 3-SAT instances at the phase transition, primarily to facilitate comparison with past work. Despite the fact that our models did not have access to hand-engineered features and that they were only able to learn linear-time computable features, we achieved substantially better performance than that of Xu et al. [2012]. Specifically, we respectively achieved 78.9% and 79.1% accuracy on average across problems ranging from 100–600 variables for the exchangeable and message passing architectures, compared to 73% accuracy on average for random forests with features and a random guessing baseline of 50%. On 600-variable problems (which typically take hours to solve), we achieved > 80% accuracy with both deep learning architectures. Like Xu et al. [2012], we were able to build models that generalized to much larger problems than those upon which they were trained. For example, we showed that the exchangeable architecture from Chapter 2 achieved 81% prediction accuracy on 600-variable problems when trained on 100-variable problems.

Overall, our work introduces the first example of state-of-the-art performance for the end-to-end modeling of the relationship between the solution to a combinatorial decision problem and its raw problem representation on a distribution that is challenging for modern solvers. We expect our work to be useful for the solving and modeling of SAT and other constraint satisfaction problems.

The rest of this chapter begins with a survey of important related work (Section 4.2); then, we describe the permutation-invariant neural network architectures that we use to represent SAT instances (Section 4.3). We apply these architectures to predicting satisfiability (Section 4.4), and validate them by comparing our results to past work and evaluating their generalization across different instance sizes. Finally, we summarize our contributions (Section 4.6).

4.2 Related Work

Using learning to reason about NP-complete problems Over the past two decades, the combinatorial optimization community has become increasingly interested in

using machine learning to make instance-specific predictions about properties of problems. Much work has focused on the problem of predicting how long a solver will take to run [Finkler and Mehlhorn, 1997; Smith-Miles and Lopes, 2012; Hutter et al., 2014]. These methods have shown surprisingly good performance across a wide range of problems, solvers, and instance distributions. Indeed, many still find it counter-intuitive that it is even possible to predict the behavior of algorithms that run in exponential time in the worst case.

Most work for learning to reason about SAT problems builds on the set of features first proposed by Nudelman et al. [2004]. They generated 84 features that they considered predictive of solver performance, which they derived from known heuristics (e.g., ratio of positive to negative occurrences of clauses and per variables), tractable subclasses (e.g., proximity to Horn formulae), and other proxies for problem complexity (e.g., statistics of the solution to LP relaxations of the SAT problem, and statistics about the progress of SAT solvers over time-bounded runs). The features vary in computational complexity from trivial (e.g., the size of the problem) to expensive (e.g., the LP relaxation, which is roughly cubic).

These features have subsequently been combined with a variety of machine learning models [Xu et al., 2007], and they form the basis of the random forest models studied by Xu et al. [2012]. Hutter et al. [2014] used hand-engineered features to predict per-instance runtimes; features have also been used to build algorithm portfolios, where performance can be improved by selecting different solvers on a per-instance basis [Xu et al., 2008; Lindauer et al., 2015].

Neural network representations for combinatorial problems [Selsam et al., 2019]'s work is the closest in spirit to our own. They learned a neural SAT solver that performs "search" via neural message passing. They focus on the problem of *solving* SAT formulas, but they do so by supervising a message passing architecture with only the satisfiability status of an instance. They therefore cast the problem of determining how to instantiate variables as one of predicting the formula's satisfiability status, and refine this prediction with every recurrent iteration.

Allamanis et al. [2016] and Sekiyama and Suenaga [2018] used TreeNNs to learn end-to-end models for instance-specific predictions of propositional formulae. They considered arbitrary propositional formulae, which lack the same logical

invariances of formulae in CNF, and they considered problems with at most 50 variables.

Loreggia et al. [2016] created fixed-size representations for SAT instances by converting CNF representations to 128×128 images and applying a convolutional neural network. Although this representation is not invariant to variable or clause permutations, the resulting algorithm selector outperformed the best individual solver (but still fell far short of methods based on hand-engineered features).

Both Li et al. [2018] and Selsam and Bjørner [2019] used graphical neural networks to learn heuristics for guiding SAT solvers.

Deep networks have also been used to attack a variety of combinatorial problems beyond SAT; see Bengio et al. [2018] for a recent survey. Most notably, Evans et al. [2018] use a recurrent network for predicting logical entailment; this RNN architecture is not permutation-invariant, and it does not scale to the size of instances we considered in our own work. Prates et al. [2019] also studied predicting optimal tours in Euclidean traveling salesman problems.

Deep networks for exchangeable data A large body of recent work has studied deep network architectures for exchangeable arrays, such as sets [Zaheer et al., 2017], matrices and tensors [Hartford et al., 2018; Bloem-Reddy and Teh, 2019], and graph structured data [see Hamilton et al., 2017b; Battaglia et al., 2018, and references therein]. All of these approaches build deep network layers respecting the in- or equi- variances[1] implied by the exchangeable array (e.g., sets are permutation-invariant, while matrices are equivariant under permutations of rows or columns), but they differ in design decisions about which representations to aggregate over, how multiple layers are composed, and which permutation-invariant functions are used to perform the aggregation.

In this chapter, we explore the first two of these design dimensions. First, we use the exchangeable matrix architecture presented in Hartford et al. [2018] because it supports a natural CNF-style matrix encoding of SAT problems without requiring

[1] A function is *equivariant* if permutations of its input only result in a corresponding permutation of its output, and is *invariant* if permutations of its input leave the output unchanged. For instance, given a permutation matrix Π and an input matrix $X \in \mathbb{R}^{n \times m}$, a function $g : \mathbb{R}^{n \times m} \to \mathbb{R}^{n \times m}$ is equivariant iff $g(\Pi X) = \Pi g(X)$ for all Π and X, and $g : \mathbb{R}^{n \times m} \to \mathbb{R}$ is invariant iff $g(\Pi X) = g(X)$ for all Π and X.

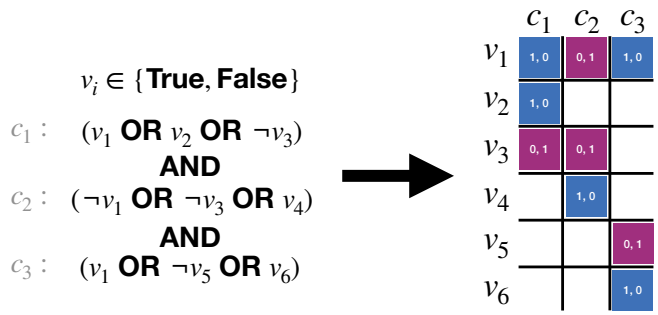

Figure 4.1: An example encoding from a small SAT problem in conjunctive normal form to an exchangeable matrix; $[0,0]$ entries are not shown because they are not explicitly represented in the sparse encoding.

an intermediate graph representation. It was also shown to be maximally expressive among parameter sharing-based architectures for exchangeable matrices, and thus generalizes all approaches based on parameter sharing. Second, in terms of deciding how multiple layers are composed, exchangeable layers can either be treated like standard feedforward layers and stacked (with each layer having its own trainable parameters), or the same layer can be repeatedly applied for some number of steps as with a recurrent network. The latter approach is often referred to as neural message passing[2] [Gilmer et al., 2017], and was used by Selsam et al. [2019] for learning to solve SAT problems; in this chapter, we evaluate their NeuroSAT model on the SAT prediction problem. More expressive attention-based aggregation functions [Vaswani et al., 2017] offer the hope of further performance improvements, but their quadratic complexity makes them infeasible for problems of the size that we study.

4.3 Model Architecture

Data representation A SAT instance with n clauses and m variables in CNF can be represented as an $n \times m \times 2$ clause-variable tensor, where entry (i, j) is the one-hot

[2]The term "message passing" refers to an aggregation step in a graph-based model, so it is sometimes also used for deep networks with separate feedforward-style layers. In this thesis we limit it to the case where the same layer is applied repeatedly, analogous to the repeated application of local updates in message passing for graphical models.

vector $[1,0]$ if the *true* literal for variable i appears in clause j, $[0,1]$ if the *false* literal for variable i appears in clause j, and $[0,0]$ otherwise (see Figure 4.1). This tensor is a sparse exchangeable tensor: each clause will typically only reference a few variables (exactly three in the case of random 3-SAT), so most entries in the tensor will be the zero vector, and permuting the first two dimensions (rows or columns) will not change the SAT problem's satisfiability status.

We denote this input tensor X and let $\mathbb{I} = \{(i,j) : \text{clause } i \text{ references variable } j\}$ denote an index set of the non-zero entries of X. The number of non-zero entries of X is given by $|\mathbb{I}| = \bar{m}n$, where $\bar{m}$ is the average number of variables that appear in each clause. For the random 3-SAT problems that we consider in the experiments $\bar{m} = 3$, so $|\mathbb{I}| = 3n$.

In order to predict the satisfiability of a problem, we need to map from the raw representation of the problem, X, to a scalar output, $y \in [0,1]$, that indicates the probability of satisfiability. We achieve this in two steps that are trained jointly. First, we map each element of the raw input X to a D-dimensional embedding, using a permutation-equivariant function $\phi : \mathbb{R}^{|\mathbb{I}|\times 2} \to \mathbb{R}^{|\mathbb{I}|\times D}$. We then pool the output of $\phi(\mathsf{X})$ to produce a single D-dimensional representation of the SAT problem. This is fed into a second function, $\rho : \mathbb{R}^D \to \mathbb{R}$, which is used to predict the probability that the problem is satisfiable.

We represent ϕ using a sequence of exchangeable matrix layers [Hartford et al., 2018]. These layers apply to tensors in $\mathbb{R}^{n\times m\times k}$; a tensor consists of k channels of exchangeable matrices, each of which is equivariant with respect to permutations of their n rows and m columns. For any given layer, the oth output channel's (i,j)th entry can be computed as follows (with bias terms omitted for clarity):

$$Y_{o,i,j} = \sigma\Bigg(\sum_{k=1}^{K}\Bigg(\theta_1^{\langle k,o\rangle}\mathsf{X}_{i,j,k} + \frac{\theta_2^{\langle k,o\rangle}}{|\mathbb{C}(j)|}\sum_{i'\in\mathbb{C}(j)}\mathsf{X}_{i',j,k} + \frac{\theta_3^{\langle k,o\rangle}}{|\mathbb{V}(i)|}\sum_{j'\in\mathbb{V}(i)}\mathsf{X}_{i,j',k} + \frac{\theta_4^{\langle k,o\rangle}}{|\mathbb{I}|}\sum_{i',j'\in\mathbb{I}}\mathsf{X}_{i',j',k}\Bigg)\Bigg),$$

where $\sigma(\cdot)$ is a nonlinear activation function applied element-wise, X denotes an $n\times m\times K$ input tensor, $\theta_i^{\langle k,o\rangle}$ are trainable weights, $\mathbb{C}(j)$ denotes the indices of all

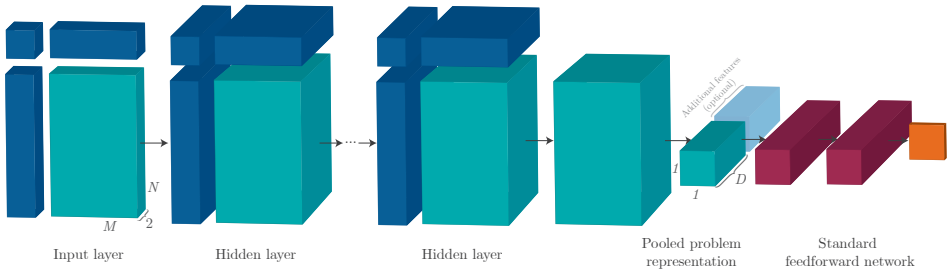

Figure 4.2: Illustration of the exchangeable architecture.

variables referenced in clause j, and $\mathbb{V}(i)$ denotes the indices of all clauses which contain variable i.

Exchangeable architecture While the notation is complex, an exchangeable layer has a simple interpretation as a feed-forward network applied to each layer's literal representation (the terms associated with θ_1), as well as the respective mean-pooled representations of the associated variables (θ_2 terms), clauses (θ_3 terms), and the entire problem (θ_4 terms). The latter three terms provide the mechanism through which the network is able to share information about the assignment of literals between associated clauses and variables. By stacking multiple layers, longer chains of information propagation become possible.

The second function, ρ, acts on the pooled the output of $\phi(X)$. We use a standard multi-layer perceptron as follows:

$$\hat{y} = \rho\left(\frac{1}{|\mathbb{I}|}\sum_{i,j\in\mathbb{I}} \phi(X_{i,j,:}); \beta\right),$$

where $\phi(X_{i,j,:})$ is a D-dimensional vector associated with clause i and variable j, and ρ is a multi-layer perceptron with weights β.

Given a dataset $\mathcal{D} = \{(X_i, y_i)\}_{i\in[1,\ldots,n]}$ of SAT problems, X_i, and targets $y_i \in \{0,1\}$ indicating whether or not a given problem is satisfiable, we train both networks jointly by optimizing β and θ to minimize the binary cross-entropy loss

$$\mathcal{L}_S = \sum_{(X_i,y_i)\in\mathcal{D}} -y_i \log(\sigma(\hat{y}_i)) - (1-y_i)\log(1-\sigma(\hat{y}_i)).$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$.

Assignments In our training set, we often have access to a satisfying assignment[3] in addition to a formula's overall satisfiability. This architecture can easily be extended to also predict satisfying these variable assignments, which can then be used as an auxiliary prediction task to (hopefully) learn better representations of SAT problems; at test time, these predictions are ignored. Given the representation $\phi(X)$, we can pool across clauses to produce variable-specific representations. Then, we can apply a third function, $\mu : \mathbb{R}^{m \times D} \to \mathbb{R}^m$, which yields the probability of each variable taking the value *true*. Again, we use a multi-layer perceptron to represent μ, $\hat{v}_j = \mu\Big(\frac{1}{|\mathbb{C}(j)|} \sum_{i' \in \mathbb{C}(j)} \phi(X_{i',j,:})\Big)$.

As before, μ is trained in conjunction with the rest of the architecture by optimizing binary cross-entropy loss for each variable. The combined loss is,

$$\mathcal{L} = \sum_{(X_i, y_i, \mathbf{v}_i) \in \mathcal{D}} \Big(\mathcal{L}_S(y_i, \hat{y}_i) + \frac{1}{\|\mathbf{v}(i)\|_0} \sum_j \mathcal{L}_A(v_{i,j}, \hat{v}_{i,j}) \Big),$$

where $\mathcal{L}_A(\cdot, \cdot)$ is the binary cross-entropy loss function, $\mathbf{v}_i$ is the true vector of assignments for problem i, and $\|\mathbf{v}(i)\|_0$ is the length of $\mathbf{v}_i$, i.e. the number of variables in the problem.

Message passing In the exchangeable model described above, each layer has its own set of parameters. Another approach is to have all layers share the same set of parameters in a manner similar to an unrolled recurrent neural network (RNN), where each "layer" corresponds to a single recurrent step. This approach is taken by the NeuroSAT architecture.

NeuroSAT represents SAT problems as a bipartite graph with one set of vertices containing true and false literals, and the other clauses; edges denote a literal appearing in a clause. The model has two separate RNNs – one for clauses and one

[3]For any given problem, we only have access to a single assignment—the assignment that the solver found when determining a given formula's satisfiability—but any formula may have many assignments. As a result, by maximizing the likelihood of an assignment, even the Bayes optimal model will have some uncertainty over the true assignment implied by the distribution over possible solutions.

for literals. Each iteration involves first updating the clauses by aggregating over the neighboring literals and applying the clause RNN, and then updating the literals by aggregating over the updated neighboring clauses and applying the literal RNN. Each iteration can be expressed formally as

$$(C^{(t+1)}, C_h^{(t+1)}) \leftarrow g_c([C_h^{(t)}, \sum_{i \in \mathcal{N}(C)} f_c(L_i^{(t)})])$$
$$(L^{(t+1)}, L_h^{(t+1)}) \leftarrow g_l([L_h^{(t)}, \sum_{i \in \mathcal{N}(L)} f_l(C_i^{(t+1)})])$$

where $g_c(\cdot)$ and $g_l(\cdot)$ denote the clause and literal RNNs, C^t and C_h^t are the clause representation and recurrent hidden state (similarly for the literals), $f_c(\cdot)$ and $f_l(\cdot)$ are multilayer perceptrons (MLP), and $\mathcal{N}(\cdot)$ returns the indices of the neighbors of a clause or literal.

By applying the clause and literal RNNs sequentially and using MLPs in the aggregation operation, the NeuroSAT layer introduces multiple intermediate nonlinearities that make exact comparisons with the exchangeable layer impossible. Loosely, the recurrent hidden state plays the same role as the θ_1 terms, and aggregation across clauses and literals corresponds to a nonlinear version of the θ_2 and θ_3 terms.

4.4 Experimental Setup

Data generation To evaluate the networks, we generated uniform-random 3-SAT instances at the solubility phase transition. Following Crawford and Auton [1996], we used a clause (n) to variable (m) ratio of $n = 4.258m + 58.26m^{-2/3}$ to approximate the location of the phase transition. We created 11 datasets, each with a fixed number of variables ranging from 100 to 600 variables at intervals of 50; each dataset contained 10,000 instances, including exactly 5000 satisfiable instances and 5000 unsatisfiable instances.[4] 100-variable instances can trivially be solved by modern SAT solvers in milliseconds; 300-variable instances require several seconds; and 600-variable instances take several hours to solve.

[4]To confirm that we sampled instances at the phase transition, we examined the fraction of satisfiable instances that we generated; we found no evidence that it diverged significantly from 50%.

We believe that this dataset constitutes a useful benchmark for deep models on exchangeable data. Our data generation process creates hard-to-predict problems, but with a noise-free target that offers the asymptotic potential for 100% accuracy. Indeed, training sets of arbitrary size can be generated (albeit at significant computational cost), and testing whether models generalize to unseen instance sets is easily possible because of the natural relationship between sizes.[5]

For our experiments, we randomly split both satisfiable and unsatisfiable problems of each instance size into training, validation, and test sets according to an 80:10:10 ratio. We report the test performance of the parameter checkpoints which performed best on the validation set.

Network training procedure We evaluated four variants of deep network architectures: the standard exchangeable architecture and message passing NeuroSAT architecture predicting only satisfiability, and an extension to both architectures where we jointly predicted satisfying variable assignments.

For the exchangeable architecture, we adapted the public implementation of exchangeable matrix layers from Hartford et al. [2018]. We instantiated the permutation-equivariant portion of the exchangeable architecture as eight exchangeable matrix layers with 128 output channels, with leaky RELU as the activation function. We mapped the final layer to an output width of $D = 64$, which was pooled to a vector before being mapped to the output. We also experimented with inserting a hidden layer between the pooled vector and the output, but observed no significant impact on performance.

For training the exchangeable architecture, we used the Adam optimizer with a learning rate of 0.0001, and mini-batches of 32 examples. Since instances can vary in size, with some being very large, we accumulated gradients for the mini-batches sequentially, back-propagating losses individually for each instance. This slowed training, but was necessary because entire mini-batches could not always be accommodated in memory.

For the message passing network, we used Selsam et al. [2019]'s implementation in the Tensorflow framework. We used the hyperparameters of Selsam et al. [2019]:

[5] For the raw data and the full details of our data generation process, please see http://www.cs.ubc.ca/labs/beta/Projects/End2EndSAT/.

a dimension of 128 for clause and variable embeddings, 3 hidden layers and a linear output layer for each of the MLPs, a scaling factor of 10^{-10} for the ℓ_2 norm to regularize the parameters, 32 iterations of message passing for each problem, a learning rate of 2×10^{-5} for Adam, and a clipping ratio of 0.65 for clipping gradients by global norm. To jointly predict assignments and satisfiability, we added an additional MLP to the aggregation operation that maps literal representations to assignments for each variable, and optimize the same combined loss function described in Section 4.3. Like Selsam et al., we created batches of problems containing up to 12,000 clause and literal nodes that we fed through the network at once. We selected the best-performing checkpoint with a validation set, and report test set accuracy obtained by running the model with 32 message passing iterations per problem.

Baselines To compare our results with the state of the art, we evaluated the performance of decision forests trained on the hand-engineered features used by Xu et al. [2012]. We also implemented a feed-forward neural network that took as input the same hand-engineered features, to ensure that any performance differences were not driven by Xu et al.'s choice of model family.

We also compared our results to two simple baselines that could be trained end-to-end. First, we considered convolutional neural networks (CNNs) to evaluate the impact of capturing permutation invariance. The input to the CNN is a dense representation of the sparse tensor described in Section 4.3. Additionally, to determine whether a simple version of permutation invariance was sufficient to achieve good performance, we also investigated a permutation-invariant nearest-neighbor approach. We used the graph edit distance between variable-clause graphs to determine nearest neighbors, and predicted the satisfiability status of a new point as the satisfiability status of its nearest neighbor.

4.5 Experimental Results

Prediction accuracy We evaluated prediction accuracy for the four variants of deep neural network architectures and the four baselines. For the nearest-neighbor

baseline, we only considered the 100- and 200-variable datasets because of the high computational cost of computing graph edit distances; for all other approaches, we considered 11 fixed-size datasets.

Both nearest neighbor and CNNs performed poorly. Nearest neighbor never achieved prediction accuracies above 53%, even when we used the expensive Hausdorff graph edit distance. The performance of CNNs consistently approached that of random guessing, achieving no more than 50.5% on any of the 11 datasets after 48 hours of training. We concluded that permutation-invariant architectures made a significant impact on predictive performance.

Table 4.1 presents the performance results for the permutation-invariant methods. Like Xu et al., we observed that prediction accuracy *increased* with instance size for the exchangeable variants of the permutation-invariant architecture. With the exchangeable model variant predicting only satisfiability, we achieved prediction accuracies between 71% and 82% — 1–7% higher than random forests and 1–8% higher than the fully-connected neural network using hand-engineered features. A two-sided Wilcoxon signed-rank test for differences in the distribution of performance across all 11 datasets between each hand-engineered model and the exchangeable variant predicting only satisfiability showed the results were significant ($p < 0.01$ for both tests of the models on hand-engineered features).

Using the exchangeable model variant where satisfiability and assignments were jointly predicted, we achieved prediction accuracies between 72% and 84%, with an improvement in the model by an additional 1–2% across all datasets. We achieved similar levels of performance with message passing architectures, achieving prediction accuracies usually between 75% and 83%, however accuracies were roughly uncorrelated with instance size. Using the message passing variant where satisfiability and assignments were jointly predicted, we achieved an additional 1.6% accuracy on average. Averaged across all datasets, the exchangeable and message passing architectures with assignments respectively achieved 78.9% and 79.1% prediction accuracy. Neither architecture was better than the other on more than 6/11 of the instance sets. In terms of prediction accuracy when testing on

	Hand-Engineered		Exchangeable		MP	
#Vars	RF	NN	SAT	+Assign	SAT	+Assign
100	0.702	0.704	0.712	0.726	0.675	**0.751**
150	0.712	0.714	0.731	0.745	**0.778**	0.771
200	0.734	0.718	0.760	0.772	0.767	**0.781**
250	0.703	0.723	0.776	**0.800**	0.758	0.788
300	0.744	0.725	0.789	**0.800**	0.758	0.788
350	0.734	0.730	0.787	0.809	**0.834**	0.812
400	0.711	0.710	0.765	**0.790**	0.777	0.781
450	0.700	0.710	0.788	0.789	0.774	**0.803**
500	0.773	0.778	0.800	**0.809**	0.791	0.795
550	0.756	0.761	0.804	0.810	0.789	**0.813**
600	0.813	0.810	0.811	**0.837**	0.816	0.818

Table 4.1: Comparison of prediction accuracy for satisfiability in the epoch with the lowest validation error. RF denotes random forests; NN, the standard feed-forward network; Exchangeable, the standard exchangeable network; and MP, the message passing model of Selsam et al. [2019]. SAT denotes the permutation-invariant model variants trained to predict satisfiability; +Assigns, the permutation-invariant model variants trained to predict satisfiability and satisfying assignments. Boldface indicates the best-performance.

the same instance sizes used for training, we found no reason to hold a preference between the exchangeable and message passing architectures[6].

Running time Permutation-invariant neural networks are far more expensive to train than the random forest baseline, requiring at least 40 hours of training time and 500 MB of GPU memory (for exchangeable networks; message passing is more expensive, requiring 0.6 to 3.3 GB per instance). However, evaluating this class of models requires only time linear in the size of the input, whereas the hand-engineered features upon which the random forest models depend have roughly

[6] A Wilcoxon test does not detect a significant difference between the performance of the exchangeable model with assignments and either message passing architecture ($p = 0.12$ and $p = 0.96$ for the variants without and with assignments, respectively)

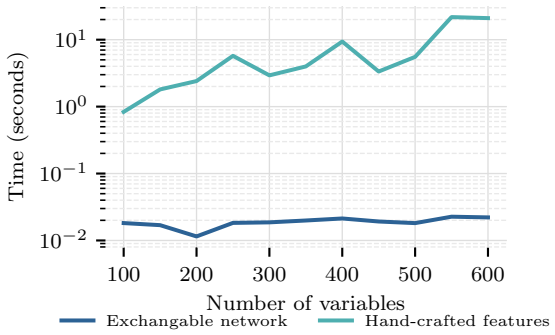

Figure 4.3: Average running times per instance as size varies for the exchangeable architecture and hand-engineered features. Note the log scale on the y-axis.

cubic time complexity. This asymptotic difference is not overwhelmed by constants: at the input sizes we investigated, exchangeable models are faster to evaluate by several orders of magnitude, as shown in Figure 4.3. We note that this difference might make such models particularly attractive for constructing algorithm portfolios, where time saved on feature computation can be reallocated to solving problems.

Generalizing across sizes To verify that permutation-invariant models are able to capture general structural properties of the given SAT instances rather than simply memorizing instances at particular sizes, we evaluated the prediction accuracy of networks trained on 100-variable instances using all of the other datasets. Our results, along with analogous results for random forests, are shown in Table 4.2.

With the exchangeable architecture trained on 100-variable instances, we achieved nearly undiminished performance and definitively outperformed random forests on all instance sizes. Notably, the model trained on our dataset of trivial 100-variable random 3-SAT instances achieved 81% accuracy on the dataset of hard 600-variable 3-SAT instances. We again observed the increase in prediction accuracy with instance size reported by Xu et al.

Prediction accuracy for message passing was usually between 68–74%. We note that this range is similar to the prediction accuracy achieved when testing message passing on 100-variable instances. By comparison, considering models trained on 100-variable instances, the performance of the message passing model did not gen-

#Vars	RF-100	Exch-100	MP-100
150	0.695	**0.758**	0.734
200	0.695	**0.759**	0.704
250	0.654	**0.776**	0.722
300	0.711	**0.780**	0.729
350	0.705	**0.791**	0.725
400	0.681	**0.756**	0.711
450	0.692	**0.778**	0.699
500	0.716	**0.777**	0.686
550	0.722	**0.768**	0.669
600	0.739	**0.809**	0.683

Table 4.2: Comparison of satisfiability prediction accuracy achieved by testing a model trained on 100-variable instances on other datasets. RF-100 denotes the random forests trained on 100 variables and tested on other sizes; Exchangeable-100, the exchangeable network trained to predict satisfiability and assignments on 100 variables, and tested on other sizes; and MP-100, the message passing model of Selsam et al. [2019] trained on 100 variables and tested on other sizes. Boldface indicates the best-performing approach for each dataset. Wilcoxon signed-rank test across these 10 datasets shows Exch-100 significantly improves on the other two models ($p < 0.01$) and no significant different between MP-100 and RF-100 ($p = 0.695$)

eralize as well as the exchangeable architecture, which achieved better performance on all instance sizes. In fact, we observed that the accuracy of message passing decreased with instance size, and that the exchangeable architecture achieved 12% better accuracy for the biggest instance sizes (550 and 600 variables).

Overall, the exchangeable architecture and message passing achieved comparable performance when trained on the same distribution. However, because of its superior generalization performance and lower memory requirements, we recommend exchangeable models over message passing for predicting satisfiability.

SAT invariances Visualizing the embedding space provides a way to verify that the exchangeable architecture performed as desired. We used the following observation

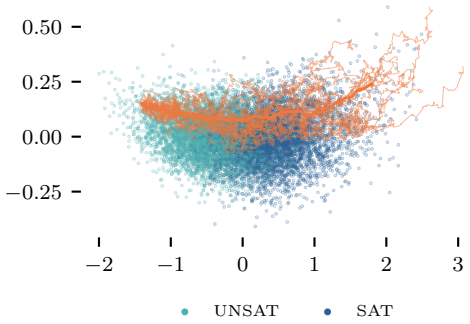

Figure 4.4: In the space of problem embeddings in the exchangeable architecture, orange lines show paths through towards the SAT portion of the space as clauses are removed from an UNSAT instance, with the thicker line showing the average path. The original 64-dimensional space is projected down to the first two principal components of a model trained on the 300-variable dataset.

to examine its behavior as instances were modified: the more clauses are removed from any unsatisfiable instance, the greater the chance that it will become satisfiable, since removing a clause can never reduce the solution space. Taking a random instance predicted to be unsatisfiable by the network, we iteratively removed randomly selected clauses. At every step, we recorded the network's pooled representation of the instance. A projection of the paths from 20 independently sampled trajectories is shown in Figure 4.4. As clauses were removed, the representation of the instance shifted from the portion of the space associated with unsatisfiable instances to the portion associated with satisfiable instance and the associated predicted probability of satisfiability also increased, as expected. This shows that the network has correctly learned that these simple transformations that makes the problem iteratively less constrained, also make the problem more likely to be satisfiable.

Latent space distribution We explored the distribution of problems in the exchangeable architecture's latent space as problem size changed. Figure 4.5 shows a kernel density plot of these distributions from the final exchangeable layer for the exchangeable architecture projected down to the first principal component for both satisfiable and unsatisfiable instances in four different-sized datasets. We observed that the instances become more concentrated around a particular point as problem

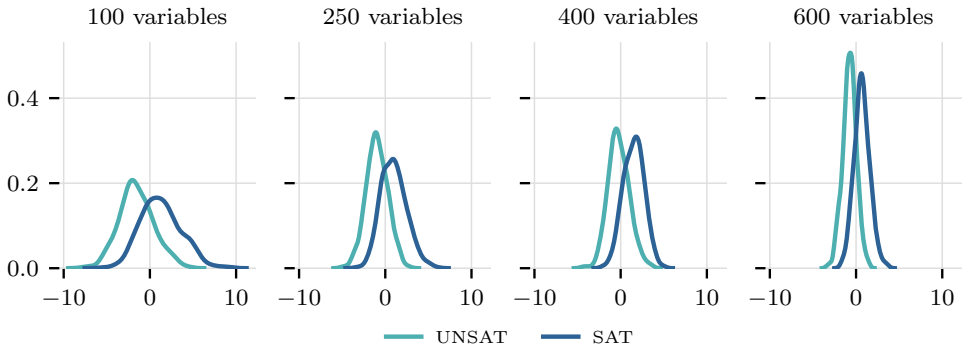

Figure 4.5: Comparison of variance in problem embeddings of the exchangeable architecture as instance size increases. The original 64-dimensional space is projected down to the first principal component based on models trained with 300 variables. For each size, a kernel density function is fit for UNSAT and SAT problems.

sizes increased. Certain functions on random graphs converge as $n \to \infty$ (e.g., the size of the maximum clique in Erdos-Renyi graphs). We suspect that the deep networks are learning a function that concentrates with increasing problem size.

4.6 Conclusions

This chapter is the first to study end-to-end learning of the satisfiability status of empirically challenging SAT problems based on their CNF representations. We showed that both deep exchangeable and neural message passing models achieved state-of-the-art prediction performance on random 3-SAT problems at the phase transition, consistently outperforming models based on sophisticated hand-engineered features that have been central to machine learning in SAT for over a decade. These models also have a clear computational advantage over hand-engineered feature-based models: for 600-variable problems, a forward pass of the exchangeable architecture was more than two orders of magnitude faster than computing hand-engineered features. We also showed out-of-sample generalization to much larger instance sizes at nearly undiminished levels of accuracy. Indeed, the exchangeable network architecture trained on 100-variable instances (milliseconds to solve) achieved per-

formance on 600-variable instances (hours to solve) which was on par with that of hand-engineered feature-based models trained on 600-variable instances.

Part II

Causal Inference with Instrumental Variables

Chapter 5

DeepIV

In Part I, we leveraged knowledge about structural invariances in the data to design and apply equivariant deep network architectures that were able to generalize across different sized instances, and even to different datasets in the extreme cases shown in Section 2.4.2. In this second part, we develop methods to generalize in a different way: predicting how decision makers will act under new policies, using only offline data collected under an incumbent policy. The key to this causal inference approach is its use of "instrumental variables": variables that only affect the data generating process in a specific, limited way. We leverage this structure to predict the effect of interventions that result in changes in distribution.

5.1 Introduction

In this chapter we aim to develop machine learning methods that allow us to predict the effect of policy changes or *interventions* on some data generating process (DGP). To see the difference between predicting the effect of interventions and standard prediction problems, consider the following example: a data scientist is trying to build a sales prediction model that can predict sales of airline tickets, y, given prices, p. They collect a dataset of historical prices and ticket sales (shown in Figure 5.1 (left)) and estimate a model, $\hat{g} : p \to y$ (the line shown in Figure 5.1 (left)). For the purposes of this example, we can assume that the model is the Bayes optimal predictor—that is, it is the minimizer of expected generalization error—and so it

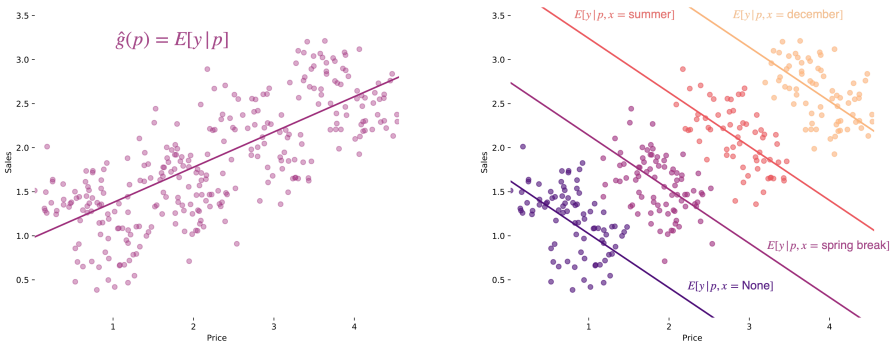

Figure 5.1: An example of confounding: the apparent positive correlation between sales and price (left) is the result of the effect of different holidays on the price-sales relationship (right).

performs well on held out data when responding to queries of the form, "what will sales be when we observe some price, p?" But, notice that if we instead tried to use the model to ask the policy question, "how do we set prices in order to maximize sales?"—equivalently, what is $p^* = \arg\max'_p \hat{g}(p')$—the upward sloping relationship between price and sales implies that the model predicts we could set price to infinity and achieve infinite sales.

This seems puzzling: how could a model that accurately predicts sales under the current pricing regime make such obviously wrong policy recommendations? In this example, this occurs because there is a latent variable that affects both prices and sales, and as a result, at different times of year there are different price–ticket sales relationships (Figure 5.1 (right)). For example, December is a popular time to fly because people want to get home to visit family, but airlines are aware of this and set prices high, knowing that enough people will be prepared to fly despite the higher than normal prices. So at any particular time of year, the real price–ticket sales relationship is downward sloping (as we would expect), but because it shifts outward during holidays, price and tickets sales appear positively correlated.

Now, we could correctly estimate the true price–ticket sales relationship by fitting a model that conditions on both price and time of year (as shown in the downward sloping curves in Figure 5.1 (right)), but it should concern us that we

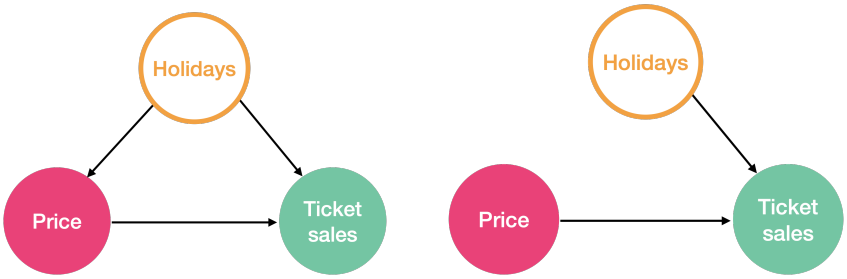

Figure 5.2: In the observational distribution (left), prices react to holidays according to some (unknown) policy. In the interventional distribution (right), we set price to some fixed value.

make very different policy recommendations (increase or decrease price in Fig. 5.1 (left) and (right) respectively) depending on the variables on which we choose to condition. In this example, we have a strong prior of what the true relationship should look like, but in general that will not be the case. So, how do we know which of the two policy recommendations to trust in settings when we do not already know what the true effect should look like?

To address this, we first need to distinguish between the *observational distribution* and the *interventional distribution*. In our example, the data scientist has access to a dataset of examples drawn from some subset of the variables that make up the observational distribution: they observe price, p, and ticket sales, y, but not holidays, x. Under this distribution, price is set according to some unknown policy (indicated by the arrow from holidays to price in Figure 5.2), but as long as this policy is not changed, predictions of the form $\mathbb{E}[y|p]$ will perform well under held-out data on this distribution. To predict the effect of policy interventions, however, we need to reason about the interventional distribution, predicting $\mathbb{E}[y|\text{do}(p')]$, where the $\text{do}(\cdot)$ operator indicates that we replace the observed distribution over prices with some arbitrary fixed value p' and consider the resulting conditional distribution (graphically, we cut all incoming edges into the price variable Figure 5.2 (right), [Pearl, 2009]); if we had access $\mathbb{E}[y|\text{do}(p')]$ for all p', we could then optimize over p' to make policy recommendations.

The challenge is that we only have access to the observational distribution but want to make predictions about the interventional distribution. One way to proceed

would be to discard the observational data and instead collect interventional data directly via a randomized control trial. When possible, this is regarded as the gold standard for causal inference because all confounding is removed by design. But, there are many settings where it is either unethical, too expensive or simply not possible to run an experiment where we explicitly randomize the treatment variable (price in our example). The alternative is to work with observational data, but doing so requires explicit assumptions about the causal structure of the DGP[1] [Bottou et al., 2013]. If these assumptions imply that the variables that we observed are sufficient to "control" for confounding—in our example we only have to consider holidays; more generally one needs a valid adjustment set, x [see Pearl, 2009, for details]—then we can estimate the effect of interventions using the adjustment formula, $\mathbb{E}[y|\text{do}(p')] = \mathbb{E}_{x\sim P(x)}[\mathbb{E}[y|p', x]]$. This would be equivalent to estimating the price–ticket sales relationship at each different time of year (the curves Figure 5.1 (right)) and then taking a weighted average of these estimates, weighted by the marginal distribution of x. In this *unconfounded* setting, there are many efficient estimators [see e.g. Athey and Imbens, 2016; Johansson et al., 2016], but they all rely on the assumption that the variables that make up a valid adjustment set are observed.

What do we do in settings where confounding is unavoidable? For example, the data scientist may know about the holiday effect, x, but sometimes demand for air tickets is high due to conferences, e, whose attendees are also price insensitive. If the incumbent pricing policy raises prices for conferences, we would again have high price, high ticket sales observations which would bias naive estimates. In these cases, the causal effect $\mathbb{E}[y|\text{do}(p')]$ is not identifiable without additional assumptions. Here we assume we have access to an instrumental variable (which we will denote z): some variable which, by definition, only affects the response variable (ticket sales) via its effect on the treatment variable (price); see Figure 5.3. In our airline example, the cost of fuel could be such an instrument if its variation is independent of demand for airline tickets and it affects tickets sales only via its effect on prices.

[1] This chapter focuses on causal inference which is the task of estimating a causal effect given a set of identifying assumptions (typically supplied by a domain expert) which can be expressed as a causal graph. One could also attempt to identify the causal graph from data, but this *causal discovery* task is especially challenging for the problems we focus on in this chapter (allowing for unobserved confounding). See Peters et al. [2017] for details.

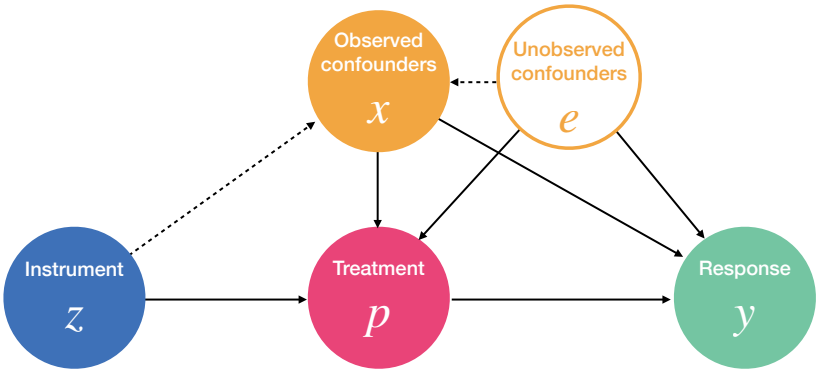

Figure 5.3: A general structure of the DGP under our IV specification; dotted lines indicate edges which are allowed under the IV assumptions, but are not used in our running example. x represents observable features, p is our treatment variable of interest, z represents the instruments, and latent effects e influence the response y additively. In our running air-travel demand example, the treatment variable is price, p, sales is the response, y, and holidays are observable covariates, x. There is a big 'conference', e, unobserved to the policy-maker, that drives demand and (due to the airline's pricing algorithms) price. The instrument is the cost of fuel, z, which influences sales only via price.

If we have a valid instrumental variable, instrumental variable estimation is an inverse problem. We can observe both the effect of the instrument on the treatment variable (how ticket prices react to changes in the fuel price) and the effect of the instrument on the response variable (how ticket sales react to changes in the fuel price) and because we assumed that the only relationship between the instrument and the response is that which is mediated via the treatment, we can attribute the difference between these two observations to the causal effect of the treatment on the response. More formally, the instrument's effect on the response can be expressed as $\mathbb{E}[y|z]$ and instrument's effect on the treatment is $P[p|z]$, and both of these quantities can be estimated from the observational distribution. We can then estimate the

causal effect by finding the function $g : p \to y$, such that $\mathbb{E}[y|z] = \mathbb{E}[g(p)|z]$ [Newey and Powell, 2003]. If there is only one g that makes this equality hold, then the problem is identified. In general, this will not be the case [Balke and Pearl, 1997] so we need additional structural assumptions about the true function that determines our response variable y. For example, if we assume linearity such that $y = \beta p + e$ for some unknown β, then by linearity of expectation, $\mathbb{E}[y|z] = \beta\,\mathbb{E}[p|z]$, so β can be estimated in two stages by first regressing p on z to estimate $\mathbb{E}[p|z]$, and then using the predicted values of p in a second stage regression.[2] This is the widely used two-stage least squares approach which is a key technique in econometrics and epidemiology.[3]

A weaker assumption, which we leverage in this chapter, is to assume additive unobserved confounding, such that $y = g(p, x) + e$ for some true function g. Using this assumption we derive a deep network-based approach to estimating $\mathbb{E}[y|\text{do}(p), x]$. Our approach also has two stages: first, we model the conditional distribution of the treatment variable given the instruments and covariates using standard conditional density estimation techniques. Second, we optimize a loss function involving integration over the conditional treatment distribution from the first stage. Both stages use deep neural nets trained via stochastic gradient descent [Robbins and Monro, 1951; Bottou, 2010]. We also present an out-of-sample causal validation procedure for selecting hyperparameters of the models on a validation set. We refer to this setup as the Deep IV framework.

Section 5.3 describes our general IV specification and its decomposition into two learning tasks. Section 5.4 outlines neural network estimation for these tasks with particular attention paid to the SGD routine used in model training and our causal validation procedure. Section 5.5 presents experimental results that illustrate the benefits of our methods, while Section 5.5.2 applies Deep IV for inference on the causal effect of ad position in web search results.

[2] Note that trying to estimate β directly via linear regression would be biased because $p \not\perp e$. In our example, this naive approach would product the upward sloping curve shown in Figure 5.1 (left).

[3] This assumption of "constant treatment effects" implied by the constant β implies that every individual responds in exactly the same way to a treatment. This is obviously an unrealistic assumption in many settings where IV methods are applied. In the case of binary treatments and instruments weaker assumptions are possible; see Angrist et al. [1996].

5.2 Related work

The IV framework has a long history, especially in economics [e.g., Wright, 1928; Reiersøl, 1945]. It provides methods for learning the regression function that relates the treatment and response variables under the interventional distribution for DGPs that conform to the graphical model shown in Figure 5.3 [Pearl, 2009]. Most IV applications make use of a two-stage least squares procedure [2SLS; e.g., Angrist and Pischke, 2008] that applies a model of linear and homogeneous treatment effects (e.g., all airline customers must have the same price sensitivity). Nonparametric IV methods from the econometrics literature relax these assumptions [e.g., Newey and Powell, 2003; Darolles et al., 2011]. However, these methods typically work by modeling the outcome as an unknown linear combination of a prespecified set of basis functions of the treatment and other covariates (e.g. Hermite polynomials, wavelets, or splines) *and* then modeling the conditional expectation of each of these basis functions in terms of the instruments (i.e., the number of parameters is quadratic in the number of basis functions). This requires a strong prior understanding of the DGP by the researcher; also, the complexity of both specification and estimation explodes when there are more than a handful of inputs.

5.3 Nonlinear instrumental variable estimation

We aim to predict the value of some outcome variable y (e.g., sales in our airline example) under an intervention in a policy or treatment variable p (e.g., price). There exists a set of observable covariate features x (e.g., holidays) that we know affect both p and y. There also exist unobservable latent variables e (e.g., conferences) that may affect p and y. We aim to recover $\mathbb{E}[y|\text{do}(p), x]$ in the context of the graphical model given by Figure 5.3, where the $\text{do}(\cdot)$ operator indicates that we have intervened to set the value of the policy variable p [as per Pearl, 2009]. We assume the y is structurally determined by p, x and e as

$$y = g(p, x) + e. \tag{5.1}$$

That is, $g(\cdot)$ is some unknown and potentially non-linear continuous function of both x and p, and we assume that the latent variables (or "error") e enter additively

with unconditional mean $\mathbb{E}[e] = 0$. In our example, this amounts to assuming that sales might be some flexible nonlinear function of price and observed confounding factors like holidays, but that the conferences only have an additive effect on sales. Because any latent confounding factors, e, are potentially correlated with the inputs we allow $\mathbb{E}[e|x, p] \neq 0$ and, in particular we expect that, $\mathbb{E}[pe|x] \neq 0$.

The true function $g(\cdot)$ is not identifiable because the latent confounder's affect on y could vary as a function of x. Instead we will focus on estimating conditional causal effects, $\mathbb{E}[y|\text{do}(p), x]$ [Pearl, 2009],

$$h(p, x) \equiv \mathbb{E}[y|\text{do}(p), x] = g(p, x) + \mathbb{E}[e|x], \tag{5.2}$$

Note that we condition only on x and not p in the term $\mathbb{E}[e|x]$; this term is typically nonzero, but it will remain constant under arbitrary changes to our policy variable p.[4] The conditional causal effect is sufficient to evaluate policy options (e.g. changing the ticket price from p_0 to p_1) because we can look at the difference in outcomes $h(p_1, x) - h(p_0, x) = g(p_1, x) - g(p_0, x)$.

In standard supervised learning settings, the prediction model is trained to fit $\mathbb{E}[y|p, x]$. This will typically be *biased* against our structural objective because

$$\mathbb{E}[y|p, x] = g(p, x) + \mathbb{E}[e|p, x] \neq h(p, x) \tag{5.3}$$

since our treatment is not independent of the latent errors by assumption and hence $\mathbb{E}[e|p, x] \neq \mathbb{E}[e|x]$. This object is inappropriate for policy analysis as it will lead to biased treatment effects:

$$\mathbb{E}[y|p_1, x] - \mathbb{E}[y|p_0, x] = g(p_1, x) - g(p_0, x) + \Big(\mathbb{E}[e|p_1, x] - \mathbb{E}[e|p_0, x]\Big). \tag{5.4}$$

In our airline example, the higher than normal prices and demand during conferences would imply that $\mathbb{E}[e|p_1, x] - \mathbb{E}[e|p_0, x]$ will be positive; this occurs because in the observed distribution, higher than normal prices were observed during higher than

[4] It may be easier to think about a setting where $e \perp\!\!\!\perp x$, so that the latent error is simply *defined* as being due to factors orthogonal to the observable controls. In that case, $h(p, x) = g(p, x)$. All of our results apply in either setup.

normal demand. This will result in the incorrect prediction that higher prices are associated with higher sales if this bias is sufficiently large.

Fortunately, the presence of *instruments* allows us to estimate an unbiased $\hat{h}(p, x)$ that captures the structural relationship between p and y. These are sets of variables z that satisfy the following three conditions.

Relevance $F(p|x,z) \neq F(p|x)$, the distribution of p given x and z, is not independent of z. E.g, relevance requires that the airline change prices, p, in response to changes in the fuel prices, z.

Exclusion z does not directly affect y—i.e., $z \perp\!\!\!\perp y \mid (x, p, e)$; note that this is implied by Equation (5.1). E.g, exclusion requires that none of the airline customers consider the oil price when deciding whether to buy a ticket. This assumption would be violated if the data included oil traders who might be more likely to fly when fuel prices are high.

Unconfounded Instrument $z \perp\!\!\!\perp e \mid x$. The instrument, z is conditionally independent of any unobserved confounder, e.[5] This assumption rules out latent factors that affect both the instrument and the response. COVID-19 would represent a violation of this assumption because it reduced both fuel cost (via lower oil prices) and demand for airline tickets.

Taking the expectation of both sides of Equation (5.1) conditional on $[x,z]$, applying the unconfounded instrument assumption and the definition of $\hbar$ (Equation (5.2)) establishes the relationship [cf. Newey and Powell, 2003]:

$$\begin{aligned}\mathbb{E}[y|x,z] &= \mathbb{E}[\, g(p,x)|x,z] + \mathbb{E}[e|x] \\ &= \int \hbar(p,x) dF(p|x,z), \end{aligned} \tag{5.5}$$

[5] Under the additive confounder assumption made in Eq. (5.1), unconfoundedness of the instrument is not necessary: we could replace this assumption with the weaker mean independence assumption $\mathbb{E}[e|x,z] = 0$ without changing anything that follows. We use the stronger assumption to facilitate extensions, e.g. to estimating quantiles under interventions. Our assumption is similar to the 'unconfoundedness' assumption in the Neyman–Rubin potential outcomes framework [Rosenbaum and Rubin, 1983] (i.e. $p \perp\!\!\!\perp e \mid x$). But our assumption is weaker—in particular, we allow for $p \not\perp\!\!\!\perp e|x$—and so the matching and propensity-score re-weighting approaches often used in that literature will not work here.

where, again, $dF(p|x,z)$ is the conditional treatment distribution. The relationship in Equation (5.5) defines an *inverse problem* for h in terms of two directly observable functions: $\mathbb{E}[y|x,z]$ and $F(p|x,z)$. IV analysis typically splits this into two stages: first estimating $\hat{F}(p|x_t,z_t) \approx F(p|x_t,z_t)$, and then estimating $\hat{h}$ using $\hat{F}$.

Most existing approaches to IV analysis assume linear models for the treatment density function $\hat{F}$ and the counterfactual prediction function $\hat{h}$ to solve Equation (5.5) in closed form. For example, the two-stage least-squares (2SLS) procedure [e.g., Angrist et al., 1996] posits $y = \gamma p + x\beta_y + e$ and $p = \tau z + x\beta_p + v$, with the assumptions that $\mathbb{E}[e|x,z] = 0$, $\mathbb{E}[v|x,z] = 0$, and $\mathbb{E}[ev] \neq 0$ (which implies $\mathbb{E}[ep] \neq 0$). This procedure is straightforward: fit a linear model for p given x and z and use the predicted values $\hat{p}$ in a second linear model of y. This is a statistically efficient way to estimate the effect of the policy variable (i.e. γ) as long as two strong assumptions hold: linearity (i.e., both first- and second-stage regressions are correctly specified) and homogeneity (i.e., the policy affects all individuals in the same way).[6]

Flexible nonparametric extensions of 2SLS either replace the linear regressions with a linear projection onto a series of known basis functions [Newey and Powell, 2003; Blundell et al., 2007; Chen and Pouzo, 2012] or use kernel-based methods as in Hall and Horowitz [2005] and Darolles et al. [2011]. This system of series estimators is an effective strategy for introducing flexibility and heterogeneity with low dimensional inputs, but the approach faces the same limitations as kernel methods in general: their performance depends on the choice of kernel function; and they often become computationally intractable in high-dimensional feature spaces $[x,z]$ or with large numbers of training examples.

5.4 Estimating and validating DeepIV

We now describe how to use deep networks to perform flexible, scalable IV analysis in a framework we call DeepIV. We make two contributions that are each necessary components of the approach. First, we propose a loss function and optimization procedure that allows us to optimize deep networks to make valid predictions under

[6] The estimated $\hat{\gamma}$ remains interpretable as a 'local average treatment effect' (LATE) under less stringent assumptions [see Angrist et al., 1996, for an overview].

interventions. Second, we describe a general procedure for out-of-sample validation of two-stage instrument variable methods. This allows us to perform causally valid hyperparameter optimization, which in general is necessary for achieving good predictive performance using deep networks.

Our approach is conceptually simple given the inverse problem described above. Rather than constraining ourselves to analytic solutions to the integral in Equation (5.5), we instead directly optimize our estimate of the structural equation, $\hat{h}$. Specifically, to minimize ℓ_2 loss given n data points and given a function space $\mathcal{H}$ (which may or may not include the true h), we solve

$$\min_{\hat{h}\in\mathcal{H}} \sum_{t=1}^{n} \left(y_t - \int \hat{h}(p, x_t) dF(p|x_t, z_t) \right)^2. \tag{5.6}$$

Since the treatment distribution is unknown, we estimate $\hat{F}(p|x,z)$ in a separate first stage.

So the DeepIV procedure has two stages: a first stage density estimation procedure to estimate $\hat{F}(p|x,z)$ and a second that optimizes the loss function described in Equation (5.6). In both stages hyperparameters can be chosen to minimize the respective loss functions on a held out validation set, and improvements in performance against this metric will correlate with improvements on the true structural loss which cannot be evaluated directly. We briefly discuss these two stages before describing our methods for optimizing the loss given in Equation (5.6) and our causal validation procedure.

First stage: Treatment network In the first stage we learn $F(p|x,z)$ using an appropriately chosen distribution parameterized by a deep neural network (DNN), say $\hat{F} = F_\phi(p|x,z)$ where ϕ is the set of network parameters. Since we will be integrating over F_ϕ in the second stage, we must fully specify this distribution.

In the case of discrete p, we model $F_\phi(p|x,z)$ as a categorical $\mathrm{Cat}(p \mid \pi(x,z;\phi))$ with $\mathrm{P}(p = p^k) = \pi_k(x,z;\phi)$ for each treatment category p^k and where $\pi_k(x,z;\phi)$ is given by a DNN with softmax output. For continuous treatment, we model F as a mixture of Gaussian distributions where component weights $\pi_k(x,z;\theta)$ and parameters $[\mu_k(x,z;\phi), \sigma_k(x,z;\phi)]$ form the final layer of a neural network parametrized

by ϕ. This model is known as a mixture density network, as detailed in §5.6 of Bishop [2006]. With enough mixture components it can approximate arbitrary smooth densities. To obtain mixed continuous–discrete distributions we replace some mixture components with point masses. In each case, fitting F_ϕ is a standard supervised learning problem.

Second stage: Outcome network In the second stage, the conditional causal effect h is approximated by a DNN with real-valued output, say h_θ. We optimize network parameters θ to minimize the integral loss function in Equation (5.6) over training data D of size $T = |D|$ from the joint DGP $\mathcal{D}$,

$$\mathcal{L}(D;\theta) = |D|^{-1} \sum_t \left(y_t - \int h_\theta(p, x_t) d\hat{F}_\phi(p|x_t, z_t) \right)^2. \tag{5.7}$$

Note that this loss involves the *estimated* treatment distribution function, $\hat{F}_\phi$, from our first stage.[7] Once this second stage network, h_θ, is trained, we can discard $\hat{F}_\phi(p|x_t, z_t)$ and simply use h_θ as our predictor of $E[y|\text{do}(p), x]$.

5.4.1 Optimization for DeepIV networks

We use stochastic gradient descent to train the network weights. For F_ϕ, standard off-the-shelf methods apply, but second stage optimization (for h_θ) needs to account for the integral in Equation (5.7). SGD convergence only requires that each sampled gradient $\nabla_\theta \mathcal{L}_t$ is *unbiased* for the population gradient, $\nabla_\theta \mathcal{L}(\mathcal{D};\theta)$. Lower variance for $\nabla_\theta \mathcal{L}_t$ will tend to yield faster convergence [Zinkevich, 2003] while the computational efficiency of SGD on large datasets requires limiting the number of operations going into each gradient calculation [Bousquet and Bottou, 2008].

We can approximate the integral with respect to a probability measure with the average of draws from the associated probability distribution: $\int h(p) dF(p) \approx B^{-1} \sum_b h(p_b)$ for $\{p_b\}_1^B \overset{iid}{\sim} F$. Hence we can get an unbiased estimate of Equation (5.7) by replacing the integral with a sum over samples from our fitted treatment

[7] We can replace Eq. (5.7) with other functions, e.g., a softmax for categorical outcomes, but use ℓ_2 loss for most of our exposition.

distribution function, $\hat{F}_\phi$:

$$\mathcal{L}(D;\theta) \approx |D|^{-1} \sum_t \left(y_t - \frac{1}{B} \sum_{\dot{p} \sim \hat{F}_\phi(p|x_t,z_t)} h_\theta(\dot{p}, x_t) \right)^2 =: \hat{\mathcal{L}}(D;\theta). \tag{5.8}$$

This equation can be used to estimate $\nabla_\theta \mathcal{L}$ with an important caveat: if we want to maintain unbiased gradient estimates, *independent* samples must be used for each instance of the integral in the gradient calculation. To see this, note that the gradient of Equation (5.8) has expectation

$$\begin{aligned} \mathbb{E}_{\mathcal{D}} \nabla_\theta \mathcal{L}_t &= -2\mathbb{E}_{\mathcal{D}} \Big(\mathbb{E}_{F_\phi(p|x_t,z_t)} \Big[y_t - h_\theta(p^k, x_t) \Big] \cdot \mathbb{E}_{F_\phi(p|x_t,z_t)} \Big[h'_\theta(p^k, x_t) \Big] \Big) \\ &\neq -2\mathbb{E}_{\mathcal{D}} \mathbb{E}_{F_\phi(p|x_t,z_t)} \Big[\big(y_t - h_\theta(p^k, x_t) \big) h'_\theta(p^k, x_t) \Big], \end{aligned} \tag{5.9}$$

where the inequality holds so long as $\operatorname{cov}_{F_\phi(p|x_t,z_t)} \Big[\big(y_t - h_\theta(p^k, x_t) \big) h'_\theta(p^k, x_t) \Big] \neq 0$. We thus need a gradient estimate based on unbiased MC estimates for each $\mathbb{E}_{F_\phi(p|x_t,z_t)}$ term in Equation (5.9). We obtain such an estimate by taking two samples $\{\dot{p}_b\}_1^B, \{\ddot{p}_b\}_1^B \overset{iid}{\sim} F_\phi(p|x_t, z_t)$ and calculating the gradient as

$$\widehat{\nabla}^B_\theta \mathcal{L}_t \equiv -2 \left(y_t - B^{-1} \sum_b h_\theta(\dot{p}_b, x_t) \right) B^{-1} \sum_b h'_\theta(\ddot{p}_b, x_t). \tag{5.10}$$

Independence of the two samples ensures that $\mathbb{E}\widehat{\nabla}^B_\theta \mathcal{L}_t = \mathbb{E}_{\mathcal{D}} \nabla_\theta \mathcal{L}_t = \nabla_\theta \mathcal{L}(\mathcal{D};\theta)$, as desired. The variance of our estimate depends on B, the number of samples that we draw. Each of these samples is relatively expensive to compute because they require a forward pass through the network $\hat{h}_\theta(\dot{p}, x_t)$. If this varies significantly with $\dot{p}$ we might need a large number of samples to get a low-variance estimate of the gradient, which is computationally intensive.

An alternative is to optimize an upper bound on Equation (5.8). By using Jensen's inequality and the fact that the squared error function is convex we get that

$$\hat{\mathcal{L}}(D;\theta) \leq |D|^{-1} \sum_t \sum_{\dot{p} \sim \hat{F}_\phi(p|x_t,z_t)} \big(y_t - h_\theta(\dot{p}, x_t) \big)^2. \tag{5.11}$$

Taking the RHS of Equation (5.11) as the objective and calculating the gradient leads to a version of Equation (5.10) in which a single draw can be used instead of two independent draws. This objective is easy to implement in practice as it just involves drawing samples during training. This is well supported in deep network implementations because it is analogous to the data augmentation procedures that are commonly used to encourage invariance in deep networks. The analogy is more than just aesthetic—by optimizing this loss, we are essentially encouraging the network to be invariant to variations in the treatment that cannot be explained by our features and instrument. This encourages the network to ignore the effects of unobserved confounding variables.

However, we do not have theoretical guarantees that optimizing this upper bound on $\mathcal{L}(D;\theta)$ leads to good predictions on the interventional distribution. While it may converge more quickly because it exhibits lower variance, it will have worse asymptotic performance as it only approximates the desired loss function. We evaluated this tradeoff experimentally by comparing optimizing the upper bound to the more computationally expensive unbiased procedure.

5.5 Experiments

We evaluated our approach on both simulated and real data. We used simulations to assess DeepIV's ability to recover an underlying structural equation both in a low-dimensional domain with informative features and in a high-dimensional domain with features consisting of pixels of a handwritten image. We compared our approach to 2SLS and to a standard feed-forward network, evaluated the effectiveness of hyperparameter optimization, and contrasted our biased and unbiased loss functions with various numbers of samples underlying the SGD step. We also considered a real-world dataset where ground truth was not available, showing that we could replicate the findings of a previous study in a dramatically more automatic fashion.

5.5.1 Simulations

Our simulation models a richer version of the airline example described in Section 5.1. We assume that there are 7 customer types $s \in \{1, ..., 7\}$ that each ex-

hibit different levels of price sensitivity. Each type is represented as a one-hot vector e_i, and its effect is just its integer sensitivity $s_i = [1, 2, ..., 7] \cdot e_i$. We model the holiday effect on sales by letting the customer's price sensitivity vary continuously throughout the year according to a complex non-linear function, $\psi_t = 2\left((t-5)^4/600 + \exp\left[-4(t-5)^2\right] + t/10 - 2\right)$. The time of year t is an observed variable, generated as $t \sim \text{unif}(0, 10)$. Prices are a function of ψ_t and of the fuel price z, with the motivation that they are chosen strategically by the airline in order to move with average price sensitivity. In our example, the high demand that results from conferences breaks the conditional independence between our treatment variable p and the latent effects e, thereby violating the "unconfoundedness" assumption. We model this abstractly by generating our latent errors e with a parameter ρ that allows us to smoothly vary the correlation between p and e. Sales y are then generated as

$$y = 100 + (10 + p)s\psi_t - 2p + e, \quad p = 25 + (z + 3)\psi_t + v$$
$$z,\ v \sim \mathcal{N}(0, 1) \ \text{ and } \ e \sim \mathcal{N}(\rho v, 1 - \rho^2).$$

Our target structural equation is $\hbar(t, s, p) = (10 + p)s\psi_t - 2p$. To evaluate the model, we consider the interventional question, "What would sales have been if prices had been set to p'?" Thus the price in our test set is set deterministically over a *fixed grid* of price values that spans the range of training set prices.

Low dimensional domain Our simulation models a richer version of the airline example described in the introduction. We evaluated structural mean square error (MSE) while varying both the number of training examples and ρ, the correlation between e and p. In addition to Deep IV, we considered a regular feed-forward network (FFNet) with the same architecture as our outcome network, a non-parametric IV polynomial kernel regression [NonPar, Darolles et al., 2011] using Hayfield, Racine, et al. [2008]'s R implementation, and two-stage least squares (2SLS).

The results are summarized in Figure 5.4. The performance of NonPar, of 2SLS, and of our Deep IV model was mostly unaffected by changes in ρ, reflecting the fact that these models are designed to be resilient to unobserved confounders. For 1000 data points, NonPar's mean performance was better than 2SLS but failed to

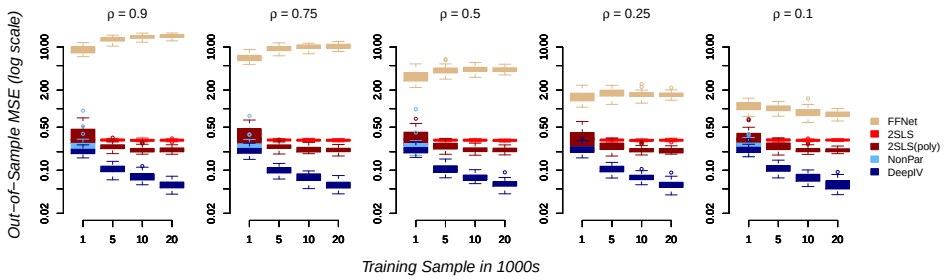

Figure 5.4: Out-of-sample predictive performance for different levels of correlation, ρ, between the treatment p and unobserved confounder e. Note that the test samples were generated with independent errors conditional upon a fixed grid of price values, breaking this correlation that existed in the training sample; this is why the feed-forward network did so poorly. Each model was fitted on 40 random samples from the DGP for each sample size and ρ-level.

match DeepIV. Because of NonPar's excessive computational requirements we were not able to fit it to the larger datasets. 2SLS is constrained by its homogeneity and linearity assumptions, and so did not improve with increasing amounts of data. Adding regularized polynomial basis functions to 2SLS (2SLS(poly)) gives some empirical improvements in performance over 2SLS on larger datasets but the procedure is not causally valid because it violates 2SLS's linearity assumptions. Both forms of 2SLS performed far better than FFNet which did a good job of estimating $\hat{h}(t,s,p) + \mathbb{E}[e|p]$ but a terrible job of recovering the true structural equation. As ρ dropped, decreasing $\mathbb{E}[e|p]$, FFNet's performance improved but even with low levels of correlation between p and e it remained far worse than simple 2SLS. This occurred because we evaluated the models with respect to a fixed grid of treatment values which induced a covariate shift at test time. In contrast, Deep IV was the best performing model throughout and its performance improved as the amount of data grew.

High dimensional feature space In real applications, we do not typically get to observe variables like *customer type* that cleanly delineate our training examples into explicit classes, but may instead observe a large number of features that correlate

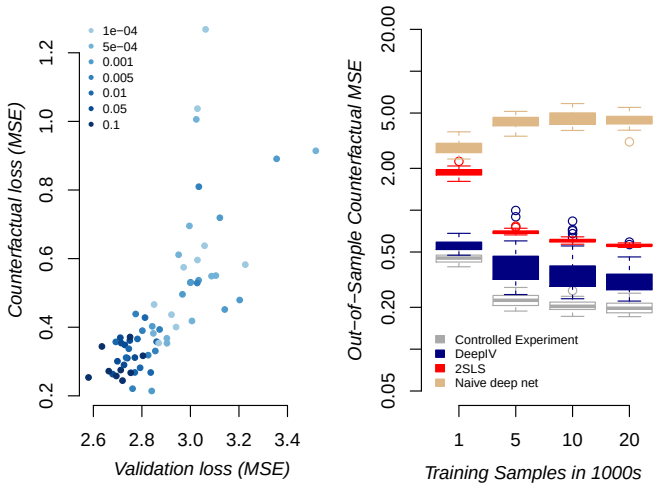

Figure 5.5: For the high-dimensional feature space problem we used a four-layer convolutional network to build an embedding of the image features which was concatenated with the observed time features and the instrument (first stage) and the treatment samples (second stage) and fed the resulting vector through another hidden layer before the output layer. *(Left)* Grid search over L2 and dropout parameters for the embedding used in the convolution network. *(Right)* Performance on an image experiment.

with such types. To simulate this, we replaced the customer type label $s \in \{0, 1, ..., 6\}$ with the pixels of the corresponding handwritten digit from the MNIST dataset [LeCun and Cortes, 2010]. The task remained the same, but the model was no longer explicitly told that there were 7 customer types and instead had to infer the relationship between the image data and the outcome.

In this far more challenging domain, performance is sensitive to the choice of hyperparameters, necessitating optimization on a validation set. Figure 5.5 shows an evaluation of the appropriateness of our loss function for hyperparameter tuning, comparing our validation-set loss after grid search over Dropout and L2-regularization parameters to test set loss. We found a clear linear relationship between the losses; the best performing validation set model was among the best five performing models under the true causal loss.

We can get an upper bound on the performance of a particular model architecture by comparing its performance to the same architecture trained on data from a simulated randomized experiment on the same data-generating process. We simulated this by generating the outcome y with independent noise e and by generating p uniformly at random over its support. Thus the controlled model had to solve a standard supervised learning problem where errors were generated independently and there was no test-time covariate shift. As before, the naive deep network also shared the same architecture in addition to taking the instrument as input. This experiment showed that DeepIV was able to make up most of the loss in estimating the causal effect that the naive network suffered by not accounting for the causal prediction problem. However, there was still a gap in performance: with 20 000 data points the controlled experiment achieved an average mean squared error of 0.20 while DeepIV managed 0.32.

5.5.2 Application: Search-advertisement position effects

Our experiments so far have considered synthetic data. We now evaluate the utility of our approach on real data for which we do not have access to ground truth. This means that we cannot evaluate models in terms of their predictions; instead, we show that we can replicate the results of a previously published study in a dramatically more automated fashion. Specifically, we examine how advertiser's position on the Bing search page (their "slot") affects the probability of a user click, allowing for different treatment effects for different advertiser-query pairs. For example, we aim to detect differences in the importance of ad position when Coke bids on the word "Coke" (an "on-brand query") versus when Pepsi bids on "Coke" (an "off-brand query") versus when Coke bids on "www.coke.com" (an "on-nav query", occurring when a user types a url in the search box by mistake) versus when Pepsi bids on "www.coke.com" (an "off-nav query"). This question was studied by Goldman and Rao [2014] using a nonparametric IV estimation approach that involved a detailed construction of optimal instruments, as well as a separate hand-coded classification of advertiser–query pairs into the four categories above.

Our goal is to replicate these results in an *automated* fashion. Advertiser position is correlated with latent user intent (e.g. when a user searches for "Coke", it is likely

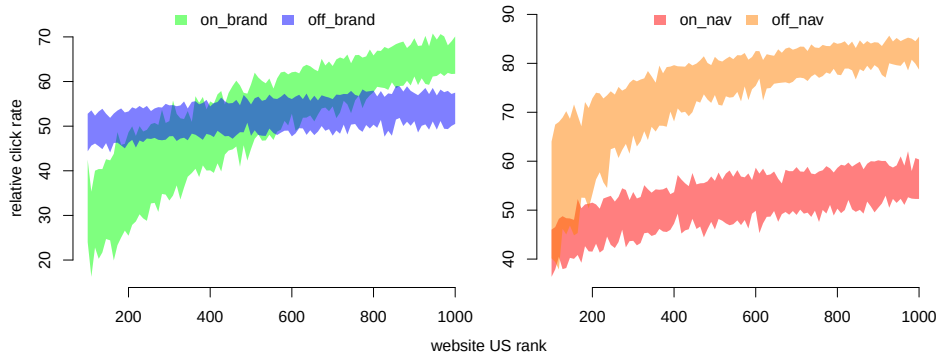

Figure 5.6: The inter-quartile range in advertiser-query specific estimates of the relative drop in click rate from moving from position 1 to position 2 (i.e. y-axis denotes $\frac{\text{cr}_1 - \text{cr}_2}{\text{cr}_1}\%$), as the popularity of the advertiser varies along the x-axis (as measured by visit rank on Alexa.com), for on-brand versus off-brand queries (left panel) and on-nav versus off-nav queries (right panel) for a single combination of advertiser and search-query text.

both that they will click and that "Coke" will be the top advertiser), so we need instruments to infer causation. The instruments proposed by Goldman and Rao [2014] are a series of indicators for experiments run by Bing in which advertiser–query pairs were randomly assigned to different algorithms that in turn scored advertisers differently in search auctions, resulting in random variation in position. Our estimation algorithm takes the experiment ID directly as an instrument.

As features, we gave the deep nets the query url and user query as text. The url was parsed into tokens on dashes and dots and these tokens were then parsed on punctuation and whitespace. Given the outcome variable (an indicator for user click), the features, and the instruments, we applied our methodology directly to the data without any of the additional feature engineering and construction of optimal instruments performed by Goldman and Rao [2014]. This approach was tractable despite the fact that the dataset contained over 20 million observations. Figure 5.6 shows the results. We were able to replicate the original study's broad findings, namely that (i) that for "on-brand" queries, position was worth more to small websites; and (ii) that the value of position for on-nav queries was much smaller than for off-nav queries. A naive non-causal regression found an unrealistic

average treatment effect (ATE; across sampled advertisers and queries) drop in click rate of 70% from 1st to 2nd position; in contrast, the causal estimate of the ATE was a more modest 12% drop.

5.6 Discussion

We have presented DeepIV, an approach that leverages instrument variables to train deep networks that directly estimate causal effects and validate the resulting models on held-out data. DeepIV significantly reduced error under interventions measured in simulation experiments and was able to replicate previous IV experiments without extensive feature engineering. In future work, we plan to discuss interference techniques for the DeepIV framework and explore how this approach generalizes to other causal graphs given appropriate assumptions.

Chapter 6

Valid Causal Inference with (Some) Invalid Instruments

This chapter develops a method to weaken instrumental variable assumptions that are required by the estimation procedure that we introduced in Chapter 5. We show that in settings with more than two candidate instruments, valid inference is possible under agreement assumptions that do not require that all the candidates are valid.

6.1 Introduction

As we demonstrated Chapter 5, instrumental variable (IV) methods are a powerful approach for estimating treatment effects: they are robust to unobserved confounders and they are compatible with a variety of flexible nonlinear function approximators [see e.g. Newey and Powell, 2003; Darolles et al., 2011; Hartford et al., 2017; Bennett et al., 2019; Singh et al., 2019], thereby allowing nonlinear estimation of heterogeneous treatment effects. Recall that in order to use an IV approach, one must make three assumptions which we described in detail in Section 5.3 on page 74. The first, *relevance*, asserts that the treatment is not independent of the instrument. This assumption is relatively unproblematic, because it can be verified with data. The second assumption, *unconfounded instrument*, asserts that the instrument and outcome do not share any common causes. This assumption cannot be verified directly, but in some cases it can be justified via knowledge

of the system; e.g. the instrument may be explicitly randomized or may be the result of some well understood random process. The final assumption, *exclusion*, asserts that the instrument's effect on the outcome is entirely mediated through the treatment. This assumption is even more problematic; not only can it not be verified directly, but it can be very difficult to rule out the possibility of direct effects between the instrument and the outcome variable. Indeed, there are prominent cases where purported instruments have been called into question for this reason. For example, in economics, the widely used "judge fixed effects" research design [Kling, 2006] uses random assignment of trial judges as instruments and leverages differences between different judges' propensities to incarcerate to infer the effect of incarceration on some economic outcome of interest [see Frandsen et al., 2019, for many recent examples]. Mueller-Smith [2015] points out that exclusion is violated if judges also hand out other forms of punishment (e.g. fines, a stern verbal warning etc.) that are not observed. Similarly, in genetic epidemiology "Mendelian randomization" [Davey Smith and Ebrahim, 2003] uses genetic variation as instruments to study the effects of some exposure on an outcome of interest. For example, given genetic markers that are known to be associated with a higher body mass index (BMI), we can estimate the effect of BMI on cardiovascular disease if we are prepared to assume that these genes only affect cardiovascular disease via their affect on BMI. This assumption fails it is possible that the same genetic markers influence the risk of cardiovascular disease via some other pathway. The possibility of such "direct effects"—referred to as "horizontal pleiotropy" in the genetic epidemiology literature—is regarded as a key challenge for Mendelian randomization [Hemani et al., 2018].

It is sometimes possible to identify *many* candidate instruments, each of which satisfies the relevance assumption; in such settings, demonstrating exclusion is usually the key challenge, though in principle unconfounded instrument could also be a challenge. For example, many such candidate instruments can be obtained in both the judge fixed effects and Mendelian randomization settings, where individual judges and genetic markers, respectively, are treated as different instruments. Rather than asking the modeler to gamble by choosing a single candidate about which to assert these untestable assumptions, this chapter advocates making a weaker assumption about the whole set of candidates. Most intuitively, we can assume

majority validity: that at least a majority of the candidate instruments satisfy all three assumptions, even if we do not know which candidates are valid and which are invalid. Or we can go further and make the still weaker assumption of *modal validity*: that the modal relationship between instruments and response is valid. Observe that modal validity is a weaker condition because if a majority of candidate instruments are valid, the modal candidate–response relationship must be characterized by these valid instruments. Modal validity is satisfied if, as Tolstoy might have said, "All happy instruments are alike; each unhappy instrument is unhappy in its own way."

This chapter introduces ModeIV, a robust instrumental variable technique. ModeIV allows the estimation of nonlinear causal effects and lets us estimate conditional average treatment effects that vary with observed covariates. It is simple to implement—it involves fitting an ensemble with a modal aggregation function—and is black-box in the sense that it is compatible with any valid IV estimator, which allows it to leverage any of the recent machine learning-based IV estimators. Despite its simplicity, ModeIV has strong asymptotic guarantees: we show consistency and that even on a worst-case distribution, it converges point-wise to an oracle solution at the same rate as the underlying estimators. We experimentally validated ModeIV using both a modified version of the demand simulation from Section 5.5.1 and a more realistic Mendelian randomization example modified from Hartwig et al. [2017]. In both settings—even with data with a very low signal-to-noise ratio—we observed ModeIV to be robust to exclusion-restriction bias and accurately recovered conditional average treatment effects.

6.2 Related work

Inference with invalid instruments in linear settings Much of the work on valid inference with invalid instruments is in the Mendelian randomization literature, where violations of the exclusion restriction are common. For a recent survey, see Hemani et al. [2018]. There are two broad approaches to valid inference in the presence of bias introduced by invalid instruments: averaging over the bias, or eliminating the bias with ideas from robust statistics. In the first setting, valid inference is possible under the assumption that each instrument introduces a random

bias, but that the mean of this process is zero (although this assumption can be relaxed [c.f. Bowden et al., 2015; Kolesár et al., 2015]). Then, the bias tends to zero as the number of instruments grow. Methods in this first broad class have the attractive property that they remain valid even if none of the instruments is valid, but they rely on strong assumptions that do not easily generalize to the nonlinear setting considered in this paper.

The second class of approaches to valid inference assumes that some fraction of the instruments are valid and then uses the fact that biased instruments are outliers whose effect can be removed by leveraging robust estimators. For example, by assuming *majority validity* and constant linear treatment effects[1], Kang et al. [2016] and Guo et al. [2018] show that it is possible to consistently estimate the treatment effect via a Lasso-style estimator that uses the sparsity of the ℓ_1 norm to remove invalid instruments. Under the same linearity and constant effect assumptions, Hartwig et al. [2017] showed that one can estimate the treatment effect under *modal validity* by estimating the mode of a set of Wald estimators. In this paper, we use the same modal insight as Hartwig et al., but generalize the approach to a nonlinear setting, thereby removing the strong assumption of constant treatment effects. Finally, Kuang et al. [2020] recently showed that, under majority validity, it is possible to leverage structure learning techniques produce a "summary (valid) IV" that can be plugged into downstream estimators. They focus on a setting with binary instruments, responses and confounders whereas we aim for a generic method that places no constraints on the data generating process beyond those necessary for identification.

Ensemble models Ensembles are widely used in machine learning as a technique for improving prediction performance by reducing variance [Breiman, 1996] and combining the predictions of weak learners trained on non-uniformly sampled data [Freund and Schapire, 1995]. These ensemble methods frequently use modal predictions via majority voting among classifiers, but they are designed to reduce variance. Both the median and mode of an ensemble of models have been explored as a way of improve robustness to outliers in the forecasting literature [Stock and

[1] That is, assuming that the true structural equation is some linear function of the treatment and invalid instruments, and that all units share the same treatment effect parameter, β.

Watson, 2004; Kourentzes et al., 2014], but we are not aware of any prior work that explicitly uses these aggregation techniques to eliminate bias from an ensemble.

Mode estimation If a distribution admits a density, the mode is defined as the global maximum of the density function. More generally, the mode can be defined as the limit of a sequence of modal intervals—intervals of width h that contains the largest proportion of probability mass—such that $x_{\text{mode}} = \lim_{h\to 0} \arg\max_x F([x - h/2, x + h/2])$. These two definitions suggest two estimation methods for estimating the mode from samples: either one may try to estimate the density function and the maximize the estimated function [Parzen, 1962], or one might search for midpoints of modal intervals from the empirical distribution functions. To find modal intervals, one can either fix an interval width, h, and choose x to maximize the number of samples within the modal interval [Chernoff, 1964], or one can solve the dual problem by fixing the target number of samples to fall into the modal interval and minimizing h [Dalenius, 1965; Venter, 1967]. We use this latter Dalenius–Venter approach as the target number of samples can be parameterized by the number of valid instruments, thereby avoiding the need to select a kernel bandwidth h.

6.3 ModeIV

In this paper, we assume we have access to a set of k candidate variables, $\mathcal{Z} = \{z_1, \ldots, z_k\}$, which are 'valid' instrumental variables if they satisfy relevance, exclusion and unconfounded instrument, and are 'invalid' otherwise. Denote the set of valid instruments, $\mathcal{V} := \{z_i : z_i \not\perp t, z_i \perp \epsilon, z_i \perp y | x, t, \epsilon\}$, and the set of invalid instruments, $\mathcal{I} = \mathcal{Z} \setminus \mathcal{V}$. We further assume that each valid instrument identifies the causal effect. In the additive confounder setting, this amounts to assuming that the unobserved confounder's effect on y is additive, such that $y = f(t, x, z_{i:i\in\mathcal{I}}) + \epsilon$ for some function f and $E[y|x, z_{i:i\neq j}, z_j] = \int f(t, x, z_{i:i\neq j}) dF(t|x, z_{i:i\neq j}, z_j)$ has a the same unique solution for all j in $\mathcal{V}$.

The ModeIV procedure requires the analyst to specify a lower bound $V \geq 2$ on the number of valid instruments and then proceeds in three steps.

1. Fit an ensemble of k estimates of the conditional outcome $\{\hat{f}_1, \ldots, \hat{f}_k\}$ using a non-linear IV procedure applied to each of the k instruments. Each $\hat{f}$ is a

function mapping treatment t and covariates x to an estimate of the effect of the treatment conditional on x.

2. For a given test point (t, x), select $[\hat{l}, \hat{u}]$ as the smallest interval containing V of the estimates $\{\hat{f}_1(t, x), \ldots, \hat{f}_k(t, x)\}$. Define $\hat{\mathcal{I}}_{\text{mode}} = \{i : \hat{l} \leq \hat{f}_i(t, x) \leq \hat{u}\}$ to be the indices of the instruments corresponding to estimates falling in the interval.

3. Return $\hat{f}_{\text{mode}}(t, x) = \frac{1}{|\hat{\mathcal{I}}_{\text{mode}}|} \sum_{i \in \hat{\mathcal{I}}_{\text{mode}}} \hat{f}_i(t, x)$

Figure 6.1 shows this procedure graphically. The idea is that the estimates from the valid instruments will tend to cluster around the true value of the effect, $E[y|\text{do}(t), x]$. We assume that the most common effect is a valid one; i.e., that the modal effect is valid. To estimate the mode, we look for the tightest cluster of points which, by definition, are the points contained in $\hat{\mathcal{I}}_{\text{mode}}$. Intuitively, each estimate in this interval should be approximately valid and hence approximates the modal effect. Finally, we average these estimates to reduce variance.

The next theorem formalizes this intuition by showing that ModeIV asymptotically identifies and consistently estimates the causal effect.

Theorem 5. *Fix a test point (t, x) and let $\hat{\beta}_1, \ldots, \hat{\beta}_k$ be estimators of the causal effect of t at x corresponding to k (possibly invalid) instruments. E.g., $\hat{\beta}_j = \hat{f}_j(t, x)$. Denote the true effect as $\beta = E[y|do(t), x]$. Suppose that*

1. *(consistent estimators) $\hat{\beta}_j \to \beta_j$ almost surely for each instrument. In particular, $\beta_j = \beta$ whenever the jth instrument is valid.*

2. *(modal validity) At least v of the instruments are valid, and no more than $v - 1$ of the invalid instruments agree on an effect. That is, v of the instruments yield the same estimand if and only if all of those instruments are valid.*

Let $[\hat{l}, \hat{u}]$ be the smallest interval containing v of the instruments and let $\hat{\mathcal{I}}_{\text{mode}} = \{i : \hat{l} \leq \hat{\beta}_i \leq \hat{u}\}$. Then,

$$\sum_{i \in \hat{\mathcal{I}}_{\text{mode}}} \hat{w}_i \hat{\beta}_i \to \beta$$

almost surely, where $\hat{w}_i, w_i$ are any non-negative set of weights such that each $\hat{w}_i \to w_i$ a.s. and $\sum_{i \in \hat{\mathcal{I}}_{\text{mode}}} w_i = 1$.

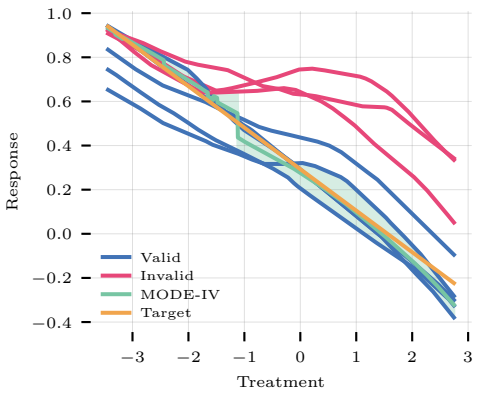

Figure 6.1: Example of the ModeIV algorithm with 7 candidates (4 valid and 3 invalid) from the biased demand simulation (see Section 6.4.1). The 7 estimators shown in the plot are each trained with a different candidate, and at every test point t, the mode of the 7 predictions is computed point-wise. The region highlighted in green contains the 3 predictions that formed part of the modal interval for each given input. The ModeIV prediction—the mean of the 3 closest prediction—is shown in solid green.

We defer all proofs to the Section C of the supplementary material.

Of course, the ModeIV procedure can be generalized to allow estimators of the mode that are different from the one used in Steps 2 and 3. The key advantage of the Dalenius–Venter modal estimator is that the optimal choice for its only hyper-parameter, V, does not depend on the distribution of the estimators at a given test point. By contrast, kernel density-based modal estimators require tuning a length-scale parameter, where the optimal choice may vary as a function of the test point, (t, x). It is also straightforward to implement and relatively insensitive to the choice of V. The procedure as a whole is, however, k times more computationally expensive than running single estimation procedure at both training and test time.

Despite its simplicity, ModeIV has strong point-wise worst-case guarantees. Theorem 6 shows that if each estimate is bounded,[2] then even in the worst case where $v-1$ invalid candidates all agree on an effect, ModeIV converges at the same rate as the underlying estimators to the solution of an oracle that uniformly averages the valid instruments. In particular, if the estimators achieve the parametric rate, $1/\sqrt{n}$, in the number of instances n, then ModeIV also converges at $1/\sqrt{n}$.

Theorem 6. *For some test point (t,x), let $\mathcal{Z} = \{\hat{\beta}_1, \ldots, \hat{\beta}_k\}$ be k estimates of the causal effect of t at x. Assume,*

*[**Bounded estimates**] Each estimate is bounded by some constants, $[a_i, b_i]$*

*[**Convergent estimators**] Each estimator converges in mean squared error at a rate n^{-r} (where $r = \frac{1}{2}$ if the estimator achieves the parametric rate), and hence each estimator has finite variance, $Var(\hat{\beta}_i) = \frac{\sigma_i}{n^{-2r}}$ for some σ_i.*

Then, if $\sigma = \max_{i \in \mathcal{V}} \sigma_i$ there exists a, C, such that $E[(\text{ModeIV}(\mathcal{Z}) - \beta)^2 - (\frac{1}{v}\sum_{i \in \mathcal{V}} \hat{\beta}_i - \beta)^2] \leq 9kC\sigma n^{-r}$.

6.4 Experiments

We studied ModeIV empirically in two simulation settings. First, we investigated the performance of ModeIV for non-linear effect estimation as the proportion of invalid instruments increased for various amounts of direct effect bias. Second, we applied ModeIV to a realistic Mendelian randomization (MR) simulation to estimate heterogeneous treatment effects. For all experiments, we use DeepIV [Hartford et al., 2017] as the nonlinear estimator. The existing methods for addressing bias from invalid instruments are designed for the linear setting, so as baselines we compare to DeepIV with oracle access to the set of valid instruments (DeepIV-opt); the ensemble mean (Mean) which tests whether any performance improvements that we observe are driven by variance reduction from ensembling; and a naive approach that fits a single instance of DeepIV treating all instruments as valid (DeepIV-all). For the MR experiments we also compare to Guo et al. [2018]. The heterogeneous effects in our MR simulation violates Guo et al.'s linearity assumption, but their

[2] Boundedness is benign as long as we are not extrapolating too far outside of the range of data we observe: standard estimators do not typically make predictions for $E[y|\text{do}(t), x]$ outside of the range $[\min_i y_i, \max_i y_i]$ of observed y_i's.

method is designed with MR in mind so the comparison illustrates the effect of incorrectly assuming linearity in this setting. In Section C.7 of the appendix, we also evaluated the performance of ModeIV when the true effect is linear on Guo et al.'s MR data generating process; we found that as long as a large enough sample is used, it accurately recovered the true effect.

6.4.1 Biased demand simulation

We evaluated the effect of invalid instruments on estimation by modifying the low dimensional demand simulation from Chapter 5.5.1 to include multiple candidate instruments. The original demand simulation models a scenario where the treatment effect varies as a function, ψ, of time, x_0, and other observed covariates x.

$$
\begin{aligned}
&z_{1:k}, \nu \sim \mathcal{N}(0,1) \quad x_0 \sim \text{unif}(0,10) \quad e \sim \mathcal{N}(\rho\nu, 1-\rho^2), \\
&t = 25 + (z^T \beta^{(zt)} + 3)\psi(x_0) + \nu \\
&y = 100 + 10 x_{1:d}^T \beta^{(x)} \psi(x_0) + \underbrace{(x_{1:d}^T \beta^{(x)} \psi(x_0) - 2)t}_{\text{Treatment effect}} \\
&\quad + \underbrace{\gamma \sin(z^T \beta^{(zy)})}_{\text{Exclusion violation}} + e
\end{aligned}
$$

We highlight the differences between this data generating process and the original in red: here we have k instruments whose effect on the treatment is parameterized by $\beta^{(zt)}$, instead of a single instrument in the original; we include an exclusion violation term which introduces bias into standard IV approaches whenever γ is non-zero. The vector $\beta^{(zy)}$ controls the direct effect of each instrument: invalid instruments have nonzero $\beta_i^{(zy)}$ coefficients, while valid instrument coefficients are zero.

We fitted an ensemble of k different DeepIV models that were each trained with a different instrument z_i. In Figure 6.2, we compare the performance of ModeIV with three baselines: DeepIV with oracle access to the set of valid instruments (DeepIV-opt); the ensemble mean (Mean); and a naive approach that fit a single instance of DeepIV treating all instruments as valid (DeepIV-all). The x-axis of the plots indicates the scaling factor γ, which scales the amount of bias introduced via violations of the exclusion restriction.

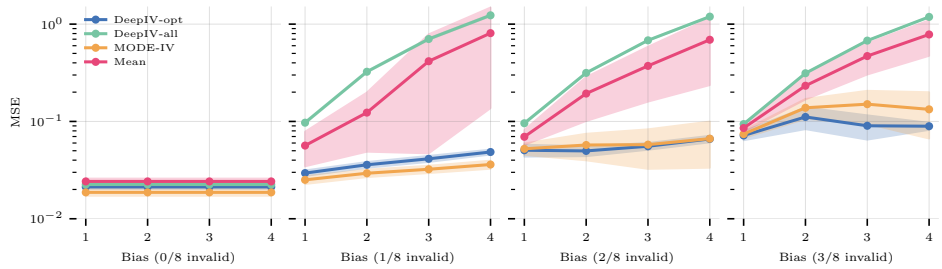

Figure 6.2: Performance on the biased demand simulation for various numbers of invalid instruments. The x-axis shows, γ, the scaling factor that scales the amount of exclusion violation bias.

All methods performed well when all the instruments were valid. Once the methods had to contend with invalid instruments, Mean and DeepIV-all performed significantly worse than ModeIV because of both methods' sensitivity to the biased instruments. ModeIV's mean squared error closely tracked that of the oracle method as the number of biased instruments increased, and the raw mean squared errors of both methods also increased as the number of valid instruments in the respective ensembles correspondingly fell.

Sensitivity When using ModeIV, one key practical question that an analyst faces is choosing V, the lower bound on the number of valid instruments. We evaluated the importance of this choice in Figure 6.3 by testing the performance of ModeIV across the full range of choices for V with different numbers of biased instruments. We found that, as expected, the best performance was achieved when V equaled the true number of valid instruments, but also that similar levels of performance could be achieved with more conservative choices of V. That said, with only 5 valid instruments, ModeIV tended to perform worse when V was set too small. For an illustration of why this occurs, consider Figure 6.1 which visualizes the ModeIV procedure. In the figure, there are a number of regions of the input space where the invalid instruments agreed by chance (e.g. $t \in [-1, 0.]$), so these regions bias ModeIV for small mode set sizes. Overall, we observed that setting $V = \lfloor k/2 \rfloor$ (where k is the number of instruments) tended to work well in practice.

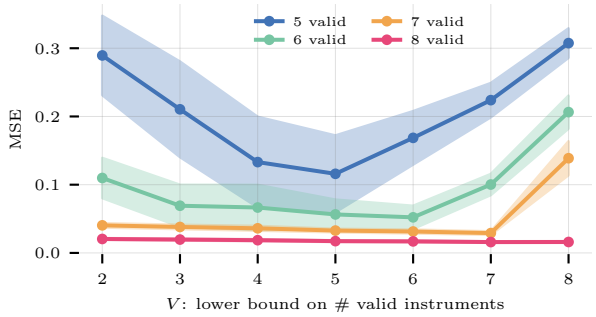

Figure 6.3: ModeIV's sensitivity to the choice of number of valid instruments parameter V. Best performance is achieved when V is equal to the true number of valid instruments, but the method is relatively insensitive to more conservative choices of V.

Asymptotically, ModeIV remains consistent when fewer than half of the instruments are valid, but when this is the case there are far more ways that Assumption 2 of Theorem 5 can be violated. This is illustrated in Figure 6.1 which shows that there are a number of regions where the bias instruments agree by chance. Because of this, we recommend only using ModeIV when one can assume that the majority of instruments are valid, unless one has prior knowledge to justify *modal validity* without assuming the majority of instruments are valid[3].

Bootstrap inference When using deep learning-based estimators, one typically does not have closed form expressions for confidence intervals or knowledge of the joint distribution over estimators, so we evaluated the performance of ModeIV with bootstrap confidence intervals. Table 6.1 summarizes the results. On this simulation we found that bootstrap confidence intervals with ModeIV were reasonably accurate as long as V was set low enough: with $V = 2$ or 3, coverage was above 90% for 95% confidence intervals. With larger settings of V, we found worse performance as the narrower intervals did not account for occasional selection of biased instruments. Figure 6.4 shows a plot of the bootstrap confidence intervals for both the average dose-response curve and various conditional averages. The intervals are narrow

[3]For example, if direct effects are strictly monotone and disagree, chance agreements among invalid instruments can only occur in a finite number of locations.

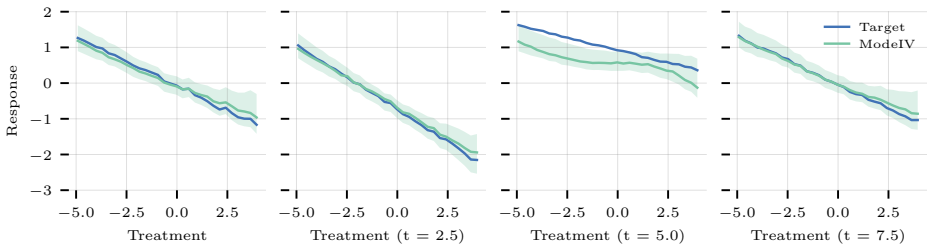

Figure 6.4: Bootstrap 95% confidence interval for the (conditional) dose-response curve of ModeIV with $V = 3$ for the biased demand simulation. The left plot shows the unconditional curve, and the remaining three curves are conditioned on the time variable, $t \in \{2.5, 5, 7.5\}$.

enough that they are able to indicate the true dose-response curve, while still providing reasonable coverage.

	4/7 valid	5/7 valid	6/7 valid
ModeIV-2	90.26%	94.79%	94.27%
ModeIV-3	87.82%	92.13%	92.79%
ModeIV-4	85.30%	90.10%	91.16%
ModeIV-5	72.80%	85.88%	88.06%
ModeIV-6	51.87%	70.63%	83.80%
ModeIV-7	34.12%	45.05%	66.44%
DeepIV-All	16.85%	18.48%	19.89%
DeepIV-Opt	95.07%	95.85%	95.79%
Ens-Mean	34.37%	45.29%	66.65%

Table 6.1: Empirical average point-wise coverage for bootstrap 95% confidence intervals on the biased demand simulation with 7 instruments.

6.4.2 Mendelian randomization simulation

For the second experiment, we evaluated our approach on simulated data adapted from Hartwig et al. [2017], which is designed to reflect violations of the exclusion restriction in Mendelian randomization studies.

Instruments, z_i, represent SNPs—locations in the genetic sequence where there is frequent variation among people—modeled as random variables drawn from a

Binomial$(2, p_i)$ distribution corresponding to the frequency with which an individual gets one or both rare genetic variants. The treatment and response are both continuous functions of the instruments with Gaussian error terms. The strength of the instrument's effect on the treatment, α_i, and direct effect on the response, δ_i, are both drawn from Uniform$(0.01, 0.2)$ distributions for all i. For all experiments we used 100 candidate instruments and varied the number of valid instruments from 50 to 100 in increments of 10; we set δ_i to 0 for all valid instruments. More formally,

$$z_i \sim \text{Binomial}(2, p_i) \quad \beta(x) := \text{round}(x^T \gamma^{(xt)}, 0.1).$$
$$t := \sum_{j=1}^{K} \alpha_j z_j + \rho u + \epsilon_x, \; y := \beta(x)t + \sum_{j=1}^{K} \delta_j z_j + u + \epsilon_y$$

In the original Hartwig et al. simulation, the treatment effect β was fixed for all individuals. Here, we make the treatment effect vary as a function of observable characteristics to model a scenario where treatments may affect different sub-populations differently. We simulate this by making the treatment effect, $\beta(x)$, a sparse linear function of observable characteristics, $x \in R^{10}$, where 3 of the 10 coefficients, $\gamma_i^{(xt)}$ were sampled from $U(0.2, 0.5)$ and the remaining $\gamma_i^{(xt)}$ were set to 0. We introduce non-linearity by rounding to the nearest 0.1, which makes the learning problem harder, while making it easier to visually show differences between the fitted functions and their targets.

Mendelian randomization problems tend to have low signal-to-noise ratios because typical response variables tend to be influenced by a large number of unobserved factors; in this simulation, the treatment explains only 1-3% of the response variance. This makes the setting challenging for neural networks, which tend to perform best on low-noise regimes.To address this, we leveraged the inductive bias that the data is conditionally linear in the treatment effect, by using a neural network to parameterize the slope of the treatment variable rather than outputting the response directly. So, for these problems, we defined $\hat{f}(t, x) = g(\phi(x))t + h(\phi(x))$, where $g(\cdot)$ and $h(\cdot)$ are linear layers that act on a shared representation $\phi(x)$.

Among the DeepIV-based benchmarks, the general trends that we observed on the Mendelian randomization simulation—summarized in Table 6.2—were similar to those we observed in the biased demand simulation: DeepIV-all performed poorly

and ModeIV closely tracked the performance of our oracle, DeepIV-opt. On this simulation the mean ensemble (Mean) achieved stronger performance, but still did not match ModeIV.

Aside from the heterogeneity induced by $\beta(x)$, this data generating process is linear so we can use it to evaluate the effect of heterogeneity on methods that assume a constant linear treatment effect. Guo et al.'s Two-Stage Hard Thresholding (TSHT) accurately recovered the average treatment effect (ATE) on this problem (see Table C.3 in the appendix), but as Table 6.2 shows, it was not able to match the performance of ModeIV in predicting $E[y|\text{do}(t), x]$. Note that this is a challenging baseline: with the sample size used in this simulation (400 000), Guo et al.'s method has very few false positives in identifying the valid instruments (see Table C.3 in the appendix), so it is essentially running two-stage least squares on a linear model with a 'random' (from the perspective of the model) coefficient $\beta(x)$. Linear models are optimal in this setting, so ModeIV can only outperform TSHT by leveraging the interaction between x and β. That said, there is a trade-off: TSHT was unbiased in predicting the ATE, but both DeepIV-Opt and ModeIV picked up some bias: DeepIV-Opt over-estimated the conditional average treatment effect by 0.035 and ModeIV by 0.045 for true effect sizes that range between -0.3 and 0.3 (see Table C.1 in the appendix). This bias is small but significant, and is also reflected in lower coverage from bootstrap confidence intervals. We found that when targeting a 90% confidence interval, DeepIV-Opt achieved only 80% coverage and ModeIV only managed 60% (Table C.2 in the appendix).

Conditional average treatment effects and bootstrap inference Figure 6.5 shows the predicted dose–response curves for a variety of different levels of the true treatment effect. The six plots correspond to six different subspaces of x that all have the same true conditional treatment effect. Each of the light blue lines shows ModeIV's prediction for a different value of x. The model is not told that the true β is constant for each of these sub-regions, but instead has to learn that from data so there is some variation in the slope of each prediction. Despite this, the majority of predicted curves match the sign of the treatment effect for each subgroup of x and accurately predicted the relative differences between the subgroups.

Model	50% valid	60% valid	70% valid	80% valid	90% valid	100% valid
DeepIV (valid)	0.035 ± (0.001)	0.035 ± (0.001)	0.034 ± (0.001)	0.034 ± (0.001)	0.032 ± (0.0)	0.024 ± (0.001)
MODE-IV 30%	0.037 ± (0.001)	0.037 ± (0.001)	0.038 ± (0.001)	0.039 ± (0.001)	0.041 ± (0.001)	0.032 ± (0.001)
MODE-IV 50%	0.037 ± (0.001)	0.037 ± (0.001)	0.038 ± (0.001)	0.039 ± (0.001)	0.04 ± (0.001)	0.032 ± (0.001)
Mean	0.041 ± (0.001)	0.041 ± (0.001)	0.043 ± (0.001)	0.043 ± (0.001)	0.045 ± (0.001)	0.036 ± (0.001)
DeepIV (all)	0.099 ± (0.004)	0.116 ± (0.003)	0.149 ± (0.005)	0.149 ± (0.005)	0.142 ± (0.003)	0.025 ± (0.0)
TSHT	0.089 ± (0.005)	0.075 ± (0.003)	0.073 ± (0.003)	0.073 ± (0.003)	0.072 ± (0.003)	0.072 ± (0.003)

Table 6.2: Performance on the Mendelian randomization simulation for various proportions of valid instruments. The ensemble methods performed far better than the DeepIV model, which treated all instruments as valid, and ModeIV, which gave significantly better performance than the mean ensemble, was close to the performance of DeepIV on the valid instruments.

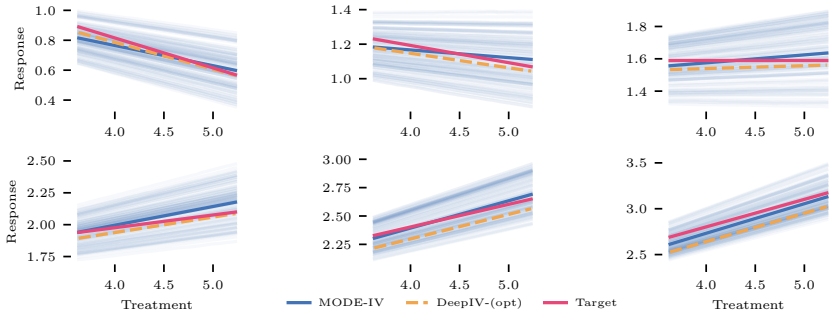

Figure 6.5: Estimated conditional dose–response curves for the Mendelian randomization simulation. Each light blue curve shows ModeIV's estimate $f(t,x)$ for some IID sample of x; each figure's dark curve represents the average over all samples of x. The plots show the different subsets of the range, x, where true slope $\beta(x)$ is *(left to right)* $-0.2, -0.1, \ldots, 0.3$.

6.5 Discussion and Limitations

The conventional wisdom for IV analysis is: if you have many (strong) instruments and sufficient data, you should use all of them so that your estimator can maximize statistical efficiency by weighting the instruments appropriately. This remains true in our setting—indeed, DeepIV trained on the valid instruments typically outperformed any of the ensemble techniques—but of course requires a procedure for identifying the set of valid instruments. In the absence of such a procedure, falsely assuming

that all candidate instruments are valid can lead to large biases, as illustrated by the poor performance of DeepIV-all. ModeIV gives up some efficiency by filtering instruments, but it gains robustness to invalid instruments with strong worst case asymptotic guarantees, and in practice we found that the loss of efficiency was negligible. Of course, that empirical finding will vary across settings. A useful future direction would find a procedure for recovering the set of valid instruments to further reduce the efficiency trade-offs.

There are, however, some important settings where ModeIV will either not work or require more careful assumptions. First, our key assumption was that each valid instrument consistently estimates the same function, $f(t, x)$. In settings with discrete treatments, one typically only identifies a "(conditional) local average treatment effect" (CLATE / LATE respectively) for each instrument. The LATE for instrument i can be thought of as the average treatment effect for the sub-population that changes its behavior in response to a change in the value of instrument i; if the LATEs differ across instruments, this implies that each instrument will result in a different estimate of $E[\hat{f}_i(t, x)]$ regardless of whether any of the instruments are invalid. In such settings, ModeIV will return the average of the V closest $\hat{f}_i(t, x)$'s, but one would need additional assumptions on how these estimates cluster relative to biased estimates to apply any causal interpretation to this quantity. The alternative is the approach that we take here: assume that a common function $f(t, x)$ is shared across all units and allow for heterogeneous treatment effects by allowing the treatment effect to vary as a function of observed covariates x. This shared heterogeneous effect assumption is weaker than the assumptions made in prior work on robust IV, which requires a "constant effect" assumption that every individual responds in exactly the same way to the treatment via a linear parameter, β.

Second, we assume that each of the valid instruments is unconfounded. This is easiest to achieve in settings where each instrument is independent. ModeIV does extend to settings where some instruments are confounded or all the instruments share a common cause, but the conditions for valid inference are more delicate because one has to ensure all backdoor paths between the instruments and response are blocked. For example, if all the instruments share a common cause, u, then the set of "valid" instruments is only valid if we control for the invalid candidates, such that we block the path $z_{\text{valid}} \leftarrow u \rightarrow z_{\text{invalid}} \rightarrow y$ (see Appendix C.3).

Bibliography

Allamanis, Miltiadis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles A. Sutton (2016). "Learning Continuous Semantic Representations of Symbolic Expressions". In: *arXiv preprint*. arXiv: 1611.01423.

Anandkumar, Animashree, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky (2014). "Tensor decompositions for learning latent variable models". In: *The Journal of Machine Learning Research* 15.1, pp. 2773–2832.

Angrist, J. D., G.W. Imbens, and D. B. Rubin (1996). "Identification of causal effects using instrumental variables". In: *Journal of the American Statistical Association* 91.434, pp. 444–455.

Angrist, Joshua D and Jörn-Steffen Pischke (2008). *Mostly harmless econometrics: An empiricist's companion*. Princeton university press.

Athey, Susan and Guido Imbens (2016). "Recursive partitioning for heterogeneous causal effects". In: *Proceedings of the National Academy of Sciences* 113, pp. 7353–7360.

Balke, Alexander and Judea Pearl (1997). "Bounds on treatment effects from studies with imperfect compliance". In: *Journal of the American Statistical Association* 92.439, pp. 1171–1176.

Battaglia, Peter W., Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. (2018). "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint*. arXiv: 1806.01261.

Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal (2019). "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116.32, pp. 15849–15854.

Bengio, Y., A. Courville, and P. Vincent (2013). "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8, pp. 1798–1828. ISSN: 0162-8828.

Bengio, Yoshua, Andrea Lodi, and Antoine Prouvost (2018). "Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon". In: *arXiv preprint*. arXiv: 1811.06128.

Bennett, Andrew, Nathan Kallus, and Tobias Schnabel (2019). "Deep Generalized Method of Moments for Instrumental Variable Analysis". In: *arXiv preprint arXiv:1905.12495*. URL: https://arxiv.org/abs/1905.12495.

Berg, Rianne van den, Thomas N Kipf, and Max Welling (2017). "Graph Convolutional Matrix Completion". In: *arXiv preprint arXiv:1706.02263*.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blei, David M (2012). "Probabilistic topic models". In: *Communications of the ACM* 55.4, pp. 77–84.

Bloem-Reddy, Benjamin and Yee Whye Teh (2019). "Probabilistic symmetry and invariant neural networks". In: *arXiv preprint*. arXiv: 1901.06082.

Blundell, Richard, Xiaohong Chen, and Dennis Kristensen (2007). "Semi-Nonparametric IV Estimation of Shape-Invariant Engel Curves". In: *Econometrica* 75, pp. 1630–1669.

Bottou, L. (2010). "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, pp. 177–186.

Bottou, Léon, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson (2013). "Counterfactual reasoning and learning systems: The example of computational advertising". In: *The Journal of Machine Learning Research* 14.1, pp. 3207–3260.

Bousquet, Olivier and Léon Bottou (2008). "The tradeoffs of large scale learning". In: *Advances in neural information processing systems (NIPS)*, pp. 161–168.

Bowden, Jack, George Davey Smith, and Stephen Burgess (June 2015). "Mendelian randomization with invalid instruments: effect estimation and bias detection through Egger regression". In: *International Journal of Epidemiology* 44.2, pp. 512–525.

Breiman, Leo (1996). "Bagging Predictors". In: *Machine Learning* 24.2, pp. 123–140.

Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2014). "Spectral networks and locally connected networks on graphs". In: *ICLR*.

Camerer, C.F., T.H. Ho, and J.K. Chong (2004). "A Cognitive Hierarchy Model of Games". In: *Quarterly Journal of Economics* 119.3, pp. 861–898.

Camerer, Colin (2003). *Behavioral game theory: Experiments in strategic interaction*. Princeton University Press.

Candès, Emmanuel J and Benjamin Recht (2009). "Exact matrix completion via convex optimization". In: *Foundations of Computational mathematics*.

Cheeseman, Peter, Bob Kanefsky, and William M. Taylor (1991). "Where the Really Hard Problems Are". In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. IJCAI '91. Morgan Kaufmann, pp. 331–337.

Chen, X. and D. Pouzo (2012). “Estimation of Nonparametric Conditional Moment Models With Possibly Nonsmooth Generalized Residuals”. In: *Econometrica* 80, pp. 277–321.

Chernoff, Herman (Dec. 1964). “Estimation of the mode”. en. In: *Annals of the Institute of Statistical Mathematics* 16.1, pp. 31–41.

Clark, Christopher and Amos J. Storkey (2015). “Training Deep Convolutional Neural Networks to Play Go”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 1766–1774.

Clarke, Edmund, Armin Biere, Richard Raimi, and Yunshan Zhu (2001). “Bounded model checking using satisfiability solving”. In: *Formal Methods in System Design* 19.1, pp. 7–34.

Cohen, Taco and Max Welling (20–22 Jun 2016). “Group Equivariant Convolutional Networks”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 2990–2999.

Cohen, Taco S, Mario Geiger, and Maurice Weiler (2019). “A general theory of equivariant cnns on homogeneous spaces”. In: *Advances in neural information processing systems* 32, pp. 9145–9156.

Cohen, Taco S., Mario Geiger, Jonas Koehler, and Max Welling (Feb. 2018). “Spherical CNNs”. en. In: *Proceedings of the 6th International Conference on Learning Representations (ICLR), 2018*. arXiv: 1801.10130. (Visited on 04/07/2020).

Costa-Gomes, M., V.P. Crawford, and B. Broseta (2001). “Cognition and Behavior in Normal-Form Games: An Experimental Study”. In: *Econometrica* 69.5, pp. 1193–1235.

Crawford, James M. and Larry D. Auton (1996). “Experimental Results on the Crossover Point in Random 3SAT”. In: *Artificial Intelligence* 81, pp. 31–57.

Dalenius, Tore (1965). “The Mode–A Neglected Statistical Parameter”. en. In: *Journal of the Royal Statistical Society. Series A (General)* 128.1, p. 110.

Darolles, Serge, Yanqin Fan, Jean-Pierre Florens, and Eric Renault (2011). “Nonparametric instrumental regression”. In: *Econometrica* 79.5, pp. 1541–1565.

Davey Smith, George and Shah Ebrahim (2003). “‘Mendelian randomization’: can genetic epidemiology contribute to understanding environmental determinants of disease?” In: *International journal of epidemiology* 32.1, pp. 1–22.

De Finetti, B. (1930). “Fuzione caratteristica di un fenomeno aleatorio.” In:

Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst (2016). “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in Neural Information Processing Systems*, pp. 3844–3852.

Deng, Zhiwei, Rajitha Navarathna, Peter Carr, Stephan Mandt, Yisong Yue, Iain Matthews, and Greg Mori (2017). "Factorized variational autoencoders for modeling audience reactions to movies". In:

Duvenaud, David K, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams (2015). "Convolutional networks on graphs for learning molecular fingerprints". In: *Advances in neural information processing systems*.

Dziugaite, Gintare Karolina and Daniel M Roy (2015). "Neural network matrix factorization". In: *arXiv preprint arXiv:1511.06443*.

Edelman, B., M. Ostrovsky, and M. Schwarz (2007). "Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords". In: *The American Economic Review* 97.1, pp. 242–259.

Evans, Richard, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette (2018). "Can Neural Networks Understand Logical Entailment?" In: *arXiv preprint*. arXiv: 1802.08535.

Finkler, Ulrich and Kurt Mehlhorn (1997). "Runtime prediction of real programs on real machines". In: *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '97. Society for Industrial and Applied Mathematics, pp. 380–389.

Frandsen, Brigham R, Lars J Lefgren, and Emily C Leslie (2019). *Judging Judge Fixed Effects*. Tech. rep. National Bureau of Economic Research.

Fréchette, Alexandre, Neil Newman, and Kevin Leyton-Brown (2016). "Solving the Station Repacking Problem". In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. AAAI '16. AAAI Press, pp. 702–709.

Freund, Yoav and Robert E Schapire (1995). "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *European conference on computational learning theory*. Springer, pp. 23–37.

Geirhos, Robert, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann (2020). "Shortcut Learning in Deep Neural Networks". In: *arXiv preprint arXiv:2004.07780*.

Getoor, Lise and Ben Taskar (2007). *Introduction to statistical relational learning*. MIT press.

Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl (2017). "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. ICML '17. JMLR, pp. 1263–1272.

Goldman, Mathew and Justin M Rao (2014). "Experiments as instruments: heterogeneous position effects in sponsored search auctions". In: *EAI Endorsed Trans. Serious Games*.

Goldstein, A. A. (Sept. 1964). “Convex programming in Hilbert space”. In: *Bull. Amer. Math. Soc.* 70.5, pp. 709–710.

Guo, Zijian, Hyunseung Kang, T Tony Cai, and Dylan S Small (2018). “Confidence intervals for causal effects with invalid instruments by using two-stage hard thresholding with voting”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 80.4, pp. 793–815.

Hall, Peter and Joel L. Horowitz (Dec. 2005). “Nonparametric methods for inference in the presence of instrumental variables”. In: *Ann. Statist.* 33.6, pp. 2904–2929.

Hamilton, Will, Zhitao Ying, and Jure Leskovec (2017a). “Inductive representation learning on large graphs”. In: *Advances in Neural Information Processing Systems*.

Hamilton, William L., Rex Ying, and Jure Leskovec (2017b). “Representation learning on graphs: Methods and applications”. In: *arXiv preprint*. arXiv: 1709.05584.

Harper, F. Maxwell and Joseph A. Konstan (2015). “The MovieLens Datasets: History and Context.” In: *ACM Transactions on Interactive Intelligent Systems*.

Hartford, Jason, Greg Lewis, Kevin Leyton-Brown, and Matt Taddy (2017). “Deep IV: A flexible approach for counterfactual prediction”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 1414–1423.

Hartford, Jason S., Devon R. Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh (2018). “Deep Models of Interactions Across Sets”. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 1914–1923.

Hartwig, Fernando Pires, George Davey Smith, and Jack Bowden (2017). “Robust inference in summary data Mendelian randomization via the zero modal pleiotropy assumption”. In: *International journal of epidemiology* 46.6, pp. 1985–1998.

Hayfield, Tristen, Jeffrey S Racine, et al. (2008). “Nonparametric econometrics: The np package”. In: *Journal of statistical software* 27.5, pp. 1–32.

Hemani, Gibran, Jack Bowden, and George Davey Smith (2018). “Evaluating the potential role of pleiotropy in Mendelian randomization studies”. In: *Human molecular genetics* 27.2, pp. 195–208.

Hernán, Miguel A and James M Robins (2020). “Causal Inference: What If”. en. In: p. 311.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation*.

Holmes, Michael V, Leslie A Lange, Tom Palmer, Matthew B Lanktree, Kari E North, Berta Almoguera, Sarah Buxbaum, Hareesh R Chandrupatla, Clara C Elbers, Yiran Guo, et al. (2014). “Causal effects of body mass index on

cardiometabolic traits and events: a Mendelian randomization analysis". In: *The American Journal of Human Genetics* 94.2, pp. 198–208.

Hutter, Frank, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown (2014). "Algorithm runtime prediction: methods & evaluation". In: *Artificial Intelligence* 206, pp. 79–111.

Imbens, Guido W. and Donald B. Rubin (Apr. 2015). *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction*. en. 1st ed. Cambridge University Press. DOI: 10.1017/CBO9781139025751. (Visited on 02/27/2020).

Johansson, F. D., U. Shalit, and D. Sontag (2016). "Learning Representations for Counterfactual Inference". In: *Proceedings of the 33nd International Conference on Machine Learning, (ICML)*, pp. 3020–3029.

Kalofolias, Vassilis, Xavier Bresson, Michael Bronstein, and Pierre Vandergheynst (2014). "Matrix completion on graphs". In: *arXiv preprint arXiv:1408.1717*.

Kang, Hyunseung, Anru Zhang, T Tony Cai, and Dylan S Small (2016). "Instrumental variables estimation with some invalid instruments and its application to Mendelian randomization". In: *Journal of the American statistical Association* 111.513, pp. 132–144.

Kingma, Diederik and Jimmy Ba (2014). "ADAM: A method for stochastic optimization". In: *International Conference on Learning Representations (ICLR)*.

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907*.

Kling, Jeffrey R (2006). "Incarceration length, employment, and earnings". In: *American Economic Review* 96.3, pp. 863–876.

Kolesár, Michal, Raj Chetty, John Friedman, Edward Glaeser, and Guido W Imbens (2015). "Identification and inference with many invalid instruments". In: *Journal of Business & Economic Statistics* 33.4, pp. 474–484.

Kondor, Risi and Shubhendu Trivedi (Nov. 2018). "On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups". en. In: *35th International Conference on Machine Learning (ICML 2018)*. arXiv: 1802.03690.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix factorization techniques for recommender systems". In:

Kourentzes, Nikolaos, Devon K. Barrow, and Sven F. Crone (July 2014). "Neural network ensemble operators for time series forecasting". en. In: *Expert Systems with Applications* 41.9, pp. 4235–4244.

Kuang, Zhaobin, Frederic Sala, Nimit Sohoni, Sen Wu, Aldo Córdova-Palomera, Jared Dunnmon, James Priest, and Christopher Re (26–28 Aug 2020). "Ivy: Instrumental Variable Synthesis for Causal Inference". In: ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. Online: PMLR, pp. 398–410.

LeCun, Y., B. Boser, J.S. Denker, D Henderson, R.E. Howard, W. Hubbard, and L.E. Jackel (Dec. 1989). "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4, pp. 541–551. ISSN: 0899-7667.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (May 2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444. URL: http://dx.doi.org/10.1038/nature14539.

LeCun, Yann and Corinna Cortes (2010). *MNIST handwritten digit database*. http://yann.lecun.com/exdb/mnist/. URL: http://yann.lecun.com/exdb/mnist/.

Lee, Joonseok, Seungyeon Kim, Guy Lebanon, and Yoram Singer (2013). "Local low-rank matrix approximation." In: *Sanjoy Dasgupta and David McAllester, editors, Proceedings of the 30th International Conference on Machine Learning (ICML), volume 28 of Proceedings of Machine Learning Research*, pp. 82–90.

Li, Sheng, Jaya Kawale, and Yun Fu (2015). "Deep collaborative filtering via marginalized denoising auto-encoder". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, pp. 811–820.

Li, Zhuwen, Qifeng Chen, and Vladlen Koltun (2018). "Combinatorial optimization with graph convolutional networks and guided tree search". In: *Advances in Neural Information Processing Systems 31*. NIPS '18, pp. 539–548.

Lindauer, Marius, Holger H. Hoos, Frank Hutter, and Torsten Schaub (2015). "AutoFolio: Algorithm Configuration for Algorithm Selection". In: *Workshops at the 29th AAAI Conference on Artificial Intelligence*. AAAI '15. AAAI Press.

Loreggia, Andrea, Yuri Malitsky, Horst Samulowitz, and Vijay Saraswat (2016). "Deep Learning for Algorithm Portfolios". In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. AAAI '16. AAAI Press, pp. 1280–1286.

McKelvey, R.D. and T.R. Palfrey (1995). "Quantal Response Equilibria for Normal Form Games". In: *GEB* 10.1, pp. 6–38.

Milgrom, Paul and Ilya Segal (2014). "Deferred-acceptance Auctions and Radio Spectrum Reallocation". In: *Proceedings of the Fifteenth ACM Conference on Economics and Computation*. EC '14. Palo Alto, California, USA: ACM, pp. 185–186. ISBN: 978-1-4503-2565-3.

Mitchell, David, Bart Selman, and Hector Levesque (1992). "Hard and Easy Distributions of SAT problems". In: *Proceedings of the 10th AAAI Conference on Artificial Intelligence*. AAAI '92, pp. 459–465.

Mnih, Andriy and Ruslan R Salakhutdinov (2008). "Probabilistic matrix factorization". In: *Advances in neural information processing systems*.

Monti, Federico, Michael Bronstein, and Xavier Bresson (2017). "Geometric matrix completion with recurrent multi-graph neural networks". In: *Advances in Neural Information Processing Systems*.

Mu, Zongxu and Holger H. Hoos (2015). "On the empirical time complexity of random 3-SAT at the phase transition". In: *Proceedings of the 24th Interna-*

tional Joint Conference on Artificial Intelligence. IJCAI '15. Morgan Kaufmann, pp. 367–373.

Mueller-Smith, Michael (2015). "The criminal and labor market impacts of incarceration". In: *Unpublished Working Paper* 18. URL: https://sites.lsa.umich.edu/mgms/wp-content/uploads/sites/283/2015/09/incar.pdf.

Newey, Whitney K and James L Powell (2003). "Instrumental variable estimation of nonparametric models". In: *Econometrica* 71.5, pp. 1565–1578.

Nudelman, Eugene, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham (2004). "Understanding Random SAT: Beyond the Clauses-to-Variables Ratio". In: *Tenth International Conference on Principles and Practice of Constraint Programming*. CP '04. Springer Berlin Heidelberg, pp. 438–452.

Orbanz, Peter and Daniel M Roy (2015). "Bayesian models of graphs, arrays and other exchangeable random structures". In: *IEEE transactions on pattern analysis and machine intelligence*.

Parkes, David C. and Michael P. Wellman (2015). "Economic reasoning and artificial intelligence". In: *Science* 349.6245, pp. 267–272. ISSN: 0036-8075.

Parzen, Emanuel (Sept. 1962). "On Estimation of a Probability Density Function and Mode". en. In: *The Annals of Mathematical Statistics* 33.3, pp. 1065–1076.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8024–8035.

Pearl, Judea (2009). *Causality*. Cambridge university press.

Peters, Jonas, Dominik Janzing, and Bernhard Schölkopf (2017). *Elements of causal inference: foundations and learning algorithms*. The MIT Press.

Prates, Marcelo, Pedro HC Avelar, Henrique Lemos, Luis C Lamb, and Moshe Y Vardi (2019). "Learning to Solve NP-Complete Problems: A Graph Neural Network for Decision TSP". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 4731–4738.

Raedt, Luc De, Kristian Kersting, Sriraam Natarajan, and David Poole (2016). "Statistical relational artificial intelligence: Logic, probability, and computation". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10.2, pp. 1–189.

Rao, Nikhil, Hsiang-Fu Yu, Pradeep K. Ravikumar, and Inderjit S. Dhillon (2015). "Collaborative filtering with graph information: Consistency and scalable methods". In: *C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Informa- tion Processing Systems 28*, pp. 2107–2115.

Ravanbakhsh, Siamak, Jeff Schneider, and Barnabas Poczos (Aug. 2017). "Equivariance Through Parameter-Sharing". In: *Proceedings of the 34th International*

Conference on Machine Learning. Vol. 70. JMLR: WCP. Sydney, Australia, GB.

Reiersøl, O. (1945). "Confluence Analysis by Means of Instrumental Sets of Variables". In: *Arkiv för Matematik, Astronomi och Fysik* 32a.4, pp. 1–119.

Robbins, H. and S. Monro (1951). "A stochastic approximation method". In: *The Annals of Mathematical Statistics*, pp. 400–407.

Rosenbaum, P. R. and D. B. Rubin (1983). "Assessing Sensitivity to an Unobserved Binary Covariate in an Observational Study with Binary Outcome". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 45.2, pp. 212–218.

Russell, Stuart J. and Peter Norvig (2010). *Artificial intelligence: a modern approach*. en. 3rd ed. Prentice Hall series in artificial intelligence. Upper Saddle River: Prentice Hall.

Salakhutdinov, Ruslan, Andriy Mnih, and Geoffrey Hinton (2007). "Restricted Boltzmann machines for collaborative filtering". In: *Proceedings of the 24th international conference on Machine learning*. ACM, pp. 791–798.

Sandholm, Tuomas W. (1996). "A second order parameter for 3SAT". In: *Proceedings of the 13th AAAI Conference on Artificial Intelligence*. AAAI '96. AAAI Press, pp. 259–265.

Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini (2009). "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80.

Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61. Published online 2014; based on TR arXiv:1404.7828, pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

Sedhain, Suvash, Aditya Krishna Menon, Scott Sanner, and Lexing Xie (2015). "Autorec: Autoencoders meet collaborative filtering". In: *Proceedings of the 24th International Conference on World Wide Web*. ACM, pp. 111–112.

Sekiyama, Taro and Kohei Suenaga (2018). "Automated proof synthesis for propositional logic with deep neural networks". In: *arXiv preprint*. arXiv: 1805.11799.

Selsam, Daniel and Nikolaj Bjørner (2019). "Guiding High-Performance SAT Solvers with Unsat-Core Predictions". In: *Theory and Applications of Satisfiability Testing - SAT '19*. Lecture Notes in Computer Science. Springer.

Selsam, Daniel, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. (2019). "Learning a SAT Solver from Single-Bit Supervision". In: *Proceedings of the 7th International Conference on Learning Representations*. ICLR '19.

Shawe-Taylor, J. (1989). "Building symmetries into feedforward networks". In: *1989 First IEE International Conference on Artificial Neural Networks, (Conf. Publ. No. 313)*, pp. 158–162.

Shoham, Y. and K. Leyton-Brown (2008). *Multiagent Systems: Algorithmic, Game-theoretic, and Logical Foundations*. Cambridge University Press.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Singh, Rahul, Maneesh Sahani, and Arthur Gretton (2019). "Kernel Instrumental Variable Regression". In: *arXiv preprint arXiv:1906.00232*. URL: http://arxiv.org/abs/1906.00232.

Smith-Miles, Kate and Leo Lopes (2012). "Measuring Instance Difficulty for Combinatorial Optimization Problems". In: *Computers and Operations Research* 39.5, pp. 875–889.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research*.

Stahl, D. and P. Wilson (1994). "Experimental evidence on players' models of other players". In: *JEBO* 25.3, pp. 309–327.

Stock, James H. and Mark W. Watson (Sept. 2004). "Combination forecasts of output growth in a seven-country data set". en. In: *Journal of Forecasting* 23.6, pp. 405–430.

Tambe, Milind (2011). *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. 1st. New York, NY, USA: Cambridge University Press. ISBN: 1107096421, 9781107096424.

Tompson, Jonathan, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler (2015). "Efficient object localization using convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 648–656.

Varian, Hal R. (Dec. 2007). "Position auctions". In: *International Journal of Industrial Organization* 25.6, pp. 1163–1178.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008.

Venter, J. H. (Oct. 1967). "On Estimation of the Mode". en. In: *The Annals of Mathematical Statistics* 38.5, pp. 1446–1455.

Wang, Hao, Naiyan Wang, and Dit-Yan Yeung (2015). "Collaborative deep learning for recommender systems". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 1235–1244.

Wang, Yixin and David M Blei (2019). "The blessings of multiple causes". In: *Journal of the American Statistical Association* just-accepted, pp. 1–71.

Wood, Jeffrey and John Shawe-Taylor (1996). "Representation theory and invariant neural networks". In: *Discrete Applied Mathematics* 69.1, pp. 33–60. ISSN: 0166-218X.

Wright, James R and Kevin Leyton-Brown (2014). "Level-0 Meta-Models for Predicting Human Behavior in Games". In: *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, pp. 857–874.

— (2010). "Beyond Equilibrium: Predicting Human Behavior in Normal-Form Games." In: *AAAI*. Ed. by Maria Fox and David Poole. AAAI Press.

— (2012). "Behavioral Game-Theoretic Models: A Bayesian Framework For Parameter Analysis". In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2012)*. Vol. 2, pp. 921–928.

Wright, P. G. (1928). *The Tariff on Animal and Vegetable Oils*. Macmillan.

Xu, Lin, Holger H. Hoos, and Kevin Leyton-Brown (2007). "Hierarchical Hardness Models for SAT". In: *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*. CP '07. Springer-Verlag, pp. 696–711.

— (2012). "Predicting Satisfiability at the Phase Transition". In: *Proceedings of the 26th AAAI Conference on Artificial Intelligence*. AAAI '12. AAAI Press, pp. 584–590.

Xu, Lin, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown (2008). "SATzilla: Portfolio-based Algorithm Selection for SAT". In: *Journal of Artificial Intelligence Research* 32, pp. 565–606.

Yang, R., C. Kiekintveld, F. Ordonez, M. Tambe, and R. John (2013). "Improving Resource Allocation Strategies Against Human Adversaries in Security Games: An Extended Study". In: *Artificial Intelligence Journal (AIJ), 195:440-469*.

Yang, Zhilin, William W Cohen, and Ruslan Salakhutdinov (2016). "Revisiting semi-supervised learning with graph embeddings". In: *arXiv preprint arXiv:1603.08861*.

Zaheer, Manzil, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruskan R. Salakhutdinov, and Alexander J. Smola (2017). "Deep sets". In: *Advances in Neural Information Processing Systems*. NIPS '17, pp. 3391–3401.

Zheng, Yin, Bangsheng Tang, Wenkui Ding, and Hanning Zhou (2016). "A neural autoregressive approach to collaborative filtering". In: *Proceedings of the 33nd International Conference on Machine Learning*, pp. 764–773.

Zinkevich, Martin (2003). "Online convex programming and generalized infinitesimal gradient ascent". In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*.

Appendix A

Deep Networks for Exchangeable Arrays

A.1 Notation

- $X \in \mathbb{R}^{N_1 \times ... \times N_D}$, the data tensor
- $x \in \mathbb{R}^{\prod_i N_i}$, vectorized X, also denoted by vec(X).
- $[N]$: the sequence $\{n\}_{n=1,...,N} = (1, 2, ...N)$
- $[N_1 \times ... \times N_D]$, the sequence $\{(n_1, ..., n_D)\}_{n_1 \in [N_1], ..., n_D \in [N_D]}$ of D-dimensional tuples over $N_1 \times ... \times N_D$
- $\underline{n} = (n_i, n_{-i})$, an element of $N_1 \times ... \times N_D$
- $\mathcal{N} = \prod_i N_i$
- $\mathcal{S}^{(N)}$, symmetric group of *all* permutations of N objects
- $\mathcal{G}^{(N)} = \{g_i^{(N)}\}_i$, a permutation group of N objects
- $g_i^{(N)}$ or $G_i^{(N)}$, both can refer to the matrix form of the permutation $g_i^{(N)} \in \mathcal{G}^{(N)}$.
- $g_i^{(N)}(n)$ refers to the result of applying $g_i^{(N)}$ to n, for any $n \in [N]$.

- σ, an arbitrary, element-wise, strictly monotonic, nonlinear function such as sigmoid.

A.2 Proofs

Proposition 7. *Let $g^{(\mathcal{N})} \in \mathcal{S}^{(\mathcal{N})}$ be an "illegal" permutation of elements of the tensor X – that is $g^{(\mathcal{N})} \notin \mathcal{S}^{(N_1)} \times \mathcal{S}^{(N_2)} \times ... \times \mathcal{S}^{(N_D)}$. Then there exists a dimension $i \in [D]$ such that, for some $n_i, n_i', n_{-i}, n_{-i}'$:*

$$g^{(\mathcal{N})}((n_i, n_{-i})) = (n_i', n_{-i}), \quad \textit{but}$$
$$g^{(\mathcal{N})}((n_i, n_{-i}')) \neq (n_i', n_{-i}').$$

A.2.1 Proof of Proposition 3

Proof. Let $X \in \mathbb{R}^{N_1 \times ... \times N_D}$ be the data matrix. We prove the contrapositive by induction on D, the dimension of X. Suppose that, for all $i \in [D]$ and all $n_i, n_i', n_{-i}, n_{-i}'$, we have that $g^{(\mathcal{N})}((n_i, n_{-i})) = (n_i', n_{-i})$ implies $g^{(\mathcal{N})}((n_i, n_{-i}')) = (n_i', n_{-i}')$. This means that, for any $\underline{n} = (n_i, n_{-i}) = (n_1, ..., n_i, ..., n_D)$ the action $g^{(\mathcal{N})}$ takes on n_i is independent of the action $g^{(\mathcal{N})}$ takes on n_{-i}, the remaining dimensions. Thus we can write

$$g^{(\mathcal{N})}(\underline{n}) = g^{(N_i)}(n_i) \times g^{(\mathcal{N}/N_i)}(n_{-i})$$

Where it is understood that the order of the group product is maintained (this is a slight abuse of notation). If $D = 2$ (base case) we have $g^{(\mathcal{N})}((n_1, n_2)) = g^{(N_1)}(n_1) \times g^{(N_2)}(n_2)$. So $g^{(\mathcal{N})} \in \mathcal{S}^{(N_1)} \times \mathcal{S}^{(N_2)}$, and we are done. Otherwise, an inductive argument on $g^{(\mathcal{N}/N_i)}$ allows us to write $g^{(\mathcal{N})}(\underline{n}) = g^{(N_1)}(n_1) \times g^{(N_2)}(n_2) \times ... \times g^{(N_D)}(n_D)$. And so $g^{(\mathcal{N})} \in \mathcal{S}^{(N_1)} \times \mathcal{S}^{(N_2)} \times ... \times \mathcal{S}^{(N_D)}$, completing the proof. □

A.2.2 Proof of Proposition 4

Proof. First, observe that

$$g^{(\mathcal{N})}(\underline{n}) = \underline{n}' \iff (G^{(\mathcal{N})})_{\underline{n}', \underline{n}} = 1$$

Now, let $i \in [D]$ be such that, for some $n_i, n_i', n_{-i}, n_{-i}'$ we have

$$g^{(\mathcal{N})}((n_i, n_{-i})) = (n_i', n_{-i}), \quad \text{and}$$
$$g^{(\mathcal{N})}((n_i, n_{-i}')) \neq (n_i', n_{-i}').$$

Then by the observation above we have:

$$(G^{(\mathcal{N})})_{(n_i', n_{-i}),(n_i, n_{-i})} = 1, \quad \text{and}$$
$$(G^{(\mathcal{N})})_{(n_i', n_{-i}'),(n_i, n_{-i}')} \neq 1.$$

And so:

$$\begin{aligned}(G^{(\mathcal{N})}W)_{(n_i', n_{-i}'),(n_i, n_{-i})} &= (G^{(\mathcal{N})})_{(n_i', n_{-i}'),*}(W)_{*,(n_i, n_{-i})} \\ &= \sum_{\underline{k} \in [N_1 \times \ldots \times N_D]} (G^{(\mathcal{N})})_{(n_i', n_{-i}'),\underline{k}}(W)_{\underline{k},(n_i, n_{-i})} \\ &\neq W_{(n_i, n_{-i}'),(n_i, n_{-i})}\end{aligned}$$

The last line follows from the observation above and the fact that $G^{(\mathcal{N})}$ is a permutation matrix and so has only one 1 per row. Similarly,

$$\begin{aligned}(WG^{(\mathcal{N})})_{(n_i', n_{-i}'),(n_i, n_{-i})} &= (W)_{(n_i', n_{-i}'),*}(G^{(\mathcal{N})})_{*,(n_i, n_{-i})} \\ &= \sum_{\underline{k} \in [N_1 \times \ldots \times N_D]} (W)_{(n_i', n_{-i}'),\underline{k}}(G^{(\mathcal{N})})_{\underline{k},(n_i, n_{-i})} \\ &= (W)_{(n_i', n_{-i}'),(n_i', n_{-i})}\end{aligned}$$

Where again the last line follows from the above observation. Now, consider $W_{(n_i, n_{-i}'),(n_i, n_{-i})}$ and $W_{(n_i', n_{-i}'),(n_i', n_{-i})}$. Observe that (n_i, n_{-i}') differs from (n_i, n_{-i}) at exactly the same indices that (n_i', n_{-i}') differs from (n_i', n_{-i}). Let $\mathbb{S} \subseteq [D]$ be the set of indices at which n_{-i} differs from n_{-i}'. We therefore have

$$W_{(n_i, n_{-i}'),(n_i, n_{-i})} = W_{(n_i', n_{-i}'),(n_i', n_{-i})} = \theta_{\mathbb{S}},$$

Which completes the proof. □

A.2.3 Proof of Theorem 2

Proof. We will prove both the forward and backward direction:
($\Leftarrow$) Suppose W has the form given by (2.6). We must show the layer is only equivariant with respect to permutations in $\mathcal{S}^{(N_1)} \times ... \times \mathcal{S}^{(N_D)}$:

- **Equivariance**: Let $g^{(\mathcal{N})} \in \mathcal{S}^{(N_1)} \times ... \times \mathcal{S}^{(N_D)}$, and let $G^{(\mathcal{N})}$ be the corresponding permutation matrix. Then a simple extension of Theorem 2.1 in [Ravanbakhsh et al., 2017] implies $G^{(\mathcal{N})}WX = WG^{(\mathcal{N})}X$ for all X, and thus the layer is equivariant. Intuitively, if $g^{(\mathcal{N})} \in \mathcal{S}^{(N_1)} \times ... \times \mathcal{S}^{(N_D)}$ we can "decompose" $g^{(\mathcal{N})}(\underline{n})$ into D permutations $\mathcal{S}^{(N_1)}(n_1), ..., \mathcal{S}^{(N_D)}(n_D)$ which act independently on the D dimensions of X.

- **No equivariance wrt any other permutation**: Let $g^{(\mathcal{N})} \in \mathcal{S}^{(\mathcal{N})}$, with $g^{(\mathcal{N})} \notin \mathcal{S}^{(N_1)} \times \cdots \times \mathcal{S}^{(N_D)}$, and let $G^{(\mathcal{N})}$ be the corresponding permutation matrix. Using Propositions 3 and 4 we have:

  $$G^{(\mathcal{N})}W \neq WG^{(\mathcal{N})}$$

  So there exists an index at which these two matrices differ, call it $(\underline{n}, \underline{n}')$. Then if we take $\text{vec}(X)$ to be the vector of all 0's with a single 1 in position $\underline{n}'$, we will have:

  $$G^{(\mathcal{N})}W\,\text{vec}(X) \neq WG^{(\mathcal{N})}\,\text{vec}(X).$$

  And since σ is assumed to be strictly monotonic, we have:

  $$\sigma(G^{(\mathcal{N})}W\,\text{vec}(X)) \neq \sigma(WG^{(\mathcal{N})}\,\text{vec}(X)).$$

  And finally, since $G^{(\mathcal{N})}$ is a permutation matrix and σ is applied element-wise, we have:

  $$G^{(\mathcal{N})}\sigma(W\,\text{vec}(X)) \neq \sigma(WG^{(\mathcal{N})}\,\text{vec}(X)).$$

  Therefore, the layer $\sigma(W\,\text{vec}(X))$ is not a exchangeable tensor layer, and the proof is completed.

This proves the first direction.

($\Rightarrow$) We prove the contrapositive. Suppose $W_{\underline{n},\underline{n}'} \neq W_{\underline{m},\underline{m}'}$ for some $\underline{n},\underline{n}',\underline{m},\underline{m}' \in N_1 \times ... \times N_D$ with $\{i \ : \ n_i = n_i'\} = \{i \ : \ m_i = m_i'\}$. We want to show that the layer $Y = \text{vec}^{-1}(\sigma(W\,\text{vec}(X)))$ is not equivariant to some permutation $g^{(\mathcal{N})} \in \mathcal{S}^{(N_1)} \times ... \times \mathcal{S}^{(N_D)}$. We define this permutation as follows:

$$g^{(\mathcal{N})}(\nu) = \begin{cases} \underline{m} & \text{if} \quad \nu = \underline{n} \\ \underline{m}' & \text{if} \quad \nu = \underline{n}' \\ \underline{n} & \text{if} \quad \nu = \underline{m} \\ \underline{n}' & \text{if} \quad \nu = \underline{m}' \\ \nu & \text{otherwise} \end{cases}$$

That is, $g^{(\mathcal{N})}$ "swaps" $\underline{n}$ with $\underline{m}$ and $\underline{n}'$ with $\underline{m}'$. This is a valid permutation first since it acts element-wise, but also since $\{i \ : \ n_i = n_i'\} = \{i \ : \ m_i = m_i'\}$ implies that $\underline{n} = \underline{n}'$ iff $\underline{m} = \underline{m}'$ (and so $g^{(\mathcal{N})}$ is injective, and thus bijective). So if $G^{(\mathcal{N})}$ is the permutation matrix of $g^{(\mathcal{N})}$ then we have $(G^{(\mathcal{N})})_{(\underline{n}',\underline{n})} = (G^{(\mathcal{N})})_{(\underline{m}',\underline{m})} = 1$, and:

$$\begin{aligned} (G^{(\mathcal{N})}W)_{(\underline{m},\underline{n}')} &= \sum_{\underline{k} \in [N_1 \times ... \times N_D]} (G^{(\mathcal{N})})_{(\underline{m},\underline{k})} W_{(\underline{k},\underline{n}')} \\ &= W_{(\underline{n},\underline{n}')} \\ &\neq W_{(\underline{m},\underline{m}')} \\ &= \sum_{\underline{k} \in [N_1 \times ... \times N_D]} W_{(\underline{m},\underline{k})} (G^{(\mathcal{N})})_{(\underline{k},\underline{n}')} \\ &= (WG^{(\mathcal{N})})_{(\underline{m},\underline{n}')} \end{aligned}$$

And so $G^{(\mathcal{N})}W \neq WG^{(\mathcal{N})}$ and by an argument identical to the above, with appropriate choice of X, this layer does not satisfy the requirements of an exchangeable tensor layer. This completes the proof. $\square$

A.2.4 Proof of Theorem 1

A simple reparameterization allows us to write the matrix W of (2.6) in the form of (2.3). Thus Theorem 1 is just the special case of Theorem 2 where $D = 2$ and the proof follows from that.

A.3 Details of Architecture and Training

Self-Supervised Model. Details of architecture and training: we train a simple feed-forward network with 9 exchangeable matrix layers using a leaky ReLU activation function. Each hidden layer has 256 channels and we apply a channel-wise dropout with probability 0.5 after the first to seventh layers. We found this channel-wise dropout to be crucial to achieving good performance. Before the input layer we mask out a proportion of the ratings be setting their values to 0 uniformly at random with probability 0.15. We convert the input ratings to one-hot vectors and interpret the model output as a probability distribution over potential rating levels. We tuned hyper-parameters by training on 75% of the data, evaluating on a 5% validation set. We test this model using the canonical u1.base/u1.test training/test split, which reserves 20% of the ratings for testing. For the MovieLens-1M dataset, we use the same architecture as for ML-100k and trained on 85% of the data, validating on 5%, and reserving 10% for testing. The limited size of GPU memory becomes an issue for this larger dataset, so we had to employ conditional sampling for training. At validation time we used full batch predictions using the CPU in order to avoid memory issues.

Factorized Exchangeable Autoencoder Model. Details of architecture and training: we use three exchangeable matrix layers for the encoder. The first two have 220 channels, and the third layer maps the input to 100 features for each entry, with no activation function applied. This is followed by mean pooling along both dimensions of the input. Thus, each user and movie is encoded into a length 100 vector of real-valued latent features. The decoder uses five similar exchangeable matrix layers, with the final layer having five channels. We apply a channel-wise dropout with probability 0.5 after the third and fourth layers, which we again found to be crucial for good performance. We convert the input ratings to one-hot vectors and optimize using cross-entropy loss.

Appendix B

Deep Learning for Predicting Human Strategic Behaviour

B.1 Representing Behavioural Features

The architecture presented in Section 3.3 is sufficiently flexible to approximate most of the known behavioural functions that appear in the behavioural game theory literature. For completeness, we include explicit constructions here that how these features can be approximated.

Proposition 8. *An network with two exchangeable layers (of the form presented in Section 3.3), one hidden unit per layer, max pooling units at every layer and rectified linear unit activation functions can approximate* min max regret*,* min min unfairness*,* max min payoff*,* max max payoff*, and* max max efficiency*.*

Proof. By expanding the sums from the definition of the network, we see the first hidden layer has the following functional form:

$$\mathbf{H}^{(1,1)} = \text{relu}(w_{1,r}\mathbf{U}^{(r)} + w_{1,c}\mathbf{U}^{(c)} + w_{1,rc}\mathbf{U}_c^{(r)} + w_{1,rr}\mathbf{U}_r^{(r)} + w_{1,cc}\mathbf{U}_c^{(c)} + w_{1,cr}\mathbf{U}_r^{(c)} + b_{1,1}).$$

where $\mathbf{U}^{(r)}$ is the row player's payoff matrix and $\mathbf{U}_c^{(r)}$ is the row player's payoff matrix aggregated using the column-preserving pooling unit where we use the max function to perform the aggregation. Similarly, the second hidden layer can be

written as,

$$\mathbf{H}^{(2,1)} = \text{relu}(w_{2,1}\mathbf{H}^{(1,1)} + w_{2,c}\mathbf{H}_c^{1,1} + w_{2,r}\mathbf{H}_r^{(1,1)} + b_{2,1}).$$

We denote $\mathbf{H}^{(1,1)}$ as the output of the first hidden layer and $\mathbf{H}_c^{(1,1)}$ and $\mathbf{H}_r^{(1,1)}$ are its respective pooled outputs.

Game theoretic features can be interpreted as outputting a strategy (a distribution over a player's actions) given a description over the game. We express features in a style similar to [Wright and Leyton-Brown, 2014] by outputting a vector $\mathbf{f}$ such that $f_i \approx 0$ for all $f_i \in \mathbf{f}$ if action i does not correspond to the target feature, and $f_i \approx \frac{1}{l}$ where l is the number of actions that correspond to the target feature (with $l = 1$ if the actions uniquely satisfies the feature). Because features are all constructed from a sparse subset of the parameters, we limit notational complexity by letting $w_{i,j} = 0$ and $b_{i,j} = 0$ for all $i, j \in 1, 2, r, c$ unless stated otherwise.

Max Max Payoff

Required: $f^{\text{maxmax}}(i) = \begin{cases} \frac{1}{l} & \text{if } i \in \arg\max_{i\in\{1,\dots,m\}} \max_{j\in\{1,\dots,n\}} u_{i,j} \quad u_{i,j} \in \mathbf{U}^{(r)} \\ 0 & \text{otherwise} \end{cases}$

Let $w_{1,r} = 1, w_{2,r} = c$ where c is some large positive constant and $b_{1,1} = b$ where is some scalar $b \geq \min_{i,j} \mathbf{U}_{i,j}^{(r)}$ and all other parameters $w_{i,j}, b_{i,j} = 0$. Then $\mathbf{H}^{(1,1)}$ reduces to,

$$\mathbf{H}^{(1,1)} = \text{relu}(\mathbf{U}^{(r)} + b) = \mathbf{U}^{(r)} + b \quad \text{since } \mathbf{U}^{(r)} + b \geq 0 \text{ by definition of b}$$

$$\mathbf{H}^{(2,1)} = \text{relu}(c\mathbf{H}_r^{(1,1)}) \Rightarrow h_{j,k} = c(\max_k u_{j,k} + b) \quad \forall u_{j,k} \in \mathbf{U}^{(r)}, h_{j,k} \in \mathbf{H}^{(2,1)}$$

That is, all the elements in each row of $\mathbf{H}^{(2,1)}$ equal an positive affine transformation of the maximum element from the corresponding row in $\mathbf{U}^{(r)}$.

$$\mathbf{f}_i^{(1)} = \text{softmax}(\sum_{k=1}^{n} h_{j,k}) = \text{softmax}\left(\sum_{k=1}^{n} c(\max_k u_{j,k} + b)\right) = \text{softmax}\left(nc(\max_k u_{j,k} + b)\right)$$

Therefore, as $c \to \infty$, $\mathbf{f}_i^{(1)} \to f^{maxmax}(i)$ as required.

Max Min Payoff

Required: $f^{\text{maxmin}}(i) = \begin{cases} \frac{1}{l} & \text{if } i \in \arg\max_{i\in\{1,\dots,m\}} \min_{j\in\{1,\dots,n\}} u_{i,j} \quad u_{i,j} \in \mathbf{U}^{(r)} \\ 0 & \text{otherwise} \end{cases}$

Max Min Payoff is derived similarly to Max Max except with $w_{1,r} = -1$, and $b_{1,1} = b$ where $b \geq \max_{i,j} \mathbf{U}^{(r)}_{i,j}$; we keep $w_{2,r} = c$ as some large positive constant.

Then $\mathbf{H}^{(1,1)}$ reduces to,

$$\mathbf{H}^{(1,1)} = \text{relu}(-\mathbf{U}^{(r)} + b) = -\mathbf{U}^{(r)} + b \quad \text{since } -\mathbf{U}^{(r)} + b \geq 0 \text{ by definition of b}$$

$$\mathbf{H}^{(2,1)} = \text{relu}(c\mathbf{H}^{(1,1)}_r) \Rightarrow h_{j,k} = c(\max_k(-u_{j,k} + b)) = c(\min_k u_{j,k} + b) \quad \forall u_{j,k} \in \mathbf{U}^{(r)}, h_{j,k} \in \mathbf{H}^{(2,1)}$$

Since $\max_i -x_i + b = \min_i x_i + b$. Thus,

$$\mathbf{f}^{(1)}_i = \text{softmax}(\sum_{k=1}^{n} h_{j,k}) = \text{softmax}\left(nc(\min_k u_{j,k} + b)\right)$$

Therefore, as $c \to \infty$, $\mathbf{f}^{(1)}_i \to f^{maxmin}(i)$ as required.

Max Max Efficiency

Required: $f^{\text{max max efficiency}}(i) = \begin{cases} \frac{1}{l} & \text{if } i \in \arg\max_{i\in\{1,\dots,m\}} \max_{j\in\{1,\dots,n\}} u^{(c)}_{i,j} + u^{(r)}_{i,j} \\ 0 & \text{otherwise} \end{cases}$

Max Max Efficiency follow from the derivation of Max Max except with $w_{1,r} = 1, w_{1,c} = 1, w_{2,r} = c$ and $b_{1,1} = b$ where $b \geq \min_{i,j}(\mathbf{U}^{(r)}_{i,j} + \mathbf{U}^{(r)}_{i,j})$.

Following the same steps we get,

$$\begin{aligned} \mathbf{f}^{(1)}_i &= \text{softmax}(\sum_{k=1}^{n} h_{j,k}) = \text{softmax}\left(\sum_{k=1}^{n} c(\max_k (u^{(r)} + u^{(c)})_{j,k} + b)\right) \\ &= \text{softmax}\left(nc(\max_k (u^{(r)} + u^{(c)})_{j,k} + b)\right) \\ &= f^{\text{max max efficiency}}(i) \qquad \text{as } c \to \infty \end{aligned}$$

Minimax Regret

Regret is defined as $r(i,j) = \max_i u_{i,j} - u_{i,j} \qquad u_{i,j} \in \mathbf{U}^{(r)}$

Required: $f^{\text{minimax regret}}(i) = \begin{cases} \frac{1}{l} & \text{if } i \in \arg\min_{i\in\{1,\dots,m\}} \max_{j\in\{1,\dots,n\}} r(i,j) \\ 0 & \text{otherwise} \end{cases}$

Let $w_{1,rc} = 1$, $w_{1,r} = -1$, and $b_{1,1} = 0$; we keep $w_{2,r} = c$ as some large positive constant.

Then $\mathbf{H}^{(1,1)}$ reduces to,

$$\mathbf{H}^{(1,1)} = \text{relu}(\mathbf{U}_c^{(r)} - \mathbf{U}^{(r)}) = \mathbf{U}_c^{(r)} - \mathbf{U}^{(r)} \quad \text{since } \mathbf{U}_c^{(r)} \geq \mathbf{U}^{(r)} \text{ by definition of } \mathbf{U}_c^{(r)}$$

$$\mathbf{H}^{(2,1)} = \text{relu}(c\mathbf{H}_r^{(1,1)}) \Rightarrow h_{j,k} = c(\max_k(\max_j u_{j,k} - u_{j,k})) \quad \forall u_{j,k} \in \mathbf{U}^{(r)}, h_{j,k} \in \mathbf{H}^{(2,1)}$$

Thus,

$$\mathbf{f}_i^{(1)} = \text{softmax}(\sum_{k=1}^{n} h_{j,k}) = \text{softmax}\left(nc(\max_k(\max_j u_{j,k} - u_{j,k}))\right)$$

Therefore, as $c \to \infty$, $\mathbf{f}_i^{(1)} \to f^{\text{minimax regret}}(i)$ as required.

Min Min Unfairness

Required: $f^{\text{min min unfairness}}(i) = \begin{cases} \frac{1}{l} & \text{if } i \in \arg\max_{i\in\{1,\dots,m\}} \min_{j\in\{1,\dots,n\}} |u_{i,j}^{(r)} - u_{i,j}^{(c)}| \\ 0 & \text{otherwise} \end{cases}$

To represent Min Min Unfairness, we add an additional hidden unit in the first layer. Let $\mathbf{H}^{(1,2)}$ be defined in the same manner as $\mathbf{H}^{(1,1)}$.

For $\mathbf{H}^{(1,1)}$, we let $w_{1,r}^1 = 1, w_{1,c}^1 = -1$ and $b_{1,1} = 0$ such that,

$$\mathbf{H}^{(1,1)} = \text{relu}(\mathbf{U}^{(r)} - \mathbf{U}^{(c)}) = \max(\mathbf{U}^{(r)} - \mathbf{U}^{(c)}, 0) \quad \text{where the max is applied element-wise}$$

For $\mathbf{H}^{(1,2)}$, we let $w_{1,r}^1 = -1, w_{1,c}^1 = -$ and $b_{1,1} = 0$ such that,

$$\mathbf{H}^{(1,2)} = \text{relu}(\mathbf{U}^{(c)} - \mathbf{U}^{(r)}) = \max(\mathbf{U}^{(c)} - \mathbf{U}^{(r)}, 0)$$

Now, notice that if $w_{2,1} = 1$ and $w_{2,2} = 1$,

$$\mathbf{H}^{(2,1)} = \mathbf{H}^{(1,1)} + \mathbf{H}^{(1,2)} = \max(\mathbf{U}^{(r)} - \mathbf{U}^{(c)}, 0) + \max(\mathbf{U}^{(c)} - \mathbf{U}^{(r)}, 0) = |\mathbf{U}^{(r)} - \mathbf{U}^{(c)}|$$

Which gives us a measure of "unfairness" as the absolute difference between the two payoffs.

We can therefore simulate $f^{\text{min min unfairness}}(i)$ by letting $w_{2,1} = -1$ and $w_{2,2} = -1$, and using the output of $\mathbf{H}_r^{(2,1)}$ (which gives us min unfairness), then constructing $\mathbf{f}_i$ by letting $c \to \infty$.

□

B.2 Multi-layer Perceptron Performance

Figure B.1 compares the performance of our architecture with that of a regular multilayer perceptron, with and without data augmentation, and the previous state-of-the-art model on this dataset. It shows that the feed-forward network dramatically overfitted the data without data augmentation. Data augmentation improved test set performance, but it was still unable to match state of the art performance. A three layer instantiation of our model (two layers of 50 hidden units and a single AR layer) matched the previous state of the art but failed to improve upon it. We suspect that this may be because the subset of the data that contains only 3×3 games is too small to take advantage of the flexibility of our model.

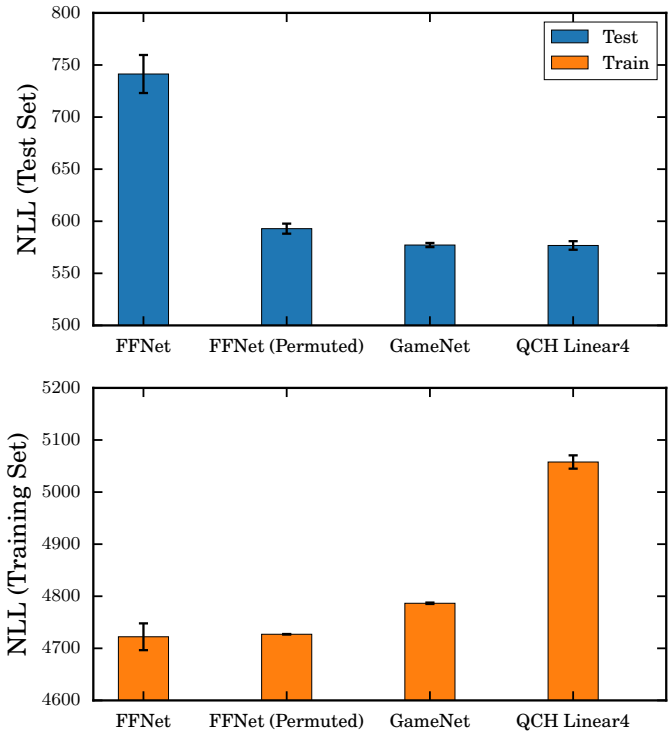

Figure B.1: Performance comparison on 3×3 games of a feed-forward neural network (FFNet), a feed-forward neural network with data augmentation at every epoch (FFNet (Permuted)), our architecture fit with the same hyper parameters as used for our best performing model in the main results (GameNet), and Quantal Cognitive Hierarchy with four hand-crafted features (QCH Linear4).

Appendix C

Valid Causal Inference with (Some) Invalid Instruments Appendix

C.1 Proof of Theorem 5

Theorem 1. *Fix a test point (t, x) and let $\hat{\beta}_1, \ldots, \hat{\beta}_k$ be estimators of the causal effect of t at x corresponding to k (possibly invalid) instruments. E.g., $\hat{\beta}_j = \hat{f}_j(t, x)$. Denote the true effect as $\beta = E[y|do(t), x]$. Suppose that*

1. *(consistent estimators) $\hat{\beta}_j \to \beta_j$ almost surely for each instrument. In particular, $\beta_j = \beta$ whenever the jth instrument is valid.*

2. *(modal validity) At least v of the instruments are valid, and no more than $v - 1$ of the invalid instruments agree on an effect. That is, v of the instruments yield the same estimand if and only if all of those instruments are valid.*

Let $[\hat{l}, \hat{u}]$ be the smallest interval containing v of the instruments and let $\hat{\mathcal{I}}_{\text{mode}} = \{i : \hat{l} \leq \hat{\beta}_i \leq \hat{u}\}$. Then,

$$\sum_{i \in \hat{\mathcal{I}}_{\text{mode}}} \hat{w}_i \hat{\beta}_i \to \beta$$

almost surely, where $\hat{w}_i, w_i$ *are any non-negative set of weights such that each* $\hat{w}_i \to w_i$ *a.s. and* $\sum_{i \in \hat{\mathcal{I}}_{\text{mode}}} w_i = 1$. *Further if the individual estimators are also asymptotically jointly normal,*

$$\sqrt{n}[\hat{\beta}_1, \ldots, \hat{\beta}_k]^T \to N([\beta_1, \ldots, \beta_k]^T, \Sigma)$$

with some covariance matrix $\Sigma \in \mathbb{R}^{k \times k}$. *Then it also holds that the modal estimator is asymptotically normal:*

$$\sqrt{n} \sum_{i \in \hat{\mathcal{I}}_{\text{mode}}} \hat{w}_i \hat{\beta}_i \to N(\beta, w^T \Sigma_{\mathcal{I}_{mode}} w).$$

where $\Sigma_{\mathcal{I}_{mode}} \in \mathbb{R}^{V \times V}$ *denotes the covariance of the selected instruments.*

Proof. First we argue that $\hat{\mathcal{I}}_{\text{mode}}$ converges to a set that contains only valid instruments. All valid instruments converge to a common value β. The distance between any two valid instruments is at most twice the distance between β and the furthest valid instrument. Since at least v of the instruments are valid, this means that there is an interval (containing the mode) with distance going to 0 that contains v of the instruments. Eventually this must be the smallest interval containing v of the instruments, because the limiting β_j of the invalid instruments are spaced out by assumption.

The result follows by the the continuous mapping theorem. □

C.2 Proof of Theorem 6

Theorem 2. *For some test point* (t, x), *let* $\hat{\beta}_1, \ldots, \hat{\beta}_k$ *be* k *estimates of the causal effect of* t *at* x. *Assume,*

*[**Bounded estimates**] Each estimate is bounded by some constants,* $[a_i, b_i]$

*[**Convergent estimators**] Each estimator converges in mean squared error at a rate* n^{-r} *(where* $r = \frac{1}{2}$ *if the estimator achieves the parametric rate), and hence each estimator has finite variance,* $Var(\hat{\beta}_i) = \frac{\sigma_i}{n^{2r}}$ *for some* σ_i.

Then, there exists some constant, C, *such that* $E[(\text{ModeIV}(\mathcal{Z}) - \beta)^2] - E[(\frac{1}{v} \sum_{i \in \mathcal{V}} \hat{\beta}_i - \beta)^2] \leq 9kC \frac{\sigma}{n^r}$.

Proof. Fix some test point (t, x) and without loss of generality, assume that the true effect, $E[y|\text{do}(t), x] = 0$. Because ModeIV optimizes for the closest V points, we can upper bound its worst case performance with an adversarial distribution that deterministically places all invalid instruments on the same point y such that $\hat{\beta}_i = y$ for all i in $\mathcal{I}$; assume $y > 0$ (again wlog, the argument is symmetric).

Consider the case where there are v valid instruments and $v-1$ invalid candidates. Let $d(\mathcal{V}) = \max_{i,j\in\mathcal{V}} |\hat{\beta}_i - \hat{\beta}_j|$ denote the largest distance between the any two valid estimates, and $d^* = \max_{i,j\in\hat{\mathcal{I}}_{\text{mode}}} |\hat{\beta}_i - \hat{\beta}_j|$ denote the largest distance among the set of the v closest candidates (regardless of whether or not they are valid). If there is some candidate $i \in \mathcal{V}$ such that $d^* = \min_{i\in\mathcal{V}} |\hat{\beta}_i - y| < d(\mathcal{V})$, then ModeIV returns $\frac{v-1}{v}y + \frac{1}{v}\hat{\beta}_i$; otherwise $d(\mathcal{V}) = d^*$ and ModeIV matches the oracle performance.

Now, consider the interval $C(y) = [-\frac{y}{3}, \frac{y}{3}]$; if all the valid instruments fall within $C(y)$, then $d(\mathcal{V}) = d^* \leq \frac{2y}{3}$ by construction. So,

$$E\left[(\text{ModeIV}(\mathcal{Z}))^2\right] - E\left[\left(\frac{1}{v}\sum_{i\in\mathcal{V}}\hat{\beta}_i\right)^2\right] \leq \Big(1 - P(\hat{\beta}_i \in C(y) \text{ for all } i \in \mathcal{V})\Big)\left(\frac{v-1}{v}y + \frac{1}{v}\hat{\beta}_i\right)^2$$
$$\leq \Big(1 - P(\hat{\beta}_i \in C(y) \text{ for all } i \in \mathcal{V})\Big)y^2 \qquad \text{(C.1)}$$

Since each $\hat{\beta}_i$ is bounded $\in [a_i, b_i]$, we know they are $\tilde{\sigma}_i$-sub-Gaussian random variables for some $\tilde{\sigma}_i \leq \frac{b_i - a_i}{4}$. Let $\tilde{\sigma} = \max_{i\in\mathcal{V}} \tilde{\sigma}_i$ and note that there exists some constant C, such that for all n and i, $\tilde{\sigma} \leq C\sigma_i n^{-r}$. Now,

$$P(\hat{\beta}_i \in C(y) \text{ for all } i \in \mathcal{V}) = \prod_{i=1}^{k} P\Big(\hat{\beta}_i \in C(y)\Big) = \prod_{i=1}^{k}\left(1 - P\left(|\hat{\beta}_i| > \frac{y}{3}\right)\right) \geq \prod_{i=1}^{k}\left(1 - \exp\left(-\frac{y^2}{18\tilde{\sigma}_i}\right)\right)$$
$$\geq \left(1 - \exp\left(-\frac{y^2}{18\tilde{\sigma}}\right)\right)^k$$
$$\geq 1 - k\exp\left(-\frac{y^2}{18\tilde{\sigma}}\right)$$

where the first inequality applies a tail bound on sub-Gaussian random variables and last inequality uses the fact that $(1+x)^k \geq 1 + kx$ for $x > -1$ and $k \geq 1$.

By substituting back into equation (C.1), we see that,

$$E\left[(\text{ModeIV}(\mathcal{Z}))^2\right] - E\left[\left(\frac{1}{v}\sum_{i\in\mathcal{V}}\hat{\beta}_i\right)^2\right] \leq y^2 k \exp\left(-\frac{y^2}{18\tilde{\sigma}}\right)$$

so we can get worst case performance by optimizing over, y. Taking partial deviates with respect to y and setting it to zero, we get,

$$\left[2y^* k - \frac{k(y^*)^3}{9\tilde{\sigma}}\right]\exp\left(-\frac{(y^*)^2}{18\tilde{\sigma}}\right) = 0 \quad \rightarrow \quad y^* = 3\sqrt{2\tilde{\sigma}}$$

is the only local maximum such that $y > 0$. And hence, the worst case mean square error is bounded by,

$$E[(\text{ModeIV}(\mathcal{Z}))^2] - E\left[\left(\frac{1}{v}\sum_{i\in\mathcal{V}}\hat{\beta}_i\right)^2\right] \leq \frac{18}{e}k\tilde{\sigma} \leq 9k\tilde{\sigma} \leq 9kC\frac{\sigma}{n^r}$$

□

C.3 Relaxing Independence of Instrumental Variables

ModeIV is simplest in the context of independent candidate instruments: this setting is shown in Figure C.1 (*left*) where we have k candidates, $\{z_i : i \in 1,\ldots,k\}$, some of which are valid (shown in blue), and some of which are invalid (e.g. z_k shown in pink has a direct effect on the response). This independent candidates setting is most common where the instruments are explicitly randomized: e.g. in judge fixed effects where the selection of judges is random.

A more complex setting is shown in Figure C.1 (*right*). Here, the candidates share a common cause, u. In this scenario, if u is not observed, each of the previously valid instruments (e.g. z_1, z_2 and z_{k-1} in the figure) are no longer valid because they fail the unconfounded instrument assumption via the backdoor path $z_1 \leftarrow u \rightarrow z_k \rightarrow y$. However, if we condition on all the candidates that have a direct effect on y and treat them as observed confounders, we block this path which allows for valid inference. Of course we do not know which of the candidates have a direct effect, so when

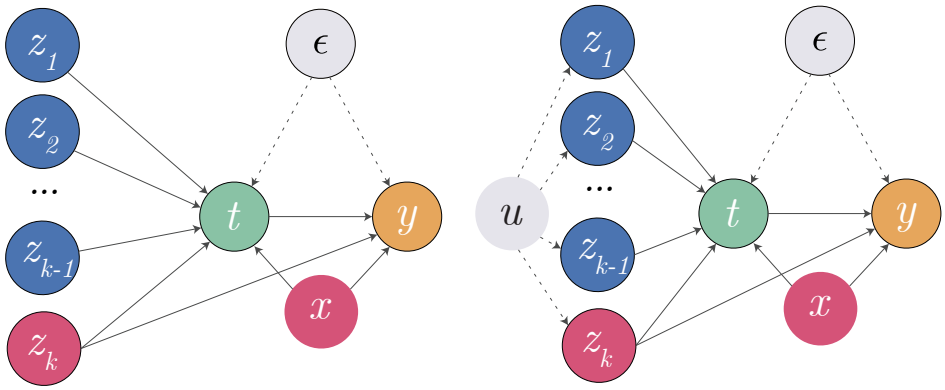

Figure C.1: We primarily focus on the setting on the left where each candidate, z_i, is independent. It is also possible to apply the method in the setting on the right where the candidates share a common cause, more care is needed. See the discussion in section C.3.

building an ensemble, for each candidate z_i, we treat all $z_{j\neq i}$ as observed confounds to block these potential backdoor paths. This addresses the issue as long as there is not some z_{k+1} which is not part of our candidate set, but nevertheless opens up a backdoor path $z_1 \leftarrow u \rightarrow z_{k+1} \rightarrow y$.

If u is observed, we can simply control for it. This suggests a natural alternative approach would be to try to estimate u and control for its effect, using an approach analogous to Wang and Blei [2019]. We plan to investigate this in future work.

C.4 Selecting Instruments?

ModeIV constitutes a consistent method for making unbiased predictions but, somewhat counter-intuitively, it does not directly offer a way of inferring the set of valid instruments. For example, one might imagine identifying the set of candidates that most often form part of the modal interval $\hat{\mathcal{I}}_{\text{mode}}$. The problem is that while candidates that fall within the modal interval $\hat{\mathcal{I}}_{\text{mode}}$ tend to be close to the mode, the interval can include invalid instruments that yielded an effect close to the mode by chance. Since these invalid estimates are close to the truth, they do not hurt the

	50 / 100 valid	60 / 100 valid	70 / 100 valid	80 / 100 valid	90 / 100 valid	100 / 100 valid
DeepIV (opt)	0.036 ± (0.0048)	0.042 ± (0.003)	0.0371 ± (0.0032)	0.0364 ± (0.003)	0.0326 ± (0.0021)	0.0278 ± (0.0021)
MODE-IV 20	0.0525 ± (0.0062)	0.0483 ± (0.0049)	0.0478 ± (0.0049)	0.0473 ± (0.0048)	0.0466 ± (0.0047)	0.04 ± (0.0038)
MODE-IV 30	0.0525 ± (0.0062)	0.0483 ± (0.0049)	0.0478 ± (0.0049)	0.0473 ± (0.0048)	0.0467 ± (0.0047)	0.0399 ± (0.0038)
MODE-IV 40	0.0524 ± (0.0062)	0.0483 ± (0.0049)	0.0478 ± (0.0049)	0.0473 ± (0.0048)	0.0468 ± (0.0047)	0.0399 ± (0.0038)
MODE-IV 50	0.0525 ± (0.0062)	0.0483 ± (0.0049)	0.0479 ± (0.0049)	0.0474 ± (0.0048)	0.0466 ± (0.0047)	0.0398 ± (0.0038)
Mean	0.0529 ± (0.0059)	0.0498 ± (0.005)	0.0498 ± (0.0052)	0.0461 ± (0.0047)	0.0484 ± (0.0048)	0.0403 ± (0.004)
Deepiv (all)	0.1637 ± (0.011)	0.1744 ± (0.0075)	0.2078 ± (0.0069)	0.185 ± (0.0087)	0.1387 ± (0.0081)	0.0297 ± (0.0018)

Table C.1: Average absolute bias in estimation of the conditional average treatment effect. The ensemble methods tended to have slightly larger bias than the optimal model, but far less than the naive approach which uses all instruments. The mean aggregation function performs relatively well on this task, but this approach comes with no guarantees, so it degrades in settings with more bias.

estimate. We can see this in Figure 6.1 where invalid instruments form part of the modal interval in the region $t \in [-3.5, -2]$, without introducing bias.

C.5 Additional Experimental Details

Network architectures and experimental setup All experiments used the same neural network architectures to build up hidden representations for both the treatment and response networks used in DeepIV, and differed only in their final layers. Given the number of experiments that needed to be run, hyper-parameter tuning would have been too expensive, so the hyper-parameters were simply those used in the original DeepIV paper. In particular, we used three hidden layers with 128, 64, and 32 units respectively and ReLU activation functions. In the Mendelian randomization experiments, we used The treatment networks all used mixture density networks with 10 mixture components and the response networks were trained using the two sample unbiased gradient loss [see Hartford et al., 2017, equation 10]. We used our own PyTorch [Paszke et al., 2019] re-implementation of DeepIV to run the experiments.

For all experiments, 10% of the original training set was kept aside as a validation set. Both the demand and Mendelian randomization simulations had 90 000 training examples, 10 000 validation examples and 50 000 test examples. All

mean squared error numbers reported in the paper are calculated with respect to the true y (with no confounding noise added) on a uniform grid of 50 000 treatment points between the 2.5^{th} and 97.5^{th} percentiles of the training distribution. As a result, we are measuring mean squared error relative to the target y on the uniform grid of treatment values corresponds to $E[y|\text{do}(t'), x, z]$ for each t' on the grid. 95% confidence intervals around mean performance were computed using Student's t distribution.

The networks were trained on a large shared compute cluster that has around 100 000 CPU cores. Because each individual network was relatively quick to train (less than 10 minutes on a CPU), we used CPUs to train the networks. This allowed us to fit the large number of networks needed for the experiments. Each experiment was run across 30 different random seeds, each of which required 10 (demand simulation) and 102 (Mendialian randomization) network fits. In total, across all experimental setups, random seeds, ensembles, etc. approximately 100 000 networks were fit to run all the experiments.

Biased demand simulation The biased demand simulation code was modified from the public DeepIV implementation. In the original DeepIV implementation, both the treatment and response are transformed to make them approximately mean zero and standard deviation 1; we left these constants unchanged ($t_{\text{std}} = 3.7, t_\mu = 17.779, y_{\text{std}} = 158, y_\mu = -292.1$). The observed features include a time feature, $x_0 \sim \text{unif}(0, 10)$, and $x \sim \text{Categorical}(\frac{1}{7}, \ldots, \frac{1}{7})$, a one-hot encoding of 7 different equally likely "customer types"; each type modifies the treatment via the coefficient $\beta^{(x)} = [1, 2, \ldots, 7]^T$. These values are unchanged from the original Hartford et al. data generating process. We introduce multiple instruments, $z_{1:k}$, whose effect on the treatment, t, and response, y, is via two different linear maps $\beta^{(zt)} \in \mathbb{R}^k$ and $\beta^{(zy)} \in \mathbb{R}^k$; each of the coefficients in these vectors are sampled independently so $\beta_i^{(z*)} \sim \text{unif}(0.5, 1.5)$, with the exception of the valid instruments where $\beta_i^{(zy)} = 0$ for all $i \in \mathcal{V}$. The γ parameter scales the amount of bias introduced via exclusion violations; note that because the sin varies between $[-1, 1]$, we scale up this bias by a factor of 60 so that the effect it introduces is of the same order of magnitude as the variation in the original Hartford et al. data generating process (where std. dev$(y) \approx y_{\text{std}} = 158$). The full data generating

process is as follows,

$$z_{1:k}, \nu \sim \mathcal{N}(0,1) \quad x_0 \sim \text{unif}(0,10) \quad e \sim \mathcal{N}(\rho\nu, 1-\rho^2), \quad x \sim \text{Categorical}(\frac{1}{7},\ldots,\frac{1}{7})$$

$$t' = 25 + (z^T\beta^{(zt)} + 3)\psi(x_0) + \nu$$

$$y' = 100 + 10x^T\beta^{(x)}\psi(x_0) + \underbrace{(x^T\beta^{(x)}\psi(x_0) - 2)}_{\text{Treatment effect}}t' + \underbrace{\gamma 60 \sin(z^T\beta^{(zy)})}_{\text{Exclusion violation}} + e$$

$$t = (t' - t_{\text{std}})/t_\mu \qquad y = (y' - y_{\text{std}})/y_\mu$$

Mendelian randomization simulation This data generating process closely follows that of Simulation 1 of Hartwig et al. [2017], but was modified to include heterogeneous treatment effects. This description is an abridged version of that given in Hartwig et al.; we refer the reader to Hartwig et al. [2017] for more detail on the choice of parameters, etc.. The instruments are the genetic variables, z_i, which were generated by sampling from a Binomial (2, p_i) distribution, with p_i drawn from a Uniform(0.1 ,0.9) distribution, to mimic bi-allelic SNPs in Hardy-Weinberg equilibrium. The parameters that modulate the genetic variable effect on the treatment are given by, $\alpha_i = \frac{\sqrt{0.1}}{\sigma_{zx}}\nu_i$, where $\nu_i \sim \text{unif}(0.01, 0.2)$ and $\sigma_{zx} = \text{std. dev}(\sqrt{0.1}\sum_i \nu_i z_i)$. Similarly, the exclusion violation parameters, $\delta_i = \frac{|\mathcal{I}|\sqrt{0.1}}{k\sigma_{zy}}\nu_i$, where again $\nu_i \sim \text{unif}(0.01, 0.2)$ and $\sigma_{zy} = \text{std. dev}(\sqrt{0.1}\sum_i \nu_i z_i)$. Note that δ_i is scaled by $\frac{|\mathcal{I}|}{k}$ (the proportion of invalid instruments), which ensures that the average amount of bias introduced is constant as the number of invalid instruments vary. Error terms u, ϵ_x, ϵ_y were independently generated from a normal distribution, with mean 0 and variances σ_u^2, σ_y^2 and σ_y^2, respectively, whose values were chosen to set the variances of u, x and y to one. These scaling parameters are chosen to enable an easy interpretation of the average treatment effect, β: with this scaling, $\beta = 0.1$ implies that one standard deviation of t causes a 0.1 standard deviation of y and hence the causal effect of t on y explains $0.1^2 = 0.01 = 1\%$ of the variance of y. The only place our simulation differs from Hartwig et al., is the treatment effect is a function of observed coefficients, $x \in \mathbb{R}^{10}$, with each $x_i \sim \text{uniform}(-0.5, 0.5)$, and the treatment effect is defined as, $\beta(x) := \text{round}(x^T\gamma^{(xt)}, 0.1)$, with $\gamma^{(xt)}$ a sparse vector of length 10, with three non-zeros $\gamma_i^{(xt)} \sim \text{uniform}(0.2, 0.5)$. The resulting true

$\beta(x)$, takes on values in $\{-0.3, -0.2, \ldots, 0.2, 0.3\}$. We also use 100 genetic variants instead of the 30 used in Hartwig et al., so that we could test ModeIV in a larger scale scenario. The resulting data generating process is given by,

$$z_i \sim \text{Binomial}(2, p_i) \quad \text{for } i \text{ in } [1 \ldots K], \quad \beta(x) := \text{round}(x^T \gamma^{(xt)}, 0.1).$$

$$t := \sum_{j=1}^{K} \alpha_j z_j + \rho u + \epsilon_x$$

$$y := \beta(x) t + \sum_{j=1}^{K} \delta_j z_j + u + \epsilon_y$$

C.6 Details on the Bootstrap Experiments

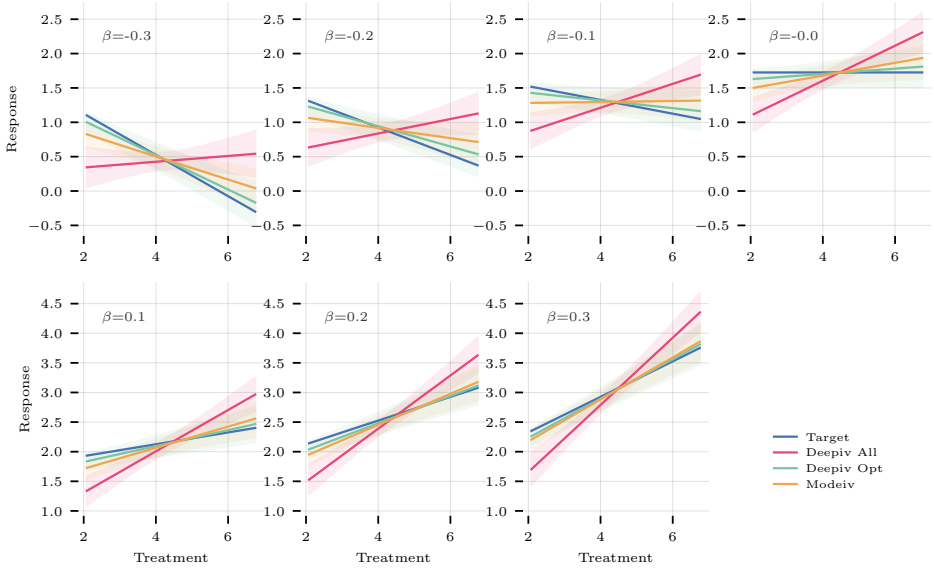

Figure C.2: Bootstrap 90% confidence intervals of conditional dose-response curves. These plots show the average prediction and intervals, averaged across values of the conditioning variable, x, such that the true slope, $\beta(x)$, is given by the value indicated in the plots.

For all the bootstrap experiments we estimated a bootstrap confidence interval point-wise so, at a given point (t, x, z) the interval is computed as,

$$CI_\alpha(t,x,z) = \bar{\hat{f}}(t,x,z) \pm \mathrm{z}_{\alpha/2}\mathrm{se}(\hat{f}(t,x,z)) \text{ where } \bar{\hat{f}}(t,x,z) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}(t,x,z;D_b^*),$$

$$\mathrm{se}(\hat{f}(t,x,z)) = \sqrt{\frac{1}{B}\sum_{b=1}^{B}\left(\hat{f}(t,x,z;D_b^*) - \bar{\hat{f}}(t,x,z)\right)^2}$$

We used $B = 50$ bootstrap samples where for each bootstrap sample of the dataset D_b^*, we run the full ModeIV procedure (i.e. we fit k estimates of DeepIV on D_b^*, each with a different one of the the k instruments). $\mathrm{z}_{\alpha/2}$ denotes the $\alpha/2$ quantile of a standard normal distribution.

We compute coverage by evaluating the proportion of samples for which the true y falls within an estimated bootstrap confidence interval,

$$E[\mathbf{1}(y \in CI_\alpha(t,x,z))] \approx \frac{1}{n}\sum_i \mathbf{1}(y_i \in I_\alpha(t_i,x_i,z_i))$$

For both the demand and Mendelian randomization experiments we used $n =$ 10 000 test points.

Demand simulations The bootstrap results presented in Table 6.1 give empirical coverage estimates for the demand simulation with between 4/7 and 6/7 valid instruments. We repeat the experiment across 20 random seeds and report average coverage across those experiments.

Mendelian randomization The Mendelian randomization experiments were too computationally expensive to give bootstrap inference results across the whole range of valid instruments and random seeds that we tested in point estimate experiments. Instead, we just tested coverage with 70% valid instruments on a single random seed[1]. The results are summarized in Table C.2 and Figure C.2.

[1] i.e. the random seed for the data generating process was fixed to 1 (chosen arbitrarily) which fixes the γ, δ and α parameters

Model	Coverage (70 / 100 valid)
ModeIV-10	64.76%
ModeIV-20	63.94%
ModeIV-30	62.64%
ModeIV-40	60.38%
ModeIV-50	59.78%
DeepIV-Opt	80.96%
DeepIV-All	30.96%

Table C.2: Mendelian randomization empirical coverage for 90% bootstrap confidence intervals. All the approaches underestimate coverage: the true y falls with the interval 80% of the time, while the various Mode-IV approaches get between 60% and 65%. This is far better than DeepIV-all which only manages 30% because of its far more significant bias.

Figure C.2 shows the confidence intervals for the conditional does-response curves. Recall that in this simulation, the conditional average treatment effect is given by $\beta(x) := \text{round}(x^T\gamma^{(xt)}, 0.1)$, where $\gamma^{(xt)}$ are some unknown parameters, and the rounding ensures that $\beta(x)$ can only take on a fixed number of discrete values. This approach makes visualizing conditional average treatment effects easier as we can examine the average predicted effect for each of the different ground truth values of $\beta(x)$. Of course, these plots are only possible to make given the knowledge of the ground truth $\beta(x)$, so they are only useful as a diagnostic tool in simulated data, not something that we would be able to plot on real data. These conditional does-response curves are given by,

$$E[\hat{y}|t,\beta'] = E_z[E_{x:\beta(x)=\beta'}[\hat{f}(t,x,z)|z]|t,\beta]$$

That is, we average the predictions over all x such that the true value of $\beta(x)$ is β'. Note that the curves are linear because we parameterize the deep network such the the condition treatment-response relationship is linear. The confidence intervals are computed in the same manner,

$$E[CI_\alpha(\hat{y})|t,\beta'] = E_z[E_{x:\beta(x)=\beta'}[CI_\alpha(t,x,z)|z]|t,\beta]$$

The plots show that both ModeIV and DeepIV-Opt do a good job of recovering the ground-truth relationship, particularly for positive values of the ground truth relationship. For negative values, some bias remains, which is the likely cause of the poor coverage numbers shown in Table C.2. The fact that even DeepIV-opt underestimates coverage, suggests this may be the result of finite-sample bias rather than a problem with ModeIV itself.

C.7 Details on Two-Stage Hard Thresholding Experiments

# valid / 100	ATE: $\hat{\beta}$	True Positive	False Positive	True Negative	False Negative
50	0.08	0.47	0.19	0.31	0.03
60	0.03	0.56	0.09	0.31	0.04
70	0.02	0.66	0.04	0.26	0.04
80	0.01	0.75	0.01	0.19	0.05
90	0.00	0.84	0.00	0.10	0.06
100	0.00	0.93	0.00	0.00	0.07

Table C.3: Average treatement effect and confusion matrices for Guo et al. [2018]'s two-stage hard thresholding algorithm on the Mendelian randomization dataset. When 70 or more instruments were valid, it had a low false positive rate. As a result, it accurately recovered the average treatment effect for this simulation ($\beta = 0.$).

We used Guo et al. [2018]'s public R implementation of Two-Stage Hard Thresholding (available here). To estimate $E[y|\text{do}(t), x]$, we ran two-stage least squares on the predicted valid instruments while controlling for the invalid candidates. All experiments were run 30 times with different seeds for the data generating process and we report mean performance.

ModeIV for linear constant treatment effects

We also evaluated ModeIV on Guo et al. [2018]'s low-dimensional data generating process.[2] We used the same network architecture as the MR experiments with a conditionally linear output so we could observe the coefficient on t (i.e. $\hat{\beta}$) directly.

[2]Guo et al.'s high-dimensional DGP involves a large number of candidates that fail the relevance assumption. Given that relevance is testable and ModeIV is computationally expensive, relevance testing is better addressed as a pre-processing step.

Model	n	$\hat{\beta}_{10}$	$\hat{\beta}_{25}$	$\hat{\beta}_{50}$	$\hat{\beta}_{75}$	$\hat{\beta}_{90}$
ModeIV-0.2	1000	1.043	1.151	1.285	1.471	2.565
	10000	0.989	1.013	1.044	1.087	1.249
ModeIV-0.3	1000	1.056	1.151	1.270	1.423	1.734
	10000	0.993	1.013	1.040	1.075	1.132
ModeIV-0.4	1000	1.066	1.155	1.267	1.412	1.665
	10000	0.997	1.015	1.039	1.072	1.125
ModeIV-0.5	1000	1.079	1.161	1.268	1.415	1.667
	10000	1.000	1.016	1.039	1.072	1.128
DeepIV-opt	1000	0.908	0.980	1.059	1.142	1.243
	10000	0.976	0.987	1.004	1.019	1.036

Table C.4: With enough data, ModeIV performs will in the linear setting. Here we show performance on Guo et al. [2018]'s data generating process. The true $\beta = 1$; because ModeIV controls for invalid candidates, there is some variation in predicted $\hat{\beta}$ across examples; we show this with percentiles of the $\hat{\beta}$ estimates which are denoted $\hat{\beta}_p$.

Table C.4 reports percentiles of the estimated ATE with n of 1000 and 10000. This is a massively over-parameterized model for the task (roughly 20 000 parameters for each of the 10 candidates; note that you should never do this if you know the problem is linear!). As expected, ModeIV was less efficient but with 10 000 training examples it picked up only a small amount of bias, $\hat{\text{ATE}} = 1.039$ (for a true ATE of 1). With 1000 samples ModeIV incurred large bias from the invalid candidates.