numbers=sortcompress=open=[=close=]=citesep=,=notesep=,
=

authoryear=open=[=close=]=citesep=;=aysep==yysep=,=notesep=,
=

Transfer Learning to Identify Fair, Causal, or General Representations in Deep Neural Networks

JACY REESE ANTHIS, University of Chicago, USA

This project begins to explore the use of transfer learning to explore the parametric structure of deep neural networks. Deep learning systems tend to have millions, if not billions or trillions, of parameters, and because of the emergent nature of this structure, it remains largely a black box. This is an obstacle to several aims of deep learning: ensuring models are facilitating fair treatment of individuals; understanding when and how models are representing the world in sophisticated, causal, robust ways across domains; and iterating model improvement in a more coordinated manner than mere backpropagation. We propose how transfer learning can be used to identify internal fair, causal, or general representations across the emergent parametric structure. Initial results, consistent with our theory, show how this approach can be useful in the interpretability toolkit for a variety of model goals.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Applied computing** → **Sociology**.

Additional Key Words and Phrases: deep learning, neural networks, interpretability, explainability, fairness, causality, domain adaptation

ACM Reference Format:
Jacy Reese Anthis. 2021. Transfer Learning to Identify Fair, Causal, or General Representations in Deep Neural Networks. In *Proceedings of CMSC 35200: Deep Learning Systems (CMSC 35200).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep neural networks have state-of-the-art performance across a wide range of tasks. By design, these models are made up of many weights, often millions and increasingly many billions, that each determine the effect of a single input node on a single output node. Many general features of the model's structure are explicated by the designer, including the number of nodes per layer, the activation function by which weights affect the next node, the optimizer, and so on. However, the design includes very little about the sort of internal structure of these nodes. This data-driven quality is a key strength of deep learning, but it inherently presents an immense challenge for interpretability and explainability.

By interpretability, we mean the extent to which an outsider to the system can interpret the model. By explainability, we mean the extent to which the system provides an accessible explanation of itself. These closely related concepts (henceforth, we emphasize interpretability, the more general of the two) are important across a wide range of machine learning goals. An interpretable system is better able to be improved upon because its structure can be more easily analyzed, either through logical reasoning or empirical experimentation. It is better able to be tuned towards ethical goals, such as by modifying the model to avoid the unfair treatment of certain groups or other negative effects on people.

In this project, we analyze the use of transfer learning, the process of training a model on one task then applying it to a different task, to identify representations within a deep neural network. One motivating example is assessing the fairness of models with respect to a sensitive characteristic that they did not take as input data. In 2019, the Apple Card was critiqued as customers came to believe that the card was giving unfairly low credit limits to female customers. The company behind the Apple Card, Goldman Sachs, defended itself by saying that their algorithm does not take sex or gender as an input, so it cannot be sexist. In 2021, the New York State Department of Financial Services concluded that there was no evidence of discriminatory practices (Nasiripour and Farrell, 2021). This raises an important question: If the model does not take a certain feature means that the output of the model is invariant to the protected class, which could include invariance across classification rates (demographic parity), across accuracy rates (equalized odds), or other fairness measures. Invariant representations are also a way to operationalize the generalization of a model across domains (Anselmi et al., 2014; Ben-David et al., 2007, 2010; Ganin et al., 2016; Zhao et al., 2018, 2020c), differential privacy (Coavoux et al., 2018; Hamm, 2015, 2017), and even causation in which the model should be invariant to the counterfactual pathway that led to the data we possess (Johansson et al., 2018, 2021; Shalit et al., 2017). It not uncontroversial that deep neural networks are even capable of forming causal representations, as Pearl and Mackenzie (2018) argue that they cannot understand causation, only the mere association of variables. Moreover, it is unclear how to operationalize notions such as fairness, causality, and generality in this context. These may also be linked together in the same representations. For example, Veitch et al. (2021) notes that if we use counterfactual fairness—the requirement that changing protected class shouldn't change model predictions—as the sole meta-criterion by which to evaluate fairness metrics, then we can infer the ideal metric based on the causal structure of the underlying data. When a random variable X causes Y, then if a protected class Z causes X (but not Y), the notion of counterfactual fairness is equivalent to demographic parity, but if instead Z causes Y (but not X), counterfactual fairness is equivalent to a different fairness metric, equalized odds. For example, if having a disability (Z) causes someone to have lower qualifications (X) for a certain job, then for a hiring algorithm (i.e., a prediction of job success; Y) to be counterfactually fair (i.e., disability doesn't affect hiring) means that no matter one's disability, they are equally likely to be hired. If having a disability instead affects job success through means other than qualifications (e.g., people with disabilities are discriminated against in interviews and on the job), then counterfactual fairness means that the model is "disability-blind" in the sense that it accounts only for qualifications.

These various goals are often not entirely aligned with the sheer performance of the model. We face trade-offs. However, given that there are often many models that work almost equally well for a given dataset and task, within the margin of random error, large gains towards some goals may be achieved with little to no sacrifice. This is referred to as "underspecification" (D'Amour et al., 2020), the "multiplicity of good models," or the "Rashomon effect," named after a 1950 Japanese film in which several witnesses report a crime with similar facts but very different impressions of what really happened (Breiman, 2001; D'Amour, 2021; Semenova et al., 2021).

There is ample research on model interpretability in machine vision. Experimental work varying model features can "autopsy" its internal structure (Aafaq et al., 2019; Lorieul et al., 2016). Saliency maps and localization techniques can identify the pixels with the maximum gradient or other criteria to approximate model attention to certain areas of the image (Baehrens et al., 2009; Simonyan et al., 2014; Selvaraju et al., 2017). These mappings can be constrained such that the changed image stays within a manifold determined by the original dataset (Miller et al., 2019; Nguyen et al., 2016), and a common example of such a manifold is a Generative Adversarial Network (GAN), which produces output that can help interpret the generative network (Chen et al., 2016; Kobayashi et al., 2017).

2 PROBLEM SETUP

We take the typical machine learning problem: An input dataset $X \in \mathcal{X}$ has n vectors $x_1, x_2, ..., x_n \in X$ each with length p, and each component is a feature of each datum. The output variable $Y \in \mathcal{Y}$ has n components $y_1, y_2, ..., y_n \in Y$, and each component is the output of the model for each datum. The goal is to estimate a prediction function $f : \mathcal{X} \mapsto \mathcal{Y}$ that minimizes $\mathbb{E}[\ell(f(X), Y)]$ for some loss function $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$.

The typical motivation for identifying internal representations within the model is **invariant representation learning**, in which there is a third random variable A on which we prefer our prediction function to be invariant. For the goal of fairness, this can be a protected class such as race, gender, or disability. Formally, we prefer if the model learns a transformation function $Z = g(X)$ that retains as much information about the target Y without retaining information about the sensitive characteristic A. Following (Zhao et al., 2020b), we define:

$$R_Y^*(g) := \inf_h \mathbb{E}_{\mathcal{D}}[\ell(h(g(X)), Y)]$$

as the optimal risk in predicting Y using the feature encoding $Z = g(X)$ under loss ℓ, where h is the post-representation component of the prediction function that takes as input the output of the representation function g. Similarly,

$$R_A^*(g) := \inf_h \mathbb{E}_{\mathcal{D}}[\ell(h(g(X)), A)]$$

as the optimal risk in predicting A using the representation of X denoted by $Z = g(X)$ under some loss ℓ. The invariant goal is to find a representation function g such that the response prediction loss $R_Y^*(g)$ is minimized while the sensitive characteristic prediction loss $R_A^*(g)$ is maximized.

The goal of the present work is to provide a metric for the extent to which information about A is retained in such a representation $Z = g(X)$ present in a given model.

3 TRANSFER LEARNING TO IDENTIFY REPRESENTATIONS

It is often not trivial to specify the component functions g or h (i.e., encoding and decoding representations, or representing and extracting the data). In a deep neural network with q layers, we can conceptualize g and h as each having $q-1$ subcomponent functions. The first layer has only a g_1 component; the second has a g_2 and h_2 component; and so on until the second-to-last layer has g_q and h_{q-1} components, and the last layer has only a h_q component. Alternatively, we can view each layer as only having a g representation function, in which X is transformed representation-by-representation into an approximate representation of Y. The challenge, then, is to analyze some meaningful portion of g and the information therein. There are at least three plausible metrics for this:

(1) *Transfer learning* by freezing inner layers and retraining the network to a task (e.g., predicting that variable) in which higher performance corresponds to the extent to which the frozen internal representations contain information about the variable of interest,
(2) *Sampling* many subsets of parameters within the network and measuring each subset's correlation or other associations with the variable of interest, or
(3) *Projecting* a dimension of the variable of interest onto an intermediate or output layer and measuring correlation or other associations with the projection of other dimensions (i.e., projections of other independent variables or the dependent variable),
(4) *Generating* model output using a method such as General Adversarial Networks (GAN) and measuring the output's correlation or other associations with other content.

The focus of this work is transfer learning from a source domain $\mathcal{D}_S = \{\mathcal{X}_S, \mathbb{P}(\mathcal{X}_S)\}$ and its learning task $\mathcal{T}_S = \{\mathcal{Y}_S, f_S(X_S)\}$ to a target domain $\mathcal{D}_T = \{\mathcal{X}_T, \mathbb{P}(\mathcal{X}_T)\}$ and its learning task $\mathcal{T}_T = \{\mathcal{Y}_T, f_T(X_T)\}$. Without frozen parameters, the entire f prediction function will be trainable in the new model, and changes may correspond to the g representation component and h post-representation component. Freezing all but the final layer ensures g is held constant because the final layer only serves to predict Y based on a representation Z. By allowing the h_q subcomponent of the last layer to vary, we can analyze Z. Freezing fewer layers may be informative as well, as it allows more variation in h. By allowing changes in g, new representations can be formed by the model, but as long as the information flow cannot skip layers (e.g., a long short-term memory model) and the frozen layers are a contínguous set beginning with the first layer, the training can only make use of the representations formed by the last frozen layer, and thus performance after transfer learning can only depend on g representations developed in the source domain $\mathcal{D}_S$.

A fundamental challenge is how to differentiate whether a network that performs well on the transfer learning task has a high-fidelity representation of the sensitive characteristic, many other social characteristics and uses those others to only form a representation of the sensitive characteristic when retrained, or both sensitive and non-sensitive characteristics. In the latter two cases, the network may be seen as generally rigorous and using them in an acceptable way. In fact, one can imagine that the network is even forming a sensitive representation in order to mitigate its own bias. To address this, we can modify (a) the initial transfer learning metric:

$$a = \ell(A, \hat{A})$$

to (b) transfer learning performance on prediction of the sensitive characteristic relative to the prediction of m other variables $B_1, ..., B_m$ masked from trainining in the source domain $\mathcal{D}$,:

$$b = \frac{\ell(A, \hat{A})}{\frac{1}{m}\sum_1^m \ell(B, \hat{B})}$$

Similarly, one could differentiate whether the network merely retained understanding of the sensitive characteristic versus increasing its understanding of the sensitive characteristic by measuring (c) performance relative to the performance of a uninomial or multinomial logistic regression trained on the input data, with loss denoted $\hat{A}_{LM}$:

$$c = \frac{\ell(A, \hat{A})}{\ell(A, \hat{A}_{LM})}$$

Another approach would be to (d) condense the model's representation of sensitive characteristics by using a network architecture with a low-dimensional bottleneck layer, as used in autoencoders, and perform transfer learning on a network frozen only up to that layer, with bottleneck loss denoted $\hat{A}_{BN}$:

$$d = \ell(A, \hat{A}_{BN})$$

We can also form 4 additional metrics based on permutations: (bc) performance relative to other variables and logistic regression, where weights $\alpha + \beta = 1$ are the relative importance of the two terms in the denominator:

$$bc = \frac{\ell(A, \hat{A})}{\alpha\ell(A, \hat{A}_{LM}) + \frac{\beta}{m}\sum_1^m \ell(B, \hat{B})}$$

(bd) performance with a bottleneck layer relative to other variables:

$$bd = \frac{\ell(A, \hat{A}_{BN})}{\frac{1}{m}\sum_1^m \ell(B, \hat{B})}$$

(cd) performance with a bottleneck layer relative to logistic regression.

$$cd = \frac{\ell(A, \hat{A})}{\ell(A, \hat{A}_{LR})}$$

and (bdc) performance with a bottleneck layer relative to other variables and logistic regression:

$$bcd = \frac{\ell(A, \hat{A}_{BN})}{\alpha \ell(A, \hat{A}_{LR}) + \frac{\beta}{m} \sum_1^m \ell(B, \hat{B}_{BN})}$$

Each of these metrics can also be normalized for comparison.

4 SIMULATIONS

We conduct simulations to demonstrate the application of these metrics to a protypical dataset with a coherent data-generating process. In empirical datasets, the issue of inner representations arises when the model can form a better predictin of Y by forming a nonlinear representation $Z = g(X)$, where Z contains information about A. If the representation were linear, the model would need no representation because the extraction function h could be completed in the final layer based on a linear combination of inputs. Thus, we generate several noisy, random variables $x_1, ..., x_10 \in X$, sensitive variables A and B that are nonlinearly predicted by x_1, x_2 and x_3, x_4 respectively (i.e., an exclusive disjunction (XOR) gate of variable signs) with random noise, and an outcome variable Y that is predicted by A with random noise.

In our simulations, each linear model has approximately 50% accuracy, no better than chance because there is no linear information about Y, A, or B in X. For intepretability, we use accuracy of the predicted signs as the (inverted) loss function $\ell : \mathcal{Y} \mapsto \mathcal{Y}$.

We then define and train a TensorFlow neural network to predict Y based on X. The model does not have direct access to A, as in the Apple Card example, but it is incentivized to form a $Z \approx A$ as that is the best prediction of Y that the model could make. However, the network's ability to represent A depends on having sufficient nodes to do so. The minimum number of nodes for an XOR gate on x_1, x_2 is 3: 1 AND with 1 input from an OR and 1 input from a NAND, both on the input layer. However, in larger networks, the model may not have sufficient (or any) incentive to parsimoniously represent the XOR gate. Because there are (infinitely, with continuous weights) less parsimonious networks to represent this logic, it is very unlikely (impossible with precision) that a larger network would happen upon the parsimonious representation. This is analogous to word embeddings that represent words in many dimensions: it is very unlikely that a natural semantic dimension (e.g., plural words, association with wealth, association with gender) will align with a single one of the model's dimensions. Thus, we experiment with a range of layer widths and depths. See Fig. 1 for an example of this XOR logic.

With the minimal number of nodes for XOR logic—and thus no room for a bottleneck layer so no metrics involving (d)—we find that the model can predict the sign of Y with approximately 64% accuracy on the test set. To produce the metric (a), we need to freeze all except the final layer of the network and retrain it to predict A, which results in approximately 69% accuracy, which is even better than Y prediction because there is less random noise in A than Y, relative to X.

Retraining the output layer to predict B leads to approximately 50% accuracy, indicating that Z has much less information about B, which is expected in the minimally sized network. Using this data and inaccuracy as the loss function (such that higher metrics indicate more informative representation), we can calculate metrics (a), (b), (c), and (bc) with $\alpha = \beta = 0.5$:

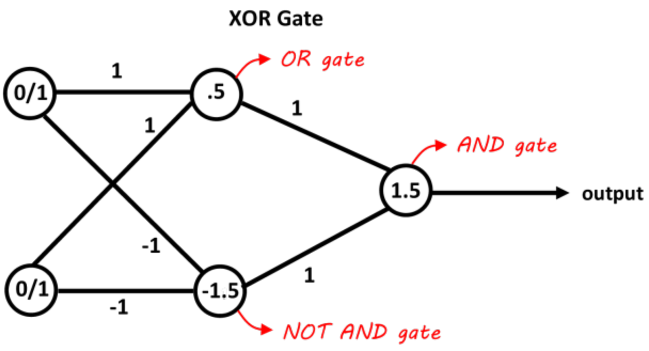

Fig. 1. Example of XOR implemented in a neural network from https://blog.abhranil.net/2015/03/03/training-neural-networks-with-genetic-algorithms/.

$$\begin{aligned}
a &= \ell(A, \hat{A}) \\
&= 0.36 \\
b &= \frac{\ell(A, \hat{A})}{\frac{1}{m}\sum_1^m \ell(B, \hat{B})} \\
&= \frac{0.36}{0.5} \\
&= 0.72 \\
c &= \frac{\ell(A, \hat{A})}{\ell(A, \hat{A}_{LM})} \\
&= \frac{0.36}{0.5} \\
&= 0.72 \\
bc &= \frac{\ell(A, \hat{A})}{\alpha \ell(A, \hat{A}_{LM}) + \frac{\beta}{m}\sum_1^m \ell(B, \hat{B})} \\
&= \frac{0.36}{0.5 * 0.5 + 0.5 * 0.5} \\
&= 0.72
\end{aligned}$$

In some experiments with this node size, the accuracy for B was as high as 53%. Why might this not be 50% (± random noise)? I think that, in general, the model does not end up with weights of 0 for $x_3, ...x_10$, even though these are not part of the actual process used to create Y. Thus, if the 2 inner nodes happen to have appropriately signed weights for producing certain logic, they can be repurposed by the output layer for estimation of B. In fact, I found a very interesting result when I ran a similar model with 100,000 samples (instead of 10,000) and 100 epochs (instead of 50). As reproduced below, the training and validation loss were stable until the 40th epoch, but then in the next 18 epochs, the model very quickly fell in loss before stabilizing again. Looking at the weights of this network, I believe this is an example of the phenomenon I just described, where the model was able to find a useful interpretation of its 2 inner nodes as logical gates to predict B from x_3, x_4, despite the much larger influence of x_1, x_2. See Fig. 2 for this drop in loss.

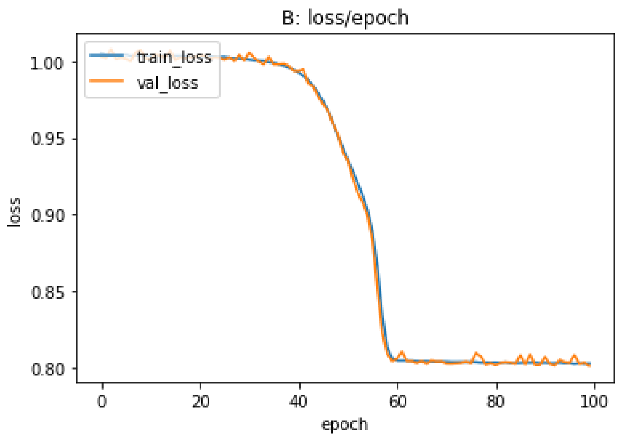

Fig. 2. Drop in model loss when training on B.

We can rebuild the model with a modified data generating process such that Y is the sum of A and B with random noise. While the linear model results stay the same—and thus (c) stays the same—(b) and (bc) are both reduced. Namely, the accuracy for training on B rises to 52%.

5 DISCUSSION

This project was ambitious and challenging, so it did not go as I planned. I hoped to use a real world dataset and was originally using TabNet, a neural network designed specifically for tabular data given the usual focus of architectures on visual or text data (Arik and Pfister, 2020), but it is difficult to make neural networks perform well on tabular data and the existing datasets are limited. For example, the common fairness-and-finance datasets are on the order of 1,000 observations. I also have ended up thinking that the case for transfer learning as a metric for model representation is not as promising as I expected; it is certainly a "garden of forking paths," and I am not sure if peer reviewers would be convinced that this sort of work is valuable.

Nonetheless, this project has been very useful in thinking through the (1)-(4) methods for identifying model representation, as well as formally writing out the mathematical representations and going through the nuts and bolts of building a small neural network and trying to get meaningful results from the simulation.

In the future, I would like to experiment with larger networks, bottleneck layers, and manually checking the correlations of various weights with input variables as well as projecting the input variables onto the spaces spanned by the inner layer weights. I have also seen a talk by Sendhil Mullainathan, Professor of Computation and Behavioral Science at Chicago Booth, in which he presented a similar interpretability analysis using GANs to identify what image features were predictive of judge decisions in criminal bail hearings. I am enrolled in Sendhil's class, BUSN 3920 (Data Science, Algorithms, and Society), where hopefully I can continue this line of research. I still hope to develop it into my third-year research paper required by the Econometrics & Statistics department.

I am also looking into a similar interpretability project using large language models. For example, at the end of November, the Machine Intelligence Research Institute (MIRI) published a call for work on their ”Visible Thoughts Project” (Soares, 2021), which is aiming to train language models to produce ongoing ”thoughts” or ”explanations” of their text generation as users walk through an AI Dungeon environment, similar to the role of the Dungeon Master in the popular tabletop game Dungeons and Dragons. You can imagine, for example, fine-tuning the model to predict not just the next words in its main corpuses (e.g., Common Crawl), but to simulataneously predict the sort of explanation a human would give for those next words. MIRI's focus right now is just to build a dataset of such human-provided explanations, but perhaps I could get access to that dataset or do a similar project during my PhD.

ACKNOWLEDGMENTS

Many thanks to Ian Foster, Rick Stevens, Peng Ding, and students in the Autumn 2021 CMSC 35200 Deep Learning Systems course at the University of Chicago for fruitful discussions.

REFERENCES

Nayyer Aafaq, Naveed Akhtar, Wei Liu, and Ajmal Mian. 2019. Empirical Autopsy of Deep Video Captioning Frameworks. *arXiv:1911.09345 [cs]* (Nov. 2019). arXiv:1911.09345 [cs]

Fabio Anselmi, Joel Z. Leibo, Lorenzo Rosasco, Jim Mutch, Andrea Tacchetti, and Tomaso Poggio. 2014. Unsupervised Learning of Invariant Representations in Hierarchical Architectures. *arXiv:1311.4158 [cs]* (March 2014). arXiv:1311.4158 [cs]

Sercan O. Arik and Tomas Pfister. 2020. TabNet: Attentive Interpretable Tabular Learning. *arXiv:1908.07442 [cs, stat]* (Dec. 2020). arXiv:1908.07442 [cs, stat]

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Mueller. 2009. How to Explain Individual Classification Decisions. *arXiv:0912.1128 [cs, stat]* (Dec. 2009). arXiv:0912.1128 [cs, stat]

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A Theory of Learning from Different Domains. *Machine Learning* 79, 1-2 (May 2010), 151–175. https://doi.org/10.1007/s10994-009-5152-4

Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. 2007. Analysis of Representations for Domain Adaptation. In *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt, and T. Hoffman (Eds.), Vol. 19. MIT Press.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. ACM, Virtual Event Canada, 610–623. https://doi.org/10.1145/3442188.3445922

Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. 2016. Man Is to Computer Programmer as Woman Is to Homemaker? Debiasing Word Embeddings. *arXiv:1607.06520 [cs, stat]* (July 2016). arXiv:1607.06520 [cs, stat]

Leo Breiman. 2001. Statistical Modeling: The Two Cultures. *Statist. Sci.* 16, 3 (Aug. 2001). https://doi.org/10.1214/ss/1009213726

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *arXiv:1606.03657 [cs, stat]* (June 2016). arXiv:1606.03657 [cs, stat]

Alexandra Chouldechova. 2017. Fair Prediction with Disparate Impact: A Study of Bias in Recidivism Prediction Instruments. *arXiv:1703.00056 [cs, stat]* (Feb. 2017). arXiv:1703.00056 [cs, stat]

Maximin Coavoux, Shashi Narayan, and Shay B. Cohen. 2018. Privacy-Preserving Neural Representations of Text. *arXiv:1808.09408 [cs]* (Aug. 2018). arXiv:1808.09408 [cs]

Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. 2017. Algorithmic Decision Making and the Cost of Fairness. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Halifax NS Canada, 797–806. https://doi.org/10.1145/3097983.3098095

Alexander D'Amour. 2021. Revisiting Rashomon: A Comment on ”The Two Cultures”. *Observational Studies* 7, 1 (2021), 59–63. https://doi.org/10.1353/obs.2021.0022

Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D. Hoffman, Farhad Hormozdiari, Neil Houlsby, Shaobo Hou, Ghassen Jerfel, Alan Karthikesalingam, Mario Lucic, Yian Ma, Cory McLean, Diana Mincu, Akinori Mitani, Andrea Montanari, Zachary Nado, Vivek Natarajan, Christopher Nielson, Thomas F. Osborne, Rajiv Raman, Kim Ramasamy, Rory Sayres, Jessica Schrouff, Martin Seneviratne, Shannon Sequeira, Harini Suresh, Victor Veitch, Max Vladymyrov, Xuezhi Wang, Kellie Webster, Steve Yadlowsky, Taedong Yun, Xiaohua Zhai, and D. Sculley. 2020. Underspecification Presents Challenges for Credibility in Modern Machine Learning. *arXiv:2011.03395 [cs, stat]* (Nov. 2020). arXiv:2011.03395 [cs, stat]

Harrison Edwards and Amos Storkey. 2016. Censoring Representations with an Adversary. *arXiv:1511.05897 [cs, stat]* (March 2016). arXiv:1511.05897 [cs, stat]

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-Adversarial Training of Neural Networks. *arXiv:1505.07818 [cs, stat]* (May 2016). arXiv:1505.07818 [cs, stat]

Jihun Hamm. 2015. Preserving Privacy of Continuous High-Dimensional Data with Minimax Filters. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, Guy Lebanon and S. V. N. Vishwanathan (Eds.), Vol. 38. PMLR, 324–332.

Jihun Hamm. 2017. Minimax Filter: Learning to Preserve Privacy from Inference Attacks. *arXiv:1610.03577 [cs]* (Dec. 2017). arXiv:1610.03577 [cs]

Fredrik D. Johansson, Uri Shalit, Nathan Kallus, and David Sontag. 2021. Generalization Bounds and Representation Learning for Estimation of Potential Outcomes and Causal Effects. *arXiv:2001.07426 [cs, stat]* (March 2021). arXiv:2001.07426 [cs, stat]

Fredrik D. Johansson, Uri Shalit, and David Sontag. 2018. Learning Representations for Counterfactual Inference. *arXiv:1605.03661 [cs, stat]* (June 2018). arXiv:1605.03661 [cs, stat]

Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. 2016. Inherent Trade-Offs in the Fair Determination of Risk Scores. *arXiv:1609.05807 [cs, stat]* (Nov. 2016). arXiv:1609.05807 [cs, stat]

Masayuki Kobayashi, Masanori Suganuma, and Tomoharu Nagao. 2017. Generative Adversarial Network for Visualizing Convolutional Network. In *2017 IEEE 10th International Workshop on Computational Intelligence and Applications (IWCIA)*. IEEE, Hiroshima, 153–158. https://doi.org/10.1109/IWCIA.2017.8203577

Titouan Lorieul, Antoine Ghorra, and Bernard Merialdo. 2016. Static and Dynamic Autopsy of Deep Networks. In *2016 14th International Workshop on Content-Based Multimedia Indexing (CBMI)*. IEEE, Bucharest, Romania, 1–4. https://doi.org/10.1109/CBMI.2016.7500267

Andrew Miller, Ziad Obermeyer, John Cunningham, and Sendhil Mullainathan. 2019. Discriminative Regularization for Latent Variable Models with Applications to Electrocardiography. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 4585–4594.

Shahien Nasiripour and Greg Farrell. 2021. Goldman Cleared of Bias in New York Review of Apple Card. *Bloomberg* (2021).

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. 2016. Synthesizing the Preferred Inputs for Neurons in Neural Networks via Deep Generator Networks. *arXiv:1605.09304 [cs]* (Nov. 2016). arXiv:1605.09304 [cs]

Judea Pearl and Dana Mackenzie. 2018. *The Book of Why: The New Science of Cause and Effect.* Basic Books, New York.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Lesia Semenova, Cynthia Rudin, and Ronald Parr. 2021. A Study in Rashomon Curves and Volumes: A New Perspective on Generalization and Model Simplicity in Machine Learning. *arXiv:1908.01755 [cs, stat]* (April 2021). arXiv:1908.01755 [cs, stat]

Uri Shalit, Fredrik D. Johansson, and David Sontag. 2017. Estimating Individual Treatment Effect: Generalization Bounds and Algorithms. *arXiv:1606.03976 [cs, stat]* (May 2017). arXiv:1606.03976 [cs, stat]

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [cs]* (April 2014). arXiv:1312.6034 [cs]

Nate Soares. 2021. Visible Thoughts Project and Bounty Announcement.

Victor Veitch, Alexander D'Amour, Steve Yadlowsky, and Jacob Eisenstein. 2021. Counterfactual Invariance to Spurious Correlations: Why and How to Pass Stress Tests. *arXiv:2106.00545 [cs, stat]* (Nov. 2021). arXiv:2106.00545 [cs, stat]

Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning Fair Representations. *Proceedings of Machine Learning Research* 28, 3 (2013), 325–333.

Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. 2018. Mitigating Unwanted Biases with Adversarial Learning. *arXiv:1801.07593 [cs]* (Jan. 2018). arXiv:1801.07593 [cs]

Han Zhao, Amanda Coston, Tameem Adel, and Geoffrey J. Gordon. 2020a. Conditional Learning of Fair Representations. *arXiv:1910.07162 [cs, stat]* (Feb. 2020). arXiv:1910.07162 [cs, stat]

Han Zhao, Chen Dan, Bryon Aragam, Tommi S. Jaakkola, Geoffrey J. Gordon, and Pradeep Ravikumar. 2020b. Fundamental Limits and Tradeoffs in Invariant Representation Learning. *arXiv:2012.10713 [cs, stat]* (Dec. 2020). arXiv:2012.10713 [cs, stat]

Han Zhao, Junjie Hu, and Andrej Risteski. 2020c. On Learning Language-Invariant Representations for Universal Machine Translation. *arXiv:2008.04510 [cs, stat]* (Aug. 2020). arXiv:2008.04510 [cs, stat]

Han Zhao, Shanghang Zhang, Guanhang Wu, José M. F. Moura, Joao P Costeira, and Geoffrey J Gordon. 2018. Adversarial Multiple Source Domain Adaptation. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach,

H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc.

A SIMULATION CODE

The simulation code is available at
https://github.com/jacyanthis/cmsc35200/blob/main/simulation.py and copied below for convenience.

```
---------------------------------------------------------------------------
import numpy as np
import pydot
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Activation, Dropout
from tensorflow.keras.models import Model
from keras.utils.vis_utils import plot_model
import pytorch_tabnet
from pytorch_tabnet.tab_model import TabNetClassifier
import torch

def identify_representations(num_samples=1000, num_features=6, p=.9,
                             layer_widths=[2], batch_size=10, epochs_per_run=10):
  start = time()

  # Set random seeds for replicability
  np.random.seed(1)
  tf.random.set_seed(1)

  # Create random features
  X = np.random.randn(num_samples, num_features)
  q = 1 - p

  # Create a variable A that is well-predicted by two of the random variables
  A = np.array([])
  for x in X:
    xor = -1 if (np.sign(x[0]) == np.sign(x[1])) else 1
    A = np.append(A, np.random.choice([xor*abs(np.random.randn()), -xor*abs(np.random.ra

  # Create a variable that is well-predicted by two other random variables
  B = np.array([])
  for x in X:
    xor = -1 if (np.sign(x[2]) == np.sign(x[3])) else 1
    B = np.append(B, np.random.choice([xor*abs(np.random.randn()), -xor*abs(np.random.ra

  # Create a variable Y that is well-predicted by A
  Y = np.array([])
```

```
for a in A:
  Y = np.append(Y, np.random.choice([a, -a], p=[p, q]))
# Alternatively, well-predicted by A and B
# for i in range(len(A)):
#   Y = np.append(Y, np.random.choice([A[i] + B[i], -(A[i] + B[i])], p=[p, q]))

# Prepare training test split
X_train, X_test, Y_train, Y_test, A_train, A_test, B_train, B_test = train_test_split(
    np.array(X), np.array(Y), np.array(A), np.array(B), test_size=0.2, random_state=1)

# Fit linear models for comparison
lm1 = sm.OLS(Y_train, X_train).fit()
lm2 = sm.OLS(A_train, X_train).fit()
lm3 = sm.OLS(B_train, X_train).fit()

# Make predictions for linear models
predictions1r = lm1.predict(X_test)
i = 0
for idx, j in enumerate(predictions1r):
    if (np.sign(j) == np.sign(Y_test[idx])):
        i += 1
print(f'LM accuracy on Y: {i / len(X_test)}')

predictions2r = lm2.predict(X_test)
i = 0
for idx, j in enumerate(predictions2r):
    if (np.sign(j) == np.sign(A_test[idx])):
        i += 1
print(f'LM accuracy on A: {i / len(X_test)}')

predictions3r = lm3.predict(X_test)
i = 0
for idx, j in enumerate(predictions3r):
    if (np.sign(j) == np.sign(B_test[idx])):
        i += 1
print(f'LM accuracy for B: {i / len(X_test)}')


# Define model layers, varying extent of representations
# To-do: Allow for > 2 layers
ip_layer = Input(shape=(X.shape[1]))
if len(layer_widths) >= 2:
    if len(layer_widths) > 2:
      print('Too many layers. Using first 2.')
    dl1 = Dense(layer_widths[0], activation='relu')(ip_layer)
    dl2 = Dense(layer_widths[1], activation='relu')(ip_layer)
    output = Dense(1)(dl2)
else:
        dl1 = Dense(layer_widths[0], activation='relu')(ip_layer)
        output = Dense(1)(dl1)

    # Define model and compile
    global model
    model = Model(inputs=ip_layer, outputs=output)
    model.compile(loss=tf.keras.losses.MeanSquaredError(),
                  optimizer="adam")

    # Train model 1
    history1 = model.fit(X_train, Y_train,
                        validation_data=(X_test,Y_test),
                        batch_size=batch_size, epochs=epochs_per_run)

    # Plot performance 1
    fig1 = plt.figure()
    plt.plot(history1.history['loss'])
    plt.plot(history1.history['val_loss'])
    plt.title('Y: loss/epoch')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.ylim([0.8, 1.1])
    plt.legend(['train_loss','val_loss'], loc='upper left')

    # Make predictions 1
    predictions1n = model.predict(X_test).T[0]
    i = 0
    for idx, j in enumerate(predictions1n):
        if (np.sign(j) == np.sign(Y_test[idx])):
            i += 1
    print(f'NN accuracy on Y: {i / len(X_test)}')

    # Freeze all except final layer for transfer learning
    for i in range(len(model.layers)-1):
      model.layers[i].trainable = False

    # Train model 2
    history2 = model.fit(X_train, A_train,
                        validation_data=(X_test,A_test),
                        batch_size=batch_size, epochs=epochs_per_run)

    # Plot performance 2
    fig2 = plt.figure()
    plt.plot(history2.history['loss'])
    plt.plot(history2.history['val_loss'])
    plt.title('A: loss/epoch')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.ylim([0.8, 1.1])
    plt.legend(['train_loss','val_loss'], loc='upper left')

    # Make predictions 2
    predictions2n = model.predict(X_test).T[0]
    i = 0
    for idx, j in enumerate(predictions2n):
        if (np.sign(j) == np.sign(A_test[idx])):
            i += 1
    print(f'NN accuracy on A: {i / len(X_test)}')

    # Train model 3
    history3 = model.fit(X_train, B_train,
                         validation_data=(X_test,B_test),
                         batch_size=batch_size, epochs=epochs_per_run)

    # Plot performance 3
    fig3 = plt.figure()
    plt.plot(history3.history['loss'])
    plt.plot(history3.history['val_loss'])
    plt.title('B: loss/epoch')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.ylim([0.8, 1.1])
    plt.legend(['train_loss','val_loss'], loc='upper left')

    # Make predictions 3
    predictions3n = model.predict(X_test).T[0]
    i = 0
    for idx, j in enumerate(predictions3n):
        if (np.sign(j) == np.sign(B_test[idx])):
            i += 1
    print(f'NN accuracy on B: {i / len(X_test)}')

    print(f'Time elapsed: {(time() - start)/60} minutes')

# Run the simulation

identify_representations(num_samples=10000, num_features=10, p=.9,
                         layer_widths=[2], batch_size=10, epochs_per_run=50)
---------------------------------------------------------------------------
```