Learning Latent Causal Structures with a Redundant Input Neural Network

Jonathan D. Young, Bryan Andrews, Gregory F. Cooper, Xinghua Lu
Intelligent Systems Program, University of Pittsburgh, Pittsburgh, PA
jdy10@pitt.edu, bja43@pitt.edu, gfc@pitt.edu, xinghua@pitt.edu

Abstract

Most causal discovery algorithms find causal structure among a set of observed variables. Learning the causal structure among latent variables remains an important open problem, particularly when using high-dimensional data. In this paper, we address a problem for which it is known that inputs *cause* outputs, and these causal relationships are encoded by a causal network among a set of an unknown number of latent variables. We developed a deep learning model, which we call a redundant input neural network (RINN), with a modified architecture and a regularized objective function to find causal relationships between input, hidden, and output variables. More specifically, our model allows input variables to directly interact with all latent variables in a neural network to influence what information the latent variables should encode in order to generate the output variables accurately. In this setting, the direct connections between input and latent variables makes the latent variables partially interpretable; furthermore, the connectivity among the latent variables in the neural network serves to model their potential causal relationships to each other and to the output variables. A series of simulation experiments provide support that the RINN method can successfully recover latent causal structure between input and output variables.

1 INTRODUCTION

Causal discovery is a type of machine learning that focuses on learning causal relationships from data, often solely from observational data. In this context, a causal structure is represented as a graph $G = (V, E)$ containing a set of vertices V and a set of edges E, and the goal is to infer the data-generating causal structure from data. The majority of contemporary causal discovery algorithms involve searching the space of possible structures to identify the structure supported by the data, and there have been numerous algorithms developed for this purpose [Spirtes et al., 2000, Cooper, 1999, Heinze-Deml et al., 2018, Peters et al., 2017, Lagani et al., 2016]. Most causal discovery algorithms find causal structure among the observed variables of a dataset. Learning the causal structure among latent variables remains an important open problem, particularly when using high-dimensional data or attempting to leverage *a priori* knowledge that one set of observed variables causally influences a non-overlapping (disjoint) set of observed variables. Latent causal structure can provide additional important insight into the causal mechanisms of a system or process. Thus, new approaches to discovering latent causal structure are needed. In this work, we explore the utility of using deep learning models for finding latent causal structure when the data consist of two sets of variables and it is known *a priori* that one set ("inputs") causes the other ("outputs")—and the causal path from inputs to outputs is mediated by a set of an unknown number of latent variables, among which the causal structure is also unknown.

This computational problem has important applications, including its application to cancer biology. Cancer results from somatic genomic alterations (SGAs) (e.g., mutations in DNA) that perturb the functions of signaling proteins and cause aberrant activation or inactivation of signaling pathways, which often eventually influences gene expression. Typically, an SGA only directly affects the function of a single signaling protein, and its impact is transmitted through a cascade of signaling proteins to influence gene expression. Since the functional states of signaling proteins in pathways are usually not measured, the whole signaling system can be thought of as a set of hierarchically organized latent variables of which we would like to infer how changed states of some signaling proteins causally influence the state of others. In a cancer cell, when one signaling protein is perturbed by an SGA event (an observed input variable), it may causally affect the functional states of the proteins in a pathway (which are not observed) and eventually result in changed gene expression (observed output variables). Thus, our task is to learn how a set of observed input variables causally influence a set of observed output variables, through signals transmitted among a set of latent variables.

In contrast to traditional causal discovery methods, here we explore a modified deep learning model to learn latent causal relationships. Deep learning represents a family of machine learning algorithms or strategies that originated from artificial neural networks (ANN). ANNs represent a framework, loosely inspired by biological neurons, for learning a function (represented as a set of parameters or weights) that maps inputs to outputs. An essential characteristic of deep learning models is their ability to learn compositional representations of the data [Lee et al., 2008, LeCun et al., 2015]. A deep learning model is composed of multiple layers of latent variables (hidden nodes or units) [LeCun et al., 2015], which learn to represent the complex statistical structure embedded in the data, such that different hidden layers capture statistical structure of different degrees of complexity. Researchers have found that deep learning models can represent the hierarchical organization of signaling molecules in a cell, with latent variables as natural representations of unobserved activation states of signaling molecules [Chen et al., 2016, Young et al., 2017, Lu et al., 2018, Tao et al., 2020].

Conventional neural networks behave like black boxes in that it is difficult to interpret what signal a hidden node in the network encodes. We hypothesize that with certain modifications inspired by the biological problem mentioned above, one may learn a partially interpretable deep neural network so that the relationships between latent variables can be interpreted as part of a causal chain from input to output variable. To this end, we designed a modified deep neural network, the redundant input neural network (RINN) that allows each input variable to directly connect to each latent variable within the deep learning hierarchy. This redundant input structure allows one to learn direct causal relationships between an input variable and any latent variable. We also developed a robust pipeline for testing an algorithm's ability to capture latent causal structure, including causal simulated data inspired by cellular signaling pathways, a method for visualizing the weights of a neural network, and a method for measuring precision and recall of the causal structure. In addition, we compare the RINN with a conventional feedforward deep neural network (DNN), a deep belief network (DBN), a RINN optimized with a constrained evolutionary strategy, and the DM algorithm (a causal discovery algorithm).

2 RELATED WORK

The work in this study concentrates on finding latent causal relationships when the causal direction between inputs and outputs is known. The DM (Detect MIMIC (Multiple Indicators Multiple Input Causes)) algorithm is a causal discovery algorithm for finding latent causal relationships between inputs and outputs [Murray-Watters and Glymour, 2015]. The DM algorithm uses a series of heuristics based on conditional independence and Sober's criterion [Sober, 1998] to identify the latent causal structure. One limitation of the DM algorithm is that it constrains each latent variable to be adjacent to at least one input and one output variable. This limits the hierarchical and compositional relationships that DM can identify, which makes it less accurate for cellular signaling pathway discovery. In addition to the DM algorithm, there are other algorithms developed to find latent structure, including factor analysis algorithms, algorithms that use variations of the Expectation-Maximization (EM) algorithm to find latent structure [Friedman et al., 1997, Elidan and Friedman, 2005], and algorithms based on fast causal inference (FCI) [Spirtes et al., 1995, Colombo et al., 2012]. These algorithms are, in general, highly constrained, intractable in high-dimensional spaces, return only a subset of the latent causal relationships, *or* don't leverage prior knowledge that input variables cause output variables.

Recently, there has been an increased interest in the deep learning community to combine deep learning with causal discovery. Kalainathan et al. [2018] developed an algorithm named structural agnostic model (SAM) that trains multiple generative adversarial networks (GAN), one GAN to generate each variable in $\boldsymbol{X}$, the measured variables or features in a dataset. Each GAN uses $\boldsymbol{X}_{-j}$ variables (i.e., all variables in $\boldsymbol{X}$ other than X_j) multiplied by a weight mask, $\boldsymbol{a_j}$, to generate X_j. All of the $\boldsymbol{a_j}$ vectors, once learned, describe the learned causal structure. Somewhat similar to SAM, Ke et al. [2019] also used boolean masks applied to inputs in an ensemble of neural networks to model the causal relationship between observed variables and the observed variable's parents. Lopez-Paz et al. [2017] used supervised CNNs to predict the direction of the causal arrow between two variables in static images and Harradon et al. [2018] built a Bayesian causal model from lower dimensional representations (of images) learned by an autoencoder CNN.

Experimenting with different architectures is a common theme in deep learning research. Various versions of fully-connected neural networks (i.e., each node is connected to every other node in the network) have been studied since 1990 [Fahlman and Lebiere, 1990]. The dense convolutional network (DenseNet) [Huang et al., 2017] is a fully-connected neural network that concatenates all previous feature-maps (i.e., hidden layer outputs) into a single tensor and use this tensor as input to the next hidden layer in the network. Highway networks [Srivastava et al., 2015] and deep residual networks (ResNets) [He et al., 2016] are deep learning models with "skip" connections, allowing the output of a previous hidden layer to directly influence the output of a future hidden layer, after skipping one or more hidden layers. The RINN is perhaps most closely related to a recurrent neural network (RNN). The RINN model can be thought of as an RNN with non-shared weights, time-invariant (i.e., static) input, and output generated only at the last time-step [Liao and Poggio, 2016].

3 REDUNDANT INPUT NEURAL NETWORK (RINN)

In this study, we desired to model inputs that directly connect to any hidden layer in a neural network. To accomplish this, we developed the redundant input neural network (RINN) (Figure 1). The RINN has an extra copy of the input $\boldsymbol{x}$ directly connected to all hidden layers after the first hidden layer. In contrast to the RINN, the inputs of a conventional feedforward DNN are only connected to the first hidden layer. Just like a DNN, the RINN is trained through backpropagation and stochastic gradient descent. Each hidden layer of a RINN with redundant inputs is calculated according to

$$\boldsymbol{h}^{(i)} = \phi([\boldsymbol{h}^{(i-1)}, \boldsymbol{x}] \cdot \boldsymbol{W}_i) + \boldsymbol{b_i}$$

where $\boldsymbol{h}^{(i-1)}$ represents the previous layer's output vector, $\boldsymbol{x}$ is the vector input to the neural network, $[\boldsymbol{h}^{(i-1)}, \boldsymbol{x}]$ represents concatenation into a single vector, $\boldsymbol{W}_i$ represent the weight matrix between hidden and redundant nodes in layer $i-1$ and hidden layer i, ϕ is a nonlinear function (e.g., ReLU), $\cdot$ represents vector-matrix multiplication, and $\boldsymbol{b_i}$ represents the bias vector for layer i. In contrast to a RINN, a plain DNN calculates each hidden layer as $\boldsymbol{h}^{(i)} = \phi(\boldsymbol{h}^{(i-1)} \cdot \boldsymbol{W}_i) + \boldsymbol{b_i}$.

In order to use a neural network to capture the ground truth causal structure in the weights, the weights needed to be regularized or constrained in some way. We evaluated both L_1 and L_2 regularization and L_1 provided far superior results for finding the causal structure of our ground truth DAG (results not shown). The objective

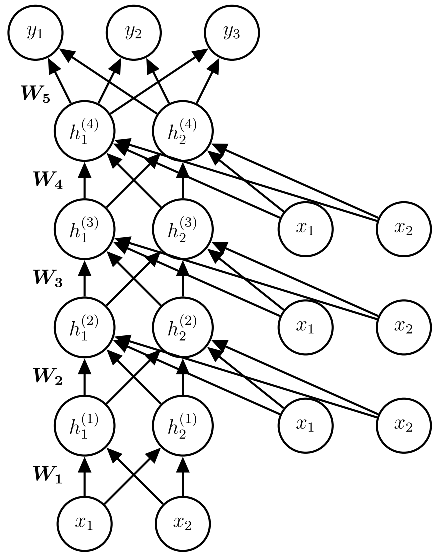
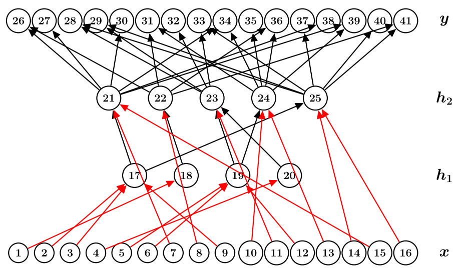


Figure 1: Redundant Input Neural Network (RINN)
A RINN with four hidden layers ($h^{(1)}$, $h^{(2)}$, $h^{(3)}$, $h^{(4)}$, each with two nodes), two inputs (x_1, x_2 on bottom), three outputs (y_1, y_2, y_3), and three sets of redundant inputs (x_1, x_2 on right side). Each node represents a scalar value and each edge represents a scalar weight. The weights between layers are collected in weight matrices $\boldsymbol{W_1}$, $\boldsymbol{W_2}$, $\boldsymbol{W_3}$, $\boldsymbol{W_4}$, and $\boldsymbol{W_5}$.

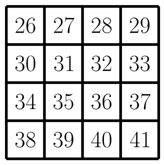
functions for all neural network-based strategies used in this study included L_1 regularization of the weights.

4 SIMULATING DATA FROM A KNOWN CAUSAL STRUCTURE

Ultimately, we are interested in biological cellular signaling systems and constructing these causal signaling pathways from mutation and expression data. However, accurate and complete ground truth cellular signaling pathway knowledge (i.e., causal structure) is not available. To evaluate how well a neural network approach can discover causal structure through its weights, we generated simulated data from an artificial causal hierarchical structure. We first manually created a ground truth directed acyclic graph (DAG), G_T, that fit the requirements described above (i.e., hierarchical structure between a set of inputs and outputs). In G_T (Figure 2a), the input layer is directly connected to both the first and second hidden layers to allow us to recover direct causal relationships between inputs and hidden variables. This structure is also the most biologically plausible way that mutations would affect biological entities in a signaling pathway (assuming the latent variables to be biological entities), as mutations can have an effect at any location in the signaling pathway hierarchy, not just at one level (i.e., hidden layer) of the hierarchy. The ground truth DAG

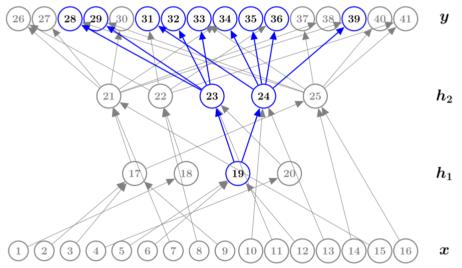


(a) Ground truth DAG

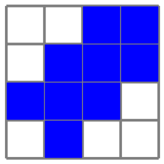


26	27	28	29
30	31	32	33
34	35	36	37
38	39	40	41

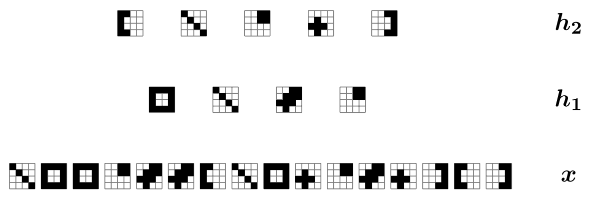
(b) $\boldsymbol{y}$ reshaped from a vector to a matrix



(c) Effect of node 19 in output space



(d) Node 19 mapped to output space. This heatmap is an alternative representation of the the causal structure in (c) from node 19 forward.



(e) Effects of all input and hidden nodes in output space. These are the ground truth patterns to look for in the weights of a neural network. Note that there are seven unique patterns, two of these seven are combinations of other patterns.

Figure 2: Ground Truth DAG for Simulated Data and Visualizing Output Space

had 16 inputs $\boldsymbol{x}$, 16 outputs $\boldsymbol{y}$, and two hidden layers $\boldsymbol{h_1}$, $\boldsymbol{h_2}$ (Figure 2a). We needed an easy way to visualize G_T, so we connected the nodes in a specific way to be able to generate visibly recognizable patterns (relative to output space) for each node in G_T (Figure 2 and Section 8). Given a specific node x_i in G_T, these patterns (heatmaps) indicate if an output node is affected by the value of x_i, i.e., Which output nodes have x_i as an ancestor? (Figure 2c, 2d).

We constructed G_T with inputs $\boldsymbol{x}$ representing the presence or absence of DNA alterations (i.e., SGAs) and the outputs $\boldsymbol{y}$ representing differentially expressed genes (DEGs). Mutation and expression data are readily available from many sources, e.g., The Cancer Genome Atlas (TCGA) Weinstein et al. [2013]. Between these input and output layers, we envisioned a biological signaling pathway with the hidden layer $\boldsymbol{h}_1$ closest to the inputs representing major biological pathways and $\boldsymbol{h}_2$ representing transcription factors. Biologically, mutations (i.e., input nodes $\boldsymbol{x}$) directly target transcription factors and proteins higher up in the hierarchy (e.g., cell membrane receptors). To reflect this in G_T, we allowed input mutations in $\boldsymbol{x}$ to directly connect to nodes in $\boldsymbol{h}_1$ or $\boldsymbol{h}_2$. Each mutation affected only one node, but each node could be affected by multiple mutations. This is similar to how biological mutations function. G_T is a crude representation of a biological signaling pathway, and one obvious omission in the DAG abstraction are feedback cycles, which are prevalent in biological signaling pathways. We generated six different simulated datasets. Datasets 1-4 had 16 inputs, 16 outputs, and 5,000 instances. More information on the simulated datasets can be found in the Appendix.

4.1 SIMULATED DATA

1. **Matrix Multiplication with Interventions**—*no noise, binary input, positive integer-valued output.* We simulated data from G_T by having the inputs *inter-*

vene on their adjacent hidden nodes. An "active" (i.e., 1) input intervention node would set the value of an adjacent hidden node to 0 for that sample—intended to capture the effect of a biological loss-of-function mutation on a protein.

2. **Linear Gaussian SEM**—*high noise, input and output* $\in \mathbb{R}$. See Appendix.

$$N_j^l = \sum_{i \in parents_{G_T}(j^l)} W_{ij}^l X_i^{l-1} + b_j^l$$

3. **OR Logical Operator**—*high noise, binary*. We modeled the value of a node in G_T as the output of a logical OR applied to the values of the parents of said node. We sampled a probability, p, to represent

$$P(X = 1 | parents_{G_T}(X))$$

for all possible binary combinations of a node's parents' values. p was sampled from Beta distributions peaked close to 0.10 or 0.90 depending on the values of the parents.

4. **AND, OR, XOR Logical Operators**—*moderate noise, binary*. We added AND and XOR operators to replace some of the OR operators in Dataset 3.

5. **TCGA + OR**. Biological datasets with mutation and expression data, and a robust, known ground truth causal structure, to our knowledge, do not exist. So we embedded simulated data (Dataset 3—OR) within a larger biological dataset from TCGA to see if G_T could be discovered. Data dimensions: 650 inputs (SGAs) by 84 outputs (DEGs) for 5,000 instances.

6. **TCGA + OR, XOR, AND**. The same as Dataset 5, but embedded with Dataset 4 (instead of Dataset 3).

5 OTHER MODELS

5.1 DNN

We compared the RINN with a feedforward DNN. A DNN learns a function mapping inputs ($\boldsymbol{x}$) to outputs ($\boldsymbol{y}$) according to

$$f(\boldsymbol{x}) = \phi(...\phi(\phi(\boldsymbol{x}\boldsymbol{W}_1)\boldsymbol{W}_2)...\boldsymbol{W}_n) = \hat{\boldsymbol{y}}$$

where $\boldsymbol{W}_i$ represent the weight matrices between layers of a neural network, ϕ is a nonlinear function (e.g., sigmoid), and $\hat{\boldsymbol{y}}$ is the predicted output. DNNs also have bias vectors added at each layer, which have been omitted from above for clarity.

5.2 DBN

A deep belief network or DBN [Hinton and Salakhutdinov, 2006] is a type of autoencoder, an unsupervised DNN. The DBNs used in this paper were trained using only $\boldsymbol{y}$ data, i.e., DBNs learned to map $\boldsymbol{y}$ to a lower dimensional representation of $\boldsymbol{y}$ (in the hidden layers) and then to reconstruct $\boldsymbol{y}$ from the lower dimensional representation. Error is measured by comparing $\boldsymbol{y}$ to $\hat{\boldsymbol{y}}$. We evaluated the trained "decoder" network only (i.e., the second half of the network going from lowest dimensional representation to $\boldsymbol{y}$) for evidence of G_T latent variables. We had difficulty recovering any of the causal ground truth graph (Figure 2a) with a default DBN, but adding L_1 regularization to the finetuning step objective function allowed us to recover at least some of the causal structure.

5.3 Constrained Evolutionary Strategy (ES-C)

We compared the conventional RINN (using gradient descent optimization) to a RINN using an evolutionary algorithm for weight optimization. We developed an evolutionary strategy (ES) to optimize the weights of a RINN to explore optimization strategies that make it easier to interpret the weights of a neural network as causal relationships. Specifically, we used a constrained evolutionary strategy, where we constrained the possible values for the weights (e.g., $weight \in \{-1, -0.5, 0, 0.5, 1\}$) (see Appendix). We represented an individual in the population to be evolved as a vector of weights. This vector of weights represents a RINN. Initially these weights were set to all zeros, then if a weight was selected for mutation (as dependent on a mutation rate), it was randomly changed to a weight in the range of legal weight values. For a fitness function, we used L_1 regularization plus either binary cross-entropy error or squared error depending on the dataset. ES-C was not set up to be parallelized across processors, as early prototyping experiments suggested that ES-C would not perform nearly as well as RINN with gradient descent optimization.

5.4 DM Algorithm

As another comparison, we evaluated the performance of the DM (Detect MIMIC) algorithm [Murray-Watters and Glymour, 2015] on the first four simulated datasets. The DM algorithm takes two tuning parameters: an alpha level for Fisher's Z test of conditional independence and an alpha level for Sober's criterion. We performed an abbreviated grid search over these tuning parameters.

6 CAUSAL ASSUMPTIONS

The following are assumed to be true regarding the true graph and data generating process. These assumptions allow the weights learned by a neural network to be interpreted as causal relationships.

A1. Causal Markov Assumption: D-separation (a criterion for determining independence between vertices in a causal graph) in the true data-generating graph G_T implies conditional independence in the data distribution.

A2. $\boldsymbol{x} \rightarrow\rightarrow \boldsymbol{y}$: We assume it is known *a priori* that $\boldsymbol{x}$ causes $\boldsymbol{y}$ and that there is at least one latent variable on the path from $\boldsymbol{x}$ to $\boldsymbol{y}$. For example, mutations in DNA cause changes in gene expression through modifications to cell signaling. The methods developed in this study can be used with any datasets where the causal direction between two sets of variables is known.

A3. Causal Sufficiency: Given $\boldsymbol{x}, \boldsymbol{y}$, the observed input and output variables (i.e., all observed variables), there are no latent variables Z, $Z \notin \boldsymbol{x}$, $Z \notin \boldsymbol{y}$, such that Z causally influences at least one variable in $\boldsymbol{x}$ and one variable in $\boldsymbol{y}$ (i.e., latent confounding). In a biological context, with $\boldsymbol{x} = \boldsymbol{SGA}$ and $\boldsymbol{y} = \boldsymbol{DEG}$, this assumption is generally true.

A4. We assume away selection bias.

A5. The true graph has no latent to latent cycles (acyclic): In a biological context, this assumption is violated by many signaling pathways as feedback is a common characteristic of molecular signaling pathways. Assuming acyclicity is a limitation of using the methods in this study.

A6. Faithfulness: Independence in the data distribution implies d-separation in the true data-generating graph G_T. This can be seen as an appeal to Occam's razor—preferring the most parsimonious graph over a denser graph. L_1 regularization encourages learning the most parsimonious graph.

A7. Latent Variable Identifiability: The latent variables are *identifiable*.

A latent variable is *identifiable* if, 1. the intersection of its ancestors in G_T and $\boldsymbol{x}$ is unique with respect to the other latent variables, *AND* 2. the intersection of its descendants in G_T and $\boldsymbol{y}$ is unique with respect to the other latent variables.

A8. The ground truth causal relationships can be modeled with one or multiple affine transformations plus nonlinear functions (hidden layer calculations in a neural network).

The causal ground truth DAG in Figure 2a and simulated data adhere to these assumptions. The *structure* learned by the RINN represents the underlying data generating DAG if the RINN reaches the global optimum and the assumptions above are not violated. This means that the weights of a RINN can be interpreted as edges in a causal graph, with all edges oriented in the direction that is towards $\boldsymbol{y}$, i.e., $x \rightarrow h_1 \rightarrow h_2 \rightarrow, ..., h_n \rightarrow y$, where n is the number of hidden layers. We suspect that reaching the global optimum is not necessarily required, and correct causal structure (but not necessarily correct causal parameters) could still be recovered from models close to the global optimum.

7 BALANCING SPARSITY AND ERROR

In order to find simple and accurate models, we needed to modify our approach to model selection in order to best balance sparsity of weights and prediction error. After training a model with various combinations of hyperparameters, we selected the models (sets of weight matrices for each trained network) that were relatively sparse and had a relatively low prediction error. Under-regularizing leads to overfitting the data, while over-regularizing leads to underfitting the data. To balance the trade-off between prediction error and sparsity, in plots of prediction error versus sparsity (See Appendix Figure 4), we measured the Euclidean distance from each point (model) to the origin according to: $d_x = \sqrt{\left(\sum_{i=1}^{m}\sum_{j=1}^{r_i}\sum_{k=1}^{c_i}|w_{j,k}^{(i)}|\right)^2 + L^2}$

where L is the validation set loss for neural network x, m is the number of weight matrices in neural network x, r_i and c_i are the number of rows and columns in matrix i respectively, and w is a scalar weight. Prior to calculating d_x, we removed values outside the 95% quantile for each axis and scaled the axes using min-max scaling.

8 VISUALIZING A NEURAL NET

To increase our understanding of what the weights of a neural network capture after training, we developed a simple visualization technique according to

$$\boldsymbol{M}_j = \boldsymbol{W}_j\boldsymbol{W}_{j+1}\boldsymbol{W}_{j+2}, ..., \boldsymbol{W}_h$$

$$\boldsymbol{v}_j = reshape(\boldsymbol{M}_j)$$

where $\boldsymbol{M}_j$ is an $m \times n$ matrix for layer j, m is the number of nodes in layer j, n is the number of nodes in the output $\boldsymbol{y}$, $\boldsymbol{W}_j$ is the weight matrix between layer j and $j+1$, and $\boldsymbol{W}_h$ is the last weight matrix. Each row in $\boldsymbol{M}_j$ represents what a single node in layer j maps to in output space. $reshape$ means to reshape each row of $\boldsymbol{M}_j$ to a 4×4 matrix (assuming $n = 16$) (Figure 2b). Therefore, $\boldsymbol{v}_j$ is a list of 4×4 matrices (heatmaps) for all hidden nodes in layer j. Next, the magnitude of each value in the 4×4 matrices is interpreted as a pixel intensity (or heatmap intensity), and displayed as a given intensity of color (Figure 2d). In this way, we generated 2-D heatmaps for each hidden and input node in a trained neural network. This visualization represents the weights connecting a node to output space, or the influence of the activation of that node on the output values (Figure 2e). The heatmap visualizations of all nodes' effects in output space is a representation of the causal structure learned by a neural network.

9 EXPERIMENTS

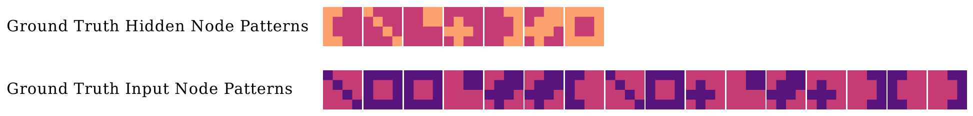
In addition to evaluating the different deep learning strategies and distance metric (d_x) for finding causal structure, we also needed a method to quickly evaluate many networks without relying on visual inspection. The solution to this problem came from our method to visualize the weights of a neural network. Figure 3 shows visualizations of the weights for two different DNNs. Each square in these visualizations represents a specific node in a neural network and the change in the output values induced by the activation of this node (Section 8). Figure 3a shows the ground truth patterns/visualizations based on the ground truth DAG (Figure 2a). These are the patterns we are looking for in the weights of a network to indicate the correct causal structure. The color of these heatmaps is of minimal importance, while the pattern in the heatmaps is of maximal importance.

Figure 3b shows a visualization of the weights of a 3-hidden layer DNN trained on Dataset 1. The visualizations labeled as $\boldsymbol{x}$ show what each of the input nodes for this DNN map to in output space. If we compare this set of images to the ground truth input node patterns in Figure 3a (bottom), we see that the input nodes in the DNN 100% match with the ground truth input node patterns. This example immediately suggests a possible method for evaluating how well neural networks capture causal structure—by calculating precision and recall based on the ground truth patterns that should be present within these visualizations for the causal structure to be correct.

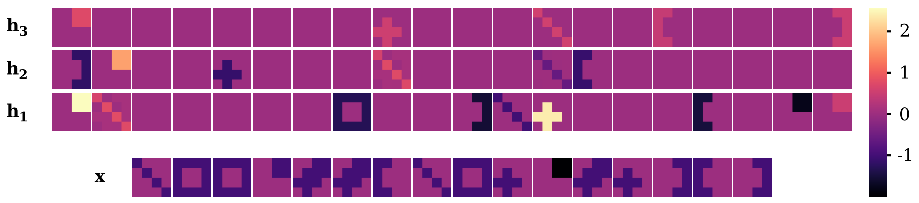
To calculate precision and recall of the causal structure, we ran the visualization algorithm on all layers in our trained RINN, DNN, DBN, or ES-C, obtaining a vector with 16 values for each node in the neural network, representing the influence of that node on output space. We took the absolute value and then binarized these vectors in order to match them to the ground truth, thus being able to calculate precision and recall. This required setting a threshold, above which a value would be set to 1 and otherwise set to 0. We chose this threshold by searching through our best models for a single threshold that provided the best F_1 score. The same threshold was used for all models. We obtained precision and recall measurements for each neural network by comparing these binary vectors to ground truth binary vectors. To calculate the ground truth vectors, we used the same algorithm as in Section 8 applied to the ground truth binary weight matrices. True positives (TP) here represent ground truth hidden nodes or input nodes (remember, a node is represented as a vector of numbers in output space) that were captured by the weights of the trained network. False positives (FP) represent predicted vectors from the trained network that did not match any of the ground truth nodes or the zero vector. False negatives (FN) represent ground truth hidden nodes or input nodes that were not captured by the weights of the trained network. When comparing a predicted output space vector to a ground truth vector, we considered the predicted to match the ground truth if the binary vectors were identical within ± 1 value (i.e., pixel). The network in Figure 3b achieved 100% *input node* precision and recall.

After examining the vector of images labeled as $\boldsymbol{h}_1$, in Figure 3b, we see that this hidden layer by itself captures six of the seven ground truth hidden node patterns from Figure 3a ($\boldsymbol{h_2}$ and $\boldsymbol{h_3}$ do not find any additional ground truth hidden node patterns). The calculations for *hidden node* precision and recall for this DNN would be as follows: $Precision(P) = \frac{TP}{TP+FP} = \frac{6}{6+0} = 1.00$, $Recall(R) = \frac{TP}{TP+FN} = \frac{6}{6+1} = 0.86$. So despite regularization encouraging the DNN to represent patterns as simply as possible, this DNN did not capture the ground truth hidden node that is a combination of the "plus sign" and "little square" (second from the right in Figure 3a top). Nodes 17 and 19 in Figure 2a and 2e are referred to as "combination" nodes, as they represent a combination of two other nodes. Figure 3c shows a visualization of a different DNN that did not capture the causal structure as nicely as Figure 3b. The input nodes map 100% correctly, but there are at least three FPs and only four TPs found within the hidden layer nodes. $P = \frac{4}{4+3} = 0.57$, $R = \frac{4}{4+3} = 0.57$. For RINN visualizations, see the Appendix.

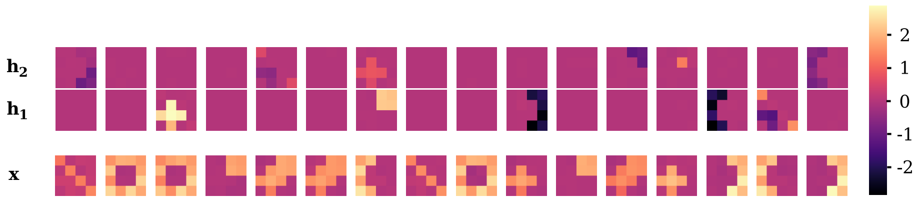
The first hidden layer of a DNN must capture all the hidden node patterns required to completely map inputs to outputs, as it does not have another chance to capture information it may have missed in the first hidden layer



(a) Ground truth patterns used when calculating precision and recall.



(b) A DNN (trained on Dataset 1) that finds all input patterns and finds all hidden node patterns except one (i.e., one false negative).



(c) A different DNN (trained on Dataset 3) that finds all input patterns, but has some difficulty with the hidden patterns (3 false positives*, 4 true positives, and 3 false negatives). *False positives are nodes that don't match any ground truth ±1 pixel and repeated false positives are only counted once.

Figure 3: DNN Weight Visualizations

later in the network. The hidden layer representations of a DNN are alternative representations of all the information in the input necessary to calculate the output. This is in contrast to a RINN, which can simply learn the patterns at a later hidden layer because a RINN has redundant inputs. So the hidden layers of a RINN are not necessarily alternative representations of the input, but they capture salient aspects of the input data in some way desirable to calculate the output.

The major purpose of this work was to compare the ability of different neural network-based strategies to find causal structure in their weights. We were also interested in the limitations of these strategies depending on the characteristics of the data. Table 1 shows the *input* node precision and recall results across all datasets and strategies used in this study. Table 1 also shows the average test set error and AUROC (where appropriate) for all strategies and datasets. This table is best thought of as showing how well the input mapped to output without indicating the exact causal structure in the weights, i.e., this metric does not indicate how the latent variables interacted to achieve the output. The precision and recall (as well as error and AUROC) were averaged across the ten best models (as ranked by our distance metric) for each strategy and dataset for Tables 1 and 2.

Table 1 shows that both DNN and RINN performed quite well at mapping input to output across all datasets, with the RINN performing a little better in terms of precision and recall. The RINN performed noticeably better than the DNN on the Linear Gaussian and TCGA+OR datasets. The test errors and AUROCs for DNN and RINN were almost identical. This means that if one captures more causal structure than the other, it isn't due to one strategy being able to predict better than the other, but rather due to some other difference. ES-C performed similar to DNN and RINN on the first three datasets, but markedly worse on the OR, XOR, AND dataset in terms of precision and recall. ES-C had errors and AUROCs that were similar to DNN and RINN (and on Dataset 4 were better than RINN and DNN). The DM algorithm, despite getting 100% recall, performed very poorly on all datasets. This is because the DM algorithm isn't identifying any false-negatives, just a lot of false-positives and maybe 1 or 2 true-positives.

The results in Table 2 are the main results of this work and show how well the ground truth causal structure was captured by each of the strategies for each dataset. The

Table 1: Average Input Node Precision and Recall Across Ten Best Models

		Input Mapping			Test Set	
Data	Model	precision	recall	F_1	error	AUROC
1	DBN	NA	NA	NA	0.05 ± 0.1	NA
	DNN	$\mathbf{1.00} \pm 0.0$	$\mathbf{1.00} \pm 0.0$	$\mathbf{1.00} \pm 0.0$	0.01 ± 0.0	NA
	RINN	$\mathbf{1.00} \pm 0.0$	$\mathbf{1.00} \pm 0.0$	$\mathbf{1.00} \pm 0.0$	$\mathbf{0.00} \pm 0.0$	NA
	ES-C	0.95 ± 0.1	$\mathbf{1.00} \pm 0.0$	0.97 ± 0.0	0.03 ± 0.0	NA
	DM	0.31	**1.00**	0.48	NA	NA
2	DBN	NA	NA	NA	$\mathbf{1.82} \pm 0.1$	NA
	DNN	0.94 ± 0.1	0.93 ± 0.0	0.93 ± 0.1	9.31 ± 0.2	NA
	RINN	$\mathbf{0.98} \pm 0.0$	$\mathbf{1.00} \pm 0.0$	$\mathbf{0.99} \pm 0.0$	9.19 ± 0.1	NA
	ES-C	0.91 ± 0.1	0.91 ± 0.0	0.90 ± 0.1	10.12 ± 0.2	NA
	DM	0.69	**1.00**	0.81	NA	NA
3	DBN	NA	NA	NA	$\mathbf{0.55} \pm 0.2$	0.68 ± 0.2
	DNN	$\mathbf{0.99} \pm 0.0$	0.98 ± 0.1	$\mathbf{0.98} \pm 0.0$	0.56 ± 0.0	$\mathbf{0.75} \pm 0.0$
	RINN	0.96 ± 0.1	$\mathbf{1.00} \pm 0.0$	$\mathbf{0.98} \pm 0.0$	0.56 ± 0.0	$\mathbf{0.75} \pm 0.0$
	ES-C	0.81 ± 0.2	0.95 ± 0.1	0.87 ± 0.1	0.57 ± 0.0	0.74 ± 0.0
	DM	0.56	**1.00**	0.72	NA	NA
4	DBN	NA	NA	NA	0.58 ± 0.1	0.57 ± 0.1
	DNN	0.78 ± 0.2	0.99 ± 0.0	0.87 ± 0.1	0.45 ± 0.0	0.84 ± 0.0
	RINN	$\mathbf{0.92} \pm 0.1$	0.91 ± 0.1	$\mathbf{0.91} \pm 0.1$	0.45 ± 0.0	0.84 ± 0.0
	ES-C	0.46 ± 0.2	0.65 ± 0.2	0.53 ± 0.2	$\mathbf{0.41} \pm 0.0$	$\mathbf{0.87} \pm 0.0$
	DM	0.44	**1.00**	0.61	NA	NA
5	DBN	NA	NA	NA	$\mathbf{0.34} \pm 0.2$	$\mathbf{0.88} \pm 0.1$
	DNN	0.90 ± 0.1	0.86 ± 0.2	0.88 ± 0.1	0.42 ± 0.0	0.86 ± 0.0
	RINN	$\mathbf{0.95} \pm 0.1$	$\mathbf{0.98} \pm 0.0$	$\mathbf{0.96} \pm 0.1$	0.40 ± 0.0	0.87 ± 0.0
6	DBN	NA	NA	NA	0.53 ± 0.2	0.64 ± 0.2
	DNN	$\mathbf{0.79} \pm 0.3$	0.90 ± 0.3	0.84 ± 0.3	$\mathbf{0.42} \pm 0.0$	$\mathbf{0.86} \pm 0.0$
	RINN	0.78 ± 0.1	$\mathbf{0.98} \pm 0.0$	$\mathbf{0.87} \pm 0.0$	0.43 ± 0.0	$\mathbf{0.86} \pm 0.0$

RINN had a higher F_1 score than all other strategies for most datasets, and consistently had the highest recall across all datasets and strategies. RINN and DNN had F_1 scores within 0.01 of each other on datasets 4 and 6. The RINN performed considerably better than any other strategy on the Linear Gaussian dataset. The Linear Gaussian dataset was also the dataset with the most noise, so it is encouraging that the RINN was able to considerably outperform all other strategies on this noisy dataset, as we suspect that most biological data has a high amount of noise. There didn't seem to be much of a difference between continuous and binary datasets—the strategies were still able to learn causal structure from completely binary inputs and outputs (Datasets 3 and 4). Rather, the most important characteristic of the datasets seemed to be the amount of noise. The RINN outperformed all other strategies on the TCGA+OR dataset and performed similarity to a DNN on the TCGA+OR, XOR, AND dataset. These datasets had inputs and outputs of much higher dimensionality than any of the other simulated datasets, and shows the RINNs utility even in a higher dimensionality setting.

ES-C performed moderately well, but was outperformed by DNN and RINN on all datasets except Linear Gaussian, where ES-C outperformed DNN. This suggests that using a parallelized evolutionary algorithm is a direction worth pursuing for finding latent structure, as ES-C performed well despite relatively less model selection, when compared to RINN, after taking into account that ES-C has more hyperparameters to be tuned than RINN. ES-C was also trained on significantly less data (20%) than the other strategies to decrease the runtime of ES-C even further. ES-C wasn't run on datasets 5 and 6 because of the very long running time it had on the small datasets, meaning that running on the much larger datasets would be impractical. The DM algorithm and DBN performed quite poorly across all datasets, with the DM algorithm performing the worst on most datasets (note that some of the DM algorithms assumptions are violated by G_T). The DM algorithm wasn't run on datasets 5 and 6 because of its poor performance on the smaller datasets. Overall, the RINN was better able to recover the causal structure in Figure 2a than any other strategy explored in this work.

Table 2: Average Hidden Node Precision and Recall Across Ten Best Models.

		Hidden Mapping		
Sim Data	Model	precision	recall	F_1
1 Interv.	DBN	0.73 ± 0.32	0.61 ± 0.22	0.66 ± 0.26
	DNN	0.97 ± 0.07	0.80 ± 0.07	0.87 ± 0.06
	RINN	$\mathbf{1.00} \pm 0.00$	$\mathbf{0.93} \pm 0.08$	$\mathbf{0.96} \pm 0.04$
	ES-C	0.72 ± 0.08	0.80 ± 0.13	0.76 ± 0.08
	DM	0.33	0.29	0.31
2 Lin Gaus	DBN	0.35 ± 0.13	0.57 ± 0.17	0.43 ± 0.14
	DNN	0.34 ± 0.30	0.43 ± 0.24	0.37 ± 0.26
	RINN	$\mathbf{0.65} \pm 0.11$	$\mathbf{0.87} \pm 0.18$	$\mathbf{0.74} \pm 0.13$
	ES-C	0.41 ± 0.10	0.54 ± 0.06	0.46 ± 0.09
	DM	0.33	0.29	0.31
3 or	DBN	0.20 ± 0.30	0.16 ± 0.23	0.17 ± 0.25
	DNN	$\mathbf{0.88} \pm 0.17$	0.76 ± 0.15	0.81 ± 0.14
	RINN	0.87 ± 0.19	$\mathbf{0.83} \pm 0.13$	$\mathbf{0.85} \pm 0.15$
	ES-C	0.71 ± 0.22	0.57 ± 0.18	0.63 ± 0.19
	DM	0.33	0.29	0.31
4 or,xor,and	DBN	0.38 ± 0.22	0.33 ± 0.19	0.33 ± 0.17
	DNN	$\mathbf{0.87} \pm 0.18$	0.79 ± 0.17	$\mathbf{0.82} \pm 0.16$
	RINN	0.81 ± 0.17	$\mathbf{0.83} \pm 0.11$	0.81 ± 0.11
	ES-C	0.44 ± 0.09	0.74 ± 0.16	0.55 ± 0.10
	DM	0.29	0.29	0.29
5 TCGA+3	DBN	0.08 ± 0.17	0.06 ± 0.14	0.08 ± 0.15
	DNN	0.29 ± 0.17	0.59 ± 0.26	0.33 ± 0.12
	RINN	$\mathbf{0.37} \pm 0.25$	$\mathbf{0.63} \pm 0.24$	$\mathbf{0.41} \pm 0.19$
6 TCGA+4	DBN	0.04 ± 0.08	0.16 ± 0.22	0.07 ± 0.12
	DNN	$\mathbf{0.57} \pm 0.26$	0.74 ± 0.16	$\mathbf{0.61} \pm 0.20$
	RINN	0.53 ± 0.24	$\mathbf{0.80} \pm 0.10$	0.60 ± 0.17

10 DISCUSSION

The results presented here show that deep learning models (RINN and DNN with L_1 weight regularization) can learn latent variables that capture, with high accuracy (precision and recall), the compositional structure generated by causal relationships in simulation experiments. In doing so, the models are capable of correctly ordering the hierarchical relationships among latent variables that encode the signal of the data-generating process, i.e., the causal relationships among latent variables. The RINN performed better than the DNN (and all other algorithms) across almost all metrics measured for recovery of the ground truth hidden nodes. By more accurately capturing the statistical structure of the data-generating process, the RINN can better recover the causal relationships among the latent variables. These results indicate that allowing inputs to directly access latent variables encouraged the latent variables to capture salient statistical structure connecting input and output variables. It has not escaped our notice that the RINN architecture simplifies the interpretability of a deep learning model. The architecture of the RINN allows hidden nodes to be more easily mapped to a set of input variables than a DNN, as inputs in a RINN are directly connected to each hidden node. A set of input variables can then be used to represent a hidden node, and thus the hidden nodes become partially interpretable. That is, one can interpret that a latent variable encodes the signal of specific input variables. These results illustrate two main characteristics (advances) brought forward by the RINN model: the capability of learning a partially interpretable deep learning model and the capability of learning causal relationships.

We conjecture that the capability of the RINN model to learn causal structure lies in its capability of mimicking the data-generating process. Here we employed a set of hierarchically organized latent variables to mimic the data-generating process. We hypothesized that the most efficient way for a model to accurately capture the relationships between input and output is to force latent variables to mimic the data-generating process, and we applied L_1 penalization (i.e., following Occam's Razor or the minimal description length principle [Rissanen, 1978]) to maximize the amount of information encoded by each latent variable with respect to both input and output variables. Interestingly, both RINN and DNN were capable of learning the data-generating process. Furthermore, if the data-generating process involves a series of causal relationships that lead to compositional statistical structure, the model can naturally capture such information. Hence, we hypothesize that if a deep learning model is initially set up with an architecture that is sufficient to enclose the causal structures of the data generating process, then one can employ a parameter-searching approach to learn the causal structure among latent variables, in contrast to an explicit search in structure space. In terms of complexity, searching through the parameter space of a deep learning model can be done with polynomial complexity, whereas searching through the structure space of a set of variables is super-exponential, and greedy algorithms are often used to reduce the complexity.

There are some limitations relevant to this work. When interpreting the weights of a RINN (or DNN) as a causal graph, it is necessary to look at the values of all the weights in a trained network and, based on a threshold, decide which weights are significant (i.e., edges in a graph) and which are not. Using L_1 regularization causes the network to set most of the values in the weight matrices to small numbers, but in our experiments very few of the weights are actually zero—rather there are many weights with values in the interval $[0.0001, 0.5]$. This thresholding problem can be improved by using non-differentiable optimization procedures that constrain the allowed values for weights in a neural network (e.g., ES-C). Another limitation is that calculating precision and recall did not explicitly take into account the ordering of the nodes. Anecdotally, after looking through hundreds of trained networks, we rarely observed incorrect ordering of causal structure. Using L_1 regularization encourages the network to learn nodes in the correct order and hierarchy, as learning G_T represents the weight structure with the lowest number of edges.

The evolutionary strategy explored here provided promising results despite limited model selection and data, and more sophisticated versions are worth exploring in future experiments. Modifying the RINN by using differential regularization and faster evolutionary algorithms, as well as extending the causal framework developed here could lead to improvements in the interpretability of deep neural networks and in the use of deep learning models for the discovery of latent causal structure.

Acknowledgements

Funding for this work provided by NIH grant R01 LM012011.

References

C. Cai, G. Cooper, K. Lu, X. Ma, S. Xu, Z. Zhao, X. Chen, Y. Xue, A. Lee, N. Clark, V. Chen, S. Lu, L. Chen, L. Yu, H. Hochheiser, X. Jiang, Q. Wang, and X. Lu. Systematic discovery of the functional impact of somatic genome alterations in individual tumors through tumor-specific causal inference. *PLoS Computational Biology*, 15(7):e1007088, 2019.

L. Chen, C. Cai, V. Chen, and X. Lu. Learning a hierarchical representation of the yeast transcriptomic machinery using an autoencoder model. *BMC Bioinformatics*, 17(1):S9, 2016.

D. Colombo, M. H. Maathuis, M. Kalisch, and T. S. Richardson. Learning high-dimensional directed acyclic graphs with latent and selection variables. *The Annals of Statistics*, pages 294–321, 2012.

G. Cooper. An overview of the representation and discovery of causal relationships using bayesian networks in causation, prediction, and search. In *Computation, Causation, and Discovery*, pages 3–62. Menlo Park, Calif. : AAAI Press ; Cambridge, Mass. : MIT Press, 1999.

G. Elidan and N. Friedman. Learning hidden variable networks: The information bottleneck approach. *Journal of Machine Learning Research*, 6(Jan):81–127, 2005.

S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, pages 524–532, 1990.

N. Friedman et al. Learning belief networks in the presence of missing values and hidden variables. In *International Conference on Machine Learning*, volume 97, pages 125–133, 1997.

M. Harradon, J. Druce, and B. Ruttenberg. Causal learning and explanation of deep neural networks via autoencoded activations. *arXiv preprint arXiv:1802.00541*, 2018.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, pages 770–778, 2016.

C. Heinze-Deml, M. H. Maathuis, and N. Meinshausen. Causal structure learning. *Annual Review of Statistics and Its Application*, 5:371–391, 2018.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

D. Kalainathan, O. Goudet, I. Guyon, D. Lopez-Paz, and M. Sebag. Sam: Structural agnostic model, causal discovery and penalized adversarial learning. *arXiv preprint arXiv:1803.04929*, 2018.

N. R. Ke, O. Bilaniuk, A. Goyal, S. Bauer, H. Larochelle, C. Pal, and Y. Bengio. Learning neural causal models from unknown interventions. *arXiv preprint arXiv:1910.01075*, 2019.

V. Lagani, S. Triantafillou, G. Ball, J. Tegner, and I. Tsamardinos. Probabilistic computational causal discovery for systems biology. In *Uncertainty in Biology*, pages 33–73. Springer, 2016.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems*, pages 873–880, 2008.

Q. Liao and T. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

D. Lopez-Paz, R. Nishihara, S. Chintala, B. Schölkopf, and L. Bottou. Discovering causal signals in images. In *Computer Vision and Pattern Recognition*, 2017.

S. Lu, X. Fan, L. Chen, and X. Lu. A novel method of using deep belief networks and genetic perturbation data to search for yeast signaling pathways. *PloS One*, 13(9):e0203871, 2018.

A. Murray-Watters and C. Glymour. What is going on inside the arrows? discovering the hidden springs in causal models. *Philosophy of Science*, 82(4):556–586, 2015.

J. Peters, D. Janzing, and B. Schölkopf. *Elements of causal inference: foundations and learning algorithms*. MIT Press, 2017.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

E. Sober. Black box inference: When should intervening variables be postulated? *The British Journal for the Philosophy of Science*, 49(3):469–498, 1998.

P. Spirtes, C. Meek, and T. Richardson. Causal inference in the presence of latent variables and selection bias. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 499–506. Morgan Kaufmann Publishers Inc., 1995.

P. Spirtes, C. N. Glymour, and R. Scheines. *Causation, prediction, and search*. MIT press Cambridge, 2000.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2377–2385, 2015.

Y. Tao, C. Cai, W. W. Cohen, and X. Lu. From genome to phenome: Predicting multiple cancer phenotypes based on somatic genomic alterations via the genomic impact transformer. In *Pacific Symposium on Biocomputing*, volume 25, pages 79–90, 2020.

J. N. Weinstein, E. A. Collisson, G. B. Mills, K. R. M. Shaw, B. A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, J. M. Stuart, C. G. A. R. Network, et al. The cancer genome atlas pan-cancer analysis project. *Nature Genetics*, 45(10):1113, 2013.

J. D. Young, C. Cai, and X. Lu. Unsupervised deep learning reveals prognostically relevant subtypes of glioblastoma. *BMC Bioinformatics*, 18(11):381, 2017.

Appendix

A Simulated Data

A.1 Dataset 2: Linear Gaussian SEM (high noise, continuous)

$$\boldsymbol{b} \sim \mathcal{N}(0, \sigma^2) \quad \sigma^2 \sim \mathcal{U}(1, 3) \quad \boldsymbol{W} \sim \pm\mathcal{U}(0.5, 1.5)$$

Let G_T be our ground truth DAG with nodes $\boldsymbol{N}$ (input, hidden, and output), N_j^l be node j at layer l, X_i^{l-1} be the value of parent (i.e., direct cause) i in layer $l-1$. Then we simulate data using a structural equation model (SEM) to calculate the value of each node N_j^l in G_T as

$$N_j^l = \sum_{i \in parents_{G_T}(j^l)} W_{ij}^l X_i^{l-1} + b_j^l$$

A.2 Dataset 3: OR Logical Operator (high noise, binary)

We sampled p from Beta distributions peaked close to 0.10 or 0.90 depending on the values of the parents (i.e., OR operator applied to the values of the parents) to add noise to the OR operations. This sampling from Beta distributions provided a collection of Bernoulli distributions for all possible binary combinations of parents. We then used these Bernoulli distributions to generate noisy data from OR operators (see Pseudocode below, but assume all OR operators).

A.3 Dataset 4: AND, OR, XOR Logical Operators (noisy, binary)

To increase complexity, we added AND and XOR operators to replace some of the OR operators in Dataset 3 and followed the same procedure as for Dataset 3 (see Pseudocode below). In contrast to Dataset 3, we sampled p from Beta distributions peaked close to 0.05 and 0.95 for Dataset 4, meaning that Dataset 4 is less noisy than Dataset 3.

A.4 Dataset 5: TCGA + OR

We wanted to test our algorithms on larger, biologically-related datasets. However, biological datasets with mutation and expression data, and a robust, known ground truth causal structure, to our knowledge, do not exist. So our solution was to embed simulated data within a larger biological dataset and see if G_T could be discovered within the much larger dataset, and to also evaluate how the different algorithms performed simply on the prediction task for this biologically related dataset. To accomplish this, we obtained mutation and expression data from TCGA and used an algorithm from [Cai et al., 2019] called TCI (tumor-specific causal inference) to perform feature selection on both the mutation and expression data, ultimately leading to a final dataset with 634 alterations (inputs) and 68 DEGs (outputs) for 5,000 tissue samples. Next, we embedded simulated Dataset 3 (OR operator) into the feature-reduced TCGA dataset described above. This generated a final dataset with 650 inputs and 84 outputs for 5,000 samples. Please see [Cai et al., 2019] for a detailed explanation of how TCI works.

A.5 Dataset 6: TCGA + OR, XOR, AND

Following the same procedure detailed Dataset 5 above and using the same feature-reduced TCGA data, we embedded simulated Dataset 4 (AND, OR, XOR operators) into the TCGA data.

B DM Algorithm

The assumptions required by the DM algorithm are somewhat similar to the assumptions required for the RINN. There are a few differences between the assumptions made by each algorithm. The DM algorithm requires each latent variable to have have at least one observed cause and one observed effect; however, the RINN can represent a latent variable without an observed cause through the use of a nonzero bias node. The DM algorithm also assumes that there is only one directed path between any two observed variables, meaning that each output variable has only one directed edge into it. *This is a very limiting assumption for our purposes and the RINN algorithm does not assume this.* Biologically, this assumption is violated as there is often redundancy in cellular signaling pathways and many ways to activate a given protein (e.g., transcription factor). The DM algorithm is also dependent on conditional independence tests, which have additional hyperparameters that need to be set. In general, the DM algorithm has more constraints than the RINN algorithm. However, the correctness of the RINN is dependent upon reaching the global optimum, which is most often not the case when performing gradient descent to optimize neural networks. Another major difference between the RINN and the DM algorithm is that the DM algorithm only returns a causal structure (without parameterization), whereas the RINN returns a causal structure and parameterization. The DM algorithm is available in Tetrad (http://www.phil.cmu.edu/tetrad/, https://github.com/cmu-phil/tetrad).

C Model Selection

For the deep learning strategies, each simulated dataset was separated into training, validation, and test sets according to the following ratio 0.75/0.15/0.10, respectively. We wanted to avoid the overfitting that often occurs if using a single validation dataset, so using the 90% (training + validation) split, we randomly selected 15% for one validation set (while training on the 75% remaining data) and a different 15% for the other validation set (while again training on the remaining 75%). We used a combined random and grid search approach to select the hyperparameters (learning rate, regularization rate, size of hidden layer, etc.) for the best models with the main objective of finding the optimal balance between sparsity and prediction error to encourage the discovery of latent causal structure. See Figure 4 for example model selection results.

Our search through hyperparameter space varied somewhat depending on the data and the deep learning strategy being evaluated. Datasets 5 and 6 were considerably larger than the other simulated datasets as they contained a large amount of real TCGA data. This increased the number of hyperparameters to be evaluated (i.e., more hidden layer sizes to search through because the input was larger), so we performed model selection on 100 more sets of hyperparameters for these larger datasets. Also, DNNs, DBNs, and ES-C have additional hyperparameters to search through relative to the RINN. For DNNs and DBNs, we need to find the optimal number of hidden layers (whereas the structure of the RINN allows one to largely avoid determining the ideal number of hidden layers—Section 3). For simplicity, we assumed all DNN hidden layers to be of the same size. However, DBN model selection uses the decreasing nature of the hidden layers as a driving force to learn the most salient aspects of the data. Therefore, DBN model selection included searching over different sizes of the hidden layers for each of the hidden layers. This means more hyperparameters to search through. The additional hyperparameters for ES-C included, elite ratio, mutation rate, and set of legal weight values.

For binary outputs, binary cross-entropy plus L_1 regularization was used as the objective function. For non-binary outputs, mean squared error (MSE) plus L_1 regularization was used. Almost all of the best performing models used ReLU (Rectified Linear Unit) activation functions, although we performed model selection using sigmoid and tanh activation functions as well.

A difference between the RINN and a DNN in this paper is that the RINN was always trained with eight hidden layers. We hypothesize (for eventually using this in a biological setting) that most biological pathways will not have more than an 8-level hierarchy. Importantly, since copies of the input are available to use at each hidden layer (should the algorithm decide to use it), we don't need to determine the best number of hidden layers (as a model selection hyperparameter)—this means less model selection is required for RINNs. If the optimal way to learn the structure in the data with a RINN is by using a 3-hidden layer network, the RINN will learn nonzero weights starting with the copy of the input that is connected to hidden layer 6. This would create a 3-hidden layer network (hidden layers 6, 7, and 8), with all weights in the network before the redundant input connected to hidden layer 6 being equal to 0.0. This happens because of L_1 regularization, which constrains the RINN to map input to output using as few weights as possible. If there is reason to think that there may be more hierarchical levels in a particular dataset, the number of maximum hidden layers can always be increased with a RINN, or treated as a hyperparameter.

Table 3: Number of Sets of Hyperparameters Evaluated for each Deep Learning Strategy

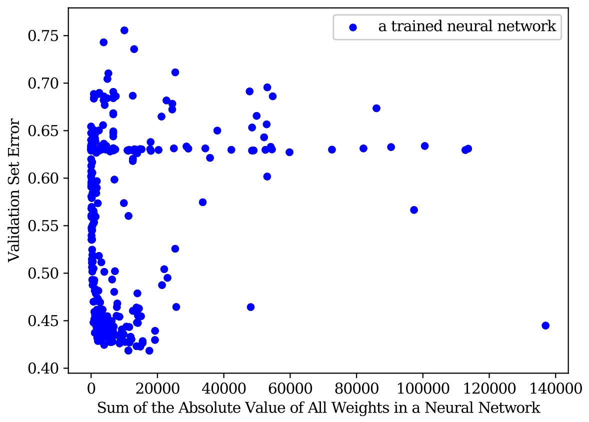
Strategy	Datasets 1,2,3,4	Datasets 5,6
RINN	440	540
DNN	732	832
DBN	940	1040
ES-C	500	NA

For ES-C, we performed model selection over multiple hyperparameters including, elite ratio (percent of population saved to mate and produce the next generation), mutation rate, regularization rate, and the legal weight values. We used a population size of 200 and evolved for up to 13,000 generations or epochs. The *MATE* function involved choosing two individuals in the elite population and randomly combining 50% of the weights from one individual with 50% of the weights from the other individual (Algorithm 2). The evolutionary strategy used in this work was not set up to be parallelized across processors, as early prototyping experiments suggested that ES-C would not perform nearly as well as RINN. This, unfortunately, reduced the amount of model selection we could perform in a timely manner. To speed up ES-C, we only trained on 20% of the data that was used to train the other algorithms. Testing 500 sets of hyperparameters on one dataset took approximately three weeks on a single desktop computer.

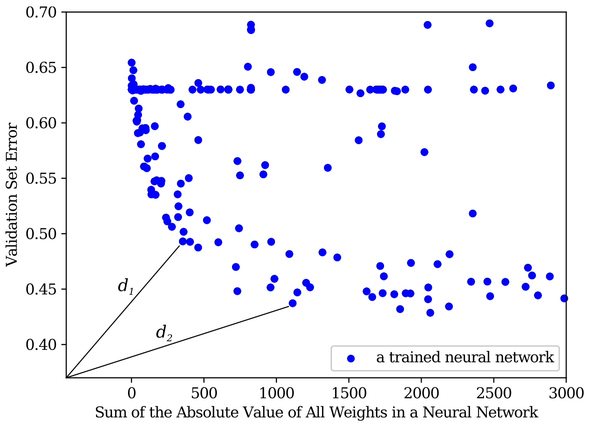
See Table 3 for a breakdown of the number of sets of hyperparameters that were evaluated for each strategy.

For the DM algorithm, we selected values for the first alpha level $\alpha \in \{0.05, 0.01, 0.001, 0.0001\}$ and values for the second alpha level $\alpha \in \{0.1, 0.05, 0.01\}$. Results for the best performing combinations of tuning parameters for the DM algorithm were reported.

Figure 4 shows model selection results for RINNs on simulated Dataset 5. These are typical model selection results across all deep learning strategies and datasets. Figure 4a shows the validation set error and the sum of the absolute value of all weights in a network for 1,080 (540 times two validation sets) fully trained RINNs. There is a concentration of networks (shown as blue circles) in the lower left corner of this figure. It is within these models that we hypothesize one will find the models with the highest probability of capturing the causal structure of the data within their weights. Figure 4b shows a magnification of the lower left of Figure 4a, and here it becomes apparent that model selection based solely on prediction error or sparsity (sum of the absolute values of the weights) is inadequate. We also don't know, 1. Will deep learning strategies capture causal relationships in their weights? and 2. If they do capture this structure in their weights, how can we identify them during model selection (i.e., Where would these models be located in the plots in Figure 4?)? We hypothesized that the models with best chances of capturing causal relationships in their weights will be the models that optimally balance both prediction error and sparsity (i.e., the models in bottom left of Figure 4a). To quantify this hypothesis and provide a ranking of the most optimal models, we measured the distance between each model's point location in the plot in Figure 4b and the origin, and used this as a metric to rank the models (Section 7 for more details). We hypothesize that the models closer to the origin (using the distance metric described above) would better capture the ground truth causal structure in their weights.



(a) Example Model Selection Results (RINN Trained on TCGA+OR Dataset). This figure represents 1080 trained neural networks.



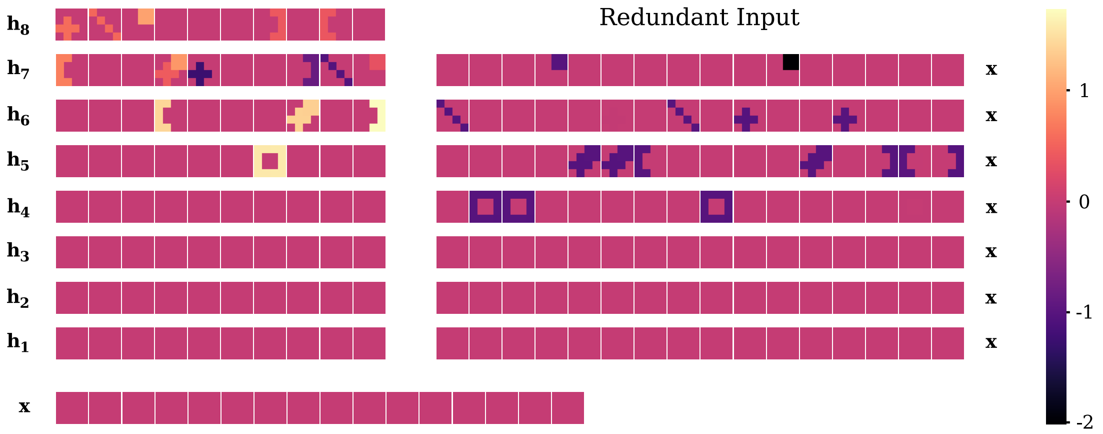
(b) Lower left of (a) magnified. Lines d_1 and d_2 represent distance from origin to a trained neural network. In practice, the distance is calculated using the point $(0, 0)$ and axes scaled to the interval $[0, 1]$ (neither of which are depicted here).

Figure 4: Measuring Euclidean Distance to Rank Model Selection Results

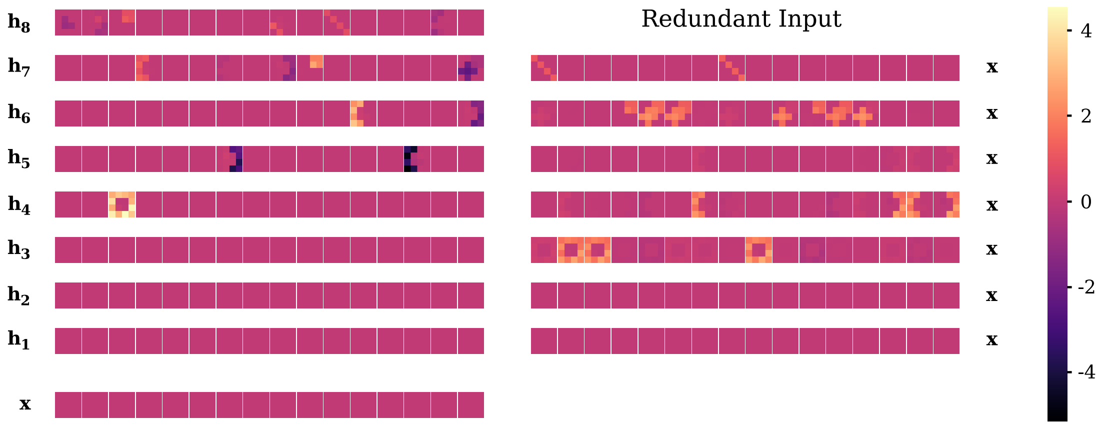
D RINN Visualization

Figure 5 shows the visualizations for all weight matrices in two different trained RINNs. The RINN visualized in Figure 5a is an example of a network that scores 100% precision and recall for both input and hidden nodes. To evaluate the input nodes of a RINN, all input node heatmaps for a particular node are summed together, and this final sum of heatmaps is compared to the known ground truth for that input. We can see in Figure 5a that there is no need to sum the heatmaps because the input nodes learn the correct mapping (Figure 3a (bottom)) at only a single redundant input layer (for each input node). When we look at the heatmaps for the hidden nodes, we see that all seven ground truth patterns from Figure 3a (top) are present in either $\boldsymbol{h}_5$, $\boldsymbol{h}_6$, $\boldsymbol{h}_7$, or $\boldsymbol{h}_8$. Figure 5b shows the visualization of a different RINN that doesn't score as well as Figure 5a. The input nodes of this RINN achieve 100% precision and recall. But the hidden nodes for this RINN show four FPs, six TPs (although one of the combination nodes, i.e., nodes 17 and 19 in Figure 2a and 2e, is very close to missing the threshold), and one FN ($7 - 6 = 1$). The final precision and recall for the hidden nodes would be: $P = \frac{6}{6+4} = 0.60$, $R = \frac{6}{6+1} = 0.86$.

Precision and recall for the TCGA+OR and TCGA+OR, XOR, AND datasets were calculated based only on the recovery of the simulated data ground truth—as we don't know the ground truth for the TCGA part of the data and therefore have no way to calculate precision and recall.



(a) A RINN (trained on Dataset 1) that finds all input and hidden patterns (i.e., 100% precision and recall).



(b) A different RINN (trained on Dataset 3) that finds all input patterns, but has some difficulty with the hidden patterns (3 or 4 false positives*, 6 true positives, and 1 false negative). *All matching of patterns depends on setting a threshold, and anything above this threshold counts as a nonzero value in the heatmap.

Figure 5: RINN Weight Visualizations

E Pseudocode

Algorithm 1 Redundant Input Neural Network (RINN)

```
$\boldsymbol{W}_{l,a}$ is weight matrix bet hid layer $l-1$ and hid layer $l$
$\boldsymbol{W}_{l,b}$ is weight matrix between input $l-1$ and hid layer $l$
$\boldsymbol{h}_l$ is the vector of values representing hidden layer $l$
$\boldsymbol{x_i}$ is the input vector for instance $i$
$\boldsymbol{y}_i$ is the output vector for instance $i$
$s$ is the number of samples in the dataset
$ERROR$ is the objective function to be optimized

for $i = 1$ to $s$ do
    $\boldsymbol{a}_1 = ReLU(\boldsymbol{x}_i \cdot \boldsymbol{W}_{0,b})$
    for $j = 1$ to $num\_hid\_layers$ do
        $\boldsymbol{a}_j$ = CONCATENATE($\boldsymbol{a}_j, \boldsymbol{x}_i$)
        $\boldsymbol{W}_j$ = CONCATENATE($\boldsymbol{W}_{j,a}, \boldsymbol{W}_{j,b}$)
        $\boldsymbol{a}_{j+1} = ReLU(\boldsymbol{a}_j \cdot \boldsymbol{W}_j)$
    end for
    $\hat{\boldsymbol{y}}_i = sigmoid(\boldsymbol{a}_{j+1} \cdot \boldsymbol{W}_{j+1,a})$        ▷ for binary $\boldsymbol{y}$
    Update all $\boldsymbol{W}$ by descending their gradient:

            $\nabla_{\boldsymbol{W}} ERROR(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i)$

end for
```

Algorithm 2 Genetic Algorithm for Neural Network Weight Optimization

```
$\boldsymbol{f}$ is a vector of fitness values
$\boldsymbol{Data}$ represents all input and output data
$W_i$ is all the weights of neural net $i$ reshaped into a vector
$pw$ is the set of possible weight values (e.g., $\{-1, 0, 1\}$)

for $i = 1$ to $num\_generations$ do    ▷ for each generation
    for $j = 1$ to $population\_size$ do    ▷ for each neural net
        $f_j$ = FITNESS($W_j$, $\boldsymbol{Data}$)
    end for
    $elites$ = TOP_20_PERCENT($\boldsymbol{f}$)    ▷ get best neural nets
    for $k = 1$ to $p$ do
        $W_k$ = MATE(RAND($elites$), RAND($elites$))
        $W_k$ = MUTATE($W_k, pw$)    ▷ $\Delta$ some $W_k$ randomly
    end for
end for
RETURN $W_{best}$        ▷ neural network with highest fitness function
```

Algorithm 3 Simulate Data using Matrix Multiplication and Interventions

```
$p$ is the Bernoulli success prob    ▷ prob that mutation present
$s$ is the number of samples we want to generate
$\boldsymbol{Data}$ is a container for the simulated data
$\boldsymbol{W}_l$ is the $G_T$ weight matrix between hidden layer $l-1$ and $l$
$\boldsymbol{h}_l$ is the vector of values representing hidden layer $l$
$\boldsymbol{W}_x$ is a ground truth matrix with all input node adjacencies

$p = 0.10$
for $i = 1$ to $s$ do
    SAMPLE $\boldsymbol{x} \sim \mathcal{B}(16, p)$        ▷ Bernoulli Trials
    $\boldsymbol{h}_1 = [1, 1, 1, 1]^\top$
    for $j = 1$ to $2$ do
        set nodes in $\boldsymbol{h}_j$ targeted by $\boldsymbol{x}$ to 0        ▷ use $\boldsymbol{W}_x$
        $\boldsymbol{h}_{j+1} = \boldsymbol{h}_j \boldsymbol{W}_{j+1}$
    end for
    SAVE $\boldsymbol{x}$ and $\boldsymbol{h}_{j+1}$ in $\boldsymbol{Data}$
end for
$return$ $\boldsymbol{Data}$
```

Algorithm 4 Simulate Data from AND, OR, XOR Logical Operators

```
$G_T = (V, E)$        ▷ ground truth DAG
PARENTS($node$) returns binary values of the parents of $node$
$\boldsymbol{X}$ is a $n \times m$ matrix
$\boldsymbol{Data}$ is a $n \times m$ matrix
$\boldsymbol{D}$ is a container of Bernoulli distributions
$s$ is the number of samples we want to generate

for $node$ in $V$ do
    $op$ = RANDOM_CHOICE($AND, OR, XOR$)
    for each possible binary combo, $b$, of parents of $node$ do
        if $op(b)$ is $True$ then
            SAMPLE $D_{node,b} \sim Beta(95, 5)$
        else if $op(b)$ is $False$ then
            SAMPLE $D_{node,b} \sim Beta(5, 95)$
        end if
    end for
end for

SAMPLE $\boldsymbol{X} \sim \mathcal{U}(0, 1)$
for $i = 1$ to $s$ do
    for $node$ in $V$ do    ▷ start from input and go to output
        $b$ = PARENTS($node$)
        $Data_{i,node} = D_{node,b} < X_{i,node}$
    end for
end for
$return$ $\boldsymbol{Data}$
```