Unsupervised Deep Structure Learning by Recursive Independence Testing

Raanan Y. Yehezkel Rohekar, Guy Koren, Shami Nisimov, Gal Novik
Intel Corporation

Abstract

We introduce a principled approach for unsupervised structure learning of deep, feed-forward, neural networks. We propose a new interpretation for depth and inter-layer connectivity where conditional independencies in the input distribution are encoded hierarchically in the network structure. Neurons in deeper layers encode low-order (small condition sets) independencies and have a wide scope of the input, whereas neurons in the first layers encode higher-order (larger condition sets) independencies and have a narrower scope. Thus, the depth of the network is equal to the maximal order of independence in the input distribution. Moreover, this results in structures allowing neurons to connect to neurons in any deeper layer, skipping intermediate layers. The proposed algorithm constructs three main graphs: 1) a deep generative-latent-graph, learned recursively from data using a conditional independence test, 2) a stochastic inverse, and 3) a discriminative graph constructed from the stochastic inverse. We prove that conditional-dependency relations in the learned generative latent graph are preserved in both the stochastic inverse and the class-conditional discriminative graphs. Finally, a deep neural network structure is constructed from the discriminative graph. We demonstrate on image classification benchmarks that the deepest layers (convolutional and dense layers) of common convolutional networks can be replaced by significantly smaller learned structures, achieving high classification accuracy. Our structure learning algorithm requires a small computational cost and runs efficiently on a standard desktop CPU.

1 Introduction

In this paper, we focus on the design of neural network topology—structure learning. Generally, exploration of this design space is a time consuming iterative process that requires close supervision by a human expert. Recent studies have focused on automating the exploration of the design space, posing it as a hyper-parameter optimization problem and proposing various approaches to solve it. For example, methods based on reinforcement-learning (Zoph & Le, 2016; Zoph et al., 2017; Real et al., 2017; Baker et al., 2016) and evolutionary methods (Real et al., 2017; Miikkulainen et al., 2017). Common to all these recent studies is the fact that structure learning is done in a supervised manner and requires large compute resources, rendering the solution unfeasible for many applications given limited compute and time resources.

The problem of model structure learning has been widely researched for many years in the probabilistic graphical models domain. Specifically, Bayesian networks for density estimation and causal discovery (Pearl, 2009; Spirtes et al., 2000). Two main approaches were studied: score-based (search-and-score) and constraint-based. Score-based approaches combine a scoring function (Cooper & Herskovits, 1992; Ripley, 2007) with a search strategy, such as greedy equivalence search (Chickering, 2002). Adams et al. (2010) introduced an algorithm for sampling deep belief networks (generative model) and demonstrated its applicability to image datasets.

Second workshop on Bayesian Deep Learning (NIPS 2017), Long Beach, CA, USA.

Constraint-based approaches (Pearl, 2009; Spirtes et al., 2000) find the optimal structures in the large sample limit by testing conditional independence between variables. They are generally faster than score-based approaches (Yehezkel & Lerner, 2009) and have a well-defined stopping criterion (Dash & Druzdzel, 2003). However, these methods are sensitive to errors in the independence tests, especially in the case of high-order conditional-independence tests and small training sets.

Motivated by these methods, we propose a new approach for learning the structure of deep neural networks in an unsupervised manner. We do not claim to identify the presence of confounders and their inter-relations as in Elidan et al. (2001); Silva et al. (2006); Asbeh & Lerner (2016). Instead, we augment a fully observed Bayesian network with latent variables, while preserving conditional dependence.

2 Recursive Deep Structure Learning

Preliminaries. Consider $\boldsymbol{X} = \{X_i\}_{i=1}^{N}$ a set of observed (input) random variables, $\boldsymbol{H}$ a set of latent variables, and Y a class variable. Our algorithm constructs three graphical models and an auxiliary graph. Each variable is represented by a single node, and a single edge may connect two distinct nodes. Graph $\mathcal{G}$ is a generative DAG defined over the observed and latent variables $\boldsymbol{X} \cup \boldsymbol{H}$. Graph $\mathcal{G}_{\text{Inv}}$ is called a stochastic inverse of $\mathcal{G}$. Graph $\mathcal{G}_{\text{D}}$ is a discriminative model defined over the observed, latent, and class variables $\boldsymbol{X} \cup \boldsymbol{H} \cup Y$. An auxiliary graph $\mathcal{G}_X$ is defined over $\boldsymbol{X}$ (a CPDAG; a Markov equivalence class) and is generated and maintained as an internal state of the algorithm. The parents set of a node X in $\mathcal{G}$ is denoted $\boldsymbol{Pa}(X;\mathcal{G})$. The order of an independence relation is defined to be the condition set size. For example, if X_1 and X_2 are independent given X_3, X_4, and X_5, denoted $X_1 \perp\!\!\!\perp X_2 | \{X_3, X_4, X_5\}$, then the independence order is three.

First, $\mathcal{G}$, a deep generative latent structure, is learned from data (using a conditional independence test). The key idea is to recursively introduce a new and deeper latent layer by testing n-th order conditional independence (among $\boldsymbol{X}$) and connect it to latent layers created by subsequent recursive calls that test conditional independence of order $n+1$. Yehezkel & Lerner (2009) introduced an efficient algorithm (RAI) for constructing a CPDAG over $\boldsymbol{X}$ by a recursive application of conditional independence tests with increasing condition set sizes. Our algorithm is based on this framework for updating the auxiliary graph $\mathcal{G}_X$. Our proposed algorithm is defined in Algorithm 1 (a flow chart is given in Appendix A). A 2-layer toy-example is given in Figure 1.

The algorithm starts with $n = 0$, $\mathcal{G}_X$ a complete graph, and a set of exogenous nodes $\boldsymbol{X}_{\text{ex}} = \emptyset$. The set $\boldsymbol{X}_{\text{ex}}$ is exogenous to $\mathcal{G}_X$ and consists of parents of $\boldsymbol{X}$.

The function `IncreaseResolution` (Algorithm 1-line 5) disconnects (in $\mathcal{G}_X$) conditionally independent variables in two steps. First, it tests dependency between $\boldsymbol{X}_{\text{ex}}$ and $\boldsymbol{X}$, i.e., $X \perp\!\!\!\perp X' | \boldsymbol{S}$ for every connected pair $X \in \boldsymbol{X}$ and $X' \in \boldsymbol{X}_{\text{ex}}$ given a condition set $\boldsymbol{S} \subset \{\boldsymbol{X}_{\text{ex}} \cup \boldsymbol{X}\}$ of size n. Next, it tests dependency within $\boldsymbol{X}$, i.e., $X_i \perp\!\!\!\perp X_j | \boldsymbol{S}$ for every connected pair $X_i, X_j \in \boldsymbol{X}$ given a condition set $\boldsymbol{S} \subset \{\boldsymbol{X}_{\text{ex}} \cup \boldsymbol{X}\}$ of size n. After removing the corresponding edges, the remaining edges are directed by applying two rules (Pearl, 2009; Spirtes et al., 2000). First, v-structures are identified and directed. Then, edges are continually directed, by avoiding the creation of new v-structures and directed cycles, until no more edges can be directed. Following the terminology of Yehezkel & Lerner (2009), we say that this function increases the graph d-separation resolution from $n-1$ to n.

The function `SplitAutonomous` (Algorithm 1-line 6) identifies autonomous sets, one descendant set $\boldsymbol{X}_{\text{D}}$ and K ancestor sets $\boldsymbol{X}_{\text{A}1}, \ldots, \boldsymbol{X}_{\text{A}K}$ in two steps. First, the nodes having the lowest topological order are grouped into $\boldsymbol{X}_D$. Then, $\boldsymbol{X}_{\text{D}}$ is removed (temporarily) from the graph revealing unconnected sub-structures. The number of unconnected sub-structures is denoted K and the nodes of each sub-structure is denoted $\boldsymbol{X}_{\text{A}i}$ ($i \in \{1 \ldots K\}$).

An autonomous set in $\mathcal{G}_X$ includes all its nodes' parents (complying with the Markov property) and therefore a corresponding latent structure can be further learned independently, using a recursive call. Thus, the algorithm is called recursively and independently for the ancestor sets (Algorithm 1 lines 7–8), and then for the descendant set, treating the ancestor sets as exogenous (Algorithm 1 line 9).

Each recursive call returns a latent structure for each autonomous set. Recall that each latent structure encodes a generative distribution over the observed variables where layer $\boldsymbol{H}^{(n+1)}$, the last added layer (parentless nodes), is a representation of the input $\boldsymbol{X'} \subset \boldsymbol{X}$. Thus, latent variables are introduced as parents of the $\boldsymbol{H}^{(n+1)}$ layers (Algorithm 1 lines 11–13).

Algorithm 1: Recursive Latent Structure Learning (multi-layer)

RecursiveLatStruct $(\mathcal{G}_X, \boldsymbol{X}, \boldsymbol{X}_{\text{ex}}, n)$

Input: an initial DAG $\mathcal{G}_X$ over observed $\boldsymbol{X}$ & exogenous nodes $\boldsymbol{X}_{\text{ex}}$ and a desired resolution n.
Output: $\mathcal{G}$, a latent structure over $\boldsymbol{X}$ and $\boldsymbol{H}$

if *the maximal indegree of* $\mathcal{G}_X(\boldsymbol{X})$ *is below* $n+1$ **then** ▷ `exit condition`
  $\mathcal{G} \longleftarrow$ an observed layer $\boldsymbol{X}$
  return $\mathcal{G}$

$\mathcal{G}'_X \longleftarrow$ `IncreaseResolution`$(\mathcal{G}_X, n)$ ▷ n-`th order independencies`

$\{\boldsymbol{X}_{\text{D}}, \boldsymbol{X}_{\text{A}1}, \ldots, \boldsymbol{X}_{\text{A}K}\} \longleftarrow$ `SplitAutonomous`$(\boldsymbol{X}, \mathcal{G}'_X)$ ▷ `identify autonomies`

for $i \in \{1 \ldots K\}$ **do**
  $\mathcal{G}_{\text{A}i} \longleftarrow$ `RecursiveLatStruct`$(\mathcal{G}'_X, \boldsymbol{X}_{\text{A}i}, \boldsymbol{X}_{\text{ex}}, n+1)$ ▷ `a recursive call`

$\mathcal{G}_{\text{D}} \longleftarrow$ `RecursiveLatStruct`$(\mathcal{G}'_X, \boldsymbol{X}_{\text{D}}, \boldsymbol{X}_{\text{ex}} \cup \{\boldsymbol{X}_{\text{A}i}\}_{i=1}^{K}, n+1)$ ▷ `a recursive call`

$\mathcal{G} \longleftarrow$ `Group`$(\mathcal{G}_{\text{D}}, \mathcal{G}_{\text{A}1}, \ldots, \mathcal{G}_{\text{A}K})$ ▷ `merge results`

create K latent variables $\boldsymbol{H}^{(n)} = \{H_1^{(n)}, \ldots, H_K^{(n)}\}$ in $\mathcal{G}$ ▷ `create a latent layer`

set each $H_i^{(n)}$ to be a parent of $\{\boldsymbol{H}_{\text{A}i}^{(n+1)} \cup \boldsymbol{H}_{\text{D}}^{(n+1)}\}$ ▷ `connect`

where $\boldsymbol{H}_{\text{A}i}^{(n+1)}$ and $\boldsymbol{H}_{\text{D}}^{(n+1)}$ are the sets of parentless latents in $\mathcal{G}_{\text{A}i}$ and $\mathcal{G}_{\text{D}}$, respectively.

return $\mathcal{G}$

It is important to note that conditional independence is tested only between input variables $\boldsymbol{X}$, and condition sets do not include latent variables. Conditioning on latent variables or testing independence between them is not required.

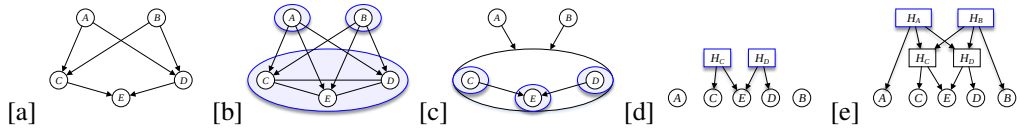


Figure 1: An example of learning a 2-layer latent generative model. [a] An example of a "ground-truth" Bayesian network encoding the distribution that will be discovered by the algorithm. Recall that our algorithm learns a deep generative model that can mimic this DAG (not necessarily equivalent to it). [b] $\mathcal{G}_X$ after marginal independence testing ($n = 0$). [c] $\mathcal{G}_X$ after a recursive call to learn the structure of nodes $\{C, D, E\}$ with $n = 2$. Exit condition is met in subsequent recursive calls and thus latent variables are added to $\mathcal{G}$ at $n = 2$ [d], and then at $n = 0$ [e] (the final structure).

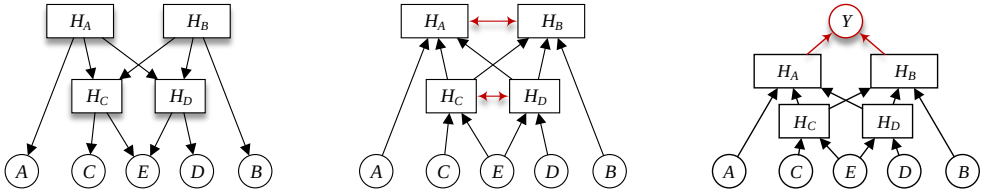
Next, a stochastic inverse $\mathcal{G}_{\text{Inv}}$ (Stuhlmüller et al., 2013; Paige & Wood, 2016), such that $\mathcal{G}_{\text{Inv}}$ can mimic $\mathcal{G}$ over all the variables (preserves conditional dependence over latents and observed; Proposition 1, Section 3), is constructed in two steps:

1. Invert all $\mathcal{G}$ edges (invert inter-layer connectivity).
2. Connect each pair of latent variables, sharing a common child in $\mathcal{G}$, with a bi-directional edge.

We avoid limiting $\mathcal{G}_{\text{Inv}}$ to a DAG. Instead, we consider it to be a projection of another latent structure (Pearl, 2009). That is, we assume the presence of *additional* hidden variables $\boldsymbol{Q}$ that are not in $\mathcal{G}_{\text{Inv}}$ but induce dependency[1] among $\boldsymbol{H}$, which are represented by the bi-directional edges.

Finally, a discriminative graph $\mathcal{G}_{\text{D}}$ is constructed by replacing bi-directional dependency relations in $\mathcal{G}_{\text{Inv}}$ (induced by $\boldsymbol{Q}$) with explaining-away relations, by adding the observed class variable Y. Node Y is set in $\mathcal{G}_{\text{D}}$ to be the common child of the leaves in $\mathcal{G}_{\text{Inv}}$ (latents introduced after testing marginal independencies in $\boldsymbol{X}$). See an example in Figure 2. This ensures the preservation of conditional dependency relations in $\mathcal{G}_{\text{Inv}}$. That is, $\mathcal{G}_{\text{D}}$ can mimic $\mathcal{G}_{\text{Inv}}$ over $\boldsymbol{X}$ and $\boldsymbol{H}$ given Y (Section 3). A neural network is then constructed by replacing each latent variable with a dense neural network layer and the class variable with an output layer (e.g., a softmax layer). See one result in Figure 3.

[1] For example, "interactive forks" (Pearl, 2009).



[a] [b] [c]

Figure 2: An example of the three graphs constructed by our algorithm: [a] a generative deep latent structure, [b] its stochastic inverse, and [c] a discriminative structure (class node Y is added).

3 Preservation of Conditional Dependence

Conditional dependence relations encoded by the genrative structure $\mathcal{G}$ are preserved by the discriminative structure $\mathcal{G}_{\mathrm{D}}$ conditioned on the class Y. That is, $\mathcal{G}_{\mathrm{D}}$ conditioned on Y can mimic $\mathcal{G}$; denoted by preference relation $\mathcal{G} \preceq \mathcal{G}_{\mathrm{D}}|Y$. While the parameters of a model can learn to mimic conditional independence relations that are not expressed by the graph structure, they are not able to learn conditional dependence relations (Pearl, 2009).

Proposition 1. *Graph $\mathcal{G}_{\mathrm{Inv}}$ preserves all conditional dependencies in $\mathcal{G}$ (i.e., $\mathcal{G} \preceq \mathcal{G}_{\mathrm{Inv}}$).*

Proof. Graph $\mathcal{G}_{\mathrm{Inv}}$ can be constructed using the procedures described by Stuhlmüller et al. (2013) where nodes are added, one-by-one, to $\mathcal{G}_{\mathrm{Inv}}$ in a reverse topological order (lowest first) and connected (as a child) to existing nodes in $\mathcal{G}_{\mathrm{Inv}}$ that d-separate it, according to $\mathcal{G}$, from the remainder of $\mathcal{G}_{\mathrm{Inv}}$. Paige & Wood (2016) showed that this method ensures the preservation of conditional dependence $\mathcal{G} \preceq \mathcal{G}_{\mathrm{Inv}}$. We set an equal topological order to every pair of latents (H_i, H_j) sharing a common child in $\mathcal{G}$. Hence, jointly adding nodes H_i and H_j to $\mathcal{G}_{\mathrm{Inv}}$, connected by a bi-directional edge, requires connecting them (as children) only to their children and the parents of their children (H_i and H_j themselves, by definition) in $\mathcal{G}$. That is, without loss of generality, node H_i is d-separated from the remainder of $\mathcal{G}_{\mathrm{Inv}}$ given its children in $\mathcal{G}$ and H_j. ■

It is interesting to note that the stochastic inverse $\mathcal{G}_{\mathrm{Inv}}$, constructed without adding inter-layer connections, preserves all conditional dependencies in $\mathcal{G}$.

Proposition 2. *Graph $\mathcal{G}_{\mathrm{D}}$, conditioned on Y, preserves all conditional dependencies in $\mathcal{G}_{\mathrm{Inv}}$ (i.e., $\mathcal{G}_{\mathrm{Inv}} \preceq \mathcal{G}_{\mathrm{D}}|Y$).*

Proof. It is only required to prove that the dependency relations that are represented by bi-directional edges in $\mathcal{G}_{\mathrm{Inv}}$ are preserved in $\mathcal{G}_{\mathrm{D}}$. The proof follows directly from the d-separation criterion (Pearl, 2009). A latent pair $\{H, H'\} \subset \boldsymbol{H}^{(n+1)}$, connected by a bi-directional edge in $\mathcal{G}_{\mathrm{Inv}}$, cannot be d-separated by any set containing Y, as Y is a descendant of a common child of H and H'. In Algorithm 1-line 12, a latent in $\boldsymbol{H}^{(n)}$ is connected, as a child, to latents $\boldsymbol{H}^{(n+1)}$, and Y to $\boldsymbol{H}^{(0)}$. ■

We formulate $\mathcal{G}_{\mathrm{Inv}}$ as a projection of another latent model (Pearl, 2009) where bi-directional edges represent dependency relations induced by latent variables $\boldsymbol{Q}$. We construct a discriminative model by considering the effect of $\boldsymbol{Q}$ as an explaining-away relation induced by a class node Y. Thus, conditioned on Y, the discriminative graph $\mathcal{G}_{\mathrm{D}}$ preserves all conditional (and marginal) dependencies in $\mathcal{G}_{\mathrm{Inv}}$.

Proposition 3. *Graph $\mathcal{G}_{\mathrm{D}}$, conditioned on Y, preserves all conditional dependencies in $\mathcal{G}$ (i.e., $\mathcal{G} \preceq \mathcal{G}_{\mathrm{D}}$).*

Proof. It immediately follows from Propositions 1 & 2 that $\mathcal{G} \preceq \mathcal{G}_{\mathrm{Inv}} \preceq \mathcal{G}_{\mathrm{D}}$ conditioned on Y. ■

Thus $\mathcal{G} \preceq \mathcal{G}_{\mathrm{Inv}} \preceq \mathcal{G}_{\mathrm{D}}$ conditioned on Y.

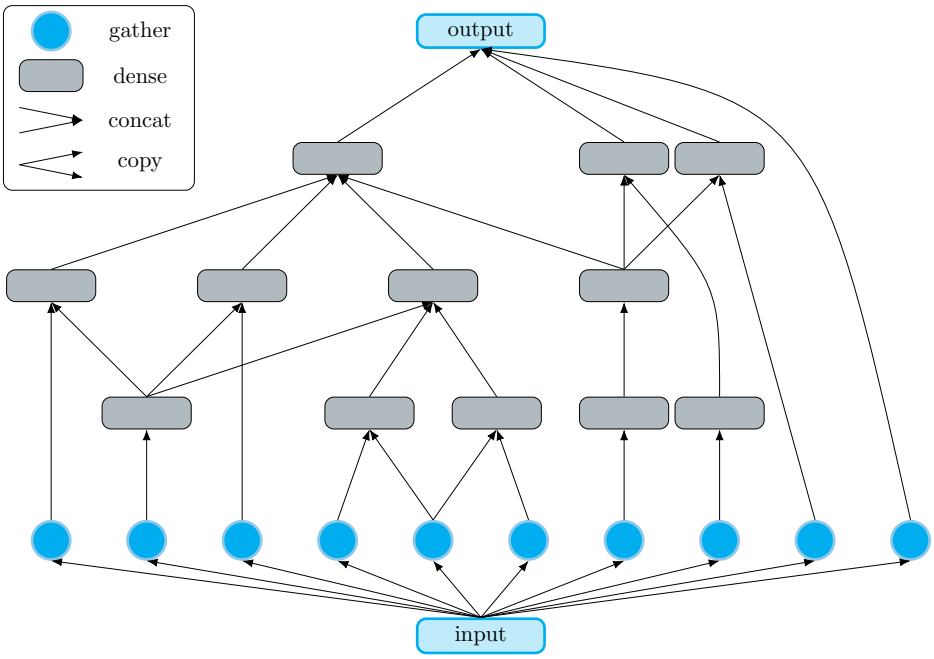


Figure 3: An example of a structure learned by our algorithm (classifying MNIST digits). Neurons in a layer may connect to neurons in any deeper layer. Depth is determined automatically. Each gather layer selects a subset of the input, where each input variable is gathered only once. A neural route, starting with a gather layer, passes through densely connected layers where it may split (copy) and merge (concatenate) with other routes in correspondence with the hierarchy of independencies identified by the algorithm. All routes merge into the final output layer (e.g., a softmax layer).

4 Experiments

We evaluate the quality of learned structures using five image classification benchmarks. We compare the learned structures to common topologies (and simpler hand-crafted structures), which we call "vanilla topologies", with respect to network size and classification accuracy. The benchmarks and vanilla topologies are described in Table 1. Our structure learning algorithm is implemented using the Bayesian network toolbox (Murphy, 2001) and runs efficiently on a standard desktop CPU (excluding NN parameters learning). Our main result is given in Table 2. A detailed description and an extended evaluation is given in Appendix B. For SVHN only the basic training data is used, i.e., 13% of the available training data, and for ImageNet 5% of the training data is used.

Our structure learning algorithm runs efficiently on a standard desktop CPU, while providing structures with competitive classification accuracies. For example, the lowest classification error rate achieved by our *unsupervised* algorithm for CIFAR 10 is 4.58% with a network of size 6M (WRN-40-4 row in Table 2). For comparison, the NAS algorithm (Zoph & Le, 2016) achieves error rates of 5.5% and 4.47% for networks of sizes 4.2M and 7.1M, respectively, and requires optimizing thousands of networks using hundreds of GPUs.

5 Conclusions

We presented a principled approach for learning, in an unsupervised manner, the structure of deep neural networks. The resulting structures provide a hierarchical encoding of the independencies in the input distribution that we identified by the algorithm . Interestingly, this results in a network containing "neural routes", passing through densely connected layers, where they may split (copy) and merge (concatenate) with other routes in correspondence with the hierarchy of independencies identified by the (unsupervised) algorithm. Moreover, neurons in a layer may connect to neurons in any deeper layer, and the network depth is determined automatically.

We demonstrated that our algorithm learns small structures, and provides high classification accuracies for common image classification benchmarks. It is also demonstrated that while convolution layers are very useful at exploiting domain knowledge, such as spatial smoothness, translational invariance, and symmetry, they are mostly outperformed by a learned structure for the deeper layers. Moreover, while the use of common topologies for a variety of classification tasks is computationally inefficient, we would expect our approach to learn smaller and more accurate networks for each classification task, uniquely.

As only unlabeled data is required for learning the structure, we expect our approach to be practical for many domains, beyond image classification, such as knowledge discovery, and plan to explore the interpretability of the learned structures.

benchmark	vanilla topology		
dataset	topology	description	size
MNIST (LeCun et al., 1998)	None	learn a structure directly from pixels	
	MNIST-Man	32-64-FC:128	127K
SVHN (Netzer et al., 2011)	Maxout NiN	(Chang & Chen, 2015)	1.6M
	SVHN-Man	16-16-32-32-64-FC:256	105K
CIFAR 10 (Krizhevsky & Hinton, 2009)	VGG-16-D	(Simonyan & Zisserman, 2014)	15M
	WRN-40-4	(Zagoruyko & Komodakis, 2016)	9M
CIFAR 100 (Krizhevsky & Hinton, 2009)	VGG-16-D	(Simonyan & Zisserman, 2014)	15M
ImageNet (Deng et al., 2009)	AlexNet	(Krizhevsky et al., 2012)	61M

Table 1: Benchmarks and vanilla topologies. MNIST-Man and SVHN-Man topologies were manually created by us. MNIST-Man has two convolutional layer (32 and 64 filters each) and one dense layer with 128 neurons. SVHN-Man was created as a small network reference having reasonable accuracy compared to Maxout-NiN. In the first row we indicate that in one experiment a structure for MNIST was learned from the pixels and feature extracting convolutional layers were not used.

	vanilla topology			learned structure			
dataset	topology	size replaced/total	accuracy	accuracy	t-size	replaced-size	
MNIST	None			99.07			
	MNIST-Man	104K/127K	99.35	99.45	0.38	0.24	(4.2X)
SVHN	Maxout NiN	527K/1.6M	98.10	97.70	0.70	0.10	(10X)
	SVHN-Man	88K/105K	97.10	96.24	0.43	0.29	(3.4X)
CIFAR 10	VGG-16-D	7.4M/15M	92.32	92.94	0.52	0.0179	(55X)
	WRN-40-4	4.7M/9M	95.09	95.42	0.66	0.37	(2.7X)
CIFAR 100	VGG-16-D	7.4M/15M	68.86	70.68	0.52	0.0176	(57X)
ImageNet	AlexNet	59M/61M	57.20	57.20	0.08	0.0438	(23X)

Table 2: A summary of the highest classification accuracy achieved by replacing the deepest layers of common topologies (vanilla) with a learned structure. For the vanilla network, "size" corresponds to the number of trainable parameters in the deepest layers-to-be-replaced/entire-network. For the learned structures, "t-size" is the total network size (vanilla - replaced layers size + learned structure) normalized by the vanilla network size; "replaced-size" is the size of the learned structure normalized by the size of the vanilla network section it replaced (size reduction ratio). The first row corresponds to learning a structure directly from the MNIST images.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Adams, Ryan, Wallach, Hanna, and Ghahramani, Zoubin. Learning the structure of deep sparse graphical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 1–8, 2010.

Asbeh, Nuaman and Lerner, Boaz. Learning latent variable models by pairwise cluster comparison part ii- algorithm and evaluation. *Journal of Machine Learning Research*, 17(224):1–45, 2016.

Baker, Bowen, Gupta, Otkrist, Naik, Nikhil, and Raskar, Ramesh. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Chang, Jia-Ren and Chen, Yong-Sheng. Batch-normalized maxout network in network. *arXiv preprint arXiv:1511.02583*, 2015.

Chickering, David Maxwell. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

Chollet, François. keras. `https://github.com/fchollet/keras`, 2015.

Collobert, R., Kavukcuoglu, K., and Farabet, C. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

Cooper, Gregory F and Herskovits, Edward. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

Dash, D. and Druzdzel, M. Robust independence testing for constraint-based learning of causal structure. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pp. 167–174, 2003.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Elidan, Gal, Lotner, Noam, Friedman, Nir, and Koller, Daphne. Discovering hidden variables: A structure-based approach. In *Advances in Neural Information Processing Systems*, pp. 479–485, 2001.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Miikkulainen, Risto, Liang, Jason, Meyerson, Elliot, Rawal, Aditya, Fink, Dan, Francon, Olivier, Raju, Bala, Navruzyan, Arshak, Duffy, Nigel, and Hodjat, Babak. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

Murphy, K. The Bayes net toolbox for Matlab. *Computing Science and Statistics*, 33:331–350, 2001.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5, 2011.

Paige, Brooks and Wood, Frank. Inference networks for sequential Monte Carlo in graphical models. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *JMLR*, 2016.

Pearl, Judea. *Causality: Models, Reasoning, and Inference*. Cambridge university press, second edition, 2009.

Real, Esteban, Moore, Sherry, Selle, Andrew, Saxena, Saurabh, Suematsu, Yutaka Leon, Le, Quoc, and Kurakin, Alex. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

Ripley, Brian D. *Pattern recognition and neural networks*. Cambridge university press, 2007.

Silva, Ricardo, Scheine, Richard, Glymour, Clark, and Spirtes, Peter. Learning the structure of linear latent variable models. *Journal of Machine Learning Research*, 7(Feb):191–246, 2006.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Spirtes, P., Glymour, C., and Scheines, R. *Causation, Prediction and Search*. MIT Press, 2nd edition, 2000.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Stuhlmüller, Andreas, Taylor, Jacob, and Goodman, Noah. Learning stochastic inverses. In *Advances in neural information processing systems*, pp. 3048–3056, 2013.

Yehezkel, Raanan and Lerner, Boaz. Bayesian network structure learning by recursive autonomy identification. *Journal of Machine Learning Research*, 10(Jul):1527–1570, 2009.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.

Appendix

A Recursive Latent Structure Learning

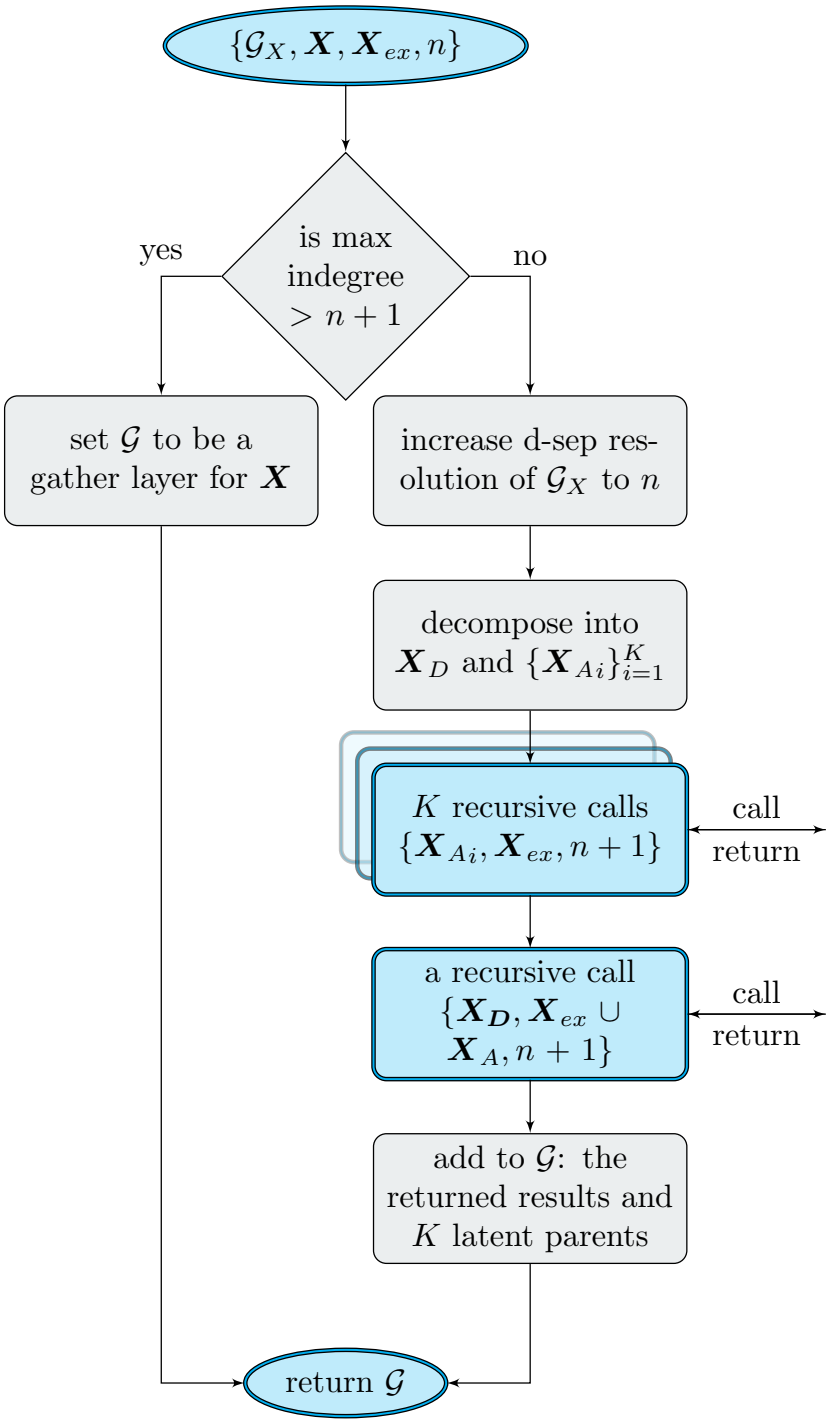


Figure 4: A flow chart of our algorithm.

B Extended Empirical Evaluation

We evaluate the quality of the learned structure in two experiments:

- Classification accuracy as a function of network depth and size for a structure learned directly from MNIST pixels.
- Classification accuracy as a function of network size on a range of benchmarks and compared to common topologies.

All the experiments were repeated five times where average and standard deviation of the classification accuracy were recorded. In all of our experiments, we used a ReLU function for activation, ADAM (Kingma & Ba, 2015) for optimization, and applied batch normalization (Ioffe & Szegedy, 2015) followed by dropout (Srivastava et al., 2014) to all the dense layers. All optimization hyper-parameters that were tuned for the vanilla topologies were also used, without additional tuning, for the learned structures. For the learned structures, all layers were allocated an equal number of neurons. Threshold for independence tests, and the number of neurons-per-layer were selected by using a validation set. Only test-set accuracy is reported.

Our structure learning algorithm was implemented using the Bayesian network toolbox (Murphy, 2001) and Matlab. We used Torch7 (Collobert et al., 2011) and Keras (Chollet, 2015) with the TensorFlow (Abadi et al., 2015) back-end for optimizing the parameters of both the vanilla and learned structures.

B.1 Network Depth, Number of Parameters, and Accuracy

We analyze the accuracy of structures learned by our algorithm as a function of the number of layers and parameters. Although network depth is automatically determined by the algorithm, it is implicitly controlled by the threshold used to test conditional independence (partial-correlation test in our experiments). For example, a high threshold may cause detection of many independencies leading to early termination of the algorithm and a shallow network (a low threshold has the opposite effect). Thus, four different networks having 2, 3, 4, and 5 layers, using four different thresholds, are learned for MNIST. We also select three configurations of neurons-per-layer: a baseline (normalized to 100%), and two configurations in which the number of neurons-per-layer is 50%, and 37.5% of the baseline network (equal number of neurons are allocated for each layer).

Classification accuracies are summarized in Table 3. When the number of neurons-per-layers is large enough (100%) a 3-layer network achieves the highest classification accuracy of 99.07% (standard deviation is 0.01) where a 2-layer dense network has only a slight degradation in accuracy, 99.04%. For comparison, networks with 2 and 3 fully connected layers (structure is not learned) with similar number of parameters achieve 98.4% and 98.75%, respectively. This demonstrates the efficiency of our algorithm when learning a structure having a small number of layers. In addition, for a smaller neuron allocation (50%), deeper structures learned by our algorithm have higher accuracy than shallower ones. However, a decrease in the neurons-per-layer allocation has a greater impact on accuracy for deeper structures.

neurons per layer	2 layers	3 layers	4 layers	5 layers
100%	99.04	99.07	99.07	99.07
50%	98.96	98.98	99.02	99.02
37.5%	98.96	98.94	98.93	98.93

Table 3: Classification accuracy [%] of structures learned from MNIST images as a function of network depth and number of neurons-per-layer (normalized). For comparison, when a structure is not learned, networks with 2 and 3 dense layers, achieve 98.4% and 98.75% accuracy, respectively (having the same size as learned structures at 100% neurons-per-layer).

B.2 Learning the Structure of the Deepest Layers in Common Topologies

We evaluate the quality of learned structures using five image classification benchmarks. We compare the learned structures to common topologies (and simpler hand-crafted structures), which we call "vanilla topologies", with respect to network size and classification accuracy. The benchmarks and vanilla topologies are described in Table 1. In preliminary experiments we found that, for SVHN and ImageNet, a small subset of the training data is sufficient for learning the structure (larger training set did not improve classification accuracy). As a result, for SVHN only the basic training data is used (without the extra data), i.e., 13% of the available training data, and for ImageNet 5% of the training data is used. Parameters were optimized using all of the training data.

Convolutional layers are powerful feature extractors for images exploiting domain knowledge, such as spatial smoothness, translational invariance, and symmetry. We therefore evaluate our algorithm by using the first convolutional layers of the vanilla topologies as "feature extractors" (mostly below 50% of the vanilla network size) and learning a deep structure from their output. That is, the deepest layers of the vanilla network (mostly over 50% of the network size; 64% on average) is removed and replaced by a structure learned by our algorithm in an unsupervised manner. Finally, a softmax layer is added and the entire network parameters are optimized.

First, we demonstrate the effect of replacing a different amount of the deepest layers and the ability of the learned structure to replace feature extraction layers. Table 4 describes classification accuracy achieved by replacing a different amount of the deepest layers in VGG-16-D. For example, column "conv.10" represents learning a structure using the activations of conv.10 layer. Accuracy and the normalized number of network parameters are reported for the overall network, e.g., up to conv.10 + the learned structure. Column "vanilla" is the accuracy achieved by the VGG-16-D network, after training under the exact same setting (a setting we found to maximize a validation-set accuracy for the vanilla topologies).

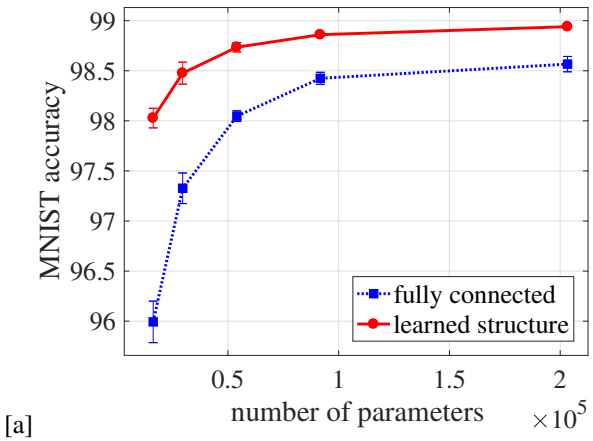
		learned				vanilla
		conv.5	conv.7	conv.10	classifier	–
CIFAR 10	accuracy	90.6	*92.61*	**92.94**	92.79	92.32
	# parameters	0.10	0.15	0.52	0.98	1.00
CIFAR 100	accuracy	63.17	*68.91*	**70.68**	69.14	68.86
	# parameters	0.10	0.13	0.52	0.98	1.00

Table 4: Classification accuracy (%) and overall network size (normalized number of parameters). VGG-16-D is the "vanilla" topology. For both, CIFAR 10/100 benchmarks, the learned structure achieves the highest accuracy by replacing all the layers that are deeper than layer conv.10. Moreover, accuracy is maintained when replacing the layers deeper than layer conv.7.

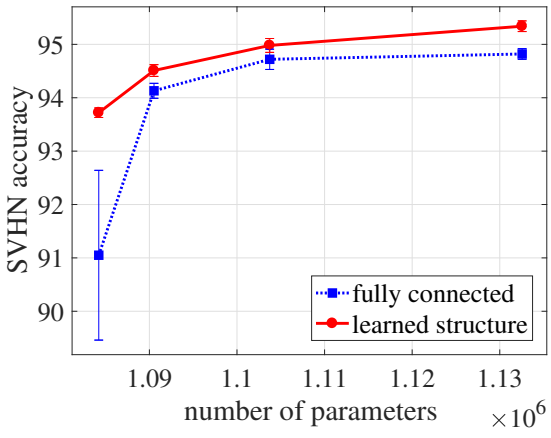
One interesting phenomenon to note is that the highest accuracy is achieved at conv. 10 rather than at the "classifier" (the last dense layer). This might imply that although convolutional layers are useful at extracting features directly from images, they might be redundant for deeper layers. By using our structure learning algorithm to learn the deeper layers, accuracy of the overall structure increases with the benefit of having a compact network. An accuracy, similar to that of "vanilla" VGG-16-D, is achieved with a structure having 85% less total parameters (conv. 7) than the vanilla network, where the learned structure is over 50X smaller than the replaced part.

Next, we evaluate the accuracy of the learned structure as a function of the number of parameters and compare it to a densely connected network (fully connected layers) having the same depth and size. For SVHN, we used the Batch Normalized Maxout Network in Network topology (Chang & Chen, 2015) and removed the deepest layers starting from the output of the second NiN block (MMLP-2-2). For CIFAR-10, we used the VGG-16-D and removed the deepest layers starting from the output of conv.10 layer. For MNIST, a structure was learned directly from pixels. Results are depicted in Figure 5. It is evident that accuracy of the learned structures is significantly higher (error bars represent 2 standard deviations) than a set of fully connected layers, especially in cases where the network is limited to a small number of parameters.

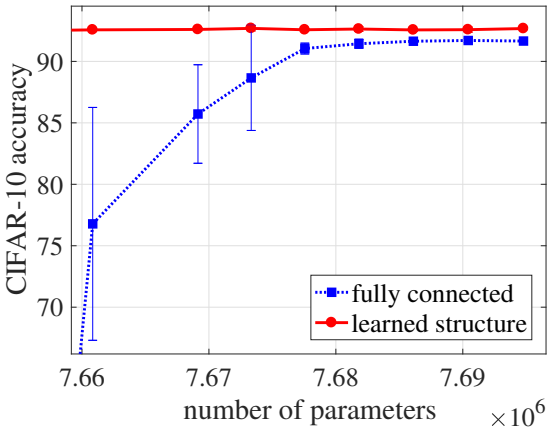
Finally, in Table 2 we provide the highest classification accuracies, achieved without limiting the size of the learned structure (selected using a validation set). In the first row, a structure is learned directly from images; therefore, it does not have a "vanilla" topology as reference (a network with 3 fully-connected layers having similar size achieves 98.75% accuracy). In all the cases, the size of the learned structure is significantly smaller than the vanilla topology, and generally has an increase in accuracy.



[a]



[b]



[c]

Figure 5: Accuracy as a function of network size. [a] MNIST, [b] SVHN. [c] CIFAR-10. Error bars represent 2 standard deviations.