Neurally-Guided Structure Inference

Sidi Lu [*1] **Jiayuan Mao** [*2,3] **Joshua B. Tenenbaum** [2,4,5] **Jiajun Wu** [2]

Abstract

Most structure inference methods either rely on exhaustive search or are purely data-driven. Exhaustive search robustly infers the structure of arbitrarily complex data, but it is slow. Data-driven methods allow efficient inference, but do not generalize when test data have more complex structures than training data. In this paper, we propose a hybrid inference algorithm, the Neurally-Guided Structure Inference (NG-SI), keeping the advantages of both search-based and data-driven methods. The key idea of NG-SI is to use a neural network to guide the hierarchical, layer-wise search over the compositional space of structures. We evaluate our algorithm on two representative structure inference tasks: probabilistic matrix decomposition and symbolic program parsing. It outperforms data-driven and search-based alternatives on both tasks.

1. Introduction

At the heart of human intelligence is the ability to infer the structural representation of data. Looking at hand-written digits from the MNIST dataset (LeCun et al., 1998), we humans effortlessly group them by digits and extract key features such as angles and thickness of strokes. Throughout the years, researchers have developed many practically useful compositional structures. A non-exhaustive list includes low-rank factorization (Mnih & Salakhutdinov, 2008), clustering, co-clustering (Kemp et al., 2006), binary latent factors (Ghahramani & Griffiths, 2006), sparse coding (Olshausen & Field, 1996; Berkes et al., 2008), and dependent GSM (Karklin & Lewicki, 2009).

An emerging research topic is to discover appropriate structures from data automatically (Kemp & Tenenbaum, 2008; Grosse et al., 2012). These methods are motivated by a key observation: structures can be expressed as hierarchical compositions of primitive components. Given the set of primitives and the production rules for composition (a.k.a. a domain-specific language), they use various inference algorithms to find the appropriate structure.

These inference algorithms can be roughly divided into two categories: search-based and data-driven. Search-based algorithms (Figure 1a) look for the best structure by an exhaustive search over the compositional space of possible structures. Candidate structures are ranked by expert-designed metrics. Such search routines are robust in inferring structures of arbitrary complexity, but they can be prohibitively slow for complex structures.

By contrast, data-driven algorithms (Figure 1b) learn to infer structures based on annotated data. Among them, deep neural networks enable efficient amortized inference: by learning from past inferences, future inferences run faster. Nevertheless, data-driven methods tend to overfit to training examples and fail to generalize to test data with more complex structures.

In this paper, we propose the Neurally-Guided Structure Inference (NG-SI, see Figure 1c), a hybrid inference algorithm that integrates the advantages of both search-based and data-driven approaches. In particular, it uses a neural network learning to guide a hierarchical, layer-wise search process. For each layer, the neural guider outputs a probability distribution over all possible production rules that can be applied. Based on this ranking, only a small number of rules are evaluated. This remarkably reduces the number of nodes in the search tree.

We evaluate NG-SI on two representative structure inference tasks: probabilistic matrix decomposition and symbolic program parsing. It outperforms data-driven and search-based alternatives in both inference robustness and efficiency.

2. Structure Inference

In this section, we formally introduce the task of inferring hierarchical structures from data.

[*]Equal contribution [1]Shanghai Jiao Tong University, [2]MIT CSAIL, [3]IIIS, Tsinghua University, [4]Department of Brain and Cognitive Sciences, MIT, [5]Center for Brains, Minds and Machines (CBMM), MIT. Correspondence to: Sidi Lu <steve_lu@apex.sjtu.edu.cn>, Jiajun Wu <jiajunwu@mit.edu>.

Project Page: http://ngsi.csail.mit.edu.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

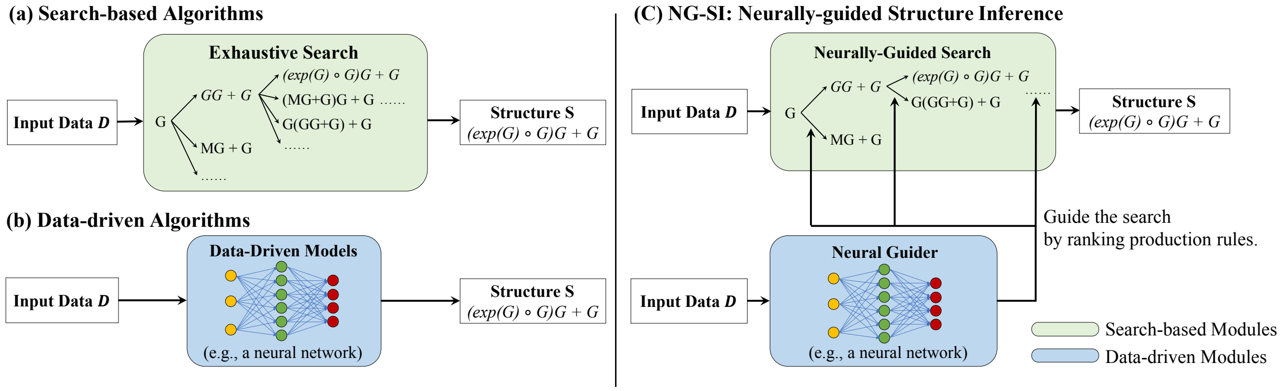


Figure 1. The illustrative flowcharts for (a) search-based algorithms, (b) data-driven algorithms, and (c) the proposed NG-SI. NG-SI uses a neural network (a data-driven module) to guide a hierarchical, layer-wise search process. It outperforms search-based and data-driven alternatives in both inference robustness and efficiency.

2.1. Problem Formulation

We want to jointly infer the structure S and its associated parameters T that best explain the observed data D. As a motivating example, the collection of hand-written digits D from the MNIST dataset can be clustered by their digits or the thickness of their strokes; they form the parameter space of T. The structure S here is "clustering": $MG + G$, where the M is a multinomial variable modeling the cluster labels, and the G's are Gaussian variables modeling the cluster centers and the data noise. Here S and T form the structural representation of D.

Simple structures can be hierarchically composed into a more complex one. Figure 2 shows an illustrative inference of the structure $S = \text{(MG + G)G + G}$ from the data matrix D (Grosse et al., 2012). The matrix has a low-rank structure in columns, while the first factorized component has a finer structure of row clustering. We use a domain-specific language (DSL) to represent the compositional space of possible structures $\{S\}$. Throughout this paper, we assume the existence of a context-free grammar for the DSL. Beginning from the start symbol, we can apply arbitrary production rules over non-terminal symbols in any order to generate syntactically correct structures. The terminal symbols in the context-free grammar represent the primitive concepts in the domain, such as cluster labels or centers.

2.2. Prior Work

The history of structure inference dates back to Vitanyi & Li (1997), where researchers discussed representing data as (program, input) pairs. Below, we discuss recent progress on structure inference. Existing methods mostly fall into two categories: search-based and data-driven.

Search-based structure inference. Search-based algorithms (Figure 1a) infer the structure of data with an exhaustive search over the compositional space of possible structures. The compositional space for structures is defined by domain experts. Kemp & Tenenbaum (2008) composed a graph where the edges represent the dependencies between data entries. Grosse et al. (2012) proposed a context-free grammar for probabilistic matrix decomposition. The grammar is powerful enough to encompass a large collection of Bayesian models for matrices: binary latent factors, dependent GSM, etc. Off-the-shelf matrix decomposition algorithms are used for data decomposition, while a posterior marginal likelihood is used to rank candidate structures. It also enables a hierarchical search process for structure inference, which significantly reduces the size of the searching space. Duvenaud et al. (2013) defined a grammar for generating kernels for Gaussian process models, while Lloyd et al. (2014) generated natural language descriptions of time series data. These frameworks require an enumeration of structures or production rules, followed by an evaluation based on the input to rank candidate structures. Thus, they are too slow in inferring complex structures on large-scale data. In this paper, we tackle this issue by introducing a data-driven module that learns to guide the hierarchical, layer-wise search process.

$$\begin{bmatrix}0&1&1&1\\1&1&1&0\\0&1&1&1\end{bmatrix} \overset{\text{Low-rank}}{=\!=} \underbrace{\begin{bmatrix}0&1&1\\1&1&0\\0&1&1\end{bmatrix} \times \begin{bmatrix}1&0&0&0\\0&1&1&0\\0&0&0&1\end{bmatrix}}_{\text{Structure: GG + G}}$$

$$\overset{\text{Clustering}}{=\!=} \underbrace{\left(\begin{bmatrix}1&0\\0&1\\1&0\end{bmatrix} \times \begin{bmatrix}0&1&1\\1&1&0\end{bmatrix}\right) \times \begin{bmatrix}1&0&0&0\\0&1&1&0\\0&0&0&1\end{bmatrix}}_{\text{Structure: (MG + G)G + G}}$$

Figure 2. An example of the hierarchical matrix decomposition.

The search-based inference is usually performed in a recursive way (Grosse et al., 2012; Kemp & Tenenbaum, 2008). An illustrative example is shown in Figure 2. First, the algorithm expands the start symbol of the grammar with all available production rules; it then decomposes the input data into multiple components, where each component corresponds to one symbol in each of the production rules (e.g., performs low-rank factorization on the original matrix). These components can be further recursively decomposed (e.g., performs clustering on the first component matrix). An expert-designed metric is applied to rank all candidate structures after each expansion. In practice, only top-ranked candidate structures are further expanded. Such algorithms are robust w.r.t. the complexity of the structures, but are slow for large-scale data.

Data-driven structure inference. Data-driven algorithms (Figure 1b) infer the structure of data by learning from annotated data. Deep neural networks enable efficient amortized inference: after learning from past inferences, they can efficiently recognize similar structures for test data. Structure inference on sequential data includes *sequence labelling*, assigning categorical labels (e.g., part-of-speech tags) for each item in the sequence (Ma & Hovy, 2016), *parsing*, generating graphical structures of input sequences such as dependency trees (Yih et al., 2014; Chen & Manning, 2014), and *sequence-to-sequence translation* (Sutskever et al., 2014), which has been further extended to sequence-to-tree learning (Dong & Lapata, 2016) and tree-to-tree learning (Chen et al., 2018b). This line of research has been adapted to applications such as program synthesis (Parisotto et al., 2017; Reed & De Freitas, 2016). As an representative example, *sequence-to-sequence* model (Sutskever et al., 2014) generates the entire structure symbol by symbol. However, the symbolic and compositional nature of structures has brought a remarkable difficulty to data-driven approaches: they tend to overfit to training examples and fail to generalize to test data with more complex structures. For example, in sequence encoding, recurrent networks easily fail when tested on longer sequences. In general, data-driven approaches are more efficient but less robust than search-based alternatives.

Guided search. Data-driven models such as neural networks can be used to improve the efficiency of search-based inference. Beneš et al. (2011) first implemented this idea in procedural modeling, where the task is to reconstruct visual data such as objects or textures using a set of generative rules. In Beneš et al. (2011), external data-driven models are used to improve the stability of the procedural modeling. Similarly, Ritchie et al. (2016) proposed to embed neural modules into the stochastic algorithm. The neural modules are trained to maximize the likelihood of the outputs generated by sequential Monte Carlo. Menon et al. (2013) and Devlin et al. (2017) proposed to use data-driven methods for program induction. Many algorithms have been proposed to improve the guided search by introducing type information (Osera & Zdancewic, 2015), learning from multiple demonstrations (Sun et al., 2018), using execution-guided inference (Chen et al., 2019; Tian et al., 2019), predicting attributes of programs (Balog et al., 2017; Ellis et al., 2018b), predicting sketches (Lezama, 2008), or inducing subroutines from existing programs (Ellis et al., 2018a). The key difference between our approach and prior works lies in our exploitation of the compositionality in structure inference: we use neural networks to amortize the per-layer inference and keep the hierarchical search process. Each production rule is associated with an expert-designed algorithm to decompose the data into multiple components, and the same inference algorithm could be recursively applied to each of the components. This helps the generalization of the amortized inference.

Algorithm 1 Neurally-Guided Structure Inference

```
Function Infer($D$, $Type$):
    $rule \leftarrow$ SelectRule($D, Type$)
    for each non-terminal symbol $s$ in $rule$ do
        $C_s \leftarrow$ DecomposeData($D, rule, s$)
        Replace $s$ in $rule$ with Infer($C_s$, $s$)
    return $rule$
```

3. Neurally-Guided Structure Inference

In this paper, we propose the Neurally-Guided Structure Inference (NG-SI, see Figure 1c), a hybrid inference algorithm that integrates the advantages of both search-based and data-driven approaches. Algorithm 1 shows the pseudocode. The algorithm builds the hierarchical structure by recursively choosing the production rule to expand a non-terminal symbol. Meanwhile, a decomposition algorithm, `DecomposeData`, decomposes the input data D into several components ($\{C_s\}$), whose structure can be recursively inferred by the `Infer` function. For simplicity, in Algorithm 1, only one production rule is applied to a non-terminal symbol (determined by `SelectRule`). One can also extend this greedy algorithm to the beam search-based inference or other variants.

We implement `SelectRule` as a neural network, namely, the neural guider. The neural guider enables efficient amortized inference at a layer level: it learns to predict the best production rule for a non-terminal symbol in the structure. By contrast, purely search-based approaches need to evaluate all possible rules to select the best one.

We illustrate this idea by a running example of matrix decomposition based on the DSL introduced by Grosse et al. (2012). Consider the simple matrix shown in Figure 2. Note that the matrix does not have a full column rank. Thus, at the first step, the neural guider should select the “low-rank factorization” rule and decompose the input data into two components. Next, note that the first decomposed matrix has a clustering structure: all row vectors can be grouped around two centers. Thus, the neural guider should select the

Low-rank	G	$\rightarrow$	$GG + G$
Clustering	G M	$\rightarrow$ $\rightarrow$	$MG + G \mid GM^T+G$ $MG + G$
Markov chains	G	$\rightarrow$	$CG + G \mid GC^T + G$
Gaussian scales	G	$\rightarrow$	$\exp(G) \circ G$
Binary factors	G B	$\rightarrow$ $\rightarrow$	$BG + G \mid GB^T + G$ $BG + G$
Misc.	M	$\rightarrow$	B

Table 1. Production rules in the context-free DSL for structure inference. The left-most column indicates the type of the production rules.

"clustering" rule at the second step. Such recursive inference may continue for data with more complex structures.

Recursion lies at the core of NG-SI. First, recursion reduces the infinitely compositional search space of structures to the space of available production rules. It also ensures generalization to arbitrarily complex data. Moreover, recursion is a critical inductive bias for the data-driven neural guider. The amortized inference happens at the layer level instead of at the full structure level. This ensures efficient structure inference while allowing combinatorial generalization.

Below, we demonstrate the advantages of our formulation on two representative structure inference tasks: probabilistic matrix decomposition (Section 4) and symbolic program parsing (Section 5).

4. Study I: Matrix Decomposition

With the proliferation of structured probabilistic models such as binary latent factors (Ghahramani & Griffiths, 2006) and sparse coding (Olshausen & Field, 1996; Berkes et al., 2008), people are getting more interested in the discovery of structures from data. In this section, we revisit the problem of structure inference for matrices (Grosse et al., 2012).

4.1. Problem Formulation

We formally introduce the DSL for structural matrix decomposition via a motivating example. One of the simplest structures—Bayesian clustering—can be written as $F = MG + G$. Here, the symbol M stands for a multinomial matrix, whose rows are sampled identically from a multinomial distribution. This matrix M is multiplied by a Gaussian matrix G, whose rows are identically sampled from a Gaussian distribution. The conceptual meaning of the structure $MG + G$ could be interpreted as such: the row vector of the matrix stands for a stochastic choice of the cluster label, and the parameters of the multinomial distribution represent the probability of choosing each cluster. The first Gaussian matrix represents the center of each cluster. The last Gaussian matrix captures the i.i.d. Gaussian noise. Other types of matrices in the DSL include C (time-series Cumulative matrices) and B (Binomial matrices).

Grosse et al. (2012) proposed a context-free grammar to describe the structures of data matrices. Table 1 shows all production rules in the grammar. The inference always starts from a single symbol G, i.e., assuming all of the data items in the matrix to be i.i.d. Gaussian.

4.2. Method

The search-based algorithm Grosse et al. (2012) requires evaluating all possible rules and ranking them with a proper metric to choose the one to be applied. Intuitively, each production rule has a specific pattern of its matrices to be decomposed. For example, the data matrices having the "clustering" structure may have different patterns with matrices having the "low-rank" structure. Based on such observation, in NG-SI, we adopt a Convolutional Neural Network (CNN) as a neural guider for the neurally-guided search. Requiring only limited synthetic data during training, the neural guider remarkably outperforms the algorithm-based exhaustive search of production rules in inference efficiency.

In detail, NG-SI infers structures in a recursive manner following the paradigm shown as Algorithm 1. At each step, the neural guider and the symbolic decomposition algorithm work jointly: given the input data D, the neural guider predicts the production rule to be applied, and the symbolic decomposition algorithm decomposes the input data into multiple components. We use the decomposition algorithms and the metrics for ranking candidate structures in Grosse et al. (2012).

Neural guider. The neural guider is trained in the following way. Given a matrix G, the neural guider learns to distinguish a finer structure from all possible candidates, including $GG+G$, $MG+G$, GM^T+G, $CG+G$, GC^T+G, $BG + G$, $GB^T + G$ and $\exp(G) \circ G$.

Training. We generate synthetic data for training the neural guider on the fly. For each data point, we first randomly sample a production rule (e.g., $G \rightarrow MG + G$) from the DSL. Then, we randomly generate a data matrix following the production rule (e.g., generate a multinomial matrix and two Gaussian matrices and compose them together) and use the underlying production rule as the label. The neural guider is trained to convergence on this dataset. It is then fixed for structure inference.

Hyperparameters. We build our neural guider as an 8-layer convolutional neural network. The detailed architecture of the neural guider can be found in the supplementary material. To handle matrices of arbitrary sizes, we always pad the input matrices to 200 × 200 by adding zero entries. We also stack the padded input matrix, along the channel

Data	Ground Truth	Grosse et al. (2012)	Mat2Seq	NG-SI (ours)
Low-rank	$GG+G$	Correct	Correct	Correct
Clustering	$MG+G$	Correct	Correct	Correct
Binary latent features	$BG+G$	Correct	Correct	Correct
Co-clustering	$M(GM^T+G)+G$	Correct	$MG+G$	Correct
Binary matrix factorization	$B(GB^T+G)+G$	Correct	$BG+G$	Correct
BCTF	$(MG+G)(GM^T+G)+G$	Correct	$BG+G$	Correct
Sparse coding	$s(G)G+G$	Correct	$s(G)$	Correct
Dependent GSM	$s(GG+G)G+G$	$s(G)G+G$	$s(G)$	$s(G)G+G$
Random walk	$CG+G$	Correct	Correct	Correct
Linear dynamical system	$(CG+G)G+G$	Correct	$CG+G$	$B(s(G)G+G)+G$
Motion Capture - Level 1	-	$CG+G$	$CG+G$	$CG+G$
Motion Capture - Level 2	-	$C(GG+G)+G$	$CG+G$	$C(GG+G)+G/CG+G$
Image Patch - Level 1	-	$GG+G$	$GG+G$	$GG+G$
Image Patch - Level 2	-	$s(G)G+G$	$GG+G$	$s(G)G+G$
Image Patch - Level 3	-	$s(GG+G)G+G/s(G)G+G$	$GG+G$	$s(G)G+G$

Table 2. Search results of our approach and several baselines. We denote $\exp(f(G)) \circ G$ as $s(f(G))$. We run each model for three times with different random seeds. If the results produced by three runs disagree, we include all possible outcomes in the table, separated by $/$. See the main text for a detailed analysis.

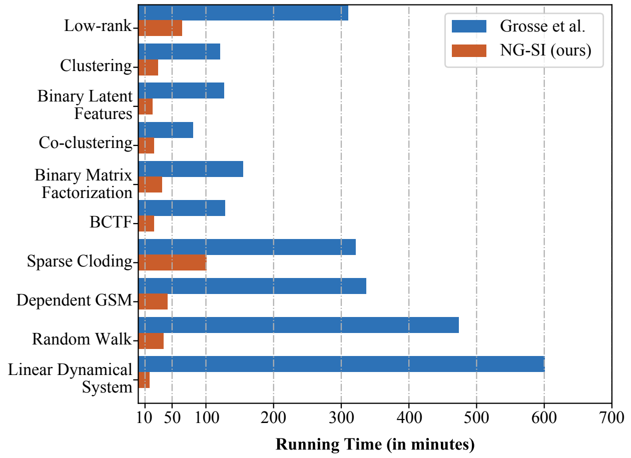


Figure 3. We compare the running time for structure inference of (Grosse et al., 2012) and NG-SI. For all experiments, we set the standard deviation for the input noise to 1. Remarkably, the speed of NG-SI is 3× $\sim$ 39× faster than the baseline.

dimension, with a padding indicator matrix P: if the value at position (i, j) belongs to the original input matrix, then $P[i, j] = 1$; otherwise, $P[i, j] = 0$.

We train the model with the Adam optimizer (Kingma & Ba, 2015). The hyperparameters for the optimizer are set to be $\beta_1 = 0.9, \beta_2 = 0.9, \alpha = 10^{-4}$. The model is trained for 100,000 iterations, with a batch size of 100.

4.3. Experiments

To evaluate the accuracy of our approach, we replicate the experiments in Grosse et al. (2012), including one synthetically generated dataset and two real-world datasets: motion capture and image patches. For the synthetic dataset, we generated matrices of size 200 × 200 from 10 models listed in Table 2. All models have a hidden dimension of 10, following Grosse et al. (2012). The dataset of human motion capture (Hsu et al., 2005; Taylor et al., 2007) consists of a person walking in a variety of styles. Each row of the data matrix describes the human pose in one frame, in the form of the person's orientation, displacement, and joint angles. The natural image patches dataset contains samples from the Sparsenet dataset proposed in Olshausen & Field (1996). It contains 10 images of natural scenes (smoothed and whitened), from which 1,000 patches of size 12 × 12 are selected and flattened as the rows of the matrix. We study the inferred structure with different search depth limits, varying from level 1 to level 3. Note that there is no groundtruth structure for such real-world datasets.

Baselines. Beside the search-based baseline in Grosse et al. (2012), we also implement a simple matrix-to-sequence model as a data-driven baseline. This model takes the data matrix as input and generates the structure of the matrix using a CNN-GRU model (Vinyals et al., 2015b). The baseline is trained on the same data as our neural guider.

Accuracy. Shown in Figure 3 and Table 2, our model successfully finds most of the optimal structures of synthetic data except for two of them: the dependent GSM and the linear dynamical system. Remarkably, the search process is accelerated with a multiplier of 3× $\sim$ 39×. NG-SI also generalizes well to real-world datasets. The inferred structures on both real-world datasets are consistent with the structures inferred by the search-based baseline.

For dependent GSM matrices, $(\exp(GG+G) \circ G)G+G$, the final structure determined by NG-SI is the sparse coding model, $(\exp(G) \circ G)G+G$. This is a typical failure case as discussed in Grosse et al. (2012), since the variational lower-bound used for ranking candidate structures cannot distinguish two structures by a margin. We attribute the fail-

	Depth = 1	Depth = 2	Depth = 3
Grosse et al. (2012)	16min	1h 32min	5h 37min
NG-SI	2min	12min	43min
Speed up	8×	7.67×	7.83×

Table 3. Running time needed for the search of structures of different depths. Our algorithm consistently outperforms the baseline in efficiency. We ran all experiments on a machine with an Intel Xeon E5645 CPU and a GTX 1080 Ti GPU.

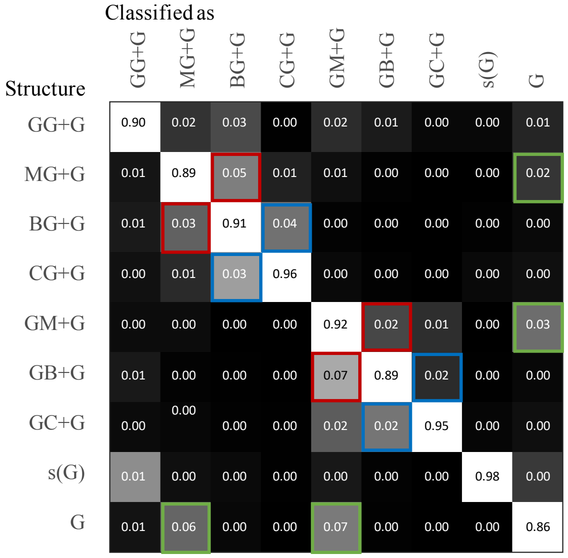
ure of the linear dynamical system case to the imbalance of training data: most production rules imply the independence of rows in the data matrix, while the rule $G \to CG+G$ does not. Thus, we see the misclassifications of structures that include this rule. In practice, the problem can be alleviated by techniques such as sampling more data with the $CG+G$ structure.

For real-world datasets, the structures inferred by NG-SI agree with the search-based baseline (Grosse et al., 2012) in most cases. Both methods show unstable results for complex structures; they sometimes fall back to a structure with a simpler but plausible form. We attribute this to the noises in real-world datasets, which affects the robustness of both the neural guider—for ranking production rules, and the variational lower-bound—for ranking inferred structures.

Efficiency. We empirically compare our algorithm against the original greedy search algorithm proposed by Grosse et al. (2012). We generate a 200 × 200 matrix from a dependent GSM model with 10 latent dimensions. Table 3 summarizes the running time needed for the search of structures of different depths. In general, our algorithm consistently speeds up the greedy search version by a factor of 8.

Rule similarity discovery. Our approach can be regarded as using an approximated probability distribution of structures conditioned on the input data to guide the structure search. Interestingly, we find that the learned distribution by the neural guider recovers the similarity between rules.

To visualize this, we first generate a dataset of matrices following different production rules, such as $G \to GG+G$ and $G \to MG+G$. All these rules start from a single matrix G. We then use the trained neural guider to predict the structure, i.e., the approximated probability distribution. We accumulate the output probabilities and visualize them as a matrix in Figure 4, where lighter entries indicate a higher probability of misclassification, or equivalently, a higher similarity between the two structures.



Structure \ Classified as	GG+G	MG+G	BG+G	CG+G	GM+G	GB+G	GC+G	s(G)	G
GG+G	0.90	0.02	0.03	0.00	0.02	0.01	0.00	0.00	0.01
MG+G	0.01	0.89	0.05	0.01	0.01	0.00	0.00	0.00	0.02
BG+G	0.01	0.03	0.91	0.04	0.00	0.00	0.00	0.00	0.00
CG+G	0.00	0.01	0.03	0.96	0.00	0.00	0.00	0.00	0.00
GM+G	0.00	0.00	0.00	0.00	0.92	0.02	0.01	0.00	0.03
GB+G	0.01	0.00	0.00	0.00	0.07	0.89	0.02	0.00	0.00
GC+G	0.00	0.00	0.00	0.00	0.02	0.02	0.95	0.00	0.00
s(G)	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00
G	0.01	0.06	0.00	0.00	0.07	0.00	0.00	0.00	0.86

Figure 4. Visualization of the similarities between production rules for expanding a "G" node. The similarities are implicitly learned by our neural guider. We manually highlight some entries indication pairs of similar production rules. □: Clustering vs. Binary factor, □: Binary factor vs. Markov Chain, and □: Clustering (mixture of Gaussian) vs. Gaussian.

Ideally, the matrix should be symmetric, as it reflects the similarity between each two of the production rules. The empirical results support this intuition. Moreover, it recovers the similarity between some production rules. For example, $MG+G$ and $BG+G$ are similar (due to the similar binary structure of M and B); $MG+G$ (mixture of Gaussian) and G (pure Gaussian) are similar. These findings are consistent with human intuition.

4.4. Application: Inspecting GANs

The generative adversarial network (GANs) learns a transformation function G_θ (the generator) from a specific distribution (e.g., Gaussian) to the target data distribution (Goodfellow et al., 2014). Such transformations are implemented as neural networks. Thus, it is usually difficult to interpret the generation process.

We view the generator of a GAN as a stack of distribution transformers, where each transformer is a single layer in the network. We show that it is possible to partially reveal the transformation process inside the GAN generator by detecting the structures of its intermediate features. By tracking these structures, we can obtain a better understanding of how GANs transform the distributions layer by layer.

As an example, we train an MLP-GAN to map a Gaussian distribution to randomly generated vectors from a set of dependent GSM distributions sampled from a common prior. The model is trained by the Wasserstein GAN-GP algorithm (Gulrajani et al., 2017). The generator is trained for 10,000 iterations, and before each, the discriminator is trained for 4 iterations. The architecture of the generator and the discovered structure of the intermediate features is summarized in Table 4. In general, the trace of the structures is consistent with the natural compositional structure of the dependent GSM: $G \to GG+G \to s(G)G+G \to s(GG+G)G+G$.

Gaussian noise z, dimension $= 128$, structure: G
Fully connected, dimension $= 256$ structure: G
ReLU nonlinearity
Fully connected, dimension $= 256$ structure: $MG + G$
ReLU nonlinearity
Fully connected, dimension $= 256$ structure: $GG + G \,/\, BG + G$
ReLU nonlinearity
Fully connected, dimension $= 256$ structure: $s(G)G + G$
ReLU nonlinearity
Fully connected, dimension $= 256$ structure: $s(G)G + G$ or $s(GG + G)G + G$

Table 4. The architecture and the discovered structures from the intermediate features of an MLP-GAN's generator. We run this experiment multiple times and show the top-ranked structures for each layer.

5. Study II: Program Parsing

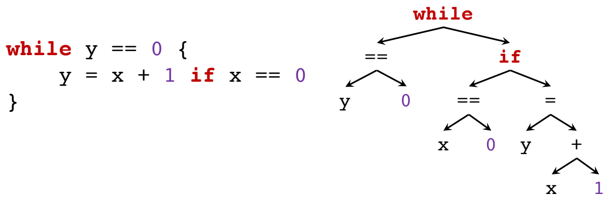
The framework of neurally-guided structure inference can be naturally extended to other domains. In this section, we consider the task of program parsing from sequential data: given a discrete program written in a programming language, we want to translate it into a symbolic abstract syntax tree.

5.1. Problem Formulation

For program parsing, we adopt the WHILE program language (Chen et al., 2018a) as our testbed. The WHILE language is defined by 73 production rules. It covers most of the functionality of modern programming languages: arithmetic expressions, variable assignments, conditions, and loops. Our goal is to translate a program written in the WHILE language into a hierarchical abstract syntax tree (AST). Figure 5 shows a sample code written in the WHILE language and its corresponding AST. Ideally, after learning from a limited number of programs and the AST labels, the algorithms should generalize to parse longer programs or more complex programs, i.e., programs with a deeper AST. As we show later, such generalizability is difficult for many data-driven algorithms such as sequence-to-sequence models (Sutskever et al., 2014).

5.2. Model

Following the same hybrid search paradigm as presented in Algorithm 1, we build a neurally-guided program parser. We keep the design of recursive inference: at each step, the neural guider infers the production rule based on the input code. Then we decompose the code into components (such as the conditions and the body for an if-statement). Thus, the data are represented as code strings. Here, we use hand-coded algorithms for code decomposition.



```
while y == 0 {
    y = x + 1 if x == 0
}
```

Figure 5. A sample code in the WHILE language (left) and its corresponding AST (right).

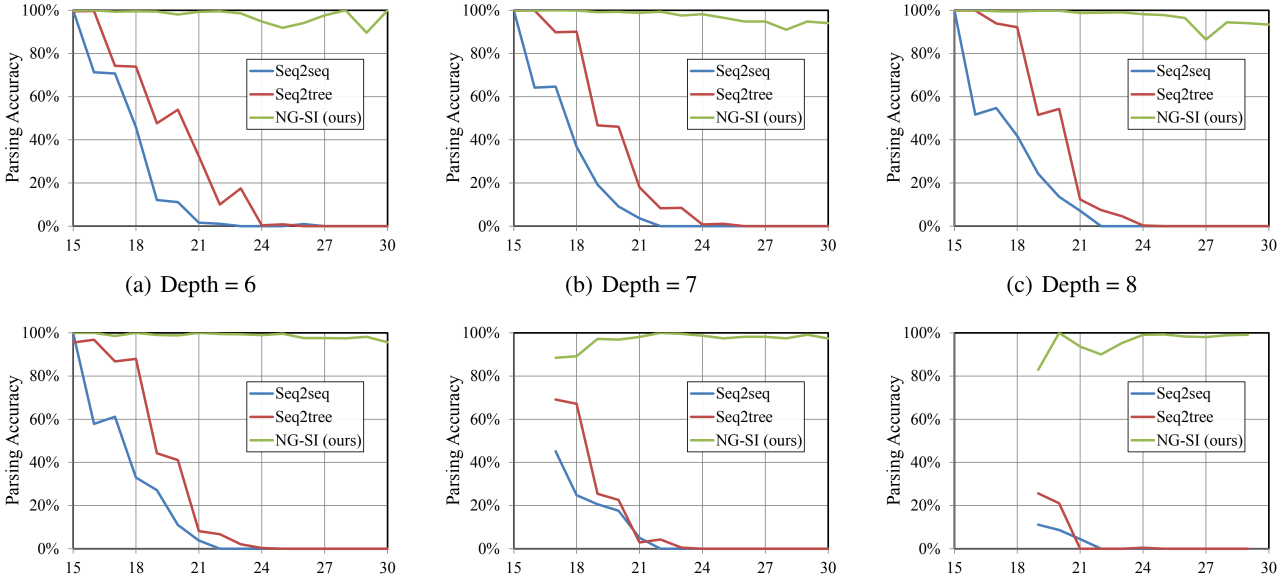
Because the neural guider determines only the top-level production rule for the input code, intuitively, it does not require specific architecture designs. In our experiments, we implement it as a combination of a GRU encoder and a linear classifier. The input code string is first fed into the GRU encoder; then the classifier takes the last hidden state of the encoded string as its input.

Training. We randomly generate training samples based on the context-free grammar of the WHILE language. Roughly, starting from the start symbol of the grammar, we randomly apply a number of production rules on non-terminal symbols. For each generated program, all of its sub-strings corresponding to one of the sub-trees in the AST are used as the training data. To test the generalizability of the models, we restrict the depth of the AST of the training examples and the length of the programs to be less than 9 and 15, respectively. The learned model is tested on longer programs or more complex programs (i.e., with a deeper AST).

Hyperparameters. We adopt a unidirectional GRU with a hidden dimension of 256 as the code string encoder for production rule selection. We train the model using the Adam optimizer, with hyperparameters $\beta_1 = 0.9, \beta_2 = 0.9, \alpha = 10^{-4}$. The batch size is set to 64. We perform curriculum learning similar to the training process described in Chen et al. (2018a), where the model is trained with programs of gradually increasing length and depth. The shortest program has a length of 5, while the longest has 15. We repeat the curriculum learning process three times for training the neural guider.

5.3. Experiments

Baselines. We implement a sequence to sequence (Seq2Seq) model (Sutskever et al., 2014) with attention and a Seq2Tree model (Dong & Lapata, 2016) model with attention as the data-driven baselines for AST inference.



(a) Depth = 6

(b) Depth = 7

(c) Depth = 8

(d) Depth = 9

(e) Depth = 10

(f) Depth = 11

Figure 6. The performance of NG-SI and the baselines (Seq2Seq and Seq2Tree) on test programs with different AST depths (6 to 11) and different lengths (15 to 30). Deep programs (depth $\geq$ 10) have a minimal length larger than 15: Figure 6(e) and Figure 6(f)

The Seq2Seq model with attention uses GRUs as the encoder and the decoder, both with a hidden dimension of 256. We slightly modify the output format of the Seq2Seq baseline to support the tree-structured output. Specifically, we perform a pre-order traversal of the AST and use the traversal order of all nodes as the label for training (Vinyals et al., 2015a).

The Seq2Tree model uses GRU as the encoder, with a hidden dimension of 256 and a hierarchical tree decoder. It generates the AST in a top-down manner. Starting from the starting symbol, the decoder iteratively expands a non-terminal symbol with a production rule predicted by the decoder network.

For training both baseline models, we use the teacher forcing method and the Adam optimizer with the same hyperparameters as the neural guider.

Results. Figure 6 shows the results. We evaluate the performance of NG-SI and the baseline models on programs with different AST depths (6 to 11) and different lengths (15 to 30). Figure 6 shows that NG-SI robustly infers the hierarchical AST from more complex and longer programs than training examples. By contrast, the performance of the purely data-driven baseline drops significantly as the complexity or the length of the program grows beyond training examples. It is worth noting that NG-SI is remarkably robust w.r.t. the program depth. Although the model has never seen programs with depth 11 during training, it achieves an accuracy $\geq 90\%$ during inference. In contrast, the accuracies of all baselines are $\leq 20\%$.

As for the running time, Seq2Seq, Seq2Tree, and our proposed NG-SI give prediction in less than 1s per instance. We also compare our model with an exhaustive search baseline. Specifically, it uses iterative deepening depth-first search for programs. The search stops when it finds an AST that exactly reconstructs the input program. The algorithm runs in a single thread on a machine with an Intel Core i7-8700 4.0GHz and 16G RAM, and the running time limit is 1 hour. With a fixed program depth = 6, the search-based baseline achieves perfect accuracy on programs with length $\leq$ 24 but exceeds the running time limit when length $>$ 24.

6. Conclusion

We have proposed the Neurally-Guided Structure Inference (NG-SI), a hybrid algorithm for structure inference that keeps the advantages of both search-based and data-driven approaches. The key idea is to use a neural network to learn to guide a hierarchical, layer-wise search process. The data-driven module enables efficient inference: it recursively selects the production rules to build the structure, so that only a small number of nodes in the search tree need to be evaluated. The search-based framework of NG-SI ensures robust inference on test data with arbitrary complexity. Results on probabilistic matrix decomposition benchmarks and program parsing datasets support our arguments.

Acknowledgements. We thank Xinyun Chen for helpful discussions and suggestions. This work was supported in part by the Center for Brains, Minds and Machines (CBMM, NSF STC award CCF-1231216), ONR MURI N00014-16-1-2007, and Facebook.

References

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., and Tarlow, D. Deepcoder: Learning to write programs. In *ICLR*, 2017. 3

Beneš, B., Šťava, O., Měch, R., and Miller, G. Guided procedural modeling. *CGF*, 30(2):325–334, 2011. 3

Berkes, P., Turner, R., and Sahani, M. On sparsity and overcompleteness in image models. In *NeurIPS*, 2008. 1, 4

Chen, D. and Manning, C. A fast and accurate dependency parser using neural networks. In *EMNLP*, 2014. 3

Chen, X., Liu, C., and Song, D. Towards synthesizing complex programs from input-output examples. In *ICLR*, 2018a. 7

Chen, X., Liu, C., and Song, D. Tree-to-tree neural networks for program translation. In *NeurIPS*, 2018b. 3

Chen, X., Liu, C., and Song, D. Execution-guided neural program synthesis. In *ICLR*, 2019. 3

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A.-r., and Kohli, P. Robustfill: Neural program learning under noisy i/o. In *ICML*, 2017. 3

Dong, L. and Lapata, M. Language to logical form with neural attention. In *ACL*, 2016. 3, 7

Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J. B., and Ghahramani, Z. Structure discovery in nonparametric regression through compositional kernel search. In *ICML*, 2013. 2

Ellis, K., Morales, L., Sablé-Meyer, M., Solar-Lezama, A., and Tenenbaum, J. Learning libraries of subroutines for neurally–guided bayesian program induction. In *NeurIPS*, 2018a. 3

Ellis, K., Ritchie, D., Solar-Lezama, A., and Tenenbaum, J. Learning to infer graphics programs from hand-drawn images. In *NeurIPS*, 2018b. 3

Ghahramani, Z. and Griffiths, T. L. Infinite latent feature models and the indian buffet process. In *NeurIPS*, 2006. 1, 4

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *NeurIPS*, 2014. 6

Grosse, R. B., Salakhutdinov, R., Freeman, W. T., and Tenenbaum, J. B. Exploiting compositionality to explore a large space of model structures. In *UAI*, 2012. 1, 2, 3, 4, 5, 6

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. Improved training of wasserstein gans. In *NeurIPS*, 2017. 6

Hsu, E., Pulli, K., and Popović, J. Style translation for human motion. In *ACM TOG*, pp. 1082–1089. ACM, 2005. 5

Karklin, Y. and Lewicki, M. S. Emergence of complex cell properties by learning to generalize in natural scenes. *Nat.*, 457(7225):83, 2009. 1

Kemp, C. and Tenenbaum, J. B. The discovery of structural form. *PNAS*, 105(31):10687–10692, 2008. 1, 2

Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T., and Ueda, N. Learning systems of concepts with an infinite relational model. In *AAAI*, 2006. 1

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. 1

Lezama, A. S. *Program synthesis by sketching*. PhD thesis, Citeseer, 2008. 3

Lloyd, J. R., Duvenaud, D. K., Grosse, R. B., Tenenbaum, J. B., and Ghahramani, Z. Automatic construction and natural-language description of nonparametric regression models. In *AAAI*, 2014. 2

Ma, X. and Hovy, E. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *ACL*, 2016. 3

Menon, A., Tamuz, O., Gulwani, S., Lampson, B., and Kalai, A. A machine learning framework for programming by example. In *ICML*, 2013. 3

Mnih, A. and Salakhutdinov, R. R. Probabilistic matrix factorization. In *NeurIPS*, 2008. 1

Olshausen, B. A. and Field, D. J. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nat.*, 381(6583):607, 1996. 1, 4, 5

Osera, P.-M. and Zdancewic, S. Type-and-example-directed program synthesis. In *ACM SIGPLAN Notices*. ACM, 2015. 3

Parisotto, E., Mohamed, A.-r., Singh, R., Li, L., Zhou, D., and Kohli, P. Neuro-symbolic program synthesis. In *ICLR*, 2017. 3

Reed, S. and De Freitas, N. Neural programmer-interpreters. In *ICLR*, 2016. 3

Ritchie, D., Thomas, A., Hanrahan, P., and Goodman, N. Neurally-guided procedural models: Amortized inference for procedural graphics programs using neural networks. In *NeurIPS*, 2016. 3

Sun, S.-H., Noh, H., Somasundaram, S., and Lim, J. Neural program synthesis from diverse demonstration videos. In *ICML*, 2018. 3

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *NeurIPS*, 2014. 3, 7

Taylor, G. W., Hinton, G. E., and Roweis, S. T. Modeling human motion using binary latent variables. In *NeurIPS*, 2007. 5

Tian, Y., Luo, A., Sun, X., Ellis, K., Freeman, W. T., Tenenbaum, J. B., and Wu, J. Learning to infer and execute 3d shape programs. In *ICLR*, 2019. 3

Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. Grammar as a foreign language. In *NeurIPS*, 2015a. 8

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. Show and tell: A neural image caption generator. In *CVPR*, 2015b. 5

Vitanyi, P. M. and Li, M. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Heidelberg, 1997. 2

Yih, W.-t., He, X., and Meek, C. Semantic parsing for single-relation question answering. In *ACL*, 2014. 3

Supplementary Materials for Neurally-Guided Structure Inference

Sidi Lu [*] [1] **Jiayuan Mao** [*] [2] [3] **Joshua B. Tenenbaum** [2] [4] [5] **Jiajun Wu** [2]

A. CNN Architecture of the Neural Guider for Matrix Decomposition

Table 1 shows the architecture of our CNN-based neural guider for probabilistic matrix decomposition.

Padding, output shape $= [200 \times 200 \times 2]$
3×3 Conv, with stride $= 1$, channels $= 64$, ReLU activation
4×4 Conv, with stride $= 2$, channels $= 64$, ReLU activation
4×4 Conv, with stride $= 2$, channels $= 128$, ReLU activation
Instance normalization
4×4 Conv, with stride $= 2$, channels $= 128$, ReLU activation
4×4 Conv, with stride $= 2$, channels $= 128$, ReLU activation
3×3 Conv, with stride $= 1$, channels $= 128$, ReLU activation
Instance normalization
4×4 Conv, with stride $= 2$, channels $= 256$, ReLU activation
4×4 Conv, with stride $= 2$, channels $= 512$, ReLU activation
3×3 Conv, with stride $= 1$, channels $= 512$, ReLU activation
Flatten
Fully connected + softmax

Table 1. Network architecture of the CNN-based neural guider.

[*] Equal contribution [1] Shanghai Jiao Tong University, [2] MIT CSAIL, [3] IIIS, Tsinghua University, [4] Department of Brain and Cognitive Sciences, MIT, [5] Center for Brains, Minds and Machines (CBMM), MIT. Correspondence to: Sidi Lu <steve_lu@apex.sjtu.edu.cn>, Jiajun Wu <jiajunwu@mit.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).