Learning Graphical Concepts

Kevin Ellis*
Department of Brain & Cognitive Sciences
MIT

Eyal Dechter
Department of Brain & Cognitive Sciences
MIT

Ryan P. Adams
School of Engineering & Applied Sciences
Harvard University

Joshua B. Tenenebaum
Department of Brain & Cognitive Sciences
MIT

Abstract

How can machine learning techniques be used to solve problems whose solutions are best represented as computer programs? For example, suppose a researcher wants to design a probabilistic graphical model for a novel domain. Searching the space of probabilistic models automatically is notoriously difficult, especially difficult when latent variables are involved. However, researchers seem able to easily adapt commonly used modeling motifs to new domains. In doing so, they draw on abstractions such as trees, chains, grids and plates to constrain and direct the kinds of models they produce. This suggests that before we ask machine learning algorithms to discover parsimonious models of new domains, we should develop techniques that enable our algorithms to automatically learn these "graphical concepts" in much the same way that researchers themselves do, by seeing examples in the literature. One natural way to think of these graphical concepts is as programs that take sets of random variables and produce graphical models that relate them. In this work, we describe the CEC algorithm, which attempts to learn a distribution over programs by incrementally finding program components that commonly help to solve problems in a given domain, and we show preliminary results indicating that CEC is able to discover the graphical concepts that underlie many of the common graphical model structures.

1 Introduction

The vast majority of research in machine learning is focused on utilizing large amounts of noisy data living in high-dimensional, real-valued vector spaces. Such research has been immensely successful and has enabled automatic and accurate modelling on a scale that was previously infeasible. However, when we want to build a car, draw a painting, or design the machine learning models to which we apply our new algorithms and techniques, we face problems that current machine learning tools are unable to handle. The highly structured objects that humans produce seems to require a different representation than the ones which machine learning algorithms have most successfully employed.

One example of a structured representation we would like to be able to learn is the probabilistic graphical model. Machine learning researchers make frequent use of graphical motifs and abstractions such as trees, chains, rings, grids, mixtures, and plates to constrain the space of graphical models they consider. But how do they learn that these are good motifs? How do they learn how to use and combine them to design new models?

One hypothesis is that these motifs and abstractions are best represented as programs that manipulate and compose graph operations. In this work, we discuss how one could learn these motifs and

*Questions and correspondence should be sent to ellisk@mit.edu.

abstractions by finding programs that generate common graphical models. In particular, we present the CEC algorithm, a general purpose multitask program induction algorithm. We discuss how CEC can be used to extract from a set of examples the kind of abstract knowledge that machine learning researchers bring to the task of designing graphical models, and we speculate that learning such representations can assist in the difficult tasks of automatically finding good latent-variable graphical models for novel datasets.

2 The CEC Algorithm

The CEC algorithm takes a library of primitives and a set of related tasks to solve; its goal is to produce a set of programs that solve these tasks. Like the EC algorithm [2], of which it an extension, the CEC algorithm maintains a distribution $\mathcal{D}$ over programs. In each iteration, CEC does two things: first, it uses $\mathcal{D}$ to explore the space of programs; and second, it compresses the solutions it found to generate a new weighted library of primitives that define an updated distribution.

Following [2] and [8], CEC learns programs expressed in simply typed combinatory calculus [11] The simplicity of the combinatory calculus makes describing a distribution over its expressions particularly simple. A more detailed explanation and discussion of this program representation can be found in [2, 8].

2.1 Generative Model

The CEC algorithm can be expressed as MAP inference in a probabilistic generative model. This generative model captures two intuitions: 1) for a set of related tasks, useful programs are composed of useful subparts, and 2) short programs are a priori more likely than long programs.

Our generative model assumes that the programs are themselves constructed by composing program fragments that are drawn from a common stochastic grammar. We write G to indicate this grammar, which is drawn from a distribution over grammars that is biased towards grammars with smaller descriptions lengths. We write $\{\rho_n\}_{n=1}^N$ to indicate the N programs, each of which solves one of the N tasks. We write $\{e_n^i\}_{i=1}^{\ell_n}$ to indicate the ℓ_n program fragments that are composed to create ρ_n, where ℓ_n is drawn uniformly from 1 to L, the maximum program length. The set $\{t_n\}_{n=1}^N$ refers to the tasks we want to write programs to solve. We use how well a program ρ_n solves task t_n as a surrogate for the likelihood function $P(t_n|\rho_n)$. This generative model is shown in Figure 1.

$G \sim P_G(\cdot)$ // Description-length prior
$\ell_n \sim \text{Uniform}(1, L)$ // Draw number of subprograms in n^{th} program
// Draw ρ_n, the n^{th} program, by composing its subprograms, e_n^i
$e_n^i \sim P_{e|G}(\cdot|G)$
$\rho_n = e_n^{\ell_n}(e_n^{\ell_n - 1}(\cdots e_n^1))$
// Draw t_n by adding noise ϵ_n to the output of ρ_n
$\epsilon_n \sim P_{\text{noise}}(\cdot)$
$t_n = \textsc{Eval}(\rho_n) + \epsilon_n$

Figure 1: The generative model underlying the SEC algorithm.

2.2 Inference

We implement a version of the EM algorithm [3] for MAP estimation of the grammar, G, given the tasks, $\{t_n\}$. The hidden data in this case is the latent program that solves each task.

The joint distribution factorizes as

$$P(G, \{\rho_n\}, \{t_n\}) = \frac{1}{L^N} P(G) \prod_n P(t_n|\rho_n) P(\rho_n|G), \tag{1}$$

where $P(\rho_n|G) = \prod_i P(e_n^i|G)$. From equation (1), the EM updates are

$$q(\rho_n) \propto P(t_n|\rho_n) P(\rho_n|G^{\text{old}}) \tag{2}$$

$$G^{\text{new}} = \text{argmin}_G \left(-\ln P(G) - \sum_n E_{q_n} \left[\ln P(\rho_n|G) \right] \right) \tag{3}$$

Exact computation of the normalizing constant in equation (2), or the expectation in equation (3), would require summing over the infinite space of programs that can be generated from the grammar. To approximate this expectation, we perform a heuristic search over the space of programs, with the goal of finding those programs for which $q(\cdot)$ is high.

The CEC algorithm uses beam search [12] to maximize $q(\cdot)$. The beam is initialized with the programs with the highest prior probability according to the grammar G. Similarly, the search moves considered are those functions with the highest prior probability under G.

Equation (3) has a natural interpretation as a form of compression. Interpreting negative log probabilities as description lengths, the update (3) picks the grammar minimizing the sum of the description length of the grammar, plus the expected description length of the programs found to solve the tasks. A slight generalization of the Neville-Manning algorithm [9] permits tractible approximate minimization of (3).

3 Learning to Construct Graphical Models

Learning the structure of graphical models that capture the independencies underlying a dataset is an active area of research[1][13][4][6]. Graphical model structures enforce the conditional independences a modeler believes exists among latent and observed random variables[10]. In practice, the sorts of graphical models that humans design – Hidden Markov Models, phylogenetic trees, topic models, Ising models – exhibit certain symmetries and recursive structures that might be amenable to synthesis by programs.

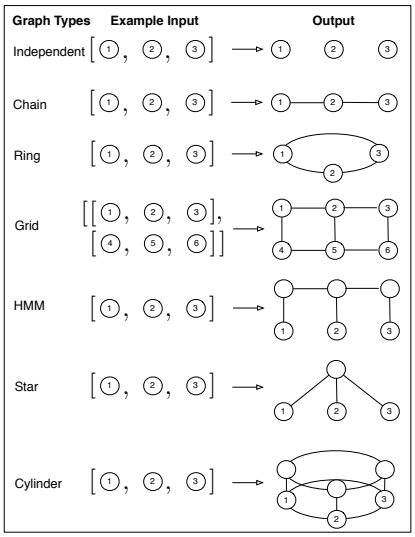

Figure 2: Example input/output pairs of learned programs. Each induced program takes as input an arbitrarily large list of observed nodes and produces an undirected graphical model. Labeled/unlabeled nodes correspond to observed/latent variables.

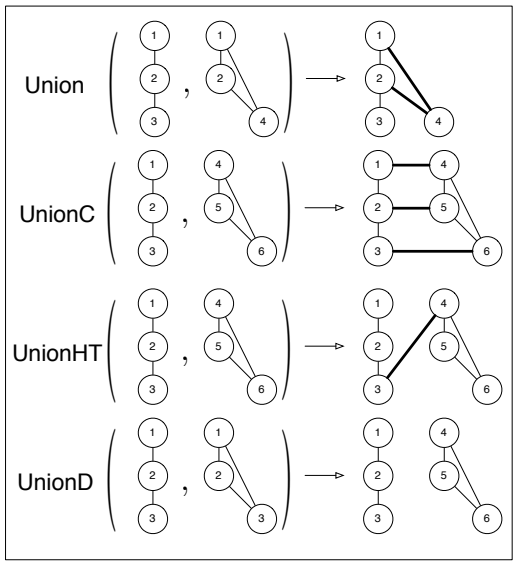

Figure 3: Graph combinators in initial library. Union: computes union of vertexes and edges. UnionC: adds edges between corresponding vertexes. UnionHT: adds edge between first (head) vertex and last (tail) vertex. UnionD: relabels input graph vertices so that they do not overlap and then computes union.

3.1 Experimental Setup

We wanted to use the CEC algorithm to learn the common abstractions and motifs that underlie the graphical models that are commonly found in the machine learning literature. To that end, we supplied CEC with a set of twelve tasks, each of which corresponds to learning how to build a specific type of graphical model, potentially with latent vertexes, given that it is provided with a list of observed vertexes. A partial list of the tasks is shown in Figure 2. For example, the CYLINDER

task requires CEC to build a program that takes as input a list of observed vertices (of any length) and produces a graph with one ring consisting of the observed vertexes and the other consists of corresponding latent variables. Tasks were specified via a set of input/output test cases, and we defined the score of a program on a task to be the fraction of test cases it passed.

The CEC algorithm was provided with an initial library of primitive combinators, which included the basis combinators of the combinatory calculus and functional list operations such as MAP and FOLDR/FOLDL. We also included four binary operations on undirected graphical models, each of which computes a union between two graphs. These graph operations are diagrammed in Figure 3.

3.2 Results

We ran the CEC algorithm on the set of twelve tasks. The algorithm is parameterized by the frontier size (the number of unique programs drawn from the library on each iteration). With a frontier size of 1000, the algorithm solves 7/12 tasks. With a frontier size of 5000, the algorithm solves 8/12 tasks. In each case, the algorithm returns solutions that partially satisfy the remaining tasks. Figure 4 shows CEC's solution to the CYLINDER task. Initially, CEC is unable to find solutions to the complicated graph structure tasks, such as the Ising Model or the Hidden Markov Model, even after enumerating ≈ 4 million programs through its heuristic search. However, it is able to find programs that generate simpler graphical models, such as Markov chains. By compressing the solutions to these simpler problems, it modifies its search procedure and, in subsequent iterations, learns more successful programs.

What grammars does CEC learn, and how do they enable the bootstrap learning of more complicated programs? The simplest example is how the Markov chain bootstraps the learning of the Ising model. After one iteration, the program fragment (FOLDR UNIONC), used in the Markov chain program, is incorporated in to the grammar. On the second iteration, the Ising model is learned using this fragment, producing the program (((S B) MAP) ((FOLDR UNIONC) NULL)).

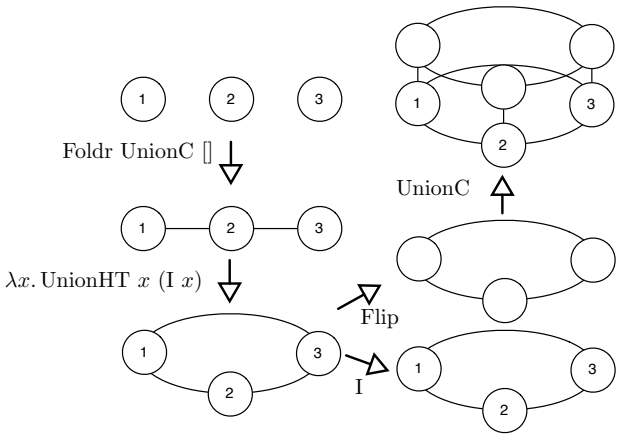

Figure 4: The program generates a cylinder from nodes by generating two rings – one latent, one visible – and connecting them.

4 Discussion

To be clear, what we have shown so far does not provide a way to learn the structure of a graphical model directly from observed data. Rather, this work suggests that we may be able to learn automatically an inductive bias that will subsequently make learning graphical model structures more efficient. Some of the programs that CEC includes in its library correspond to graph transformations, and we think that using these transformations as search operators will make structure learning of graphical models with latent variables a more tractable proposition than it currently is.

The notion that an appropriate inductive bias might help in the structure learning of graphical models is not new: [7] shows that the inductive bias induced via multitask learning helps a search algorithm find accurate graphical model structures. But while work in this field has focused on how to evaluate a given graph, we hope to show that CEC can learn appropriate search operators directly from a set of training examples.

References

[1] Ryan Prescott Adams, Hanna M. Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. *Journal of Machine Learning Research: Workshop and Conference Proceedings (AISTATS)*, 9:1–8, 05/2010 2010.

[2] Eyal Dechter, Jonathan Malmaud, Ryan P. Adams, and Joshua B. Tenenbaum. Bootstrap learning via modular concept discovery. In Francesca Rossi, editor, *IJCAI*. IJCAI/AAAI, 2013.

[3] A. P. Dempster, M. N. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 39:1–22, 1977.

[4] N Friedman and D Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1-2):95–125, JAN-FEB 2003.

[5] Johannes Fürnkranz and Thorsten Joachims, editors. *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. Omnipress, 2010.

[6] David Heckerman, Dan Geiger, and David Maxwell Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *CoRR*, abs/1302.6815, 2013.

[7] Jean Honorio and Dimitris Samaras. Multi-task learning of gaussian graphical models. In Fürnkranz and Joachims [5], pages 447–454.

[8] Percy Liang, Michael I. Jordan, and Dan Klein. Learning programs: A hierarchical bayesian approach. In Fürnkranz and Joachims [5], pages 639–646.

[9] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *CoRR*, cs.AI/9709102, 1997.

[10] Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[11] Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

[12] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[13] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, OCT 2006.