Modeling Expertise with Neurally-Guided Bayesian Program Induction

Catherine Wong (catwong@mit.edu)

Kevin Ellis (ellisk@mit.edu)

Mathias Sable-Meyer (mathias.sable-meyer@ens-cachan.fr)

Joshua B. Tenenbaum (jbt@mit.edu)
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology, MA 02139 USA

Abstract

Studies of human expertise suggest that experts and novices "see" problems differently. Experts not only acquire a body of domain-specific strategies and knowledge, but also learn to quickly identify when those concepts apply to a given problem within the domain. We propose modeling these elements as an iterative process of domain-specific language (DSL) learning, while jointly training a neural network to recognize when learned concepts apply to new problems in the domain. We show that the neural network allows the algorithm to solve problems more accurately and quickly than a model that simply acquires new concepts without learning when to use them. We also examine the implicit, vector-based problem representations learned by the neural network recognition model. Early in training, these representations cluster problems based on surface features, but they increasingly come to reflect abstract relationships between problems as the model acquires domain expertise.

Keywords: expertise; program induction; domain-specific language; relative expertise; problem representations

Introduction

Experts excel at solving problems. Within their domain of expertise, experts find the best possible solutions to presented problems (De Groot, 1965), and they find these solutions faster, more accurately, and with less cognitive effort than relative novices (Klein, Orasanu, Calderwood, & Zsambok, 1993; Alexander, 2003). Studies of human expertise have long shown that these differences extend to how experts and novices perceive and categorize problems themselves, even before working out a solution. Expert X-ray technicians recognize patterns in X-ray films that novices do not (Lesgold et al., 1988). Expert chess players identify and recall more informative chunks of midgame boards (De Groot, 1965; Chase & Simon, 1973). Expert computer programmers categorize scrambled lines of code based on principles of program function, whereas novices group them based on syntax (Adelson, 1981). Somehow, as they acquire domain-specific knowledge, experts learn to "see" the initial problems differently.

Theories of expertise often suggest that experts and novices construct differing initial problem representations for the exact same problems, based on their relative domain-related knowledge and its organization. Novices represent problems based on surface features, or the literal objects and entities explicitly described in a problem statement (Chi, Feltovich, & Glaser, 1981; Chi & VanLehn, 2012; Chi, 2006). In comparison, experts in a domain learn to represent problems based on their "deep structures", or the underlying principles governing problem solutions (Chi et al., 1981; Adelson, 1981; Chi & VanLehn, 2012). How do experts construct these initial problem representations at a glance, and how does acquiring greater domain-specific knowledge train experts to map surface features of problems to these relevant abstract structures even before they solve the problems themselves?

We propose modeling the process of expertise acquisition with an algorithm that simultaneously learns domain-specific concepts, and how and when to use them. Building on recent work by Ellis, Morales, et. al (2018), our computational model of expertise alternates between searching for solutions to program induction problems, and then distilling out new reusable subroutines from these solutions to build up a library of domain-specific concepts. For example, when acquiring expertise in drawing pictures, the model starts out with programming primitives for moving a pen across a canvas, and then learns subroutines for drawing circles (parameterized by their radii) or polygons (parameterized by size and number of sides). In tandem, we train a neural network recognition model, which learns to map problems to concepts likely to be used in solving those problems, where those concepts are drawn from the expanding, domain-specific library (DSL) of learned subroutines. This library of code (DSL) and the neural recognition network model offer two complementary notions of domain expertise: experts have at their disposal a repertoire of finely-tuned domain-specific concepts, analogous to the DSL, while at the same time forming fast intuitions about how and when to use those concepts to solve new problems, analogous to the neural recognition model.

Our choice to model solutions to problems as *programs* stems from the language-of-thought hypothesis (Fodor, 1975) and its modern computational descendents (Piantadosi, 2011; Goodman, Tenenbaum, & Gerstenberg, 2015). These computational frameworks treat inferring expressions in a language-of-thought as a kind of program induction in a domain specific language. Viewed through a language-of-thought lens, our stance is that experts augment their internal mental language with a domain-specific vocabulary, and that this domain specific vocabulary bootstraps the expert's ability to rapidly glean the gist of a problem.

We apply our model to three domains of problems - list processing, text editing, and graphics drawing - that can be solved with short programs in a domain-specific language. In

other words, solving these problems can be easy, for a model that has acquired the right concepts. Thus the challenge is to acquire the domain specific language in the first place, while jointly learning to deploy it efficiently on new problems.

We show that like human experts, this algorithm grows faster at solving problems, and solves more problems overall, as it acquires more domain-specific knowledge. However, we demonstrate that it is adding the neural network recognition model that allows the algorithm to perform significantly better than acquiring domain-specific concepts alone: expertise requires recognizing which learned concepts apply to which problems, even before searching for a solution.

Next, we examine how our model's learned problem representations shift over the course of its learning trajectory, inspired by prior work on human expertise demonstrating that novices and experts use different problem features when categorizing, sorting, and evaluating similarities between problems in the domain (Chi et al., 1981). When presented with a problem, the recognition model constructs an implicit vector-based problem representation, encoded in its output layer activations. We examine these learned initial problem representations and show that early in the model's learning trajectory, representations from novice models group related problems based on surface features, whereas later expert representations identify problems that share abstract, underlying structures in their solutions. Further, we show that as the models acquire expertise, clustering these vector-based representations captures abstract semantic similarities within the domain as a whole.

Expertise as Neurally-Guided DSL Learning

We model expertise acquisition as an iterative process of domain-specific language (DSL) learning. Our model is built on top of the Explore/Compress/Compile (EC^2) algorithm (Ellis, Morales, et. al, 2018), which we briskly review here.

Explore/Compress/Compile (EC^2)

The EC^2 algorithm offers a computational framework that combines two powerful ideas from program induction: learning new, reusable subroutines shared across multiple tasks, and methods to train bottom-up neural networks that regress from problems to programs. Unlike many other recent machine learning algorithms for program induction (Devlin et al., 2017), this algorithm uses the neural network to guide the search for programs by producing a conditional distribution over learned concepts given a task, rather than explicitly predicting a sequence of primitives or the program tree itself. At a high level, EC^2 iteratively solves problems to acquire domain-specific expertise in three stages (Figure 1):

Explore: During the Explore phase our nascent expert searches for programs that solve tasks by enumerating programs from the current DSL in decreasing order of their probability, guided by the neural recognition model. We represent programs as typed λ-calculus expressions.

Compress: During the Compress phase we refactor programs found during Explore to expose commonly reused components, and incorporate those components into the DSL, thereby 'compressing' the programs. This expands the model expert's problem-solving language to include new concepts built out of earlier, simpler concepts. For example, when becoming an expert at list processing, Compress learns a component for taking the maximum of a list, and then later builds on it to learn a DSL subroutine for finding the nth largest number of a list; when becoming an expert at drawing pictures, Compress learns a subroutine for drawing regular polygons, and then later specializes it into another subroutine for drawing squares.

Compile: During the Compile phase we train a neural recognition model to efficiently search for programs in the current DSL, by learning to map inputs and outputs of a specific problem to a distribution over acquired concepts. Intuitively, the recognition model looks at a specific problem and predicts which combinations of concepts in the DSL are likely to solve it. The recognition model learns from a pair of self-supervised streams of data, training both on solutions the model itself discovers during the Explore step, and from randomly sampled programs drawn from its DSL. This sidesteps the problem of obtaining ground-truth solutions to problems, allowing EC^2 to get off the ground without (problem, solution) pairs.

Formally, EC^2 takes as input a collection of problems, written X, which can be solved by programs, and infers a domain specific language – written $\mathcal{D}$ – which functions as a generative model over programs likely to solve problems in the domain, and which we identify as an expert's a priori beliefs about what good solutions look like. Alongside this generative model it trains a neural recognition model, in the spirit of the Helmholtz machine or wake-sleep algorithm (Dayan, Hinton, Neal, & Zemel, 1995). The recognition model takes as input a specific problem and quickly predicts an approximate posterior over programs likely to solve that problem. Writing p_x for the program solving problem $x \in X$ and $Q(p|x)$ for the distribution predicted by the recognition model, EC^2 iteratively (and approximately) solves for

$$p_x = \arg\max_p \mathbb{P}[x|p]\mathbb{P}[p|\mathcal{D}] \tag{1}$$

$$\mathcal{D} = \arg\max_{\mathcal{D}} \mathbb{P}[\mathcal{D}] \prod_{x \in X} \sum_{\substack{p: \\ p \text{ found during Explore}}} \mathbb{P}[p|\mathcal{D}] \tag{2}$$

$$Q(p|x) \approx \mathbb{P}[p|x, \mathcal{D}] \tag{3}$$

Eq. 1 corresponds to Explore; Eq. 2 corresponds to Compress; while. Eq. 3 expresses the objective of Compile.

Our aim is to show how this algorithm can shed insight on the process of acquiring expertise. We show how learning to recognize new domain-specific concepts can allow for more efficient and accurate problem solving with an expanding body of acquired knowledge. We then show how we can interpret the learned recognition model weights as vector-based problem representations that allow the model to map features of the problem to acquired concepts, in the spirit of human problem solving. We demonstrate that expert model representations implicitly capture increasingly abstract relationships between problems within the domain, even before the model has explicitly searched for solutions to these tasks.

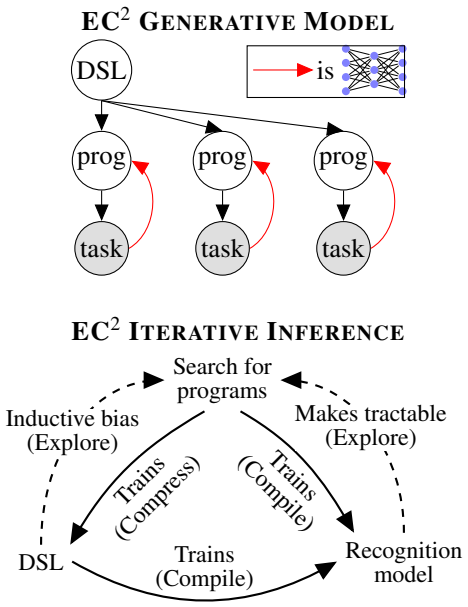


Figure 1: **Top**: EC^2 as an inference problem. Black arrows show top-down generative model (DSL→program→observed task/problem). Red arrows show bottom-up recognition model (predict program conditioned on task). **Bottom**: Iteratively solve for programs (Explore), update DSL (Compress), train recognition model (Compile), each step bootstrapping off the others.

Experiments

We investigate three problem domains (Fig 2):

List processing tasks, which require performing arithmetic operations on lists of numbers to produce desired outputs. We use a dataset of 218 human-interpretable list processing tasks taken from (Ellis et al., 2018), which were created by a human expert coder (not one of the authors).

Text editing tasks, which require manipulating strings of text, such as by repeating, replacing, or extracting characters. Text editing is a classic problem in the programming languages and AI literature (Gulwani, 2011). We train on a dataset of 128 automatically generated text editing tasks, each consisting of 4 input/output examples, also taken from (Ellis et al., 2018). We test on a standard text editing benchmark suite (Alur, Fisman, Singh, & Solar-Lezama, 2016).

LOGO Graphics tasks, which require drawing a target image. We created a corpus of 150 tasks inspired by LOGO Turtle graphics (Thornburg, 1983), split 50/50 test/train. In each, we provide a target input image, which the model must reproduce by controlling a simulated 'pen'.

In each domain, we provide a set of generic programming primitives that the algorithm can compose to iteratively construct its own DSL. For list and text we provide minimal sequence manipulation primitives like *map* and *fold*; for LOGO we provide primitives for moving/rotating/lifting a pen, along with arithmetic operations on angles and distances, and *for* loops. We consider a problem solved if executing the solution program on the inputs successfully produces the outputs (list & text) or if it renders to the target image (LOGO). Our neural recognition model predicts a distribution over subroutines in the full DSL, conditioned on their local context in the syntax tree of the program. Specifically, we consider context to be the parent node in the syntax tree and the argument being generated, effectively functioning as a bigram model over program trees. For the list and text domains, we encode problem examples using an recurrent neural net to extract features from the inputs; for the LOGO domain, we encode problem input examples using a convolutional neural net. We run the algorithm for twenty iterations on list and text, and attempt to solve a randomly sampled batch of 10 problems at each iteration. For LOGO, we run for ten iterations, and attempt to solve the 75 training tasks at each iteration.

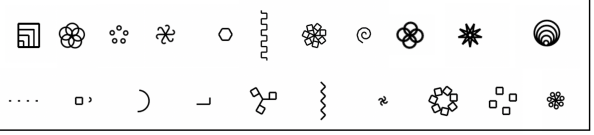
(a)	[7 0] → [7 0 7 0] [8 4 9] → [8 4 9 8 4 9] [3 2 6] → [3 2 6 3 2 6] *Repeat list*	[0 3 5 9 4] → [3 5 9] [2 1 4 7 7] → [1 7 7] [6 9 7] → [9 7] *Remove if even*	[1 2 3] → 6 [9 2] → 18 [5 4 1 6] → 120 *Multiply*
(b)	Harry Potter → H.P. Carmen Sandiego → C.S. Sherlock Holmes → S.H. *Abbreviate*	the (big) apple → big (see) spot run → see aa bbb cc (ddd) → ddd *Extract parenthesized*	one two three → one two dog cat mouse → dog cat aa b cc d ee → aa b cc d *Drop last*
(c)			

Figure 2: Sample input and outputs for (a) list editing, (b) text processing, and (c) LOGO Graphics drawing tasks.

Results

Recognizing when to use concepts allows for faster and more effective problem solving.

We demonstrate the impact of the neural recognition model on problem solving by comparing the full algorithm with an ablated version, which continues to expand its DSL with new domain-specific concepts from discovered solutions at each iteration, but does not jointly train a neural recognition model to guide search. Instead, the ablated algorithm solves problems by enumerating programs in order of their probability from the current DSL at each iteration. The ablated algorithm corresponds to using the Explore and Compress steps of the full algorithm, without the Compile step. In this sense it is analogous to the EC (Exploration-Compression) algorithm (Dechter et al., 2013) that was a forerunner of EC^2.

As seen in Figure 3, both the full and ablated algorithm successfully solve more unseen testing problems, and generally grow faster at solving individual problems, over the course of the learning trajectory. Analogous to a human expert steadily acquiring more domain-specific knowledge, expanding the DSL with new subroutines allows the model to more quickly synthesize programs that solve problems within the domain.

Notably, however, on all three domains, using the neural network recognition model to guide search within the learned DSL dramatically improves the number of tasks that the model successfully solves. On both the list and text domains, the neurally-guided model also solves these problems in significantly less time than the ablated algorithm. On the LOGO domain, the neurally-guided model solves many more tasks than the ablated version, but takes increasingly more time per task, potentially because it progressively solves more challenging tasks that take more time overall at each iteration, without reaching a learning plateau. These results suggest that training the neural network allows the model to more efficiently deploy the acquired, domain-specific concepts at its disposal.

Learned expert problem representations capture abstract semantic similarities.

Inspired by prior work on human relative expertise (Chi et al., 1981; Chi & VanLehn, 2012), we analyze the problem representations produced by the model. As the model solves problems and trains its recognition model, it constructs two complementary representations for a given problem: an initial vector-based problem representation, implicitly encoded in the output activations of the neural network when it is presented with input and output examples of a task, and a final symbolic representation, given by the actual set of programs the model ultimately produces as it searches for a solution.

Formally, we define neural network problem representations to be the final output layer activations of the neural network produced in response to a given problem; intuitively, these can be interpreted as learned representations that map problem features to a "gist" over the model's current domain-specific concepts. We also define discovered program representations to be the $|\mathcal{D}|$-dimensional distribution vector encoding the relative frequencies of each DSL primitive in the discovered program solutions, where $|\mathcal{D}|$ is the size of the final DSL; intuitively, these reflect the structure of the actual solutions.

Chi et. al (1981) discuss differences in how novices and experts sort and categorize physics problems. While both novices and experts see the same initial problem features, human experts form problem representations that reflect the higher-level principles shared between presented problems.

In Figure 4, we examine how our computational model's learned problem representations shift over the course of its learning trajectory. We cluster these representations using TSNE (Maaten & Hinton, 2008). Figure 4 depicts TSNE visualizations of the neural net representations at the first and last iteration of the algorithm, as well as the final discovered program representations produced for these same tasks.

We color-code these visualizations based on human-interpretable problem categories, which we define based on task names written apriori by the task designers. Notably, however, the models themselves never have access to these categories. Rather, as the model acquires expertise, the neural network adapts how it transforms task features, so that mathematical similarities between problem representations draw together problems to reflect semantically meaningful similarities within the domain. Comparing these clusters to those formed by the discovered program representations suggests that these initial problem representations, which serve to guide the search for solution, also reflect similarity structures between the actual program solutions.

Novice and expert similarities reflect surface vs. abstract problem features.

Chi et al. (1981) also find that novices and experts use different problem features when comparing problems within the domain. Novices tended to identify similar problems by surface features, whereas experts base their judgments on abstract concepts that may reflect potential solution methods. Figure 5 shows examples of how the models learned neural representations change in judging similarities between tasks over the course of its learning trajectory. We show selected problems, alongside the 5 most similar problems based on cosine similarity between the neural problem representations. We also compare the mean similarity between these tasks in the discovered program representations, as a quantitative metric of the similarity of their actual solutions. Qualitatively, these examples demonstrate how the models learned neural representations adapt over the course of training. Initially, the network appears to identify similar problems based on local features. Over time, however, the network appears to better judge similarity that aligns more closely with shared structures in the problem solutions.

Discussion

We have investigated how acquired expertise can be modeled by a program induction algorithm that jointly grows a library of discovered domain-specific concepts, and trains a neural recognition model to guide its search for problem solutions. In a very basic sense, the model certainly acquires domain expertise - it builds up a library of domain-specific knowledge, which allows it to solve problems that it could not before, and arrive at solutions more quickly.

However, we seek to model how this process of problem solving and knowledge acquisition can simultaneously train experts to build up something more like a domain-specific "intuition": the ability to recognize when these newly acquired concepts apply to problems, and to better structure how the model categorizes and judges similarity between problems in the domain. This combined approach of modeling domain expertise, using explicitly acquired concepts deployed by an iteratively trained pattern recognition model, leaves open many possible avenues for future work. In the short term, we can explore how this approach extends to other problem domains well studied in human novices and experts, such as mathematics, physics, or game playing problems. In the longer term, this approach could be used to explore how more direct pedagogy can guide model learning. The current algorithm discovers concepts in an unsupervised manner, but future work could explore how problem curricula, teacher-provided concepts and solutions, and even language learning can change how the model acquires and deploys its domain-specific knowledge.

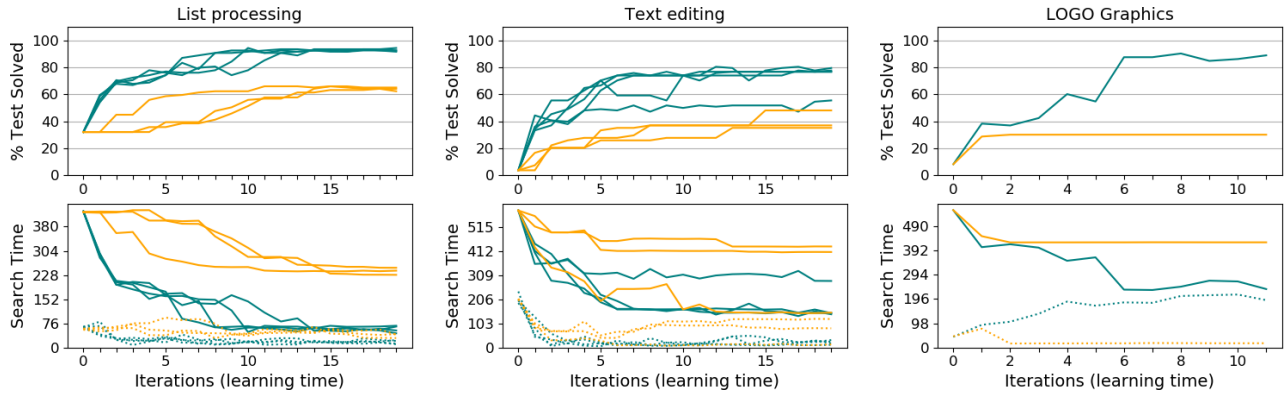


Figure 3: Learning curves showing % held-out problems solved (top) and average solve time in seconds (bottom; solid, averaged over all problems; dotted, averaged over solved problems) both with (teal; full algorithm) & without (orange; ablated algorithm) the neural network recognition model. Each trajectory is a run of the algorithm with a different random seed.

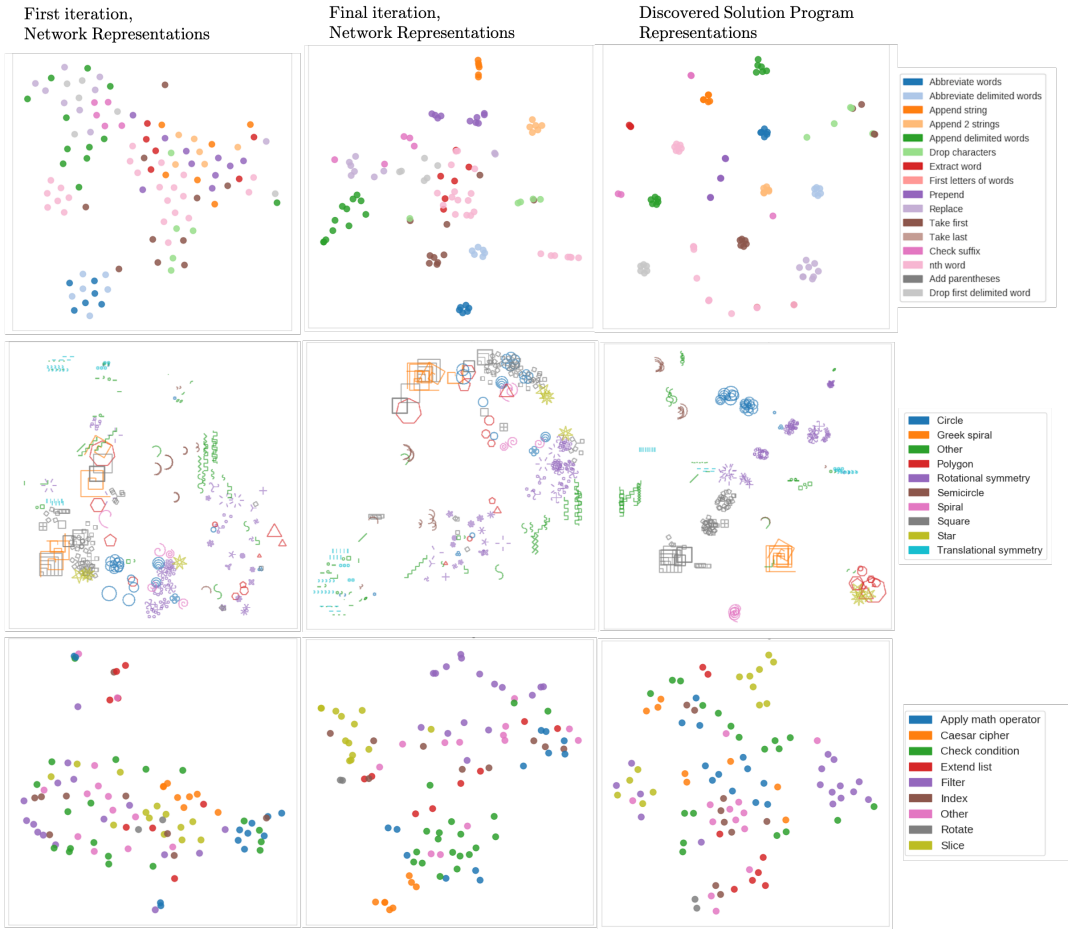


Figure 4: TSNE Visualizations of learned problem representations. From top to bottom: text processing; LOGO Graphics; list editing. From left to right: neural network representations at the first iteration of training. Middle: neural network representations at the last iteration. Right: discovered program representations encoding the program solutions. Color-coded by semantic categories unseen by the model.

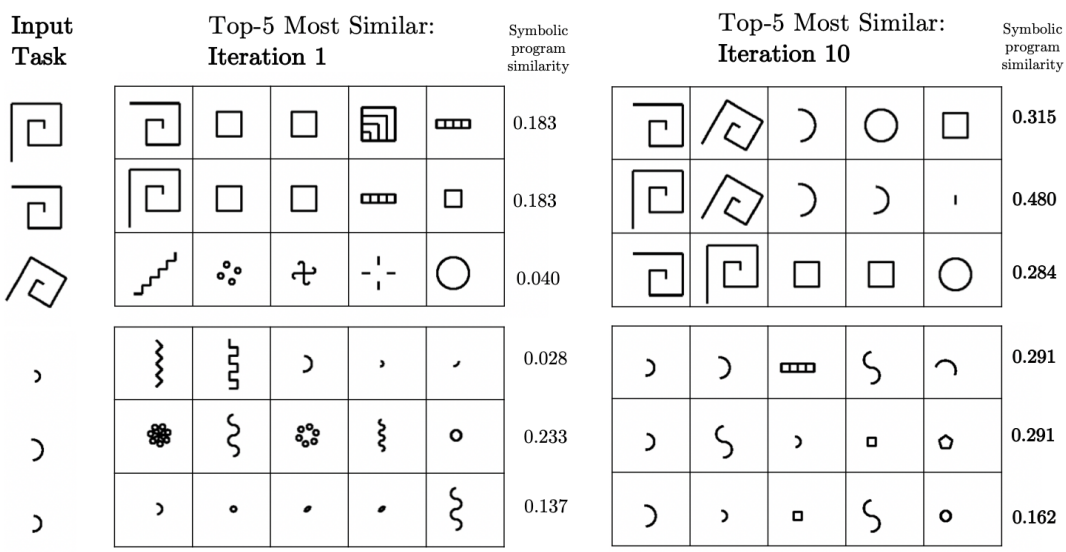


Figure 5: Left: Input tasks. Middle grids: Top-5 closely related tasks according to recognition model at iteration 1, ordered left-to-right, and similarity in the solution program representations. Right grid: Top-5 closely related tasks at iteration 10.

References

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & cognition*.

Alexander, P. A. (2003). Can we get there from here? *Educational Researcher*.

Alur, R., Fisman, D., Singh, R., & Solar-Lezama, A. (2016). Sygus-comp 2016: results and analysis. *arXiv preprint arXiv:1611.07627*.

Chalnick, A., & Billman, D. (1988). Unsupervised learning of correlational structure. In *Cogsci.*

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive psychology*.

Chi, M. T. (2006). Two approaches to the study of experts characteristics. *The Cambridge handbook of expertise and expert performance*, 21–30.

Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive science*, *5*(2).

Chi, M. T., & VanLehn, K. A. (2012). Seeing deep structure from the interactions of surface features. *Educational Psychologist*.

Dayan, P., Hinton, G. E., Neal, R. M., & Zemel, R. S. (1995). The helmholtz machine. *Neural computation*.

De Groot, A. D. (1965). *Thought and choice in chess*. Walter de Gruyter GmbH & Co KG.

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

Feigenbaum, E. A. (1963). The simulation of verbal learning behavior. In *Computers and thought.* McGraw-Hill.

Fodor, J. A. (1975). *The language of thought* (Vol. 5). Harvard University Press.

Goodman, N. D., Tenenbaum, J. B., & Gerstenberg, T. (2015). Concepts in a probabilistic language of thought. In Morgolis & Lawrence (Eds.), *The conceptual mind: New directions in the study of concepts.* MIT Press.

Gulwani, S. (2011). Automating string processing in spreadsheets using input-output examples. In *Acm sigplan notices.*

Hill, J. A. C. (1983). A computational model of language acquisition in the two-year old. *Cognition and Brain Theory*.

Klein, G. A., Orasanu, J. E., Calderwood, R. E., & Zsambok, C. E. (1993). Decision making in action: Models and methods. In *This book is an outcome of a workshop held in dayton, oh, sep 25–27, 1989.*

Lesgold, A., Rubinson, H., Feltovich, P., Glaser, R., Klopfer, D., & Wang, Y. (1988). Expertise in a complex skill: Diagnosing x-ray pictures.

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*.

Matlock, T. (2001). *How real is fictive motion?* Doctoral dissertation, Psychology Department, UCSC.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Ohlsson, S., & Langley, P. (1985). *Identifying solution paths in cognitive diagnosis* (Tech. Rep.). Carnegie Mellon University, The Robotics Institute.

Piantadosi, S. T. (2011). *Learning and the language of thought*. Unpublished doctoral dissertation, MIT.

Shrager, J., & Langley, P. (Eds.). (1990). *Computational models of scientific discovery and theory formation*. San Mateo, CA: Morgan Kaufmann.

Thornburg, D. D. (1983, March). Friends of the turtle. *Compute!*.