Downloading Culture.zip: Social learning by program induction

Anonymous CogSci submission

Abstract

Cumulative culture depends on the fidelity of learning between successive generations. When humans learn from others, in addition to observing inputs and outputs we often observe the process which led to that output. For instance, when preparing a meal, we don't just observe a pile of vegetables and then a ratatouille. Instead, we observe a causal process that transforms those ingredients into a finished food. Here, we use probabilistic programs to represent causal processes and show that the observation of an execution trace speeds up program induction, even when learning from only a single example. This model predicts that the inferences and behavior of people will be structured by these execution traces. In two behavioral experiments we show that human judgments and behavior are affected by the execution trace in the systematic ways predicted by our formal model. These findings shed light on the mechanisms that underlie high fidelity social learning in humans and unifies the role of emulation and imitation in social learning.

Keywords: social learning; program induction; Bayesian modeling; imitation learning; theory of mind

Introduction

Humans are the most sophisticated social learners in the natural world (Herrmann, Call, Hernández-Lloreda, Hare, & Tomasello, 2007). Learning from others has been called "the secret of our success" (Henrich, 2015), and is what makes humans a "a different kind of animal" (Boyd, 2017) because it enables cumulative culture. That is, we can inherit culture from previous generations, improve it, and pass on the improvements to the next generation. This is how we have gradually developed the sophisticated tools, languages, sciences, and moralities of today. But cumulative culture is only possible when social learning is highly reliable i.e., the transmission of knowledge between individuals is both high fidelity and robust (Boyd & Richerson, 1996; Lewis & Laland, 2012). Otherwise, the knowledge is lost to errors and noise.

Two forms of social learning have been widely explored (Tomasello, 1999; Tennie, Call, & Tomasello, 2006). "Imitation" involves directly copying another person's actions—for instance, learning to make a canoe by chopping a mature oak with with a handsaw, burning out its interior with coals, etc. Imitation can be high fidelity, but it not very robust. After all, the optimal sequence of actions to build a good canoe in one case may not be optimal in another (e.g., if you have a saw rather than an axe; a maple rather than an oak; an adze rather than coals). In contrast, "Emulation" involves observing finished cultural products (e.g., a canoe) and then deriving an *ad hoc* method of reproducing it. Emulation can be very robust, because *ad hoc* planning can accommodate variable circumstances. But this comes at the cost of fidelity: If a person happens to develops a better *method* for producing a canoe, mere emulation does nothing to preserve it.

We propose that human social learning naturally occupies an intermediate point that is both high fidelity and robust. Specifically, we hypothesize that human cultural knowledge often takes the form of probabilistic programs. These are akin to recipes, generative grammars, or synthetic computer programs. For instance, building a canoe might be encoded as a program like "(1) Cut a tree down; (2) Hollow the inside; ... etc. ". When we watch a person build a canoe we cannot directly observe the program in their head, but we can infer it. What we learn is thus more precise than emulation, but more abstract and robust than mere imitation.

Our goal is to model this form of social learning in computational terms. By casting social learning as program induction we can formally state how observing both process and product allow us to learn a program. Our work is deeply influenced by prior work on using programs to represent and learn sophisticated structured concepts such as handwritten characters (Lake, Salakhutdinov, & Tenenbaum, 2015), composable functions (Schulz, Tenenbaum, Duvenaud, Speekenbrink, & Gershman, 2017), geometric sequences (Amalric et al., 2017), and list operations (Rule, Schulz, Piantadosi, & Tenenbaum, 2018) but to our knowledge the application of these techniques to social learning is unique.

Specifically, we show that when people observe the sequence of actions that an expert uses to construct a cultural artifact, they are able to infer key aspects of the program that generated the artifact (Byrne & Russon, 1998). In our experiments this cannot reduce to mere imitation, because participants are readily able to generalize the program to generate novel sequences of actions. We also show that it cannot reduce to mere emulation, because observing the same end products generated by two different processes leads people to infer very different programs, and thus to generalize very differently to novel artifacts.

The remainder of the paper is structured as follows. We first motivate the importance and power of observing process, rather than mere products, when learning from others (Caldwell & Millen, 2009; Derex, Godelle, & Raymond, 2013). We then introduce a social learning task that is amend-

able to both formal computational modeling and human behavioral experiments. We then introduce a computational model based on probabilistic program induction for this task. Finally, we compare the results of this model qualitatively to behavioral data.

Example: Learning to sort a list of numbers

We begin with a simple case study of probabilistic program induction involving transforming a string of numbers. Suppose, that an observer watches an expert input the list of numbers $[5,3,4,1]$ and then output the list $[1,3,4,5]$. What program will she she learn? She might infer that the program sorts numbers, but there is significant ambiguity. A valid program to describe this transformation would be to just swap the first and last elements of the input. And, even among sorting algorithms, there are many different kinds.

The learner would be able to draw much stronger inferences if she could observe not just the inputs and the outputs, but the precise sequence of intermediate operations. (This is analogous to observing not just that a person turned a log into a canoe, but the sequence of actions they used to do so). This sequence of state transformations is called the program's *execution trace* or just *trace*. For instance, if the trace for our example is observed one might see $[5,3,4,1] \rightarrow [3,5,4,1] \rightarrow [3,4,5,1] \rightarrow [3,4,1,5] \rightarrow [3,1,4,5] \rightarrow [1,3,4,5]$. Here the entire trace of the "bubble sort" algorithm is observable, where elements are swapped if the left is greater than the right until no element on the left is greater than an element on the right. Observing this execution trace not only makes it clear that the underlying program is a sorter but also gives clues as to how to write a sorting algorithm.

As this example illustrates, observing a program's trace allows efficient and high-fidelity inference of the program's structure. Meanwhile, the program is more robust than the specific sequence of actions encoded in the trace: The trace tells you how to sort *one* list, whereas the program allows you to sort *any* list. In this way, probabilistic program induction over execution traces enables a form of efficient, high-fidelity and robust social learning.

Task: Learning from traces

In this section we describe a novel task meant to imitate some basic aspects of social learning where rich trace information is available to the learner. Our task is designed to be both intuitive for behavioral experiments and also tractable for formal modeling. In this task, a participant is asked to watch another agent create a necklace made of beads. The beads come in red and blue circles (although this can easily be extended to include arbitrary colors and shapes) and can be placed on the necklace in any order, one at a time. Participants are told that these necklaces are culturally important to the agents who make them and only certain types of necklaces are allowed. A participant either watches an agent create an allowable necklace one bead at a time ("process"), or instead observes only the resulting necklace ("product"). The task of the participant is to infer the underlying program that generated this necklace.

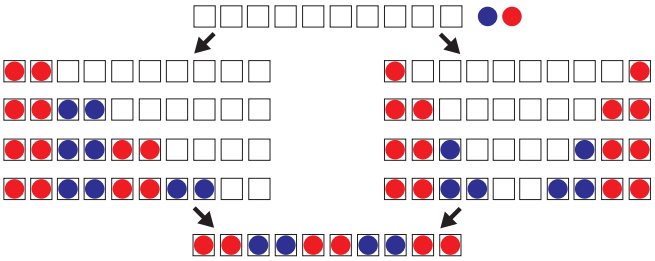

Figure 1: Two different procedures can generate the same necklace. Observing different execution traces of the same necklace leads to different inferences about the program which generated the necklace. Note: beads are placed one at a time but added in doubles to save space. (left) The `seq` method highlights a sequence of alternating beads. (right) The `outin` method highlights mirror symmetry.

To demonstrate how observing process versus product can lead to significantly different inferences about the underlying program consider the examples in Figure 1. On the left, the observer first sees two red beads placed left-to-right, then two blue beads after that, and so on in sequential order. From this example, an observer might infer that the rule that makes a necklace allowable is one where the bead colors alternate every two beads. Now consider the same necklace but built a different way: on the right of Figure 1, the two red beads are placed on the outer ends of the necklace, and then the rest of the beads are placed by alternating between the sides in an "out-in" fashion. From this example, even though the product is the same as on the left, an observer might be more likely to infer that the rule is that the necklaces must have some level of mirror symmetric structure. These are two different rules that define two completely different spaces of acceptable necklaces and they're readily differentiated with trace information. Without trace information they are completely identical.

Computational Model

To test the hypothesis that traces enable rapid and more accurate program learning we present a program induction system for the necklace learning task described above. There are 2^N possible necklaces of length N, so our goal is to describe the necklaces and the different ways of making them using simple compositional primitives. We first formalize the space of necklaces and their traces with a domain-specific language (DSL) that concisely expresses two major concepts in our task definition: repeated and symmetric structure. We then describe how latent programs can be inferred from necklaces using Bayesian inference.

Programs in a domain-specific language

The DSL defines the space of syntactically valid programs, which we denote as the set pp. The DSL in this model is

```
0: start -> program
1: program -> placer INT picker

# placer primitives
2: placer -> NONPARAM | PARAM
3: NONPARAM -> "seq" | "outin"
4: PARAM -> "skip" INT

# picker primitives
5: picker -> grow | random | repeat
6: grow -> "grow" INT picker
7: repeat -> "repeat" INT picker
8: random -> "random" INT INT |"random" LETTER INT
```

Figure 2: The syntax of the domain-specific language (DSL) for the necklace programs. Terms explained in The Model section of the main text.

shown in Figure 2. A syntactically valid program consists of a sequence of expressions, where each expression is one of six primitives in the DSL. These six primitives belong to two major types: "placers" and "pickers". Placers define the different ways that individual beads within a bead set can be placed on a necklace:

`seq(`L`,` B`)`: place the beads B from left to right on a necklace of size L,

`outin(`L`,` B`)`: place the beads B in an "out-in" pattern (alternate between placing beads on the far left and far right) on a necklace of size L,

`skip(`L`,` S`,` B`)`: place the beads B every S spaces on a necklace of size L.

Pickers are composed to generate different sequences of beads (B). These sequences are then passed as arguments to pickers:

`random(`Z`,`C`)`: randomly sample, w/o replacement, C beads and duplicate them in place Z times,

`repeat(`Z`,`λ`)`: memoize the resulting bead set from the program λ and repeat it Z times,

`grow(`Z`,`λ`)`: execute the program λ Z times and return those outputs as a contiguous bead set.

Figure 3 shows two example programs (and a sample necklace for each one) that are expressible in this DSL.

A guiding principle in the design of this DSL is that the primitives should be based on general building blocks such as repetition, memoization, and random sampling. This gives the generated program commonsense interpretations which can be translated into natural language. For example, `seq 10 repeat 5 random 1 2` translates to "pick two beads of different colors and repeat them for the length of the necklace and place them left-to-right" and `outin 10 repeat 2 grow 5 random 1 1` translates to "pick a random sequence of 5 beads, repeat that sequence and place the beads alternating between the left and right ends of the necklace".

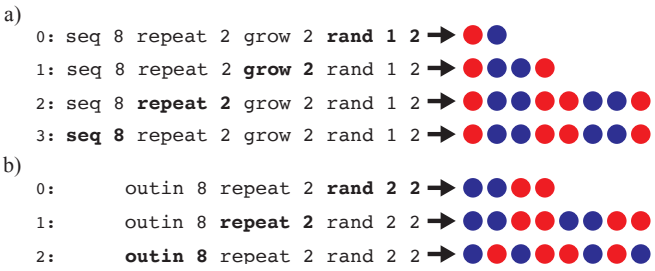

Figure 3: Example of how different programs are executed into necklaces. Each line shows how the sequence of beads is incrementally expanded by each term of the program and then eventually placed on the necklace by the placer. (a) A symmetric program that composes repeat and grow. (b) A repeating sequence.

Inference

Given a sample necklace, we formalize program induction as Bayesian inference over the possible programs defined by the DSL. We denote individual programs in this space as $\pi \in pp$ (see Figure 3 for examples of π). Each π defines a space of possible necklaces N and steps for making those necklaces (traces) T. Using Bayes' rule, the posterior probability of a program π, given an observed necklace N is:

$$P(\pi|N) \propto P(N|\pi)P(\pi) \tag{1}$$

where $P(N|\pi)$ is the likelihood that π generates N and $P(\pi)$ is the prior probability for generating π. This probabilistic formulation amplifies the weight on programs that produce N and few other necklaces over those that produce N and a large number of necklaces. This feature of probabilistic inference is sometimes called the "size principle" (Tenenbaum & Griffiths, 2001). As a result, the model predicts that people should prefer programs that can generate the observed necklace inversely proportional to how many other necklaces that program can also generate. A preference for specificity emerges out of the probabilistic approach to inference.

We also use a prior over the space of programs that weights shorter programs over longer and more complex ones (Ho, Sanborn, Callaway, Bourgin, & Griffiths, 2018):

$$P(\pi) \propto \frac{1}{|pp| + |\pi|} \tag{2}$$

where $|pp|$ is the number of possible programs defined by the DSL and $|\pi|$ is the length of the program, i.e. the number of primitives and parameters that make up π.

We have now introduced sufficient notation to formally state our hypothesis that social learning from execution traces is as efficient and accurate or more than from input/output alone. Stated formally:

$$P(\pi^*|T) \geq P(\pi^*|N) \tag{3}$$

where $P(\pi^*|T)$ is the posterior belief in a ground-truth program π^* having observed the trace of steps taken to generate

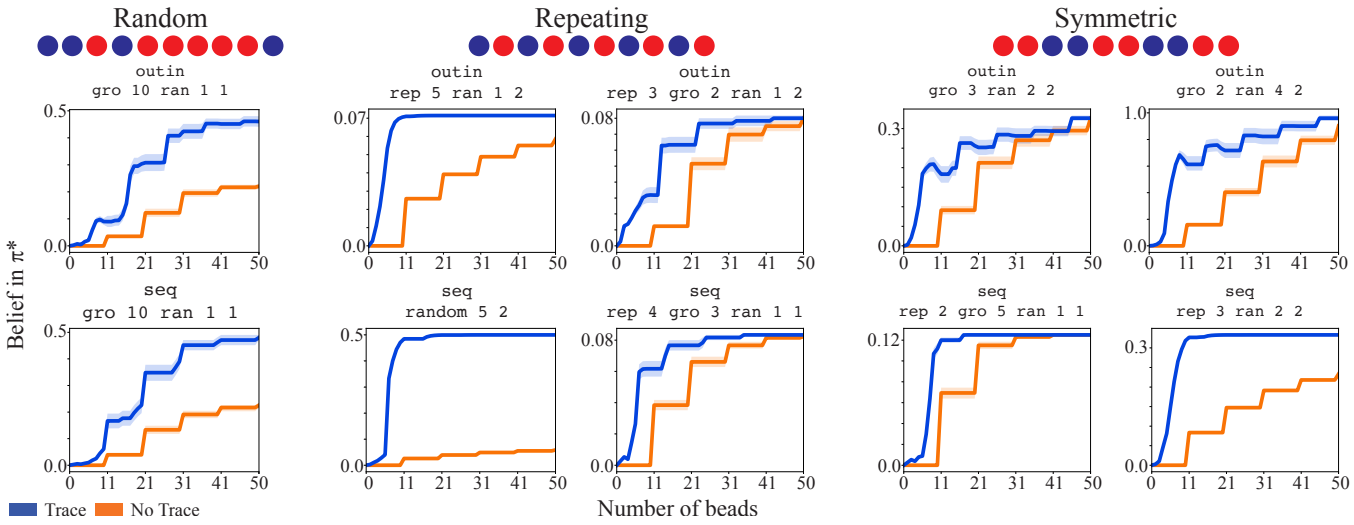

Figure 4: Results of computational experiments comparing accuracy and efficiency of inference with trace information ($P(\pi|T)$, blue) vs without ($P(\pi|N)$, blue). Experiment averaged over 50 trials. Error bands are the standard error of the mean. In each trial the model observed samples from the program listed above each plot, either bead-by-bead (trace) or all at once (no trace).

its output necklace T. Stated plainly: you will always be more accurate inferring the underlying program if you take into account trace information than if you just saw the output. Formally, the difference is in the likelihood, one only considers the complete necklace N, $P(N|\pi)$ and the other also considers the steps taken to complete the necklace T, $P(T|\pi)$. These likelihoods are computed analytically for each program by enumerating all necklaces and their corresponding traces that a given program could produce.

Computational Results

We simulated learning to empirically investigate the predictions made in Eq (3) and compared dynamics of learning with and without trace information. We tested learning on three different types of programs (random, repeating, and symmetric) and both of the DSL's placer primitives (seq and outin). Performance was measured by learning efficiency (how many data points were needed in either condition to make reasonable inferences) and asymptotic accuracy (how well it inferred the ground-truth program in either condition).

The three separate types of programs we chose have salient, differing structure (see Figure 4): random programs that generate arbitrary necklaces of a given length, repeating programs that generate necklaces with simple, repeating motifs, and symmetric programs that generate necklaces with symmetry across the middle of a necklace. For each of these types, we used both `seq` and `outin` and used necklace lengths of ten. In the trace condition, the model performed inference about the underlying program bead by bead. In the no trace condition, the model saw the complete necklace of ten beads all at once without access to the trace.

As shown in Figure 4, observing and being able to integrate trace information enabled better accuracy with less data in each of the ten cases. In two cases, the model in the trace condition placed the highest probability it could on the ground-truth program after observing only a single necklace. In some cases, this evidence was integrated after observing just a few of the beads. In contrast, without trace information, a single necklace was never enough to provide conclusive evidence of the program and eight of the twelve programs were not given maximal probability in all 50 runs as there was often still significant uncertainty about the underlying program. The belief in π^* does not reach 1 in most cases because the DSL allows for isomorphic programs which are functionally equivalent but have different code. This places a ceiling on how much weight the model can put on the ground-truth program, π^*, even as the samples grow asymptotically towards infinity.

Behavioral Experiments

We test the predictions of our framework in two behavioral experiments run on Amazon Mechanical Turk. In both experiments the key between-subjects manipulation is whether the participants observed the necklaces being made in using the `seq` or `outin` program. These experiments test whether or not human learners are sensitive to the execution trace when making judgments or generating their own allowable necklaces after an observation. If people purely emulate the goal, their inferences will be invariant to the specific trace that generated the necklace. If people are imitating the sequences they have seen they will not generate any systematic inferences about novel necklaces. If instead, humans are using program induction to learn from others, we will see systematic influences on their judgments and behavior, and these will be influenced by trace information.

Experiment 1: Judgment

In the first experiment, participants (N=145) first observed an animation of an acceptable necklace being built (either in se-

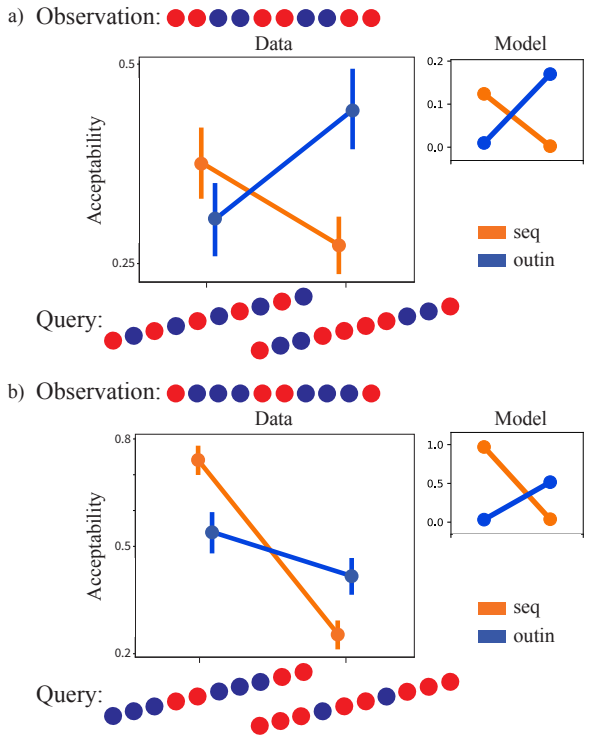

Figure 5: Participants judged whether the two necklaces on the X-axis (i.e., the queries) are allowable after observing the necklace above the graph being created with `seq` or `outin`. The inset shows model predictions which were made by assessing the probability that the space of programs inferred after observing the above necklace would generate the necklaces on the X-axis. Error bars show the standard error of the mean. The interaction terms for both (a) and (b) are statistically significant ($p < 0.01$)

quence or out-in, which randomly varied between subjects). Figure 5 shows the two necklaces used (presented in random order) that have both repeated and symmetric structure. After watching the creation of the necklace, participants made judgments about whether or not two new necklaces (shown on the X-axis) are also allowable. One of the necklaces had repeated but not symmetric structure while the other had symmetric but not repeated structure. These judgments were made on a 0-1 slider with end-points labeled "not allowable" and "allowable".

The model predicts that subjects who see the first necklace created with `seq` would find the novel necklace with repeated structure more likely to be acceptable (since they are more likely to be generated by a similar program) and that those who saw the initial necklace created `outin` would be more likely to judge the necklace with symmetric structure as allowable than the one with repeated structure. These predictions are replicated with a second set of novel stimuli shown in Figure 5. The pattern of human judgments supports all of the key model predictions. Participants' allowable judgments were significantly higher for necklaces that were

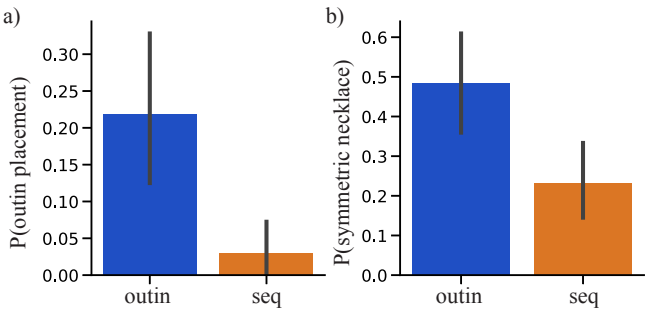

Figure 6: Statistics of participant generated necklaces. Error bars show the standard error of the mean. Both differences are statistically significant ($p < 0.01$)

consistent with the observed execution trace ($\beta = -0.26$, $CI_{95\%} = [-0.40, -0.08]$, $p = 0.004$).

Experiment 2: Generation

In a second experiment, participants (N=46) were shown the same two necklaces that contained both repeated and symmetric structure as in Experiment 1. Whether or not the participants saw the necklaces generated using `seq` or `outin` was varied between subjects. However, this time they were asked to generate two new necklaces that they believed would also be allowable. Participants used a mouse to drag beads onto empty squares in the necklace. Both the order and final necklaces were saved.

Given space constraints and diversity of the generated samples, we present a few qualitative results. The first prediction is that participants should be more likely to generate a necklace with mirror symmetry when they see the ambiguous necklace created with the `outin` program than with the `seq` program. The second prediction is that they should be more likely to adopt the `outin` placement method themselves when building a necklace. Figure 6 shows the results of the experiment on these two metrics. These differences were statistically significant. Participants who saw the necklace generated with `outin` were statistically more likely to generate symmetric necklaces ($\beta = -0.19$, $CI_{95\%} = [-0.34, -0.05]$, $p = 0.01$) and make their own necklace using an in-out procedure ($\beta = -0.24$, $CI_{95\%} = [-0.35, -0.15]$, $p < 0.001$). These results show that the trace structured participants internal representations and impacted both the structure of necklaces that they generated and also how they built the necklaces themselves.

Discussion

We have shown that human social learning involves "program induction": Inferring the generative structure that gives rise to expert performance. In our experiments participants clearly learned more than what products are permissible (as in emulation), because their judgments and behaviors were also deeply influenced by viewing the expert's process (or "trace"). And, they clearly learned more than a specific sequence of actions, because they readily generalized to novel

sequences of actions. Our model predicted the precise form of generalization. Thus, based on the observation of a single expert sequence of actions, participants were able to infer the hidden generative logic underlying their expertise. This form of social learning is efficient, high-fidelity, and robust.

These results may help to explain why human social learners occasionally reproduce even useless actions performed by apparent experts (so-called "overimitation"). This may not reflect blind imitation of actions, but rather a sophisticated (if sometimes misguided) attempt to infer a programmatic logic underlying these actions (Lyons, Young, & Keil, 2007). Experimental research suggests that great apes are much less prone to such overimitation (Tennie et al., 2006). An possibility is that apes are less likely to spontaneously interpret others actions as the product of abstract generative programs.

Additionally, social learning from program induction can be easily placed within a pedagogical framework, where teachers intentionally select better examples and learners interpret them as such (Shafto, Goodman, & Griffiths, 2014). For instance if one wants to teach that any necklace is allowable, they would wisely pick a necklace that has no repeated or symmetric structure, which rules out many alternative programs. This can generate additional efficiencies during social learning.

By representing cultural and social knowledge as a probabilistic program within a domain specific language, we cast the problem of observational learning as one of program induction. This formal approach invites connections to other forms of social learning, such as learning from feedback. In the context of our necklace paradigm, imagine that a novice is building a necklace and an expert intervenes to correct a specific action. If the feedback is given at the moment in the trace when the error occurs (like a stack trace in a debugger when a program crashes), then the learner might have a better chance of "debugging" her program and finding the right causal next step.

More generally, the challenge that a young child faces is to download `Culture.Zip`, to learn the abstract and generative knowledge of ones culture by a variety of means. As previous work emphasizes, program-like representations are compositional and modular—one could learn different pieces from different individuals and pick and choose the best parts or innovate on single sub-program without disrupting the rest of the accumulated knowledge. This enables innovation without the loss in fidelity that would occur if innovation could only come from unstructured random perturbations.

References

Amalric, M., Wang, L., Pica, P., Figueira, S., Sigman, M., & Dehaene, S. (2017). The language of geometry: Fast comprehension of geometrical primitives and rules in human adults and preschoolers. *PLoS computational biology*, *13*(1), e1005273.

Boyd, R. (2017). *A different kind of animal: How culture transformed our species* (Vol. 46). Princeton University Press.

Boyd, R., & Richerson, P. J. (1996). Why culture is common, but cultural evolution is rare. In *Proceedings-british academy* (Vol. 88, pp. 77–94).

Byrne, R. W., & Russon, A. E. (1998). Learning by imitation: A hierarchical approach. *Behavioral and brain sciences*, *21*(5), 667–684.

Caldwell, C. A., & Millen, A. E. (2009). Social learning mechanisms and cumulative cultural evolution: is imitation necessary? *Psychological Science*, *20*(12), 1478–1483.

Derex, M., Godelle, B., & Raymond, M. (2013). Social learners require process information to outperform individual learners. *Evolution: International Journal of Organic Evolution*, *67*(3), 688–697.

Henrich, J. (2015). *The secret of our success: how culture is driving human evolution, domesticating our species, and making us smarter*. Princeton University Press.

Herrmann, E., Call, J., Hernández-Lloreda, M. V., Hare, B., & Tomasello, M. (2007). Humans have evolved specialized skills of social cognition: The cultural intelligence hypothesis. *Science*, *317*(5843), 1360–1366.

Ho, M. K., Sanborn, S., Callaway, F., Bourgin, D., & Griffiths, T. (2018). Human priors in hierarchical program induction. *Computational Cognitive Neuroscience (CCN)*, *1*.

Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, *350*(6266), 1332–1338.

Lewis, H. M., & Laland, K. N. (2012). Transmission fidelity is the key to the build-up of cumulative culture. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *367*(1599), 2171–2180.

Lyons, D. E., Young, A. G., & Keil, F. C. (2007). The hidden structure of overimitation. *Proceedings of the National Academy of Sciences*, *104*(50), 19751–19756.

Rule, J., Schulz, E., Piantadosi, S. T., & Tenenbaum, J. B. (2018). Learning list concepts through program induction. *BioRxiv*, 321505.

Schulz, E., Tenenbaum, J. B., Duvenaud, D., Speekenbrink, M., & Gershman, S. J. (2017). Compositional inductive biases in function learning. *Cognitive psychology*, *99*, 44–79.

Shafto, P., Goodman, N. D., & Griffiths, T. L. (2014). A rational account of pedagogical reasoning: Teaching by, and learning from, examples. *Cognitive psychology*, *71*, 55–89.

Tenenbaum, J. B., & Griffiths, T. L. (2001). Generalization, similarity, and bayesian inference. *Behavioral and brain sciences*, *24*(4), 629–640.

Tennie, C., Call, J., & Tomasello, M. (2006). Push or pull: Imitation vs. emulation in great apes and human children. *Ethology*, *112*(12), 1159–1169.

Tomasello, M. (1999). *The cultural origins of human cognition.* Harvard University Press.