Probabilistic programming, program induction, and language of thought models

Brenden Lake & Todd Gureckis

email address for instructors:
instructors-ccm-spring2020@nyucll.org

course website:
https://brendenlake.github.io/CCM-site/
class world():
    def __init__(self):
        self.dict_strength = {}
    def clear(self): # used when sampling over possible world
        self.dict_strength = {}

    def strength(name):
        if name not in W.dict_strength:

    def lazy(name):
        return random.random() < 0.1

    def team_strength(team):
        # team : list of names
        mysum = 0.
        for name in team:
            if lazy(name):
                mysum += (strength(name) / 2.)
            else:
                mysum += strength(name)
        return mysum

    def winner(team1,team2):
        # team1 : list of names
        # team2 : list of names
        if team_strength(team1) > team_strength(team2):
            return team1
        else:
            return team2

    def beat(team1,team2):
        return winner(team1,team2) == team1

(Example from homework; Goodman et al., 2015)
Problem 1 (15 points)

2 Probabilistic inference

Each player has a latent "strength" value which describes his or her skill at tennis. This quantity is unobserved for each player, and it is a persistent property in the world. Therefore, the strength value of a player becomes known only after playing a game (as the result of the game is observed).

In a team of players, there is a team strength. The team strength is calculated by taking a weighted average of the strengths of the players in the team. The weight assigned to a player is either 0.3 if the player is in the team or 0.5 if the player is not in the team.

Problem 1: Implement a function that takes a list of players and returns the team strength. The function should consider the weights assigned to each player.

```python
import random

class world:
    def __init__(self):
        self.dict_strength = {}

    def clear(self):  # used when sampling over possible world
        self.dict_strength = {}

    def strength(self, name):
        if name not in W.dict_strength:
            W.dict_strength[name] = random.random()
        return W.dict_strength[name]

    def lazy(self, name):
        return random.random() < 0.1


W = world()

for i in range(10):
    for team in team1:
        for player in team:
            W.strength(player)
            if W.lazy(player):
                print(f'{player} is lazy in this game.

How strong is Bob?

Based on the above results, how strong do you think player TG is?

very weak very strong

OK
Probabilistic programs / probabilistic programming

- Probabilistic program: A probabilistic model defined in a structured description language (much like a programming language) using random programming primitives.

- Due to random primitives, every time the program executes it returns a different output.

- Probabilistic programs are a generalization of Bayesian networks, and many of the other Bayesian models we have discussed.

- Especially convenient when the prior is too complex to write down as a set of hypotheses, or the model is awkward or impossible to write as a Bayesian network.
Probabilistic programs: A simple example

Preliminary definitions

```python
def flip(theta=0.5):    # flip a coin with 'theta' chance of heads
    return random.random() < theta
```

Simple probabilistic program

A = flip()
B = flip()
C = flip()
D = A + B + C

Bayesian inference

\[
P(D)
\]

\[
P(A | D = 3)
\]

\[
P(A | D \geq 2)
\]

(again, notice productivity reasoning)

Key idea: A probabilistic program is a generative process for producing data

Example from Noah Goodman and Josh Tenenbaum
https://probmods.org/
Probabilistic program or Bayesian network?

\[ A = \text{flip()} \]
\[ B = \text{flip()} \]
\[ C = \text{flip()} \]
\[ D = A + B + C \]

In this case, the probabilistic program can be straightforwardly represented as a Bayesian network, although the program representation conveys more information.
Probabilistic programs: Another example

Simple probabilistic program (yet more complex than before)

A = flip()
B = flip()
C = flip()
if C:
    D = A + B + C
else:
    E = flip()
    F = (2*flip())**2
    D = A + B + C + E + F

Bayesian inference

\[ P(D) \]

\[ P(A|D \geq 2) \]
A = flip()
B = flip()
C = flip()

if C:
    D = A + B + C
else:
    E = flip()
    F = (2*flip())**2
    D = A + B + C + E + F

Bayesian networks (graphical models) do not have a mechanism for adding additional variables, and they lack general control structures that are relevant in both cognitive science and data science applications (if statements, for loops, while loops, recursion, etc.)
From Homework: Reasoning about tennis with probabilistic programs

```python
class world():
    def __init__(self):
        self.dict_strength = {}
    def clear(self):  # used when sampling over possible world
        self.dict_strength = {}

    def strength(name):
        if name not in W.dict_strength:
            W.dict_strength[name] = random.random()

    def lazy(name):
        return random.random() < 0.1

    def team_strength(team):
        # team : list of names
        mysum = 0.
        for name in team:
            if lazy(name):
                mysum += (strength(name) / 2.)
            else:
                mysum += strength(name)
        return mysum

    def winner(team1,team2):
        # team1 : list of names
        # team2 : list of names
        if team_strength(team1) > team_strength(team2):
            return team1
        else:
            return team2

    def beat(team1,team2):
        return winner(team1,team2) == team1
```

(see Goodman et al., 2015)
Reasoning about tennis with probabilistic programs

<table>
<thead>
<tr>
<th>confounded evidence</th>
<th>strong indirect evidence</th>
<th>weak indirect evidence</th>
<th>diverse evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,2)</td>
<td>(3,4)</td>
<td>(5,6)</td>
<td>(7,8)</td>
</tr>
<tr>
<td>A &gt; B</td>
<td>A &gt; B</td>
<td>A &gt; B</td>
<td>A &gt; B</td>
</tr>
<tr>
<td>A &gt; B</td>
<td>B &gt; C</td>
<td>B &lt; C</td>
<td>A &gt; C</td>
</tr>
<tr>
<td>A &gt; B</td>
<td>B &gt; D</td>
<td>B &lt; D</td>
<td>A &gt; D</td>
</tr>
</tbody>
</table>

singles games

doubles games

human judgements

model judgements

(see Gerstenberg et al., 2012; Goodman et al., 2015)
Bayesian network formulation depends on set of players

Each specification of the teams changes the Bayes net structure, and thus an additional modeling mechanism is needed to construct the Bayes nets.

sample anew for each game
Example probabilistic programming languages and software

WebPPL  
pyro  
Edward  
Church  
Stan
Example hypotheses in the LOT. These include subset-knower, CP-knower, and Mod-

Hypothesis 1
\[
\lambda S . \ (if \ (singleton? \ S) \\
\quad \ "one" \\
\quad \ (next \ (L \ (set-difference \ S \\
\quad \ (select \ S))))))
\]

Hypothesis 2
\[
\lambda S . \ (if \ (singleton? \ S) \\
\quad \ "one" \\
\quad \ (if \ (doubleton? \ S) \\
\quad \ "two" \\
\quad \ undefined))
\]

Hypothesis 3
\[
\lambda S . \ (if \ (singleton? \ S) \\
\quad \ "one" \\
\quad \ undefined)
\]

Hypothesis N ...

Which is the right program?

Most likely to have generated the data?
Program induction

• Data is generated from an unknown program, where unlike standard probabilistic programming, we don’t know the structure of the program.

• Prior over programs is usually defined by assuming a set of programming primitives and combination operations, which is also referred to as a “Language of thought” model in cognitive science (a la Jerry Fodor)

• More analogous to “structure learning” for Bayesian networks, where we are searching for the right causal model that generated the data.
Language of thought / program induction in Python

LOTlib

LOTlib is a Python 2 library for implementing "language of thought" models. A LOTlib model specifies a set of primitives and captures learning as inference over compositions of those primitives in order to express complex concepts. LOTlib permits lambda expressions, meaning that learners can come up with abstractions over compositions and define new...
Motivation: We need more than Bayesian networks to represent complex, real causal processes for generating data.

same causal process  different examples

(Figure credit: Hinton & Nair, 2006)

Is it growing too close to my house?
How will it grow if I trim it?

state-of-the-art neural net caption generation:
“A group of people standing on top of a beach”
Human-level concept learning through probabilistic program induction

Brenden M. Lake,1* Ruslan Salakhutdinov,2 Joshua B. Tenenbaum3

People learning new concepts can often generalize successfully from just a single example, yet machine learning algorithms typically require tens or hundreds of examples to perform with similar accuracy. People can also use learned concepts in richer ways than conventional algorithms—for action, imagination, and explanation. We present a computational model that captures these human learning abilities for a large class of simple visual concepts: handwritten characters from the world’s alphabets. The model represents concepts as simple programs that best explain observed examples under a Bayesian criterion. On a challenging one-shot classification task, the model achieves human-level performance while outperforming recent deep learning approaches. We also present several “visual Turing tests” probing the model’s creative generalization abilities, which in many cases are indistinguishable from human behavior.

Despite remarkable advances in artificial intelligence and machine learning, two aspects of human conceptual knowledge have eluded machine systems. First, for most interesting kinds of natural and man-made categories, people can learn a new concept from just one or a handful of examples, whereas standard algorithms in machine learning require tens or hundreds of examples to perform similarly. For instance, people may only need to see one example of a novel two-wheeled vehicle (Fig. 1A) in order to grasp the boundaries of the new concept, and even children can make meaningful generalizations via “one-shot learning” (1–3). In contrast, many of the leading approaches in machine learning are also the most data-hungry, especially “deep learning” models that have achieved new levels of performance on object and speech recognition benchmarks (4–9). Second, people learn richer representations than machines do, even for simple concepts (Fig. 1B), using them for a wider range of functions, including (Fig. 1, ii) creating new exemplars (10), (Fig. 1, iii) parsing objects into parts and relations (11), and (Fig. 1, iv) creating new abstract categories of objects based on existing categories (12, 13). In contrast, the best machine classifiers do not perform these additional functions, which are rarely studied and usually require specialized algorithms. A central challenge is to explain these two aspects of human-level concept learning: How do people learn new concepts from just one or a few examples? And how do people learn such abstract, rich, and flexible representations? An even greater challenge arises when putting them together: How can learning succeed from such sparse data yet also produce such rich representations? For any theory of

1Center for Data Science, New York University, 726 Broadway, New York, NY 10003, USA. 2Department of Computer Science and Department of Statistics, University of Toronto, 6 King’s College Road, Toronto, ON M5S 3G4, Canada. 3Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA.

*Corresponding author. E-mail: brenden@nyu.edu
Standard machine learning approach: Deep neural network with large amounts of data

MNIST:
10 classes of handwritten digits
6,000 examples each

Output classes
0 1 2...

Filter bank + non-linearity
Max pooling
Layer 2: 12 feature maps
Filter bank + non-linearity
Max pooling
Layer 1: 4 feature maps
Filter bank + non-linearity

Slide credit: Yann LeCun
People learn knowledge from less data

**less data**
People can learn a new concept from a single image.

[Diagram showing a single image and corresponding symbols and text in Kannada]

**more knowledge**
People can apply their knowledge flexibly to new tasks.

- Parsing
- Generating new concepts
- Generating new examples
**People learn knowledge from less data**

**less data**
People can learn a new concept from a single image.

![Image of a single character]

**more knowledge**
People can apply their knowledge flexibly to new tasks.

- parsing
  
  ![Image of parsing a character]

- generating new concepts
  
  ![Image of generating new concepts]

- generating new examples
  
  ![Image of generating new examples]
Omniglot stimulus set
(https://github.com/brendenlake/omniglot)

1600+ concepts
20 examples each
Human drawings

probabilistic motor programs

sub-strokes

primitive elements (sub-strokes)

people drawing a new character

Wednesday, October 17, 2012
The number of subparts $n_i$ for each part $i = 1, \ldots, k$, from their empirical distributions as measured from the background set. Second, a template for a part $S_i$ is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types $y$ and new token images $I(m)$ for $m = 1, \ldots, M$.

The function $f(\cdot, \cdot)$ transforms a subpart sequence and start location into a trajectory.

Human parses

Machine parses

Training item with model’s five best parses

Test items

Fig. 4. Inferring motor programs from images.

Parts are distinguished by color, with a colored dot indicating the beginning of a stroke and an arrowhead indicating the end. (A) The top row shows the five best programs discovered for an image along with their log-probability scores (Eq. 1). For classification, each program was refit to three new test images (left in image triplets), and the best-fitting parse (top right) is shown with its image reconstruction (bottom right) and classification score (log posterior predictive probability). Subpart breaks are shown as black dots.

(B) Nine human drawings of three characters (left) are shown with their ground truth parses (middle) and best model parses (right).

Human drawings

stroke order: 1 2 3 4 5
the number of subparts \( n \), for each part \( i = 1, \ldots, k \), from their empirical distributions as measured from the background set. Second, a template for a part \( S_i \) is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, or in a linear sequence.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types \( y \) and new token images \( I \) for \( m = 1, \ldots, M \).

The function \( f(\cdot, \cdot) \) transforms a subpart sequence and start location into a trajectory.

Human parses
Machine parses

Human drawings

Training item with model's five best parses
Test items

Fig. 4. Inferring motor programs from images. Parts are distinguished by color, with a colored dot indicating the beginning of a stroke and an arrowhead indicating the end. (A) The top row shows the five best programs discovered for an image along with their log-probability scores (Eq. 1). For classification, each program was refit to three new test images (left in image triplets), and the best-fitting parse (top right) is shown with its image reconstruction (bottom right) and classification score (log posterior predictive probability). Subpart breaks are shown as black dots. (B) Nine human drawings of three characters (left) are shown with their ground truth parses (middle) and best model parses (right).

stroke order: 1 2 3 4 5
The number of subparts $n_i$ for each part $i = 1, \ldots, k$, from their empirical distributions as measured from the background set. Second, a template for a part $S_i$ is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, or at the end, or connected.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types $y$ and new token images $I(m)$ for $m = 1, \ldots, M$.

The function $f(\cdot, \cdot)$ transforms a subpart sequence and start location into a trajectory.
the number of subparts \( n_i \), for each part \( i = 1, \ldots, k \), from their empirical distributions as measured from the background set. Second, a template for a part \( S_i \) is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, or in sequence, as specified by the program.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types \( y \) and new token images \( I(m) \) for \( m = 1, \ldots, M \).

The function \( f(\cdot, \cdot) \) transforms a subpart sequence and start location into a trajectory.

Human parses Machine parses

Training item with model’s five best parses

Test items

stroke order: 1 2 3 4 5

Fig. 4. Inferring motor programs from images. Parts are distinguished by color, with a colored dot indicating the beginning of a stroke and an arrowhead indicating the end. (A) The top row shows the five best programs discovered for an image along with their log-probability scores (Eq. 1). For classification, each program was refit to three new test images (left in image triplets), and the best-fitting parse (top right) is shown with its image reconstruction (bottom right) and classification score (log posterior predictive probability). Subpart breaks are shown as black dots. (B) Nine human drawings of three characters (left) are shown with their ground truth parses (middle) and best model parses (right).
The number of subparts $n_i$ for each part $i = 1, \ldots, k$ is determined from their empirical distributions as measured from the background set. Second, a template for a part $S_i$ is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, or at a particular location.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types $y$ and new token images $I(m)$ for $m = 1, \ldots, M$.

The function $f(\cdot, \cdot)$ transforms a subpart sequence and start location into a trajectory.

Human parses
Machine parses

Training item with model’s five best parses
Test items

Fig. 4. Inferring motor programs from images. Parts are distinguished by color, with a colored dot indicating the beginning of a stroke and an arrowhead indicating the end. (A) The top row shows the five best programs discovered for an image along with their log-probability scores (Eq. 1). For classification, each program was refit to three new test images (left in image triplets), and the best-fitting parse (top right) is shown with its image reconstruction (bottom right) and classification score (log posterior predictive probability). Subpart breaks are shown as black dots. (B) Nine human drawings of three characters (left) are shown with their ground truth parses (middle) and best model parses (right).

stroke order: 1 2 3 4 5
the number of subparts \( n_i \), for each part \( i = 1, \ldots, k \), from their empirical distributions as measured from the background set. Second, a template for a part \( S_i \) is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, or from a specified location.

Human parses

Machine parses

- Human drawings

![Human drawings](image)

stroke order: 1 2 3 4 5

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types \( y \) and new token images \( I_m \) for \( m = 1, \ldots, M \). The function \( f(\cdot, \cdot) \) transforms a subpart sequence and start location into a trajectory.

Fig. 4. Inferring motor programs from images. Parts are distinguished by color, with a colored dot indicating the beginning of a stroke and an arrowhead indicating the end. (A) The top row shows the five best programs discovered for an image along with their log-probability scores (Eq. 1). For classification, each program was refit to three new test images (left in image triplets), and the best-fitting parse (top right) is shown with its image reconstruction (bottom right) and classification score (log posterior predictive probability). Subpart breaks are shown as black dots. (B) Nine human drawings of three characters (left) are shown with their ground truth parses (middle) and best model parses (right).
Probabilistic program induction model of concept learning

- **Primitives**: (1D curvelets, 2D patches, 3D geons, actions, sounds, etc.)
- **Sub-parts**:
- **Parts**:
- **Object Template**:
- **Type Level**:
- **Token Level**:
- **Exemplars**:
- **Raw Data**:

Bayes' rule:

\[
P(\theta|I) = \frac{P(I|\theta)P(\theta)}{P(I)}
\]

- Latent program
- Raw binary image
- Rendered prior on parts, relations, etc.
Task: “Generate a new example”

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Handwritten characters are present in each cell, but not legible in this text representation.)
A “visual Turing test” for generating new examples
A “visual Turing test” for generating new examples

machine generated
A “visual Turing test” for generating new examples
A “visual Turing test” for generating new examples

machine generated

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Task: “Generate a new character from the same alphabet”

<table>
<thead>
<tr>
<th>א</th>
<th>ב</th>
<th>ג</th>
<th>ד</th>
</tr>
</thead>
<tbody>
<tr>
<td>ה</td>
<td>ו</td>
<td>ז</td>
<td>ח</td>
</tr>
</tbody>
</table>

3 seconds remaining
A “visual Turing test” for generating new concepts
A “visual Turing test” for generating new concepts

machine generated
A “visual Turing test” for generating new concepts
A “visual Turing test” for generating new concepts

machine generated
Probabilistic program induction

- Primitives: 1D curvelets, 2D patches, 3D geons, actions, sounds, etc.
- Sub-parts
- Parts
- Object template
- Type level
- Token level
- Exemplars
- Raw data

Bayes’ rule

$$P(\theta|I) = \frac{P(I|\theta)P(\theta)}{P(I)}$$

- Latent program
- Raw binary image
- Renderer
- Prior on parts, relations, etc.

Inference
Probabilistic program induction

primitives
(1D curvelets, 2D patches, 3D geons, actions, sounds, etc.)

sub-parts

parts

object template

type level

relation: attached along

---

**procedure** `GENERATETYPE`

\[
\kappa \leftarrow P(\kappa) \quad \triangleright \text{Sample number of parts}
\]

for \( i = 1 \ldots \kappa \) do

\[
n_i \leftarrow P(n_i|\kappa) \quad \triangleright \text{Sample number of sub-parts}
\]

for \( j = 1 \ldots n_i \) do

\[
s_{ij} \leftarrow P(s_{ij}|s_{i(j-1)}) \quad \triangleright \text{Sample sub-part sequence}
\]

end for

\[
R_i \leftarrow P(R_i|S_1, \ldots, S_{i-1}) \quad \triangleright \text{Sample relation}
\]

end for

\[
\psi \leftarrow \{\kappa, R, S\}
\]

return `GENERATETOKEN(\psi)` \quad \triangleright \text{Return program}`
Probabilistic program induction

primitives
(1D curvelets, 2D patches, 3D geons, actions, sounds, etc.)

sub-parts

parts

object template

procedure $\text{GENERATE}_T(\psi)$

\begin{align*}
\text{for } i = 1 \ldots \kappa \text{ do} \\
S_i^{(m)} &\leftarrow P(S_i^{(m)} | S_i) \\
L_i^{(m)} &\leftarrow P(L_i^{(m)} | R_i, T_1^{(m)}, \ldots, T_{i-1}^{(m)}) \\
T_i^{(m)} &\leftarrow f(L_i^{(m)}, S_i^{(m)}) \\
\text{end for} \\
A^{(m)} &\leftarrow P(A^{(m)}) \\
I^{(m)} &\leftarrow P(I^{(m)} | T^{(m)}, A^{(m)}) \\
\text{return } I^{(m)}
\end{align*}

$\triangleright$ Add motor variance

$\triangleright$ Sample part’s start location

$\triangleright$ Compose a part’s trajectory

$\triangleright$ Sample affine transform

$\triangleright$ Sample image

exemplars

raw data
Learning a prior distribution over programs

<table>
<thead>
<tr>
<th>learned action primitives</th>
<th>learned primitive transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>seed primitive</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1250 primitives
scale selective
translation invariant
Learning a prior distribution over programs

number of strokes

stroke start positions

Number of sub-strokes for a character with \( \kappa \) strokes

relations between strokes

number of sub-strokes

probability

relations between strokes

independent (34%) attached at start (5%) attached at end (11%) attached along (50%)

global transformations
Approximate probabilistic inference

\[ P(\theta|I) = \frac{P(I|\theta)P(\theta)}{P(I)} \]

Discrete \((K=5)\) approximation to posterior

\[ P(\theta|I) \approx \frac{\sum_{i=1}^{K} w_i \delta(\theta - \theta^i)}{\sum_{i=1}^{K} w_i} \]

such that

\[ w_i \propto P(\theta^i|I) \]

Intuition: Fit strokes to the observed pixels as closely as possible, with these constraints:

- fewer strokes
- high-probability primitive sequence
- use relations
- stroke order
- stroke directions
Approximate probabilistic inference

Step 1: characters as undirected graphs

Step 2: guided random parses

Step 3: Top-down fitting with gradient-based optimization
Human-level concept learning

the speed of learning

the richness of representation

parsing

generating new concepts

generating new examples
the number of subparts $n_i$, for each part $i = 1, \ldots, k$, from their empirical distributions as measured from the background set. Second, a template for a part $S_i$ is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types $y$ and new token images $I(m)$ for $m = 1, \ldots, M$.

The function $f(\cdot, \cdot)$ transforms a subpart sequence and start location into a trajectory.

Human parses
Machine parses

Human drawings

<table>
<thead>
<tr>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

Human parses

<table>
<thead>
<tr>
<th>.</th>
<th>.</th>
<th>.</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Machine parses

<table>
<thead>
<tr>
<th>.</th>
<th>.</th>
<th>.</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
the number of subparts $n_i$ for each part $i=1,\ldots,k$, from their empirical distributions as measured from the background set. Second, a template for a part $S_i$ is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, or together.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library, combining these subparts to make parts, and combining parts with relations to define simple programs. New tokens are generated by running these programs, which are then rendered as raw data. (B) Pseudocode for generating new types $y$ and new token images $I_m$ for $m=1,\ldots,M$.

The function $f(\cdot, \cdot)$ transforms a subpart sequence and start location into a trajectory.

Human parses

<table>
<thead>
<tr>
<th>Human drawings</th>
<th>Human parses</th>
<th>Machine parses</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{H}_1$</td>
<td>$\mathcal{H}_2$</td>
<td>$\mathcal{H}_3$</td>
</tr>
<tr>
<td>$\mathcal{H}_4$</td>
<td>$\mathcal{H}_5$</td>
<td>$\mathcal{H}_6$</td>
</tr>
<tr>
<td>$\mathcal{H}_7$</td>
<td>$\mathcal{H}_8$</td>
<td>$\mathcal{H}_9$</td>
</tr>
<tr>
<td>$\mathcal{H}_{10}$</td>
<td>$\mathcal{H}_{11}$</td>
<td>$\mathcal{H}_{12}$</td>
</tr>
</tbody>
</table>

stroke order: 1 2 3 4 5
the number of subparts $n_i$, for each part $i = 1, \ldots, k$, from their empirical distributions as measured from the background set. Second, a template for a part $S_i$ is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, for the next action.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types $y$ and new token images $I(m)$ for $m = 1, \ldots, M$. The function $f(\cdot, \cdot)$ transforms a subpart sequence and start location into a trajectory. Human parses Machine parses

![Human drawings](chart1.png)  
![Human parses](chart2.png)  
![Machine parses](chart3.png)

stroke order:  
- 1  
- 2  
- 3  
- 4  
- 5
the number of subparts \( n \), for each part \( i = 1, \ldots, k \), from their empirical distributions as measured from the background set. Second, a template for a part \( S_i \) is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, or at a specific location.

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types \( y \) and new token images \( I(m) \) for \( m = 1, \ldots, M \).

The function \( f(\cdot, \cdot) \) transforms a subpart sequence and start location into a trajectory.

Human parses

Machine parses

stroke order: 1 2 3 4 5
The number of subparts $n_i$, for each part $i = 1, \ldots, k$, from their empirical distributions as measured from the background set. Second, a template for a part $S_i$ is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, SCIENCE sciencemag.org 00 MONTH 2015 • VOL 000 ISSUE 0000

<table>
<thead>
<tr>
<th>Human drawings</th>
<th>Human parses</th>
<th>Machine parses</th>
</tr>
</thead>
<tbody>
<tr>
<td>stroke order:  1</td>
<td>stroke order:  2</td>
<td>stroke order:  3</td>
</tr>
</tbody>
</table>

Fig. 3. A generative model of handwritten characters. (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types $y$ and new token images $I(m)$ for $m = 1, \ldots, M$.

The function $f(\cdot, \cdot)$ transforms a subpart sequence and start location into a trajectory.
The number of subparts $n_i$ for each part $i = 1, \ldots, k$, from their empirical distributions as measured from the background set. Second, a template for a part $S_i$ is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters for each subpart. Last, parts are roughly positioned to begin either independently, at the beginning, or together, at the end.

**Fig. 3. A generative model of handwritten characters.** (A) New types are generated by choosing primitive actions (color coded) from a library (i), combining these subparts (ii) to make parts (iii), and combining parts with relations to define simple programs (iv). New tokens are generated by running these programs (v), which are then rendered as raw data (vi). (B) Pseudocode for generating new types $y$ and new token images $I(m)$ for $m = 1, \ldots, M$.

The function $f(\cdot, \cdot)$ transforms a subpart sequence and start location into a trajectory.

**Fig. 4. Inferring motor programs from images.** Parts are distinguished by color, with a colored dot indicating the beginning of a stroke and an arrowhead indicating the end. (A) The top row shows the five best programs discovered for an image along with their log-probability scores (Eq. 1). For classification, each program was refit to three new test images (left in image triplets), and the best-fitting parse (top right) is shown with its image reconstruction (bottom right) and classification score (log posterior predictive probability). Subpart breaks are shown as black dots. (B) Nine human drawings of three characters (left) are shown with their ground truth parses (middle) and best model parses (right).

**Human parses**

<table>
<thead>
<tr>
<th>Human drawings</th>
<th>Human parses</th>
<th>Machine parses</th>
</tr>
</thead>
<tbody>
<tr>
<td>stroke order: 1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Human parses

Machine parses
Human-level concept learning

the speed of learning

the richness of representation

parsing

generating new examples

generating new concepts
the pen (Fig. 3A, ii). To construct a new character type, first the model samples the number of parts \( k \) and the number of subparts \( n_i \) for each part \( i = 1, \ldots, k \), from the empirical distribution as measured from the background set. Second, a template for a part \( S_i \) is constructed by sampling subparts from a set of discrete primitive actions learned from the background set (Fig. 3A, i), such that the probability of the next action depends on the previous. Third, parts are then grounded as parameterized curves (splines) by sampling the control points and scale parameters.

![One-shot classification](image)

\[ P(I_{new} | I_{old}) \]
Do people represent *static characters* by their causal dynamics?

**Behavioral evidence**

- Writing experience influences perception
  (Freyd, 1983; Tse & Cavanagh, 2000; Knoblich & Prinz, 2001; James & Gauthier, 2009).

- Inferring the dynamics from static letters.
  (Babcock & Freyd, 1988)

**Neuroimaging evidence**

- Writing experience changes the functional specialization of visual cortex for letters.
  (James & Atwood, 2009; James, 2010)

- Motor areas of cortex respond to *static* letters.
  (Anderson et al., 1990; Loncamp et al., 2003; James & Gauthier, 2006; Longcamp et al., 2006; Longcamp et al., 2010)

(Figure credit: Hinton & Nair, 2006)
One-shot classification performance

After all models pre-trained on 30 alphabets of characters.

Program induction models

- People
- BPL
- BPL Lesion (wrong prior)
- BPL Lesion (no compositionality)

Deep neural networks

(deep Siamese Convnet (Koch, Zemel, Salakhutdinov. 2015)
- Deep Convnet
- Hierarchical Deep

Error rate (%)
Human-level concept learning

the speed of learning

the richness of representation

parsing

generating new concepts

generating new examples
A “visual Turing test” for generating new concepts
A “visual Turing test” for generating new concepts

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
<tr>
<td>![Image of characters]</td>
<td>![Image of characters]</td>
</tr>
</tbody>
</table>
More large-scale behavioral experiments

Generating new examples (dynamic)

Generating new concepts (from type)

Generating new concepts (unconstrained)

59% correct in visual Turing test
6 of 30 judges above chance

49% correct in visual Turing test
8 of 35 judges above chance

51% correct in visual Turing test
2 of 25 judges above chance
## More large-scale behavioral experiments

### Generating new examples (dynamic)

<table>
<thead>
<tr>
<th>Human or Machine?</th>
<th>Alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human or Machine?</td>
<td><img src="image1" alt="Characters" /></td>
</tr>
</tbody>
</table>

59% correct in visual Turing test
6 of 30 judges above chance

### Generating new concepts (from type)

<table>
<thead>
<tr>
<th>Alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image2" alt="Characters" /></td>
</tr>
</tbody>
</table>

### Generating new concepts (unconstrained)

<table>
<thead>
<tr>
<th>Alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Characters" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Human or Machine?</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4" alt="Characters" /></td>
</tr>
</tbody>
</table>

51% correct in visual Turing test
2 of 25 judges above chance

<table>
<thead>
<tr>
<th>Human or Machine?</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5" alt="Characters" /></td>
</tr>
</tbody>
</table>

49% correct in visual Turing test
8 of 35 judges above chance
Bootstrapping in a language of thought: A formal model of numerical concept learning

Steven T. Piantadosi a, *, Joshua B. Tenenbaum b, Noah D. Goodman c

a Department of Brain and Cognitive Sciences, University of Rochester, United States
b Department of Brain and Cognitive Sciences, MIT, United States
c Department of Psychology, Stanford University, United States

ABSTRACT

In acquiring number words, children exhibit a qualitative leap in which they transition from understanding a few number words, to possessing a rich system of interrelated numerical concepts. We present a computational framework for understanding this inductive leap as the consequence of statistical inference over a sufficiently powerful representational system. We provide an implemented model that is powerful enough to learn number word meanings and other related conceptual systems from naturalistic data. The model shows that bootstrapping can be made computationally and philosophically well-founded as a theory of number learning. Our approach demonstrates how learners may combine core cognitive operations to build sophisticated representations during the course of development, and how this process explains observed developmental patterns in number word learning.
Children’s development of numerical concepts

“Give-a-number” task

“Give me two”
“Give me three”

(Wynn, 1990; Wynn, 1992)
Children’s development of numerical concepts

Children progress through a series of stages
• “one-knower”, “two-knower,” “three-known,” “four-known” (sometimes), and then “cardinal-principle knower”

Example: “two knower”

give me one: 🍎
give me two: 🍎🍎
give me three: 🍎🍎🍎 OR 🍎🍎🍎 OR 🍎🍎🍎 ...

(inconsistent; arbitrary response beyond “two”)
Children’s development of numerical concepts

• Critically, children can count well-beyond the range of their “knower” status, yet they don’t understand the meaning of the numbers.
• Transition from “N-knower” to “CP-knower” happens roughly between ages 2.5 and 3.5

---

**Patterns of success in give-a-number task in Experiment 3**

<table>
<thead>
<tr>
<th>Success pattern</th>
<th>Number of children</th>
<th>Mean age</th>
<th>Counting ability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Mean</strong></td>
</tr>
<tr>
<td>1 2 3 5 6</td>
<td>1</td>
<td>2:8</td>
<td>3.00</td>
</tr>
<tr>
<td>one-knower</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>3</td>
<td>3:0</td>
<td>4.67</td>
</tr>
<tr>
<td>two-knower</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 3</td>
<td>2</td>
<td>2:11</td>
<td>4.50</td>
</tr>
<tr>
<td>three-knower</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4</td>
<td>4</td>
<td>3:5</td>
<td>5.75</td>
</tr>
<tr>
<td>CP-knower</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>7</td>
<td>3:7</td>
<td>6.00</td>
</tr>
</tbody>
</table>

Note: “+” indicates success on a numerosity; “−” indicates failure.

(data from Wynn, 1990)
Programming primitives allowed in the language of thought model

### Functions mapping sets to truth values
- **singleton? X**
  - Returns true iff the set X has exactly one element
- **doubleton? X**
  - Returns true iff the set X has exactly two elements
- **tripleton? X**
  - Returns true iff the set X has exactly three elements

### Functions on sets
- **set-difference X Y**
  - Returns the set that results from removing Y from X
- **union X Y**
  - Returns the union of sets X and Y
- **intersection X Y**
  - Returns the intersect of sets X and Y
- **select X**
  - Returns a set containing a single element from X

### Logical functions
- **and P Q**
  - Returns TRUE if P and Q are both true
- **or P Q**
  - Returns TRUE if either P or Q is true
- **not P**
  - Returns TRUE iff P is false
- **if P X Y**
  - Returns X iff P is true, Y otherwise

### Functions on the counting routine
- **next W**
  - Returns the word after W in the counting routine
- **prev W**
  - Returns the word before W in the counting routine
- **equal-word? W V**
  - Returns TRUE if W and V are the same word

### Recursion
- **L S**
  - Returns the result of evaluating the entire current lambda expression on set

(Piantadosi, Tenenbaum & Goodman)
though, we also do not include a successor function, meaning a function which maps the representation of $N$ to the representation of $N + 1$. While neither a successor function or a Mod-$N$ function is assumed, both can be constructed in this representational system.

3.4. Hypothesis space for the model

The hypothesis space for the learning model consists of all ways these primitives can be combined to form lambda expressions—lexicons—which map sets to number words. This therefore provides a space of exact numerical meanings. In a certain sense, the learning model is therefore quite restricted in the set of possible meanings it will consider. It will not ever, for instance, map a set to a different concept or a word not on the count list. This restriction is computationally convenient and developmentally plausible.

Wynn (1992) provided evidence that children know number words refer to some kind of numerosity before they know their exact meanings. For example, even children who did not know the exact meaning of “four” pointed to a display with several objects over a display with few when asked “Can you show me four balloons?” They did not show this pattern for nonsense word such as “Can you show me blicket balloons?” Similarly, children map number words to some type of cardinality, even if they do not know which cardinalities (Lipton & Spelke, 2006; Sarnecka & Gelman, 2004). Bloom and Wynn (1997) suggest that perhaps this can be accounted for by a learning mechanism that uses syntactic cues to determine that number words are a class with a certain semantics.

Within the domain of functions which map sets to words, this hypothesis space is relatively unrestricted. Some example hypotheses are shown in Fig. 1.

The hypothesis space contains functions with partial numerical knowledge—for instance, hypotheses that have the correct meaning for “one” and “two”, but not “three” or above. For instance, the 2-knower hypothesis takes an argument $S$, and first checks if $(\text{singleton? } S)$ is true—if $S$ has one element. If it does, the function returns “one”. If not, this hypothesis returns the value of $(\text{if } (\text{doubleton? } S) \text{ “two” } \text{undef})$. This expression is another if-statement, one which returns “two” if $S$ has two elements, and undef otherwise. Thus, this hypothesis represents a 2-knower who has the correct meanings for “one” and “two”, but not for any higher numbers. Intuitively, one could build much more complex and interesting hypotheses in this format—for instance, ones that check more complex properties of $S$ and return other word values.

Fig. 1 also shows an example of a CP-knower lexicon. This function makes use of the counting routine and recursion. First, this function checks if $S$ contains a single element, returning “one” if it does. If not, this function calls set-difference on $S$ and $(\text{select } S)$. This has the effect of choosing an element from $S$ and removing it, yielding a subset of $S$.

Example hypotheses in a language of thought

Example set $S$

$\lambda S$. indicates a function that takes a set $S$ as an argument.

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-knower</td>
<td>$\lambda S. (\text{if } (\text{singleton? } S) \text{ “one” } \text{undef})$</td>
</tr>
<tr>
<td>Two-knower</td>
<td>$\lambda S. (\text{if } (\text{singleton? } S) \text{ “one” } (\text{if } (\text{doubleton? } S) \text{ “two” } \text{undef}))$</td>
</tr>
<tr>
<td>Three-knower</td>
<td>$\lambda S. (\text{if } (\text{singleton? } S) \text{ “one” } (\text{if } (\text{doubleton? } S) \text{ “two” } (\text{if } (\text{tripleton? } S) \text{ “three” } \text{undef})))$</td>
</tr>
<tr>
<td>CP-knower</td>
<td>$\lambda S. (\text{if } (\text{singleton? } S) \text{ “one” } (\text{next } (L (\text{set-difference } S \text{ (select } S))))))$</td>
</tr>
<tr>
<td>Singular-Plural</td>
<td>$\lambda S. (\text{if } (\text{singleton? } S) \text{ “one” } \text{“two”})$</td>
</tr>
<tr>
<td>Mod-5</td>
<td>$\lambda S. (\text{if } (\text{or } (\text{singleton? } S) \text{ (equal-word? } (L (\text{set-difference } S \text{ (select } S)) \text{ “five”})) \text{ “one” } (\text{next } (L (\text{set-difference } S \text{ (select } S))))))$</td>
</tr>
</tbody>
</table>
Defining a prior distribution over programs
(a “Probabilistic Language of Thought”)

Formalism used:
probabilistic context-free grammar

\[
\text{Word} \\
\quad \text{B} \\
\quad \begin{cases}
\text{if } B \text{ the } W \text{ else } W \\
\text{if } (\text{singleton? } S) \text{ then } W \text{ else } W \\
\text{if } (\text{and } B \text{ } B) \text{ W else } W \\
\text{if } (\text{singleton? } S) \text{ then } \text{“one” else } W \\
\text{if } (\text{singleton? } S) \text{ then } \text{“one” else } \text{“undefined”} \\
\end{cases}
\]
Probabilistic model over programs

Example $L$

\[ \lambda S . \ (\text{if} \ (\text{singleton?} \ S) \ \\
\text{"one"} \ \\
\text{undef}) \]

Probabilistic model

\[ P(L) \quad \text{prior on programs } L \]

(defined with probabilistic grammar)

\[ P(D|L) \quad \text{Noisy likelihood where right number is usually returned, but with some noise} \]

Bayes’ rule for learning programs

\[ P(L|D) = \frac{P(D|L)P(L)}{P(D)} \]
Learning as program induction

Program ($L$) $\lambda S . \begin{cases} \text{“one”} \\
\text{(next (L (set-difference S (select S)))}) \end{cases}$

Data ($D$)

$P(L|D) = \frac{P(D|L)P(L)}{P(D)}$
Results: Program induction model follows a similar developmental trajectory

![Graph showing posterior probability over amount of data for different knower types.]

- **One-knower**
  \[ \lambda S. \ (\text{if} \ (\text{singleton?} \ S) \n \ "one"\n \ "undef") \]

- **Two-knower**
  \[ \lambda S. \ (\text{if} \ (\text{singleton?} \ S) \n \ "one")\n \ (\text{if} \ (\text{doubleton?} \ S) \n \ "two"\n \ "undef\)) \]

- **Three-knower**
  \[ \lambda S. \ (\text{if} \ (\text{singleton?} \ S) \n \ "one")\n \ (\text{if} \ (\text{doubleton?} \ S) \n \ "two")\n \ (\text{if} \ (\text{triplleton?} \ S) \n \ "three")\n \ "undef") \]

- **CP-knower**
  \[ \lambda S. \ (\text{if} \ (\text{singleton?} \ S) \n \ "one")\n \ (\text{if} \ (\text{doubleton?} \ S) \n \ "two")\n \ (\text{if} \ (\text{triplleton?} \ S) \n \ "three")\n \ (\text{next} \ (L \ (\text{set-difference} \ S \n \ (\text{select} \ S)))))) \]

Note: The graph visualizes the posterior probability of exhibiting each type of behavior as a function of the amount of data. This represents the learning curve for different knower types: One-knower, Two-knower, Three-knower, and CP-knower. The graph shows how the posterior probability changes with increasing data for each category.
Case study: Learning by asking questions

Question Asking as Program Generation

Anselm Rothe\textsuperscript{1} \hspace{1cm} Brenden M. Lake\textsuperscript{1,2} \hspace{1cm} Todd M. Gureckis\textsuperscript{1}
anselm@nyu.edu \hspace{1cm} brenden@nyu.edu \hspace{1cm} todd.gureckis@nyu.edu
\textsuperscript{1}Department of Psychology \hspace{1cm} \textsuperscript{2}Center for Data Science
New York University

Abstract

A hallmark of human intelligence is the ability to ask rich, creative, and revealing questions. Here we introduce a cognitive model capable of constructing human-like questions. Our approach treats questions as formal programs that, when executed on the state of the world, output an answer. The model specifies a probability distribution over a complex, compositional space of programs, favoring concise programs that help the agent learn in the current context. We evaluate our approach by modeling the types of open-ended questions generated by humans who were attempting to learn about an ambiguous situation in a game. We find that our model predicts what questions people will ask, and can creatively produce novel questions that were not present in the training set. In addition, we compare a number of model variants, finding that both question informativeness and complexity are important for producing human-like questions.

1 Introduction

In active machine learning, a learner is able to query an oracle in order to obtain information that is expected to improve performance. Theoretical and empirical results show that active learning can speed acquisition for a variety of learning tasks [see 21, for a review]. Although impressive, most work on active machine learning has focused on relatively simple types of information requests (most often a request for a supervised label). In contrast, humans often learn by asking far richer questions which more directly target the critical parameters in a learning task. A human child might ask "Do all dogs have long tails?" or "What is the difference between cats and dogs?" [2]. A long term goal of artificial intelligence (AI) is to develop algorithms with a similar capacity to learn by asking rich questions. Our premise is that we can make progress toward this goal by better understanding human question asking abilities in computational terms [cf. 8].

To that end, in this paper, we propose a new computational framework that explains how people construct rich and interesting queries within in a particular domain. A key insight is to model questions as programs that, when executed on the state of a possible world, output an answer. For example, a program corresponding to "Does John prefer coffee to tea?" would return \texttt{True} for all possible world states where this is the correct answer and \texttt{False} for all others. Other questions may return different types of answers. For example "How many sugars does John take in his coffee?" would return a number 0, 1, 2, etc. depending on the world state. Thinking of questions as syntactically well-formed programs recasts the problem of question asking as one of program synthesis. We show that this powerful formalism offers a new approach to modeling question asking in humans and may eventually enable more human-like question asking in machines.

We evaluate our model using a data set containing natural language questions asked by human participants in an information-search game [19]. Given an ambiguous situation or context, our model can predict what questions human learners will ask by capturing constraints in how humans construct semantically meaningful questions. The method successfully predicts the frequencies of
active learning for people and machines

rich human questions

- How do they grow their babies?
- Why is he up in the tree?
- What is the difference between a shark and a fish?

simple machine questions

- What is the category label of this object?
- What is the category label of this object?
- What is the category label of this object?
Experiment: Free-form question asking

(e.g., Markant & Gureckis, 2012, 2014)

Hidden configuration of ships

3 ships (blue, purple, red)
3 possible sizes (2-4 tiles)
1.6 million possible configurations
Experiment: Free-form question asking

Hidden configuration of ships

3 ships (blue, purple, red)
3 possible sizes (2-4 tiles)
1.6 million possible configurations

Phase 1: Sampling

Phase 2: Question asking

Is the red ship horizontal?

Constraints
- one word answers
- no combinations

Repeated for 18 different hidden configurations
<table>
<thead>
<tr>
<th>N</th>
<th>Location/standard queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>What color is at [row][column]?</td>
</tr>
<tr>
<td>24</td>
<td>Is there a ship at [row][column]?</td>
</tr>
</tbody>
</table>
| 31 | Is there a [color_incl_water] tile at [row][column]?

### Region queries

| 4  | Is there any ship in row [row]?          |
| 9  | Is there any part of the [color] ship in row [row]?
| 5  | How many tiles in row [row] are occupied by ships? |
| 1  | Are there any ships in the bottom half of the grid? |
| 10 | Is there any ship in column [column]?
| 10 | Is there any part of the [color] ship in column [column]?
| 3  | Are all parts of the [color] ship in column [column]?
| 2  | How many tiles in column [column] are occupied by ships? |
| 1  | Is any part of the [color] ship in the left half of the grid? |

### Ship size queries

| 185 | How many tiles is the [color] ship? |
| 71  | Is the [color] ship [size] tiles long? |
| 8   | Is the [color] ship [size] or more tiles long? |
| 5   | How many ships are [size] tiles long? |
| 8   | Are any ships [size] tiles long? |
| 2   | Are all ships [size] tiles long? |
| 2   | Are all ships the same size? |
| 2   | Do the [color1] ship and the [color2] ship have the same size? |
| 3   | Is the [color1] ship longer than the [color2] ship? |
| 3   | How many tiles are occupied by ships? |

### Ship orientation queries

| 94  | Is the [color] ship horizontal? |
| 7   | How many ships are horizontal? |
| 3   | Are there more horizontal ships than vertical ships? |
| 1   | Are all ships horizontal? |
| 4   | Are all ships vertical? |
| 7   | Are the [color1] ship and the [color2] ship parallel? |

### Adjacency queries

| 12  | Do the [color1] ship and the [color2] ship touch? |
| 6   | Are any of the ships touching? |
| 9   | Does the [color] ship touch any other ship? |
| 2   | Does the [color] ship touch both other ships? |

### Demonstration queries

| 14  | What is the location of one [color] tile? |
| 28  | At what location is the top left part of the [color] ship? |
| 5   | At what location is the bottom right part of the [color] ship? |
### Location/standard queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>What color is at [row][column]?</td>
</tr>
<tr>
<td>24</td>
<td>Is there a ship at [row][column]?</td>
</tr>
</tbody>
</table>
| 31 | Is there a [color_incl_water] tile at [row][column]?

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Is there any ship in column [column]?</td>
</tr>
</tbody>
</table>
| 10 | Is there any part of the [color] ship in column [column]?
| 3  | Are all parts of the [color] ship in column [column]?
| 2  | How many tiles in column [column] are occupied by ships? |
| 1  | Is any part of the [color] ship in the left half of the grid?  

### Ship size queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>185</td>
<td>How many tiles is the [color] ship?</td>
</tr>
<tr>
<td>71</td>
<td>Is the [color] ship [size] tiles long?</td>
</tr>
<tr>
<td>8</td>
<td>Is the [color] ship [size] or more tiles long?</td>
</tr>
<tr>
<td>5</td>
<td>How many ships are [size] tiles long?</td>
</tr>
<tr>
<td>8</td>
<td>Are any ships [size] tiles long?</td>
</tr>
<tr>
<td>2</td>
<td>Are all ships [size] tiles long?</td>
</tr>
<tr>
<td>2</td>
<td>Are all ships the same size?</td>
</tr>
<tr>
<td>2</td>
<td>Do the [color1] ship and the [color2] ship have the same size?</td>
</tr>
<tr>
<td>3</td>
<td>Is the [color1] ship longer than the [color2] ship?</td>
</tr>
<tr>
<td>3</td>
<td>How many tiles are occupied by ships?</td>
</tr>
</tbody>
</table>

### Ship orientation queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>Is the [color] ship horizontal?</td>
</tr>
<tr>
<td>7</td>
<td>How many ships are horizontal?</td>
</tr>
<tr>
<td>3</td>
<td>Are there more horizontal ships than vertical ships?</td>
</tr>
<tr>
<td>1</td>
<td>Are all ships horizontal?</td>
</tr>
<tr>
<td>4</td>
<td>Are all ships vertical?</td>
</tr>
<tr>
<td>7</td>
<td>Are the [color1] ship and the [color2] ship parallel?</td>
</tr>
</tbody>
</table>

### Adjacency queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Do the [color1] ship and the [color2] ship touch?</td>
</tr>
<tr>
<td>6</td>
<td>Are any of the ships touching?</td>
</tr>
<tr>
<td>9</td>
<td>Does the [color] ship touch any other ship?</td>
</tr>
<tr>
<td>2</td>
<td>Does the [color] ship touch both other ships?</td>
</tr>
</tbody>
</table>

### Demonstration queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>What is the location of one [color] tile?</td>
</tr>
<tr>
<td>28</td>
<td>At what location is the top left part of the [color] ship?</td>
</tr>
<tr>
<td>5</td>
<td>At what location is the bottom right part of the [color] ship?</td>
</tr>
</tbody>
</table>
As a first stage of our analysis, we manually coded commonalities in the meaning of questions in order to determine sequence of tiles (which are the past queries and answers). An important distinction contrasts between questions. Rich queries also work with them.

### Location/standard queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>What color is at [row][column]?</td>
</tr>
<tr>
<td>24</td>
<td>Is there a ship at [row][column]?</td>
</tr>
<tr>
<td>31</td>
<td>Is there a [color_incl_water] tile at [row][column]?</td>
</tr>
</tbody>
</table>

### Region queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Is there any ship in row [row]?</td>
</tr>
<tr>
<td>9</td>
<td>Is there any part of the [color] ship in row [row]?</td>
</tr>
<tr>
<td>5</td>
<td>How many tiles in row [row] are occupied by ships?</td>
</tr>
<tr>
<td>1</td>
<td>Are there any ships in the bottom half of the grid?</td>
</tr>
<tr>
<td>10</td>
<td>Is there any ship in column [column]?</td>
</tr>
<tr>
<td>10</td>
<td>Is there any part of the [color] ship in column [column]?</td>
</tr>
<tr>
<td>3</td>
<td>Are all parts of the [color] ship in column [column]?</td>
</tr>
<tr>
<td>2</td>
<td>How many tiles in column [column] are occupied by ships?</td>
</tr>
</tbody>
</table>

### Ship size queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>185</td>
<td>How many tiles is the [color] ship?</td>
</tr>
<tr>
<td>71</td>
<td>Is the [color] ship [size] tiles long?</td>
</tr>
<tr>
<td>8</td>
<td>Is the [color] ship [size] or more tiles long?</td>
</tr>
<tr>
<td>5</td>
<td>How many ships are [size] tiles long?</td>
</tr>
<tr>
<td>8</td>
<td>Are any ships [size] tiles long?</td>
</tr>
<tr>
<td>2</td>
<td>Are all ships [size] tiles long?</td>
</tr>
</tbody>
</table>

### Ship orientation queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>Is the [color] ship horizontal?</td>
</tr>
<tr>
<td>7</td>
<td>How many ships are horizontal?</td>
</tr>
<tr>
<td>3</td>
<td>Are there more horizontal ships than vertical ships?</td>
</tr>
<tr>
<td>1</td>
<td>Are all ships horizontal?</td>
</tr>
<tr>
<td>4</td>
<td>Are all ships vertical?</td>
</tr>
<tr>
<td>7</td>
<td>Are the [color1] ship and the [color2] ship parallel?</td>
</tr>
</tbody>
</table>

### Adjacency queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Do the [color1] ship and the [color2] ship touch?</td>
</tr>
<tr>
<td>6</td>
<td>Are any of the ships touching?</td>
</tr>
<tr>
<td>9</td>
<td>Does the [color] ship touch any other ship?</td>
</tr>
<tr>
<td>2</td>
<td>Does the [color] ship touch both other ships?</td>
</tr>
</tbody>
</table>

### Demonstration queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>What is the location of one [color] tile?</td>
</tr>
<tr>
<td>28</td>
<td>At what location is the top left part of the [color] ship?</td>
</tr>
<tr>
<td>5</td>
<td>At what location is the bottom right part of the [color] ship?</td>
</tr>
</tbody>
</table>
### Location/standard queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>What color is at [row][column]?</td>
</tr>
<tr>
<td>24</td>
<td>Is there a ship at [row][column]?</td>
</tr>
<tr>
<td>31</td>
<td>Is there a [color_incl_water] tile at [row][column]?</td>
</tr>
</tbody>
</table>

### Region queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Is there any ship in row [row]?</td>
</tr>
<tr>
<td>9</td>
<td>Is there any part of the [color] ship in row [row]?</td>
</tr>
<tr>
<td>5</td>
<td>How many tiles in row [row] are occupied by ships?</td>
</tr>
<tr>
<td>1</td>
<td>Are there any ships in the bottom half of the grid?</td>
</tr>
<tr>
<td>10</td>
<td>Is there any ship in column [column]?</td>
</tr>
<tr>
<td>10</td>
<td>Is there any part of the [color] ship in column [column]?</td>
</tr>
<tr>
<td>3</td>
<td>Are all parts of the [color] ship in column [column]?</td>
</tr>
<tr>
<td>2</td>
<td>How many tiles in column [column] are occupied by ships?</td>
</tr>
<tr>
<td>1</td>
<td>Is any part of the [color] ship in the left half of the grid?</td>
</tr>
</tbody>
</table>

### Ship size queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>185</td>
<td>How many tiles is the [color] ship?</td>
</tr>
<tr>
<td>71</td>
<td>Is the [color] ship [size] tiles long?</td>
</tr>
<tr>
<td>8</td>
<td>Is the [color] ship [size] or more tiles long?</td>
</tr>
<tr>
<td>5</td>
<td>How many ships are [size] tiles long?</td>
</tr>
<tr>
<td>8</td>
<td>Are any ships [size] tiles long?</td>
</tr>
<tr>
<td>2</td>
<td>Are all ships [size] tiles long?</td>
</tr>
<tr>
<td>2</td>
<td>Are all ships the same size?</td>
</tr>
<tr>
<td>2</td>
<td>Do the [color1] ship and the [color2] ship have the same size?</td>
</tr>
<tr>
<td>3</td>
<td>Is the [color1] ship longer than the [color2] ship?</td>
</tr>
<tr>
<td>3</td>
<td>How many tiles are occupied by ships?</td>
</tr>
</tbody>
</table>

### Ship orientation queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>Is the [color] ship horizontal?</td>
</tr>
<tr>
<td>7</td>
<td>How many ships are horizontal?</td>
</tr>
<tr>
<td>3</td>
<td>Are there more horizontal ships than vertical ships?</td>
</tr>
<tr>
<td>1</td>
<td>Are all ships horizontal?</td>
</tr>
<tr>
<td>4</td>
<td>Are all ships vertical?</td>
</tr>
<tr>
<td>7</td>
<td>Are the [color1] ship and the [color2] ship parallel?</td>
</tr>
</tbody>
</table>

### Adjacency queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>What is the location of one [color] tile?</td>
</tr>
<tr>
<td>28</td>
<td>At what location is the top left part of the [color] ship?</td>
</tr>
<tr>
<td>5</td>
<td>At what location is the bottom right part of the [color] ship?</td>
</tr>
</tbody>
</table>

### Demonstration queries

<table>
<thead>
<tr>
<th>N</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>What is the location of one [color] tile?</td>
</tr>
<tr>
<td>28</td>
<td>At what location is the top left part of the [color] ship?</td>
</tr>
<tr>
<td>5</td>
<td>At what location is the bottom right part of the [color] ship?</td>
</tr>
</tbody>
</table>
How do people think of a question to ask? 

question asking as program generation

Game primitives

“Size of the blue ship?”
(size Blue )

“Color at tile A1?”
(color A1 )

“Orientation of the blue ship?”
(orient Blue )

Primitive operators

(+ X X )

( = X X )

Novel questions

“What is the total size of all the ships?”
(+
(+
(+ ( size Blue )
( size Red )
)
( size Purple )
)

“Are the blue ship and the red ship parallel?”
( =
( ( orient Blue )
( orient Red )
)
### Questions as programs

<table>
<thead>
<tr>
<th>GROUP</th>
<th>QUESTION</th>
<th>FUNCTION</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>What color is at A1?</td>
<td>location</td>
<td>(color A1)</td>
</tr>
<tr>
<td></td>
<td>Is there a ship at A1?</td>
<td>locationA</td>
<td>(not (= (color A1) Water))</td>
</tr>
<tr>
<td></td>
<td>Is there a blue tile at A1?</td>
<td>locationD</td>
<td>(= (color A1) Blue)</td>
</tr>
<tr>
<td>segmentation</td>
<td>Is there any ship in row 1?</td>
<td>row</td>
<td>(&gt; (+ (map (λ x (and (= (row x) 1) (not (= (color x) Water))) (set A1 ... F6))) 0)</td>
</tr>
<tr>
<td></td>
<td>Is there any part of the blue ship in row 1?</td>
<td>rowD</td>
<td>(&gt; (+ (map (λ x (and (= (row x) 1) (= (color x) Blue))) (set A1 ... F6))) 0)</td>
</tr>
<tr>
<td></td>
<td>Are all parts of the blue ship in row 1?</td>
<td>rowDL</td>
<td>(&gt; (+ (map (λ x (and (= (row x) 1) (= (color x) Blue))) (set A1 ... F6))) 1)</td>
</tr>
<tr>
<td></td>
<td>How many tiles in row 1 are occupied by ships?</td>
<td>rowNA</td>
<td>(+ (map (λ x (and (= (row x) 1) (not (= (color x) Water))) (set A1 ... F6)))</td>
</tr>
<tr>
<td></td>
<td>Are there any ships in the bottom half of the grid?</td>
<td>rowX2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Is there any ship in column 1?</td>
<td>col</td>
<td>(&gt; (+ (map (λ x (and (= (col x) 1) (not (= (color x) Water))) (set A1 ... F6))) 0)</td>
</tr>
<tr>
<td></td>
<td>Is there any part of the blue ship in column 1?</td>
<td>colD</td>
<td>(&gt; (+ (map (λ x (and (= (col x) 1) (= (color x) Blue))) (set A1 ... F6))) 0)</td>
</tr>
<tr>
<td></td>
<td>Are all parts of the blue ship in column 1?</td>
<td>colDL</td>
<td>(&gt; (+ (map (λ x (and (= (col x) 1) (= (color x) Blue))) (set A1 ... F6))) 1)</td>
</tr>
<tr>
<td></td>
<td>How many tiles in column 1 are occupied by ships?</td>
<td>colNA</td>
<td>(+ (map (λ x (and (= (col x) 1) (not (= (color x) Water))) (set A1 ... F6)))</td>
</tr>
<tr>
<td></td>
<td>Is any part of the blue ship in the left half of the grid?</td>
<td>colX1</td>
<td>...</td>
</tr>
<tr>
<td>ship size</td>
<td>How many tiles is the blue ship?</td>
<td>shipsize</td>
<td>(size Blue)</td>
</tr>
<tr>
<td></td>
<td>Is the blue ship 3 tiles long?</td>
<td>shipsizeD</td>
<td>(= (size Blue) 3)</td>
</tr>
<tr>
<td></td>
<td>Is the blue ship 3 or more tiles long?</td>
<td>shipsizem</td>
<td>(or (= (size Blue) 3) (&gt; (size Blue) 3))</td>
</tr>
<tr>
<td></td>
<td>How many ships are 3 tiles long?</td>
<td>shipsizeln</td>
<td>(+ (map (λ x (= (size x) 3)) (set Blue Red Purple)))</td>
</tr>
<tr>
<td></td>
<td>Are any ships 3 tiles long?</td>
<td>shipsizeda</td>
<td>(&gt; (+ (map (λ x (= (size x) 3)) (set Blue Red Purple))) 0)</td>
</tr>
<tr>
<td></td>
<td>Are all ships 3 tiles long?</td>
<td>shipsizeld</td>
<td>(= (+ (map (λ x (= (size x) 3)) (set Blue Red Purple))) 3)</td>
</tr>
<tr>
<td></td>
<td>Are all ships the same size?</td>
<td>shipsizel</td>
<td>(= (map (λ x (size x)) (set Blue Red Purple)))</td>
</tr>
<tr>
<td></td>
<td>Do the blue ship and the red ship have the same size?</td>
<td>shipsizex1</td>
<td>(= (size Blue) (size Red))</td>
</tr>
<tr>
<td></td>
<td>Is the blue ship longer than the red ship?</td>
<td>shipsizex2</td>
<td>(&gt; (size Blue) (size Red))</td>
</tr>
<tr>
<td>orientation</td>
<td>How many tiles is the blue ship horizontal?</td>
<td>totalshipsize</td>
<td>(+ (map (λ x (size x)) (set Blue Red Purple)))</td>
</tr>
</tbody>
</table>
Defining an infinite set of questions through compositionality

Booleans

TRUE FALSE (= C C) (> N N) ...

(numbers)

Number Color Orientation Location

Color

Is there a red tile at A1?

Is there a blue tile at A1?

A

Answer

B

Boolean

= (color L) C

(> (size C) N)

... (orient C)

... (orient Red)

Is the red ship horizontal or vertical?

Is the red ship larger than 2 tiles?
Question asking as program generation

Example for ideal observer finding the optimal/most informative question:
Question asking as program generation

Example for ideal observer finding the optimal/most informative question:

```
(+
  (+
    (* 100 (size Purple))
    (* 10 (size Blue)))
(size Red)
)
```
Question asking as program generation

Example for ideal observer finding the optimal/most informative question:

```
(+
  (+
    (* 100 (size Purple))
    (* 10 (size Blue)))
[size Red]
```

Learning a probabilistic generative model of questions:

Goal: predict human questions in novel scenarios

- \( \mathcal{X} \) : question
- \( f(\cdot) \) : features (Expected Info. Gain, length, answer type, etc.)
- \( \theta \) : trainable parameters

energy:

\[
\mathcal{E}(x) = \theta_1 f_1(x) + \theta_2 f_2(x) + \cdots + \theta_K f_K(x)
\]

generative model:

\[
P(x; \theta) = \frac{\exp^{-\mathcal{E}(x)}}{\sum_{x' \in \mathcal{X}} \exp^{-\mathcal{E}(x')}}
\]
Question asking as program generation

Example for ideal observer finding the optimal/most informative question:

```
(+
  (+
    (* 100 (size Purple))
    (* 10 (size Blue)))
  (size Red))
```

Learning a probabilistic generative model of questions:

Goal: predict human questions in novel scenarios

\[ x \quad \text{: question} \]

\[ f(\cdot) \quad \text{: features (Expected Info. Gain, length, answer type, etc.)} \]

\[ \theta \quad \text{: trainable parameters} \]

Energy:

\[
\mathcal{E}(x) = \theta_1 f_1(x) + \theta_2 f_2(x) + \cdots + \theta_K f_K(x)
\]

Generative model:

\[
P(x; \theta) = \frac{\exp^{-\mathcal{E}(x)}}{\sum_{x' \in X} \exp^{-\mathcal{E}(x')}}
\]

Compositionality is key!
Asking novel questions through program generation

**Context 7**

<table>
<thead>
<tr>
<th>EIG</th>
<th>Question/Program</th>
<th>Human</th>
</tr>
</thead>
</table>
| 2.44 | How many tiles are occupied by ships?  
(++) (map (lambda x (size x)) (set Blue Red Purple)) |       |
| 1.79 | How many ships are 4 tiles long?  
(++) (map (lambda x (= (size x) 4)) (set Blue Red Purple)) |       |
| Energy | Question/Program |       |
| 6.53 | What is the column of the bottom right water tile?  
(col (bottomright (coloredTiles Water))) |       |
| 7.88 | What is the row of the top left purple tile?  
(rowl (topleft (coloredTiles Purple))) |       |
| 8.90 | Are all the ships horizontal?  
(all (map (lambda x (= H (orient x))) (set Blue Red Purple))) |       |
| 10.51 | What is the column of the bottom right of the tiles with the same color as tile 3E?  
(col (bottomright (coloredTiles (color 3E)))) |       |
| 12.89 | Are any of the ship sizes greater than 2?  
(any (map (lambda x (> (size x) 2)) (set Blue Red Purple))) |       |

**Context 9**

<table>
<thead>
<tr>
<th>EIG</th>
<th>Question/Program</th>
<th>Human</th>
</tr>
</thead>
</table>
| 1.59 | How many tiles in row 4 are occupied by ships?  
(++) (map (lambda y (and (= (rowl y) 4) (not (= (color y) Water))) (set 1A ... 6F))) |       |
| 1.56 | How many tiles is the purple ship?  
(size Purple) |       |
| Energy | Question/Program |       |
| 7.48 | What is the column of the bottom right blue tile?  
(col (bottomright (coloredTiles Blue))) |       |
| 8.74 | How many tiles have the same color as tile 4A?  
(setSize (coloredTiles (color 4A))) |       |
| 9.94 | What is the top left of all the ship tiles?  
(topleft (setDifference (set 1A ... 6F) (coloredTiles Water))) |       |
| 10.98 | What is the color of the top left of the tiles that have the same color as 5C?  
(color (topleft (coloredTiles (color 5C)))) |       |
| 16.34 | Are blue and purple ships touching and red and purple not touching (or vice versa)?  
(== (touch Blue Purple) (not (touch Red Purple))) |       |
### Asking novel questions through program generation

**Context 7**

<table>
<thead>
<tr>
<th>EIG</th>
<th>Question/Program</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.44</td>
<td>How many tiles are occupied by ships?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(+ (map (lambda x (size x)) (set Blue Red Purple)))</td>
<td></td>
</tr>
<tr>
<td>1.79</td>
<td>How many ships are 4 tiles long?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(+ (map (lambda x (= (size x) 4)) (set Blue Red Purple)))</td>
<td></td>
</tr>
</tbody>
</table>

Energy Question/Program

| 6.53 | What is the column of the bottom right water tile?   |                               |
|      | (coll (bottomright (coloredTiles Water)))           |                               |
| 7.88 | What is the row of the top left purple tile?        |                               |
|      | (rowl (topleft (coloredTiles Purple)))              |                               |
| **8.90** | Are all the ships horizontal?                      |                               |
|      | (all (map (lambda x (= H (orient x))) (set Blue Red Purple))) |                               |
| 10.51| What is the column of the bottom right of the tiles with the same color as tile 3E? | |
|      | (coll (bottomright (coloredTiles (color 3E))))      |                               |
| 12.89| Are any of the ship sizes greater than 2?            |                               |
|      | (any (map (lambda x (> (size x) 2)) (set Blue Red Purple))) |                               |

**Context 9**

<table>
<thead>
<tr>
<th>EIG</th>
<th>Question/Program</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.59</td>
<td>How many tiles in row 4 are occupied by ships?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(+ (map (lambda y (and (= (rowl y) 4) (not (= (color y) Water))) (set 1A ... 6F)))</td>
<td></td>
</tr>
<tr>
<td>1.56</td>
<td>How many tiles is the purple ship?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(size Purple)</td>
<td></td>
</tr>
</tbody>
</table>

Energy Question/Program

| 7.48 | What is the column of the bottom right blue tile?    |                               |
|      | (coll (bottomright (coloredTiles Blue)))            |                               |
| 8.74 | How many tiles have the same color as tile 4A?       |                               |
|      | (setSize (coloredTiles (color 4A)))                 |                               |
| **9.94** | What is the top left of all the ship tiles?         |                               |
|      | (topleft (setDifference (set 1A ... 6F) (coloredTiles Water))) |                               |
| 10.98| What is the color of the top left of the tiles that have the same color as 5C? | |
|      | (color (topleft (coloredTiles (color 5C))))         |                               |
| **16.34** | Are blue and purple ships touching and red and purple not touching (or vice versa)? | |
|      | (== (touch Blue Purple) (not (touch Red Purple)))    |                               |