Learning and the language of thought

Noah D. Goodman
Stanford University

ICCV SIG-11
November 12, 2011
Motivation

What are thoughts made of?
(Concepts... but what are they?)
Motivation

What are thoughts made of?
(Concepts... but what are they?)

How are concepts learned?
Statistics and composition
Statistics and composition

Thought is useful in an uncertain world
Statistics and composition

Thought is useful in an uncertain world
Thought is useful in an uncertain world.

Why did he yell at me?
Statistics and composition

Thought is useful in an uncertain world

Why did he yell at me?

He wanted to hurt me.
He thought I was a telemarketer.
Statistics and composition

Thought is useful in an uncertain world

Why did he yell at me?
He wanted to hurt me.
He thought I was a telemarketer.
Statistics and composition

Thought is useful in an uncertain world

Why did he yell at me?
Belief
Desire
Action

He wanted to hurt me.
He thought I was a telemarketer.

Probabilistic inference
Statistics and composition

Thought is useful in an uncertain world

Probabilistic inference
Statistics and composition

Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference
Statistics and composition

Thought is useful in an uncertain world

Probabilistic inference

Thought is productive: “the infinite use of finite means”

..a big green bear who loves chocolate..
Statistics and composition

Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference

..a big green bear who loves chocolate..
Statistics and composition

Thought is useful in an uncertain world.

Probabilistic inference

Thought is productive: “the infinite use of finite means”
Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference
Statistics and composition

Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference

Compositional representations

$p = mv$
Statistics and composition

Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference

Compositional representations

∀x King(x) ⇒ Man(x)
∀y Man(y) ⇔ ¬Woman(y)
Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference

Compositional representations
Statistics and composition

Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference

Compositional representations
Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference

Compositional representations
Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference

Generative models

Compositional representations
Statistics and composition

Probabilistic language of thought hypothesis

Thought is useful in an uncertain world

Thought is productive: “the infinite use of finite means”

Probabilistic inference

Generative models

Compositional representations
Probabilistic LoT

- The **probabilistic** language of thought hypothesis:
  - Mental representations are compositional,
  - Their meaning is probabilistic,
  - They encode generative knowledge,
  - Hence, they support thinking and learning by probabilistic inference.
The **probabilistic** language of thought hypothesis:

- Mental representations are compositional,
- Their meaning is probabilistic,
- They encode generative knowledge,
- Hence, they support thinking and learning by probabilistic inference.

Can this hypothesis be formalized?
Probabilistic generative models

- Bayes nets:
  - Mental model of the causal process that gives rise to observations.
  - What’s “hidden inside” the nodes and arrows?
Probabilistic generative models

- Bayes nets:
  - Mental model of the causal process that gives rise to observations.
  - What’s “hidden inside” the nodes and arrows?

<table>
<thead>
<tr>
<th></th>
<th>TB</th>
<th>no TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>cough flu</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>no flu</td>
<td>0.7</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Probabilistic generative models

• Bayes nets:
  • Mental model of the causal process that gives rise to observations.

<table>
<thead>
<tr>
<th></th>
<th>TB</th>
<th>no TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>cough</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flu</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>no flu</td>
<td>0.7</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Flu
TB

TB

Flu

TB

Cough
Probabilistic generative models

- Bayes nets:
  - Mental model of the causal process that gives rise to observations.

\[
P(H|d) \propto P(d|H)P(H)
\]
Probabilistic generative models

- Bayes nets:
  - Mental model of the causal process that gives rise to observations.
  - What’s “hidden inside” the nodes and arrows?

\[
P(H|d) \propto P(d|H)P(H)
\]

<table>
<thead>
<tr>
<th></th>
<th>TB</th>
<th>no TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>flu</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>no flu</td>
<td>0.7</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Probabilistic generative models

- **Bayes nets:**
  - Mental model of the causal process that gives rise to observations.
  - What’s “hidden inside” the nodes and arrows?

$$P(H|d) \propto P(d|H)P(H)$$

<table>
<thead>
<tr>
<th></th>
<th>TB</th>
<th>no TB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>flu</strong></td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td><strong>no flu</strong></td>
<td>0.7</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Probabilistic generative models

- Bayes nets:
  - Mental model of the causal process that gives rise to observations.
  - What’s “hidden inside” the nodes and arrows?

\[ P(H|d) \propto P(d|H)P(H) \]

<table>
<thead>
<tr>
<th></th>
<th>TB</th>
<th>no TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>flu</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>no flu</td>
<td>0.7</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Probabilistic generative models

\[ P(H|d) \propto P(d|H)P(H) \]

- **Bayes nets:**
  - Mental model of the causal process that gives rise to observations.
  - What’s “hidden inside” the nodes and arrows?

<table>
<thead>
<tr>
<th></th>
<th>Flu</th>
<th>TB</th>
<th>no TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>cough</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flu</td>
<td>0.8</td>
<td></td>
<td>0.7</td>
</tr>
<tr>
<td>no flu</td>
<td>0.7</td>
<td></td>
<td>0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>want fame</th>
<th>want sleep</th>
<th>..</th>
</tr>
</thead>
<tbody>
<tr>
<td>going to workshop</td>
<td>0.7</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>it’s raining</td>
<td>0.002</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
λ calculus
\textbf{\(\lambda\) calculus}

\textbullet\ Notation:

\textbullet\ Function have parentheses on the wrong side: 
\[
\sin x
\]

\textbullet\ Operators always go at the beginning: 
\[
+ x y
\]
\( \lambda \) calculus

- **Notation:**
  - Function have parentheses on the wrong side: \((\sin x)\) \((+ x y)\)
  - Operators always go at the beginning:

- \( \lambda \) makes functions, define binds values to symbols:

\[
(\text{define } \text{double} \(\lambda (x) (+ x x)\))
\]
\textbf{\textlambda{} calculus}

- **Notation:**
  - Function have parentheses on the wrong side: \((\sin x)\)
  - Operators always go at the beginning: \((+ x y)\)

- \textbf{\textlambda{}} makes functions, define binds values to symbols:

\begin{align*}
\text{(define } & \textbf{double} \ (\lambda \ (x) \ (+ \ x \ x))) \text{newline}
\text{(double } & 3) \Rightarrow 6
\end{align*}
λ calculus

• Notation:
  • Function have parentheses on the wrong side: \((\sin x)\)
  • Operators always go at the beginning: \((+ x y)\)

• \(\lambda\) makes functions, define binds values to symbols:

\[
\begin{align*}
(\text{define } \textbf{double} & \quad (\lambda \ (x) \ (+ \ x \ x))) \\
(\text{define } \textbf{repeat} & \quad (\lambda \ (f) \ (\lambda \ (x) \ (f \ (f \ x))))))
\end{align*}
\]

\((\text{double } 3) \Rightarrow 6\)
\[ \lambda \text{ calculus} \]

- **Notation:**
  - Function have parentheses on the wrong side: \((\sin x)\)
  - Operators always go at the beginning: 
    \((+ x y)\)

- **\(\lambda\) makes functions, define binds values to symbols:**

\[
(\text{define } \text{double} \quad (\lambda \ (x) \ (+ \ x \ x)))
\]

\[
(\text{define } \text{repeat} \quad (\lambda \ (f) \ (\lambda \ (x) \ (f \ (f \ x))))))
\]

\[
(((\text{repeat} \ \text{double}) \ 3) \quad \Rightarrow \ 12)
\]

\[
(\text{double} \ 3) \quad \Rightarrow \ 6
\]
\[ \lambda \text{ calculus} \]

- Notation:
  - Function have parentheses on the wrong side:
  - Operators always go at the beginning:

\( (\sin x) \)
\( (+ x y) \)

- \( \lambda \) makes functions, define binds values to symbols:

\[
(\text{define } \text{double} (\lambda (x) (+ x x)))
\]

\[
(\text{define } \text{repeat} (\lambda (f) (\lambda (x) (f (f x)))))
\]

\[
(\text{define } 2\text{nd-derivative} (\text{repeat} \text{derivative}))
\]

\[
(\text{double } 3) \Rightarrow 6
\]

\[
((\text{repeat } \text{double}) 3) \Rightarrow 12
\]
Church

Random primitives:

(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Random primitives:

\[
\begin{align*}
\text{(define } & \mathbf{a} \text{ (flip } 0.3)) \Rightarrow 1 \\
\text{(define } & \mathbf{b} \text{ (flip } 0.3)) \\
\text{(define } & \mathbf{c} \text{ (flip } 0.3)) \\
\text{(+ a b c)}
\end{align*}
\]
Church

Random primitives:

(define a (flip 0.3)) => 1
(define b (flip 0.3)) => 0
(define c (flip 0.3))
(+ a b c)

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Church

Random primitives:

\[(\text{define } a \ (\text{flip 0.3})) \Rightarrow 1\]
\[(\text{define } b \ (\text{flip 0.3})) \Rightarrow 0\]
\[(\text{define } c \ (\text{flip 0.3})) \Rightarrow 1\]
\[(+ \ a \ b \ c)\]
Random primitives:

```
(define a (flip 0.3)) => 1
(define b (flip 0.3)) => 0
(define c (flip 0.3)) => 1
(+ a b c) => 2
```
Church

Random primitives:

```
(define a (flip 0.3)) => 1 0
(define b (flip 0.3)) => 0 0
(define c (flip 0.3)) => 1 0
(+ a b c) => 2 0
```

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Church

Random primitives:

```scheme
(define a (flip 0.3)) => 1 0 0
(define b (flip 0.3)) => 0 0 0
(define c (flip 0.3)) => 1 0 1
(+ a b c) => 2 0 1
```

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Church

Random primitives:

\[
\begin{align*}
\text{(define } a \text{ (flip 0.3))} & \Rightarrow 1 \ 0 \ 0 \\
\text{(define } b \text{ (flip 0.3))} & \Rightarrow 0 \ 0 \ 0 \\
\text{(define } c \text{ (flip 0.3))} & \Rightarrow 1 \ 0 \ 1 \\
\text{(+ a b c)} & \Rightarrow 2 \ 0 \ 1 \ldots
\end{align*}
\]
Church

Random primitives:

(define a (flip 0.3))  =>  1 0 0
(define b (flip 0.3))  =>  0 0 0
(define c (flip 0.3))  =>  1 0 1
(+ a b c)              =>  2 0 1

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Theorem: Any computable distribution can be represented by a Church expression.

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Random primitives:

\[
\begin{align*}
(\texttt{define } a & \ (\texttt{flip 0.3})) && \Rightarrow 1 \ 0 \ 0 \\
(\texttt{define } b & \ (\texttt{flip 0.3})) && \Rightarrow 0 \ 0 \ 0 \\
(\texttt{define } c & \ (\texttt{flip 0.3})) && \Rightarrow 1 \ 0 \ 1 \\
(+ \ a \ b \ c) && \Rightarrow 2 \ 0 \ 1 ..
\end{align*}
\]

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
**Church**

Random primitives:

```lisp
(define a (flip 0.3))
(define b (flip 0.3))
(define c (flip 0.3))
(+ a b c)
```

Conditioning (inference):

```lisp
(query
  (define a (flip 0.3))
  (define b (flip 0.3))
  (define c (flip 0.3))
  (+ a b c)
  (= (+ a b) 1)
)
```

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Church

Random primitives:

(define \texttt{a} (flip 0.3)) => 1 0 0
(define \texttt{b} (flip 0.3)) => 0 0 0
(define \texttt{c} (flip 0.3)) => 1 0 1
(+ \texttt{a} \texttt{b} \texttt{c}) => 2 0 1

Conditioning (inference):

(query
   (define \texttt{a} (flip 0.3))
   (define \texttt{b} (flip 0.3))
   (define \texttt{c} (flip 0.3))
   (+ \texttt{a} \texttt{b} \texttt{c}) Query
   (= (+ \texttt{a} \texttt{b} 1))

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Church

Random primitives:

(define a (flip 0.3)) => 1 0 0
(define b (flip 0.3)) => 0 0 0
(define c (flip 0.3)) => 1 0 1
(+ a b c) => 2 0 1...

Conditioning (inference):

(query
  (define a (flip 0.3))
  (define b (flip 0.3))
  (define c (flip 0.3))
  (+ a b c) Query
  (= (+ a b) 1)) Condition, must be true

Goodman, Mansinghka, Roy, Bonawitz, Tenenabum (2008)
Random primitives:

(define a (flip 0.3))  => 1 0 0
(define b (flip 0.3))  => 0 0 0
(define c (flip 0.3))  => 1 0 1
(+ a b c)               => 2 0 1...

Conditioning (inference):

(query
  (define a (flip 0.3))
  (define b (flip 0.3))
  (define c (flip 0.3))
  (+ a b c)          Query
  (= (+ a b) 1))      Condition, must be true

Goodman, Mansinghka, Roy, Bonawitz, Tenenbaum (2008)
Example: PCFG

(define S (λ () (if (flip)
    (list (S) (S))
    's)))
Example: PCFG

\[(\text{define } \mathbf{S} (\lambda () \ (\text{if} \ (\text{flip})
\quad \ (\text{list} \ (\mathbf{S}) \ (\mathbf{S}))
\quad \ 's)))\]

\[(\text{define } (\text{tree-unfold symbol})
\quad (\text{if} \ (\text{terminal? symbol})
\quad \ symbol
\quad \ (\text{map} \ \text{tree-unfold} \ (\text{transition symbol}))))\]
PLoT

• Formalizing the PLoT: Mental representations (concepts) are functions in a stochastic lambda calculus (e.g. Church).

• Separate the process of inference from representations and the inferences they license (Cf. Marr’s levels).

• (However see our work on inference....)

http://projects.csail.mit.edu/church
If concepts are probabilistic programs, then concept learning is *probabilistic program induction.*
If concepts are probabilistic programs, then concept learning is \textit{probabilistic program induction}.

(query
 (define concept (sample-PLoT-expression))
 concept
 (and (= (noisy (sample concept)) obs1)
      (= (noisy (sample concept)) obs2)
      ...
))
Categorization

Medin & Schaffer (1978):
Categorization

Medin & Schaffer (1978):

“These are Feps”
Categorization

Medin & Schaffer (1978):

“These are Feps”

“These are not Feps”
Categorization

Medin & Schaffer (1978):

“These are Feps”

“Is this a Fep?”

“These are not Feps”

"These are Feps"

"These are not Feps"

"Is this a Fep?"
Categorization


Graded judgements

- "These are Feps"
- "These are not Feps"
- "Is this a Fep?"

- Graded judgements
Categorization


- Graded judgements
- Typicality

% Fep

“These are Feps”

“Is this a Fep?”

“These are not Feps”

• Graded judgements
• Typicality
Categorization


- Graded judgements
- Typicality
- Prototype enhancement

% Fep

"These are Feps"

"These are not Feps"

"Is this a Fep?"
Categorization

- Rule-based category learning:
  - Infinitely many concepts formed compositionally.

- Statistical category learning:
  - Graded inferences from sparse, noisy evidence.
“It’s a Fep if it has flat head and round wings”
(define fep? (\x. (and (flat-head x) (round-wings x))))

Generating rules

Feps: non-Feps:
(define fep?
  (λ (x)
    (and (flat-head x)
         (round-wings x))))

(fep? )

=> true
(define fep? 
(\( x \) 
  (and (flat-head \( x \))
  (round-wings \( x \))))
)

(fep? )
=> true

Generating rules

(Feps:)

(non-Feps:)

(define rule-generator 
(\() 
(\( if (flip 0.3) 
  (sample-feature) 
  (combine-rules (sample-feature) 
  (rule-generator)))
)

(define combine-rules 
(\( r1 \( r2 \)
  (\( x \) (and (r1 \( x \)) (r2 \( x \)))
)
(define \textit{fep?} \\
(\lambda \ (x) \\
 (\text{and} \ (\text{flat-head} \ x) \\
 (\text{round-wings} \ x)))) \\
\text{(fep? \ \ \ ?)} \\
=> \text{true} 

\text{Generating rules} \\
\text{Feps:} \\
\text{non-Feps:} \\
(\text{define \textit{rule-generator}} \\
(\lambda () \\
 (\text{if} \ (\text{flip} \ 0.3) \\
 (\text{sample-feature}) \\
 (\text{combine-rules} \ (\text{sample-feature}) \\
 (\text{rule-generator})))) \\
(\text{define \textit{combine-rules}} \\
(\lambda \ (r1 \ r2) \\
 (\lambda \ (x) \ (\text{and} \ (r1 \ x) \ (r2 \ x))))
(define fep? (λ (x) (and (flat-head x) (round-wings x))))

(fep? )
=> true

(define rule-generator (λ () (if (flip 0.3) (sample-feature) (combine-rules (sample-feature) (rule-generator))))

(define combine-rules (λ (r1 r2) (λ (x) (and (r1 x) (r2 x)))))
Generating rules

\[
\begin{align*}
\text{(define } \text{fep?} & \text{(} \lambda (x) \text{)} \\
& \quad \text{(and (flat-head x) (round-wings x)))} \\
\text{(fep? } \text{)} & \Rightarrow \text{ true}
\end{align*}
\]

\[
\begin{align*}
\text{(define } \text{rule-generator} & \text{(} \lambda () \text{)} \\
& \quad \text{(if (flip 0.3) (sample-feature) (combine-rules (sample-feature) (rule-generator)))}
\end{align*}
\]

\[
\begin{align*}
\text{(define } \text{combine-rules} & \text{(} \lambda (r1 r2) \\
& \quad \text{(} \lambda (x) \text{) (and (r1 x) (r2 x))})
\end{align*}
\]

- Longer rules have lower probability (Occam’s razor).
Generating rules

Put uncertainty over rule probabilities:

```
(define rule-prob (uniform 0 1))
(define rule-generator
  (λ ()
    (if (flip rule-prob)
        ...
    ))
```

Generate disjunctive normal form (DNF) rules:

```
(define fep?
  (λ (x)
    (and (or (flat-head x) ...)
         (or (round-wings x) ...)
         ...)))
```

The general idea: grammar-based induction.
Inducing rules

Inference:

\[
\text{(query}
\]
\[
\text{(define rule (rule-generator))}
\]
\[
\text{(rule )}
\]
\[
\text{(and (= (rule ) true)}
\]
\[
\text{(= (rule ) false)}
\]
\[
\text{(...))}
\]
Inducing rules

Inference:

(query
  (define rule (rule-generator))
  (rule )
  (and (= (noisy (rule )) true)
       (= (noisy (rule )) false)
     ...
))

Observation noise:

(define noisy
  (λ (bit) (if (flip b) bit (not bit))))
Categorization

Human data vs. Rule-induction model predictions

- Fep
- non-Fep
- transfer

Goodman, et al. (2007, 2008a, 2008b)
Categorization

The human linearly classification trained nenflugel learn

<table>
<thead>
<tr>
<th></th>
<th>A6</th>
<th>A5</th>
<th>A4</th>
<th>A3</th>
<th>A2</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>T7</th>
<th>T6</th>
<th>T3</th>
<th>T2</th>
<th>T1</th>
<th>T5</th>
<th>T4</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Posterior complexity weight</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
<td>0.6</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rational cate

separable

4fi

than

concepts
data
tbu

separablex
go

rules
t2ux

the

tau

Posterior

of

Rules

structur

accu

1

1222

2112

2222

1211

1112

A6

A5

A4

A3

A2

B5

B4

B3

B2

T7

T6

T3

T2

T1

T5

T4

T6

0.1

0.2

0.3

0.4

0.5

0.6

0.7

0.8

0.9

1.0

Rule-induction model

Human data

r = 0.99

(one free param.)

Fep

non-Fep

transfer

Goodman, et al. (2007, 2008a, 2008b)
Categorization

- Graded judgments

The human linearly classification trained model focuses on the early learn of arable concepts on the data. The Rule-induction model (e.g., Goodman, et al. (2007, 2008a, 2008b)) can be used to predict the posterior feature weight for concept matching and categorization.

For example, the agreement between human judgments and the model predictions is high, with an $r = 0.99$ correlation coefficient for one free parameter. The model maintains a good fit with the human data, even with the third of the data not used for training. Rules such as those tested in the experiment with the participants in the NLS (1978) experiment were used to derive the model predictions.

The model parameters can be adjusted to focus on the semantics of individual concepts, such as the semantic weighting of the NLS concepts. This can be achieved by normalizing the semantic weights so that they match the human judgments. This approach allows for the construction of models that can predict the human categorization of new concepts with high accuracy.
Categorization

- Graded judgments
- Typicality

Typicality

$r = 0.99$

(One free param.)

Goodman, et al. (2007, 2008a, 2008b)
Categorization

- Graded judgments
- Typicality
- Prototype enhancement

Typicality

Graded judgments

Rule-induction model

Goodman, et al. (2007, 2008a, 2008b)
Categorization

- Graded judgments
- Typicality
- Prototype enhancement
- Selective attention

Goodman, et al. (2007, 2008a, 2008b)
Evaluating languages

• Induction to the language generated by the DNF grammar explains important phenomena (and fits relevant data).

• But is this the right LoT?

• Test on wider data set?

• Compare to other propositional languages?
Broader test

- 7 Boolean features.
- 43 randomly generated concepts (3-6 pos. + 2 neg. exs)
- 128 judgements (~122 transfer questions)

Figure 13: Human categorization response frequency proportion of “yes” judgments against model posterior generalization probability. Error bars represent standard error of frequency assuming binomial distributions. Frequencies are computed by first binning responses according to model prediction. The mean of response frequencies binned according to model predictions computed for each run separately. Error bars represent standard error of the mean over runs. Bars below each data point indicate number of runs contributing to that bin. Scale on right.

The main advantages that Rational Rules offer over the other two models come from its focus on the computational theory level of analysis and the modeling power that we gain at that level, the ability to work with a minimal number of free parameters and still achieve strong quantitative data fits, the ability to separate out the effects of representational commitments and inductive logic from the search and memory processes that implement inductive computations, and the ability to seamlessly extend the model to work with different kinds of predicate-based representations, such as those appropriate for learning concepts in continuous spaces, concepts defined by causal implications (see Nx (x Goodman et al, In Press)), or concepts defined by relational predicates (see below).

Evaluating languages

• High-throughput MTurk experiment.
• 108 concepts,
  • Boolean (circle or red)
  • Context-dependent (“Determiners”)
    (unique largest, exists another with same shape)
• 2 orders per concept,
• 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)
Evaluating languages

- High-throughput MTurk experiment.
- 108 concepts,
  - Boolean (*circle or red*)
  - Context-dependent (“Determiners”) *(unique largest, exists another with same shape)*
- 2 orders per concept,
- 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)
Evaluating languages

- High-throughput MTurk experiment.
- 108 concepts,
  - Boolean (circle or red)
  - Context-dependent ("Determiners")
    (unique largest, exists another with same shape)
- 2 orders per concept,
- 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)
Evaluating languages

- High-throughput MTurk experiment.
- 108 concepts,
  - Boolean (circle or red)
  - Context-dependent (“Determiners”) (unique largest, exists another with same shape)
- 2 orders per concept,
- 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)
### Evaluating languages

<table>
<thead>
<tr>
<th>T</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
</table>

#### Key Points:
- **High-throughput MTURK experiment.**
- **108 concepts,**
  - **Boolean** (*circle or red*)
  - **Context-dependent** (“Determiners”) *(unique largest, exists another with same shape)*
- **2 orders per concept,**
- **1596 participants.**

Piantadosi, Goodman, Tenenbaum *(in prep)*
Evaluating languages

- High-throughput MTurk experiment.
- 108 concepts,
  - Boolean (*circle or red*)
  - Context-dependent (“Determiners”) 
    (*unique largest, exists another with same shape*)
- 2 orders per concept,
- 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)
Evaluating languages

• High-throughput MTurk experiment.
• 108 concepts,
• Boolean (circle or red)
• Context-dependent ("Determiners")
  (unique largest, exists another with same shape)
• 2 orders per concept,
• 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)
Evaluating languages

- High-throughput MTurk experiment.
- 108 concepts,
  - Boolean (*circle* or *red*)
  - Context-dependent ("Determiners")
    (unique largest, exists another with same shape)
- 2 orders per concept,
- 1596 participants.

Piantadosi, Goodman, Tenenbaum (in prep)
Boolean concepts

- Circle or blue
- Not [circle or blue]
- Size 2
- [Circle or triangle] implies blue
Boolean concepts

Circle or blue

- Full Boolean grammar
- Human

Not [circle or blue]

- Full Boolean grammar
- Human

Size 2

- Full Boolean grammar
- Human

[Circle or triangle] implies blue

- Full Boolean grammar
- Human
Boolean concepts

Best model performance on Boolean concepts:
Comparing languages

- **DNF**
  disjunctions of conjunctions

- **Horn clauses**
  conjunctions of implications

- **Full boolean**
  any combinations of AND, OR, NOT, IF, IFF

- **Nand**
  combinations of NAND

\[
(\lambda (x) \ (\text{or} \ (\text{and} \ (\text{red} \ x) \ (\text{circle} \ x)) \\
(\text{and} \ (\text{red} \ x) \ (\text{triangle} \ x))))
\]

\[
(\lambda (x) \ (\text{and} \ (\text{implies} \ (\text{not} \ (\text{red} \ x)) \ \text{false}) \\
(\text{implies} \ (\text{not} \ (\text{triangle} \ x)) \ (\text{circle} \ x))))
\]

\[
(\lambda (x) \ (\text{and} \ (\text{red} \ x) \ (\text{or} \ (\text{circle} \ x) \\
(\text{triangle} \ x))))
\]

\[
(\lambda (x) \ (\text{nand} \ \text{false} \ (\text{nand} \ (\text{red} \ x) \\
(\text{nand} \ (\text{nand} \ \text{false} \ (\text{circle} \ x)) \ (\text{nand} \ \text{false} \ (\text{triangle} \ x)))))
\]
Comparing languages

- Fit hyper-parameters (dirichlet on each NT) for each language.
- Evaluated against held out data.

<table>
<thead>
<tr>
<th>Grammar</th>
<th>H.O. LL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL BOOLEAN</td>
<td>-16315.27</td>
</tr>
<tr>
<td>CNF</td>
<td>-16333.59</td>
</tr>
<tr>
<td>DNF</td>
<td>-16368.31</td>
</tr>
<tr>
<td>BICONDITIONAL</td>
<td>-16385.01</td>
</tr>
<tr>
<td>IMPLIES</td>
<td>-16442.40</td>
</tr>
<tr>
<td>HORN CLAUSE</td>
<td>-16487.25</td>
</tr>
<tr>
<td>SIMPLE BOOLEAN</td>
<td>-16490.51</td>
</tr>
<tr>
<td>NAND</td>
<td>-16902.68</td>
</tr>
<tr>
<td>NOR</td>
<td>-16917.49</td>
</tr>
<tr>
<td>UNIFORM</td>
<td>-19482.72</td>
</tr>
<tr>
<td>EXEMPLAR</td>
<td>-23645.13</td>
</tr>
<tr>
<td>ONLY FEATURES</td>
<td>-31662.08</td>
</tr>
<tr>
<td>RESPONSE-BIASED</td>
<td>-37906.77</td>
</tr>
</tbody>
</table>
Non-Boolean concepts

- Best language when context-dependent concepts included is full boolean **plus quantifiers**.
Generative kinds
Learning generative kinds

<table>
<thead>
<tr>
<th>Concept Type</th>
<th>Example Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>{lambda () (\text{node 'a} \quad \text{node 'b (node 'c (node 'd) {node 'e})})}</td>
</tr>
<tr>
<td>Subconcepts</td>
<td>{begin \text{define (part) (node 'c (node 'd (node 'a) {node 'e}))} \text{lambda () (node 'a (node 'b) (node 'd (part) {node 'b (part) (node 'd)}))}}}</td>
</tr>
<tr>
<td>Single recursion</td>
<td>{begin \text{define (part) (node 'a (node 'b (if (flip) (node 'c (part))))}) \text{lambda () (node 'c (node 'd) (part) {node 'e (node 'f (part)}))}}}</td>
</tr>
</tbody>
</table>
Learning generative kinds

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Nested Prototype</th>
<th>Parts</th>
<th>Parameterized Parts</th>
<th>Single Recursion</th>
<th>Multiple Recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype</td>
<td>0.589</td>
<td>0.751</td>
<td>0.803</td>
<td>0.748</td>
<td></td>
</tr>
<tr>
<td>Nested Prototype</td>
<td>0.544</td>
<td>0.851</td>
<td>0.937</td>
<td>0.904</td>
<td></td>
</tr>
<tr>
<td>Parts*</td>
<td>0.320</td>
<td>0.617</td>
<td>0.705</td>
<td>0.835</td>
<td></td>
</tr>
<tr>
<td>Parameterized Parts</td>
<td>0.298</td>
<td>0.591</td>
<td>0.778</td>
<td>0.911</td>
<td></td>
</tr>
<tr>
<td>Single Recursion</td>
<td>0.284</td>
<td>0.499</td>
<td>0.637</td>
<td>0.773</td>
<td></td>
</tr>
<tr>
<td>Multiple Recursion</td>
<td>0.505</td>
<td>0.561</td>
<td>0.451</td>
<td>0.770</td>
<td></td>
</tr>
</tbody>
</table>
Algorithms for induction

• What algorithms are capable of learning concepts in a language of thought?

• All results so far were computed using MCMC based on constituent regeneration (Goodman, et al, 2008).

• Is this cognitively plausible? Maybe...

• But this is probably not enough on its own.

(Ullman, Goodman, Tenenbaum, 2010)
Program induction

- Program induction is especially hard. How could it be done?


```
(/ (+ 1 (* 2 3))
 (+ 1 (* 5 3)))

“inverse inlining”

(def (F x)
 (+ 1 (* x 3)))
(/ (F 2) (F 5))

“de-argument”

(def (F)
 (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))

“de-argument”

(def (F)
 (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```
Program induction

• Program induction is especially hard. How could it be done?

• Idea: anti-unification + argument compression (+ search/MCMC).

```
(define (F x) (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

"inverse inlining"

```
(define (F)
  (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))
```

"de-argument"

```
(define (F x)
  (+ 1 (* x 3)))
(/ (F) (F 1))
```

"de-argument"

```
(define (F)
  (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```
Program induction

- Program induction is especially hard. How could it be done?


```
(define (F x) (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

"inverse inlining"

```
(define (F)
  (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))
```

"de-argument"

```
(define (F x)
  (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

"de-argument"

```
(define (F)
  (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```
Program induction is especially hard. How could it be done?

Idea: anti-unification + argument compression (+ search/MCMC).

```
(define (F x) (+ 1 (* x 3)))
(/ (F 2) (F 5))
```

“inverse inlining”
```
(define (F)
  (+ 1 (* (gaussian 3.5 1.0) 3)))
(/ (F) (F))
```

“de-argument”
```
(define (F)
  (if (flip) (+ 1 (* (F) 3)) 1))
(F)
```

“de-argument”
Program induction

- Program induction is especially hard. How could it be done?


\[
\begin{align*}
\text{“inverse inlining”} & \quad \rightarrow \\
(\text{define (F x)} (\text{+ 1 (* x 3))}) & \quad \rightarrow \\
(\text{define (F x)} (\text{+ 1 (* x 3))}) & \quad \rightarrow \\
(\text{define (F x)} (\text{+ 1 (* x 3))}) & \quad \rightarrow
\end{align*}
\]
Program induction

- **Bayesian program merging algorithm:**
  (Cf. Stolcke & Omohundro, 1994)
  - Initial state is exemplar program (mixture of data).
  - Transform programs via anti-unification and argument compression.
  - **Beam search**\* with respect to the posterior score.
    (*Or stochastic search, Monte Carlo, etc.)
  - Likelihood: marginal probability of data given program
    (computed by lazy particle filter or dynamic progr.).
  - Prior: syntactic complexity.

Hwang, Stuhlmueller, Goodman (2011)
Example

```
(node r (node b)
  (node r (node b)
    (node r (node b)
      (node r (node b) b)))

(define (F x) (node r (node b) x))
(define (F) (if (flip 0.8)
               (node r (node b) (F))
               b))
```

“data incorporation”

“inverse inline”

“de-argument”

sample
(define F248 (lambda () (F247 (F246)))))

(define F247 (lambda (V994)
  (elem "rect"
    (tr "left" (tr "reflect" V994))
    (tr "right" V994))))

(define F246 (lambda ()
  (elem "tri"
    (tr "forward"
      (choose (elem "tri") (F246)))))

(F248)
Example

(begin (define F4 (lambda (V7 V8) (node (F1 V7 0.1) V8)))
  (define F3 (lambda (V6) (node (F1 V6 0.3)))))
(define F2
  (lambda (V3 V4)
    ((lambda (V5) (F4 V3 (F4 V4 V5)))
      (if (flip 9/11)
        (uniform-choice
          (node (F1 204.0 0.3) (F3 199.0)
            (F3 243.0) (F3 233.0) (F3 240.0)))
          (F4 151.0
            (node (F1 -21.0 0.3) (F3 7.0)
              (F3 49.0) (F3 3e1) (F3 44.0))))))
(define F1
  (lambda (V1 V2)
    (data (color (gaussian V1 25)) (size V2)))
(lambda ()
  (uniform-choice
    (node (F1 13.0 1) (F2 89.0 111.0)
      (F2 85.0 121.0))))

We presented an approach to creating generative models from data called Bayesian program merging. Bayesian model merging produces a program with a similar structure to the original generating program. We generated colors for our trees using Gaussians and one motivation for this would be to model the randomness in data constructors (such as random processes in the environment). There is another, more subtle, reason to introduce a noise process. Transformations made while searching were guided using the posterior of the program. The main ideas were to rerepresent data as a program and then frame finding regularities in the transformed data as identifying repeated computation in the program. Finding repeated computation was performed through program transformations that merged the structure of a program. The sequences of transformed data as identifying repeated computation in the program.

8.1 Noise and Representation

The petal function is a function that creates a branch that ends in a flower with either blue petals or red petals.
Application

- **Inverse procedural modeling:**
  - Procedural model induced from a small set of artist-created examples.
  - Jerry Talton (Talton, et al, in prep.)

![Terminals](image1)

![Examples](image2)

Posterior samples
Application

Figure 7: Playgrounds and naval ship models. (top left) The set of components used to build the exemplar models. (top right) The set of exemplar models. (bottom) Random derivations from the stochastic, context-free grammar learned by our method.
Application

Figure 8: Seussian architecture and sakura trees. (top left) The set of components used to build the exemplar models. (top right) The set of exemplar models. (bottom) Random derivations from the stochastic, context-free grammar learned by our method.
Conclusion

Probabilistic inference

Generative models

Compositional representations

Probabilistic language of thought