Mental models as probabilistic programs

Do it yourself!

Marta Kryven
mkryven@mit.edu

Thanks to Tobi Gerstenberg for the slides

August 17, 2020
Probabilistic problem example

Tug of war is a sport that pits two teams against each other in a test of strength
How strong is Tom (on the scale 1 to 10)?
How strong is Bill?
How strong is Tom?
Concepts:

person  teams  winner

strength  pulling

effort
A computational model to infer strength from observation

```javascript
var towModel = function() {
    var strength = mem(function (person) {return gaussian(50, 10)})

    var lazy = function(person) {return flip(0.1) }

    var pulling = function(person) {
        return lazy(person) ? strength(person) / 2 : strength(person) }

    var totalPulling = function (team) {return sum(map(pulling, team))}

    var winner = function (team1, team2) {
        totalPulling(team1) > totalPulling(team2) ? team1 : team2 }

    var beat = function(team1,team2){winner(team1,team2) == team1}

    condition(beat([ 'bob' ], [ 'tom' ]))

    return strength('bob')
}
```


Outline

• How can a program model human thought? What kinds of programs?

• Probabilistic language of thought hypothesis
  - Compositionality
  - Productivity
  - Inference

• Do it yourself!
  1. WebPPL basics
  2. Building generative models
  3. Doing inference
How can a program model human thought?

Formally describe intelligent inferences in human reasoning

Can be applied to engineer intelligent machines

**Computational theory of mind**

mind = computer

mental representations = computer programs

formal models of thinking = programs
What kind of a program?

Structure

Knowledge

Probability

Uncertainty

Tenenbaum, Kemp, Griffiths, & Goodman (2011) How to grow a mind: Statistics, structure, and abstraction. Science
Example of structure - knowledge of physics; Probability - stochastic simulations

Battaglia, Hamrick, & Tenenbaum (2013) Simulation as an engine of physical scene understanding. Proceedings of the National Academy of Sciences
Example of structure - motor programs in drawing; Probability - minimizing motor cost

- Task: Imitate dance
- Prior knowledge:
  - EXPERT: Mental “vocabulary” of dance moves
  - NOVICE: None
- Solution:
  - Motifs: 1 → 2 → 1
  - Actions: A → B → C → B → B → A → B → C

Learning abstract structure for drawing by efficient motor program induction
Lucas Y. Tian, Kevin Ellis, Marta Kryven, Joshua B. Tenenbaum
Example of structure - maps / representations of the environment; Probability - maximizing expected utility of actions


Probabilistic language of thought hypothesis

Informal version:

“Concepts have a language-like compositionality and encode probabilistic knowledge. These features allow them to be extended productively to new situations and support flexible reasoning and learning by probabilistic inference.”

Fodor (1975). The language of thought
**Probabilistic language of thought hypothesis**

**compositionality** creative generation of complex actions by recombining relevant components of prior knowledge, in order to solve problems.

[Image of stick figures]

---


https://papers.nips.cc/paper/7845-learning-to-infer-graphics-programs-from-hand-drawn-images
Probabilistic language of thought hypothesis

Productivity  “infinite use of finite means” (Wilhelm von Humboldt)
Probabilistic language of thought hypothesis

**Language** limited number of letters, fixed set of grammatical rules

can express an infinite number of thoughts

### Physics Formula Sheet

#### Mechanics

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[x = x_0 + v_0 t + \frac{1}{2} a t^2]</td>
<td>Position formula</td>
</tr>
<tr>
<td>[v = v_0 + at]</td>
<td>Velocity formula</td>
</tr>
<tr>
<td>[v^2 - v_0^2 = 2a(x - x_0)]</td>
<td>Velocity change formula</td>
</tr>
<tr>
<td>[\ddot{a} = \frac{\Sigma F}{m} = \frac{F_{\text{net}}}{m}]</td>
<td>Acceleration formula</td>
</tr>
<tr>
<td>[</td>
<td>\vec{F}_{\text{friction}}</td>
</tr>
<tr>
<td>[\ddot{p} = m\ddot{v}]</td>
<td>Linear momentum equation</td>
</tr>
<tr>
<td>[\Delta p = \vec{F} \Delta t]</td>
<td>Change in momentum</td>
</tr>
<tr>
<td>[\ddot{a} = \frac{\Sigma F}{I} = \frac{\vec{F}_{\text{net}}}{I}]</td>
<td>Angular acceleration formula</td>
</tr>
</tbody>
</table>

#### Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>acceleration</td>
</tr>
<tr>
<td>A</td>
<td>amplitude</td>
</tr>
<tr>
<td>A</td>
<td>Area</td>
</tr>
<tr>
<td>b</td>
<td>base length</td>
</tr>
<tr>
<td>C</td>
<td>circumference</td>
</tr>
<tr>
<td>d</td>
<td>distance</td>
</tr>
<tr>
<td>E</td>
<td>energy</td>
</tr>
<tr>
<td>f</td>
<td>frequency</td>
</tr>
<tr>
<td>F</td>
<td>force</td>
</tr>
<tr>
<td>h</td>
<td>height</td>
</tr>
<tr>
<td>I</td>
<td>current</td>
</tr>
<tr>
<td>I</td>
<td>rotational inertia</td>
</tr>
<tr>
<td>KE</td>
<td>kinetic energy</td>
</tr>
<tr>
<td>k</td>
<td>spring constant</td>
</tr>
<tr>
<td>L</td>
<td>angular momentum</td>
</tr>
<tr>
<td>\ell</td>
<td>length</td>
</tr>
<tr>
<td>m</td>
<td>mass</td>
</tr>
<tr>
<td>P</td>
<td>power</td>
</tr>
<tr>
<td>p</td>
<td>momentum</td>
</tr>
<tr>
<td>q</td>
<td>charge</td>
</tr>
<tr>
<td>r</td>
<td>radius</td>
</tr>
<tr>
<td>R</td>
<td>resistance</td>
</tr>
<tr>
<td>S</td>
<td>surface area</td>
</tr>
<tr>
<td>T</td>
<td>period</td>
</tr>
<tr>
<td>t</td>
<td>time</td>
</tr>
<tr>
<td>PE</td>
<td>potential energy</td>
</tr>
<tr>
<td>V</td>
<td>electric potential</td>
</tr>
</tbody>
</table>

#### Electricity

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[</td>
<td>\vec{E}</td>
</tr>
<tr>
<td>[\Delta V = 1R]</td>
<td>Voltage drop</td>
</tr>
<tr>
<td>[R = \frac{\rho \ell}{A}]</td>
<td>Resistance formula</td>
</tr>
<tr>
<td>[I = \frac{\Delta q}{\Delta t}]</td>
<td>Current formula</td>
</tr>
<tr>
<td>[P = I \Delta V]</td>
<td>Power formula</td>
</tr>
<tr>
<td>[R_{\text{series}} = R_1 + R_2 + ... + R_n]</td>
<td>Series resistance</td>
</tr>
<tr>
<td>[\frac{1}{R_{\text{Parallel}}} = \frac{1}{R_1} + \frac{1}{R_2} + ... + \frac{1}{R_n}]</td>
<td>Parallel resistance</td>
</tr>
</tbody>
</table>

\[V_{\text{terminal}} = \frac{Q}{C}\] | Capacitance formula |
Probabilistic programs

DIY

DO IT YOURSELF
Why WebPPL?

<table>
<thead>
<tr>
<th>WebPPL</th>
<th>other PPLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>• written by cognitive scientists for cognitive scientists</td>
<td>• Stan, Anglican, Alchemy, BUGS, Edward, PyMC</td>
</tr>
<tr>
<td>• goal: building computational models of cognition</td>
<td>• goal: building rich models for data analysis</td>
</tr>
<tr>
<td>• extremely flexible</td>
<td>• flexible but limited</td>
</tr>
<tr>
<td>• can be slow</td>
<td>• faster (for the models that can be expressed)</td>
</tr>
</tbody>
</table>

Recent PPLs (Pyro, Gen) have a steeper learning curve, require more thought about inference, but keep the flexibility of WebPPL and work faster.
1. WebPPL basics
2. Building generative models
3. Doing inference
http://bit.do/webppl

notes/1_webppl_basics.md

http://webppl.org
1. WebPPL basics
WebPPL basics

- declare variables
- data formats: numbers, strings, logicals, objects, arrays
- if-then-statements
- defining functions
- higher order functions
  - map
WebPPL basics

- The `map` function

```javascript
var mySquare = function(x){return x*x}
var someNumbers = [1,2,3,4]
display(map(mySquare, someNumbers))
```

In: 
```
[  1,  2,  3,  4  ]
[   ,   ,   ,   ]
```

Out: 
```
[ 1', 4', 9', 16 ]
```
WebPPL basics

- WebPPL = purely functional programming language
- **can’t** write **for** loops or **while** statements
- **can** create higher-order functions and recursive functions
  - for example: `map` = apply a function to each element of a list (like a **for** loop)
2. Building generative models
Building generative models

- forward sampling and random primitives
- building simple models
- sample from probability distributions
- memoization: \texttt{mem}
- recursive functions
Building generative models

• WebPPL is a language to formally describe how the world works
• random choices capture our uncertainty or ignorance
• the language is universal: it can express any computable process
• causal dependence is important: the program describes what influences what


Building generative models

relationship between sampling and probability distributions

Random primitives:

\[
\begin{align*}
\text{var } a &= \text{flip}(0.3) \\
\text{var } b &= \text{flip}(0.3) \\
\text{var } c &= \text{flip}(0.3) \\
\text{return } a + b + c
\end{align*}
\]

\[
P(n) = \binom{3}{n} 0.3^n 0.7^{3-n}
\]

\[
\begin{array}{c}
\text{probability/} \\
\text{frequency}
\end{array}
\]

Sampling \quad \equiv \quad \text{Distributions}

any computable distribution can be represented as the distribution induced by sampling from a probabilistic program

Uncertainty in Artificial Intelligence
3. Doing inference
Doing inference

Conditional inference:

Infer(
    function()
    {
        var a = flip(0.3)
        var b = flip(0.3)
        var c = flip(0.3)
        condition(a + b == 1)
        return a + b + c
    }
)

"when you have excluded the impossible, whatever remains, however improbable, must be the truth."
Doing inference

- conditioning on variables
  - rejection sampling
  - WebPPL’s inference procedures
- conditioning on arbitrary expressions
- other inference procedures
- forward, rejection, enumerate, MH, ...

- you don’t have to worry about inference. **very nice!**
- WebPPL allows us to parsimoniously describe rich generative model structures and explore the inference patterns that emerge from the interaction of model and data
“Concepts have a language-like compositionality and encode probabilistic knowledge. These features allow them to be extended productively to new situations and support flexible reasoning and learning by probabilistic inference.”

```javascript
var towModel = function() {
    var strength = mem(function (person) { return gaussian(50, 10) })
    var lazy = function(person) { return flip(0.1) }
    var pulling = function(person) {
        return lazy(person) ? strength(person) / 2 : strength(person)
    }
    var totalPulling = function (team) { return sum(map(pulling, team)) }
    var winner = function (team1, team2) {
        totalPulling(team1) > totalPulling(team2) ? team1 : team2
    }
    var beat = function(team1, team2) { winner(team1, team2) == team1 }
    
    condition(beat(['bob'], ['tom']))
    return strength('bob')
}
```
Resources: theory

• Goodman, Mansinghka, Roy, Bonawitz, & Tenenbaum (2008) Church: A language for generative models. Uncertainty in Artificial Intelligence


• Chater & Oaksford (2013) Programs as causal models: Speculations on mental programs and mental representation. Cognitive Science

  ➡ http://www.nature.com/nature/journal/v521/n7553/full/nature14541.html
Resources: practice

- https://probmods.org/
  - many more cool chapters to play around with
- http://webppl.org/
  - Editor to play around with code
- http://dippl.org/
  - Details about WebPPL
- https://github.com/probmods/webppl
  - github repository with latest developments
  - function reference for the webppl language
- http://agentmodels.org/
  - Great web book that focuses on how to model agents and inferences about agents
- http://probabilistic-programming.org/wiki/Home
  - homepage comparing different probabilistic programming languages
Thanks!