Computational Cognitive Modeling

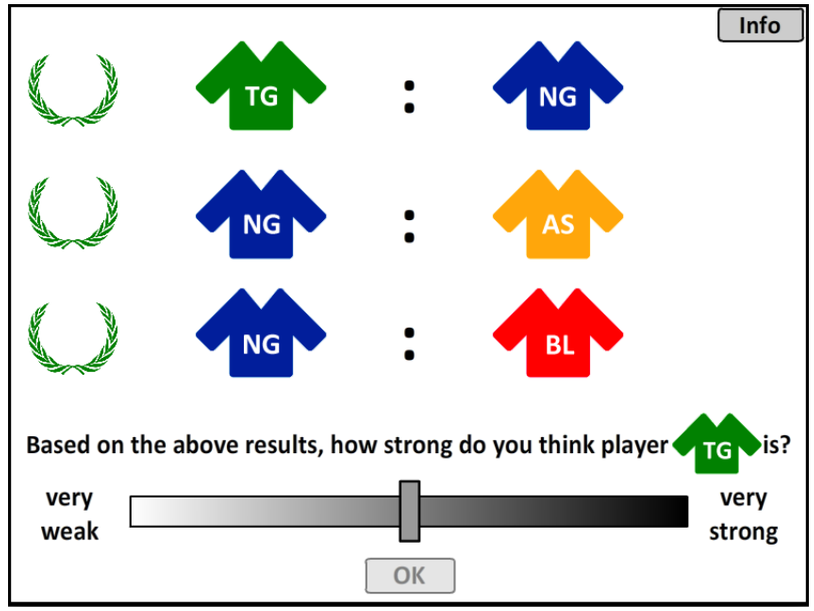
Probabilistic programming, program induction, and language of thought models

Brenden Lake & Todd Gureckis

email address for instructors:
instructors-ccm-spring2019@nyuccl.org

course website:
https://brendenlake.github.io/CCM-site/

Probabilistic programs / probabilistic programming




```
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] = lazy(name)
        return W.dict_strength[name]

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)

Probabilistic programs / probabilistic programming

Stochastic program (known structure)

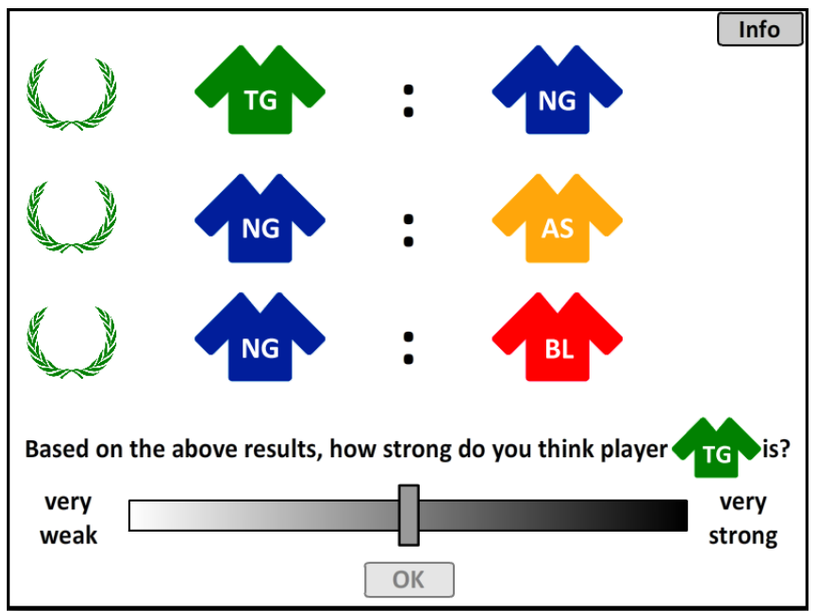
```
class world():
    def __init__(self):
        self.dict_strength = {}
    def clear(self): # used when sampling over possible world
        self.dict_strength = {}


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


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

Data

(e.g., game matches and who won)

inference

How strong is Bob?



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

```
def flip(theta=0.5):
    return random.random() < theta
```

(flip a coin with ‘theta’ chance of heads)

Simple probabilistic program

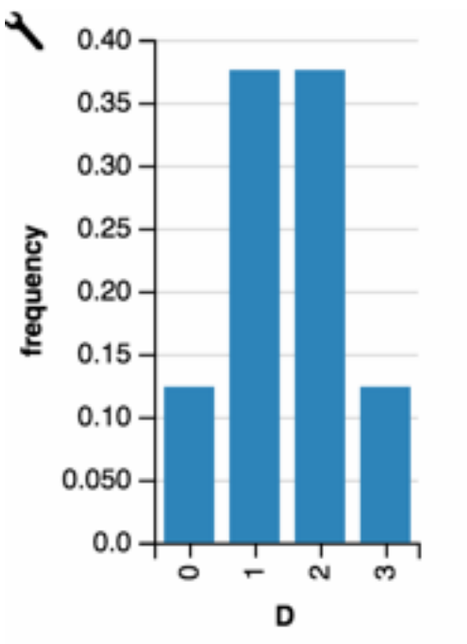
```
A = flip()
B = flip()
C = flip()
D = A + B + C
```

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

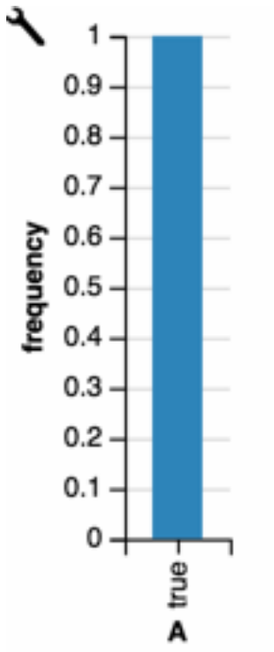
Bayesian inference

(again, notice productivity reasoning)

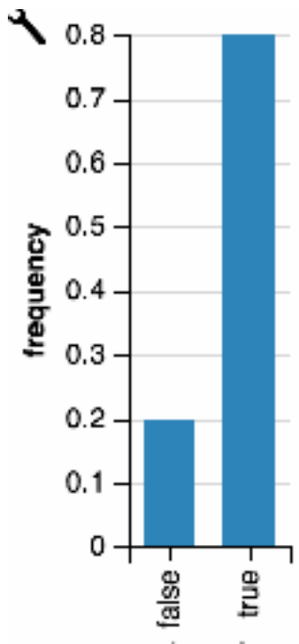
$P(D)$



$P(A|D = 3)$



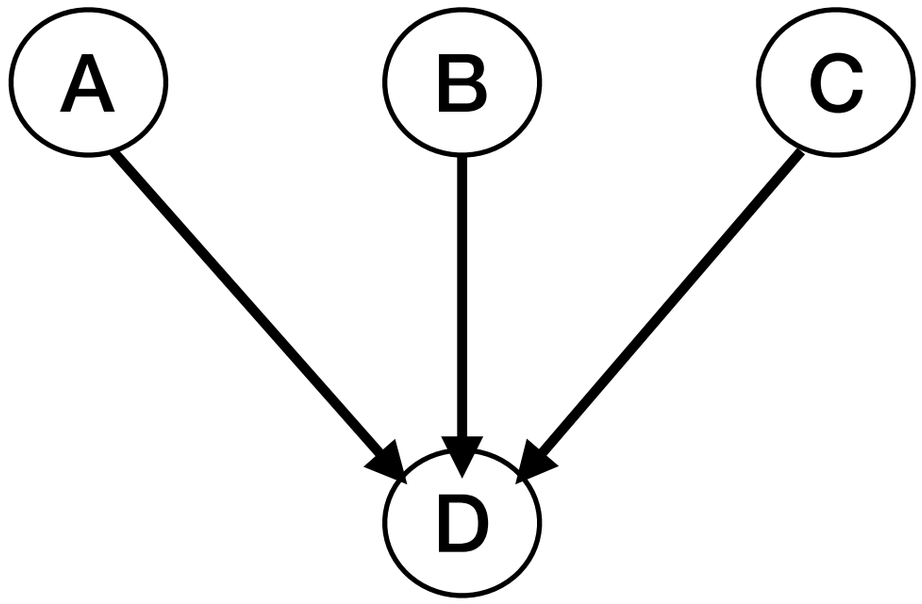
$P(A|D \geq 2)$



Example from Noah Goodman and Josh Tenenbaum https://probmods.org/

Probabilistic program or Bayesian network?

```
A = flip()
B = flip()
C = 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.

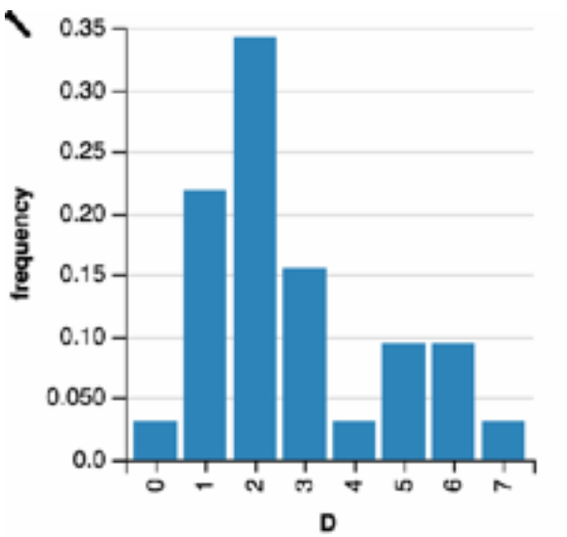
Probabilistic programs: Another example

Simple probabilistic program (yet more complex than before)

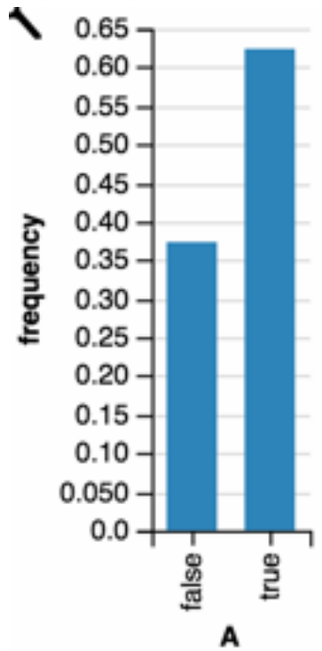
```
A = flip()
B = flip()
C = flip()
if C:
    D = A + B + C
else:
    E = flip()
    F = (2*flip())**2
    D = A + B + C +E + F
```

Bayesian inference

$P(D)$



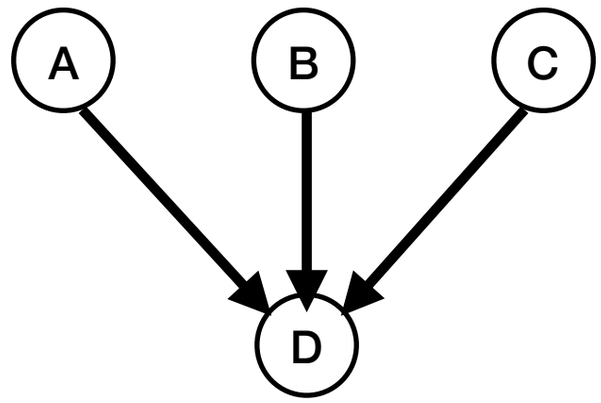
$P(A|D \geq 2)$



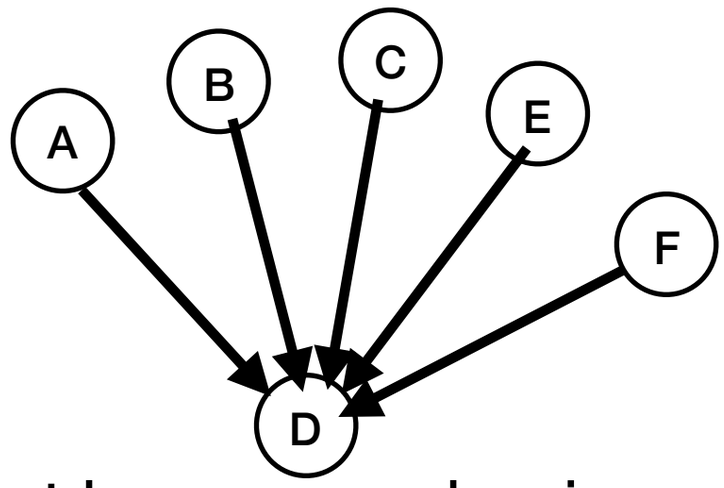
Probabilistic program or ~~Bayesian network~~?

```
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
```

if C is True:

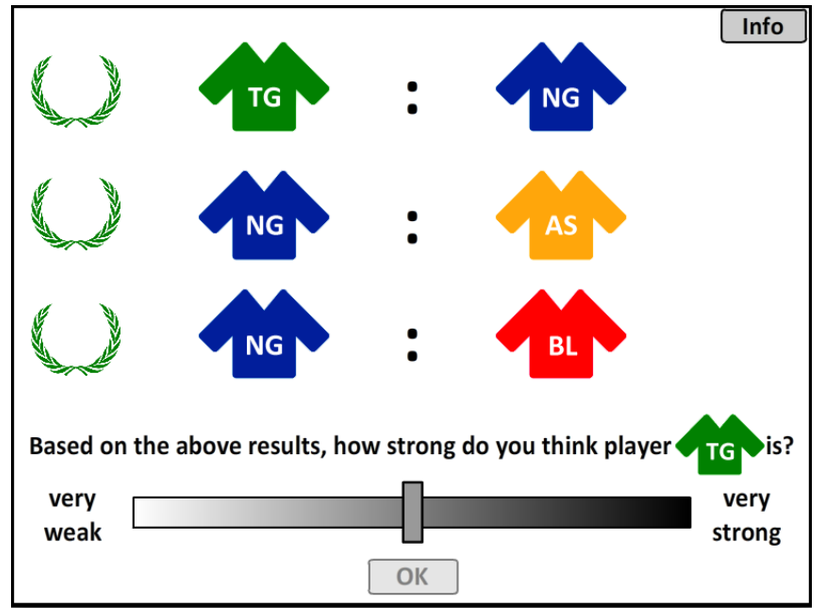


if C is False:



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



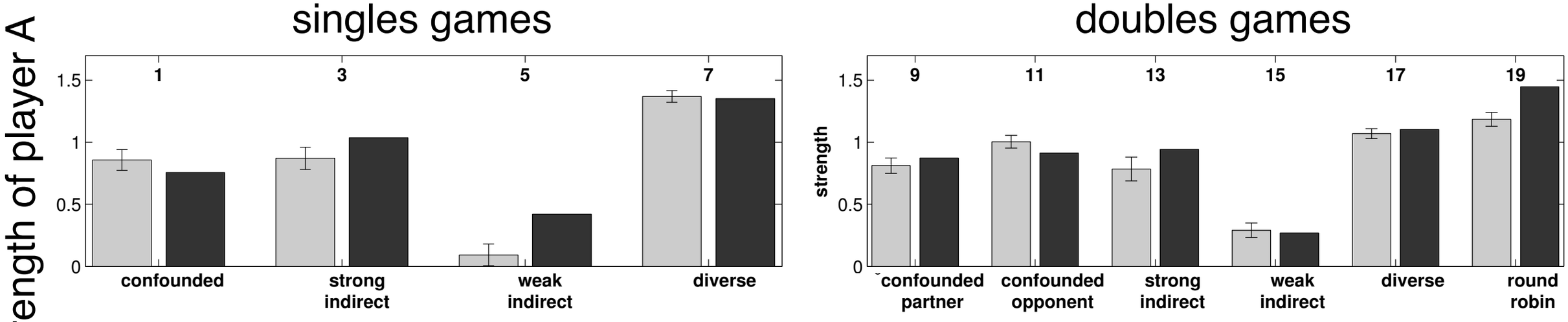

```
class world():
    def __init__(self):
        self.dict_strength = {}
    def clear(self): # used when sampling over possible world
        self.dict_strength = {}


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

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

confounded evidence (1,2)	strong indirect evidence (3,4)	weak indirect evidence (5,6)	diverse evidence (7,8)
A > B	A > B	A > B	A > B
A > B	B > C	B < C	A > C
A > B	B > D	B < D	A > D



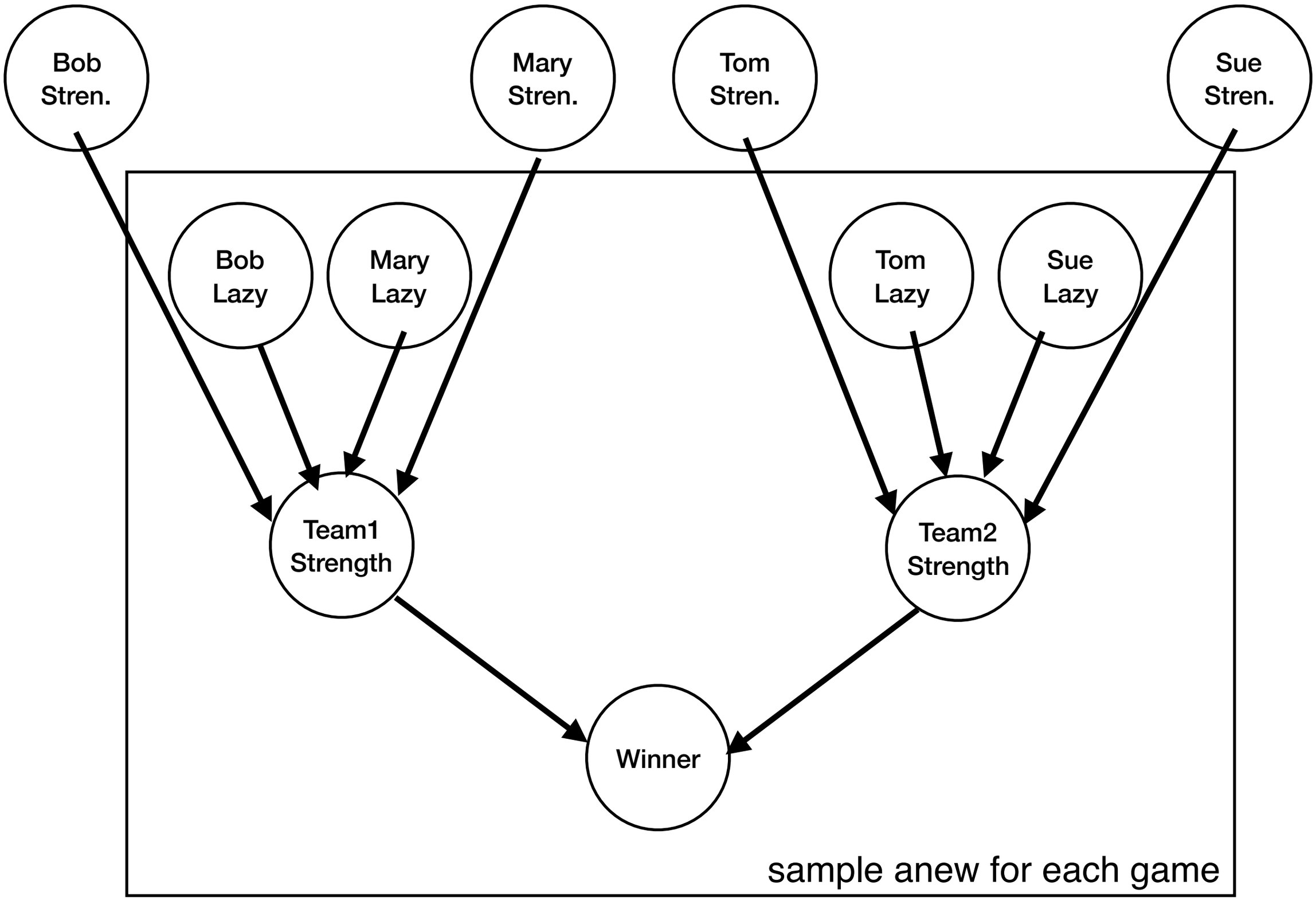
human judgements

model judgements

(see Gerstenberg et al., 2012; Goodman et al., 2015)

Bayesian network formulation depends on set of players

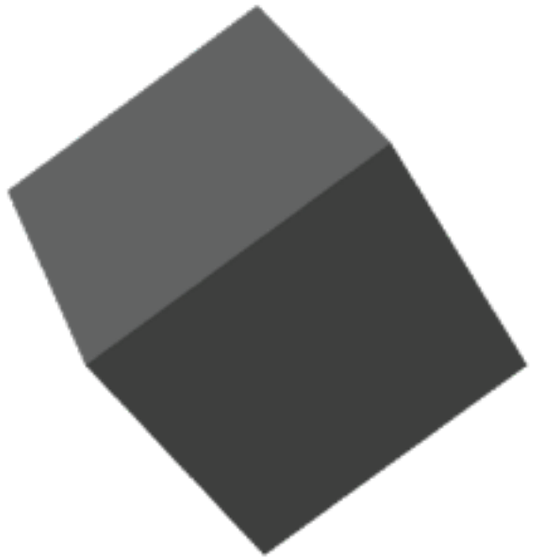
Each specification of the teams changes the Bayes net structure, and thus an additional modeling mechanism is needed to construct the Bayes nets.



Example probabilistic programming languages and software

WebPPL

pyro



Edward

Church

Stan

Program induction

Prior distribution over (unknown) possible programs

Hypothesis 1

$$\lambda\, S\,.\,(if\ (singleton?\ S)\ \text{“one”}\ (next\ (L\ (set\text{-}difference\ S\ (select\ S)))))$$

Hypothesis 2

$$\lambda\, S\,.\,(if\ (singleton?\ S)\ \text{“one”}\ (if\ (doubleton?\ S)\ \text{“two”}\ undef))$$

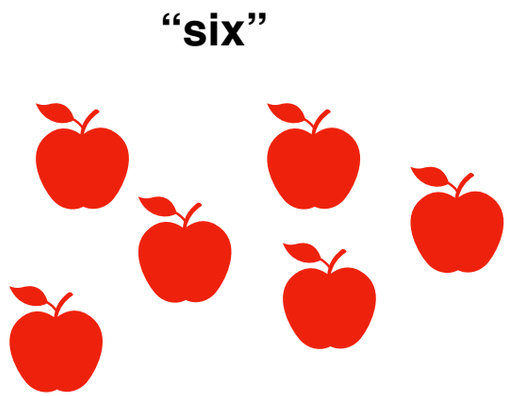
Hypothesis 3

$$\lambda\, S\,.\,(if\ (singleton?\ S)\ \text{“one”}\ undef)$$

Hypothesis N …

program generates data

Data (*D*)



inference

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

piantado / LOTlib

Watch 13 | Unstar 21 | Fork 14

Code | Issues 6 | Pull requests 0 | Projects 0 | Wiki | Insights

Lanugage of Thought (LOT) models in Python.

1,303 commits | 3 branches | 0 releases | 10 contributors | GPL-3.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

piantado added languages; revised search and model — Latest commit 83b9933 on Jul 4, 2017

Documentation	updated input to example data	2 years ago
LOTlib	added languages; revised search and model	10 months ago
.gitignore	update gitignore with double-star syntax	2 years ago
LICENSE	initial commit	5 years ago
MAJOR-CHANGES.txt	extreme simplification of proposals	2 years ago
README.md	updated Readme	a year ago
Test.sh	Revert "Revert "Merge remote-tracking branch 'origin/master'""	2 years ago

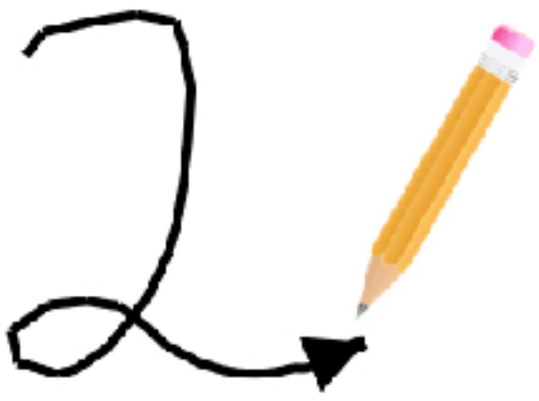
README.md

LOTlib

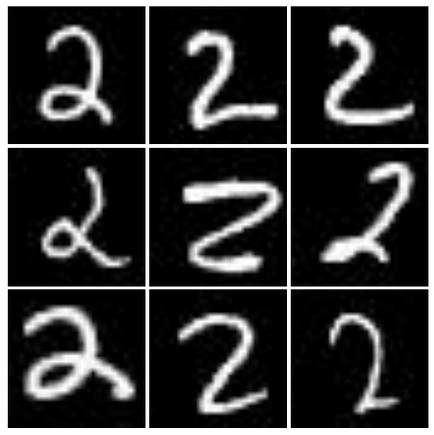
LOTlib is a Python 2 library for implementing "language of thought" models. A LOTlib model specifies a set of primitives and captures learning as inference over compositions of those primitives in order to express complex concepts. LOTlib permits lambda expressions, meaning that learners can come up with abstractions over compositions and define new

Motivation: We need more than Bayesian networks to represent complex, real causal processes for generating data

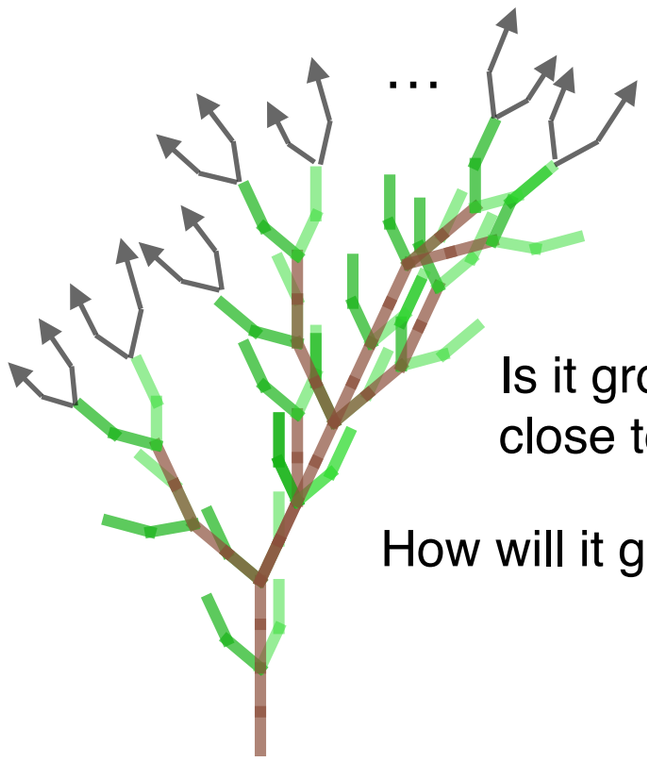
same causal process



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”

Case study: Learning new handwritten letters

RESEARCH ARTICLES

COGNITIVE SCIENCE

Human-level concept learning through probabilistic program induction

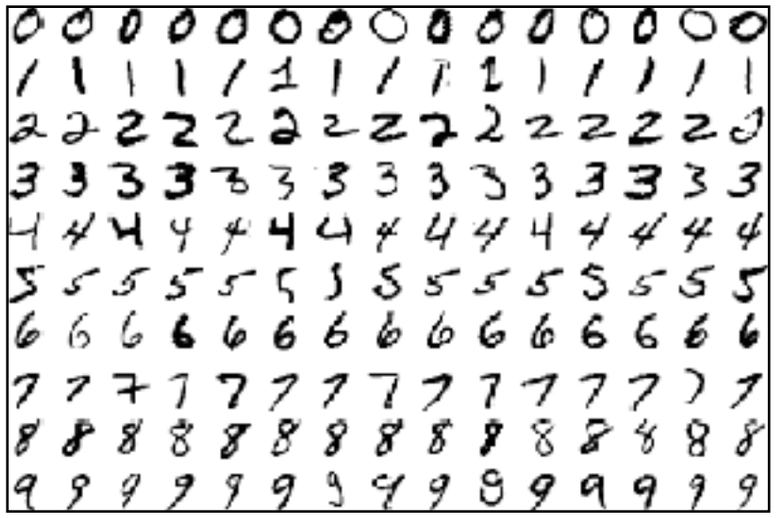
Brenden M. Lake,[1]* Ruslan Salakhutdinov,[2] Joshua B. Tenenbaum[3]

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.

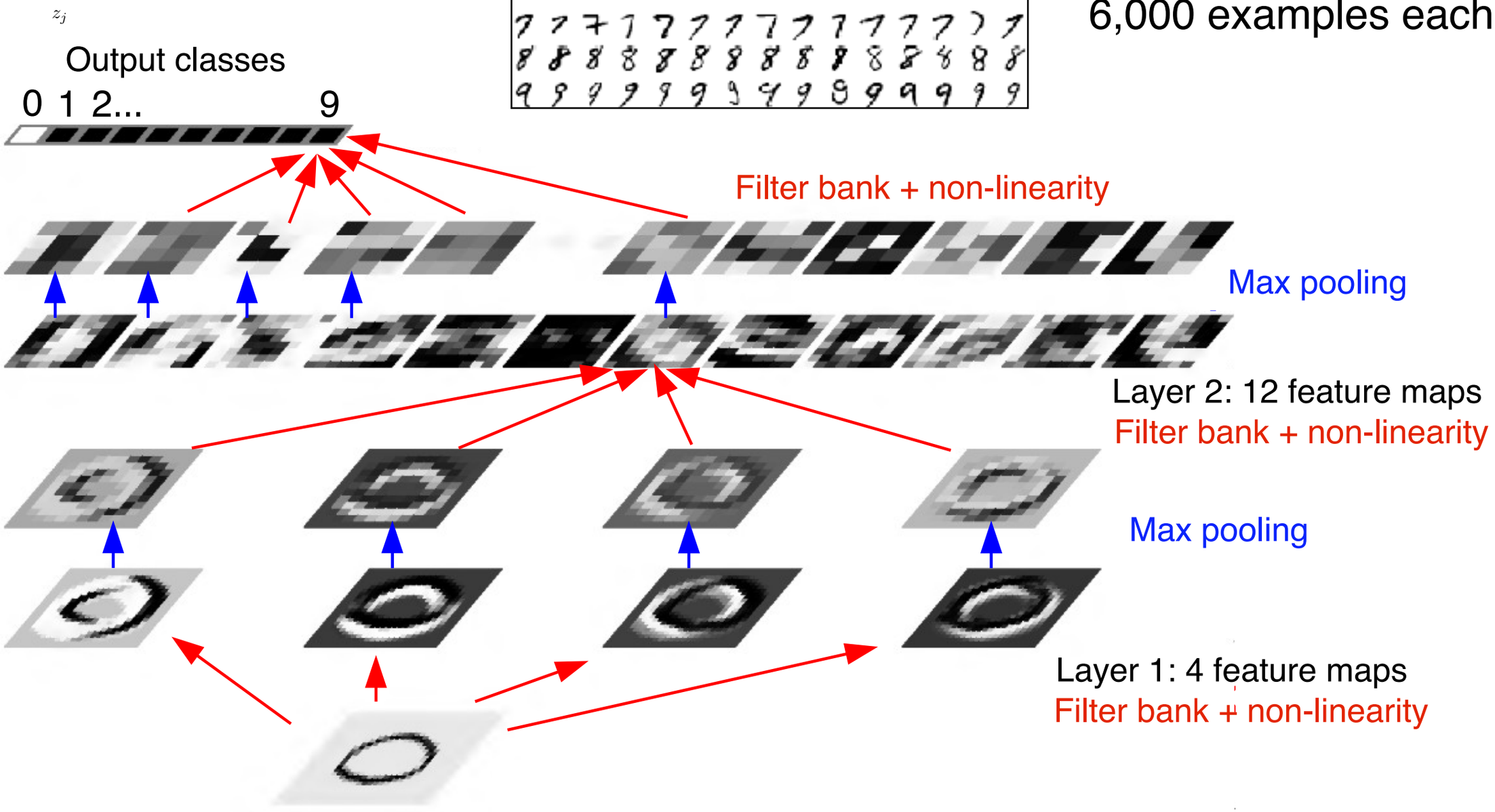
Despite remarkable advances in artificial intelligence and machine learning, two aspects of human conceptual knowledge have eluded machine systems. First, for most interesting kinds of natural and man-made categories, people can learn a new concept from just one or a handful of examples, whereas standard 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

[1]Center for Data Science, New York University, 726 Broadway, New York, NY 10003, USA. [2]Department of Computer Science and Department of Statistics, University of Toronto, 6 King's College Road, Toronto, ON M5S 3G4, Canada. [3]Department 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

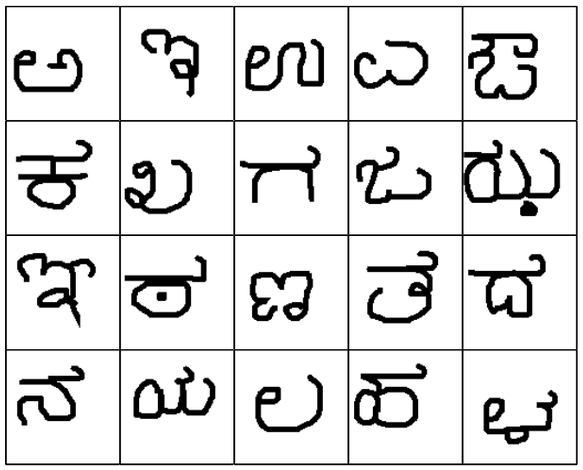


Slide credit: Yann LeCun

Learning a new letter: Comparing humans and machines

less data

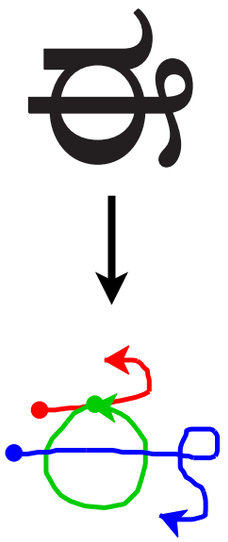
People can learn a new concept from a single image.



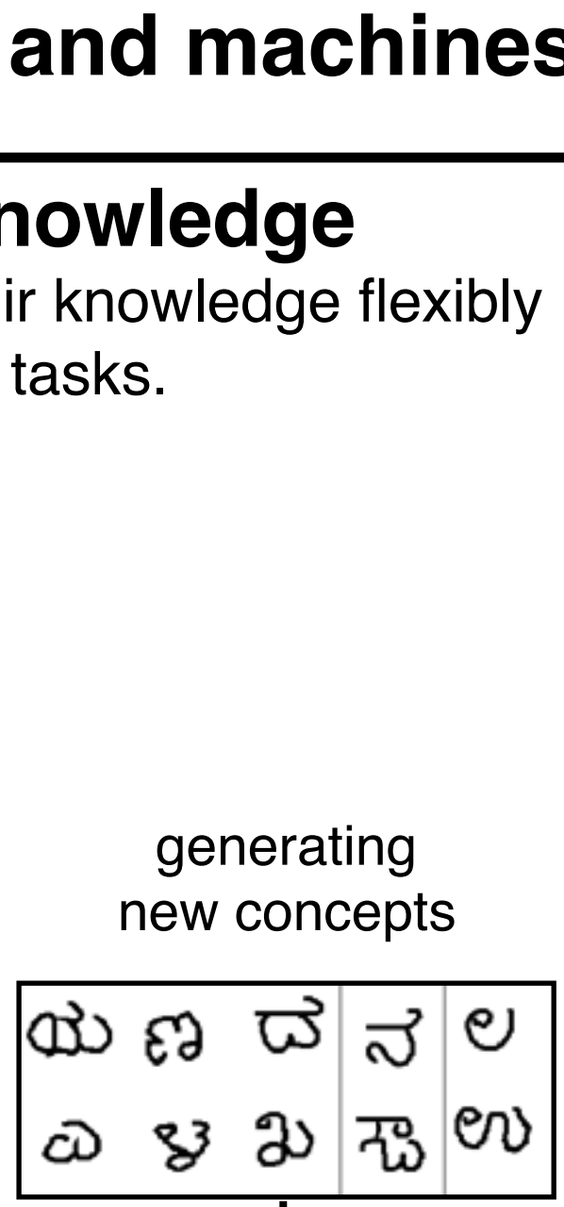
more knowledge

People can apply their knowledge flexibly to new tasks.

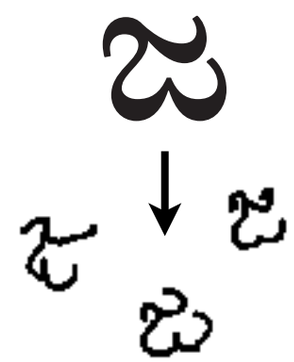
parsing



generating new concepts



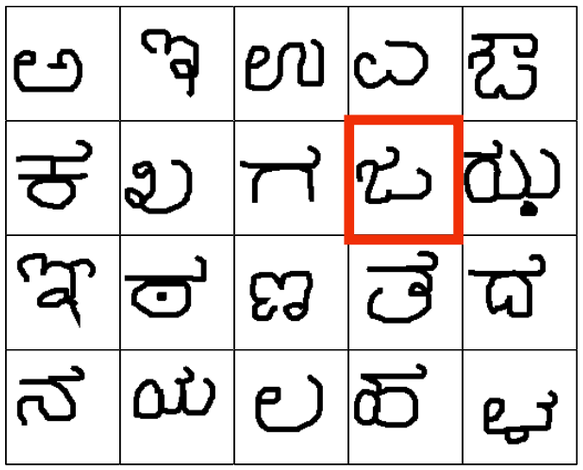
generating new examples



Learning a new letter: Comparing humans and machines

less data

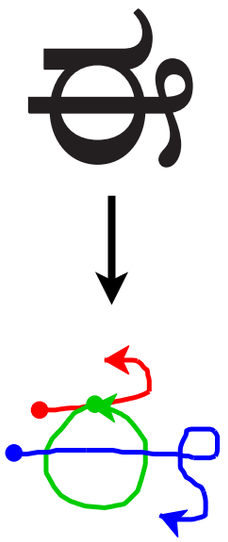
People can learn a new concept from a single image.



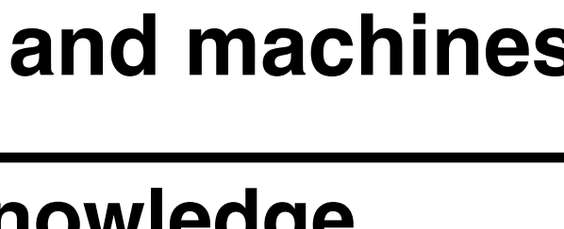
more knowledge

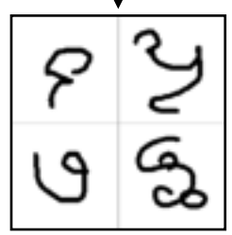
People can apply their knowledge flexibly to new tasks.

parsing

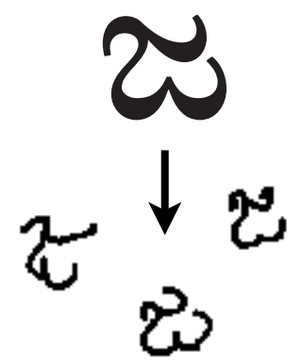


generating new concepts





generating new examples



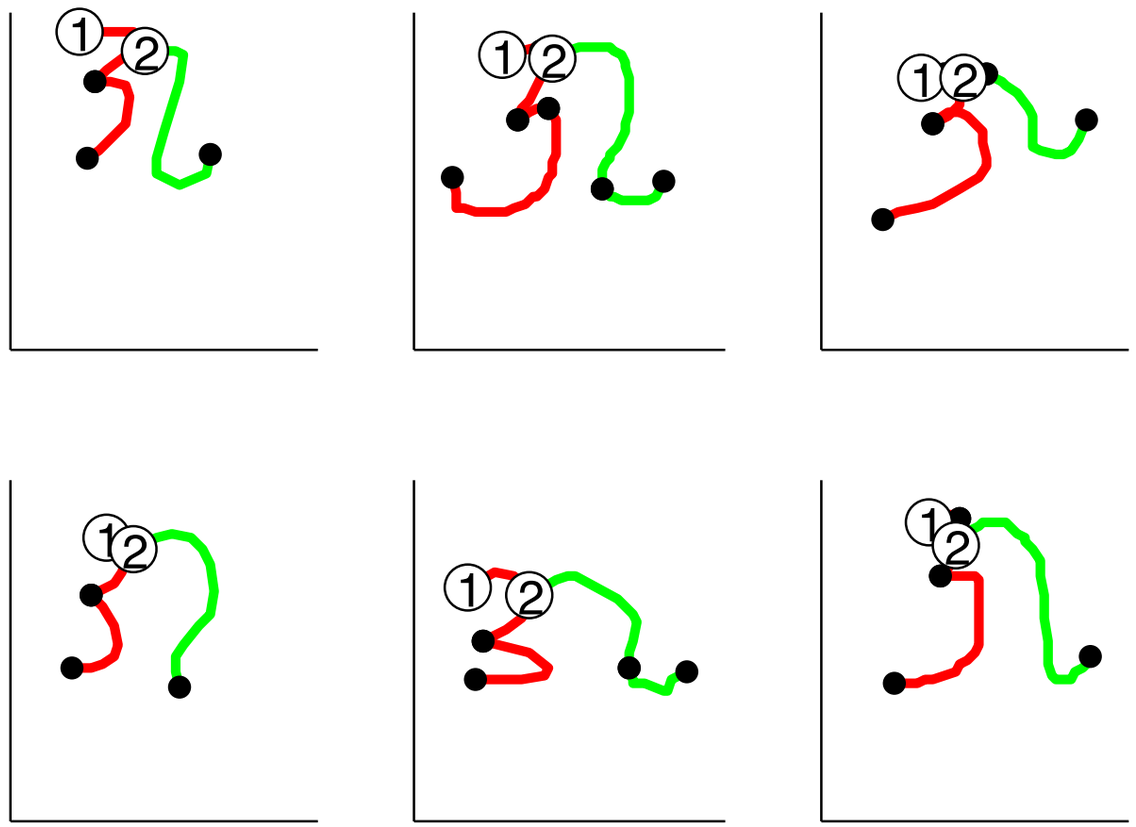
Omniglot stimulus set

(https://github.com/brendenlake/omniglot)

1600+ concepts
20 examples each

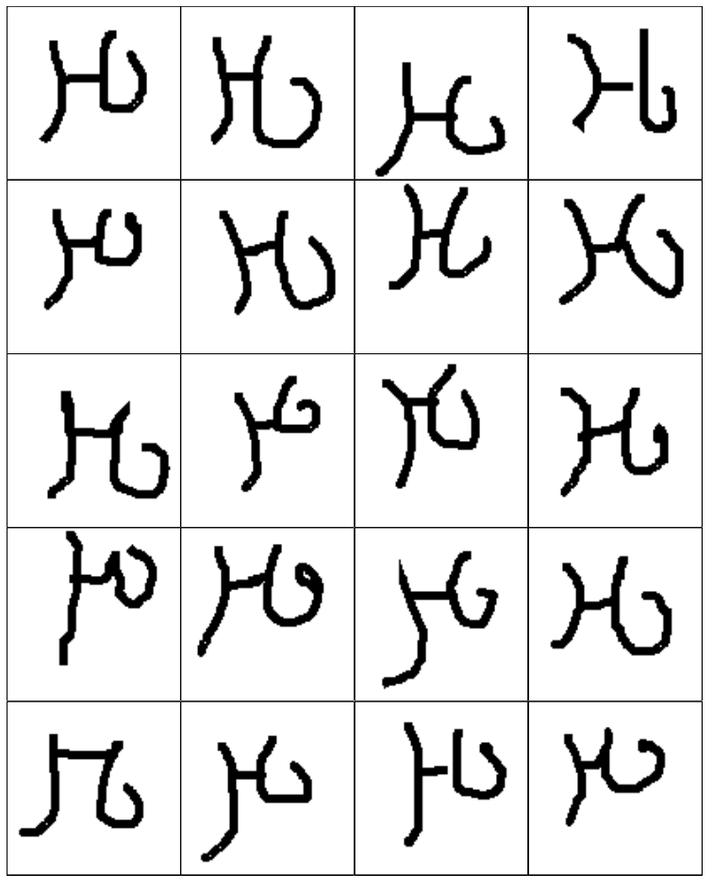
અ

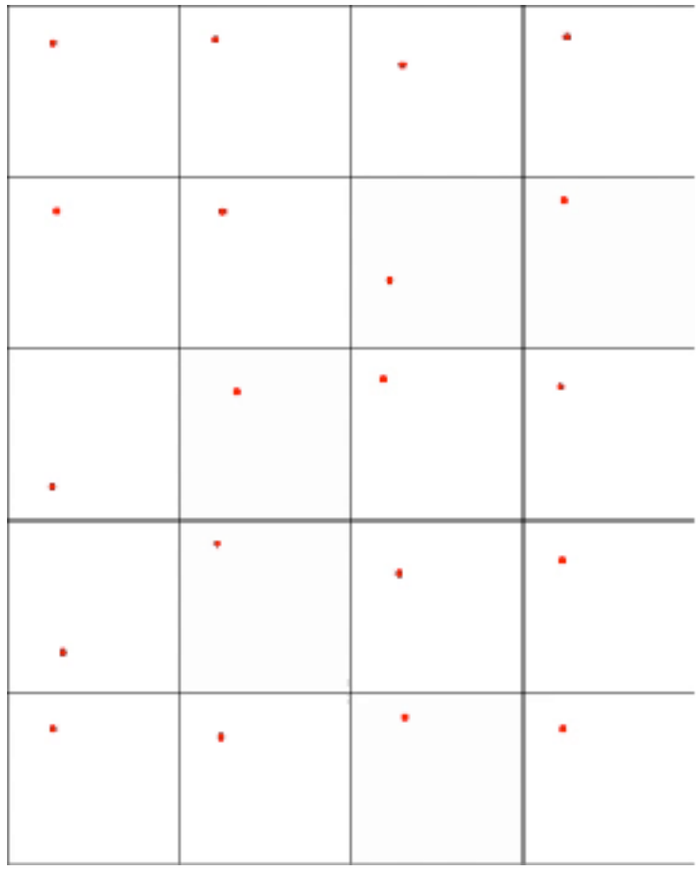
Human drawings



Original

Human drawings

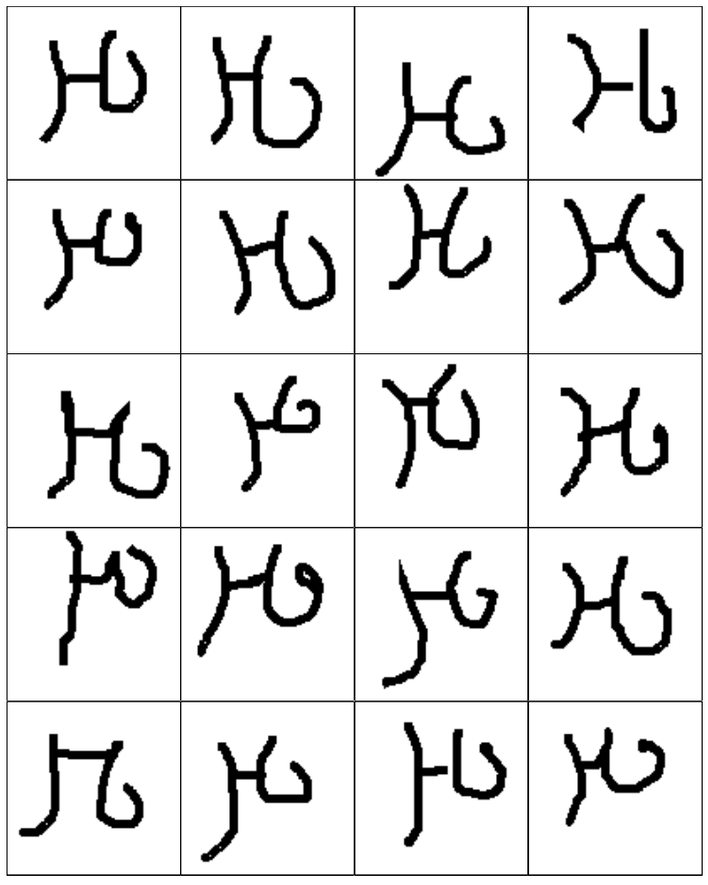


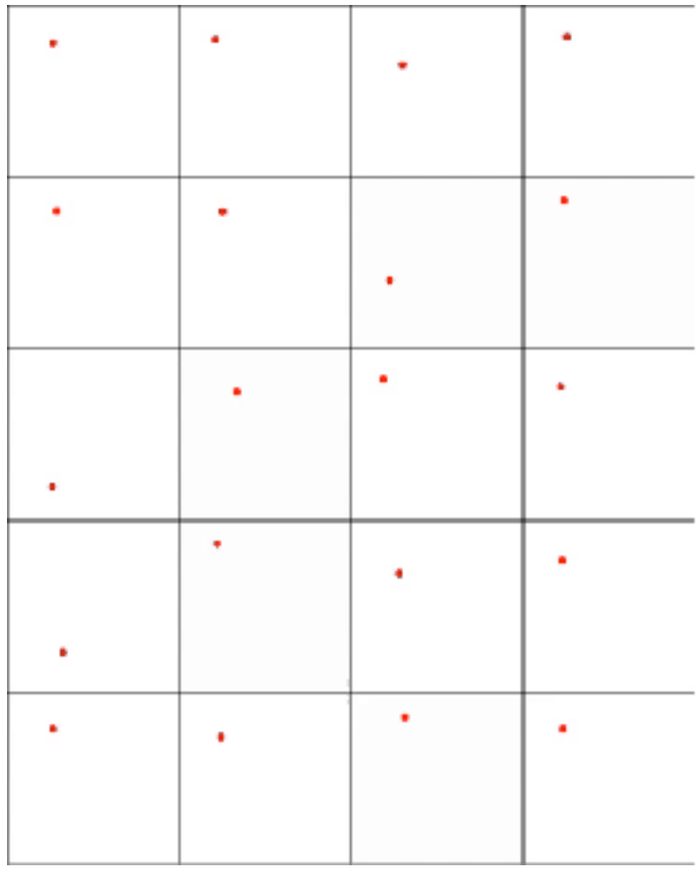


stroke order: 1 2 3 4 5

Original

Human drawings

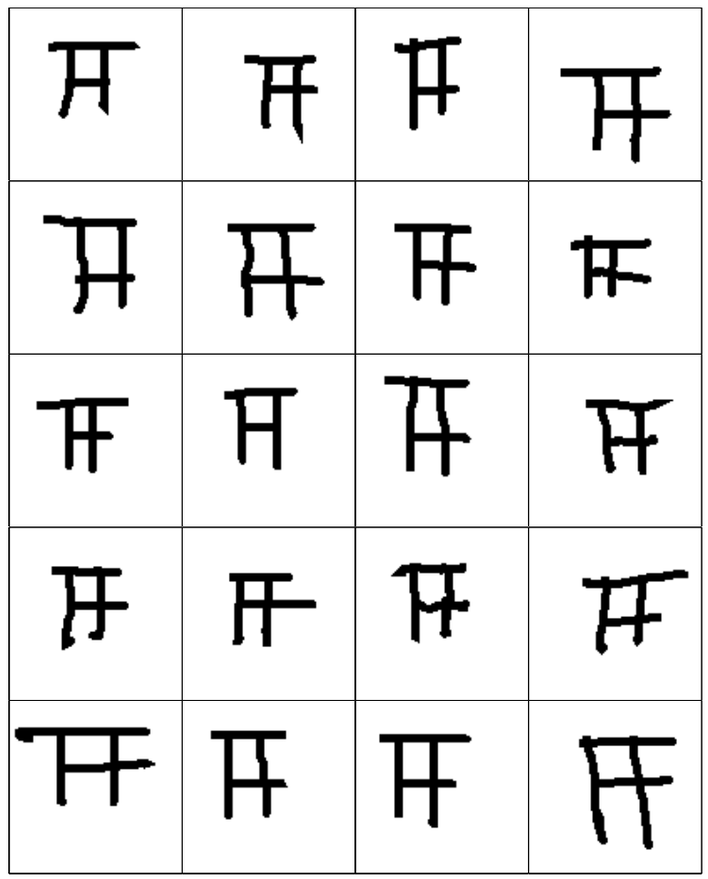


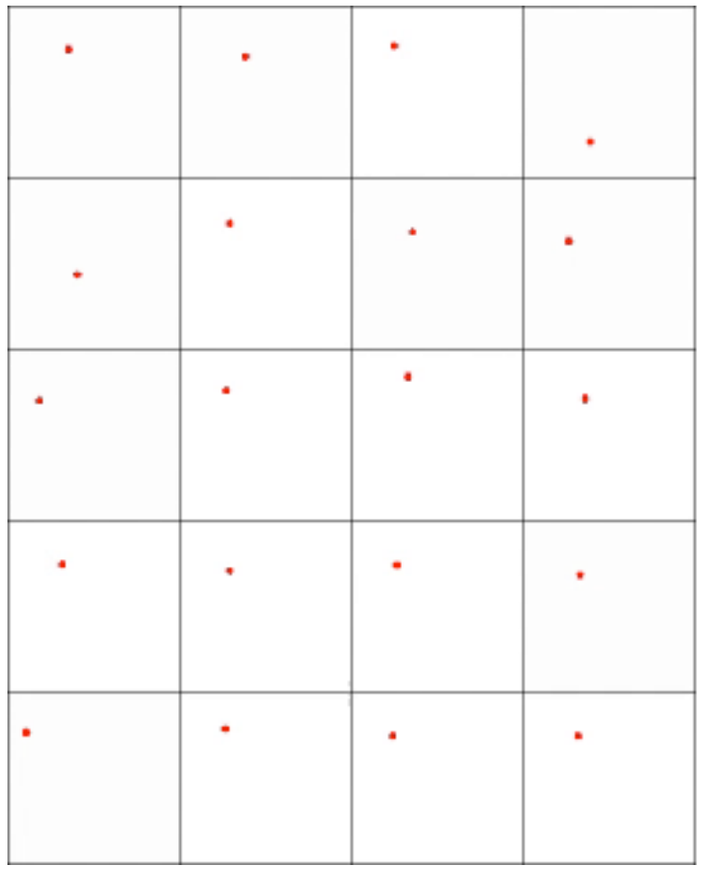


stroke order: 1 2 3 4 5

Original

Human drawings

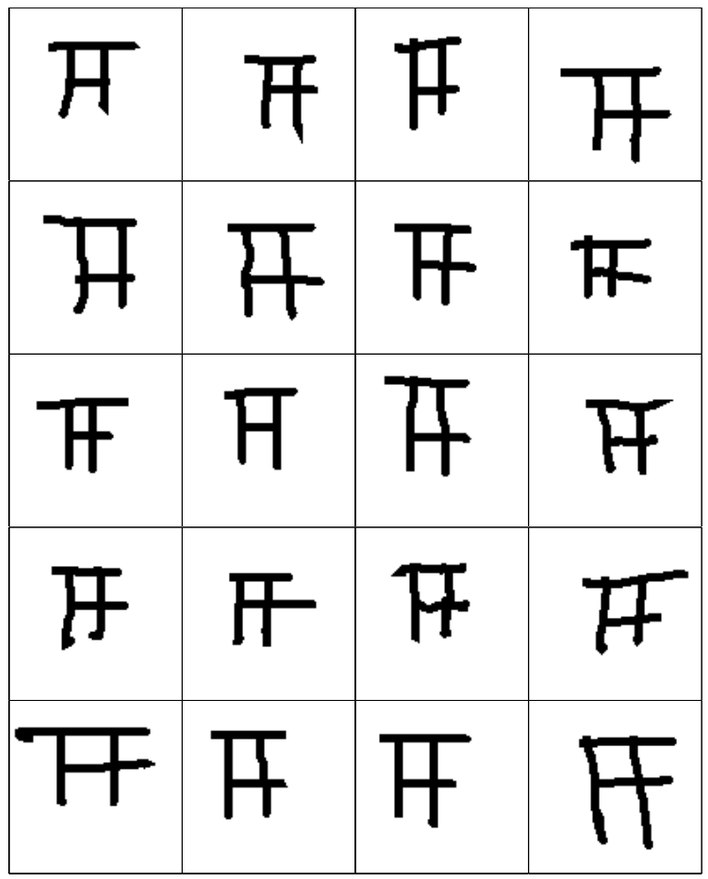


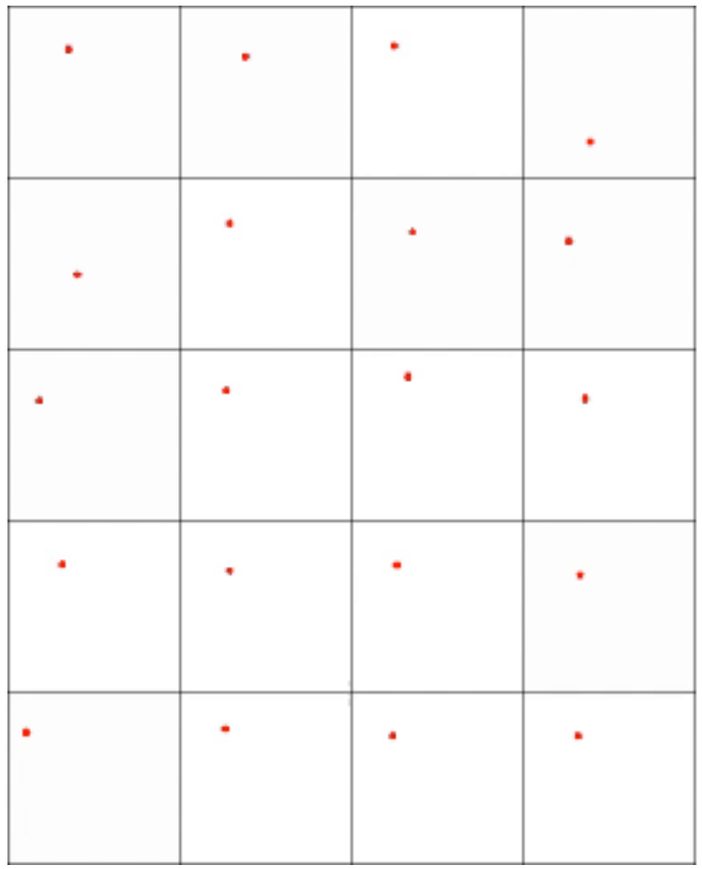


stroke order: 1 2 3 4 5

Original

Human drawings

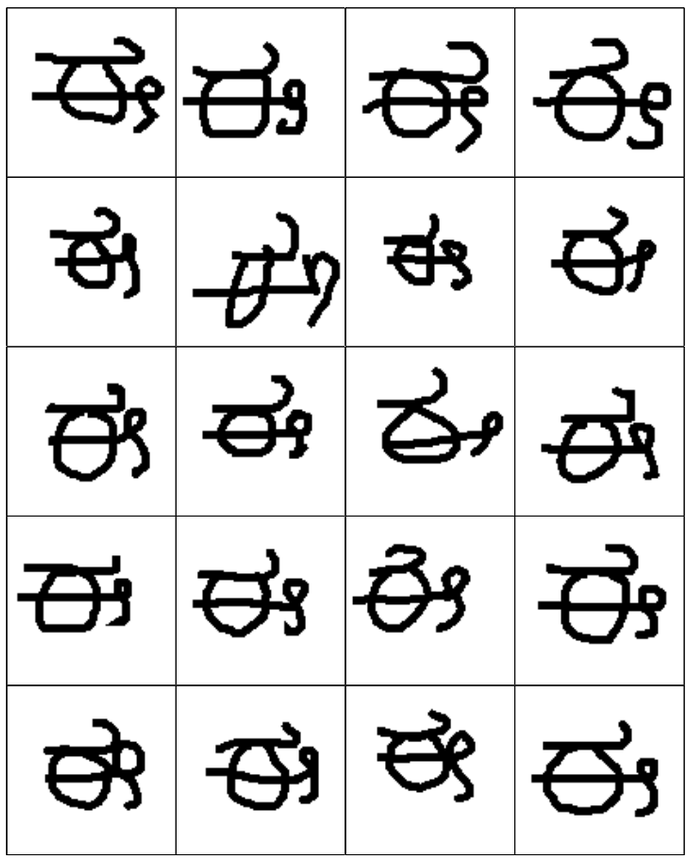


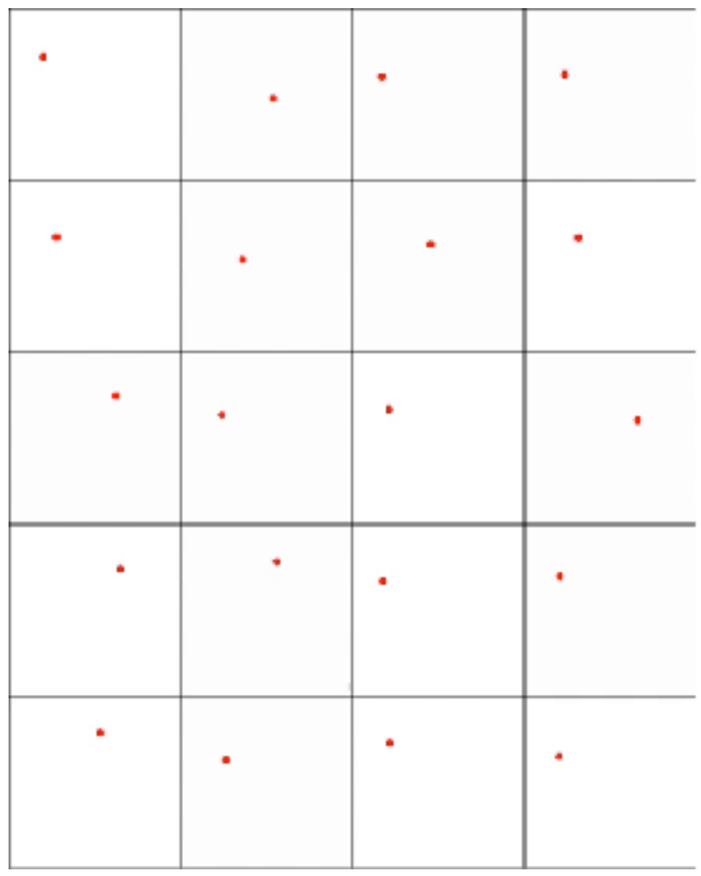


stroke order: 1 2 3 4 5

Original

Human drawings

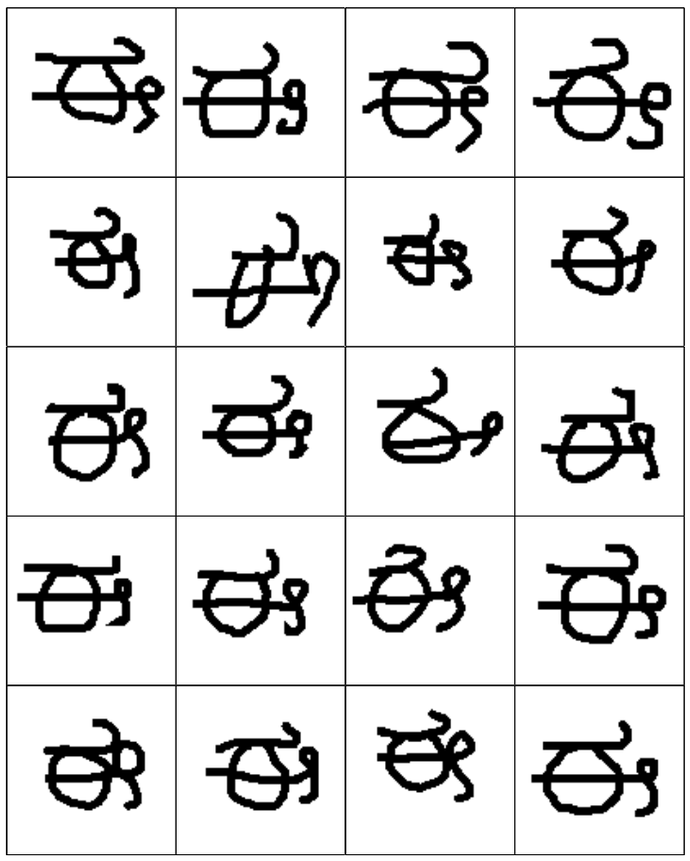


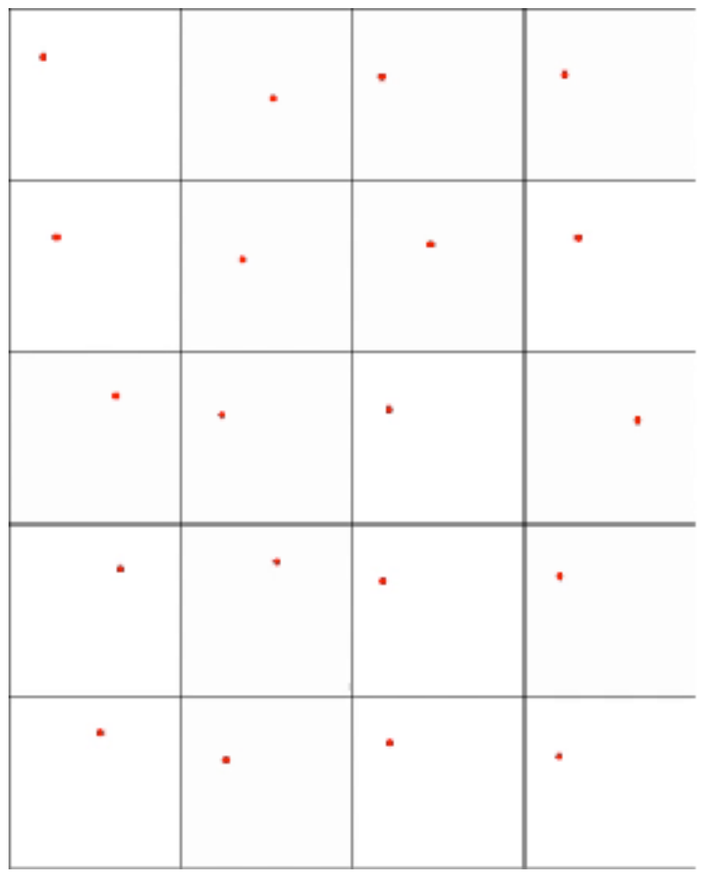


stroke order: 1 2 3 4 5

Original

Human drawings





stroke order: 1 2 3 4 5

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

θ latent program

I raw binary image

renderer

prior on parts, relations, etc.

Bayes' rule

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

relation: attached along

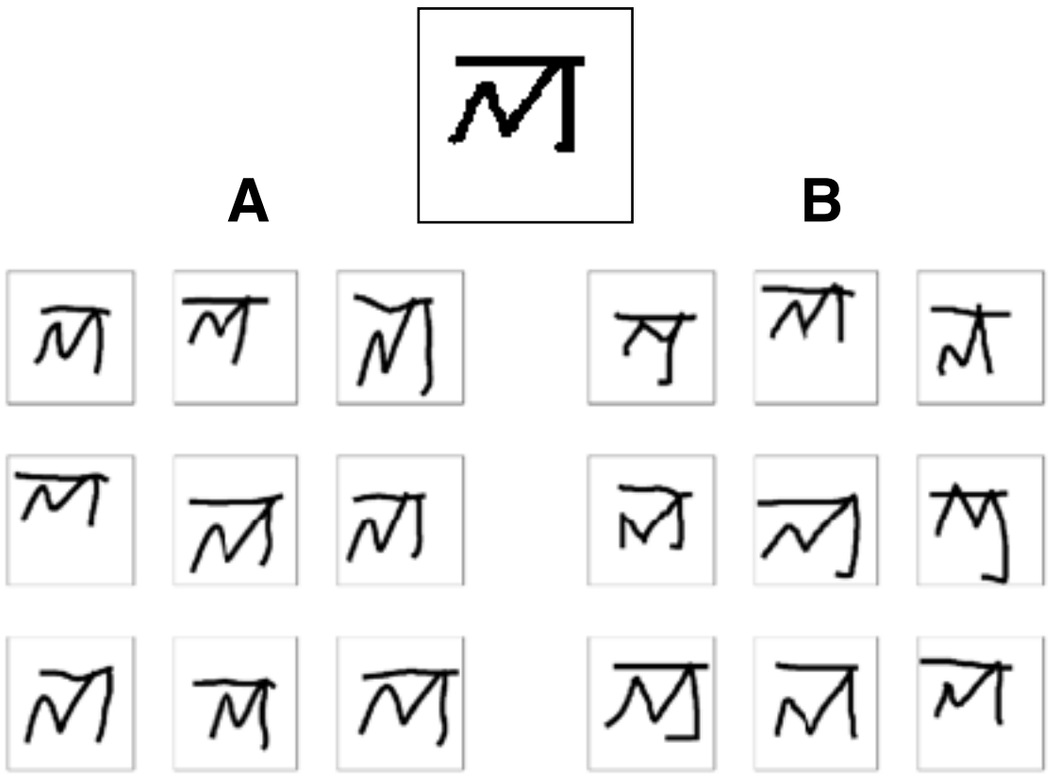
relation: attached along

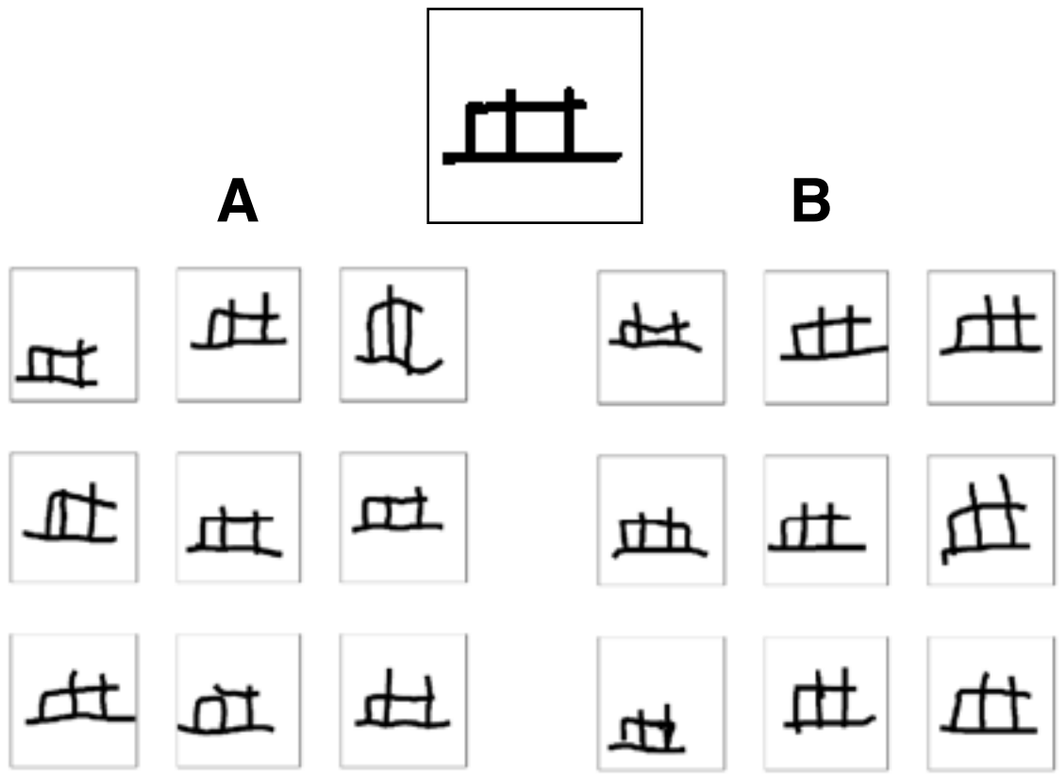
relation: attached at start

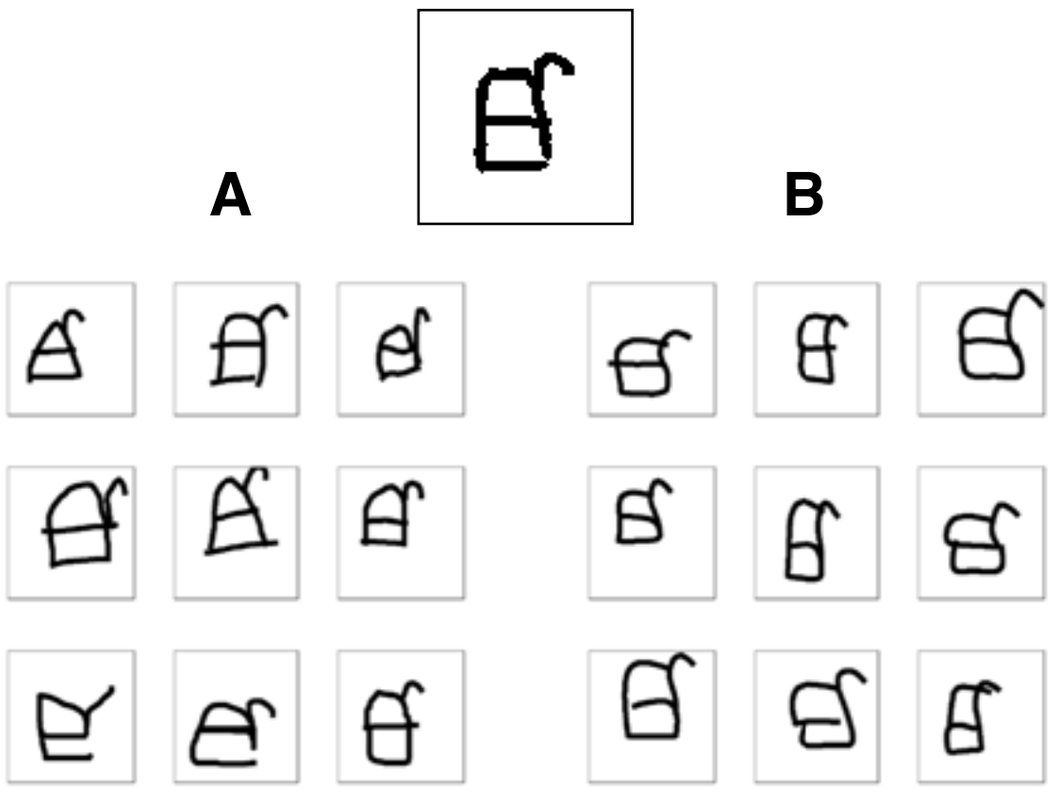
inference

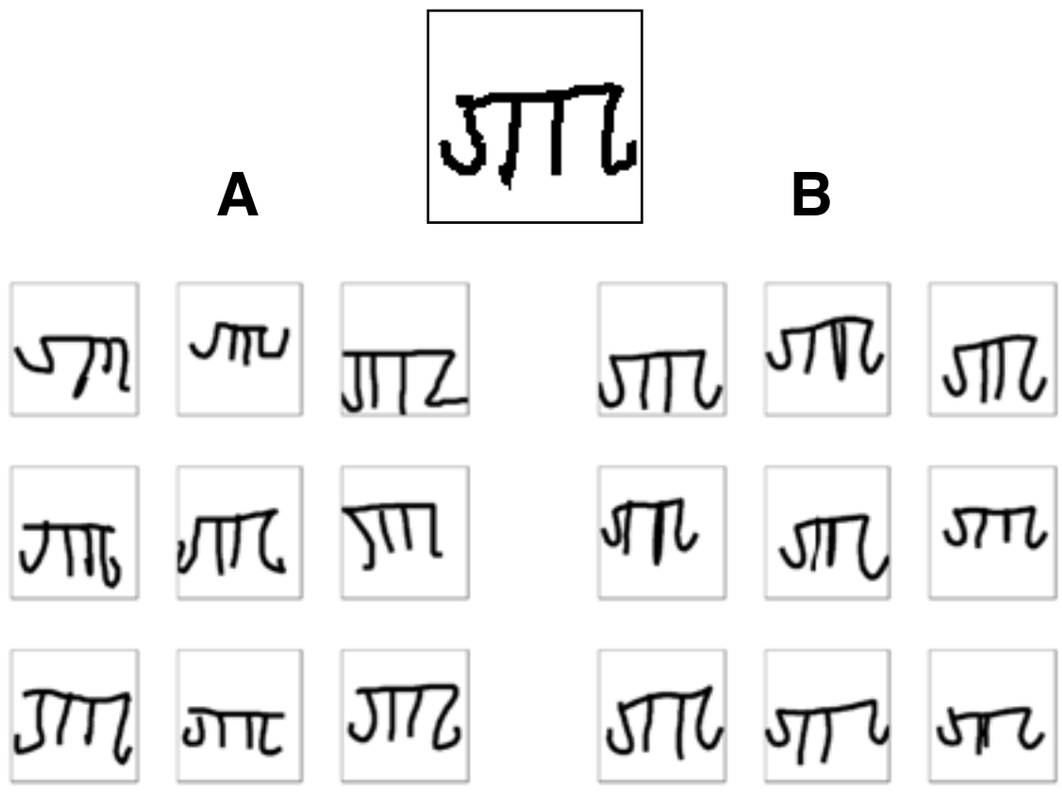
Task: “Generate a new example”

A “visual Turing test” for generating new examples

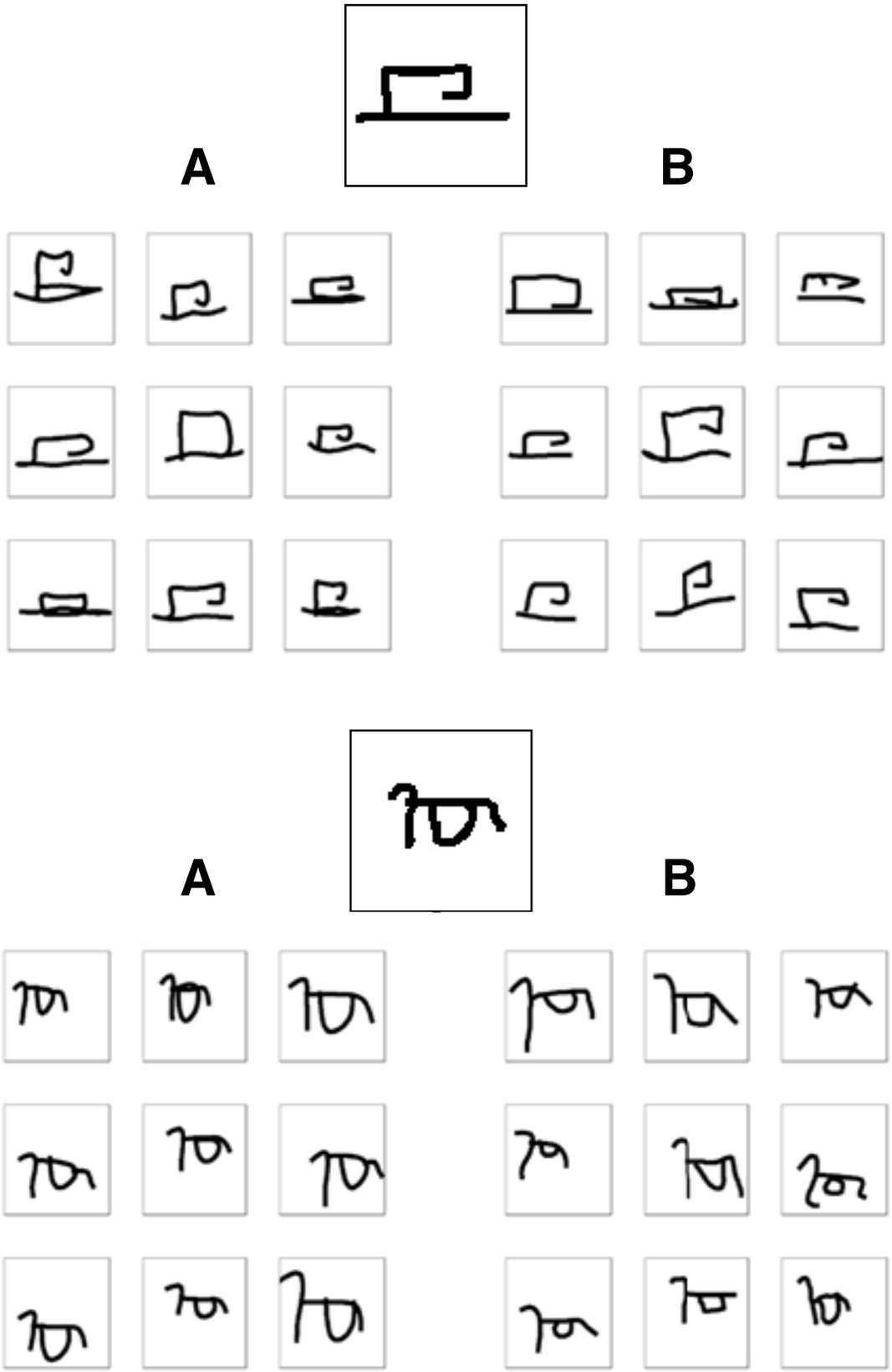


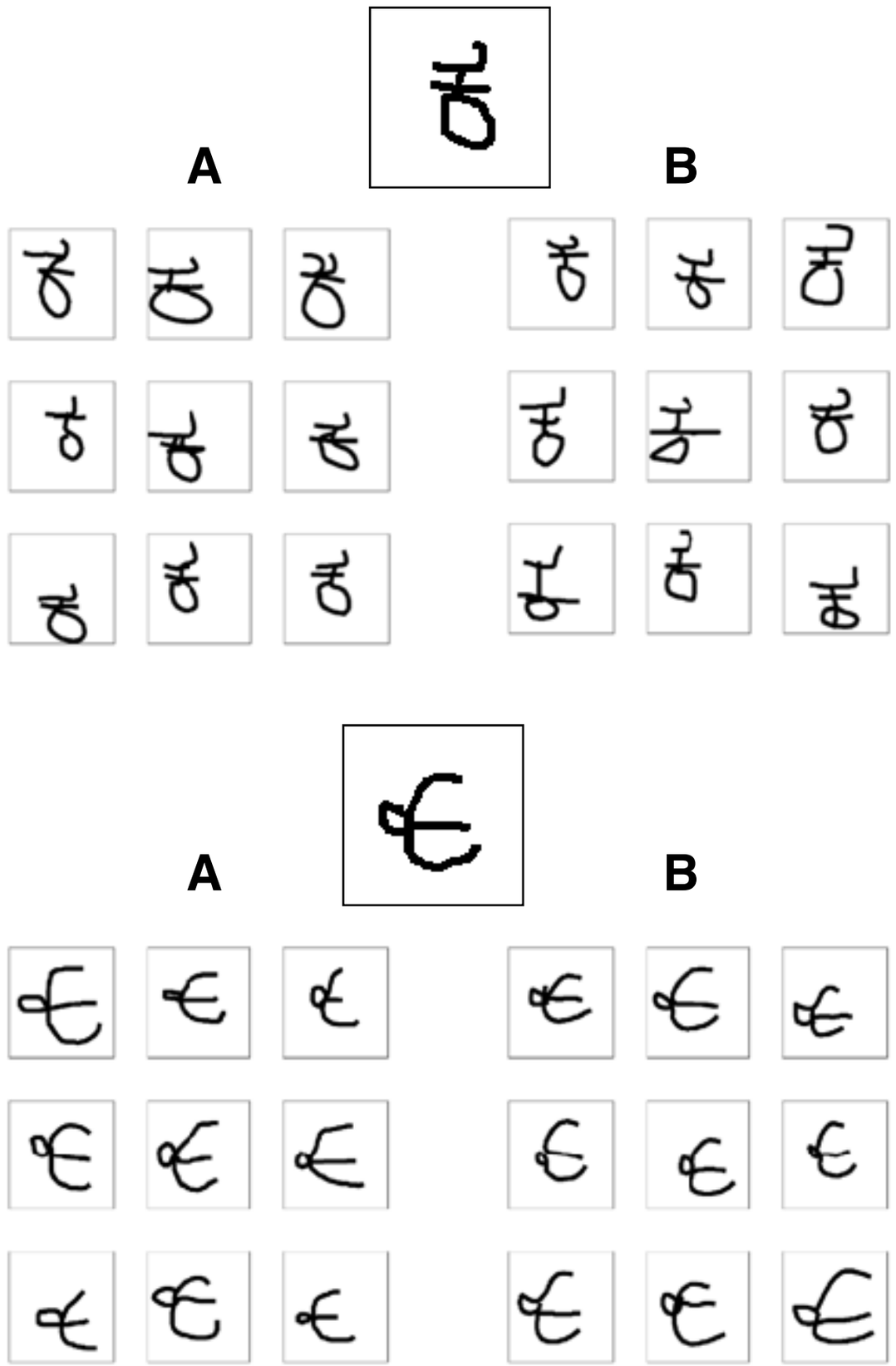






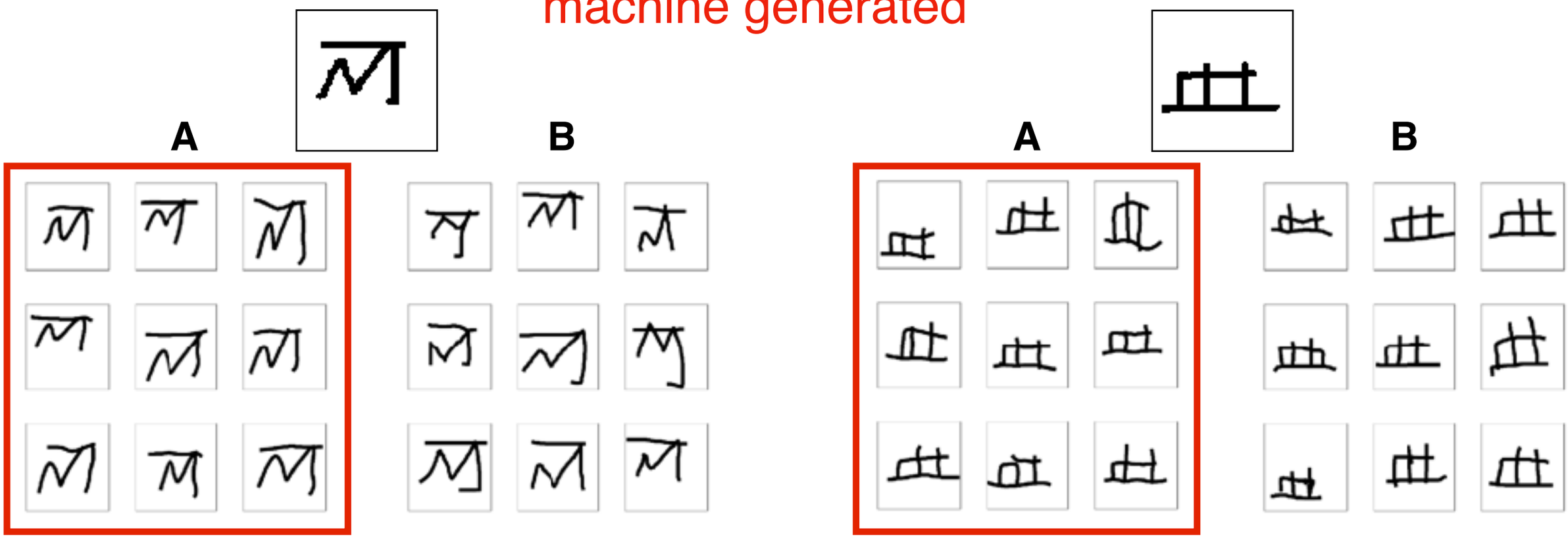
A “visual Turing test” for generating new examples

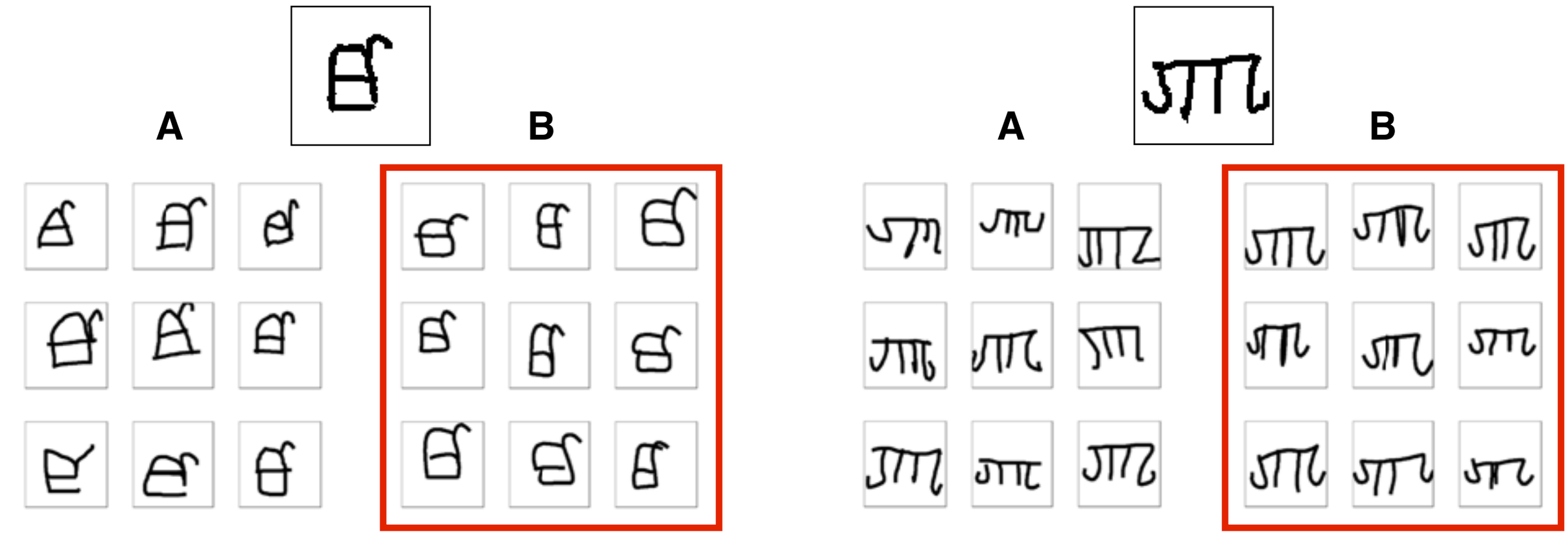




A “visual Turing test” for generating new examples

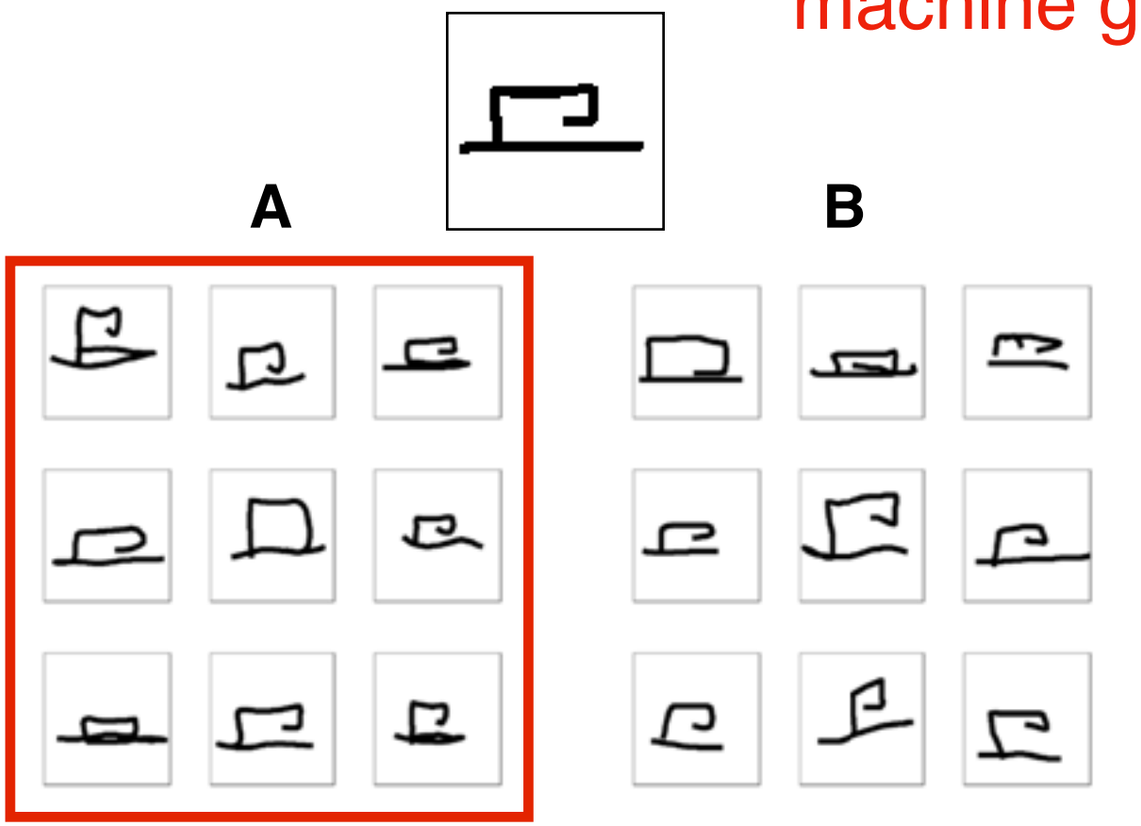
machine generated

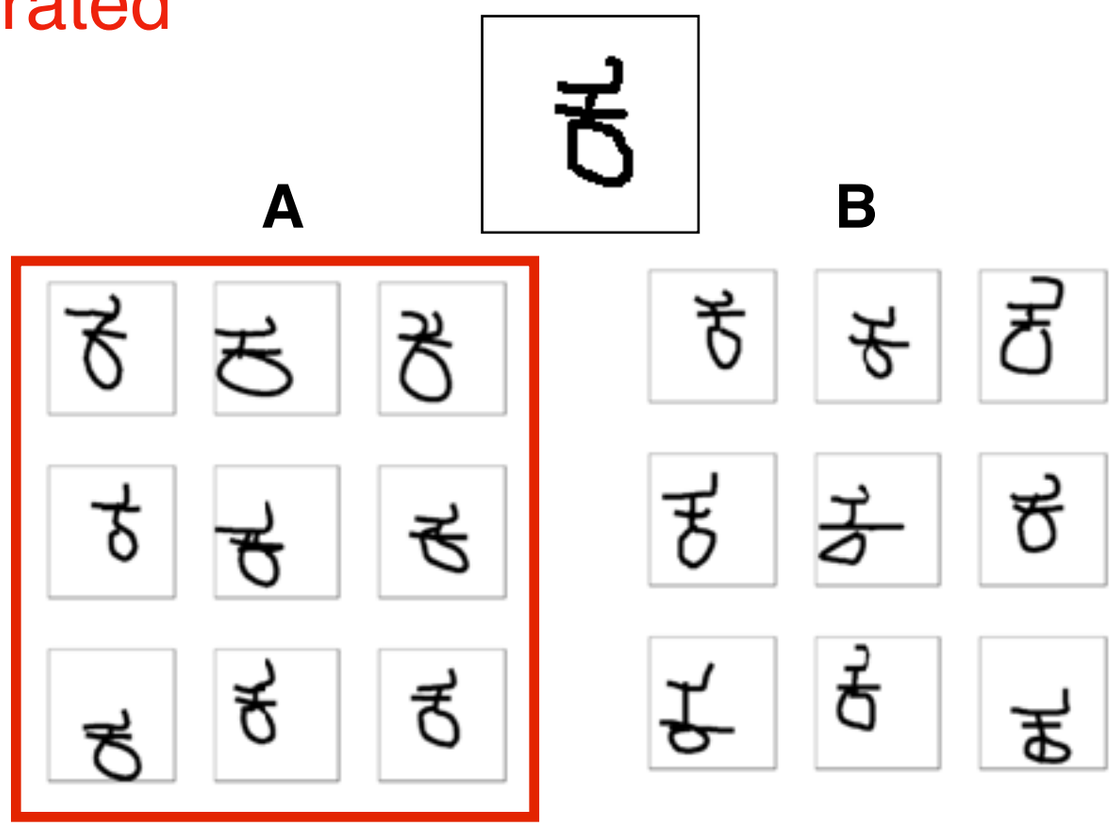


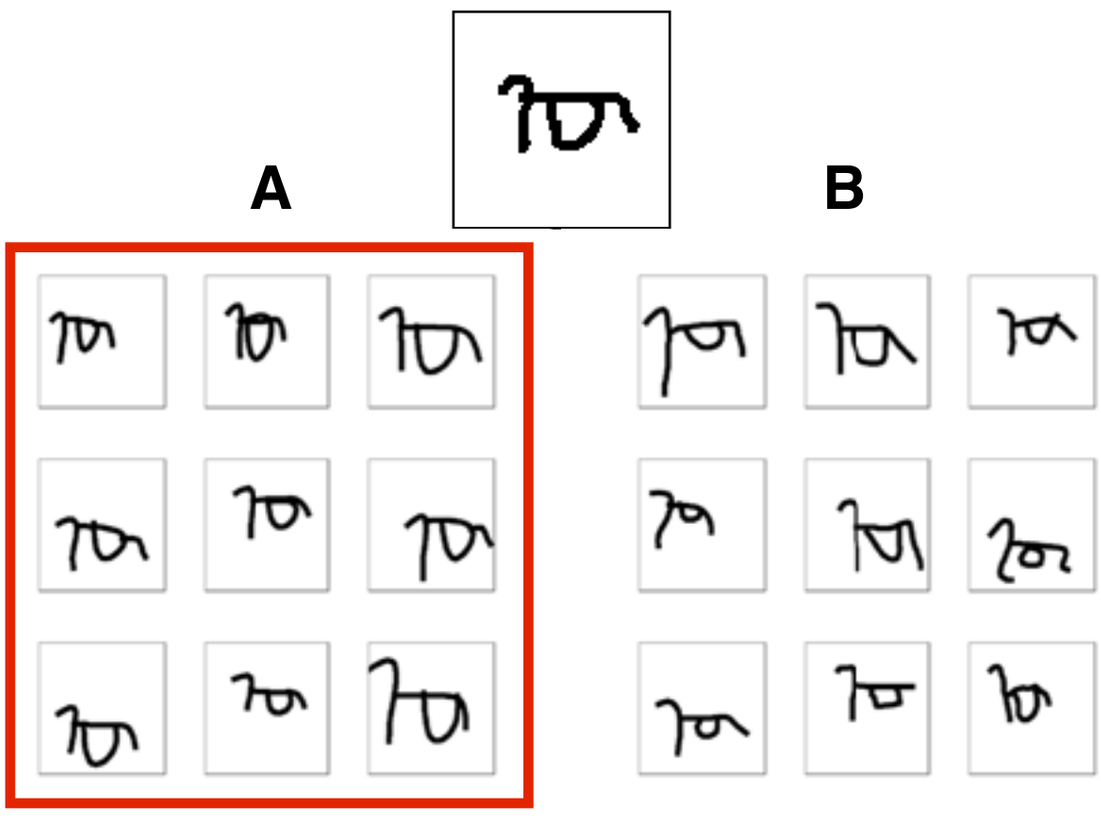


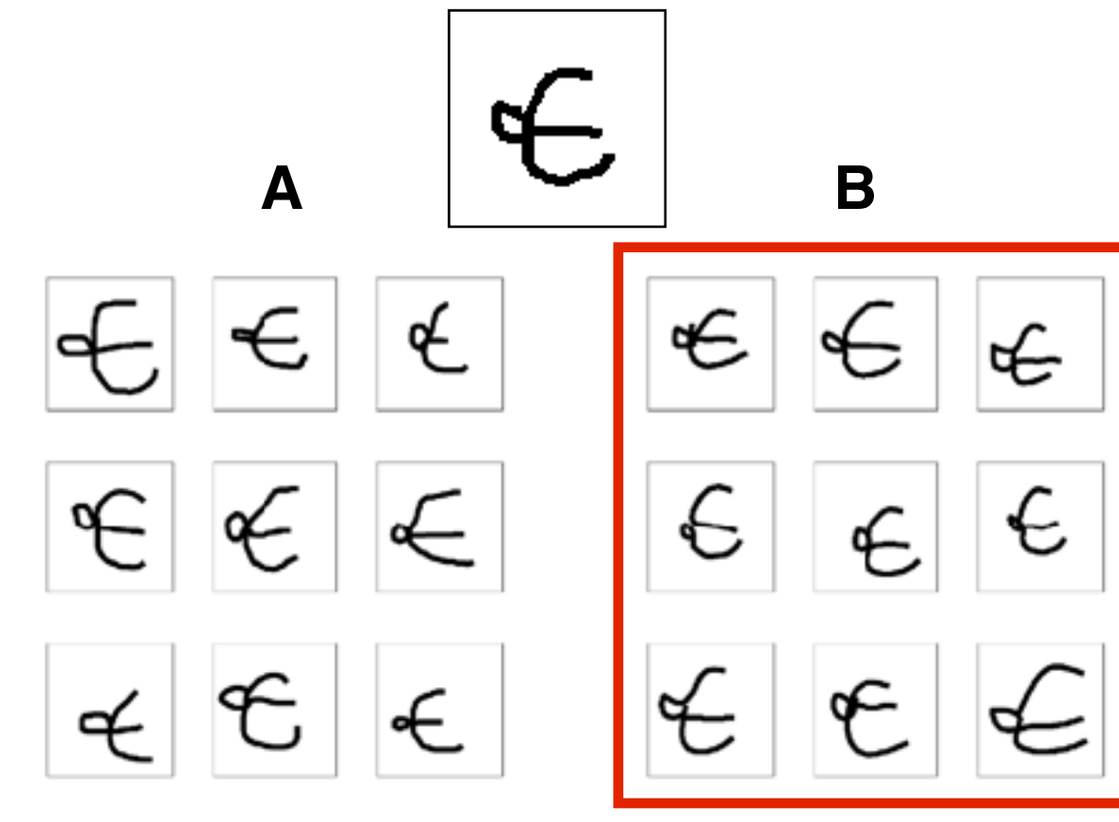
A “visual Turing test” for generating new examples

machine generated



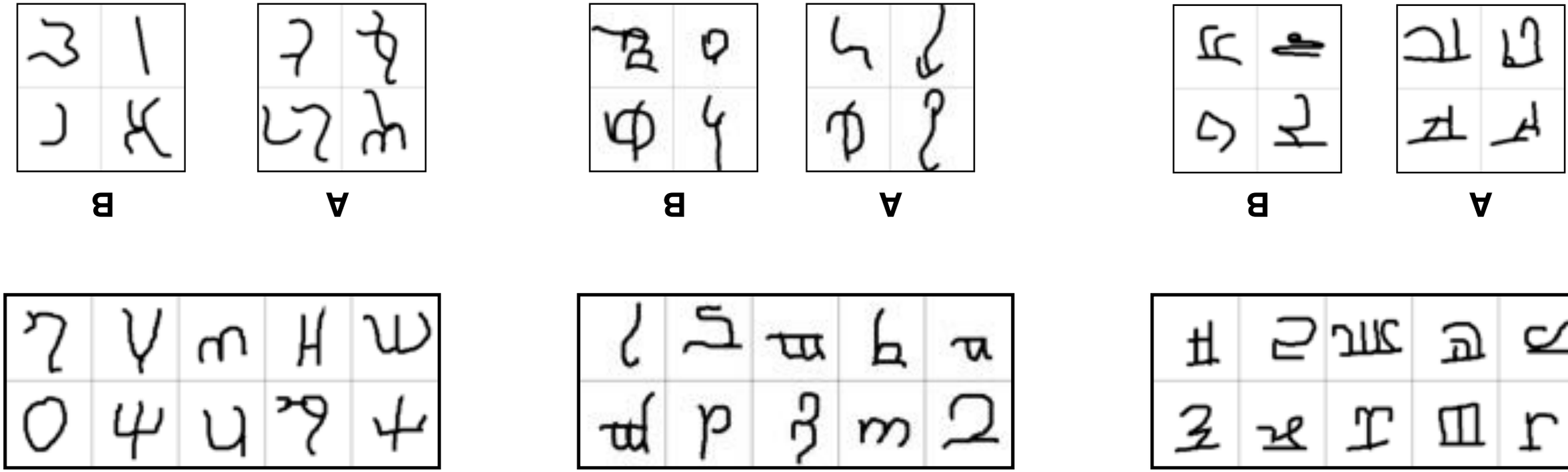






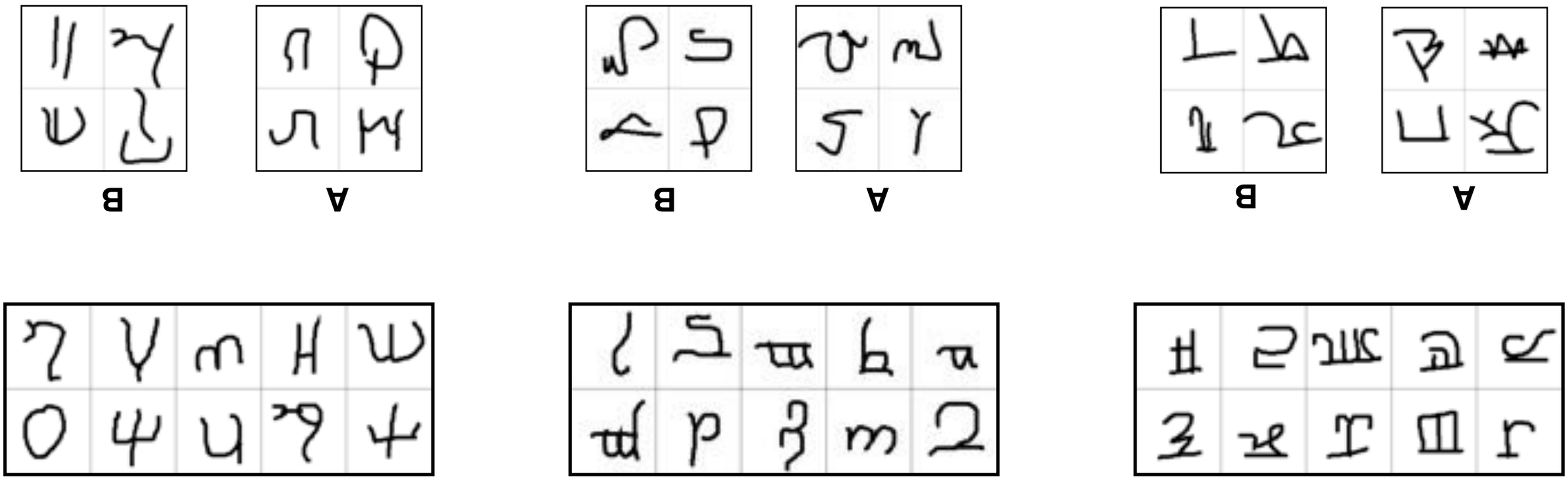
Task: “Generate a new character from the same alphabet”

3 seconds remaining



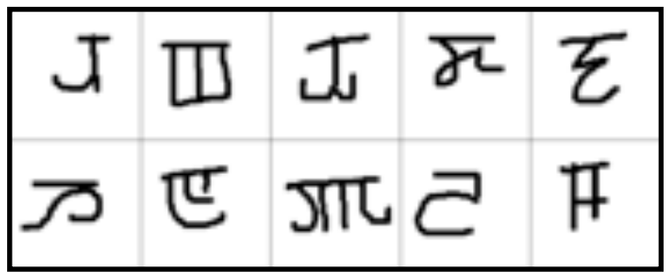
A "visual Turing test" for generating new concepts

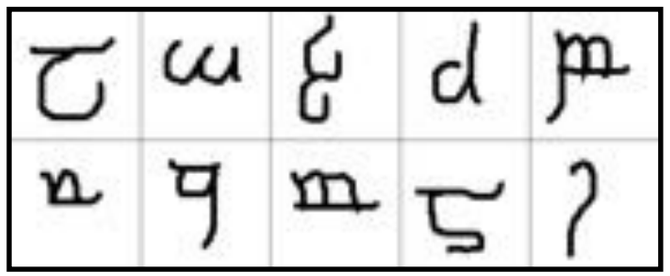
A “visual Turing test” for generating new concepts

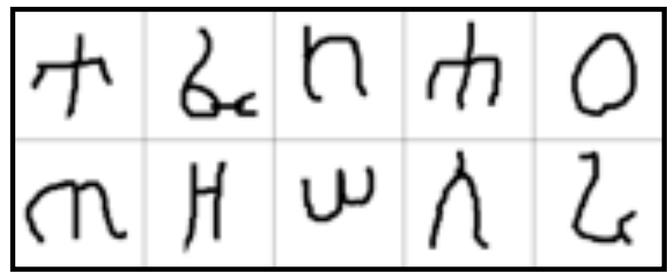


A “visual Turing test” for generating new concepts

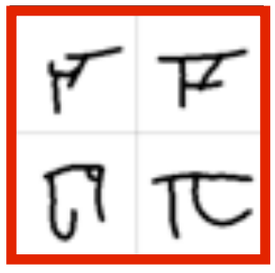
machine generated

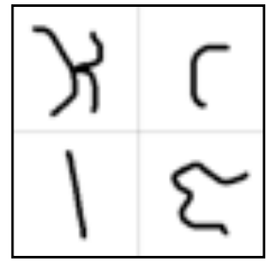






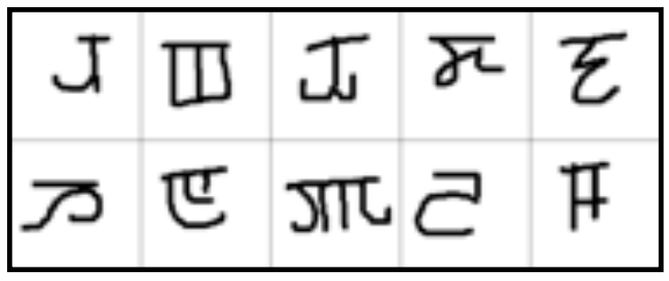
A B A B A B

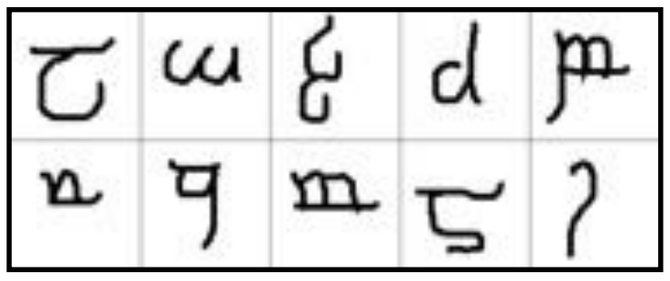


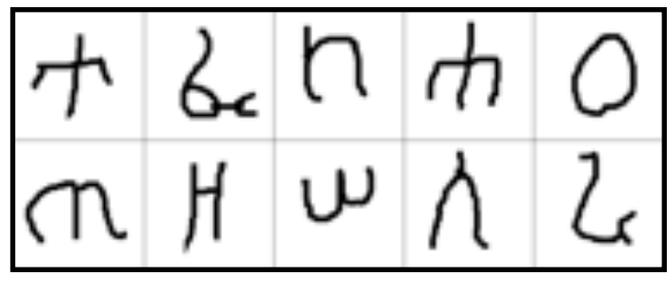


A “visual Turing test” for generating new concepts

machine generated

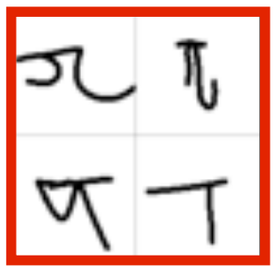






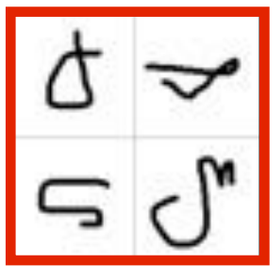
A

B

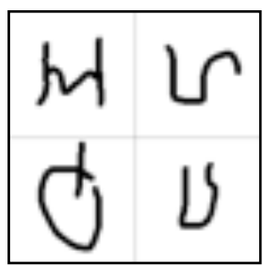


A

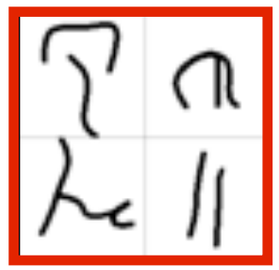
B



A



B



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

relation: attached along

relation: attached along

relation: attached at start

θ latent program

I raw binary image

renderer

prior on parts, relations, etc.

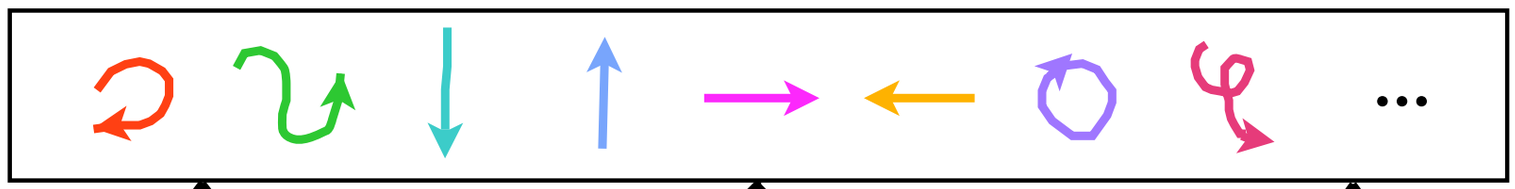
Bayes' rule

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

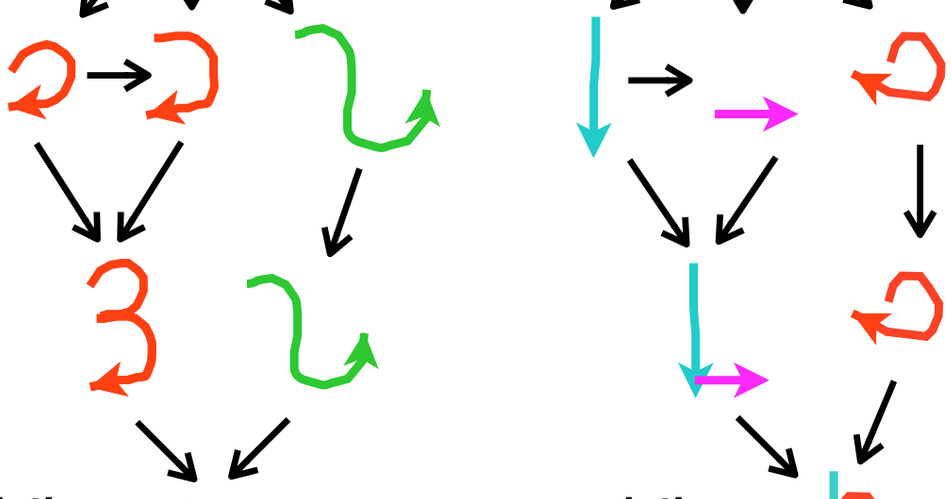
inference

Probabilistic program induction

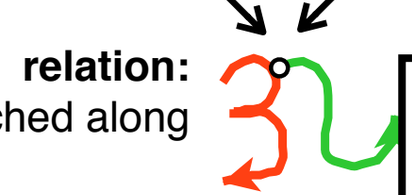
primitives
(1D curvelets, 2D
patches, 3D geons,
actions, sounds, etc.)



sub-parts



parts



relation:
attached along

object template

type level

procedure GENERATETYPE

$\kappa \leftarrow P(\kappa)$ ▷ Sample number of parts

for i = 1 ... κ **do**

$n_i \leftarrow P(n_i|\kappa)$ ▷ Sample number of sub-parts

for j = 1 ... n_i **do**

$s_{ij} \leftarrow P(s_{ij}|s_{i(j-1)})$ ▷ Sample sub-part sequence

end for

$R_i \leftarrow P(R_i|S_1, ..., S_{i-1})$ ▷ Sample relation

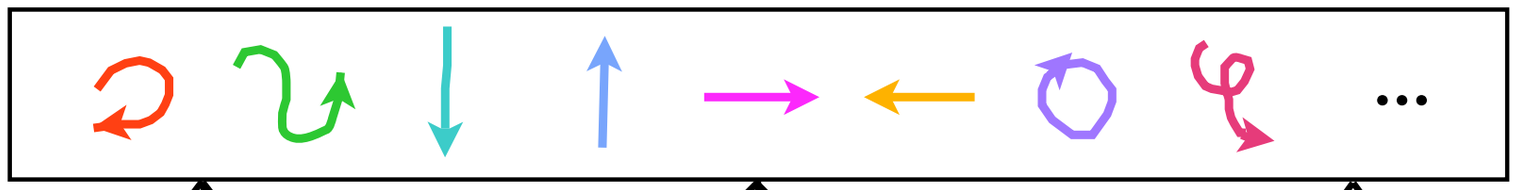
end for

$\psi \leftarrow \{\kappa, R, S\}$

return @GENERATETOKEN(ψ) ▷ Return program

Probabilistic program induction

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



sub-parts

parts

relation:
attached along

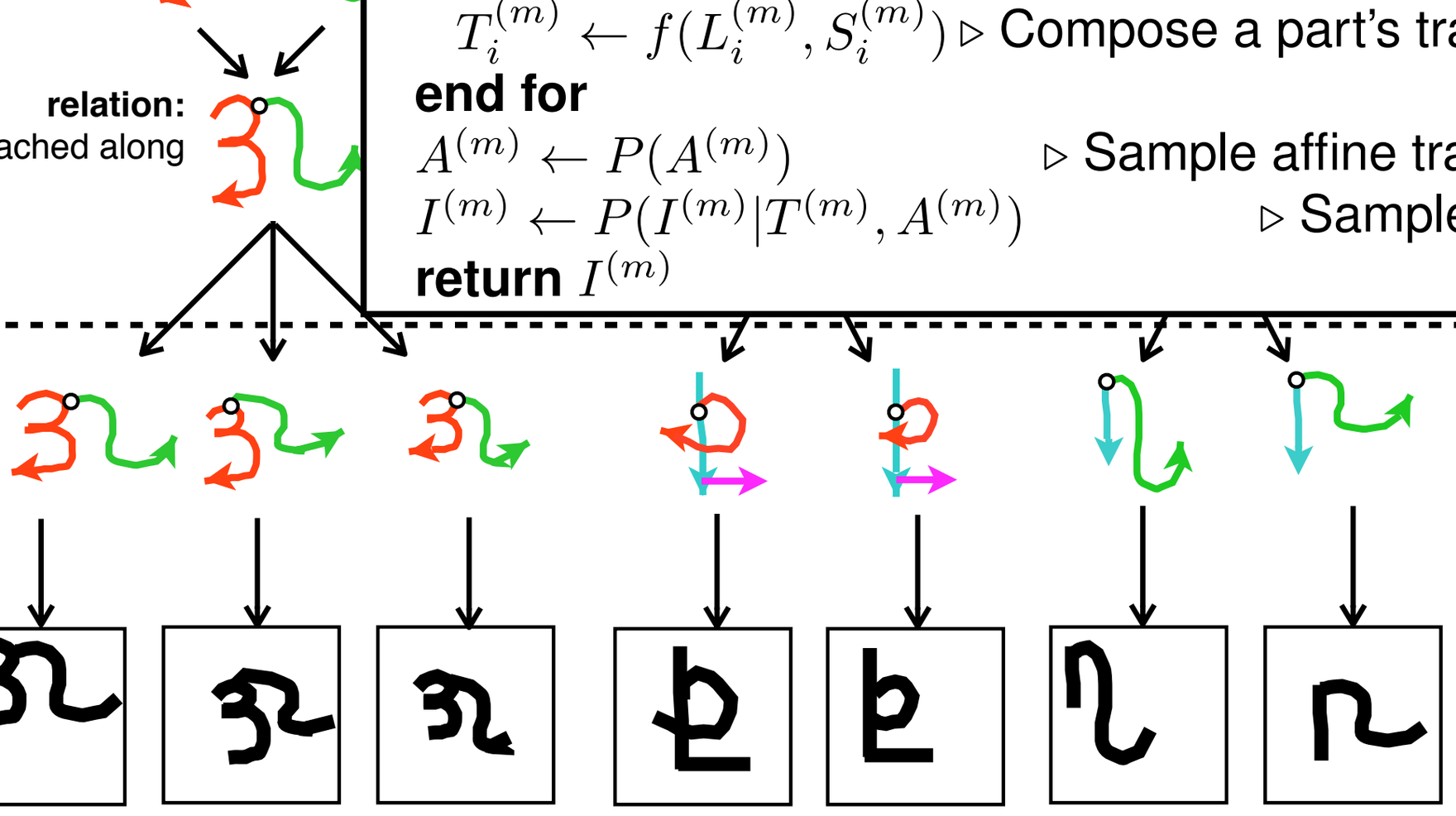
object template

type level

token level

exemplars

raw data



procedure GENERATETOKEN(ψ)

for $i = 1...\kappa$ **do**

$S_i^{(m)} \leftarrow P(S_i^{(m)}|S_i)$ ▷ Add motor variance

$L_i^{(m)} \leftarrow P(L_i^{(m)}|R_i, T_1^{(m)}, ..., T_{i-1}^{(m)})$ ▷ Sample part's start location

$T_i^{(m)} \leftarrow f(L_i^{(m)}, S_i^{(m)})$ ▷ Compose a part's trajectory

end for

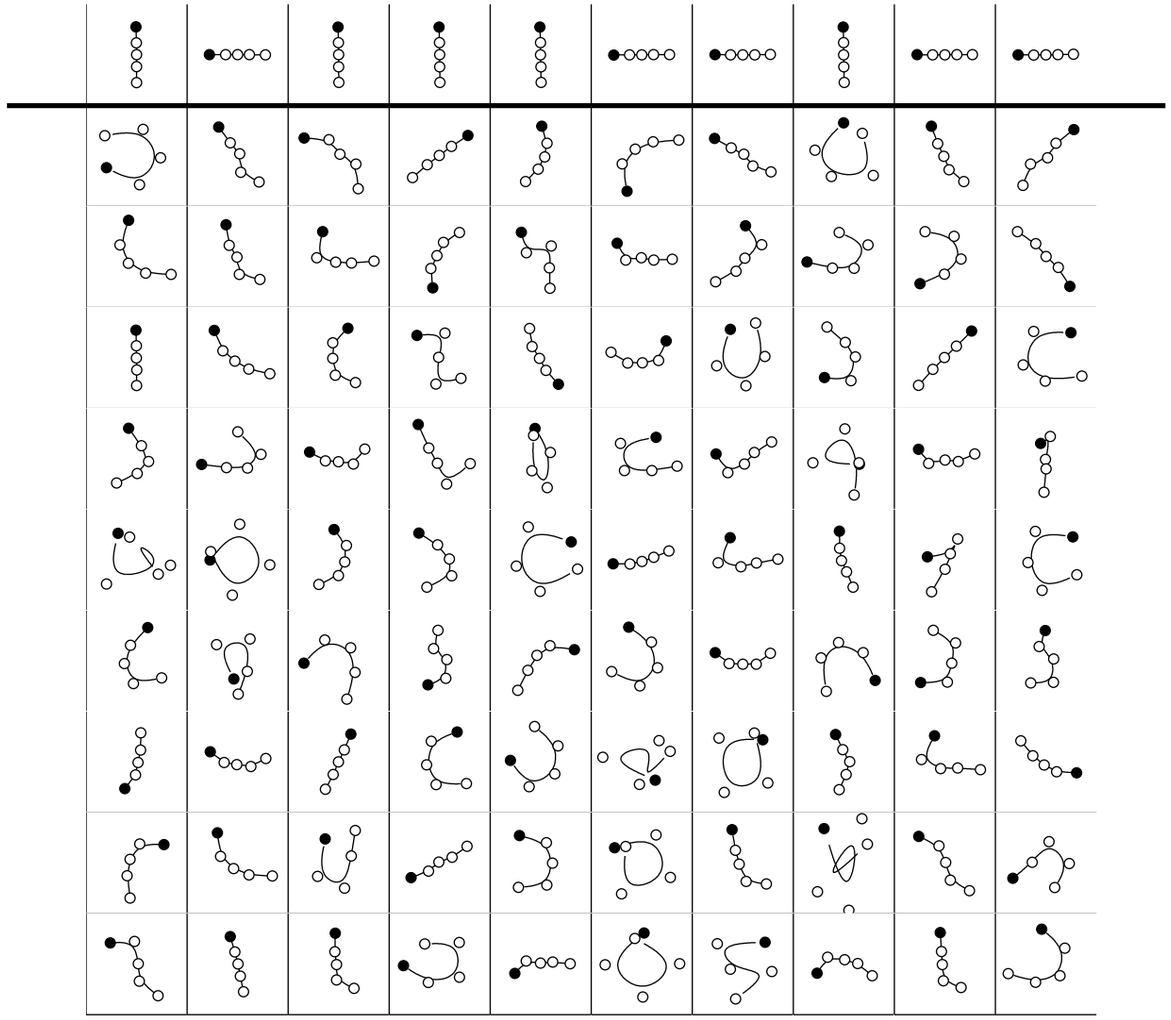
$A^{(m)} \leftarrow P(A^{(m)})$ ▷ Sample affine transform

$I^{(m)} \leftarrow P(I^{(m)}|T^{(m)}, A^{(m)})$ ▷ Sample image

return $I^{(m)}$

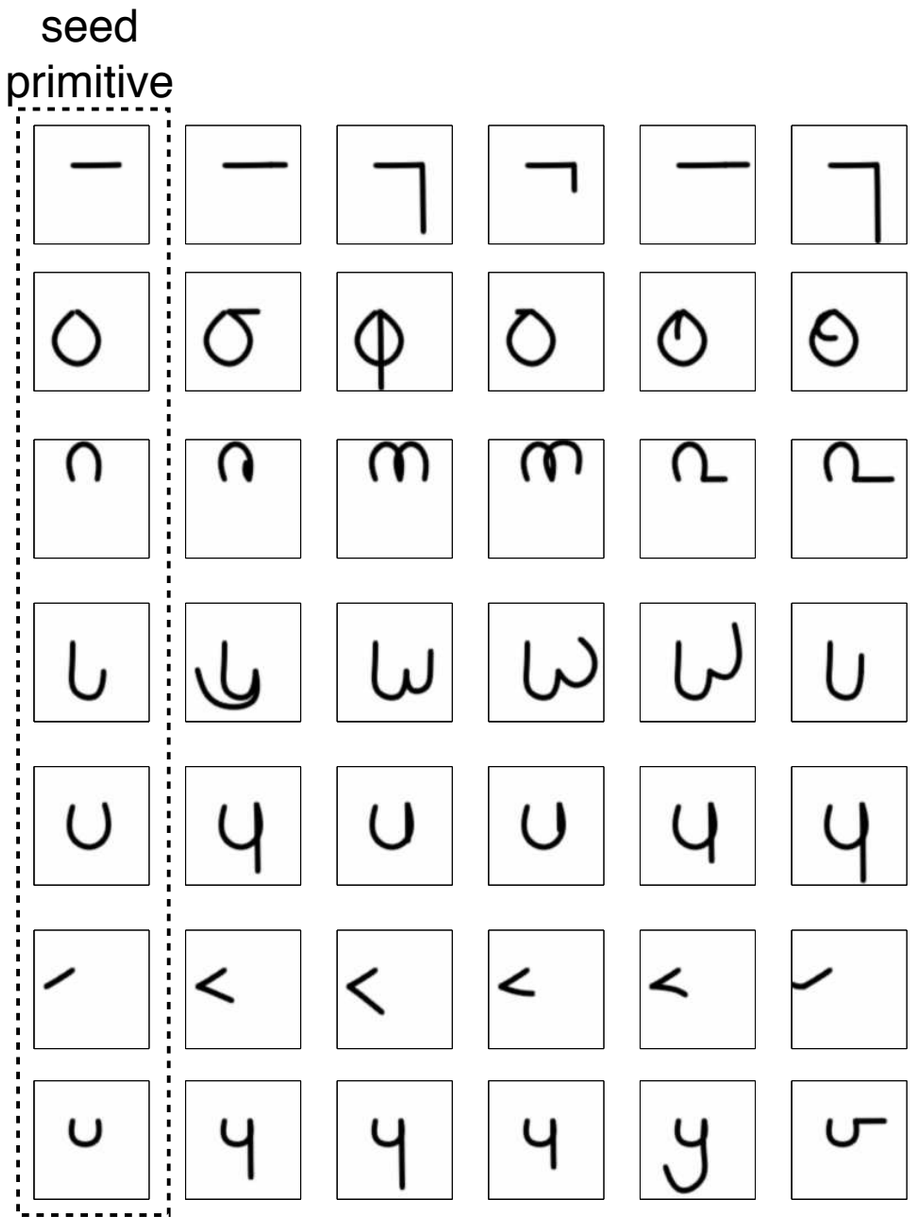
Learning a prior distribution over programs

learned action primitives



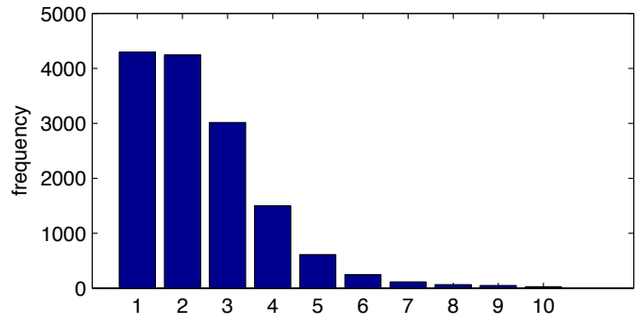
1250 primitives
scale selective
translation invariant

learned primitive transitions

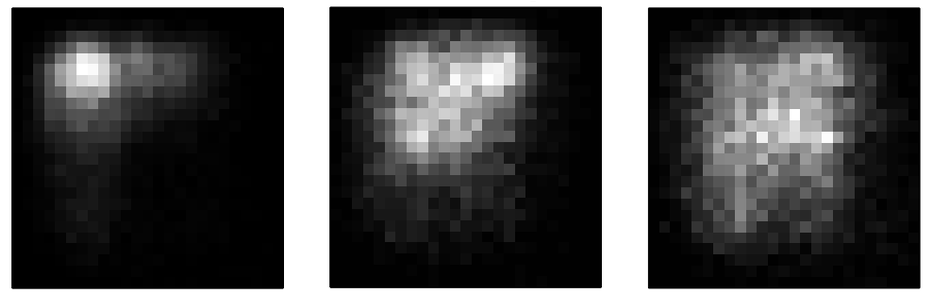


Learning a prior distribution over programs

number of strokes

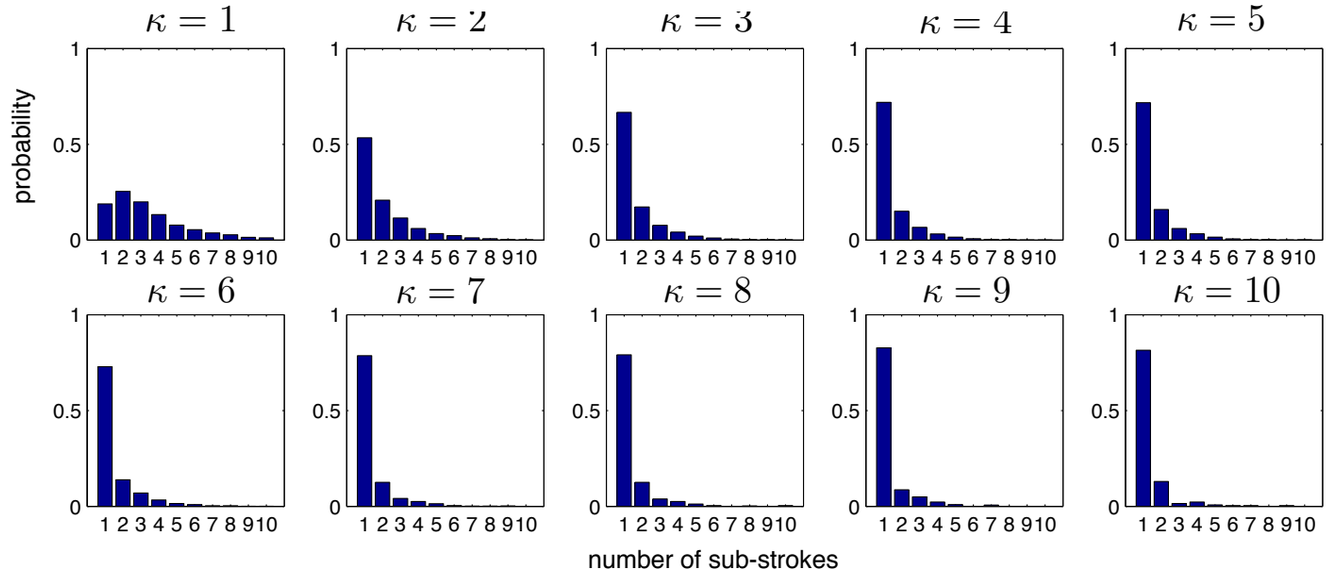


stroke start positions

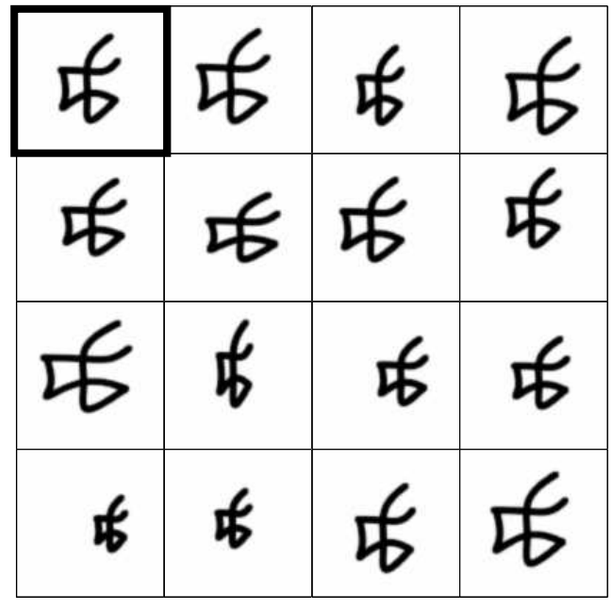


Stroke 1 2 ≥ 3

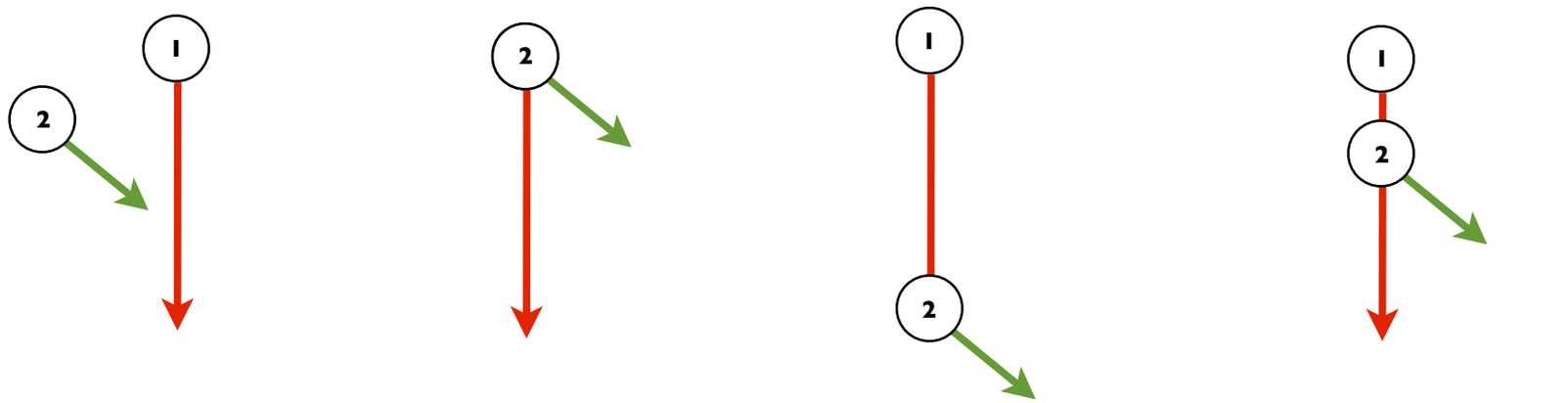
number of sub-strokes for a character with κ strokes



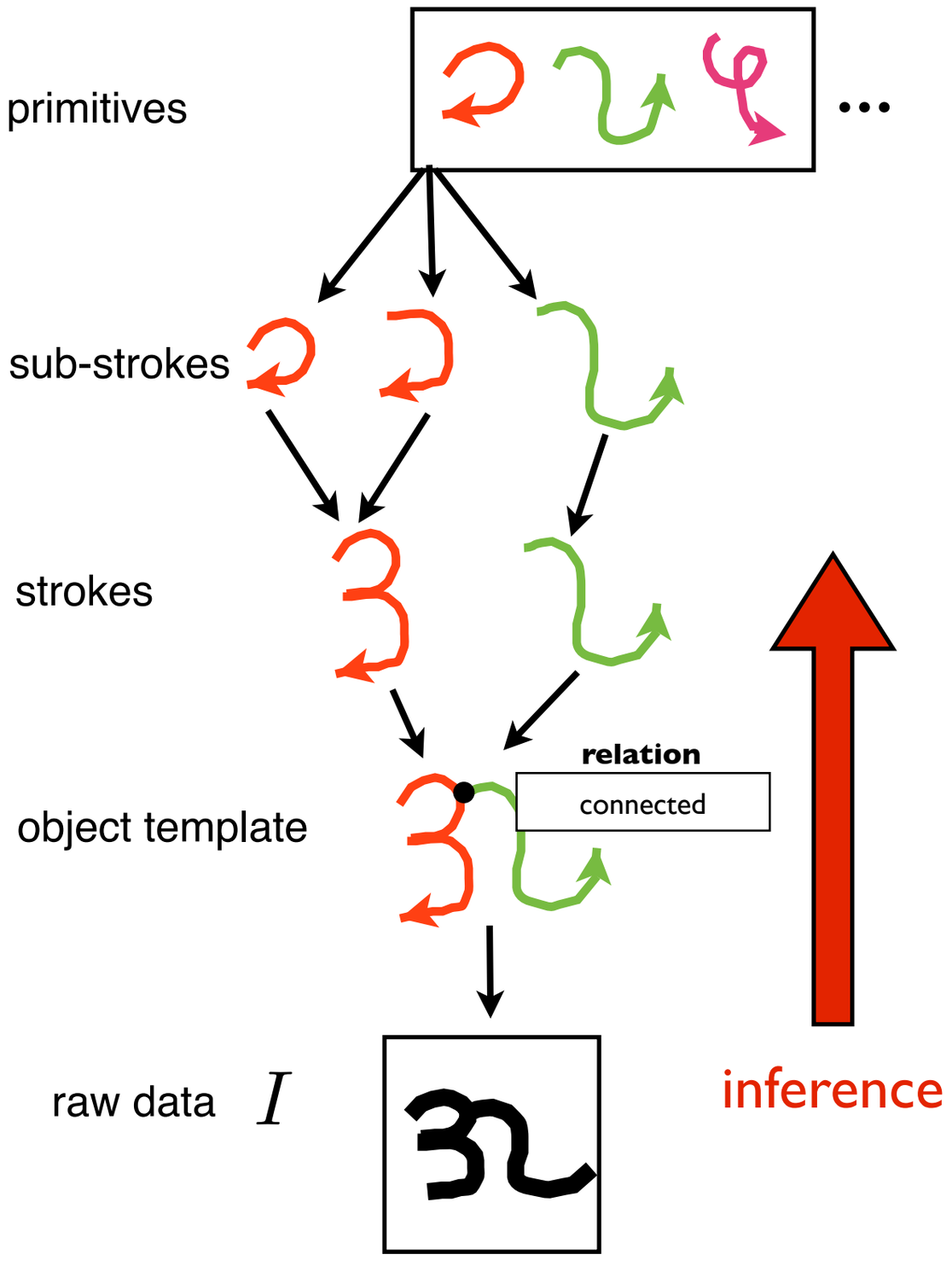
global transformations



relations between strokes



Approximate probabilistic inference



θ latent program

I raw binary image

Bayes' rule

renderer prior on programs

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

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

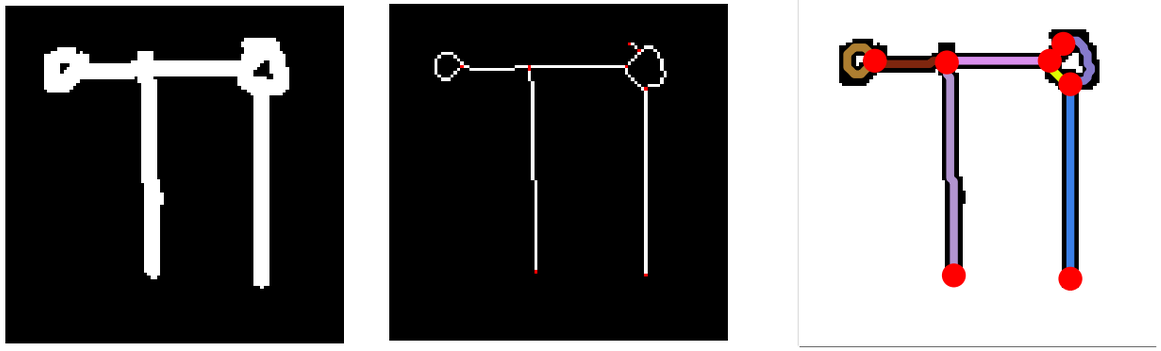
$$w_i \propto P(\theta^{[i]}|I)$$

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

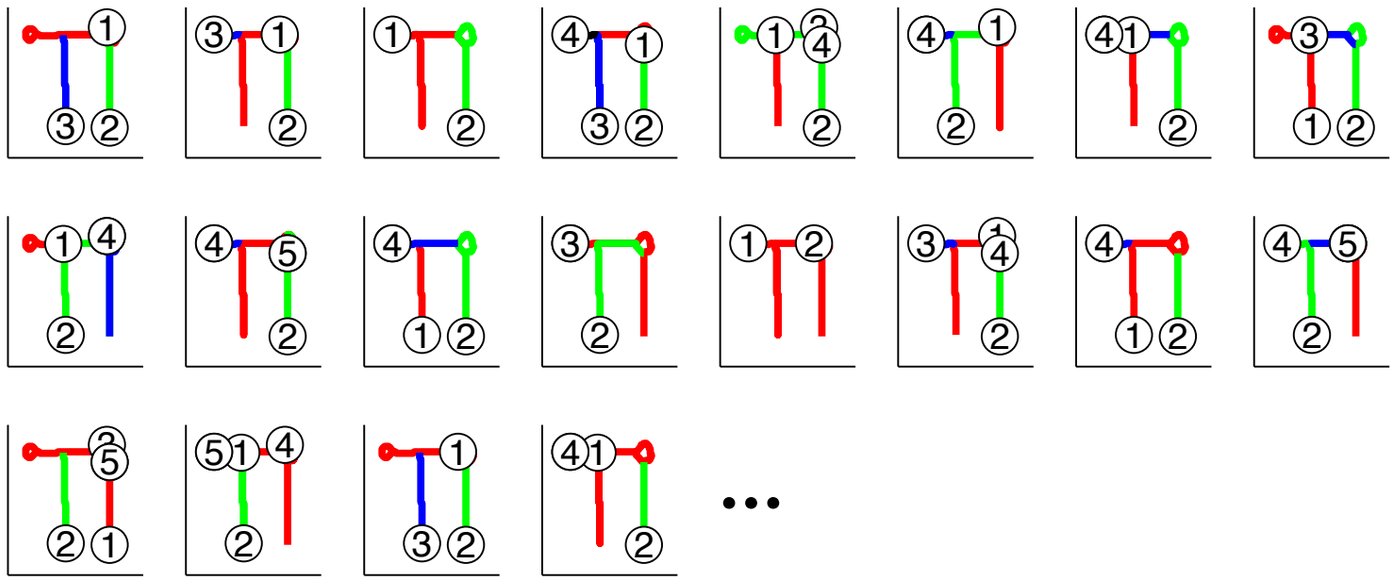
- fewer strokes
- high-probability primitive sequence
- use relations
- stroke order
- stroke directions

Approximate probabilistic inference

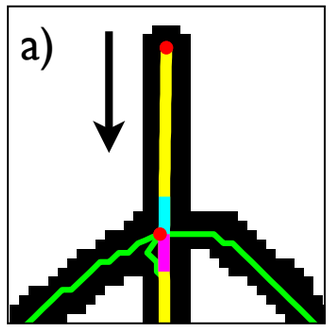
Step 1: characters as undirected graphs



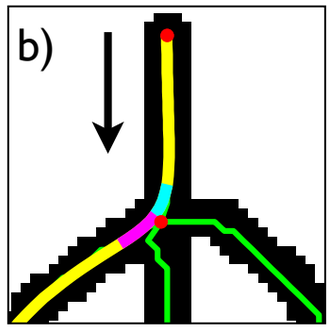
Step 2: guided random parses



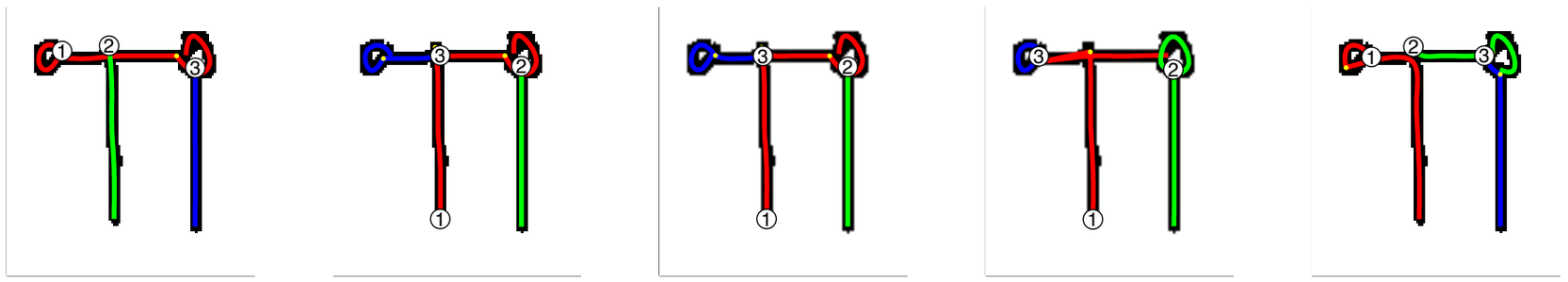
more likely



less likely



Step 3: Top-down fitting with gradient-based optimization

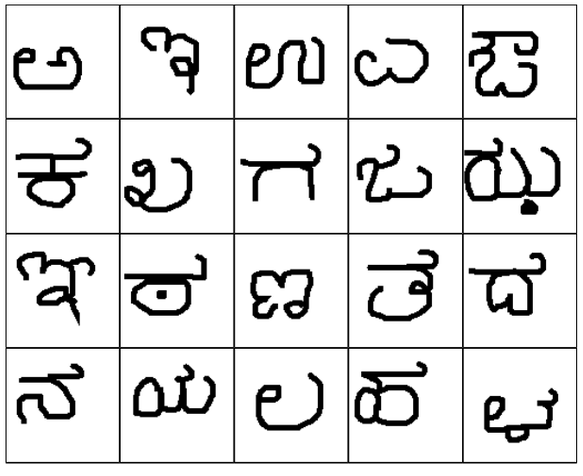


−531 −560 −568 −582 −588

log-probability

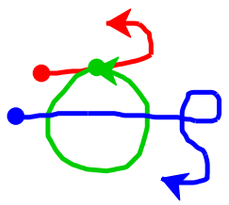
Human-level concept learning

the speed of learning

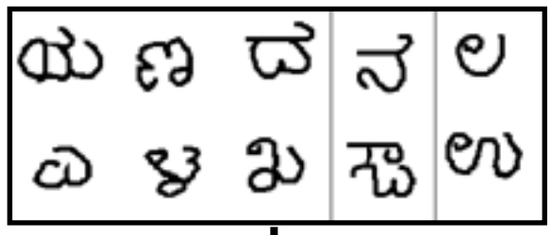


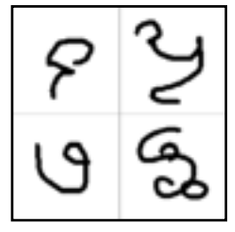
the richness of representation

parsing

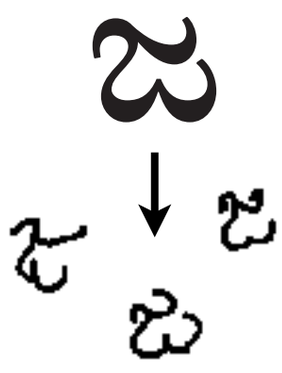


generating
new concepts

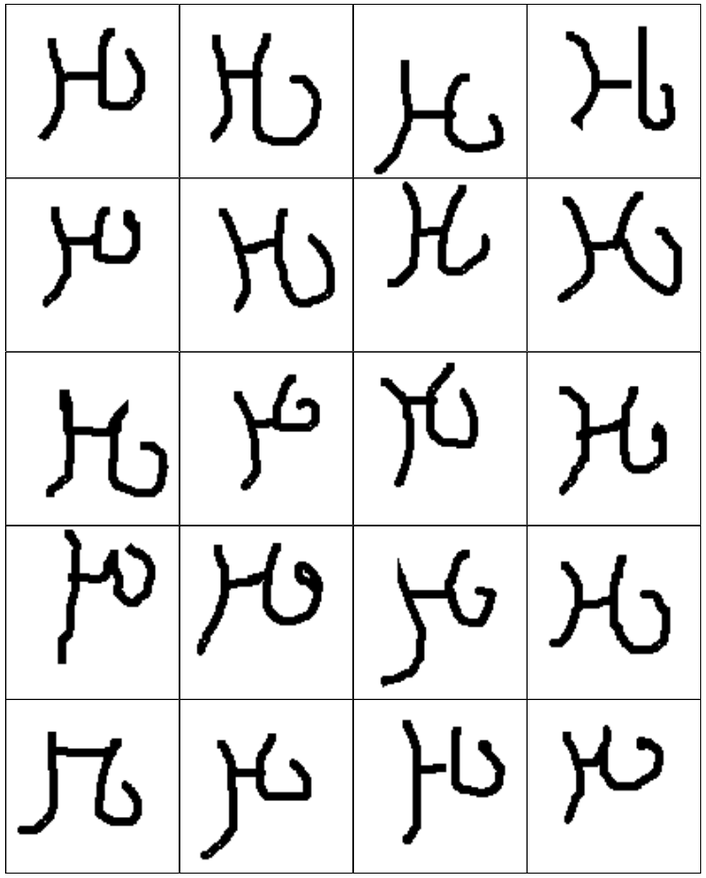




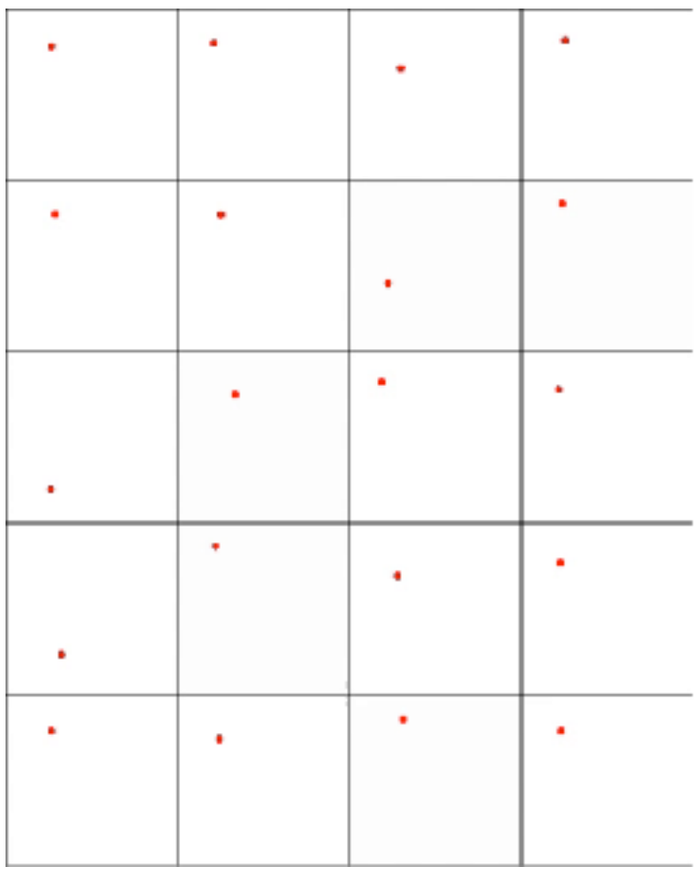
generating
new examples



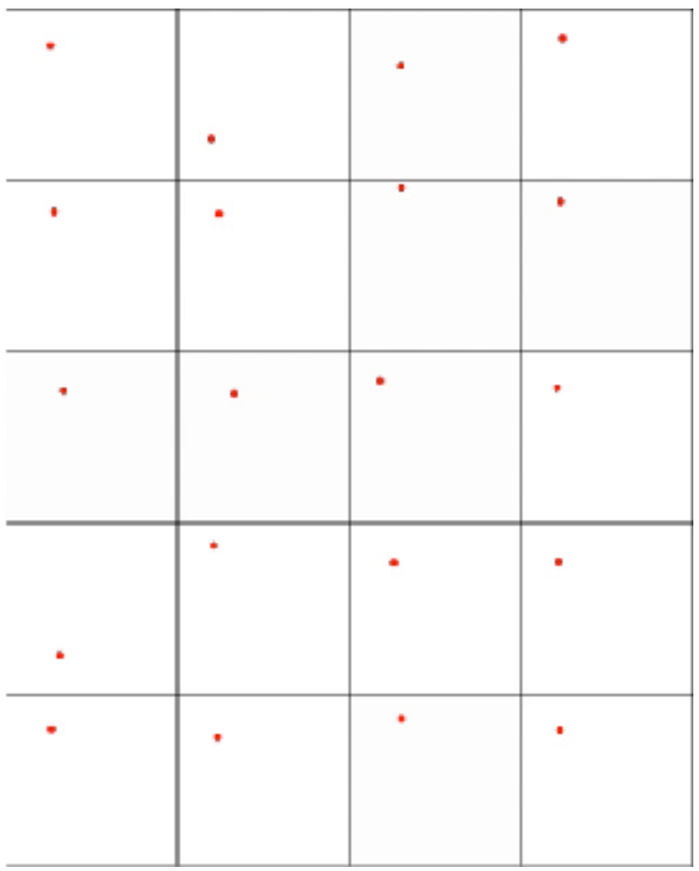
Human drawings



Human parses

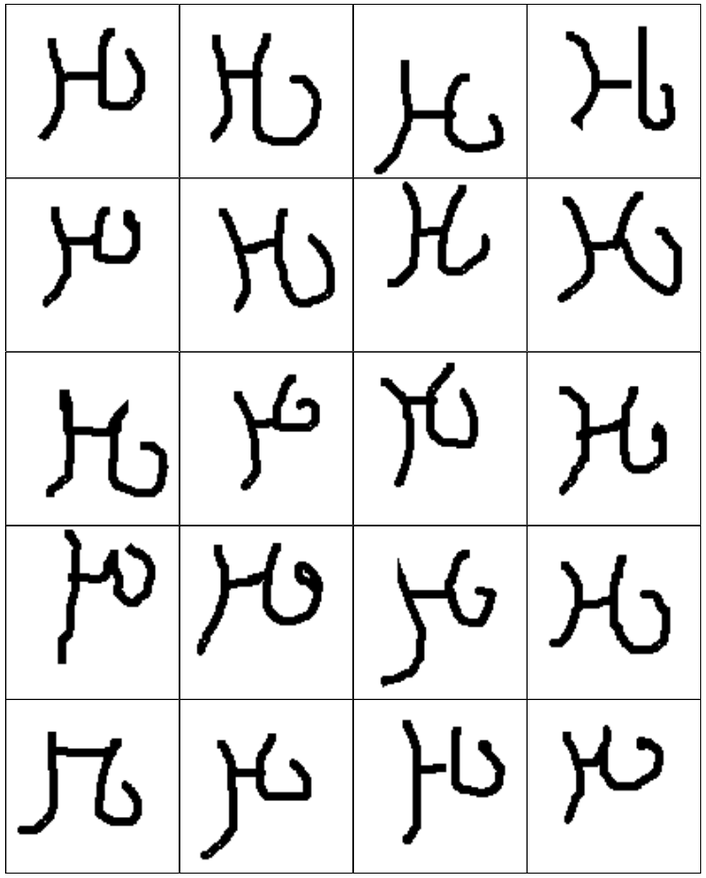


Machine parses

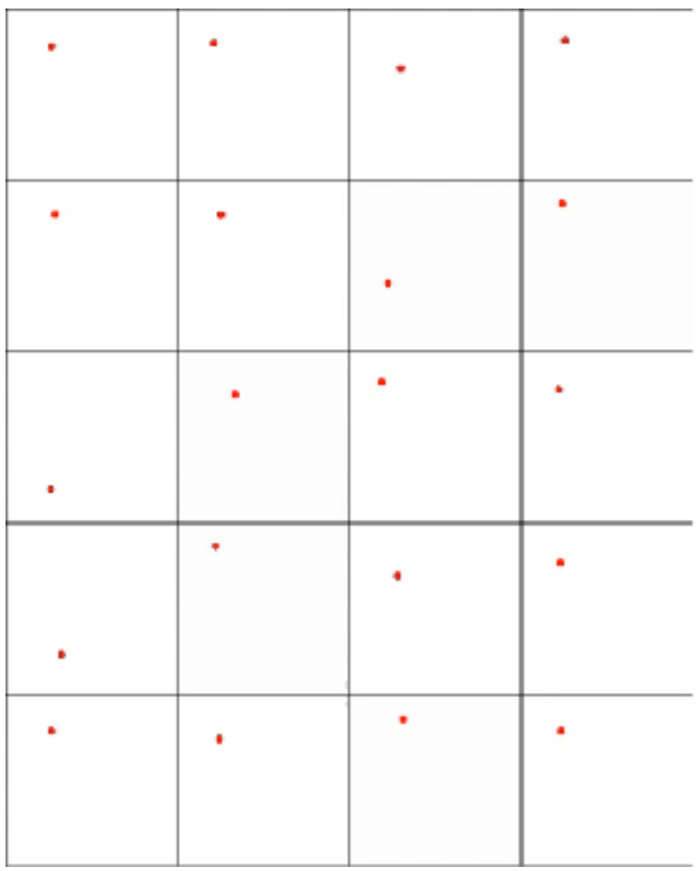


stroke order: 1 2 3 4 5

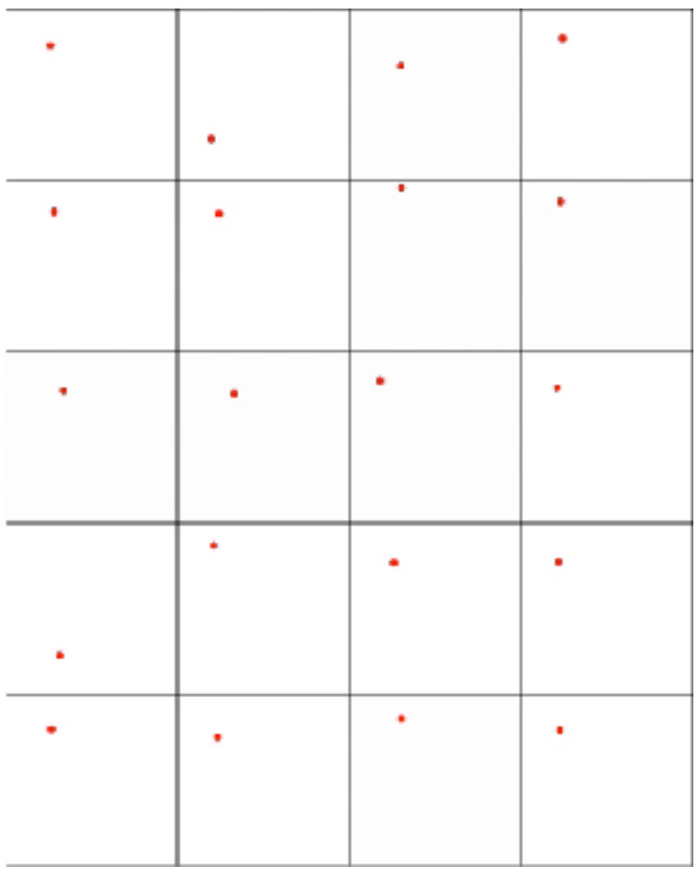
Human drawings



Human parses

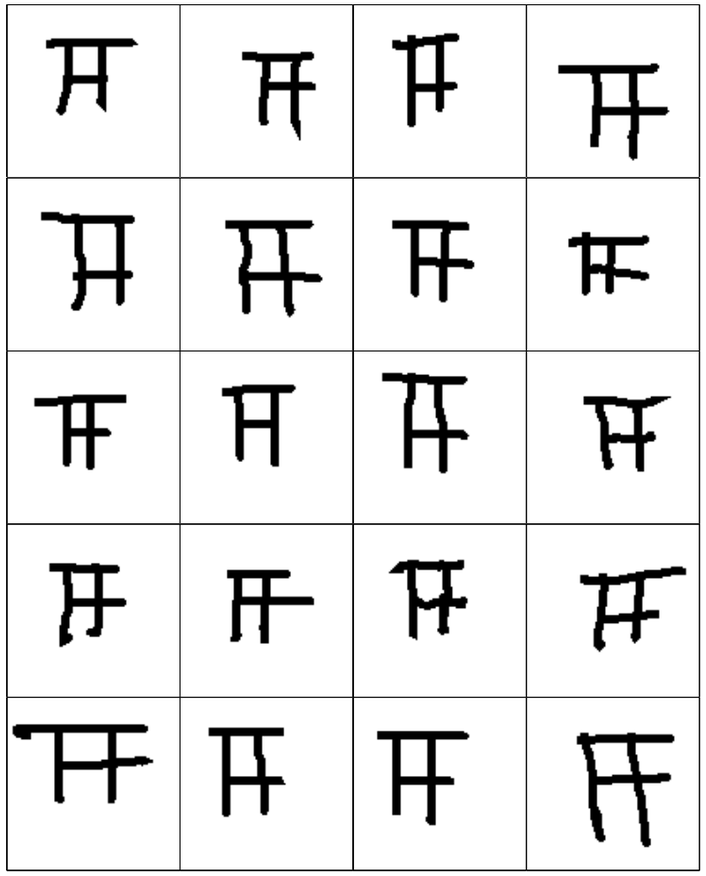


Machine parses

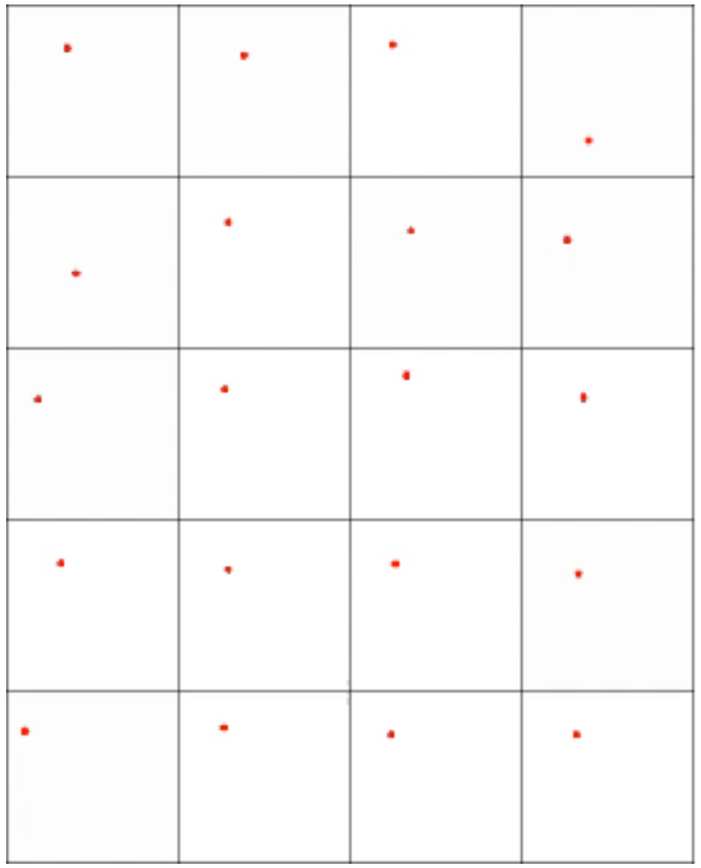


stroke order: 1 2 3 4 5

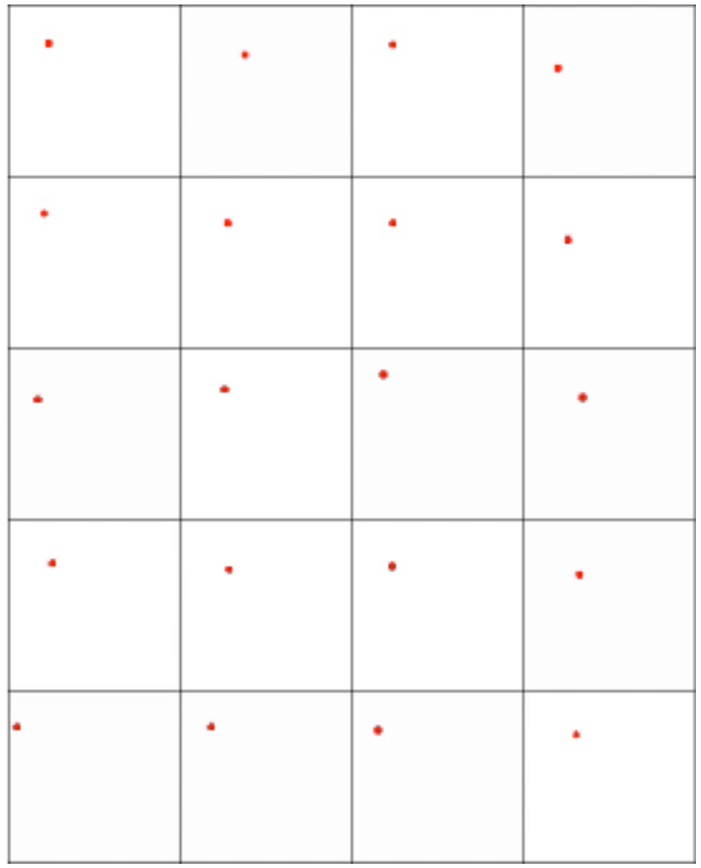
Human drawings



Human parses

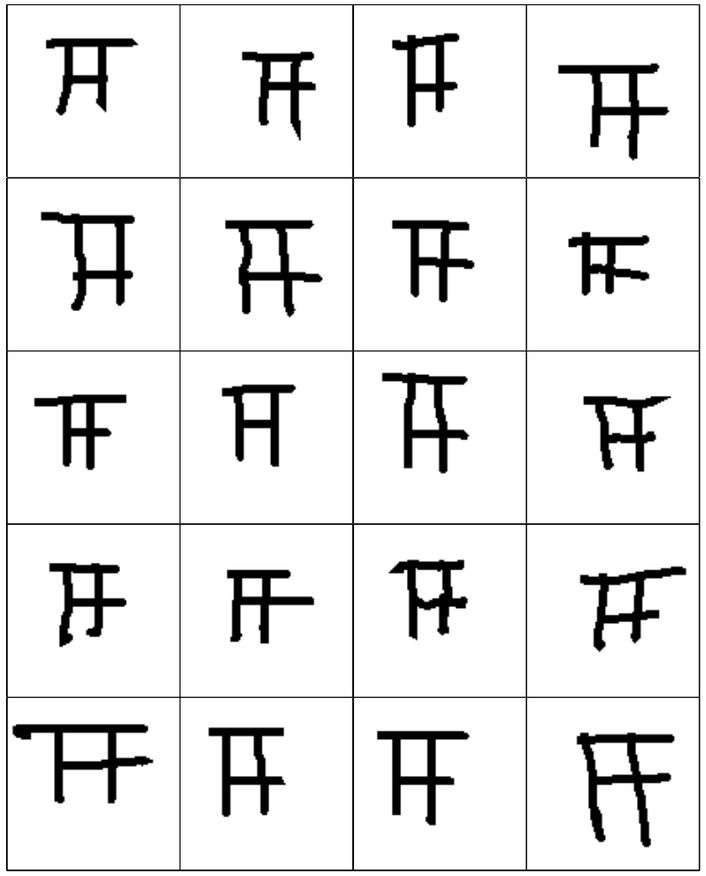


Machine parses

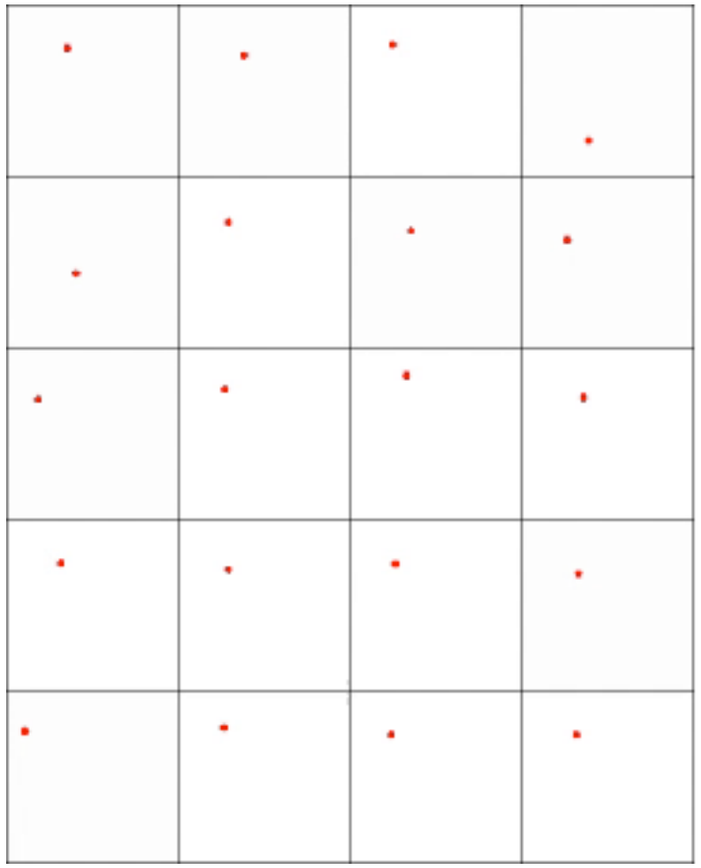


stroke order: 1 2 3 4 5

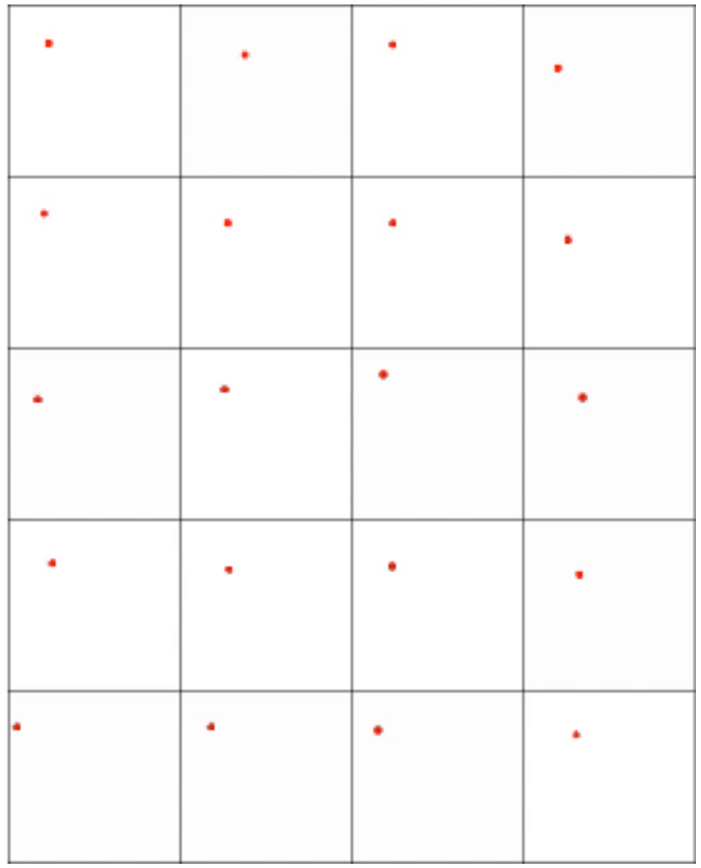
Human drawings



Human parses

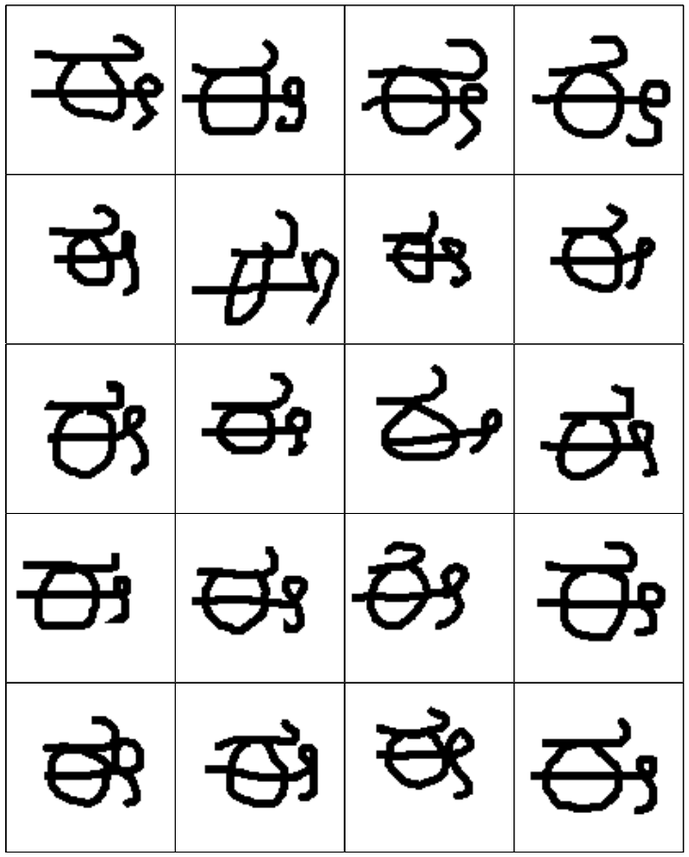


Machine parses

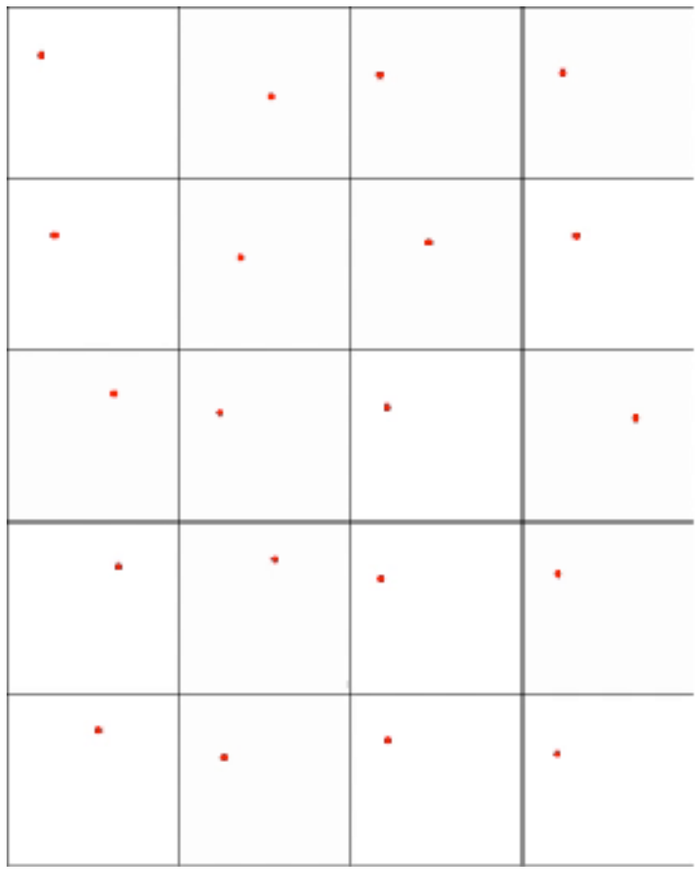


stroke order: 1 2 3 4 5

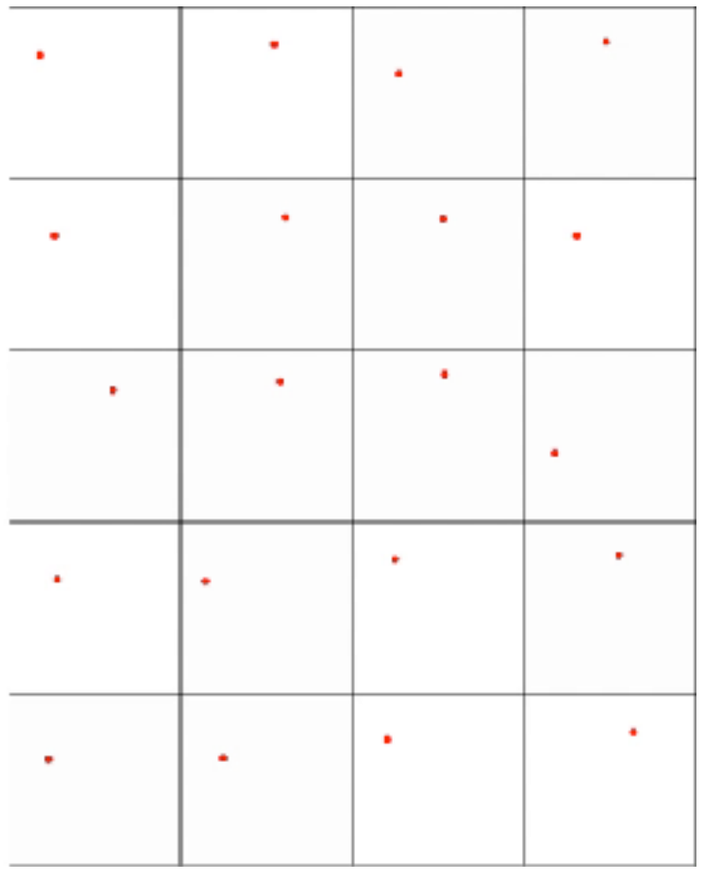
Human drawings



Human parses

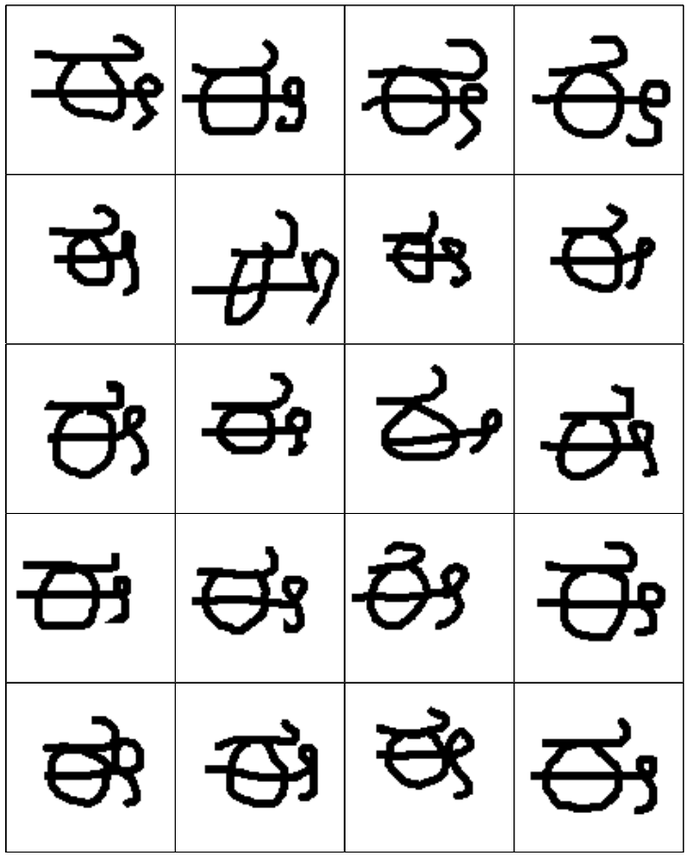


Machine parses

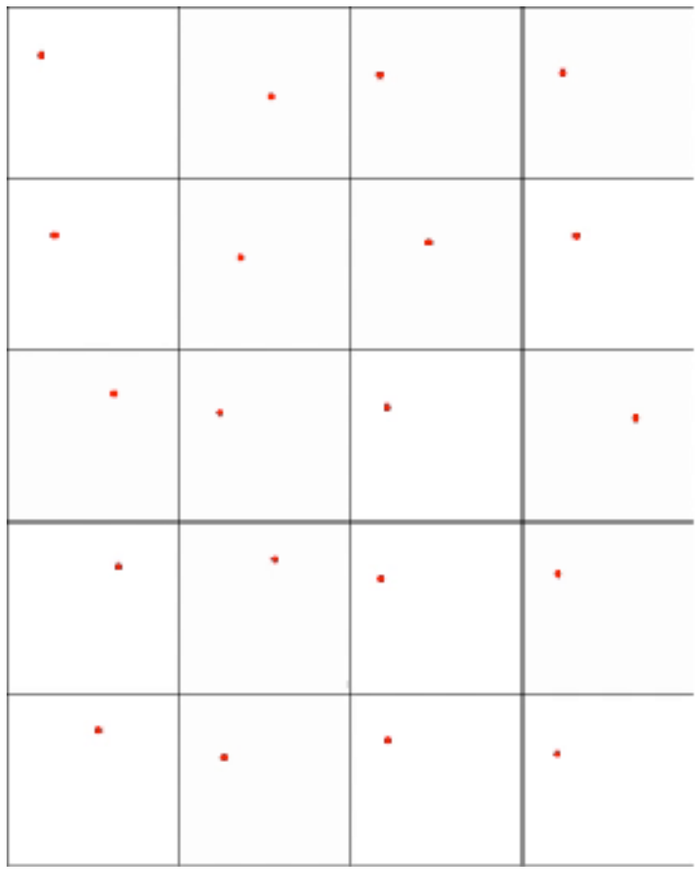


stroke order: 1 2 3 4 5

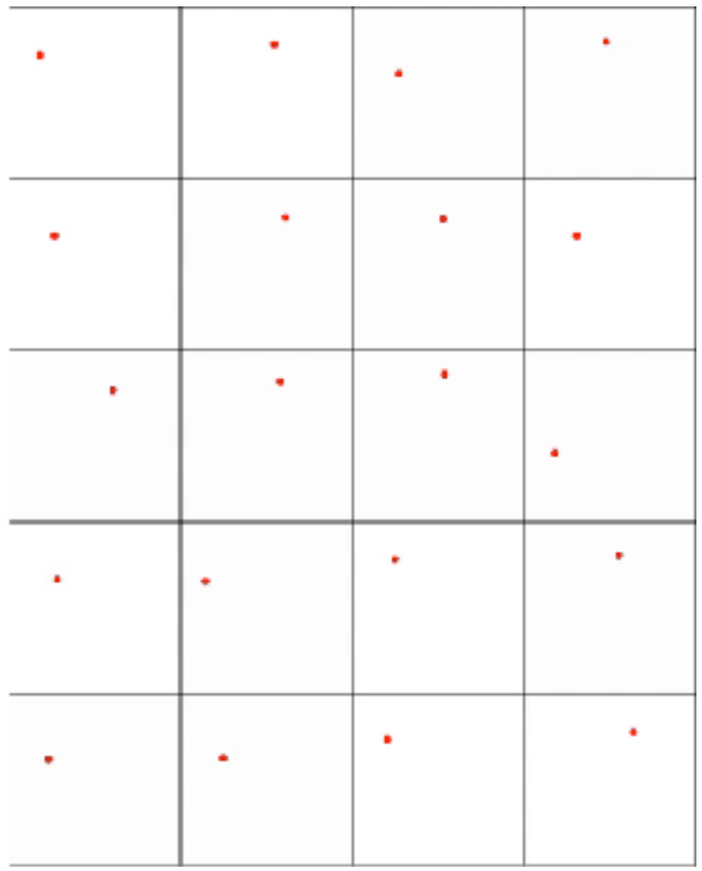
Human drawings



Human parses



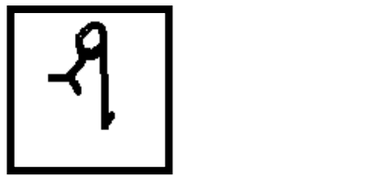
Machine parses



stroke order: 1 2 3 4 5

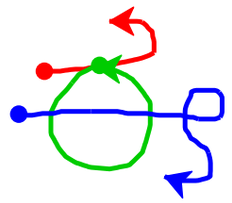
Human-level concept learning

the speed of learning

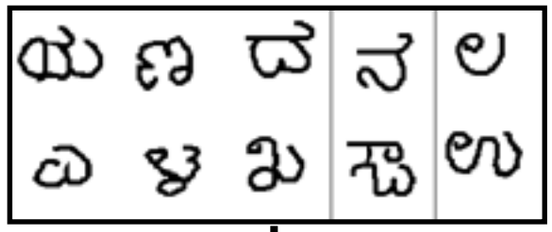


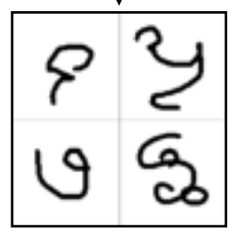
the richness of representation

parsing

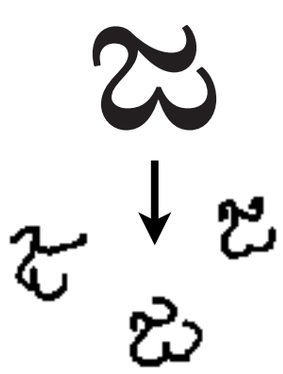


generating
new concepts

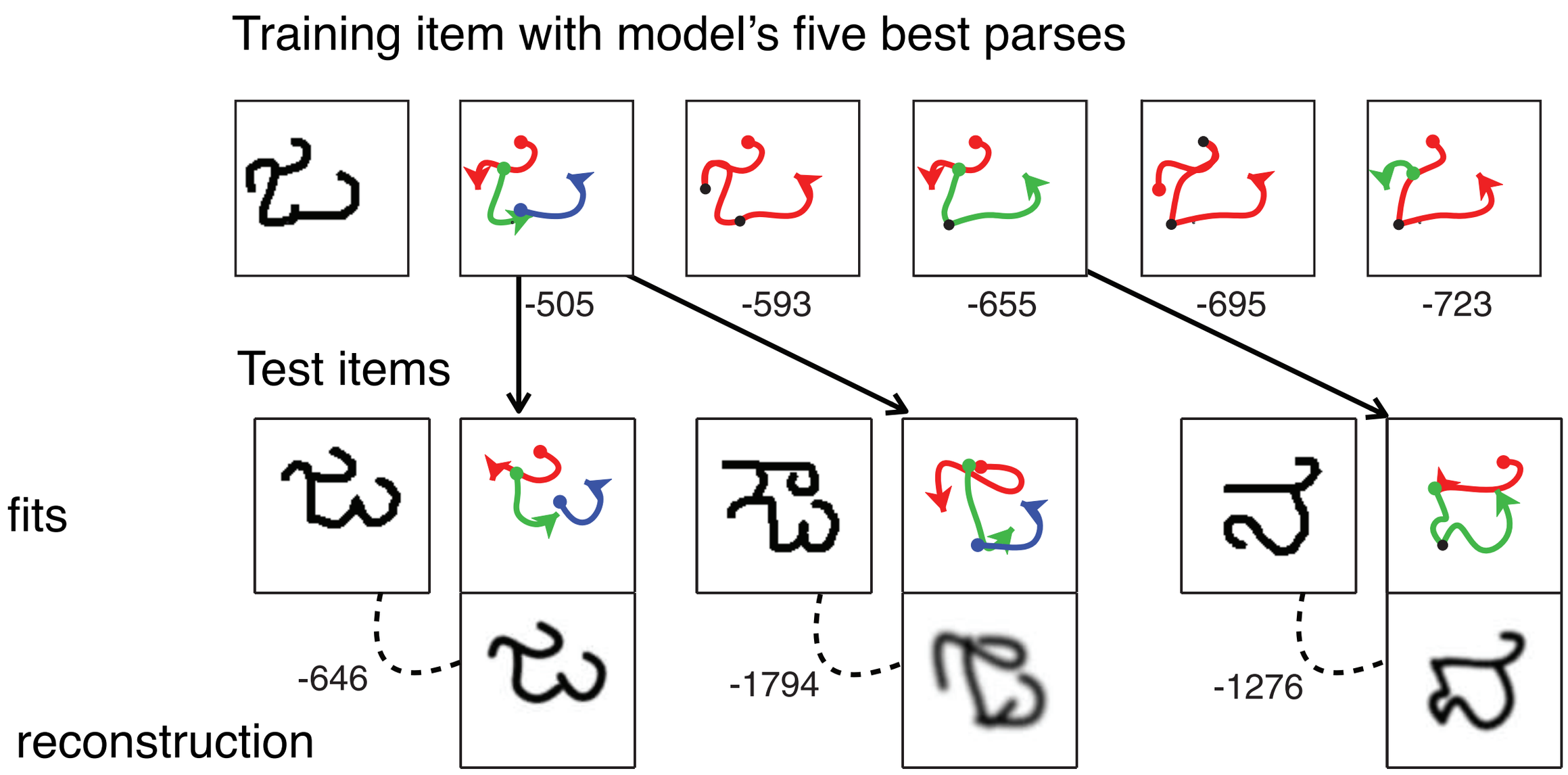




generating
new examples



One-shot classification

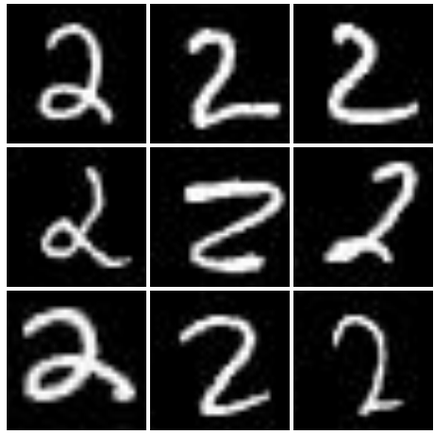


$$P(I_{new} \mid I_{old})$$

Do people represent *static characters* by their causal dynamics?

same causal process

different examples



(Figure credit: Hinton & Nair, 2006)

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)

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)

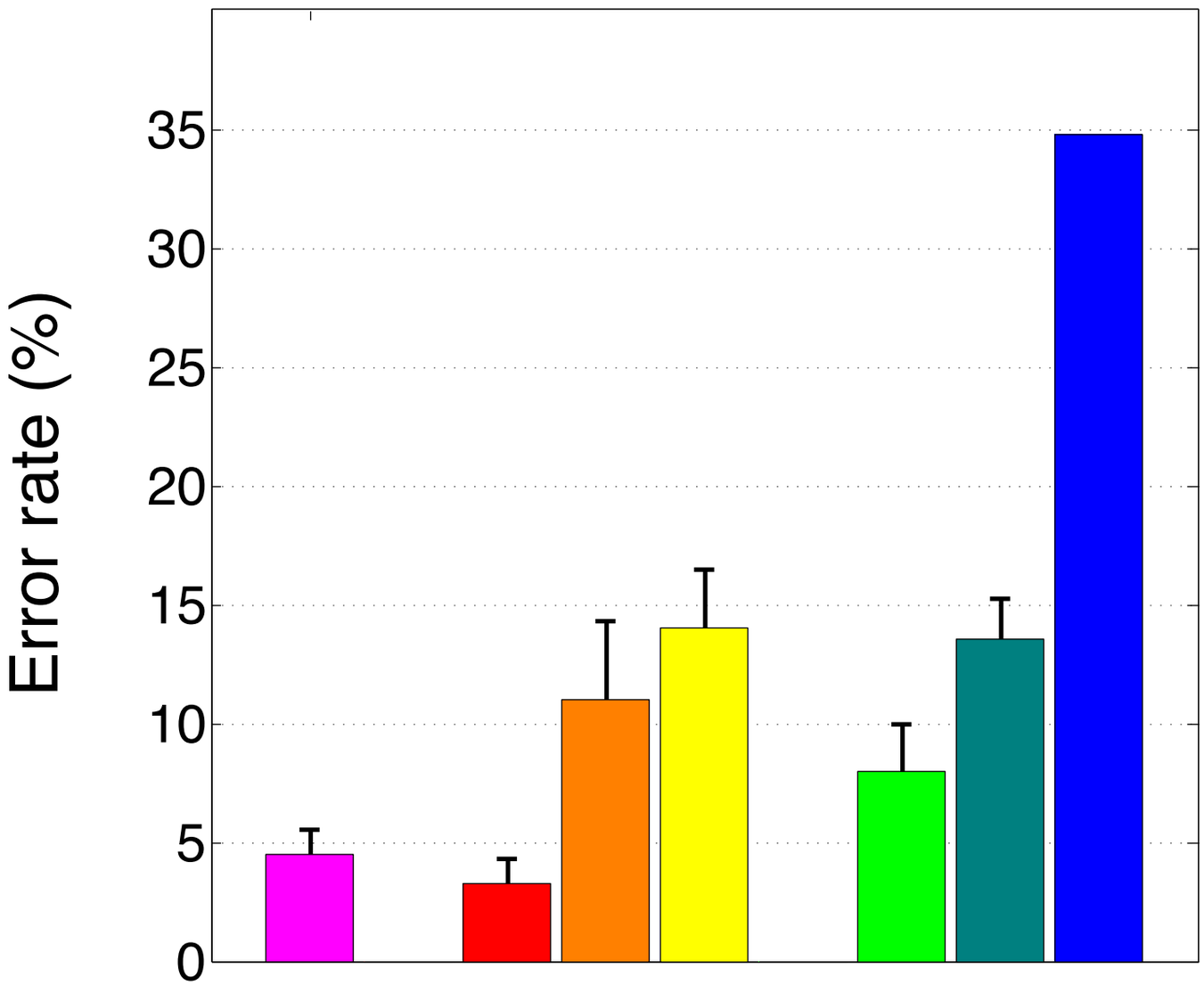
Deep neural networks

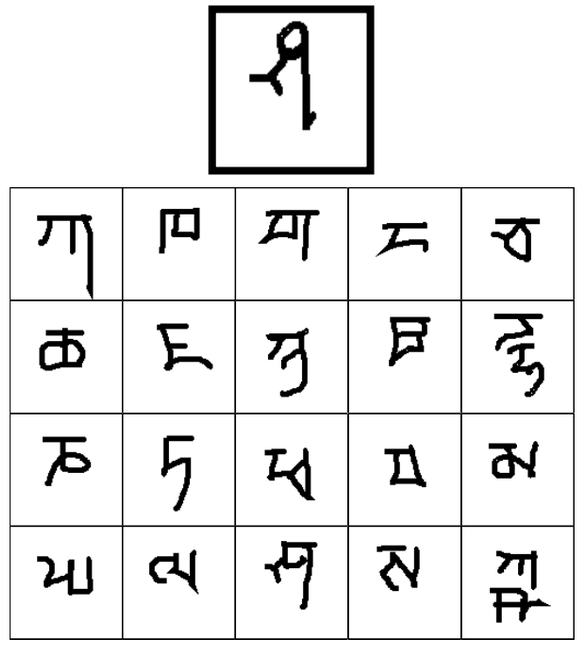
(no causality)

Deep Siamese Convnet
(Koch, Zemel, Salakhutdinov. 2015)

Deep Convnet

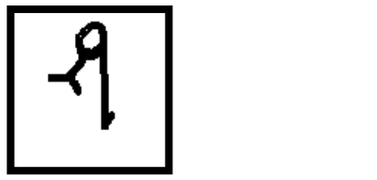
Hierarchical Deep

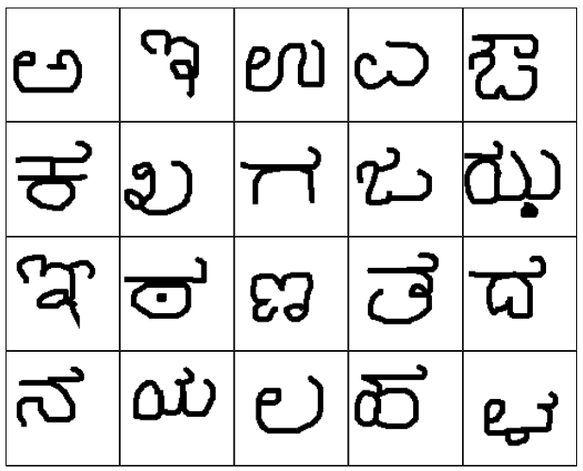




Human-level concept learning

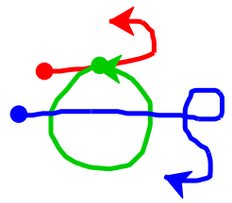
the speed of learning



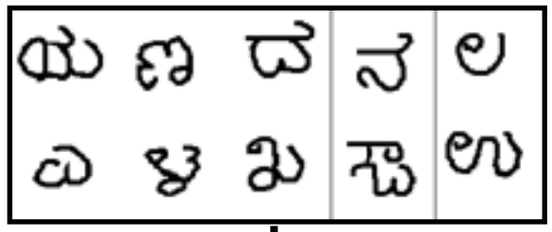


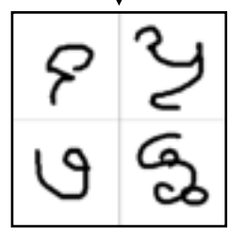
the richness of representation

parsing

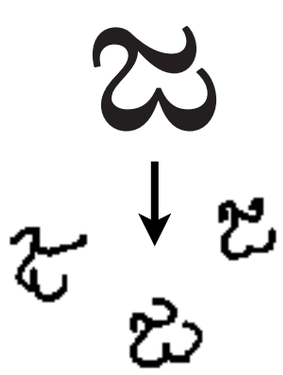


generating new concepts





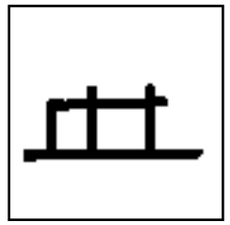
generating new examples



A “visual Turing test” for generating new concepts

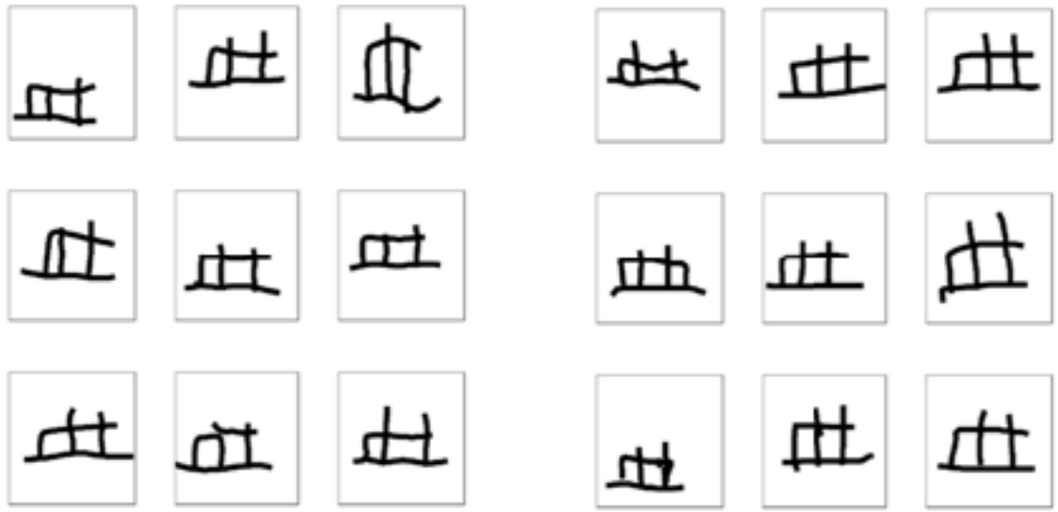
A

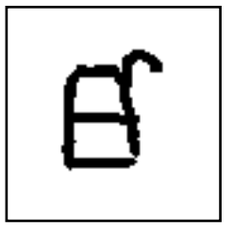
B



A

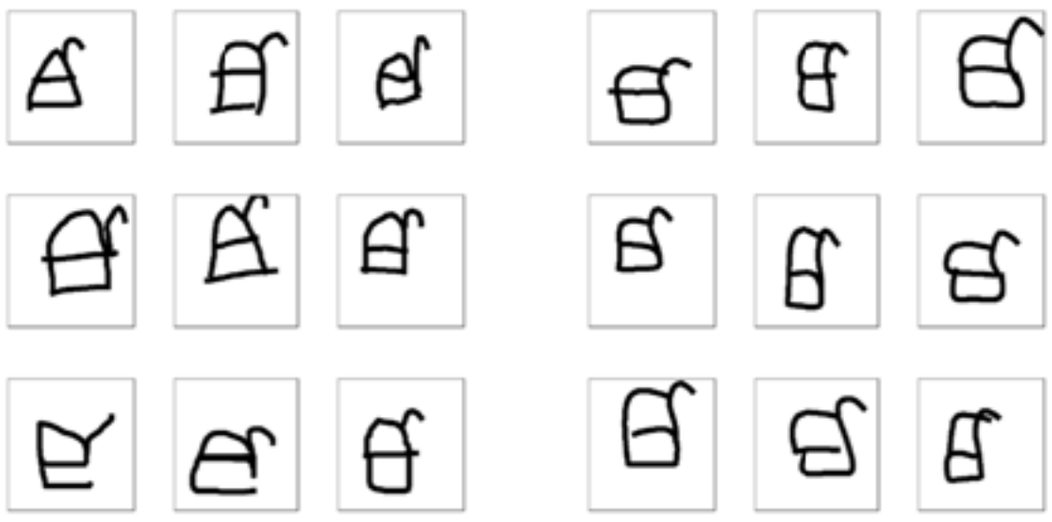
B





A

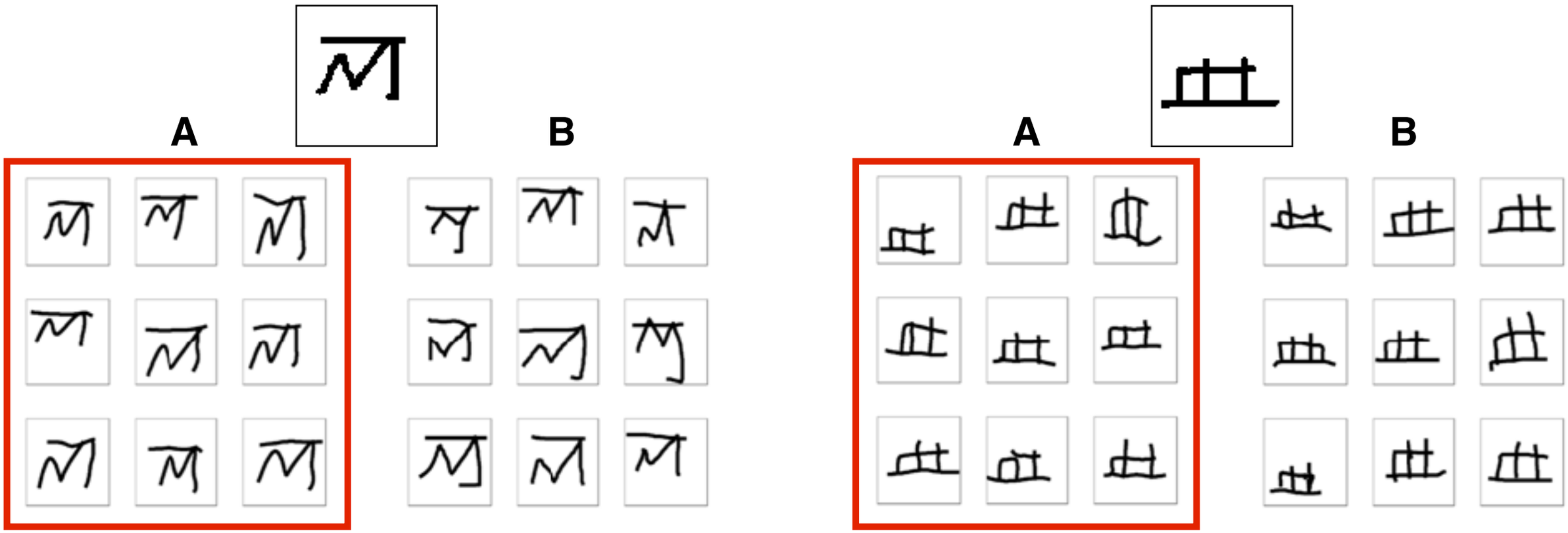
B

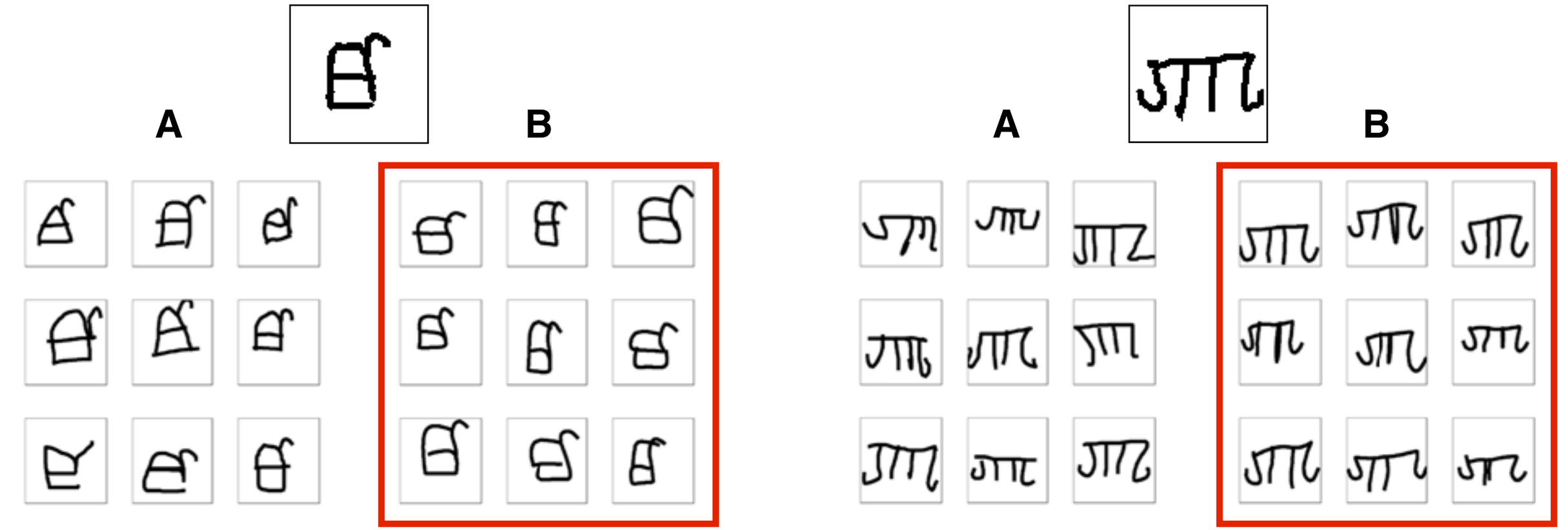


A

B

A “visual Turing test” for generating new concepts

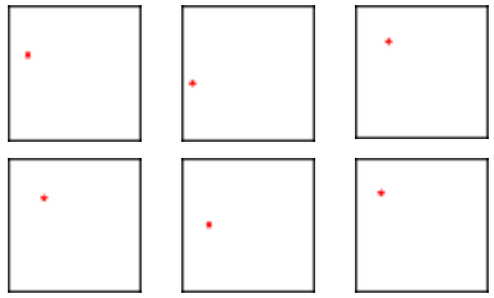




More large-scale behavioral experiments

Generating new examples (dynamic)

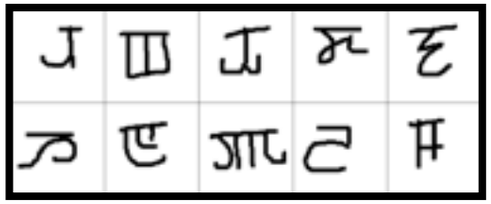
Human or Machine?



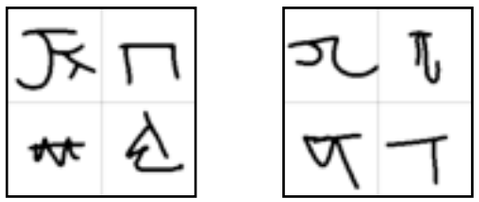
59% correct in visual Turing test
6 of 30 judges above chance

Generating new concepts (from type)

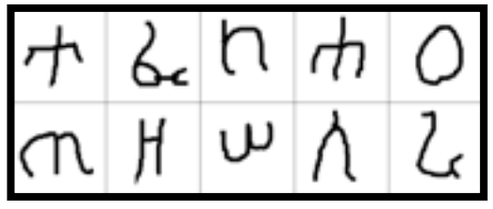
Alphabet



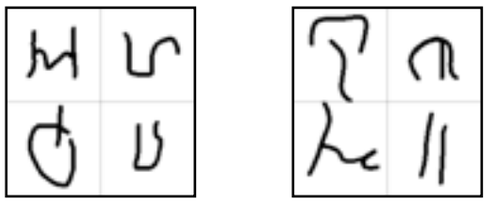
Human or Machine?



Alphabet



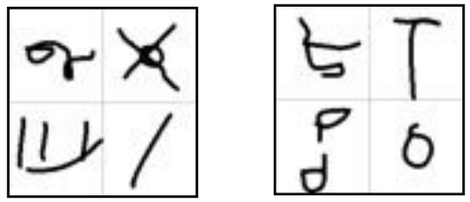
Human or Machine?



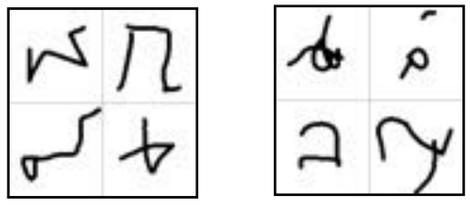
49% correct in visual Turing test
8 of 35 judges above chance

Generating new concepts (unconstrained)

Human or Machine?



Human or Machine?

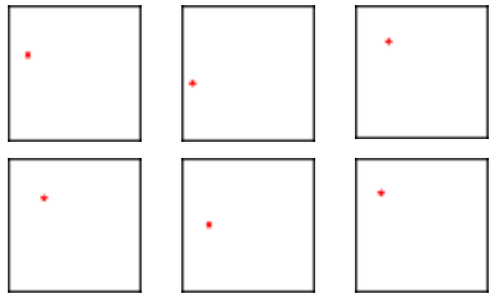


51% correct in visual Turing test
2 of 25 judges above chance

More large-scale behavioral experiments

Generating new examples (dynamic)

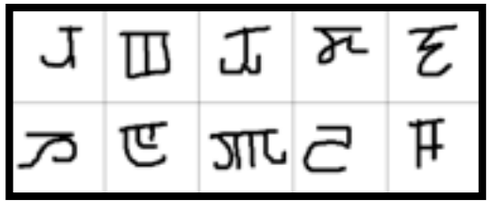
Human or Machine?



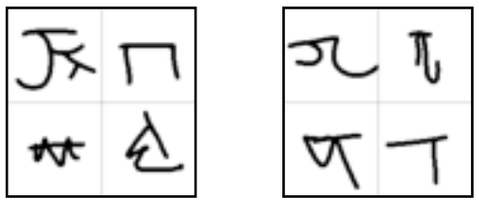
59% correct in visual Turing test
6 of 30 judges above chance

Generating new concepts (from type)

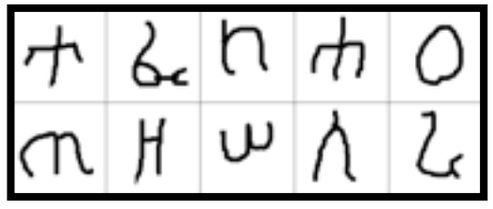
Alphabet



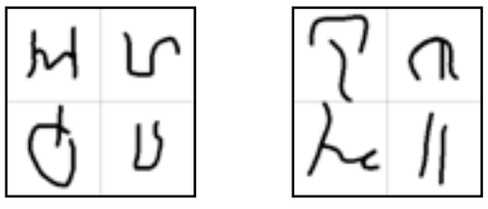
Human or Machine?



Alphabet



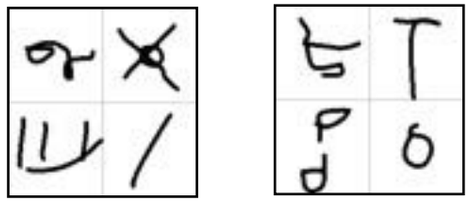
Human or Machine?



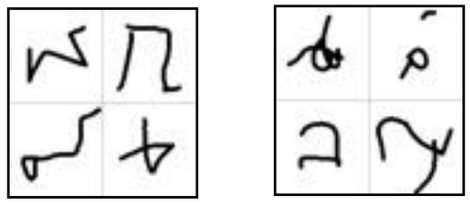
49% correct in visual Turing test
8 of 35 judges above chance

Generating new concepts (unconstrained)

Human or Machine?



Human or Machine?



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

Case study: Numerical concept learning and cognitive development

Cognition 123 (2012) 199–217

Contents lists available at SciVerse ScienceDirect

Cognition

journal homepage: www.elsevier.com/locate/COGNIT

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

ARTICLE INFO

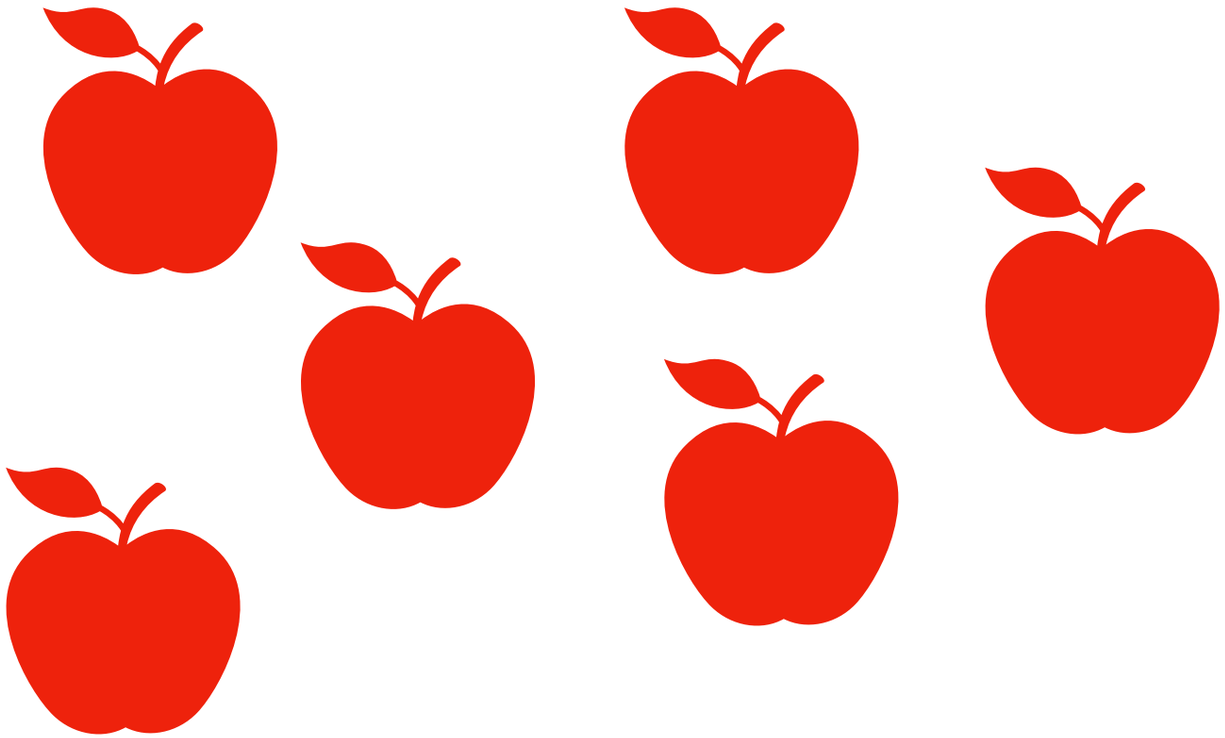
Article history:
Available online 26 January 2012

Keywords:
Number word learning
Bootstrapping
Cognitive development
Bayesian model
Language of thought
CP transition

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

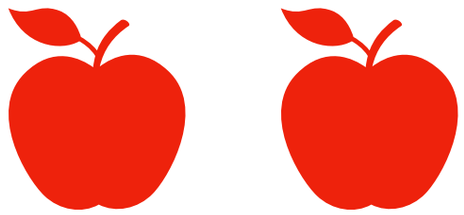
Children progress through a series of stages

- "one-knower", "two-knower," "three-known," "four-knower" (sometimes), and then "cardinal-principle knower"

Example: "two knower"

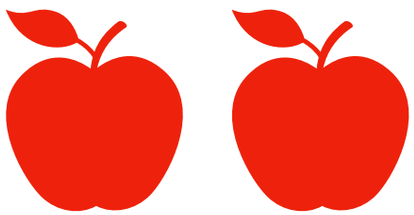
give me one:

give me two:

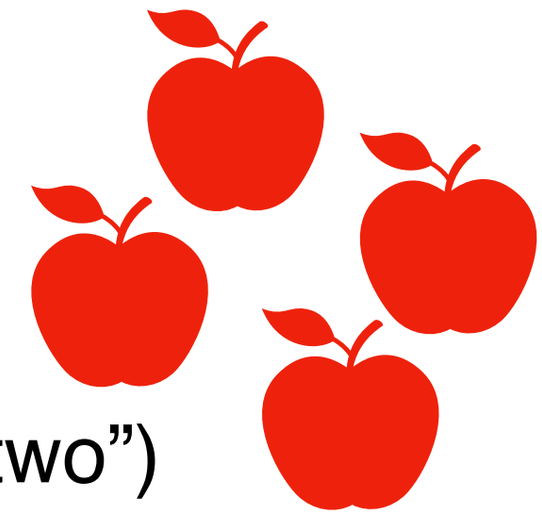


give me ~~three~~:

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

	Success pattern					Counting ability			
	1	2	3	5	6	Number of children	Mean age	Mean	(range)
	−	−	−	−	−	1	2:8	3.00	(3-3)
one-knower	+	−	−	−	−	3	3:0	4.67	(3-6)
two-knower	+	+	−	−	−	2	2:11	4.50	(3-6)
three-knower	+	+	+	−	−	4	3:5	5.75	(5-6)
CP-knower	+	+	+	+	+	7	3:7	6.00	(6-6)

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

(data from Wynn, 1990)

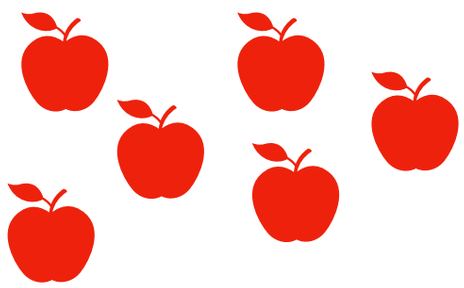
Programming primitives allowed in the language of thought model

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 se

(Piantadosi, Tenenbaum & Goodman)

Example hypotheses in a language of thought

example set *S*



$\lambda\ S\ .$

indicates a function that takes a set *S* as an argument.

One-knower

$$\lambda\, S\ .\ (\mathit{if}\ (\mathit{singleton?}\ S)\ \text{“}\mathit{one}\text{”}\ \mathit{undef})$$

Two-knower

$$\lambda\, S\ .\ (\mathit{if}\ (\mathit{singleton?}\ S)\ \text{“}\mathit{one}\text{”}\ (\mathit{if}\ (\mathit{doubleton?}\ S)\ \text{“}\mathit{two}\text{”}\ \mathit{undef}))$$

Three-knower

$$\lambda\, S\ .\ (\mathit{if}\ (\mathit{singleton?}\ S)\ \text{“}\mathit{one}\text{”}\ (\mathit{if}\ (\mathit{doubleton?}\ S)\ \text{“}\mathit{two}\text{”}\ (\mathit{if}\ (\mathit{tripleton?}\ S)\ \text{“}\mathit{three}\text{”}\ \mathit{undef}))$$

CP-knower

$$\lambda\, S\ .\ (\mathit{if}\ (\mathit{singleton?}\ S)\ \text{“}\mathit{one}\text{”}\ (\mathit{next}\ (L\ (\mathit{set\text{-}difference}\ S\ (\mathit{select}\ S)))))$$

Singular-Plural

$$\lambda\, S\ .\ (\mathit{if}\ (\mathit{singleton?}\ S)\ \text{“}\mathit{one}\text{”}\ \text{“}\mathit{two}\text{”})$$

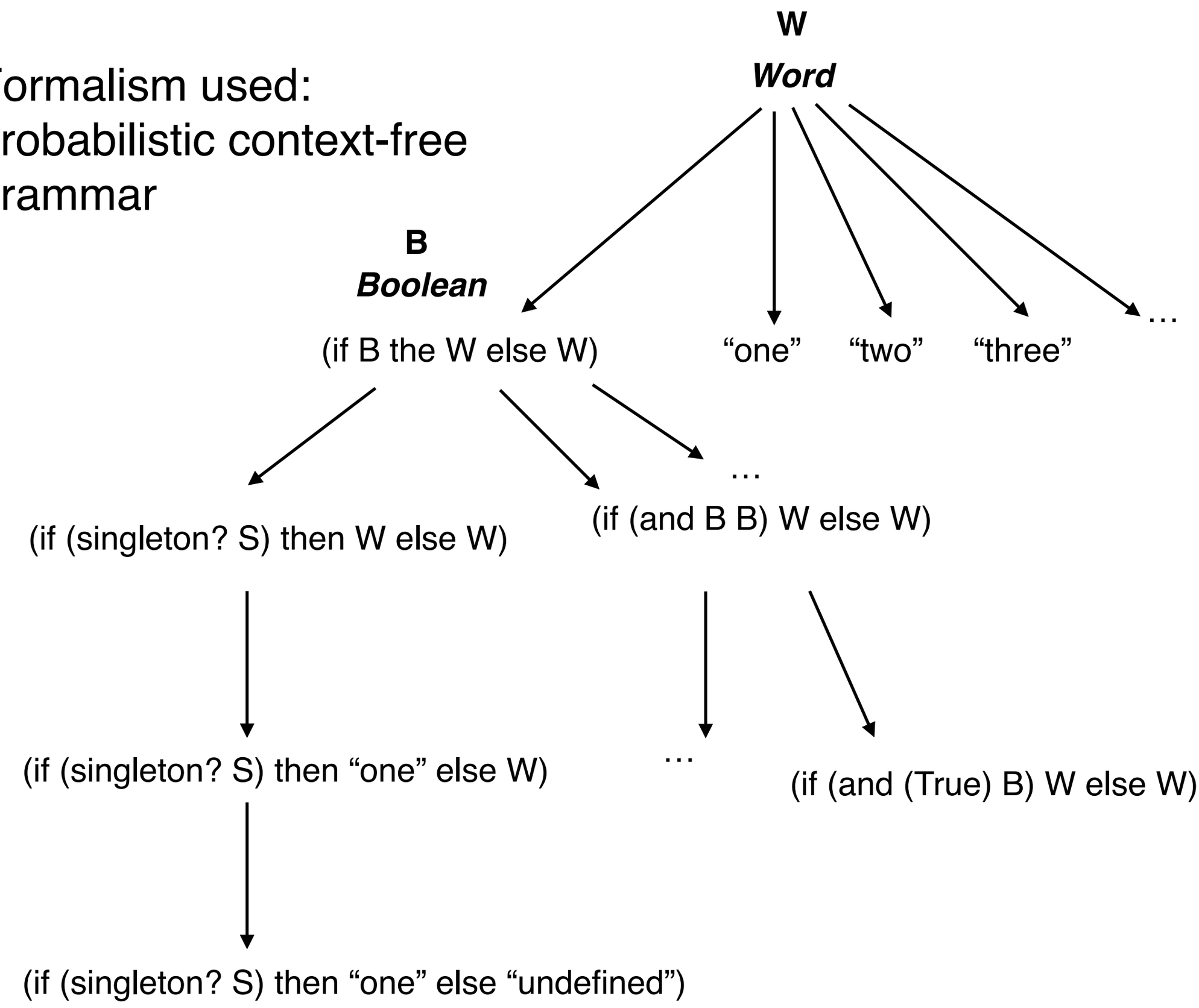
Mod-5

$$\lambda\, S\ .\ (\mathit{if}\ (\mathit{or}\ (\mathit{singleton?}\ S)\ (\mathit{equal\text{-}word?}\ (L\ (\mathit{set\text{-}difference}\ S)\ (\mathit{select}\ S))\ \text{“}\mathit{five}\text{”}))\ \text{“}\mathit{one}\text{”}\ (\mathit{next}\ (L\ (\mathit{set\text{-}difference}\ S\ (\mathit{select}\ S)))))$$

Defining a prior distribution over programs

(a “Probabilistic Language of Thought”)

Formalism used:
probabilistic context-free
grammar



Probabilistic model over programs

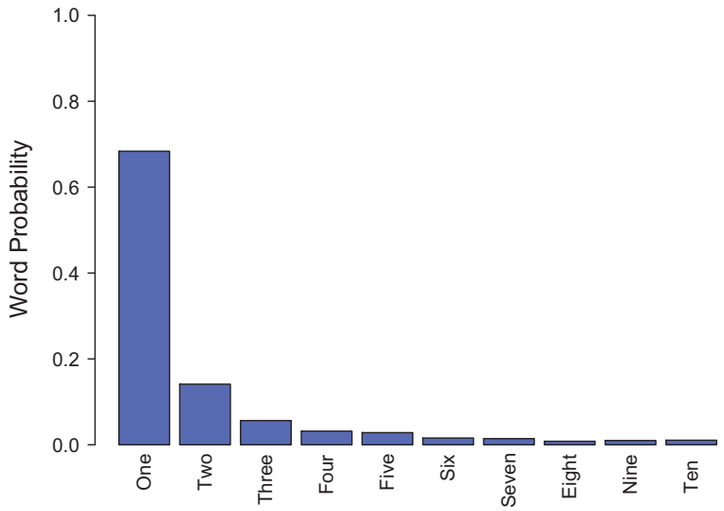
Data (*D*)

“six”

“two”

“one”

“one”



Example *L*

$$\lambda\, S\, .\, (if\ (singleton?\ S)$$
$$\text{“}one\text{”}$$
$$undef)$$

Probabilistic model

$P(L)$ prior on programs *L* (defined with probabilistic grammar)

$P(D|L)$ Noisy likelihood where right number is usually returned, but with some noise

Bayes’ rule for learning programs

$$P(L|D) = \frac{P(D|L)P(L)}{P(D)}$$

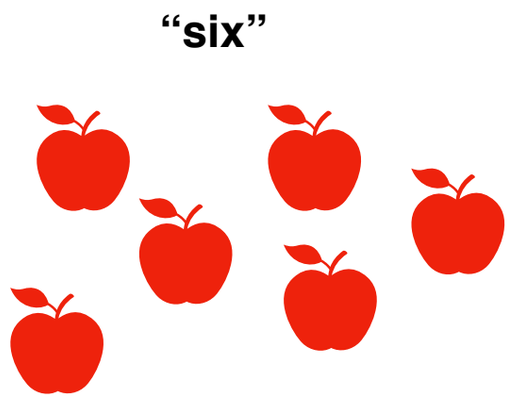
Learning as program induction

Program (*L*)

$$\lambda\, S\, .\, (if\ (singleton?\ S)$$
$$\text{“}one\text{”}$$
$$(next\ (L\ (set\text{-}difference\ S$$
$$(select\ S)))))$$

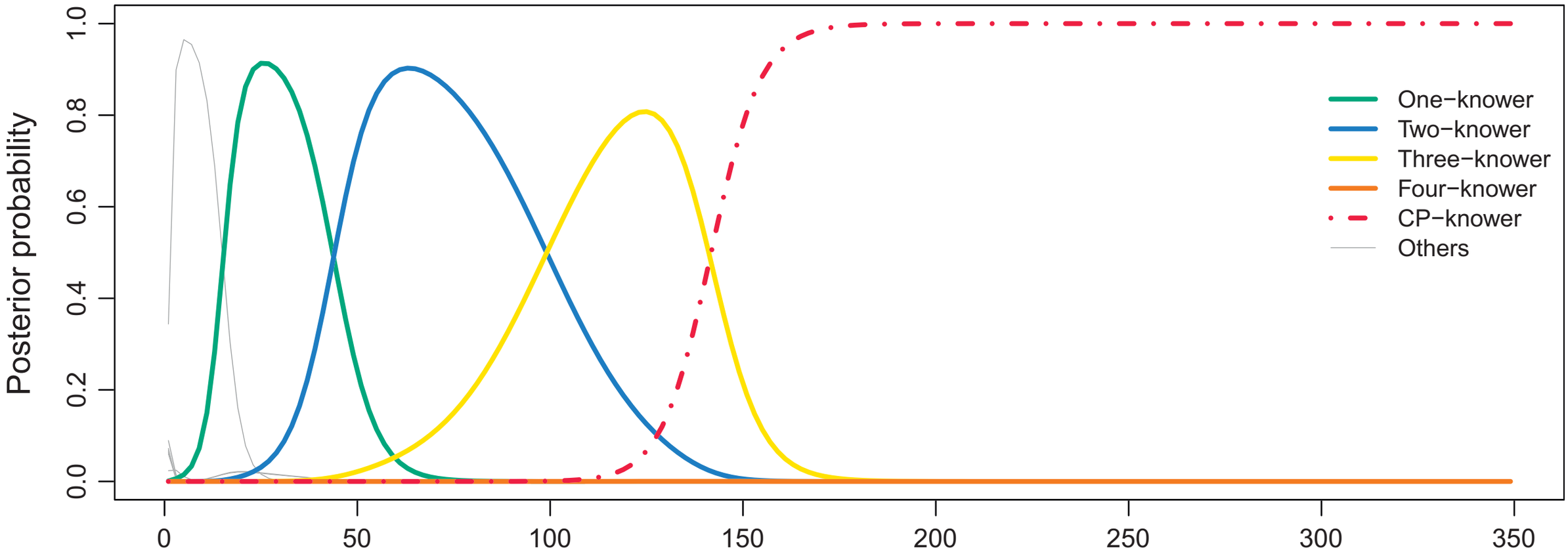
$$P(L|D) = \frac{P(D|L)P(L)}{P(D)}$$

Data (*D*)



inference

Results: Program induction model follows a similar developmental trajectory



One-knower

λ *S* . (*if* (*singleton? S*)
“one”
undef)

Two-knower

λ *S* . (*if* (*singleton? S*)
“one”
(*if* (*doubleton? S*)
“two”
undef))

Three-knower

λ *S* . (*if* (*singleton? S*)
“one”
(*if* (*doubleton? S*)
“two”
(*if* (*tripleton? S*)
“three”
undef))

CP-knower

λ *S* . (*if* (*singleton? S*)
“one”
(*next* (*L* (*set-difference S*
(*select S*)))))

Case study: Learning by asking questions

Question Asking as Program Generation

Anselm Rothe[1] anselm@nyu.edu
Brenden M. Lake[1,2] brenden@nyu.edu
Todd M. Gureckis[1] todd.gureckis@nyu.edu

[1]Department of Psychology [2]Center for Data Science
New York University

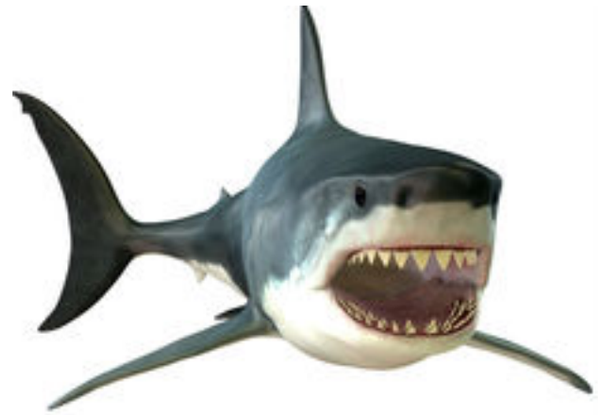
Abstract

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

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 the category label of this object?

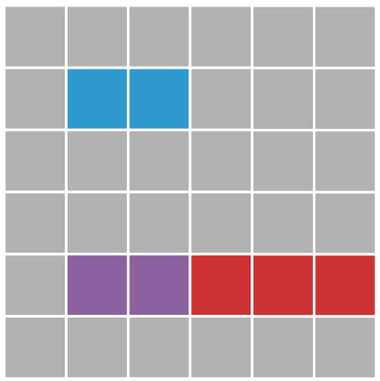
What is the the category label of this object?

What is the the category label of this object?

Experiment: Free-form question asking

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

Hidden configuration of ships

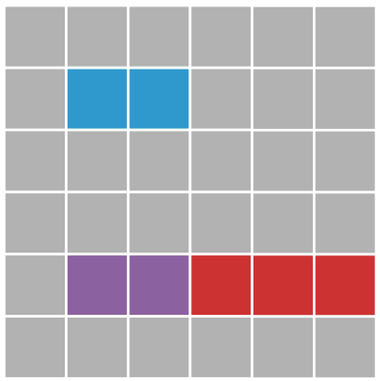


3 ships (blue, purple, red)
3 possible sizes (2-4 tiles)
1.6 million possible configurations

Experiment: Free-form question asking

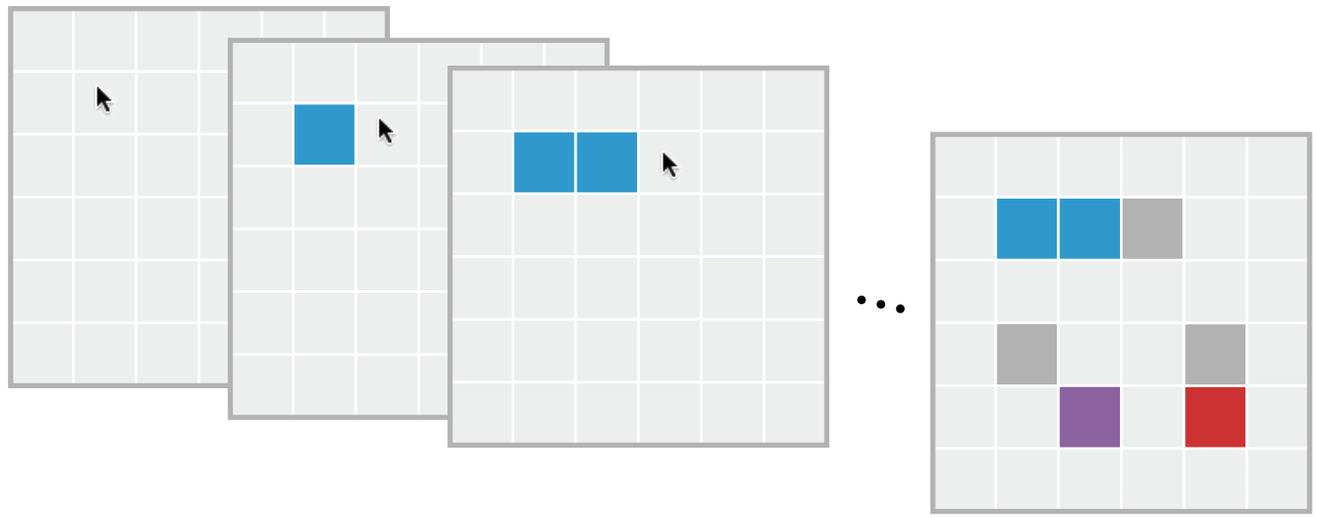
(e.g., Markant & Gureckis, 2012, 2014)

Hidden configuration of ships



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

Phase 1: Sampling



Phase 2: Question asking

Is the red ship horizontal?

Constraints
- one word answers
- no combinations

Repeated for 18 different hidden configurations

N	Location/standard queries
24	What color is at [row][column]?
24	Is there a ship at [row][column]?
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?

N	Location/standard queries
24	What color is at [row][column]?
24	Is there a ship at [row][column]?
31	Is there a [color_incl_water] tile at [row][column]?
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?

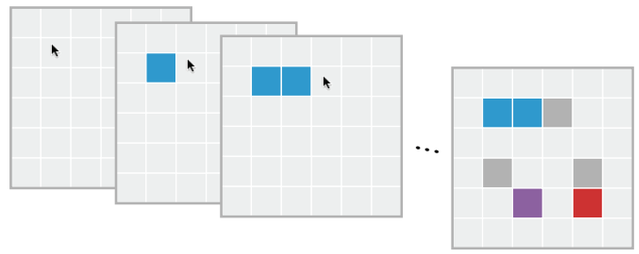
N	Location/standard queries
24	What color is at [row][column]?
24	Is there a ship at [row][column]?
31	Is there a [color_incl_water] tile at [row][column]?
	Region queries
4	Is there any ship in row [row]?
9	Is there any part of the [color] ship in row [row]?
5	How many tiles in row [row] are occupied 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]?
	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?
	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?

N	Location/standard queries
24	What color is at [row][column]?
24	Is there a ship at [row][column]?
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

	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?

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 )
)
```

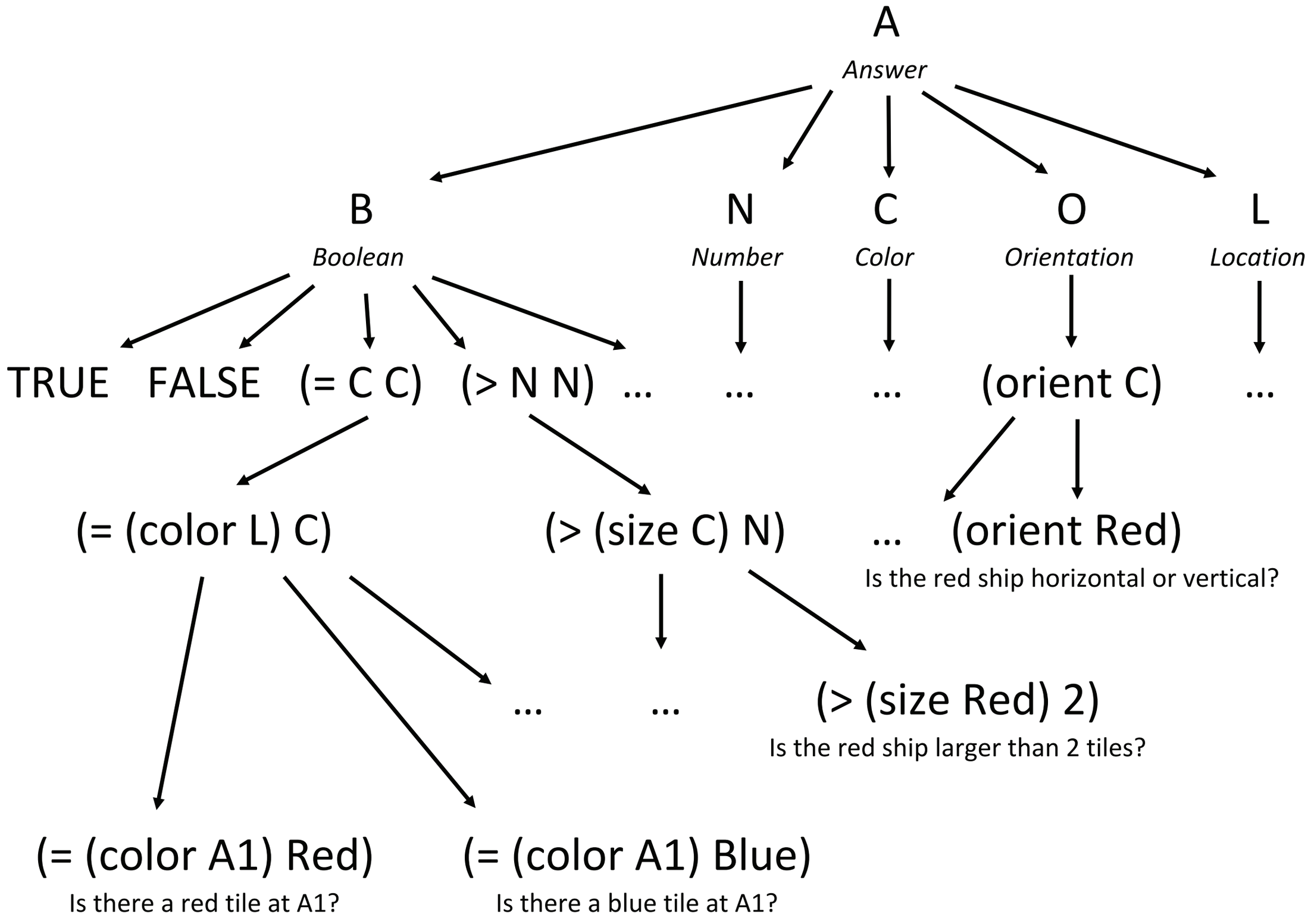
“Are the blue ship and the red ship **parallel**?”

```
( =
    ( orient Blue )
    ( orient Red )
)
```

Questions as programs

GROUP	QUESTION	FUNCTION	EXPRESSION
location	What color is at A1?	location	(color A1)
	Is there a ship at A1?	locationA	(not (= (color A1) Water))
	Is there a blue tile at A1?	locationD	(= (color A1) Blue)
segmentation	Is there any ship in row 1?	row	(> (+ (map (λ x (and (= (row x) 1) (not (= (color x) Water)))) (set A1 ... F6))) 0)
	Is there any part of the blue ship in row 1?	rowD	(> (+ (map (λ x (and (= (row x) 1) (= (color x) Blue))) (set A1 ... F6))) 0)
	Are all parts of the blue ship in row 1?	rowDL	(> (+ (map (λ x (and (= (row x) 1) (= (color x) Blue))) (set A1 ... F6))) 1)
	How many tiles in row 1 are occupied by ships?	rowNA	(+ (map (λ x (and (= (row x) 1) (not (= (color x) Water)))) (set A1 ... F6)))
	Are there any ships in the bottom half of the grid?	rowX2	...
	Is there any ship in column 1?	col	(> (+ (map (λ x (and (= (col x) 1) (not (= (color x) Water)))) (set A1 ... F6))) 0)
	Is there any part of the blue ship in column 1?	colD	(> (+ (map (λ x (and (= (col x) 1) (= (color x) Blue))) (set A1 ... F6))) 0)
	Are all parts of the blue ship in column 1?	colDL	(> (+ (map (λ x (and (= (col x) 1) (= (color x) Blue))) (set A1 ... F6))) 1)
	How many tiles in column 1 are occupied by ships?	colNA	(+ (map (λ x (and (= (col x) 1) (not (= (color x) Water)))) (set A1 ... F6)))
	Is any part of the blue ship in the left half of the grid?	colX1	...
ship size	How many tiles is the blue ship?	shipsize	(size Blue)
	Is the blue ship 3 tiles long?	shipsizeD	(= (size Blue) 3)
	Is the blue ship 3 or more tiles long?	shipsizeM	(or (= (size Blue) 3) (> (size Blue) 3))
	How many ships are 3 tiles long?	shipsizeN	(+ (map (λ x (= (size x) 3)) (set Blue Red Purple)))
	Are any ships 3 tiles long?	shipsizeDA	(> (+ (map (λ x (= (size x) 3)) (set Blue Red Purple))) 0)
	Are all ships 3 tiles long?	shipsizeDL	(= (+ (map (λ x (= (size x) 3)) (set Blue Red Purple))) 3)
	Are all ships the same size?	shipsizeL	(= (map (λ x (size x)) (set Blue Red Purple)))
	Do the blue ship and the red ship have the same size?	shipsizeX1	(= (size Blue) (size Red))
	Is the blue ship longer than the red ship?	shipsizeX2	(> (size Blue) (size Red))
	How many tiles are occupied by ships?	totalshipsize	(+ (map (λ x (size x)) (set Blue Red Purple)))
orientation	Is the blue ship horizontal?	horizontal	(= (orient Blue) H)
	How many ships are horizontal?	horizontalN	(+ (map (λ x (= (orient x) H) (set Blue Red Purple))))
	Are there more horizontal ships than vertical ships?	horizontalM	(> (+ (map (λ x (= (orient x) H) (set Blue Red Purple)))) 1)
	Are all ships horizontal?	horizontalL	(= (+ (map (λ x (= (orient x) H) (set Blue Red Purple)))) 3)
	Are all ships vertical?	verticalL	(= (+ (map (λ x (= (orient x) H) (set Blue Red Purple)))) 0)
	Are the blue ship and the red ship parallel?	parallel	(= (orient Blue) (orient Red))
touching	Do the blue ship and the red ship touch?	touching	(touch Blue Red)
	Are any of the ships touching?	touchingA	(or (touch Blue Red) (or (touch Blue Purple) (touch Red Purple)))
	Does the blue ship touch any other ship?	touchingXA	(or (touch Blue Red) (touch Blue Purple))
	Does the blue ship touch both other ships?	touchingX1	(and (touch Blue Red) (touch Blue Purple))
demonstration	What is the location of one blue tile?	demonstration	(draw (select (set A1 ... F6) Blue))*
	At what location is the top left part of the blue ship?	topleft	(topleft Blue)
	At what location is the bottom right part of the blue ship?	bottomright	(bottomright Blue)

Defining an infinite set of questions through compositionality



Learning a probabilistic generative model of questions

Goal: predict human questions in novel scenarios

x : question / program
$f(\cdot)$: features (Expected Info. Gain, length, answer type, etc.)
θ : trainable parameters

energy:

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

generative model:

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

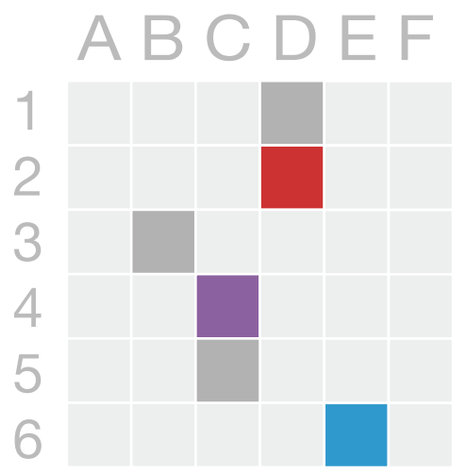
maximum likelihood fitting:

$$\text{argmax}_\theta \; P(D;\theta)$$

D : empirical data

Asking novel questions through program generation

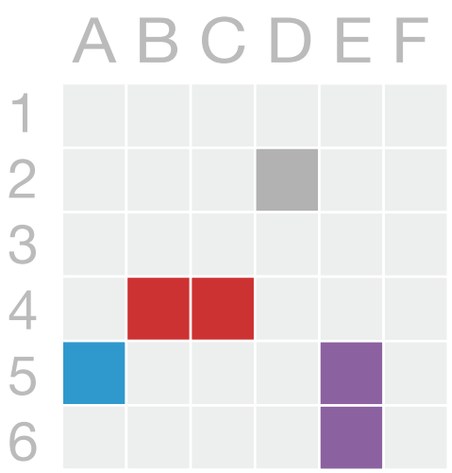
Context 7



EIG	Question/Program	Human
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? `(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



EIG	Question/Program	Human
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? `(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)))`