Learning Part-Based Abstractions for Visual Object Concepts

Haoliang Wang
Dept. of Psychology
UC San Diego
haw027@ucsd.edu

Nadia Polikarpova
Dept. of Computer Science & Engineering
UC San Diego
nadia.polikarpova@ucsd.edu

Judith E. Fan
Dept. of Psychology
UC San Diego
jefan@ucsd.edu

Abstract

The ability to represent semantic structure in the environment — objects, parts, and relations — is a core aspect of human visual perception and cognition. Here we leverage recent advances in program synthesis to develop an algorithm for learning the part-based structure of drawings as represented by graphics programs. This algorithm iteratively learns a library of abstract subroutines that can be used to more compactly represent a set of drawings by capturing common structural elements. Our experiments explore how this algorithm exploits statistical regularities across drawings to learn new subroutines. Together, these findings highlight the potential for understanding human visual concept learning via program-like abstractions.

Keywords: program synthesis; library learning; perceptual organization

Introduction

As humans, we can readily represent semantic structure in our environment — objects, parts, relations, etc. For example, we can effortlessly grasp the correspondence between a real human face and a line drawing of a face (Fig. 1), even without auxiliary cues such as color and texture. Moreover, we immediately know that the two dots in the line drawing represents the eyes, the line beneath it represents the mouth, and the big circle represents the head. How are visual concepts organized such that they robustly encode such abstract correspondences? Here we explore the notion that this robustness reflects the inherently generative and compositional organization of human conceptual knowledge (Palmer, 1977; Tenenbaum, Kemp, Griffiths, & Goodman, 2011). In this paper we present: (1) a proof-of-concept method for learning abstract structural units within objects, represented as subroutines in a graphics library; and (2) computational experiments exploring how the resulting library of learned subroutines is jointly determined by the data distribution and the cost of learning.

There is a long tradition within cognitive science of seeking to characterize the perceptual units by which humans parse the visual world (Palmer, 1977; Goldstone, 2003). Proposed solutions have ranged from feature representations discovered via dimensionality reduction techniques (Lee & Seung, 1999) to those learned by neural networks (Yamins et al., 2014), or recovered by probabilistic inference (Austerweil & Griffiths, 2013). While many approaches have focused on learning image-like internal representations, here instead we aim to learn *graphics programs*, a procedural representation that inherently captures the compositionality of visual concepts

Figure 1: Human faces are configured in consistent ways across varying degrees of visual abstraction. (McCloud, 1994)

(Lake, Ullman, Tenenbaum, & Gershman, 2017; Overlan, Jacobs, & Piantadosi, 2017; Stuhlmuller, Tenenbaum, & Goodman, 2010; Lake & Piantadosi, 2020; Yildirim & Jacobs, 2015), inspired by "vision-as-inverse-graphics" (Kulkarni, Kohli, Tenenbaum, & Mansinghka, 2015; Yildirim, Kulkarni, Freiwald, & Tenenbaum, 2015).

Graphics Programs to Represent Visual Concepts

Our graphics programs are generative models expressed in a *domain-specific language* (DSL), which contains drawing primitives (circles, lines, *etc*), geometric transformations (translation, scaling, *etc*), and, most importantly, supports defining (and calling) new *subroutines*. Here we explore the use of graphics programs with subroutines to model two important aspects of human perceptual organization: (1) *abstraction*: the ability to adapt to input statistics; and (2) *compositionality*: the ability to encode the internal part-based organization of objects. We model abstraction by learning novel subroutines with parameters, as required to explain common patterns across objects. An example of a more abstract subroutine is `face(x,y)`, where the parameters `x` and `y` stand for the shapes of eyes and mouth; the `face` subroutine represents the abstract concept of a face, which captures the positions of facial features inside the face, but not their exact shapes. We model compositionality by learning sets of subroutines that can be composed to capture more complex objects. For example, we can model the drawing of a face wearing a hat with a program that calls two subroutines, `face` and `hat`; this model captures part-based organization by grouping together all the facial features into the concept of a face.

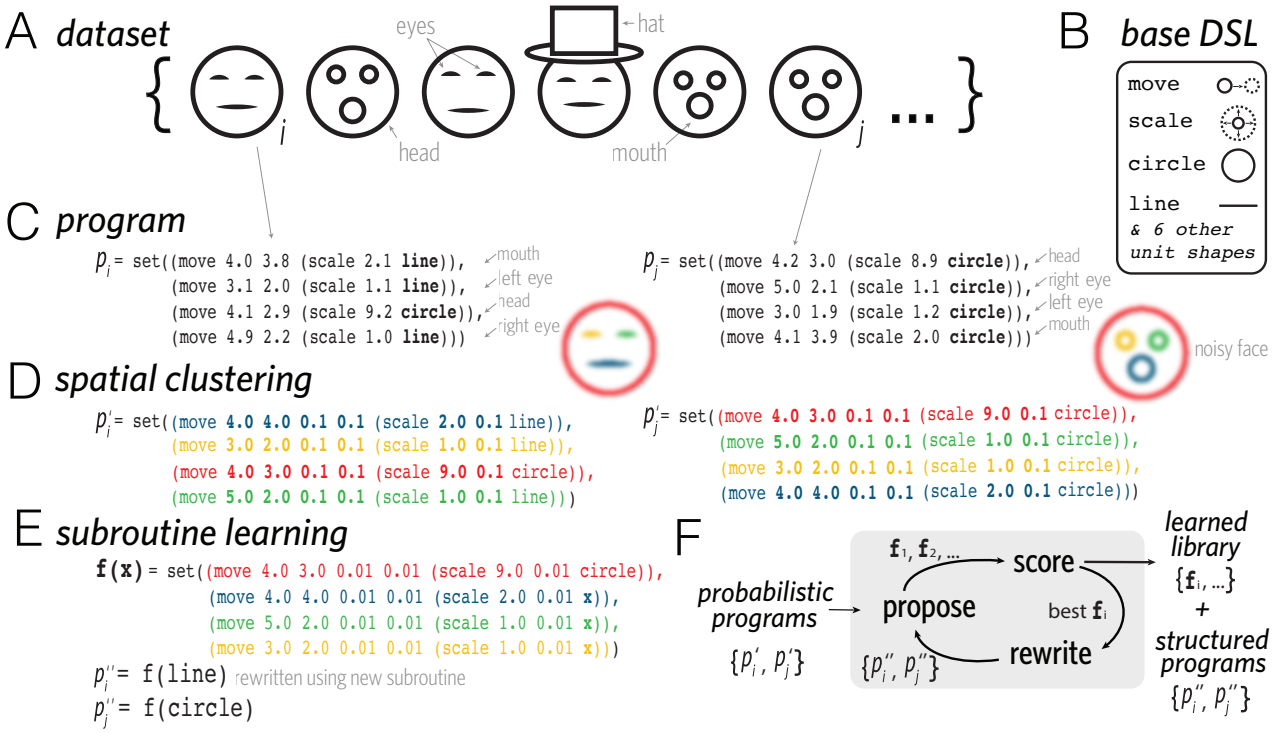

Figure 2: Overview of our algorithm. The algorithm takes as input a dataset of line drawings (A), which are represented as graphics programs (C) written in a base DSL (B). Spatial clustering transforms input programs into probabilistic programs (D). Subroutine learning rewrites probabilistic programs using learned subroutines (E), following an iterative process shown in F.

Program Synthesis

How could such compositional abstractions be learned automatically from a corpus of drawings? To answer this question we turn to an area of computer science called *program synthesis*, which studies algorithms to search for an optimal program within a DSL that reconstructs given data. Program synthesis has recently been used to model how humans recognize and generate handwritten characters (Lake, Salakhutdinov, & Tenenbaum, 2015). This prior work, however, used a fixed DSL and hence could not model the emergence of novel concepts by learning subroutines. Perhaps the study most related to the current paper comes from Tian, Ellis, Kryven, and Tenenbaum, which aimed to model motor structure learning using a recently developed program learning method (Ellis, Morales, Sablé-Meyer, Solar-Lezama, & Tenenbaum, 2018) that could iteratively augment the DSL with new subroutines. In this paper we use a similar learning method, but to our knowledge we are the first to: (1) apply this method to learn structured object concepts, and (2) systematically explore how the data distribution and model parameters affect which abstractions are learned.

Computational Model

As a proof-of-concept, we apply our learning method to the domain of "smiley faces." Fig. 2 presents an overview of our model. The model takes as input a dataset of N line drawings of faces, each represented as a collection of circles, lines, and other geometric primitives (Fig. 2A). In these drawings, facial features can take different shapes, are slightly jittered, and some features (like hats) are optional. The model produces as output: (1) a library of learned subroutines, and (2) N graphics programs that produce the same image as the input drawings, but might contain calls to the learned subroutines (Fig. 2E). The goal of the model is to *minimize* the overall description length of the output corpus, which forces it to learn subroutines that capture common structure in the input drawings, and hence can be reused multiple times.

The learning process is organized into two main stages: *spatial clustering* and *subroutine learning*, which respectively handle continuous and discrete variation in the input dataset. The first stage abstracts away the jitter in the input drawings by inferring which geometric primitives across examples are most likely to represent the same part. The second stage abstracts away shape variation by extracting common fragments of the drawing into subroutines. We will use the running example in Fig. 2 to explain these two stages in more detail.

Smiley Face Dataset

Each smiley face in the input dataset is represented as a program in the *base DSL*, which contains 8 primitive shapes, as well as compound shape expressions that represent geometric transformations and sets of shapes (Fig. 2B). Fig. 2C depicts two smiley faces from the dataset expressed in the base DSL;

as you can see from the figure, the input programs are "flat" and lack any structure: each program is simply a set of primitive shapes, and each shape is individually moved, scaled, and rotated to appear at the right position (in the figure, we omit rotations in the interest of brevity, but our experiments do use rotations). More generally, each smiley face consists of five features: head, mouth, left and right eye, and an optional hat; out of these features, the head is always a `circle`, the hat is always a `line`, and the other features can manifest as any primitive shape. The positions of each of the five features are consistent between the different faces in the corpus, but only approximately: to model our visual environment more realistically, we added noise to each facial feature's location, size, and rotation.

Abstracting Over Continuous Variation via Spatial Clustering

Due to small amounts of spatial variation in the absolute locations of parts across faces, the two heads in Fig. 2C are at slightly different locations (4.1 vs 4.2) and of a slightly different sizes (9.2 vs 8.9). If our model is to learn a structured representation of a face, it first needs to understand that these two circles are describing the same facial feature: the head (note that the order of shapes within each input program is arbitrary, so we cannot assume *e.g.* that head is always the first shape). In other words, we need to cluster the shapes across the input programs based on their spatial proximity (i.e. the numeric parameters of **`move`** and **`scale`**). The results of such clustering are depicted in Fig. 2D, where clusters are visualized as colors (red is head, blue is mouth, *etc*).

To perform the clustering, we assume that the positions of shapes that represent the same facial feature are drawn from the same Gaussian distribution. We then use an existing algorithm called the Chinese Restaurant Process (CRP) (Blei, Griffiths, Jordan, Tenenbaum, et al., 2003; Salakhutdinov, Tenenbaum, & Torralba, 2012), which has the benefit that the number of clusters need not be known a-priori. For example, CRP is able to infer that the **`scale`** ratios of the two heads in Fig. 2C are drawn from the same Gaussian with mean $\mu = 9.0$ and standard deviation $\sigma = 0.1$; similarly, the **`move`** vectors of these two shapes are also drawn from the same distribution; hence these two shapes are clustered together into the "red" cluster. Note that clustering only takes into account the parameters of **`move`** and **`scale`**, but *not* the primitive shape they are applied to; for example, the two mouths Fig. 2C are clustered together as "blue", despite having different shapes.

As a result of clustering, each input program p_k is rewritten into a *probabilistic program* p'_k, where intuitively p_k is a sample from the distribution encoded by p'_k (Fig. 2D). These new programs are written in the *probabilistic DSL*, which is a slight modification of the base DSL, where all geometric transformations have μ and σ parameters. For example, instead of **`scale`**`(9.0)`, which denotes a single scaling transformation, we now write **`scale`**`(9.0, 0.1)`, which denotes a Gaussian distribution of scaling transformations with $\mu = 9.0$ and $\sigma = 0.1$. Fig. 2D demonstrates how this probabilistic **`scale`** is used to describe the head in both p'_i and p'_j (the same parameters are used in both heads, since they were clustered together by CRP).

Abstracting Over Discrete Variation via Subroutine Learning

Consider the two probabilistic programs in Fig. 2D: they share the same structure in the sense that they place four facial features at the same (probabilistic) positions, but they differ in the shapes of eyes and mouth. Our model represents this common structure and variation explicitly by rewriting the corpus of probabilistic programs using a library of learned subroutines. Fig. 2E demonstrates this rewriting for our running example. Here the learned subroutine `f(x)` has the same structure as p'_i and p'_j, but the concrete shapes of eyes and mouth are replace with a parameter, `x`; with this subroutine at hand, we can rewrite the two programs more compactly by simply making a call to the subroutine with different arguments: `f(line)` and `f(circle)`, respectively. Note that the representation in Fig. 2E is more *compact*: while the total size of p'_i and p'_j is $37 \times 2 = 74$ tokens, after the rewriting, the total size of `f`, p''_i and p''_j is only $38 + 2 + 2 = 42$ tokens (the size of `f` is the size of its body plus the number of parameters). This dataset compression is achieved thanks to a careful choice of the subroutine, which maximally captures the common structure between p'_i and p'_j and abstracts away the differences.

Our model learns a library of subroutines automatically by repeatedly performing the following three steps: (1) proposing *fragments* from the probabilistic programs as candidate subroutines; (2) scoring these candidates according to how well they compress the dataset; and (3) re-writing all programs using the highest scoring candidate and adding it to the library (Fig. 2F). In the rest of this section, we describe the three steps in more detail.

Propose In the first step, our model proposes a set of candidate subroutines by extracting fragments from the given probabilistic programs, matching those fragments against fragments of other programs in the dataset, and abstracting away their differences into parameters using a technique known as *anti-unification* (Plotkin, 1970). Let us illustrate this procedure on our running example. Given the program p'_i, we first generate a fragment for each of the 15 non-empty subsets of its shapes; for example, the following fragment f^4 represents just the head and f^7 represents head and mouth (we omit the σ parameters in this section for brevity):

```
f4 = move(4.0, (scale(9.0, circle)))
f7 = set(move(4.0, (scale(9.0, circle))),
         move(4.0, (scale(2.0, line))))
```

Finally $f^{15} = p'_i$ is a fragment representing the whole face.

Next, the model attempts to match (or *anti-unify*) each fragment f^k_i from p'_i with each fragment f^m_j from p'_j; for the match to be successful, we require that the numeric parameters of **`move`** and **`scale`** must coincide, while the primitive shapes in

the two fragments might differ, in which case they are replaced by a parameter. For example, matching f^4 with the *mouth* fragment from p'_j fails, since their positions differ. Matching f^4 with the *head* fragment from p'_j succeeds and yields f^4 itself as a candidate subroutine, since the two fragments are identical; this subroutine is a *constant*, *i.e.* has no parameters. Finally, matching f^{15} against the whole p'_j succeeds and yields a *unary* subroutine `f(x)` from Fig. 2E, *i.e.* a subroutine with a single parameter, x, which replaces the mismatched shapes. This subroutine is unary because all mismatches between the two fragments are of the same form: `line` on the left vs `circle` on the right. More generally, matching two *incongruent faces* (where the shape of the mouth differs from the shape of the eyes), may yield a *binary* subroutine `f(x,y)` with two parameters: one for the shape of the eyes and the other one for the shape of the mouth. In total, a dataset with just $\{p'_i, p'_j\}$ yields 15 candidate subroutines, because each fragment of p'_i has a unique matching fragment in p'_j. For a larger dataset, the number of candidates can grow if we include faces with hats and/or incongruent faces.

Score In each iteration of the loop in Fig. 2F, we add a single most promising candidate subroutine f to the current library $\mathcal{L}$. To pick such f, we score each candidate using the following loss function:

$$\text{loss}(f) = w \times \text{size}(\mathcal{L} \cup \{f\}) + \frac{\sum_{k=1}^{N} \text{MDL}(p'_k \mid \mathcal{L} \cup \{f\})}{N} \quad (1)$$

This loss function consists of two terms: the first is the size of the new library, with f added, and the second one is the average *minimum description length* (MDL) of a probabilistic program conditioned on the new library. The size of a library is the total number of tokens in all its subroutines; the contribution of the library size term is controlled by a weight parameter, w, reflecting the cost of learning. The MDL of a program conditioned on a library is defined as:

$$\text{MDL}(p \mid \mathcal{L}) = \min\{\text{size}(p') \mid p' \xrightarrow{\mathcal{L}}{}^{*} p\} \quad (2)$$

Here $p' \xrightarrow{\mathcal{L}}{}^{*} p$ means that the program p' can be rewritten into p by substituting calls to subroutines from $\mathcal{L}$ with their definitions. Intuitively, the MDL is the size of the most compact version of p written in terms of the library $\mathcal{L}$. For example, the minimal description length for p'_i in Fig. 2D using the base library is 37, and when we augment the base library with `f(x)`, this terms becomes 2: one for the function itself and one for function application, shown in Fig. 2E.

The candidate f with the minimal loss hence provides maximal compression of input programs while minimizing expansion of the library. In our running example, among the 15 candidates proposed for the dataset $\{p_i, p_j\}$, the highest-scoring candidate is the unary subroutine `f(x)` from Fig. 2E, which represents the whole face. For example, with $w = 0.5$, adding this subroutine to the empty library changes the loss from $0 + 37 = 37$ to $0.5 \times 38 + 2 = 21$ In contrast, adding *e.g.* the constant subroutine f^4 defined above would result in the loss of $0.5 \times 9 + 29 = 33.5$.

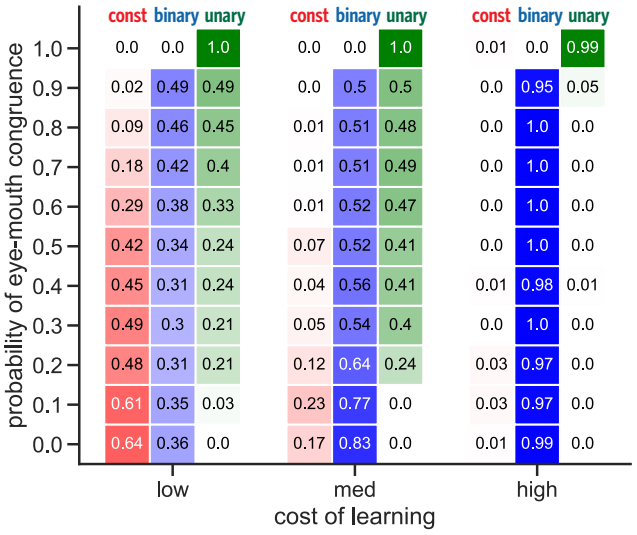

Figure 3: Heat map representing the relative frequency with which different kinds of subroutines (i.e., constant, unary, and binary) were learned, for different values for the cost of learning (x axis) and different degrees of covariance between features (y axis). The patterns are similar across different ws, and more pronounced for higher values of w.

Rewrite After the candidate subroutine with the highest score has been selected and added to the library, all current programs are re-written to make use of the new subroutine. For example, as shown in Fig. 2(E), the programs p'_i and p'_j are rewritten in terms of the newly added subroutine `f(x)`. The resulting programs p''_i and p''_j are used as the input to the next iteration of library learning. The process stops when none of the proposed candidate subroutines improve the loss any further. In our example, p''_i and p''_j each contain only a single fragment, and the result of matching these fragments is a unary candidate subroutine `g(x) = f(x)`. Rewriting p''_i and p''_j in terms of `g(x)` does not change their size (it simply replaces `f` with `g`); hence this candidate is rejected, since it increases the size of the library without compressing the corpus.

Computational Experiments

The goal of our experiments was to explore how different constraints on learning, supplied either by external variables (*i.e.*, the data distribution) or internal components of the model (*i.e.*, the cost of learning w in Eq. 1), jointly influenced the subroutines our algorithm learned.

Experiment setup

Building on classic work investigating perceptual learning in cognitive science (Goldstone, 2003; Austerweil & Griffiths, 2013), we designed our data distributions to vary along two dimensions:

- **Number of shapes**: We varied the number of different shapes that the eyes and mouth could take, which could

be any number between two (`circle` and `line`) and eight (`circle`, `line`, `heart`, etc.).

- **Covariance between features**: We varied how strongly particular facial features co-occurred. Specifically, the probability P_{cong} that the shape of eyes and mouth are congruent varied between 0.0 (where the eyes and mouth were always represented by different shapes) and 1.0 (where the eyes and mouth matched).

We generated 100 samples from each data distribution, each sample containing 100 smiley faces. We then ran our algorithm for each sample and analyzed the library of learned subroutines. To measure how the data distribution affects the level of *abstraction* of the learned concepts, we analyzed whether each library contained a "`face` subroutine" (*i.e.* a subroutine with a head, two eyes, and a mouth) with different number of parameters:

- *Constant face* (*e.g.* p_i' in Fig. 2): these subroutines contain constant shape primitives, reflecting cases where the model fails to learn more abstract structure but instead memorized subsets of the original input programs. These subroutines tend to only be applicable to a small number of faces, and thus not reusable across examples.

- *Unary face* (*e.g.* `f(x)` in Fig. 2): these subroutines have a single parameter, and hence are able to abstract over a single shape inside the face (*e.g.* eyes and mouth in a congruent face). This kind of subroutine is more expressive than the constant subroutine because it can be applied to new shapes that the model has never seen before so long as they exhibit the same structure.

- *Binary face*: this subroutine has two parameters, one for the the eyes and one for the mouth. Is is the most flexible abstraction, since it captures all variations of facial features in our dataset.

Manipulating the cost of learning

First we explored the consequences of varying w, the *cost of learning*, over a range, $0 \leq w \leq 0.07$ (Fig. 3). This cost-of-learning parameter can be interpreted as a simple proxy for various constraints on human learning and memory that can vary widely across task contexts, such as cognitive load (Sweller, 1988). We found that at a low level of w (w=0.01), the model is free to learn all three types of subroutines, because it has very little pressure against memorization. As w increases, learning constant subroutines is disfavored (from left to right in Fig. 3, the frequency of constant subroutines decreases), forcing the agent to discover regularities in the data distribution and learn abstractions that are reusable across a larger proportion of the dataset. And at a high level of w (w=0.07), the binary subroutine dominates, because it is the most expressive one and thus gives the best compression of the visual environment.

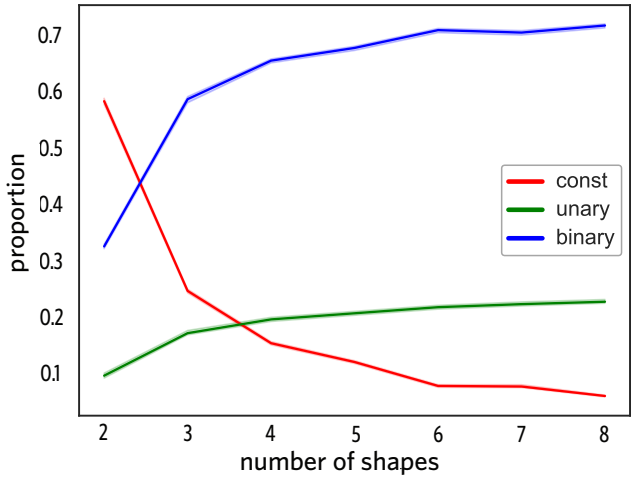

Figure 4: Proportion of samples in which a given type of subroutine was learned as a function of number of shapes that the eyes and mouth could take. Error bands reflect 95% CIs.

Manipulating the covariance between features

Insofar as our model can exploit statistical regularities in the co-occurrence of parts across different smiley faces, we hypothesized that it would be better able to learn meaningful correlations between different parts (i.e., that line-eyes and line-mouths tend to appear within the same face) when the data distribution supported these inferences. Over a wide range of w, we found that when most faces are incongruent ($P_{\text{cong}} = 0.1$), the model hardly ever learns unary subroutines. As faces become more likely to be congruent, the probability that the model learns unary subroutines gradually increases, while the prevalence of binary subroutines remains stable. In the perfectly congruent distribution, the model only learns the unary subroutine (Fig. 3).

Manipulating the number of shapes

In the experiments conducted so far there were always eight different shape primitives that the eyes and mouth could take. To explore how varying the number of unique shapes constrained the kinds of subroutines the model learned, we varied the number of primitives between 2 and 8, but otherwise replicated the same settings as in the above experiments. We computed the proportion of each type of learned subroutine by summing over values of P_{cong} and w (Fig. 4). We found that when there are only two shapes, the model tends to memorize each face rather than learn abstractions. However, as the number of shapes increases, the model is able to acquire a more compact representation of the visual environment by learning more abstract subroutines, as it becomes increasingly expensive to memorize specific combinations of features.

Manipulating the prevalence of features

So far we have we assumed that all faces always have the same four features (head, left eye, right eye, and mouth) and focused on different types of `face` subroutine the model learned

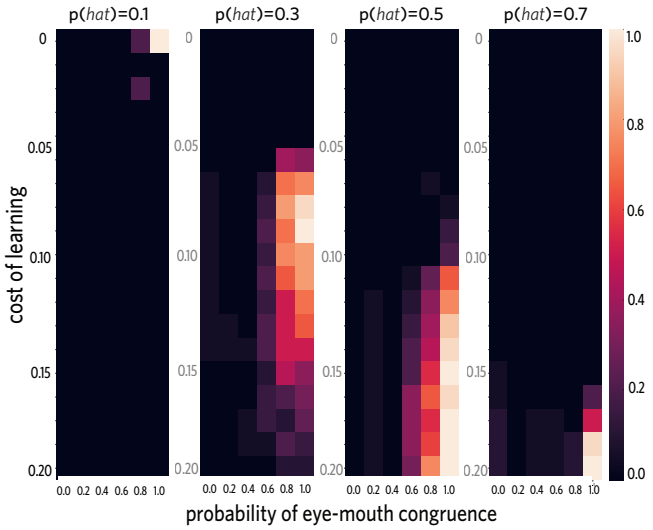

Figure 5: Each heat map corresponds to a different data distribution with a P_{hat}. Each row indicates a different value of w, each column indicates a different value of P_{cong}. Each cell represents the proportion of samples in which the `hat` subroutine was learned as a separate concept.

under different settings. A key challenge for learning good perceptual representations is to determine when to learn a unified, albeit more complex concept (e.g., a binary face subroutine) and when to instead learn separate, but simpler concepts (Goldstone, 2003). To explore this trade-off, we conducted a new set of experiments in which we manipulated the probability that a face would be wearing a "hat" (P_{hat}), the cost of learning (w), and covariance between features (P_{cong}), and measured how often `hat` was learned as a separate concept from the face (Fig. 5). We found that all three of these variables interact to determine how likely it is for the model to learn a separate `hat` subroutine: specifically, it is more likely to do so at higher values of P_{cong} when it is clearer that the shape features within the face and the hat are statistically independent. Moreover, for higher values of P_{hat}, the threshold for w at which the model learns a separate `hat` subroutine is also higher. This result reflects the fact that for higher values of P_{hat} the model observes the "hat" together with faces more often, making it more beneficial (i.e., for reducing *MDL*) to learn a "unitized" `face_hat` subroutine, rather than two separate `face` and `hat` subroutines.

Discussion

In this paper, we presented a proof-of-concept method for learning abstract structural units within objects, represented as subroutines in a graphics library, and investigated how the resulting library of learned subroutines is jointly determined by the data distribution and the cost of learning. Our approach takes inspiration from "vision-as-inverse-graphics", whereby structured visual representations are learned by synthesizing graphics programs whose internal structure captures statistical regularities in the input.

A key step in learning part-based abstractions in graphics programs is proposing novel subroutines that may represent these parts. The method we developed uses *anti-unification* (Hwang, Stuhlmüller, & Goodman, 2011; Rule, 2020), an algorithm widely used in computer science to discover commonalities between symbolic expressions, and which is also closely related to *structure mapping*, an influential algorithm that has been used to model analogical reasoning in cognitive science (Falkenhainer, Forbus, & Gentner, 1989). A promising avenue for future research would be to apply related program-synthesis techniques to model how humans detect and propose analogies.

A critical open question concerns the scalability of the algorithm, as the drawings used in the current experiments are highly simplified: the input data are represented not as images but as programs, in which key structural primitives have already been segmented (e.g., the shapes). A way to address this issue is to apply modern visual encoding algorithms to extract symbolic expressions from raw pixel input (Ellis, Ritchie, Solar-Lezama, & Tenenbaum, 2017), building on recent advances using neural networks to learn more structured, graph-based latent representations (Mrowca et al., 2018; Battaglia et al., 2018; Bear et al., 2020) that explicitly represent object-like primitives and relations. A promising direction for future work is to combine such modeling techniques with techniques from program synthesis to model human visual concept learning under realistic levels of image complexity and variation.

In the current set of experiments, each facial part was already categorized as being either continuous (e.g., location) or categorical (i.e., shape) variables, which determined whether clustering or anti-unification was applied to them to learn more abstract representations. However, in more realistic settings learners are generally not told which features are which, and thus future work should develop models that can infer what type of learning approach is appropriate.

It may also be valuable to explore applications of this learning algorithm to make predictions about which abstractions people infer when learning to parse and generate novel symbols, and what kind of experience is necessary to support such inferences (Tian et al., 2020; Lake et al., 2015). Although we focus in this paper on modeling such analogical structure in graphics programs, there is potential for applying related to techniques to model learning of compositional structure in other types of input, such as speech (Saffran, Aslin, & Newport, 1996). In the long run, these modeling approaches have strong potential for leading both more robust artificial intelligence and a deeper understanding of human cognition.

Acknowledgments

We would like to thank Shraddha Barke, Rose Kunkel, Yuyao Wang, Ed Vul, and members of the Cognitive Tools Lab at UC San Diego for very helpful discussion. This work was supported by NSF CAREER Award #2047191 to J.E.F.

References

Austerweil, J. L., & Griffiths, T. L. (2013). A nonparametric bayesian framework for constructing flexible feature representations. *Psychological review*, *120*(4), 817.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., ... others (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Bear, D. M., Fan, C., Mrowca, D., Li, Y., Alter, S., Nayebi, A., ... others (2020). Learning physical graph representations from visual scenes. *arXiv preprint arXiv:2006.12373*.

Blei, D. M., Griffiths, T. L., Jordan, M. I., Tenenbaum, J. B., et al. (2003). Hierarchical topic models and the nested chinese restaurant process. In *Nips* (Vol. 16).

Ellis, K., Morales, L., Sablé-Meyer, M., Solar-Lezama, A., & Tenenbaum, J. (2018). Library learning for neurally-guided bayesian program induction. In *Neurips*.

Ellis, K., Ritchie, D., Solar-Lezama, A., & Tenenbaum, J. B. (2017). Learning to infer graphics programs from hand-drawn images. *arXiv preprint arXiv:1707.09627*.

Falkenhainer, B., Forbus, K. D., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial intelligence*, *41*(1), 1–63.

Goldstone, R. L. (2003). Learning to perceive while perceiving to learn. In *Perceptual organization in vision* (pp. 245–290). Psychology Press.

Hwang, I., Stuhlmüller, A., & Goodman, N. D. (2011). Inducing probabilistic programs by bayesian program merging. *arXiv preprint arXiv:1110.5667*.

Kulkarni, T. D., Kohli, P., Tenenbaum, J. B., & Mansinghka, V. (2015). Picture: A probabilistic programming language for scene perception. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4390–4399).

Lake, B. M., & Piantadosi, S. T. (2020). People infer recursive visual concepts from just a few examples. *Computational Brain & Behavior*, *3*(1), 54–65.

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

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, *40*.

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*(6755), 788–791.

McCloud, S. (1994). *Understanding comics*. HarperPerennial. Retrieved from `https://books.google.com/books?id=oJ1vPwAACAAJ`

Mrowca, D., Zhuang, C., Wang, E., Haber, N., Fei-Fei, L., Tenenbaum, J. B., & Yamins, D. L. (2018). Flexible neural representation for physics prediction. *arXiv preprint arXiv:1806.08047*.

Overlan, M. C., Jacobs, R. A., & Piantadosi, S. T. (2017). Learning abstract visual concepts via probabilistic program induction in a language of thought. *Cognition*, *168*, 320–334.

Palmer, S. E. (1977). Hierarchical structure in perceptual representation. *Cognitive psychology*, *9*(4), 441–474.

Plotkin, G. (1970). *Lattice theoretic properties of subsumption*. Edinburgh University, Department of Machine Intelligence and Perception. Retrieved from `https://books.google.com/books?id=2p09cgAACAAJ`

Rule, J. S. (2020). *The child as hacker: building more human-like models of learning*. Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Saffran, J. R., Aslin, R. N., & Newport, E. L. (1996). Statistical learning by 8-month-old infants. *Science*, *274*(5294), 1926–1928.

Salakhutdinov, R., Tenenbaum, J., & Torralba, A. (2012). One-shot learning with a hierarchical nonparametric bayesian model. In *Proceedings of icml workshop on unsupervised and transfer learning* (pp. 195–206).

Stuhlmuller, A., Tenenbaum, J. B., & Goodman, N. D. (2010). Learning structured generative concepts..

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, *12*(2), 257–285.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., & Goodman, N. D. (2011). How to grow a mind: Statistics, structure, and abstraction. *science*, *331*(6022), 1279–1285.

Tian, L., Ellis, K., Kryven, M., & Tenenbaum, J. (2020). Learning abstract structure for drawing by efficient motor program induction. *Advances in Neural Information Processing Systems*, *33*.

Yamins, D. L., Hong, H., Cadieu, C. F., Solomon, E. A., Seibert, D., & DiCarlo, J. J. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, *111*(23), 8619–8624.

Yildirim, I., & Jacobs, R. A. (2015). Learning multisensory representations for auditory-visual transfer of sequence category knowledge: a probabilistic language of thought approach. *Psychonomic bulletin & review*, *22*(3), 673–686.

Yildirim, I., Kulkarni, T. D., Freiwald, W. A., & Tenenbaum, J. B. (2015). Efficient analysis-by-synthesis in vision: A computational framework, behavioral tests, and comparison with neural representations. In *Thirty-seventh annual conference of the cognitive science society* (Vol. 4).