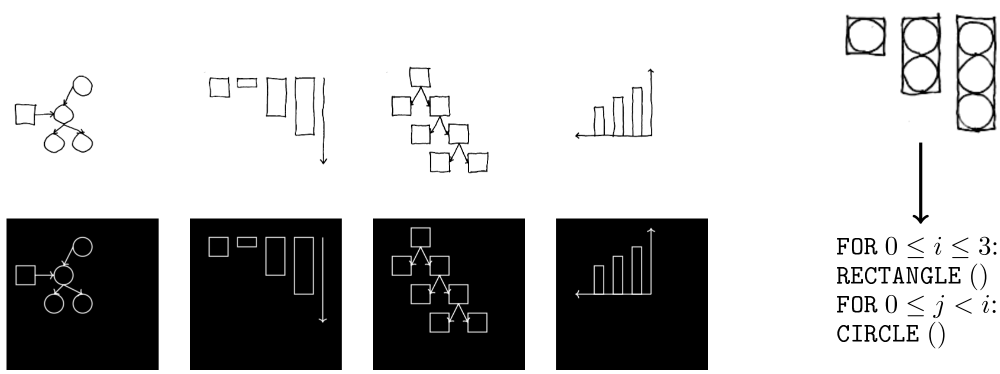
Inferring Graphics Programs from Images

Anonymous Author(s)
Affiliation
Address
email

Abstract

1 Introducing visual programs

How could an agent go from noisy, high-dimensional perceptual input to a symbolic, abstract object, like a computer program? Here we consider this problem within the context of *graphics program synthesis*. We develop an approach for converting natural images, such as hand drawings, into executable source code for drawing the original image.



(a) (b)

Figure 1: (a): Model parses hand drawings (top) into LaTeX (bottom). (b) Synthesizes high-level *graphics program* from hand drawing.

High dimensional perceptual input is ill matched to the abstract semantics of a programming language. But programs with constructs like recursion or iteration produce a simpler *execution trace* of primitive actions. Our hypothesis is that the execution trace of the program is better aligned with the perceptual input, and that the trace can act as a kind of bridge between perception and programs. We test this hypothesis by developing a model that learns to map from an image to the execution trace of the graphics program that drew it. With the execution trace in hand, we can bring to bear techniques from the program synthesis community to recover the latent graphics program.

In this work we consider programs that draw diagrams, similar to those found in papers.

We develop a hybrid architecture for inferring graphics programs. Our approach uses a deep network infer an execution trace from an image; this recovers primitive drawing operations like lines, circles, or arrows. For added robustness we use the deep network as a proposal distribution for a stochastic search over execution traces. Finally we use techniques in the program synthesis community to recover the program from its trace.

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

Each of these three components – the deep network, the stochastic search, the program synthesizer – confers its own advantages. From the deep network we get a very fast system that can recover plausible execution traces in about a minute. From the stochastic search we get added robustness; essentially the stochastic search can correct mistakes made by the deep network's proposals. From the program synthesizer we get abstraction: our system recovers coordinate transformations, for loops, and subroutines, which are useful for downstream tasks.

2 Related work

attend infer repeat: [1]. Crucial distinction is that they focus on learning the generative model jointly with the inference network. Advantages of our system is that we learn symbolic programs, and that we do it from hand sketches rather than synthetic renderings.

ngpm: [2]. We build on the idea of a guide program, extending it to scenes composed of objects, and then show how to learn programs from the objects we discover.

Sketch-n-Sketch: [3]. Semiautomated synthesis presented in a nice user interface. Complementary to our work: you could pass a sketch to our system and then pass the program to sketch-n-sketch

Converting hand drawings into procedural models using deep networks: [4, 5].

3 Neural architecture for inferring image parses

We developed a deep network architecture for efficiently inferring a execution trace, T, from an image, I. Our model constructs the trace one drawing command at a time. When predicting the next drawing command, the network takes as input the target image I as well as the rendered output of previous drawing commands. Intuitively, the network looks at the image it wants to explain, as well as what it has already drawn. It then decides either to stop drawing or proposes another drawing command to add to the execution trace; if it decides to continue drawing, the predicted primitive is rendered to its "canvas" and the process repeats.

Figure 2 illustrates this architecture. We first pass the target image and a rendering of the trace so far to a convolutional network. Given the features extracted by the convolutionional network, a multilayer perceptron then predicts a distribution over the next drawing command to add to the trace. We predict the drawing command token-by-token, and condition each token both on the image features and on the previously generated tokens. For example, the network first decides to emit the `CIRCLE` token conditioned on the image features, then it emits the x coordinate of the circle conditioned on the image features and the `CIRCLE` token, and finally it predicts the y coordinate of the circle conditioned on the image features, the `CIRCLE` token, and the x coordinate.

The distribution over the next drawing command factorizes:

$$\mathbb{P}_\theta[t_1 t_2 \cdots t_K | I, T] = \prod_{k=1}^{K} \mathbb{P}_\theta[t_k | f_\theta(I, \text{render}(T)), \{t_j\}_{j=1}^{k-1}] \tag{1}$$

where $t_1 t_2 \cdots t_K$ are the tokens in the drawing command, I is the target image, T is an execution trace, θ are the parameters of the neural network, and $f_\theta(\cdot, \cdot)$ is the image feature extractor (convolutional network). The distribution over execution traces factorizes as:

$$\mathbb{P}_\theta[T|I] = \prod_{n=1}^{|T|} \mathbb{P}_\theta[T_n | I, T_{1:(n-1)}] \times \mathbb{P}_\theta[\texttt{STOP}|I, T] \tag{2}$$

where $|T|$ is the length of execution trace T, and the `STOP` token is emitted by the network to signal that the execution trace explains the image.

We train the network by sampling execution traces T and target images I for randomly generated scenes, and maximizing (2) wrt θ by gradient ascent. Despite the architecture being recurrent, training is fully supervised. In a sense, this model is like an autoregressive variant of AIR.

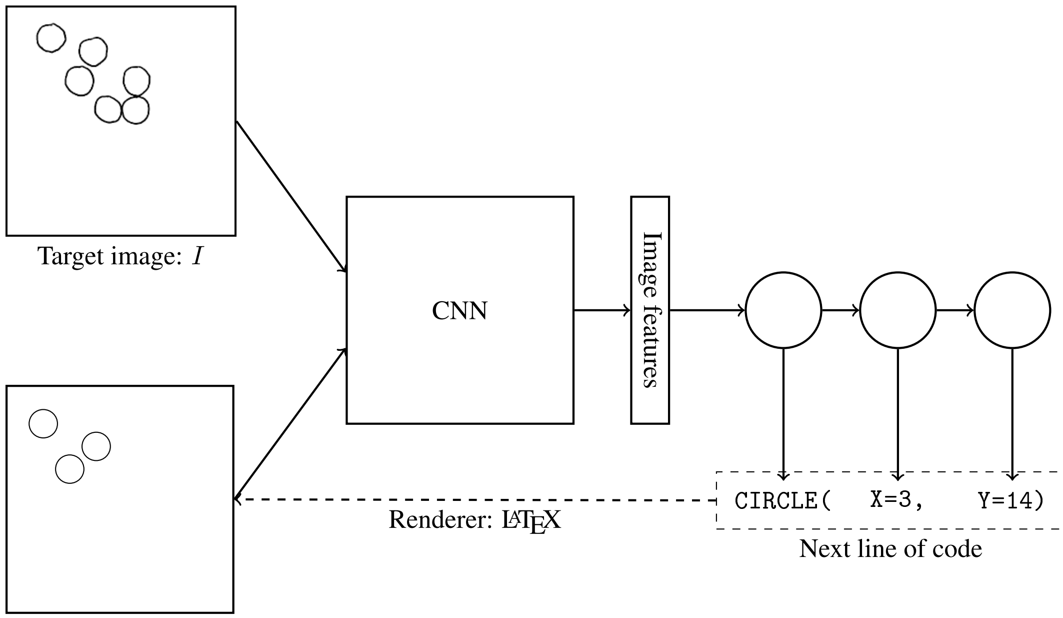


Figure 2: Our neural architecture for inferring the execution trace of a graphics program from its output.

CIRCLE(x, y)	Circle at (x, y)
RECTANGLE(x_1, y_1, x_2, y_2)	Rectangle with corners at (x_1, y_1) & (x_2, y_2)
LINE$(x_1, y_1, x_2, y_2,$ arrow $\in \{0,1\}$, dashed $\in \{0,1\})$	Line from (x_1, y_1) to (x_2, y_2), optionally with an arrow and/or dashed
STOP	Finishes execution trace inference

Table 1: The deep network in (2) predicts drawing commands, shown above.

Program→	Command; $\cdots$; Command
Command→	CIRCLE(Expression,Expression)
Command→	RECTANGLE(Expression,Expression,Expression,Expression)
Command→	LINE(Expression,Expression,Expression,Expression,Boolean,Boolean)
Command→	FOR$(0 \leq$ Var $<$ Expression$)$ { Program }
Command→	REFLECT(Axis) { Program }
Expression→	Z * Var + Z
Var→	A free (unused) variable
Z→	an integer
Axis→	X = Z
Axis→	Y = Z

Table 2: Grammar over graphics programs. We allow loops (FOR), vertical/horizontal reflections (REFLECT), and affine transformations (Z * Var + Z).

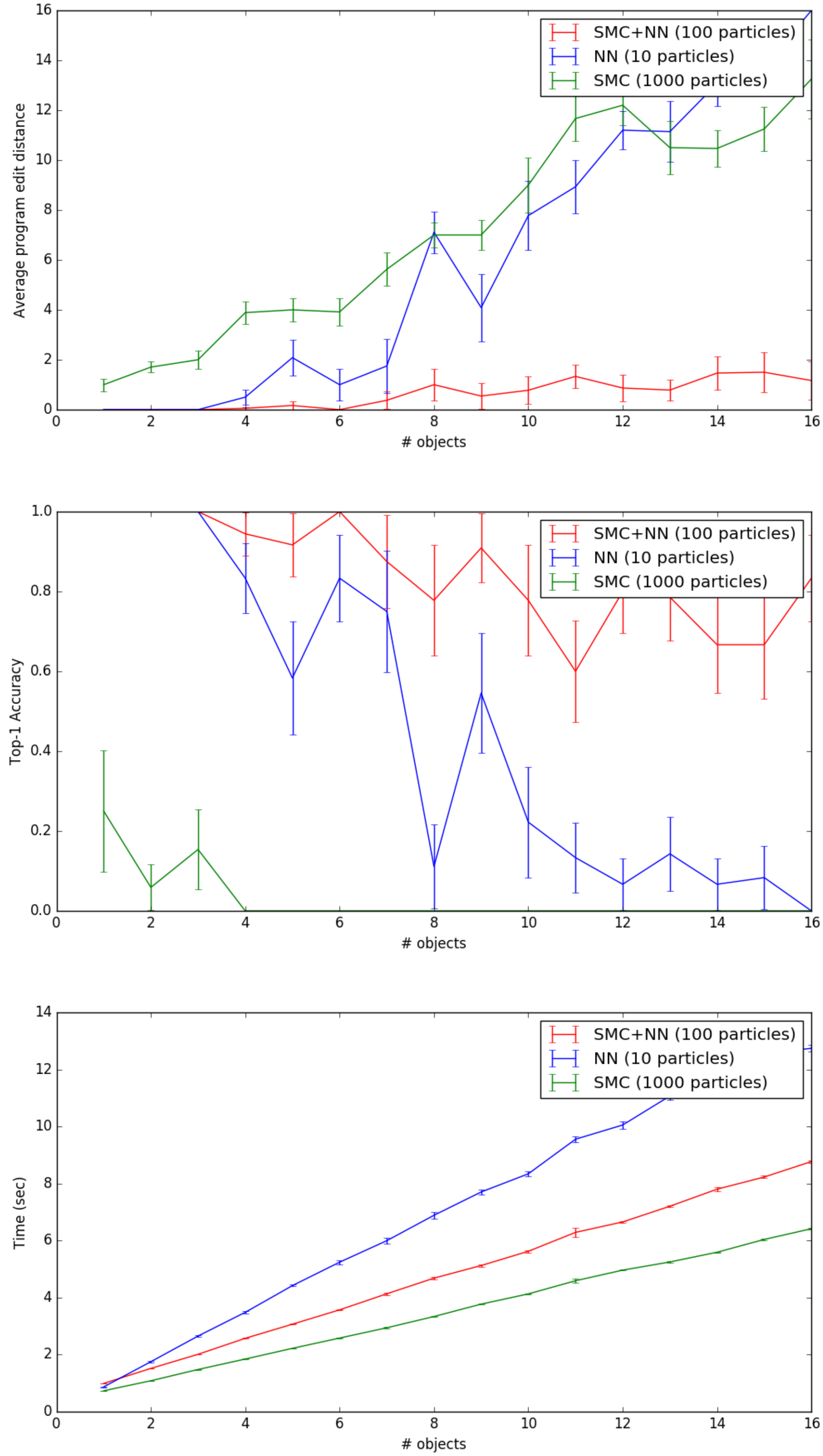
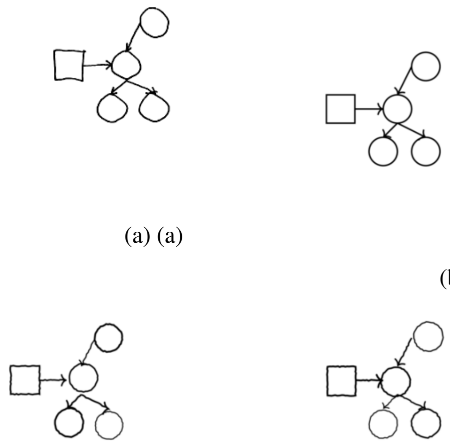


Figure 3: Using the model to parse latex output. The model is trained on diagrams with up to 8 objects. As shown above it generalizes to scenes with many more objects. Neither the stochastic search nor the neural network are sufficient on their own.



(a) (a)

(b) (b)

(c) (c)

(d) (d)

Figure 4: (a): a hand drawing. (b): Rendering of the parse our model infers for (a). We can generalize to hand drawings like these because we train the model on images corrupted by a noise process designed to resemble the kind of noise introduced by hand drawings - see (c) & (d) for noisy renderings of (b).

4 Generalizing to hand drawings

5 Synthesizing graphics programs from execution traces

6 Contributions

We see this system as a first step towards more powerful graphics program synthesizers. We look forward to, in the near future, being able to hand-draw our figures and automatically convert them into publication-quality $\LaTeX$code.

7 Neural networks for guiding SMC

Let $L(\cdot|\cdot)$: image$^2 \to \mathcal{R}$ be our likelihood function: it takes two images, an observed target image and a hypothesized program output, and gives the likelihood of the observed image conditioned on the program output. We want to sample from:

$$\mathbb{P}[p|x] \propto L(x|\text{render}(p))\mathbb{P}[p] \tag{3}$$

where $\mathbb{P}[p]$ is the prior probability of program p, and x is the observed image.

Let p be a program with L lines, which we will write as $p = (p_1, p_2, \cdots, p_L)$. Assume the prior factors into:

$$\mathbb{P}[p] \propto \prod_{l \leq L} \mathbb{P}[p_l] \tag{4}$$

Define the distribution $q_L(\cdot)$, which happens to be proportional to the above posterior:

$$q_L(p_1, p_2, \cdots, p_{L-1}, p_L) \propto q_{L-1}(p_1, p_2, \cdots, p_{L-1}) \times \frac{L(x|\text{render}(p_1, p_2, \cdots, p_{L-1}, p_L))}{L(x|\text{render}(p_1, p_2, \cdots, p_{L-1}))} \times \mathbb{P}[p_L] \tag{5}$$

Now suppose we have some samples from $q_{L-1}(\cdot)$, and that we then sample a p_L from a distribution proportional to $\frac{L(x|\text{render}(p_1,p_2,\cdots,p_{L-1},p_L))}{L(x|\text{render}(p_1,p_2,\cdots,p_{L-1}))} \times \mathbb{P}[p_L]$. The resulting programs p are distributed according to q_L, and so are also distributed according to $\mathbb{P}[p|x]$.

How do we sample p_L from a distribution proportional to $\frac{L(x|\text{render}(p_1,p_2,\cdots,p_{L-1},p_L))}{L(x|\text{render}(p_1,p_2,\cdots,p_{L-1}))} \times \mathbb{P}[p_L]$? We have a neural network that takes as input the target image x and the program so far, and produces a distribution over next lines of code (p_L). We write $\text{NN}(p_L|p_1, \cdots, p_{L-1}; x)$ for the distribution output by the neural network. So we can sample from NN and then weight the samples by:

$$w(p_L) = \frac{\mathbb{P}[p_L]}{\text{NN}(p_L|p_1, \cdots, p_{L-1}; x)} \times \frac{L(x|\text{render}(p_1, p_2, \cdots, p_{L-1}, p_L))}{L(x|\text{render}(p_1, p_2, \cdots, p_{L-1}))} \tag{6}$$

Then we can resample from these now weighted samples to get a new population of particles (here programs are particles), where each program now has L lines instead of $L - 1$.

This procedure can be seen as a particle filter, where each successive latent variable is another line of code, and the emission probabilities are successive ratios of likelihoods under $L(\cdot|\cdot)$.

Comments for Dan. Right now I'm not actually sampling from the neural network - instead, I enumerate the top few hundred lines of code suggested by the network, and then weight them by their likelihoods. So actually the form of NN is:

$$\text{NN}(p_L|p_1, \cdots, p_{L-1}; x) \propto \begin{cases} 1, & \text{if } p_L \in \text{top hundred neural network proposals} \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Do you think this is a problem? The neural network puts almost all of its mass on a few guesses. In order to get the correct line of code I sometimes need to get something like the 50th top guess, so I don't want to literally just sample from the distribution suggested by the neural network.

References

[1] SM Eslami, N Heess, and T Weber. Attend, infer, repeat: Fast scene understanding with generative models. arxiv preprint arxiv:..., 2016. *URL http://arxiv. org/abs/1603.08575*.

[2] Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah Goodman. Neurally-guided procedural models: Amortized inference for procedural graphics programs using neural networks. In *Advances In Neural Information Processing Systems*, pages 622–630, 2016.

[3] Brian Hempel and Ravi Chugh. Semi-automated svg programming via direct manipulation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 379–390, New York, NY, USA, 2016. ACM.

[4] Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. Shape synthesis from sketches via procedural models and convolutional networks. *IEEE transactions on visualization and computer graphics*, 2017.

[5] Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive sketching of urban procedural models. *ACM Trans. Graph.*, 35(4), 2016.

Algorithm 1 Neurally guided SMC

Input: Neural network NN, beam size N, maximum length L, target image x
Output: Samples of the program trace
Set $B_0 = \{\text{empty program}\}$
for $1 \leq l \leq L$ **do**
  for $1 \leq n \leq N$ **do**
    $p_n \sim \text{Uniform}(B_{l-1})$
    $p'_n \sim \text{NN}(\text{render}(p), x)$
    Define $r_n = p'_n \cdot p_n$
    Set $\tilde{w}(r_n) = \frac{L(x|r_n)}{L(x|p_n)} \times \frac{\mathbb{P}[p'_n]}{\mathbb{P}[p'_n = \text{NN}(\text{render}(p), x)]}$
  end for
  Define $w(p) = \frac{\tilde{w}(p)}{\sum_{p'} \tilde{w}(p')}$
  Set B_l to be N samples from r_n distributed according to $w(\cdot)$
end for
return $\{p : p \in B_{l \leq L}, p \text{ is finished}\}$