Program Synthesis for Images using Tree-Structured LSTM

Chenghui Zhou *
Carnegie Mellon University

Chun-liang Li
Carnegie Mellon University

Barnabas Poczos
Carnegie Mellon University

Abstract

Program synthesis has recently emerged as a promising approach for image parsing. However, prior work has relied on supervised learning using training images paired with ground truth generating programs. We present an unsupervised learning algorithm that learns to parse constructive solid geometry (CSG) images into a context-free grammar. In addition, most prior work has relied on differentiable image renderers, whereas most renderers for practical applications are non-differentiable. We design a grammar-encoded tree LSTM to effectively constrain our search space by leveraging the structure of the context-free grammar while handling non-differentiable renderers via REINFORCE and encouraging exploration by regularizing the objective with an entropy term. We propose a lower variance entropy estimator for effective exploration. We demonstrate the effectiveness of the proposed algorithm on a synthetic 2D CSG dataset.

1 Introduction

The goal of our paper is learning to parse images into programs which can be described by a context-free grammar, such as constructive solid geometry (CSG) image [Hub90]. [Sha+18] have studied the same problem using supervised learning where paired images and programs are available for training. However, in this paper we tackle a more general image parsing problem where the ground truth program is not available for training. We aim to solve this problem with only target images as input without any other form of data supervision.

In addition, our proposed approach makes use of only a non-differentiable image renderer. While prior work has relied on differentiable renderes, for most real-world applications the renderers which take programs as inputs are usually non-differentiable. The lack of direct gradient information from the non-differentiable renderer implies that we can only train our model based on comparing the final reconstructed image to the target image. We use REINFORCE as our main building component.

A naive implementation of program generation is using a simple RNN, which is not guaranteed to always generate grammatically correct programs. However, a renderer would only output blank canvases for invalid program input. Therefore, it is necessary to limit the search space to only valid programs. We leverage tree LSTM to impose a structure on our output space such that the output grammar is always valid, thus effectively reducing the search space. Another crucial step to finding the optimal solution is to encourage the exploration of the search space. We can achieve that by regularizing our training objective with an entropy term. We further propose a lower variance estimator of the entropy by leveraging the tree structure and sampling in the tree without replacement. We demonstrate experimentally that these three design features are essential to successfully parsing images into programs in a reasonable time frame.

Program synthesis has been a growing interest to researches in machine learning [Bal+17; SPS18; Dev+17; ZW18]. Prior works on converting images to programs are more directly related to our work [Sha+18; Ell+19; Tia+19; Ell+18]. The setup of our work closely follows [Sha+18] which was

*chenghuz@andrew.cmu.edu

Workshop on Perception as Generative Reasoning, NeurIPS 2019, Vancouver, Canada.

extended by [Ell+19]. Both works rely on supervised pretraining heavily before using REINFORCE to fine tune the models [Tia+19] incorporated a differentiable renderer into the learning pipeline while we treated our renderer as an external procedure independent from the learning process. [Ell+18] used a neural network to extract shapes from hand-drawn sketches, formulating the grammatical rules as constraints and obtaining the final program by solving a constraint satisfaction problem. This process can be very computationally expensive compared to a pure neural-network appraoch.

2 Problem Definition

The input of the model is an image constructed from basic shapes such as triangle, circle or triangle, each with a designated size and location. The output of the model is a program to reconstruct the input in the format of context-free grammar (CFG).

In this paper, we will use constructive solid geometry (CSG) image [Hub90] as an example for presentation. The sample images can be found in Appendix A.1. The CFG for CSG includes the binary shape operations of union, subtraction and intersection. The context-free grammar rules are as follows:

$$S \to E; \qquad E \to EET \mid P; \qquad T \to + \mid - \mid *; \qquad P \to SHAPE_1 \mid \cdots \mid SHAPE_n. \tag{1}$$

S, T, and P are non-terminal symbols for start, operations, and shapes, respectively. The operations are $+$, $*$ and $-$ which represent union, intersection and subtraction, respectively.

3 Proposed Algorithm

Our model consists of a CNN encoder for reading the canvas, an embedding layer for the action tokens and an RNN for generating the grammatical program sequences. The model parameters θ are trained with entropy regularized REINFORCE [Wil92] updates, $\Delta\theta \propto \mathbb{E}_{s\sim p_\theta(s)}[\nabla_\theta \log p(s) f(s)] + \alpha\nabla_\theta \mathcal{H}(s)$, where $\mathcal{H}(x)$ is the entropy of the sequence s for exploration and $f(s)$ is a user-defined reward function.

The output program is converted to an image by the non-differentiable renderer. The renderer outputs blank images for invalid program inputs. The generated image is compared to the target image and a reconstruction-based reward is computed. Our reward function consists of the Chamfer distance and a pixel-wise distance between the two images, $R(\mathbf{x}, \mathbf{y}) = \max(\delta, (1 - \frac{Ch(\mathbf{x},\mathbf{y})}{d})^\gamma + \frac{\sum \mathbf{x} \cap \mathbf{y}}{\sum \mathbf{x}})$, with $\mathbf{x}$ representing the original image and $\mathbf{y}$ representing the generated image, d the canvas diagonal length, and $\gamma = 20$ in this setting. We cropped the reward falling under $\delta = 0.3$ because the reward value is no longer informative.

Our setup is quite similar to [Sha+18] so far. However, several additional components were key to our model's success, which we detail below. The naive extension from [Sha+18] does not learn to generate correct grammatical programs as is. We therefore introduce a grammar-encoded tree LSTM to ensure a valid output sequence. The tree LSTM also relies on an image stack to provide intermediate images. Finally, we introduce a lower-variance entropy estimator to improve the estimation of our objective.

3.1 Grammar-Encoded Tree LSTM

There are three types of actions (non-terminals) involved in the grammatical program generation: shape selection (P), operation selection (T), and grammar selection (E). Grammar selection in this problem setting includes $E \to EET$, and $E \to P$ as in the grammar definition (1), which decides whether the program will keep expanding or not.

Let the set of shapes to be $\mathcal{P}$, the set of operations to be $\mathcal{T}$ and the set of grammar tokens to be $\mathcal{G}$. In the proposed tree LSTM, the output space of the model is of size $|\mathcal{P}| + |\mathcal{T}| + |\mathcal{G}|$. A stack is used here for each example to keep track of the program. All stacks start with a start symbol S and an end symbol \$. For each symbol popped from the stack, its corresponding valid output space only includes the actions on the right hand side of the grammar rules (1). We therefore mask the invalid actions by adding a vector to the layer before softmax. The vector has a very large negative number in the entry corresponding to the invalid actions. This makes sure that grammatically invalid options will have close to zero probability of being sampled. The corresponding output space of an end symbol only

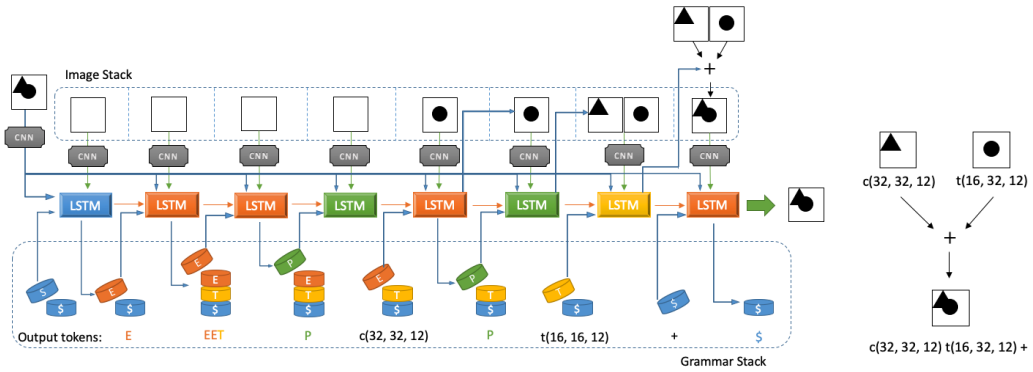

Figure 1: Left) Grammar-encoded tree LSTM at work. The top layer of canvases demonstrates the image stack and the bottom layer demonstrates the grammar stack at each time step. The tilted token in each time step represents the action popped and the letters below the tokens are the actions pushed. (Right) formulation of a CSG image example.

consists of the end symbol so that no more tokens will be produced. Leaf tokens (various shapes and operations) will not be added to the stack as they do not contribute to the program structure. Final programs that become the inputs to the renderer consist of shape and operation tokens only, with all grammatical tokens having been substituted out.

Whenever the output of the model is a shape token, the image of the shape will be pushed onto the corresponding image stack. When the output is an operation token, the top two images are popped from the image stack, the operation is applied on them, and the resulting image is pushed back onto the stack. The input of the LSTM at each time step includes the encoded target image, the encoded intermediate images from the image stack, the embedded token popped from grammar stack, and the hidden state from the LSTM's last time step. An example is shown in Figure 1.

3.2 Exploration

We adopt two features to encourage exploration – entropy regularization and sampling without replacement [KVW19].

The sampling without replacement algorithm chooses the top k branches to expand based on the $G_{\phi_{i,j}}$ score at time step j. The $G_{\phi_{i,j}}$ score is sampled from Gumble($\phi_{i,j}$), where $\phi_{i,j}$ is the log probability of i-th partial sequence at time step j, conditioned on its parent's $G_{\phi_{i,j-1}}$ score being the maximum. See Algorithm 1 for details in the branching process.

The sampling without replacement algorithm requires correct scaling of the objective functions to ensure unbiasness. The scaling term is $\frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)}$. $p_\theta(s^i)$ represents the probability of the sequence s^i and S represents the set of all sampled sequences s^i for $i = 1, 2, \cdots, K$. κ is the $(k+1)$-th largest $G_{\phi_{i,j}}$ score among all the possible branches and $q_{\theta,\kappa}(s^i) = 1 - \exp(-\exp(\phi_{i,j} - \kappa))$. Additional normalization terms $W^i(S) = W(S) - \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)} + p_\theta(s^i)$, $W(S) = \sum_{i \in S} \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)}$ are employed to reduce variance but they increase bias in the final estimation. The exact objective is as follows [KHW19]:

$$\nabla_\theta \mathbb{E}s \sim p_\theta(s)[f(s)] \approx \sum_{s^i \in S} \frac{1}{W^i(S)} \cdot \frac{\nabla_\theta p_\theta(s^i)}{q_{\theta,\kappa}(s^i)} (f(s^i) - \frac{B(S)}{W(S)}) \tag{2}$$

The baseline term is defined as $B(S) = \sum_{i \in S} \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)} f(s^i)$. Incorporating a baseline into the REINFORCE objective is a standard practice in order to reduce variance in the estimation.

Entropy estimation uses a similar scaling scheme as the REINFORCE objective. Let the sequence s^i consist of elements $X_1, X_2, X_3, \cdots, X_n$. We can calculate the entropy at each step and the entropy of a sequence can be decomposed as a summation of entropy at each step:

$$\mathcal{H}(X_1, X_2, X_3, \cdots, X_k) \approx \sum_{j=1}^{n} \frac{1}{W_j(S)} \sum_{s^i \in S} \frac{p_\theta(s^i_j)}{q_{\theta,\kappa}(s^i_j)} \mathcal{H}(X_j | X_1 = x_1, X_2 = x_2, \cdots, X_{j-1} = x_{j-1}) \tag{3}$$

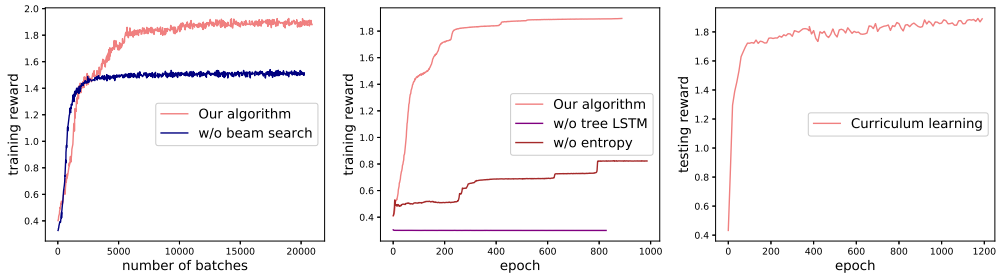

Figure 2: This shows the advantages of our algorithm design. Left: Demonstrating the effect of the beam search on the training reward per batch. With beam search (200 examples per batch with 4 beams), the algorithm's performance approaches the optimal reward while the algorithm is stuck in a local optimum without it (800 examples per batch). Middle: Demonstrating the effect of tree LSTM structure and entropy regularization. The algorithm struggles to generate valid grammar without tree LSTM structure and it learns much slower without entropy regularization. Right: The algorithm learns to generate longer program with curriculum learning.

where $W_j(S) = \sum_{s^i \in S} \frac{p_\theta(s^i_j)}{q_{\theta,\kappa}(s^i_j)}$ and s_j denotes the first j elements of the sequence s. Excluding the $\frac{1}{W_j(S)}$ term, the estimator is unbiased. The normalization term reduces the variance of the estimator.

4 Experiments

We used a synthetic dataset to test our algorithm and assess the effect of each algorithm component. The algorithm chooses from 27 shape actions, 3 operation actions and 2 grammar actions to create an image on a 64×64 canvas. The 27 shape actions have their size and position in the canvas as well as the type of geometry (circle, triangle or square) encoded in the selection. We separated our dataset by the length of the program, which also dictates the number of shapes in the image. Programs of length 5 have 3 shapes and 2 operation actions, those of length 7 have 4 shapes and 3 operation actions, etc. For each length of program, we generated all possible combinations of shape actions and operation actions and then filtered out duplicates and empty images. Images were considered duplicates if no more than 120 pixels differ between the two and are considered empty if there are no more than 120 pixels on the canvas. The training set size for length 5, 7, and 9 programs are 3600, 4800, 12000, respectively and the testing set sizes are 586, 789 and 4830, respectively.

For this dataset, we used 4 beams for each sample to approximate the objective functions. The coefficient for negative entropy is 0.02 and the learning rate is 0.02.

The results of the experiment confirmed our design of the algorithm. We compared each algorithm component's effect on the learning process on the length 7 program dataset. When we train the model without beam search, the training process is not able to get out of a discrete local optimum as seen in Figure 2 (Left). When we take out the entropy term in the objective function, the reward function appears like a staircase with very long transition time between each improvement Figure 2 (Middle). This would make the algorithm very difficult to scale up to problems with bigger actions spaces or longer program length. Without the tree structure, the reward stays around 0.3 which is the lowest possible reward Figure 2 (Middle) because the program is unable to generate a valid program to render. In order to train on the length 9 program dataset, we employed curriculum learning to improve the training result. The test reward is shown in Figure 2 (Right). For some example outputs of the algorithm tested on the length 9 dataset, please refer to 3 in Appendix A.1.

5 Discussion

In this paper, we proposed an entropy regularized REINFORCE-based algorithm with grammar-encoded tree LSTM that leverages grammatical structures to parse a CSG image into context-free grammar. To our knolwedge, we are the first to successfully parse a CSG image with a non-differentiable renderer into CFG using unsupervised learning. We demonstrated the importance of each of our design features. For a future direction, it would be interesting to extend our work into continuous parameterized action spaces.

References

[Hub90] Philip M Hubbard. *Constructive solid geometry for triangulated polyhedra*. 1990.

[Wil92] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[Bal+17] Matej Balog et al. "DeepCoder: Learning to Write Programs". In: *International Conference on Representation Learning (ICLR)*. 2017.

[Dev+17] Jacob Devlin et al. "Robustfill: Neural program learning under noisy I/O". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 990–998.

[Ell+18] Kevin Ellis et al. "Learning to infer graphics programs from hand-drawn images". In: *Advances in neural information processing systems*. 2018, pp. 6059–6068.

[Sha+18] Gopal Sharma et al. "Csgnet: Neural shape parser for constructive solid geometry". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5515–5523.

[SPS18] Richard Shin, Illia Polosukhin, and Dawn Song. "Improving neural program synthesis with inferred execution traces". In: *Advances in Neural Information Processing Systems*. 2018, pp. 8917–8926.

[ZW18] Amit Zohar and Lior Wolf. "Automatic program synthesis of long programs with a learned garbage collector". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2094–2103.

[Ell+19] Kevin Ellis et al. *Write, Execute, Assess: Program Synthesis with a REPL*. 2019. arXiv: `1906.04604 [cs.PL]`.

[KHW19] Wouter Kool, Herke van Hoof, and Max Welling. "Buy 4 REINFORCE Samples, Get a Baseline for Free!" In: (2019).

[KVW19] Wouter Kool, Herke Van Hoof, and Max Welling. "Stochastic Beams and Where To Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement". In: *International Conference on Machine Learning*. 2019, pp. 3499–3508.

[Tia+19] Yonglong Tian et al. "Learning to infer and execute 3d shape programs". In: *arXiv preprint arXiv:1901.02875* (2019).

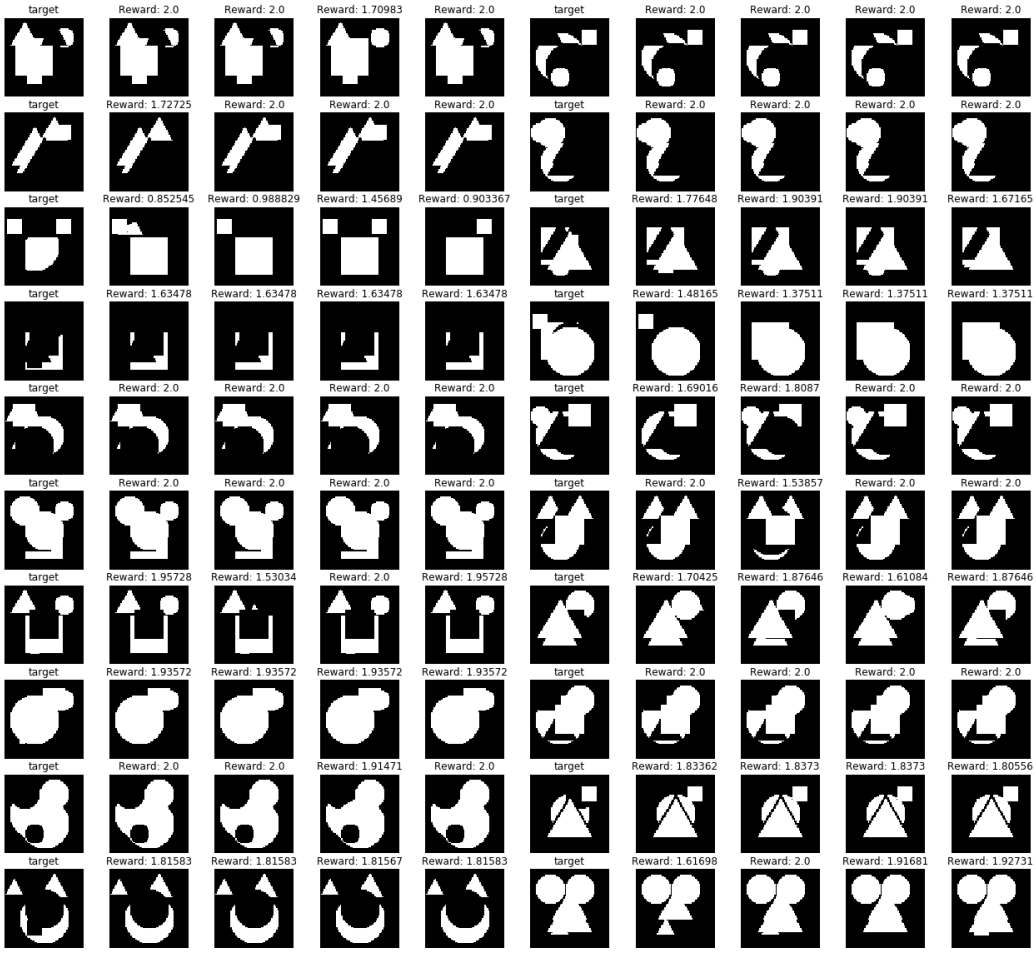

Figure 3: Some example outputs of our algorithm. On each row there are two examples each occupying 5 columns. The leftmost images of the five columns are the target images and the four images to its right are the output of four beams. Its corresponding reward is on top of each image

A Appendix

A.1 Example Output

Please refer to 3 for some example output of the algorithm.

A.2 Algorithm

Algorithm 1 Beam Search Tree LSTM

Input: Target Images T, Number of beams K
Initialize: Grammar stack S, Image stack I, Beam set $\mathbb{B}$

Encode the target image $T_{encoded} \leftarrow Encode(T)$
$\mathbb{B} = \{\mathbf{s}^i | \mathbf{s}^i = \emptyset\}$
$\mathcal{H}(v) = 0$
for $j := 0$ *to* n **do**
  for $i := 1$ *to* k **do**
    Pop the grammar stack $g \leftarrow S_i.pop()$
    Embed the grammar symbol $g_{embedded} \leftarrow Embed(g)$
    Encode the image stack $I_{encoded} \leftarrow Encode(I)$
    $Hidden_j \leftarrow LSTM(g_{embedded}, I_{encoded}, T_{encoded}, Hidden_{j-1})$
    $\mathbf{p_{i,j}} \leftarrow softmax(f(Hidden_j) + Mask(g))$
    Estimate entropy at this node $v_{i,j}$: $\mathcal{H}(v_{i,j}) = \mathbf{p_{i,j}} \cdot \log \mathbf{p_{i,j}}$
    Initialize $\tilde{\mathbf{G}}_{j+1} = \emptyset$
    Calculate the log probability of the beam including the children $\boldsymbol{\phi}_{i,j+1} = \mathbf{1} \cdot \phi_{i,j} + \log \mathbf{p_{i,j}}$
    Sample the Gumbel scores $\mathbf{G}_{\boldsymbol{\phi}_{i,j+1}} \sim$ Gumbel$(\boldsymbol{\phi}_{i,j+1})$
    $Z_{i,j+1} = \max(\mathbf{G}_{\boldsymbol{\phi}_{i,j+1}})$
    $\tilde{\mathbf{G}}_{\phi_{i,j+1}} \leftarrow -\log(\exp(-\mathbf{1} \cdot G_{\phi_{i,j}})) - \exp(-\mathbf{1} \cdot Z_{i,j+1}) + \exp(-\mathbf{G}_{\boldsymbol{\phi}_{i,j+1}})$
    Add the values in $\tilde{\mathbf{G}}_{\boldsymbol{\phi}_{i,j+1}}$ to $\tilde{\mathbf{G}}_{j+1}$: $\tilde{\mathbf{G}}_{j+1} \leftarrow \tilde{\mathbf{G}}_{j+1} \cup \tilde{\mathbf{G}}_{\boldsymbol{\phi}_{i,j+1}}$
  end for
  Estimate the entropy $\mathcal{H}(v) \leftarrow \mathcal{H}(v) + \frac{1}{W_j(S)} \sum_j \frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j)} \mathcal{H}(v_{i,j})$
  Choose top k values in $\tilde{\mathbf{G}}_{j+1}$ to expand
  Update $\mathbb{B} = \{\mathbf{s}^i | \mathbf{s}^i = \mathbf{s}^i \cup v_{i,j+1}$ for the chosen $i, j+1\}$
end for
Render the images x's for the sequences $\mathbf{s}^i$'s
Generate the rewards of the images $f(x)$'s
Maximize the following objective: $\mathbb{E}_{x \sim p_\theta(x)}[f(x)] - \alpha \mathcal{H}(v)$ as defined in (2) and (3)