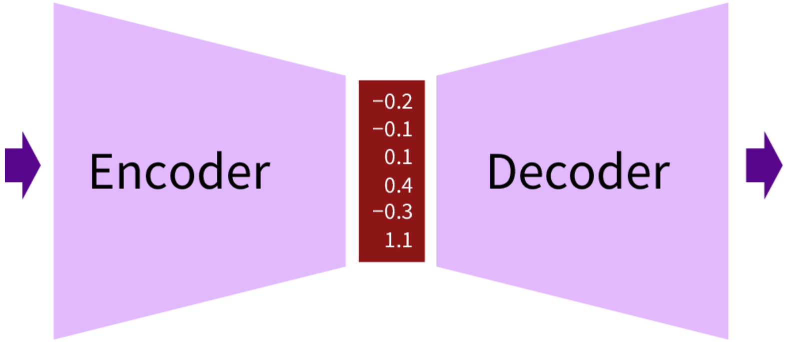
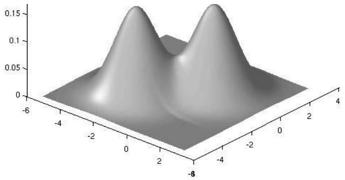
Learning to Infer and Execute 3D Shape Programs

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, Jiajun Wu

Motivation

- **Unstructured “neural” representations**





Under constrained

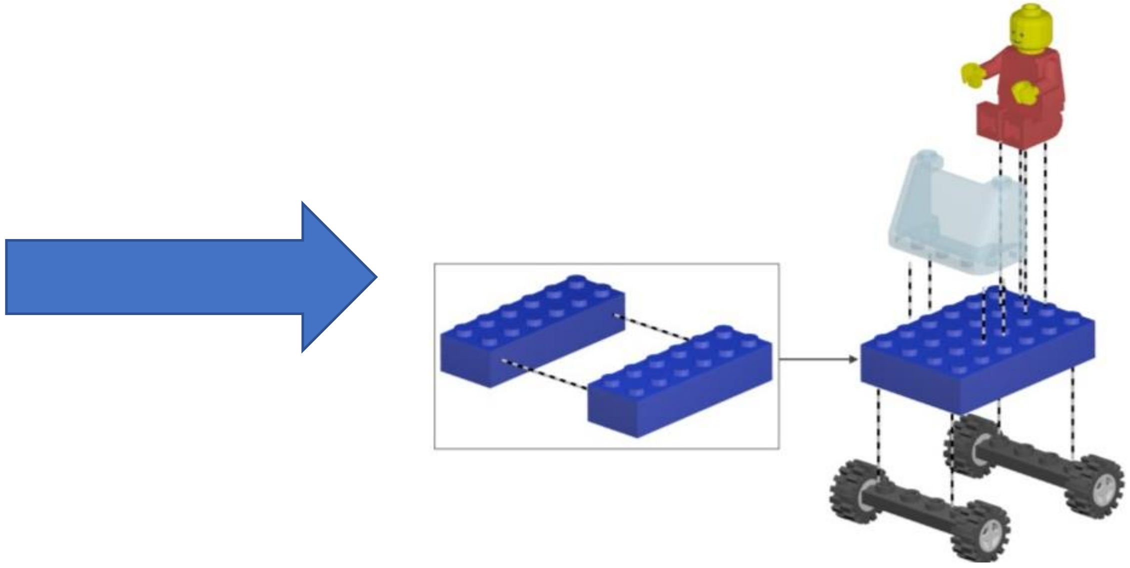
Not interpretable

What is the correct representation??

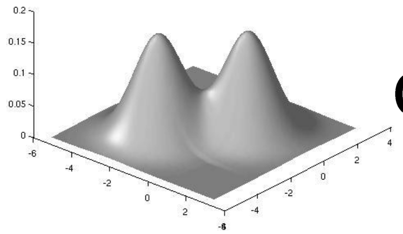
Idea of vision as "inverse graphics"

- Humans can easily understand visual input
  - Decomposition of a scene into objects
  - Decomposition of objects into components
  - Infer object relationships etc. etc.

More than just pixels in
2D image!



Taken from "Part-based modelling of compound scenes from images", van den Hengel, Anton, et al.



OR

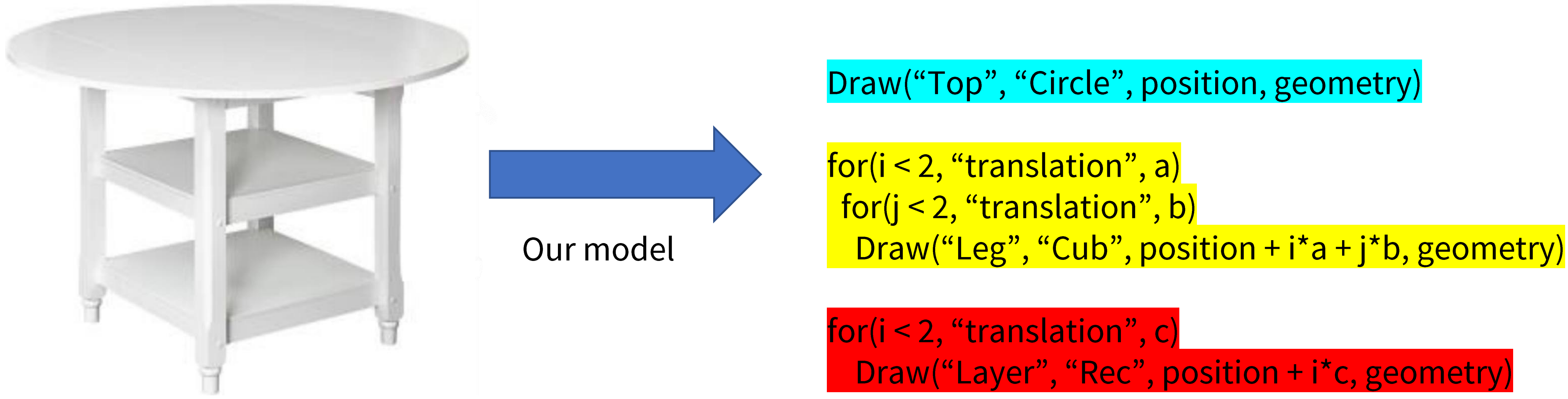
```
Draw(“Top”, “Circle”, pos0, shape_param0)
Draw(“Leg”, “Cub”, pos1, shape_param1)
Draw(“Leg”, “Cub”, pos2 , shape_param2)
Draw(“Leg”, “Cub”, pos3 , shape_param3)
Draw(“Leg”, “Cub”, pos4 , shape_param4)
Draw(“Layer”, “Rec”, pos5, shape_param5)
Draw(“Layer”, “Rec”, pos6, shape_param6)
```

OR

```
Draw(“Top”, “Circle”, position, geometry)

for(i < 2, “translation”, a)
 for(j < 2, “translation”, b)
   Draw(“Leg”, “Cub”, position + i*a + j*b, geometry)

for(i < 2, “translation”, c)
   Draw(“Layer”, “Rec”, position + i*c, geometry)
```


Our model
Draw(“Top”, “Circle”, position, geometry)
for(i < 2, “translation”, a)
for(j < 2, “translation”, b)
Draw(“Leg”, “Cub”, position + i*a + j*b, geometry)
for(i < 2, “translation”, c)
Draw(“Layer”, “Rec”, position + i*c, geometry)

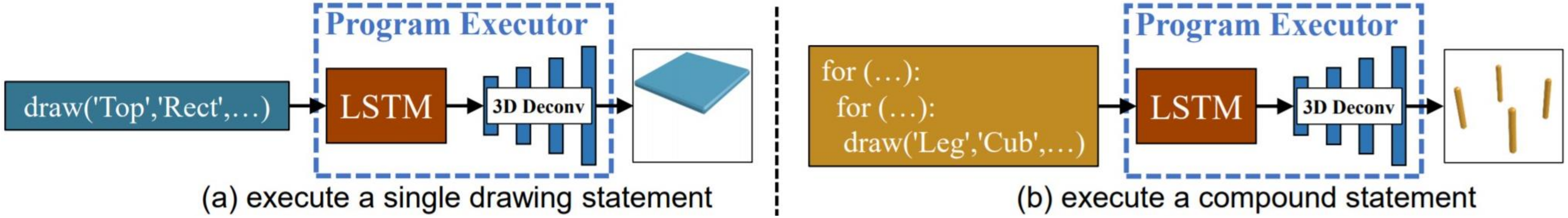
Challenges

- No dataset of sufficiently high quality exists for part based decomposition of 3D objects
  - Roll your own “synthetic” dataset

- We have templates for chairs, tables
  - Can be fuzzed for diversity
  - Hard codes loops for translation & rotation (think office chair base)
  - Possibly disjoint from “real” data

Model Components:

1. Program rendering module (program executor)
   Takes in a block of code, uses RNN + 3d deconv to output voxels



Trained purely on synthetic data

Model Components:

2. Program output LSTM

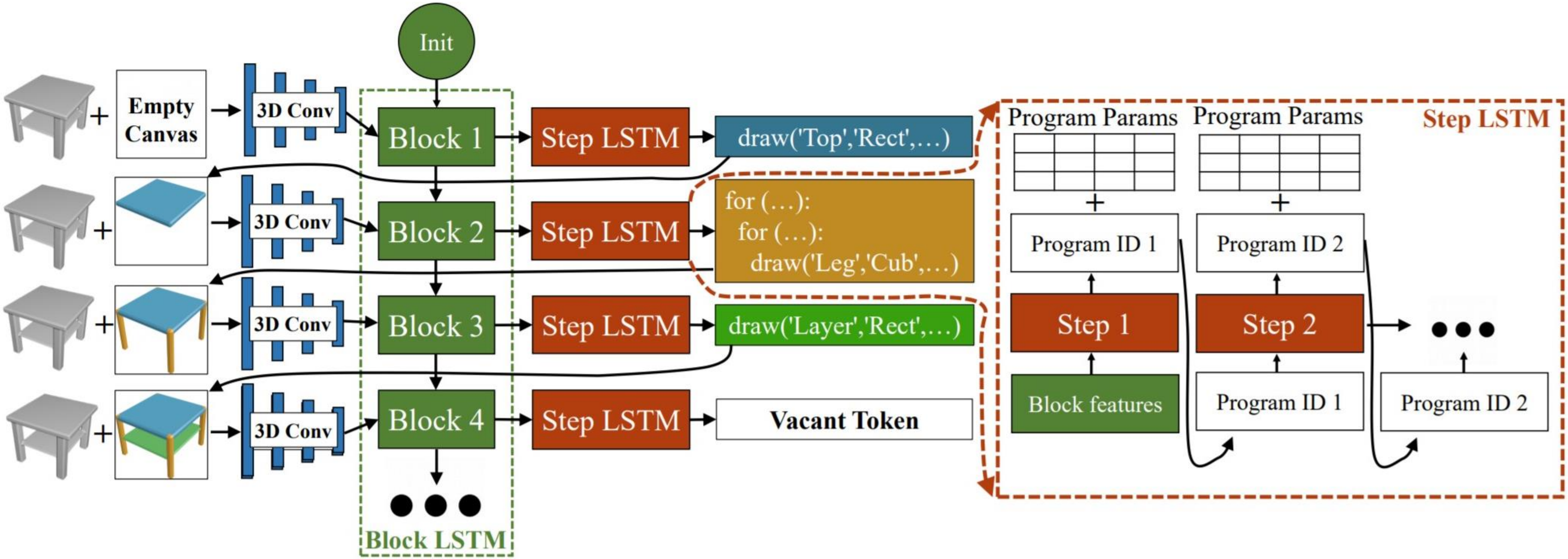
    Takes in:

    - input voxels
    - canvas (to keep track of already rendered voxels)

Actually two LSTMs, one of them inside the other one

First LSTM identifies blocks of voxels that belong together

Second LSTM takes the features produced, and outputs programs



Init
Empty Canvas
3D Conv
3D Conv
3D Conv
3D Conv
Block 1
Block 2
Block 3
Block 4
Block LSTM
Step LSTM
Step LSTM
Step LSTM
Step LSTM
draw('Top','Rect',…)
for (…):
for (…):
draw('Leg','Cub',…)
draw('Layer','Rect',…)
Vacant Token
Step LSTM
Program Params
Program Params
Program ID 1
Program ID 2
Step 1
Step 2
Block features
Program ID 1
Program ID 2

Why do we even need the executor

- Program type can be trained via cross-entropy
- Program parameters can be trained via L2
- To visualize we can just run the programs right??

Fine tuning (Guided adaptation)

- Synthetic data might be different from real data
- Remember, program executor is a neural network
- Neural network = differentiable

For a given ground truth object:

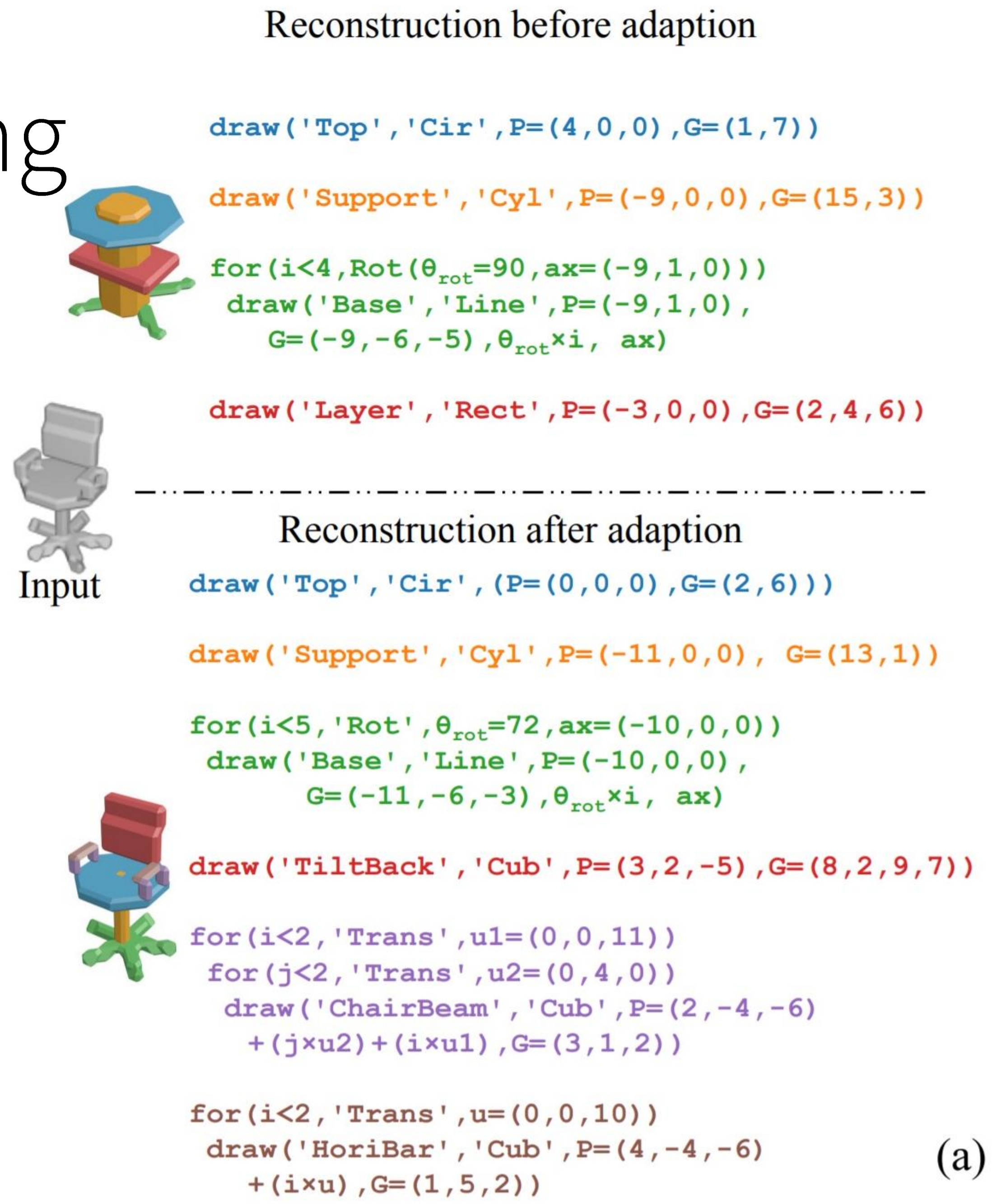
1. Generate programs
2. Execute, and backprop loss to program generator to tune
3. Repeat to get better at a given ground truth object

Results:

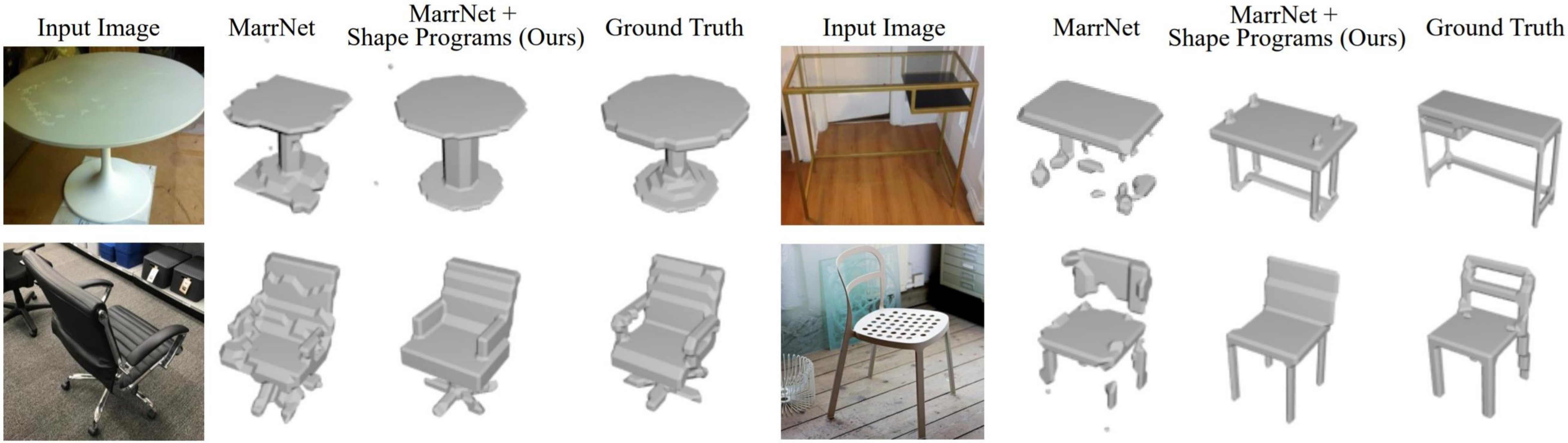
Models	IoU ↑		CD ↓		EMD ↓	
	table	chair	table	chair	table	chair
CSGNet-original	0.111	0.154	0.216	0.175	0.205	0.177
Tulsiani et al. (2017)	0.357	0.406	0.083	0.079	0.073	0.072
CSGNet-augmented	0.406	0.365	0.072	0.077	0.069	0.076
Nearest Neighbour	0.445	0.389	0.083	0.084	0.084	0.084
Shape Programs w/o GA	0.487	0.422	0.067	0.072	0.063	0.072
Shape Programs	**0.591**	**0.516**	**0.058**	**0.063**	**0.056**	**0.060**

Models	Stable (%)		Conn. (%)		Stable & Conn. (%)	
	table	chair	table	chair	table	chair
Tulsiani et al. (2017)	36.7	31.3	37.1	**68.9**	15.4	19.6
Shape Programs w/o GA	94.7	95.1	76.6	54.2	73.7	51.6
Shape Programs	**97.0**	**96.5**	**78.4**	**68.5**	**77.0**	**66.0**
Ground Truth	98.9	97.6	98.8	97.8	97.7	95.5

Before and after fine tuning

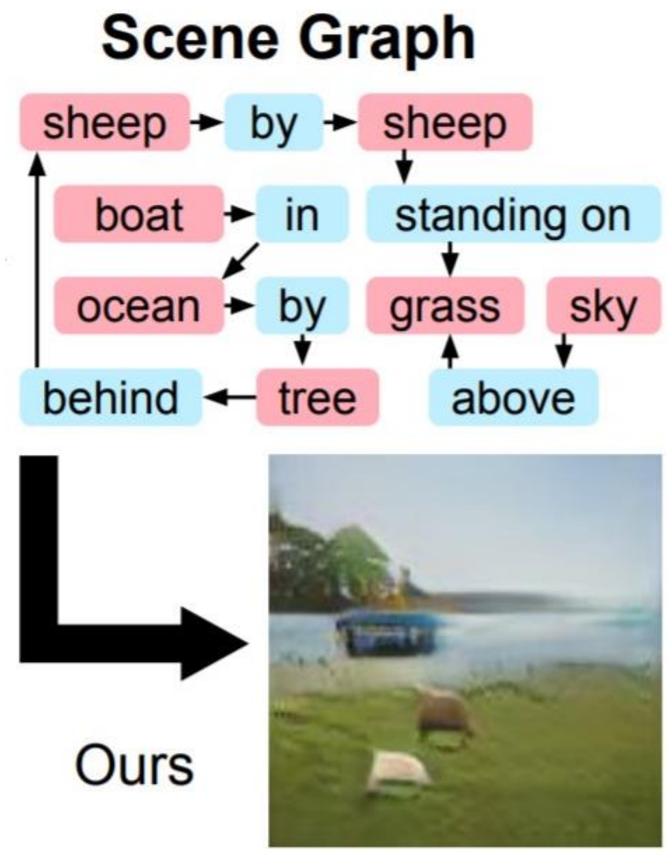


Constrained representation on reconstruction



Project 2

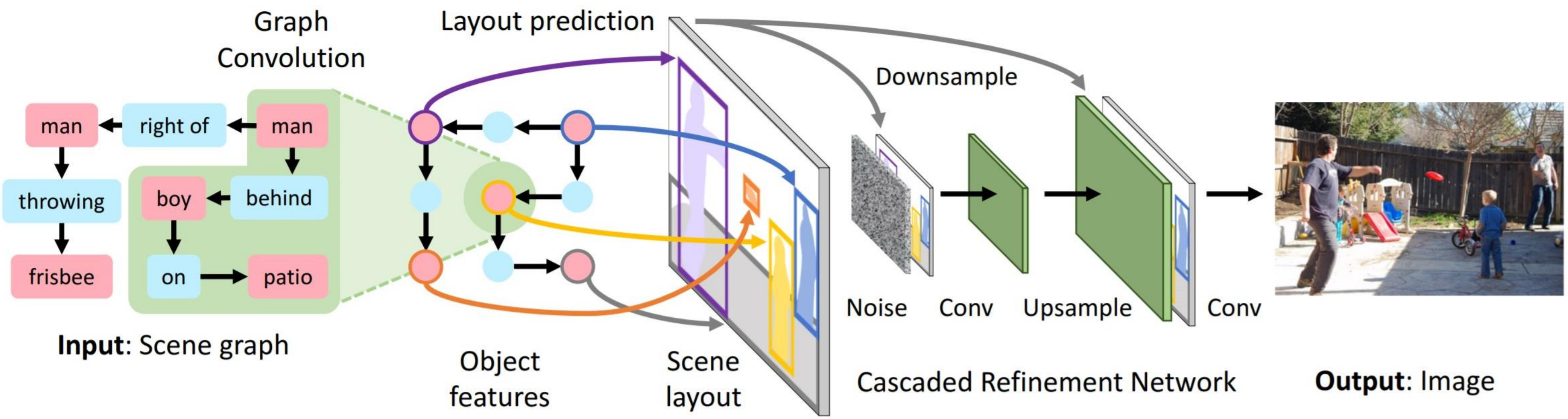
Object relationships can represented with graph

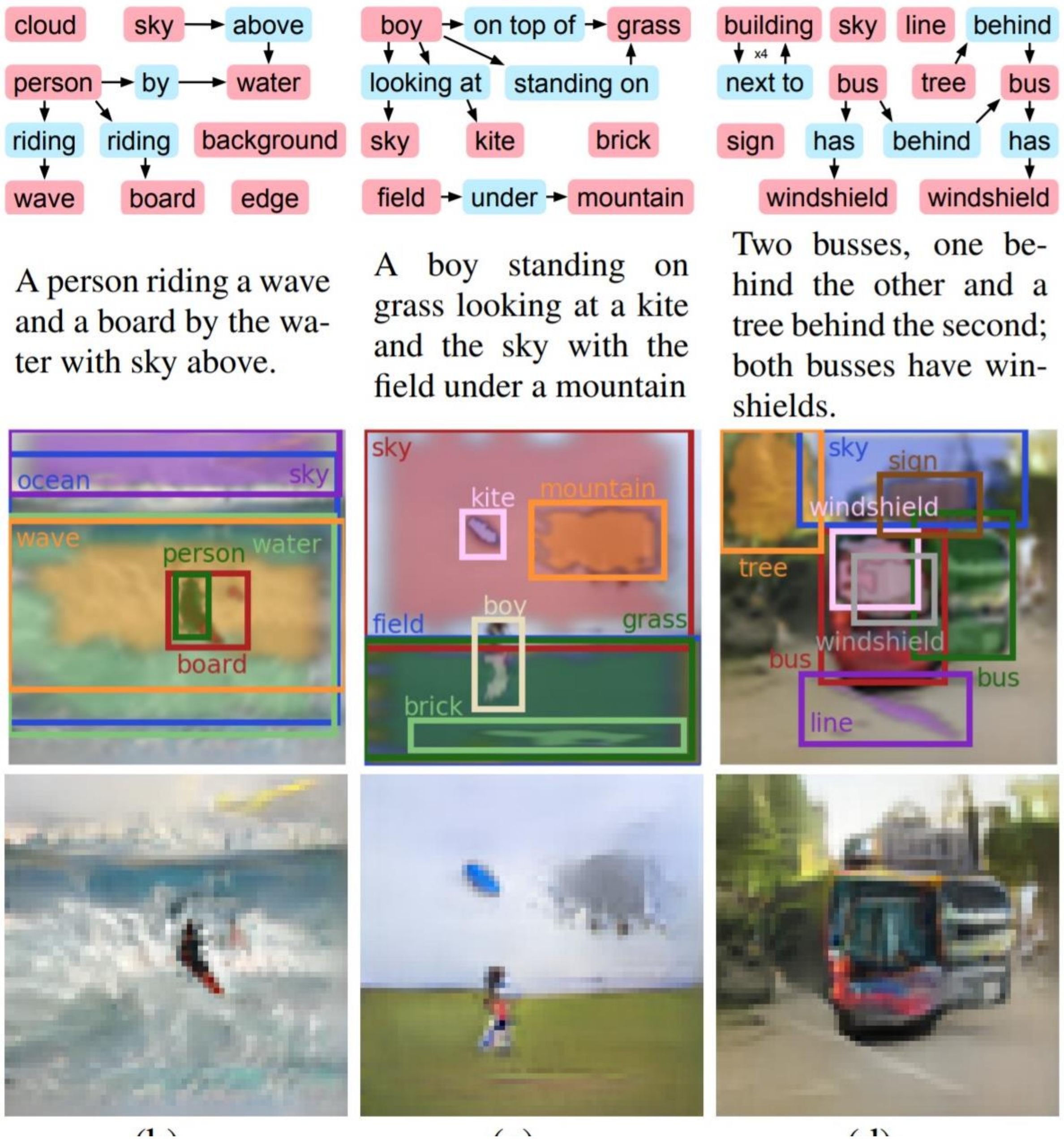


Taken from “Image Generation from Scene Graphs” by Johnson et al.

Previous approaches

- Image generation from scene graph (Johnson) – CVPR 2018

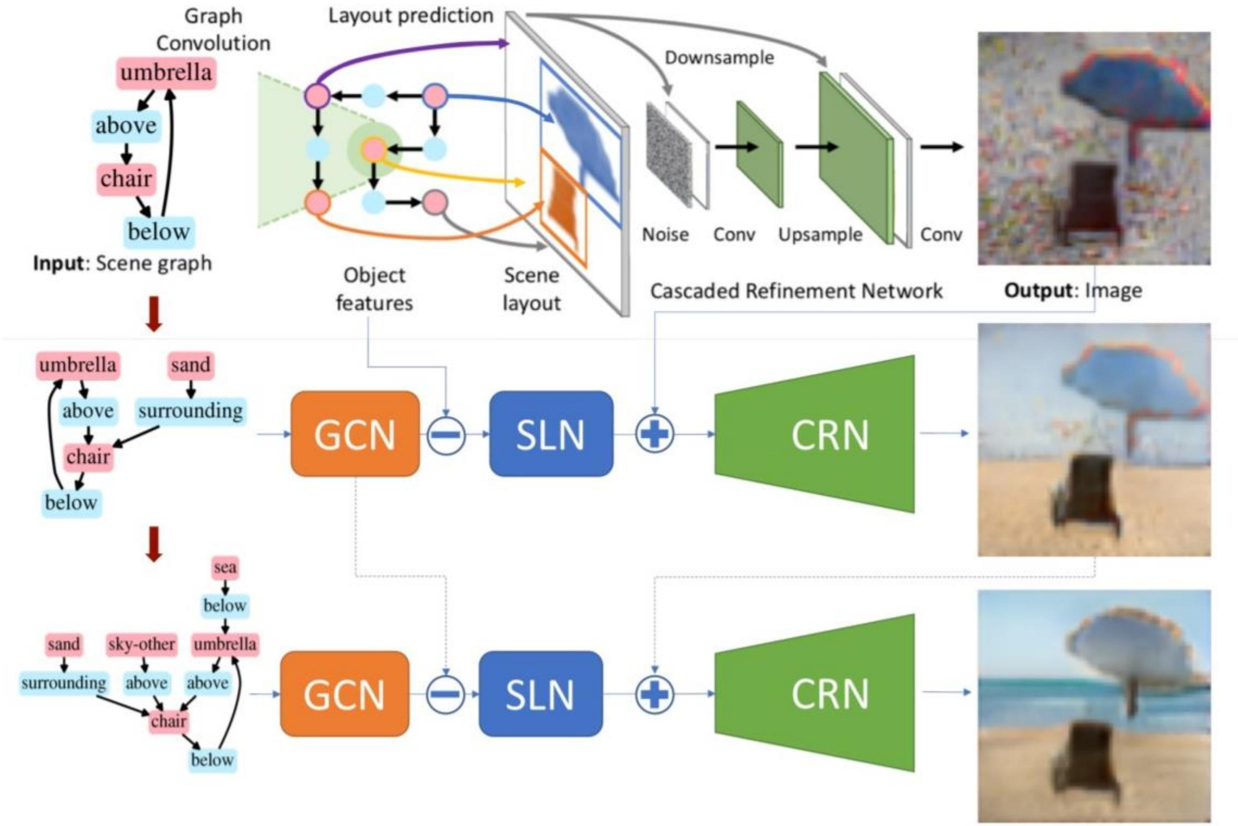




cloud
sky
above
person
by
water
riding
riding
background
wave
board
edge
A person riding a wave and a board by the water with sky above.
boy
on top of
grass
looking at
standing on
sky
kite
brick
field
under
mountain
A boy standing on grass looking at a kite and the sky with the field under a mountain
building
sky
line
behind
x4
next to
bus
tree
bus
sign
has
behind
has
windshield
windshield
Two busses, one behind the other and a tree behind the second; both busses have winshields.
ocean
sky
wave
water
person
board
sky
kite
mountain
field
boy
grass
brick
sky
sign
windshield
tree
windshield
bus
bus
line

Previous approaches

- Interactive Image Generation Using Scene Graphs – ICLR 2019 workshop



Bonus Paper:

MAML: Model-Agnostic Meta-learning

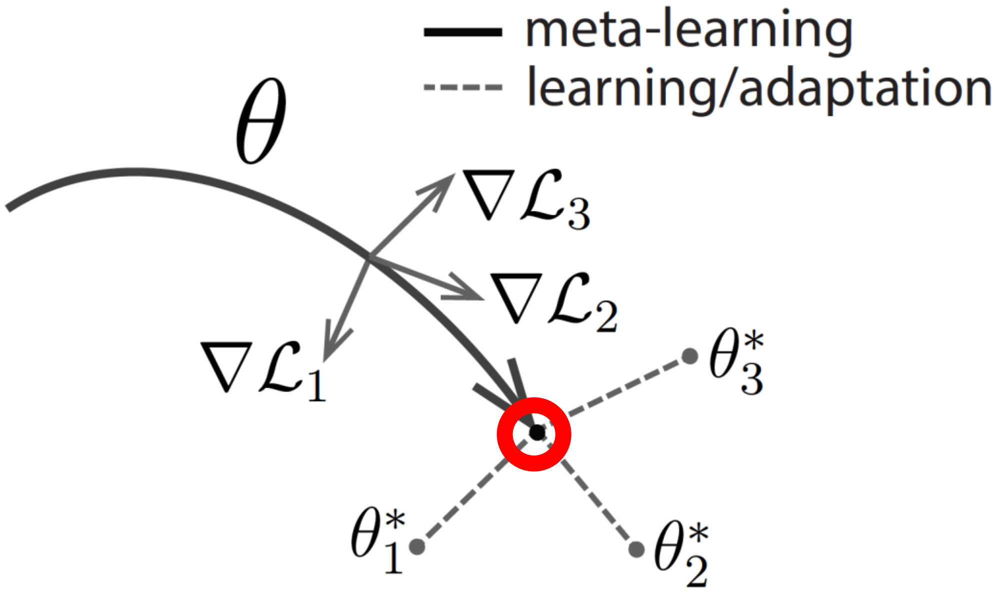
Meta Learning

- Learning to learn
  - Optimize model for fast adaptation

- Closely related to few shot learning
- Should be able to adapt to new tasks easily
- Reuse learned concepts
- See “Human-level concept learning through probabilistic program induction” by Tenenbaum et al.

MAML

- Optimization weights for better generalization
- Applies to models trained via gradient descent
- Requires higher order gradients support
  - Pytorch (broke on multi-GPU since 1.0, fixed in 1.2), Tensorflow
  - Mxnet work in progress, probably within the next 2 versions (just a guess)



Approach

- Explicitly optimize θ such that the following is minimized:

$$\sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

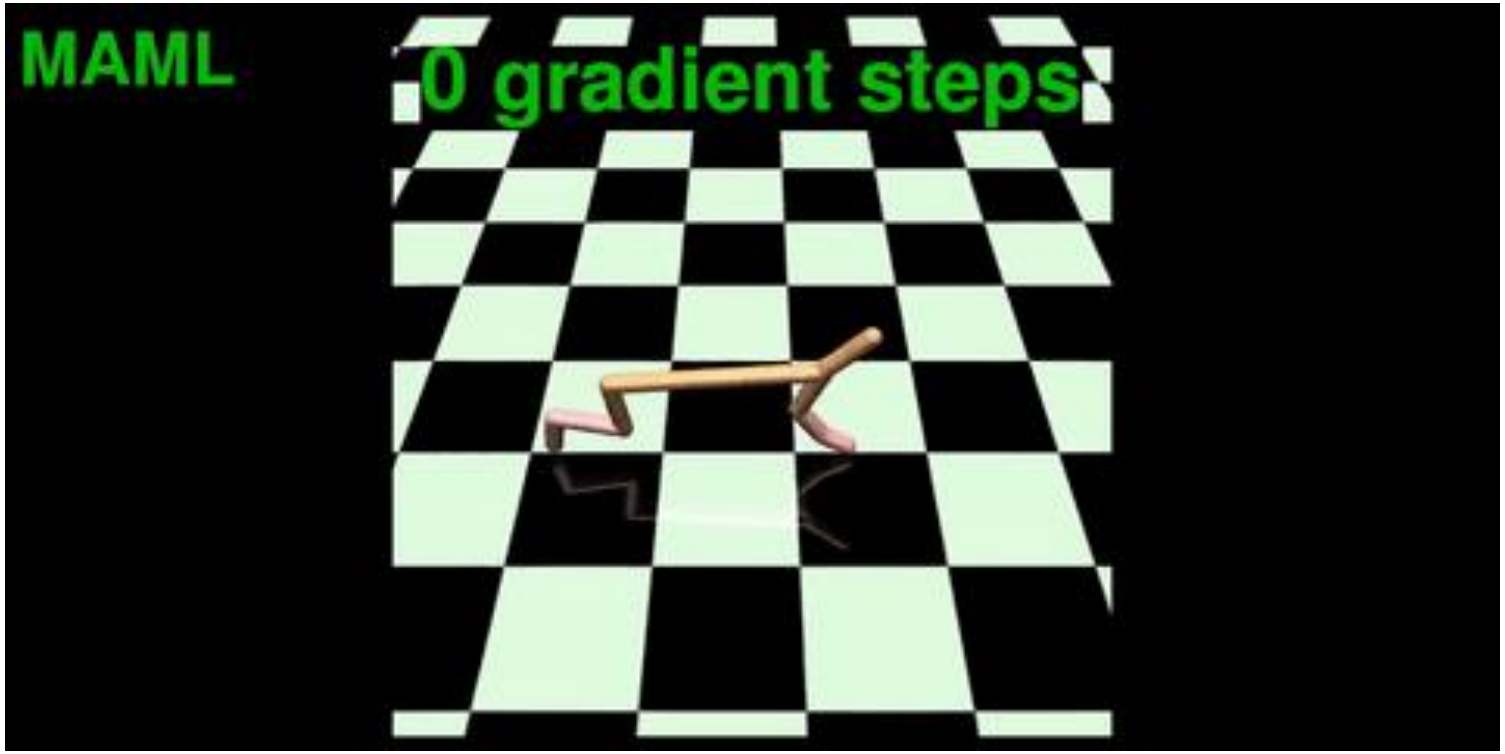
- Note: $p(\mathcal{T})$: distribution over tasks

  α: **Task learning rate (think of this as the fine tuning amount)**

Approach

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters

randomly initialize θ
while not done **do**
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ **do**
        Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
        Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while


MAML
0 gradient steps

Bonus-bonus paper

- Reptile: A Scalable Meta- Learning Algorithm
  - Faster
  - First order approximation