

# Modular Meta Learning

Ferran Alet, Tomas Lozano-Perez, Leslie P. Kaelbling (MIT CSAIL)

<https://arxiv.org/abs/1806.10166>

Presented by Ankesh Anand

- Combines ideas from **Neural Module Networks** (J. Andreas et al, 2015), and Model Agnostic Meta Learning aka **MAML** (C. Finn et al, 2017).

### Deep Compositional Question Answering with Neural Module Networks

Jacob Andreas Marcus Rohrbach Trevor Darrell Dan Klein  
 Department of Electrical Engineering and Computer Sciences  
 University of California, Berkeley  
 {jda,rohrbach,trevor,klein}@{cs,eecs,eecs,cs}.berkeley.edu

#### Abstract

Visual question answering is fundamentally compositional in nature—a question like where is the dog? shares substructure with questions like what color is the dog? and where is the cat? This paper seeks to simultaneously exploit the representational capacity of deep networks and the compositional linguistic structure of questions. We describe a procedure for constructing and learning neural module networks, which compose collections of jointly-trained neural “modules” into deep networks for question answering. Our approach decomposes questions into their linguistic substructures, and uses these structures to dynamically instantiate modular networks (with reusable components for recognizing dogs, classifying colors, etc.). The resulting compound networks are jointly trained. We evaluate our approach on two challenging datasets for visual question answering, achieving state-of-the-art results on both the VQA natural image dataset and a new dataset of complex questions about abstract shapes.

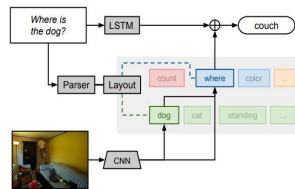


Figure 1: A schematic representation of our proposed model—the shaded gray area is a neural module network of the kind introduced in this paper. Our approach uses a natural language parser to dynamically lay out a deep network composed of reusable modules. For visual question answering tasks, an additional sequence model provides sentence context and learns common-sense knowledge.

### Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Chelsea Finn<sup>1</sup> Pieter Abbeel<sup>1,2</sup> Sergey Levine<sup>1</sup>

#### Abstract

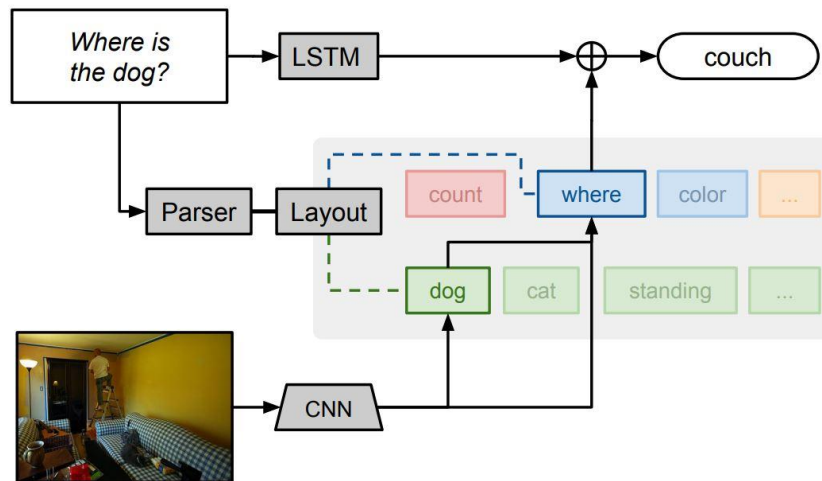
We propose an algorithm for meta-learning that is model-agnostic, in the sense that it is compatible with any model trained with gradient descent and applicable to a variety of different learning problems, including classification, regression, and reinforcement learning. The goal of meta-learning is to train a model on a variety of learning tasks, such that it can solve new learning tasks using only a small number of training samples. In our approach, the parameters of the model are explicitly trained such that a small number of gradient steps with a small amount of training data from a new task will produce good generalization performance on that task. In effect, our method trains the model to be easy to fine-tune. We demonstrate that this approach leads to state-of-the-art performance on two few-shot image classification benchmarks, produces good results on few-shot regression, and accelerates fine-tuning for policy gradient reinforcement learning with neural network policies.

the form of computation required to complete the task.

In this work, we propose a meta-learning algorithm that is general and model-agnostic, in the sense that it can be directly applied to any learning problem and model that is trained with a gradient descent procedure. Our focus is on deep neural network models, but we illustrate how our approach can easily handle different architectures and different problem settings, including classification, regression, and policy gradient reinforcement learning, with minimal modification. In meta-learning, the goal of the trained model is to quickly learn a new task from a small amount of new data, and the model is trained by the meta-learner to be able to learn on a large number of different tasks. The key idea underlying our method is to train the model's initial parameters such that the model has maximal performance on a new task after the parameters have been updated through one or more gradient steps computed with a small amount of data from that new task. Unlike prior meta-learning methods that learn an update function or learning rule (Schmidhuber, 1987; Bengio et al., 1992; Andrychowicz et al., 2016; Ravi & Larochelle, 2017), our algorithm does not expand the number of learned parameters nor place constraints on the model architecture (e.g. by requiring a recurrent model (Santoro et al., 2016) or a

# Recap: Neural Module Networks

- First analyze each question with a **semantic parser**.
- The output of the semantic parser is then used to determine which “modules” to use.
- Modules are assembled and then jointly-trained.



# Hypothesis

- A modular compositional structure can provide a useful basis for **transferring** learned competence from previous tasks to new tasks.
- Can enable learning from a **small amount** of training data in the new task

# Broad Picture

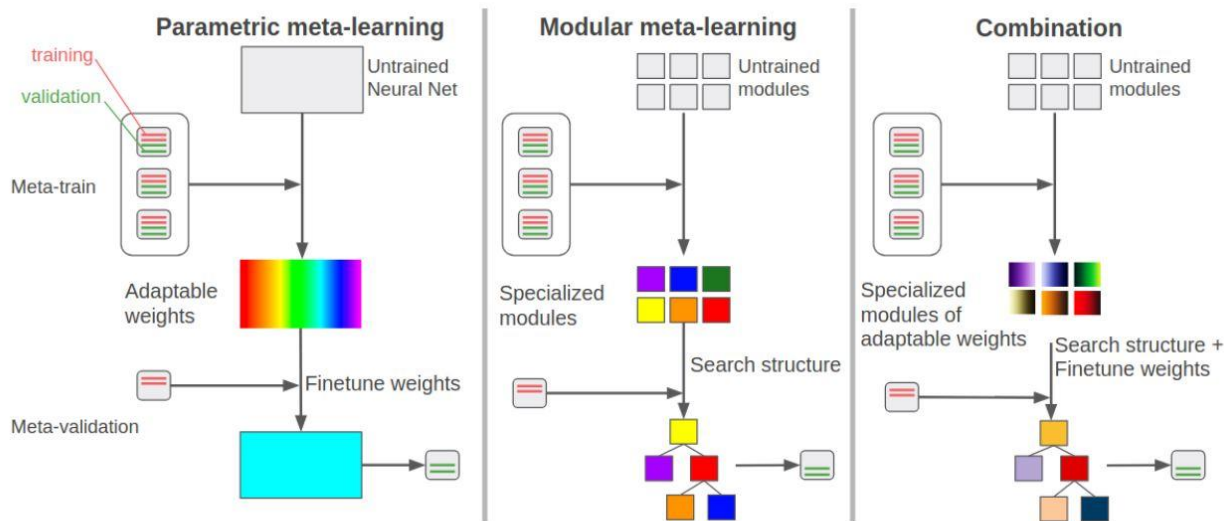


Figure 1: All methods train on a set of related tasks and obtain some flexible intermediate representation. Parametric strategies such as MAML (left) learn a representation that can be quickly adjusted to solve a new task. Our modular meta-learning method (middle) learns a repertoire of modules that can be quickly recombined to solve a new task. A combination of MAML and modular meta-learning (right) learn initial weights for modules that can be combined and adapted for a new task.

# Module Terminology

- **F** is a basis set of modules, which are functions  $f_1, f_2, \dots, f_k$

Each function has a parametric form  $y = f_i(x; \theta_i)$

All the modules are **feed-forward** neural networks.

- **C** is a compositional scheme for forming complex functions using simple modules.

Examples:

\* **Single module:**  $h(x) = f_i(x)$

\* **Fixed compositional structure:**  $h(x) = f_i(x) + f_j(x)$        $h(x) = f_i(f_j(x))$

\* **Weighted ensemble**

# How does it work?

## Two Phases:

- Meta-Learning Phase

$$\Theta = (\theta_1, \dots, \theta_k)$$

- Take tasks  $\{1, 2, \dots, k\}$  as inputs and **generate weights** for each module.
- The objective is to construct modules that will work together as good building blocks for future tasks.

- Final Adaptation Phase

- Find the best way to compose these modules on the final task.

$$S_{\Theta} = \arg \min_{S \in \mathcal{S}} e(D_{final}^{train}, S, \Theta)$$

# Final Learning Phase

$$S_{\Theta} = \arg \min_{S \in \mathcal{S}} e(D_{final}^{train}, S, \Theta)$$

- Module Parameters are fixed, we just need to find the right structure  $S$  (the way to combine these modules). Uses **simulated annealing**.

```
procedure FINAL( $D_{final}^{train}, \mathcal{S}, \Theta, T_0, \Delta_T, T_{end}$ )  
   $S$  = random simple structure from  $\mathcal{S}$   
  for  $T = T_0$ ;  $T = T - \Delta_T$ ;  $T < T_{end}$  do  
     $S' = \text{PROPOSE}_{\mathcal{S}}(S, \Theta)$   
    if ACCEPT( $e(D, S', \Theta), e(D, S, \Theta), T$ ) then  $S = S'$   
  return  $S$   
procedure ACCEPT( $v', v, T$ )  
  return  $v' < v$  or  $\text{rand}(0, 1) < \exp\{(v - v')/T\}$ 
```



# Meta-Training Phase

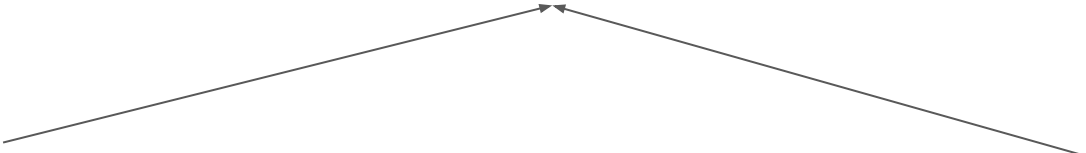
- Objective is to find **parameter values**  $\Theta$  that constitute a set of modules that can be recombined to effectively solve each of the training tasks.
- Uses **validation sets** for the meta-training tasks to avoid choosing  $\Theta$  in a way that over-fits.
- Thus training objective is to find  $\Theta$  that minimizes the average generalization performance

$$J(\Theta) = \sum_{j=1}^m e(D_j^{val}, \arg \min_{S \in \mathcal{S}} e(D_j^{train}, S, \Theta), \Theta) \quad .$$

# Algorithm

(for the meta-training phase)

```
procedure BOUNCEGRAD( $\mathcal{S}, D_1^{train}, \dots, D_m^{train}, D_1^{val}, \dots, D_m^{val}, \eta, T_0, \Delta_T, T_{end}$ )  
   $S_1, \dots, S_m =$  random simple structures from  $\mathcal{S}$ ;  $\Theta =$  neural-network weight initialization  
  for  $T = T_0$ ;  $T = T - \Delta_T$ ;  $T < T_{end}$  do  
    BOUNCE( $S_1, \dots, S_m, D_1^{train}, \dots, D_m^{train}, T, \mathcal{S}, \Theta$ )  
    GRAD( $\Theta, S_1, \dots, S_m, D_1^{val}, \dots, D_m^{val}, \eta$ )
```



```
procedure BOUNCE( $S_1, \dots, S_m, D_1^{train}, \dots, D_m^{train}, T, \mathcal{S}, \Theta$ )  
  for  $j = 1 \dots m$  do  
     $S'_j = \text{Propose}_{\mathcal{S}}(S_j, \Theta)$   
    if  $\text{Accept}(e(D_j^{train}, S'_j, \Theta), e(D_j^{train}, S_j, \Theta), T)$  then  $S_j = S'_j$ 
```

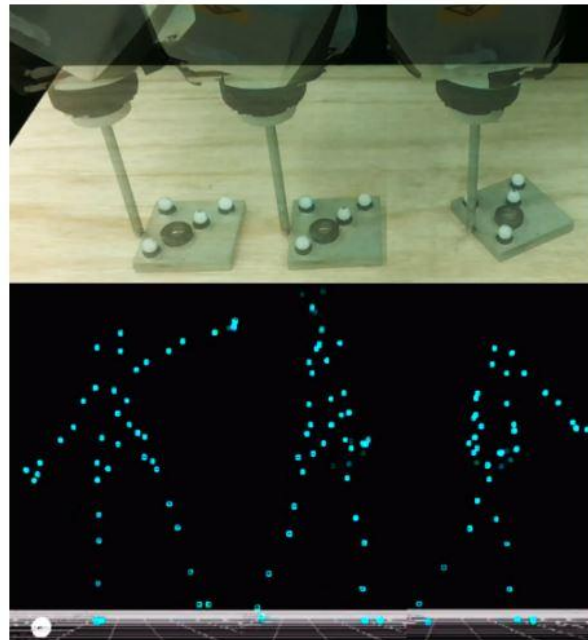
```
procedure GRAD( $\Theta, S_1, \dots, S_m, D_1^{val}, \dots, D_m^{val}, \eta$ )  
   $\Delta = 0$   
  for  $j = 1 \dots m$  do  
     $(x, y) = \text{rand\_elt}(D_j^{val})$ ;  $\Delta = \Delta + \nabla_{\Theta} L(S_{j\Theta}(x), y)$   
   $\Theta = \Theta - \eta \Delta$ 
```

# Experiments

- Sine wave Regression Task (from the MAML paper)

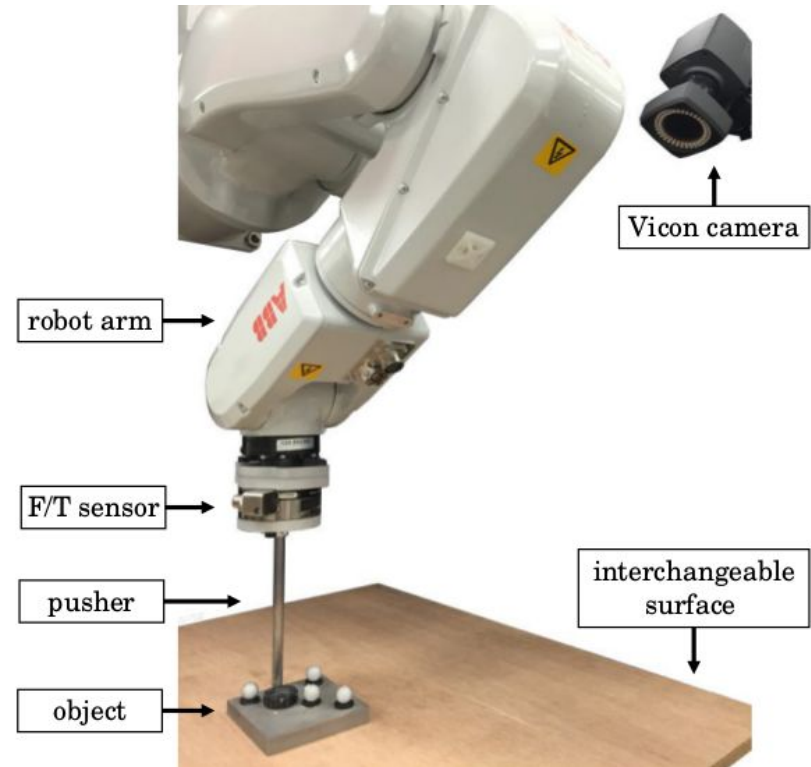
$$\sin(ax + b)$$

- MIT Push DataSet
- Berkley MoCap dataset



# MIT Push Dataset

- Given position of the object (2d), orientation of the object (1d), position of the pusher (2d), velocity of the pusher (2d).
- Predict the 3-d change in the object's position and orientation
- 11 different objects with different shapes on 4 surfaces with different friction properties.



# Results

Dataset	BIGNET	MAML	BOUNCEGRAD	MOMA	Structure
Parametrized sines	98.1	26.5	32.5	<b>19.8</b>	composition
Sum of functions (1-4pts)	32.7	19.7	<b>12.8</b>	18.0	sum
Sum of functions (16pts)	31.9	8.0	<b>0.4</b>	<b>0.4</b>	sum
MIT push: known objects	21.5	18.5	16.7	<b>14.9</b>	attention
MIT push: new objects	18.4	<b>16.9</b>	<b>17.1</b>	<b>17.0</b>	attention
Berkeley MoCap: known actions	35.7	35.5	<b>32.2</b>	<b>31.9</b>	concatenate
Berkeley MoCap: new actions	79.5	77.7	77.0	<b>73.8</b>	concatenate

Table 1: Summary of results; lower is better; bold results are not significantly different from best.

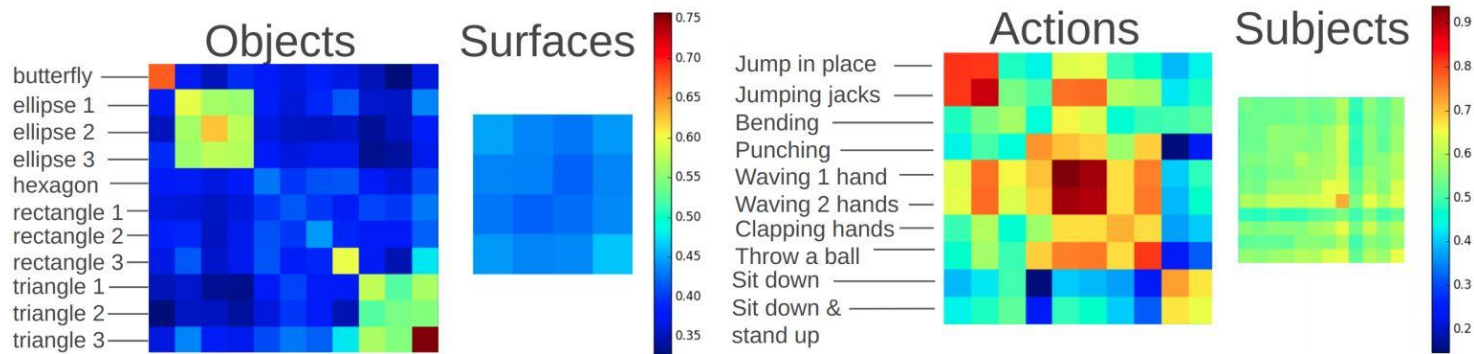


Figure 5: Shared modules show internal structure of the datasets. In particular, less important factors (surfaces and subjects) don't change the structure and big changes (objects and actions) do. Within the structural changes, there is more sharing between conceptually similar datasets.

# Remarks

- No experiments on **RL tasks** yet, would be cool to see whether modular policies are better at adapting to new tasks (intuitively, they should).
- The compositional scheme of how to combine modules is pre-defined. So, applying this method requires a bit of **hand-engineering**, would be nice to see this restriction lifted.