Modular meta-learning

Ferran Alet, Tomás Lozano-Pérez, Leslie P. Kaelbling
MIT Computer Science and Artificial Intelligence Laboratory
{alet,tlp,lpk}@mit.edu

Abstract: Many prediction problems, such as those that arise in the context of robotics, have a simplifying underlying structure that could accelerate learning. In this paper, we present a strategy for learning a set of neural network modules that can be combined in different ways. We train different modular structures on a set of related tasks and generalize to new tasks by composing the learned modules in new ways. We show this improves performance in two robotics-related problems.

1 Introduction

In many situations, such as robot-learning, training experience is very expensive. One strategy for reducing the amount of training data needed for a new task is to learn some form of prior or bias using data from several related tasks. The objective of this process is to extract information that will substantially reduce the training-data requirements for a new task. This problem is a form of transfer learning, sometimes also called meta-learning or "learning to learn" [1, 2].

Previous approaches to meta-learning for robotics have focused on finding distributions over [3] or initial values of [4, 5] parameters, based on a set of "training tasks," that will enable a new "test task" to be learned with many fewer training examples. Our objective is similar, but rather than focusing on transferring information about parameter values, we focus on finding a reusable set of modules that can form components of a solution to a new task, possibly with a small amount of tuning.

Modular approaches to learning have been very successful in structured tasks such as natural-language sentence interpretation [6], in which the input signal gives relatively direct information about a good structural decomposition of the problem. We wish to address problems that may benefit from a modular decomposition but do not provide any task-level input from which the structure of a solution may be derived. Nonetheless, we adopt a similar modular structure and parameter-adaptation method for learning our reusable modules, but use a general-purpose simulated-annealing search strategy to find an appropriate structural decomposition for each new task.

We provide an algorithm, called BOUNCEGRAD, which learns a set of modules and then combines and tunes them appropriately for a new task. We demonstrate its effectiveness by comparing it to MAML [4], a recent meta-learning method, on a set of regression tasks that represent the types of prediction-learning problems that are faced by robotics systems, and show that BOUNCEGRAD indeed achieves better prediction performance from a few training examples, and can be much faster to train. In addition, we show that this modular approach offers a strategy for explaining learned solutions to new tasks: by observing the modules that are used in a solution to a new task, we can relate this task to previous tasks that use the same modules. This approach also offers opportunities for verification and validation: the modules discovered during meta-learning may be subjected to computationally expensive analytical or empirical validation techniques off-line; they can then be recombined without tuning to address new tasks, generating solutions that can be validated more efficiently as compositions of previously understood modules.

2 Related Work

Our work draws primarily from two sources: multi-task meta-learning and modular networks. Prominent examples of meta-learning in robotic domains are MAML [4] and follow-up work [5, 7]. They perform "meta-training" on a set of related tasks with the goal of finding network weights that serve as a good *starting point* for a few steps of gradient descent in each task. Others [8, 9, 10, 11, 12] perform different types of parametric changes in the network's computation conditioned on few ex-
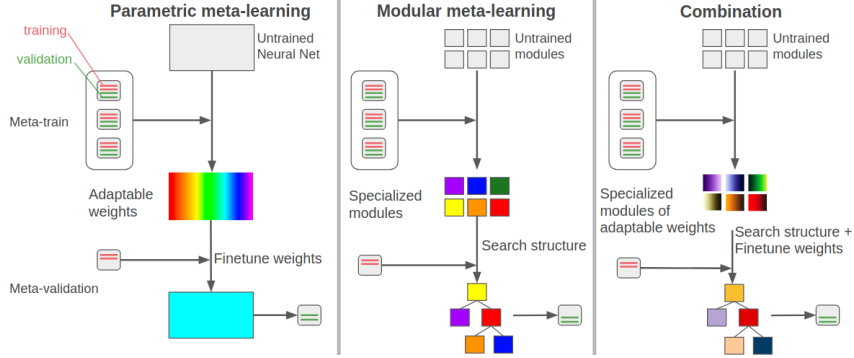
Preprint; work in progress.

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.

amples. We adapt the same basic setting, but rather than finding good starting weights, we find a good set of modules for later structural combination; see figure 1. Neural module networks [6] provide an elegant mechanism for training a set of individual modules that can be recombined to solve new problems, when the input has enough signal to guess an appropriate modular decomposition. Johnson et al. [13] later showed that the structure controller could be trained with RL; others applied similar frameworks to get more interpretability [14] or to generalize across hierarchical robotic tasks with neural programs[15]. However, as far as we know, this framework has not been applied in problems where the input does not give enough information about an appropriate structure.

Structured networks have been used for meta-learning in the reinforcement-learning setting. Devin et al. [18] use a fixed composition of two functions, one related to the robot and one to the task. Frans et al. [19] jointly optimize a master policy and a set of sub-policies (options) that can be used to solve new problems; this method can be seen as having a single fixed scheme of combination via the master policy; it is in contrast to our ability to handle a variety of computational compositions. PATHNET [20] is closely related to our work. The architecture is layered, with several modules in each layer. An evolutionary algorithm determines gates on the connections among the modules. After training on an initial task, the weights in the modules that contribute to the solution to that task are frozen, and then the architecture is trained on a second task. If the tasks are sufficiently related, the modules learned in the first task can be directly re-used to make learning more efficient in the second task. Meyerson and Miikkulainen [21] and later Liang et al. [22] expanded these ideas to the multitask setting with two particular compositional schemes: soft layer ordering and routing in DAGs. We propose a general framework of which these are two important subcases. Moreover, we operate in the metalearning setting; where, with few points per task, it is very easy to prematurely optimize the structure and run into local optima, as shown in figure 2. Therefore, we believe using Simulated Annealing rather than gradient descent[21] or irreversible evolution[22] may be a better fit for our setting. On the other hand, optimizing module architectures[22] could improve our method.

3 Modular meta-learning

We explore the problem of modular meta-learning in the context of *supervised learning* problems, in which training and validation sets of input-output pairs are available. Such problems arise in robotics, particularly in learning to predict a next state based on information about previous states and actions. We believe that techniques of a similar type can be applied to RL problems as well, but do not explore that in this paper. We use the same meta-learning problem formulation as Finn et al.[4] used to apply MAML to supervised learning problems. We assume an underlying distribution $p(\mathcal{T})$ over tasks: a task is a joint distribution $P_{\mathcal{T}}(x, y)$ over (x, y) pairs. The learning problem is: Given data drawn from m training tasks and a (small) amount of data drawn from a *final* task, where all $m + 1$ tasks are drawn from $P(\mathcal{T})$, find a hypothesis h from some given set that incurs low expected loss on unseen data drawn from the final task. In this formulation, each task is characterized by two data sets, D^{train} and D^{val}, each of which consists of a set of input-output pairs (x, y). We assume a shared differentiable loss function $L(\hat{y}, y)$ on true vs predicted outputs for all tasks.

3.1 Structured hypotheses

We begin by defining a family of structured hypothesis classes. Given the specification of a basis set of modules and a composition rule, $(\mathcal{C}, F, \Theta)$ represents a set of possible functional input-output mappings that will serve as the hypothesis space for the final learning task. F is a basis set of modules, which are functions $f_1, \ldots, f_k$; each function has a parametric form $y = f_i(x; \theta_i)$ where θ_i is a fixed-dimensional vector of parameters. In this work, all the f_i are feed-forward neural networks, potentially with different architectures, and the parameters $\Theta = (\theta_1, \ldots, \theta_k)$ are the weights of the neural networks, which differ among the modules. $\mathcal{C}$ is a *compositional scheme* for forming complex functions from simpler ones, defined by an initial structure and a set of local modification operations on the structure. Some examples include:

- Single module $h(x) = f_i(x)$, as fig. 2. The local modification is to change which module is used.
- A fixed compositional structure, e.g., $h(x) = f_i(x) + f_j(x)$ or $h(x) = f_i(f_j(x))$. The local modifications are to change which module is used for each of the component functions.
- A weighted ensemble, of the same basic form as an attention mechanism:

$$h(x) = \sum_{k=1}^{l} \frac{e^{f_{i_k}(x)}}{\sum_{k'=1}^{l} e^{f_{i_{k'}}(x)}} g_{j_k}(x)$$

  where $i_1, \ldots, i_l$ and $j_1, \ldots, j_l$ are elements of the set $\{1, \ldots, k\}$, picking out which modules to use to play these roles in the network. There are modules of two types: the f_i have a scalar output and the g_i have an output that is the output dimension of the main regression problem. The local modifications are to change which particular f and g modules are used for each role.
- A general function-composition tree, where the local modifications include both changing which f_i is used at each node, as well as adding or deleting nodes from the tree.

Let $\mathcal{S}$ be the set of possible structures and $S \in \mathcal{S}$ be a particular structure, generated by $\mathcal{C}$, including a choice of which particular functions $f_i \in F$ are included in the structure. To formulate a structured-hypothesis model, we must specify the number and parametric forms, but not parameter values, of basis functions, F, and compositional scheme $\mathcal{C}$. This is analogous to specifying the architecture of a deep neural network.

Our approach has two phases: an off-line *meta-learning* phase and an on-line *final learning* phase. In the meta-learning phase, we take training and validation data sets for tasks $1, \ldots, k$ as input and generate a parametrization for each module, $\Theta = (\theta_1, \ldots, \theta_k)$ as output; the objective is to construct modules that will work together as good building blocks for future tasks. In the final learning phase, we take a training data set for the final task as input, as well as $\mathcal{S}$ and Θ; the output is a compositional form $S \in \mathcal{S}$ which includes a selection of modules $f^1 \ldots, f^{m_s}$ to be used in that form (a single element $f_j \in F$ may occur multiple times in S). Since Θ is already specified, the choice of S completely determines a mapping from inputs to outputs; we will abuse notation slightly and write S_Θ to stand for the function from input to output generated by structure S and parameters Θ. We may optionally include a final tuning phase, which will adapt the parameter vectors.

At learning time on the final task, the space of possible structures $\mathcal{S}$ and parameters Θ are fixed, and the objective is to find and return the best structure in $\mathcal{S}$. Define $e(D, S, \Theta)$ to be the loss of the hypothesis S_Θ on data set D: $e(D, S, \Theta) = \sum_{\{(x,y) \in D\}} L(S_\Theta(x), y)$. Then our final hypothesis is

$$S_\Theta = \arg \min_{S \in \mathcal{S}} e(D_{final}^{train}, S, \Theta) \tag{1}$$

The hope is that, by choosing a limited but flexible and appropriate hypothesis space based on previous tasks, a good choice of S_Θ can be made based on a small amount of data in D_{final}^{train}.

At meta-learning time, $\mathcal{S}$ is specified, and the objective is to find parameter values Θ that constitute a set of modules that can be recombined to effectively solve each of the training tasks. We use validation sets for the meta-training tasks to avoid choosing Θ in a way that over-fits. Our training objective is to find Θ that minimizes the average generalization performance of the hypotheses chosen by equation 1 using parameter set Θ.

$$J(\Theta) = \sum_{j=1}^{m} e(D_j^{val}, \arg \min_{S \in \mathcal{S}} e(D_j^{train}, S, \Theta), \Theta) \ . \tag{2}$$
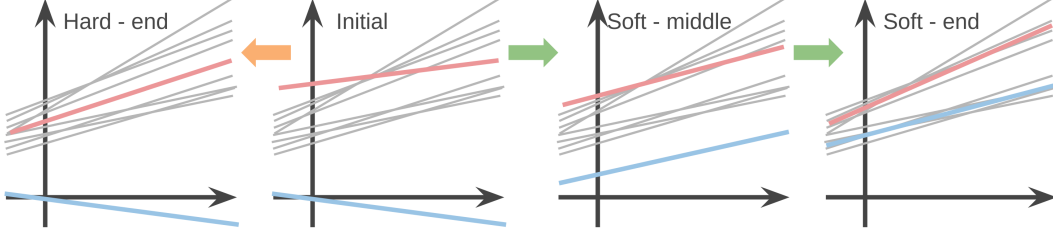
Figure 2: Gray lines are different tasks, each fitted by either the red or blue line (modules). The second frame represents the initial state of a search for parameters. If we make local steps in structure and parameter space, we will converge to the solution on the left, without ever updating the blue. However, if we consider a smoothed criterion with a non-point distribution over structures, we will update the parameters for the blue module and eventually arrive at the solution on the far right.

3.2 Learning algorithm

The optimization problems specified by equations 1 and 2 are in general quite difficult, requiring a mixed continuous-discrete search in a space with many local optima. In this section, we describe the BOUNCEGRAD algorithm, which performs local searches based on a combination of simulated annealing and gradient descent to find approximately optimal solutions to these problems.

3.2.1 Final learning phase

In the final learning phase, we have fixed the parameters Θ and need only find an optimal structure $S \in \mathcal{S}$ according to the objective in equation 1. We use simulated annealing [23] to perform this search: it is a local search strategy that employs stochasticity to avoid local optima and has asymptotic optimality guarantees. We start with an initial structure, then randomly propose structural changes using the local modification operators associated with the compositional scheme $\mathcal{S}$, accept the change if it decreases the error on the task and, with some probability, even if it does not.

> **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 rand(0, 1) $< \exp\{(v - v')/T\}$

In order for simulated annealing to converge, the *temperature* parameter T must be decreased over time. The schedule we use is too quick to satisfy theoretical convergence guarantees, yet is practically effective. Given the training set for the final task, we would run FINAL($D_{final}^{train}, \mathcal{S}, \Theta$) to obtain our hypothesis for that task.

3.2.2 Meta-learning phase

To perform the optimization in equation 2, we might use an algorithm that, in the outer loop, performs optimization over continuous parameters Θ, where the evaluation of Θ consists of running procedure FINAL on each of the training data sets, and evaluating the resulting structural hypotheses using the validation sets. This strategy is ineffective because of the prevalence of bad local optima in the space, as illustrated in figure 2. As in clustering, we can smooth out some local optima by changing the objective function, although we will do so only during search, so our final objective will remain the same. We formulate a smoothed objective

$$\widehat{J}(\Theta, T) = \sum_{j=1}^{m} \mathbb{E}_{S \sim \text{MC}(\mathcal{S}, v(s;\Theta), T)} e(D_j^{val}, S, \Theta) \tag{3}$$

Here, $\text{MC}(\mathcal{S}, v, T)$ is the Markov chain induced by executing the simulated-annealing sampler in the structure space $\mathcal{S}$ using its proposal operator, with score function $v(s; \Theta) = e(D_j^{train}, s, \Theta)$ and *fixed* temperature T. Rather than trying to find the Θ values that work best when we choose the best
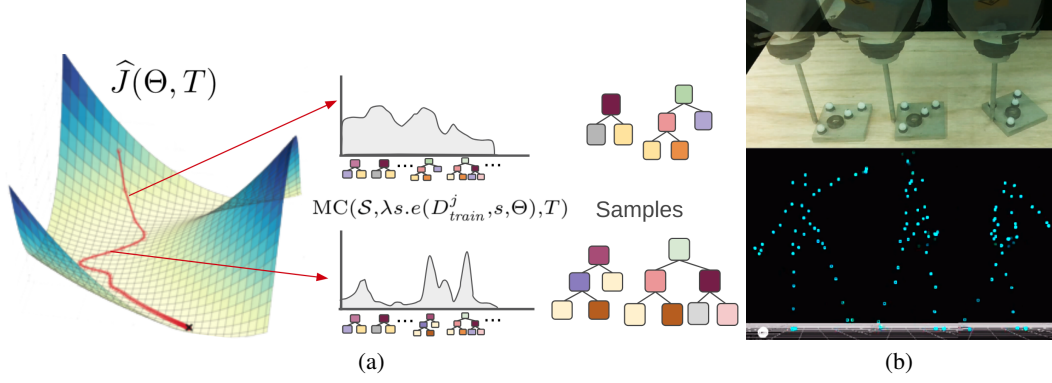
Figure 3: (a) A schematic view of the optimization landscape over Θ and T. At each point, there is a distribution over structures. As temperature decreases, the variance of the distribution decreases and modules become more specialized. (b) benchmarked robotic domains: MIT push dataset [24](top), action (throwing a ball) in the Berkeley MHAD dataset [25] (bottom)

structure S, we will instead try to find Θ values that work best in expectation given the distribution of structures induced by the Markov chain. This space is smoother and less susceptible to local optima. At the same time as we are optimizing Θ via stochastic gradient, we will cool the temperature of the Markov chain. As T approaches 0, the objective $\widehat{J}$ becomes the same as our original objective J.

Given particular structures S_j, then for each task T_j, we know the parametric form of a differentiable feed-forward function that has parameters drawn from Θ, but possibly with parameter tying within and across the tasks due to re-use of the basis functions in F and possibly with some parameters in Θ left unused. We can adjust Θ using stochastic gradient descent, as defined in procedure GRAD. It takes the structures and training data as input, as well as a step size η and performs one step of standard stochastic gradient descent, or any alternative optimizer.

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

However, we do not know the appropriate structure for each task, and so, according to the smoothed criterion in equation 3, we sample structures using a stochastic process based on simulated annealing. We define a procedure BOUNCE that takes a simulated annealing step on a structural hypothesis for each task, using the current Θ values, for a fixed temperature T.

procedure BOUNCE($S_1, \ldots, S_m, D_1^{train}, \ldots, D_m^{train}, T, \mathcal{S}, \Theta$)
    for $j = 1 \ldots m$ **do**
        $S_j' = Propose_\mathcal{S}(S_j, \Theta)$
        if $Accept(e(D_j^{train}, S_j', \Theta), e(D_j^{train}, S_j, \Theta), T)$ **then** $S_j = S_j'$

Finally, we combine these methods into an overall algorithm for optimizing equation 1 via optimizing equation 3 and decaying T.

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

We can think of the state of the optimization algorithm as consisting of both the parameters Θ and the temperature T; these values induce a distribution on structures. The optimization landscape is illustrated in figure 3a. At high temperatures, the distribution over structures is diffuse and the modules will tend to be very generalized. As the temperature decreases, modules can specialize to perform well at the roles they are being selected to play in the structures.

3.3 Parameter tuning in final phase

In the basic FINAL method, we search for the best structure for the new task, without modifying parameters Θ. In fact, in many cases it may be useful to perform some additional parameter optimization as well. One strategy would be to proceed as we have described before, running BOUNCE-GRAD on the training tasks to get Θ, finding the best S for the final task using FINAL, and then taking some gradient steps on Θ, given S, to better optimize loss on D^{train}. However, we can do better by incorporating the objective of MAML more deeply into both FINAL and BOUNCEGRAD, by redefining the inner error function used in the optimization criterion for Θ: instead of using Θ directly, we will evaluate the result of taking one (or a few) gradient steps away from Θ, specialized to optimize D. So, $e_{maml}(D, S, \Theta) = \sum_{\{(x,y) \in D\}} L(S_{O(\Theta,D,S)}(x), y)$, where the *optimized* parameters $O(\Theta, D, S)$ are obtained by a gradient update: $O(\Theta, D, S) = \Theta - \eta \nabla_\Theta e(D, S, \Theta)$. Then, the meta-learning objective becomes

$$J_{maml}(\Theta) = \sum_{j=1}^{m} e(D_j^{val}, \arg\min_{S \in \mathcal{S}} e_{maml}(D_j^{train}, S, \Theta), \Theta) \qquad (4)$$

We can therefore use e_{maml} in place of e in the GRAD and FINAL procedures, and perform a few additional gradient steps on Θ after obtaining the structure from FINAL. We will call this overall algorithm MOMA, MOdular MAml.

4 Experiments

We compare four different learning approaches: training a single network using the combined data from all tasks (BIGNET), training a single network using the MAML criterion, training a modular network without weight adaptation in the final training (BOUNCEGRAD), and training a modular network with MAML adaptation in the final training (MOMA). To make the comparisons as fair as possible, for experiments on a given data set, all networks have the same shape: generally, a feedforward structure of 3–4 layers. We selected a set of layer sizes so that BIGNET and MAML had about 10 times as many parameters as the structured methods, to compensate for BOUNCEGRAD and MOMA having about 10 modules to combine. We also verified that none of the algorithms' performance was sensitive to modest changes in these parameters. We used PyTorch and ADAM[26, 27]; the MAML code was adapted from Rakelly [28]. The code is available on `https://sites.google.com/view/modular-meta-learning`. The target output values y in all data-sets were standardized to have mean 0 and standard deviation 1. The loss function then assigns loss 100 to a mean squared error of 1. More experimental details are available in the supplement.

We tested these methods in three domains: simple functional relationships, predicting the result of a robot manipulator pushing an object, and predicting the next frame of a kinematic skeleton based on previous frames using motion-capture data. The last two domains represent the main motivation for this work: a robot's experience of interacting with various entities in real-world situations should enable it to learn more efficiently about a new entity in a new situation. There is typically some sensible decomposition of the prediction function, but that decomposition is not known in advance. We hope that BOUNCEGRAD can find an appropriate decomposition and that doing so will significantly leverage learning, as well as reveal interesting structure, in the new domain.

An additional advantage of BOUNCEGRAD is computational efficiency. Unlike MAML it does not have extra gradient steps embedded in the inner loop at meta-training time, which offers some efficiency; in addition, forward and backward passes can be done much more efficiently on GPUS by parallelizing over tasks. MAML is generally faster at final training time, since BOUNCEGRAD has to search over structures. However, this training took at most 10 seconds in our examples.

4.1 Functions

We begin by testing on an extended version of the sine-function prediction problem [4], which consisted of data-sets generated from functions of the form $\sin(ax + b)$ for varying values of a and b. The compositional scheme for BOUNCEGRAD is $h(x) = f_i(f_j(x))$; F consists of 20 feedforward neural networks, 10 with 1 hidden layer and 10 with 2. In our experiments in this section MAML and BIGNET use the same architecture as the original MAML experiments. We construct an additional illustrative domain consisting of sums of pairs of common non-linear functions, such
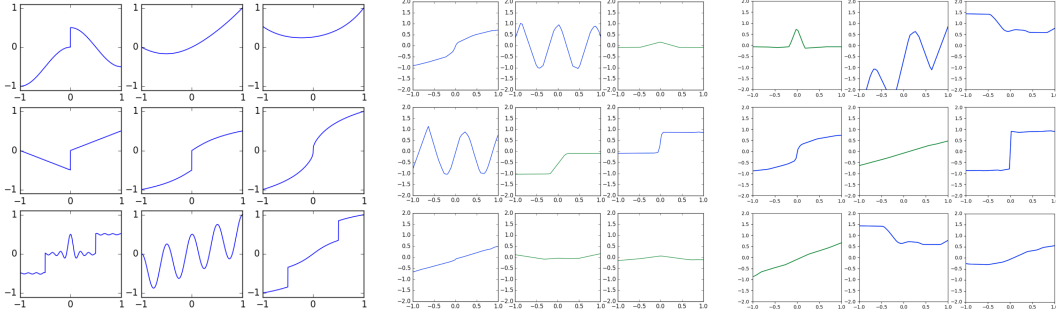
Figure 4: Random functions (left); BOUNCEGRAD (center) and MOMA (right) modules

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

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

as *exp* and *abs*, scaled so the output is contained in the range $[-1, +1]$, generating 16^2 possible prediction tasks. We use 230 randomly selected tasks for meta-training and a different task for testing. Example functions are shown on the left of figure 4. We use the same architectures for this domain as for the sine domain, except that the compositional scheme is $h(x) = f_i(x) + f_j(x)$.

The results are shown in table 1. As expected, training a single network on pooled data from all the tasks (BIGNET) works poorly in all of these domains. In the sine domain, MAML outperforms BOUNCEGRAD because the detailed parameter values are critical to performing well in a new domain, but MOMA significantly improves on both methods, showing that both the structure and gradient meta-learning methods are useful. For sums of functions, we report results in two cases: one in which we average over performance for 1–4 training examples, and one for 16 training examples. With a small amount of final training data, BOUNCEGRAD outperforms other methods because it has the proper structural prior. With more data, all methods improve, but BOUNCEGRAD and MOMA improve on MAML. The plots in the middle and right of figure 4 show some of the basis modules learned by BOUNCEGRAD and MOMA, respectively. Those learned by BOUNCEGRAD are an almost perfect recovery of the actual primitives used to generate the data, even though the algorithm had no information about those functions; MOMA has found similar functional shapes, yet with different values because it can still modify its parameters at final training time.

4.2 Learning to model results of pushing actions

An important sub-problem in robot manipulation is to acquire models of the effects of the robot's motor actions on the objects in the world. The MIT push data-set [24] contains the results of executing pushing actions with a manipulator hand, for 11 different objects with different shapes on 4 surfaces with different friction properties. The behavior of the object on these surfaces is close to quasi-static , so the state can be characterized by an input x consisting of: position of the object (2d), orientation of the object (1d), position of the pusher (2d), velocity of the pusher (2d). Given this 7-d input, the objective is to predict the 3-d change in the object's position and orientation. Each task represents experience with a particular object on a particular surface.

The compositional scheme for BOUNCEGRAD is the weighted ensemble described in section 3.1; F consists of 20 feedforward neural networks, 10 attention modules and 10 regressors. We consider
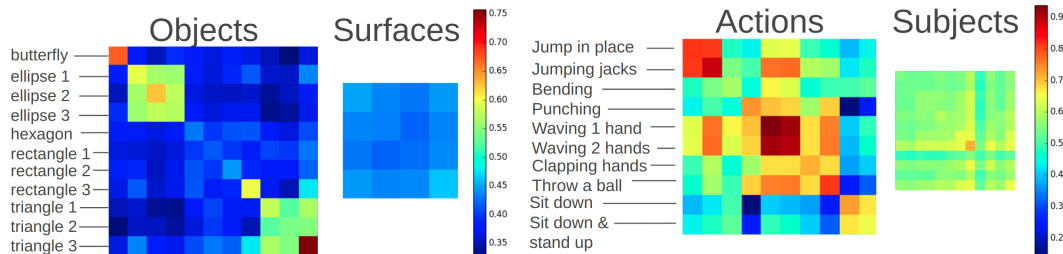
Figure 5: Shared modules show internal structure of the datasets. In particular, less important factors (surfaces and subjects) don't change the structure and big changes (objects and actions) do. Within the structural changes, there is more sharing between conceptually similar datasets.

two different meta-learning scenarios. In the first, the object-surface combination in the test case was present in some meta-training task; in the second, the objects used in the meta-training tasks differ from the object used in the test task. The results in table 1 show that, for previously encountered objects, MOMA performs best, and BOUNCEGRAD outperforms MAML. For unknown objects, all three approaches perform roughly equivalently.

Another important aspect of the structured hypothesis space is that it can give us insight into the relationships between tasks. Figure 5 illustrates the structural relationships that were uncovered in this data. The matrix on the left is indexed according to which object was being pushed. The entry in location i, j represents the average percentage of modules shared by the structure learned to predict results for object i and the structure learned for object j. We can see it distinguishes 3 clusters of data: peanut, all ellipses, and all triangles. The biggest rectangle shares modules with the biggest triangle, probably due to similar size and aspect ratio. The matrix on the right, doesn't show dependence on the surface as expected for the quasi-static regime.

4.3 Predicting skeleton configurations

Robots that interact with humans should be able to understand and predict human action, both for the purposes of safety and for task-driven collaboration. We applied meta-learning to this problem, using data from the Berkeley MHAD motion capture dataset [25]. This domain is dynamic, and so we use three previous configurations (at intervals of 0.1 sec) of a human skeleton to predict the next one. Each configuration is characterized by one 3-d position and 90 joint angles describing a kinematic tree, so the input has dimension 279 and the output has dimension 93. There are 12 subjects performing 11 different actions several times, for a few seconds each.

We constructed a compositional scheme for BOUNCEGRAD that is related to this task. It has a fixed first layer with 128 output units to compress the input, which is the same for all structures, and independent "parallel" modules that take those 128 inputs and produce kinematic sub-trees for each body part (2 legs, 2 arms, and torso). For MAML and BIGNET we use a single architecture of the same depth with 4 times more parameters. We again consider two different meta-learning scenarios. In the first, the activities used in the training task are the same as the activities used in the final task, but the human actor varies; in the second, the activities used in the training tasks differ from the activity used in the test task. The results in table 1 show that, for known activities, BOUNCEGRAD and MOMA perform best. For unknown activities, none of the methods perform very well, but MOMA outperforms the others. We obtain a similar pattern of correlation among shared modules, shown in figure 5, in which there is significant module-sharing among similar tasks and no real pattern of module-sharing among human actors.

4.4 Conclusion

We have demonstrated that modular compositional structure can provide a useful basis for transferring learned competence from previous tasks to new tasks, enabling learning from a small amount of training data in the new task. It can also yield insight into the underlying structure of the domain. The structural information to be provided in advance is not much more than would be required for specifying a typical deep-learning architecture: it consists of a few lines of code to describe the possible modifications that can be made to a structure.

Acknowledgments

References

[1] J. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[2] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[3] A. Wilson, A. Fern, S. Ray, and P. Tadepalli. Multi-task reinforcement learning: A hierarchical bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning*, New York, NY, USA, 2007. ACM.

[4] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[5] A. Nichol, J. Achiam, and J. S. Openai. On First-Order Meta-Learning Algorithms. *arXiv preprint arXiv:1803.0299*.

[6] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

[7] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.

[8] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[9] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl ^{2}: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[10] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[11] Y. Bengio, S. Bengio, and J. Cloutier. *Learning a synaptic learning rule*.

[12] H. Edwards and A. Storkey. Towards a neural statistician. *arXiv preprint arXiv:1606.02185*, 2016.

[13] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. Girshick. Inferring and executing programs for visual reasoning. *arXiv preprint arXiv:1705.03633*, 2017.

[14] M. Al-Shedivat, A. Dubey, and E. P. Xing. Contextual explanation networks. *arXiv preprint arXiv:1705.10301*, 2017.

[15] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *arXiv preprint arXiv:1710.01813*, 2017.

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

[17] T. Ullman, A. Stuhlmüller, N. Goodman, and J. Tenenbaum. Learning physical parameters from dynamic scenes. *Cogn. Psychol.*

[18] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2169–2176. IEEE, 2017.

[19] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

[20] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[21] E. Meyerson and R. Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. *arXiv preprint arXiv:1711.00108*, 2017.

[22] J. Liang, E. Meyerson, and R. Miikkulainen. Evolutionary architecture search for deep multi-task networks. *arXiv preprint arXiv:1803.03745*, 2018.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[24] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 30–37. IEEE, 2016.

[25] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy. Berkeley mhad: A comprehensive multimodal human action database. In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 53–60. IEEE, 2013.

[26] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] K. Rakelly. Pytorch-maml. `https://github.com/katerakelly/pytorch-maml`, 2018.