
Modular meta-learning in abstract graph networks for combinatorial generalization

Ferran Alet¹, Maria Bauza²,
Alberto Rodriguez², Tomás Lozano-Pérez¹, and Leslie P. Kaelbling¹
¹ CSAIL, MIT
² Mechanical Engineering, MIT
{alet, bauza, albertor, tlp, lpk}@mit.edu

Abstract

Modular meta-learning is a new framework that generalizes to unseen datasets by combining a small set of neural modules in different ways. In this work we propose *abstract graph networks*: using graphs as abstractions of a system's subparts without a fixed assignment of nodes to system subparts, for which we would need supervision. We combine this idea with modular meta-learning to get a flexible framework with combinatorial generalization to new tasks built in. We then use it to model the pushing of arbitrarily shaped objects from little or no training data.

1 Introduction

Meta-learning, or learning-to-learn, aims at fast generalization. The premise is that by training on a distribution of tasks we can learn a *learning algorithm* that, when given a new task, will learn from very little data. Recent progress in meta-learning has been very promising. However, we would like to generalize in broader ways by exploiting inherent structure and compositionality in the domains.

We use the *modular meta-learning* framework (Alet et al., 2018), which generalizes by learning a set of neural network *modules* that can be composed in different ways to solve a new task, without changing their weights. We generalize to unseen data-sets by combining learned concepts in the domain, exhibiting combinatorial generalization; "making infinite use of finite means"(von Humboldt).

Combinatorial generalization is also one of the main motivations behind *graph neural networks* (GNNs). Therefore, we suggest that GNNs are an ideal structure to use as a rule for combining modules in modular meta-learning. However, GNNs are typically only used when there is a clear underlying graph structure in the domain: sets of entities and relations (such as charged particles or body skeletons) with supervised data for each entity. This limits the domains in which graph networks can be applied. We propose *abstract graph networks* (AGNs), which use GNNs as an abstraction of a system into its sub-parts without the need to a-priori match each subpart to a concrete entity. A similar idea underlies finite element methods frequently used in mathematics and engineering.

We show that combining modular meta-learning and AGNs generalizes well by choosing different *node* and *edge* modules, leading to flexible generalization from little data. We apply this method to a new diverse real-world dataset (Bauza et al., 2018) of a robot pushing objects of varying mass distribution and shape on a planar surface. We generalize over both the distribution of objects we trained on and out-of-distribution objects and surfaces from little data. Moreover, with a second model (section 3.3) we generalize to new objects with no data, only a diagram of the object.



Figure 1: Modeled objects are made of the four possible sides. Masses can be attached in the holes. There is an extra hole in the triangle, covered by the wafers.

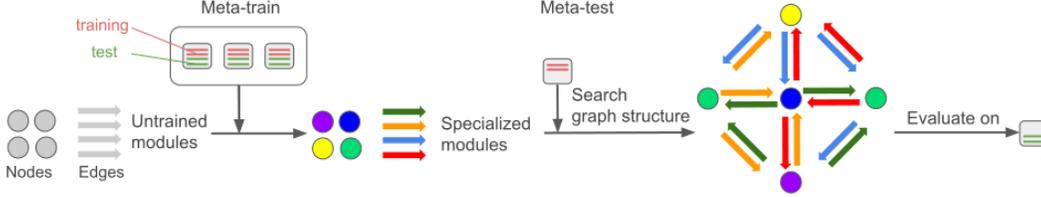


Figure 2: Modular meta-learning with Graph Networks; adapted from Alet et al. (2018).

2 Related Work

The work closest to ours in the meta-learning literature is by Kopicki et al. (2017) and Alet et al. (2018). However, the former uses a product of experts and needs information about the points of contact. The latter uses modular neural networks with a composition similar to a combination of experts, which we show generalizes less well than our abstract graph neural network model.

We combine modular meta-learning and AGNs, aiming for combinatorial generalization by composing a set of modules into different structures. This is part of a growing body of work that merges the best aspects of deep learning with structured solution spaces in order to obtain broader generalizations (Tenenbaum et al., 2011; Andreas et al., 2016; Meyerson & Miikkulainen, 2017; Chang et al., 2018).

By choosing between multiple modules for each edge and nodes, our work is related to Kipf et al. (2018). However, our meta-learning framework encodes the bias of time-independence and we are able to optimize modules as a group instead of having to select them independently of one another. More related work on GNNs can be found at the beginning of section 3.2 where we introduce abstract graph networks in its context; additional related work can be found in appendix B.

3 Methods

In this section we describe modular meta-learning, abstract graph networks, graph element networks and how they can be combined.

3.1 Modular meta-learning

Meta-learning, or learning to learn, can be seen as learning a learning algorithm. More formally, in the context of supervised learning, instead of learning a regressor f with parameters Θ with the objective that $f(\mathbf{x}_{test}, \Theta) \approx \mathbf{y}_{test}$, we aim to learn an algorithm A that takes a small training set $D_{train} = (\mathbf{x}_{train}, \mathbf{y}_{train})$ and returns a hypothesis h that performs well on the test set:

$$h = A(D_{train}, \Theta) \text{ s.t. } h(\mathbf{x}_{test}) \approx \mathbf{y}_{test}; \text{ i.e. } A \text{ minimizes } \mathcal{L}(A(D_{train}, \Theta)(\mathbf{x}_{test}), \mathbf{y}_{test}).$$

Similar to conventional learning algorithms, we optimize Θ , the parameters of A , to perform well. Analogous to the training set, we have a meta-training set of tasks each with its own train and test.

Modular meta-learning (Alet et al., 2018) learns a set of small neural network modules and generalizes by composing them. In particular, let m_1, \dots, m_k be the set of modules, with parameters $\theta_1, \dots, \theta_k$ and \mathcal{S} a set of structures that describes how modules are composed. For example, simple compositions can be adding the modules outputs, concatenating them, or using the output of several modules to guide attention over the results of other modules.

Then $\Theta = \cup \theta_i$, and the algorithm A operates by searching over the set of possible structures \mathcal{S} to find the one that best fits D_{train} , and applies it to \mathbf{x}_{test} . Let $h_{S, \Theta}$ be the function that predicts the output using the modular structure S and parameters Θ . Then:

$$S^* = \arg \min_{S \in \mathcal{S}} \mathcal{L}(h_{S, \Theta}(\mathbf{x}_{train}), \mathbf{y}_{train}); \quad A(D_{train}, \Theta) = h_{S^*, \Theta}.$$

3.2 Abstract graph networks

Graph neural networks (Gori et al., 2005; Scarselli et al., 2009; Battaglia et al., 2018) perform computations over a graph, with the aim of incorporating *relational inductive biases*: assuming the existence of a set of entities and relations between them. In general, GNNs are applied in settings where nodes and edges are *explicit* and match concrete entities of the domain. For instance, nodes

can be objects (Chang et al., 2016; Battaglia et al., 2016), parts of these objects (Wang et al., 2018; Mrowca et al., 2018) or users in a social network. In general, nodes and edges represent explicit entities and relations, and supervised data is required for each component. We propose to use GNNs as an abstraction of a system’s sub-parts, without a concrete match between entities and nodes.

Let \mathcal{G} be a graph, we define an encoding function f_{in} that goes from input to initial states of the nodes and edges in \mathcal{G} . We run several steps of message passing (Gilmer et al., 2017): in each step, nodes read incoming messages, which are a function of their neighbours’ hidden states and their corresponding edges, and use their current state and the sum of incoming messages to compute their new state. Finally, we define a function f_{out} that maps the nodes final states to the final output: $f_{out}(MP^{(T)}(f_{in}(x)))$; trainable end-to-end. This defines a means of combination for the modules, giving us different (differentiable) losses depending on the module we put in each position.

3.3 Graph element networks

Finite element methods (FEMs) are a widely used tool in engineering to solve boundary-value problems for partial differential equations. They subdivide a complex problem in a domain (such as analyzing stress forces on a bridge or temperature in a building) into a mesh of small elements where the dynamics of the PDE can be approximated. With that inspiration, we propose a subtype of abstract graph network, which we denote *graph element networks* (GEN). In GENs, we *place* nodes in a region (usually of \mathbb{R}^2 or \mathbb{R}^3); each having specific coordinates. These nodes induce a *Voronoi diagram* where a node is responsible for all points in space to which it is the closest representative.

As in general AGNs, we have to define a function between input and graph input (f_{in}) and between graph output and final output (f_{out}) and the connectivity between the nodes. 1) f_{in} : a natural choice is to map the input to an initial ‘vector field’ at a particular position in space. We can give the input to the node responsible for the corresponding Voronoi region. For example, in our pushing example we pick the initial position of the pusher and feed the input (including the said initial position) to its closest node. 2) Connectivity: A simple choice is a regular grid over the region, thus also defining the edges. A more general solution, with many desirable properties, is to use the edges from the Delaunay triangulation of the nodes. The Delaunay triangulation is the graph where two nodes are connected if and only if their corresponding Voronoi regions share an edge. 3) f_{out} : a learned integrator function that either sums or averages a transformation over the final states of each module; thus being independent of the order and number of nodes.

In GENs nodes are still abstract (meaning they do not have a 1-to-1 correspondence with a discrete element in the system), but they are *grounded* in a particular subregion. This makes it possible for other data to inform the types of node modules to put in each node. For example, in our experiments with pushing objects, the node types are informed by whether there is a mass, regular surface or empty space in that particular position in space. This information can either completely determine the node type (as in our current experiments) or only partially inform it: for instance we could imagine knowing the shape of an object but not the density of each subpart.

Note that, in contrast with *finite element methods*, the dynamics of the system follow completely unknown equations; giving us a more complicated, but also more flexible, framework. GEN nodes and edges have different types, allowing dynamics to be different in different parts of the space; for example depending on the material properties of that particular region. Moreover, we will also have a similar trade-off between accuracy and computational performance where a bigger graph of smaller Voronoi regions will have higher accuracy at the expense of more computational cost.

When graph structures are fixed and module types are completely determined by an external source (such as a picture of the object), we are doing multitask learning between the meta-train tasks and then transferring to new domains without any need for experimental data; only using the picture of the object. However, in general the nodes may not be known (a material not seen at training) or we may customize the graph to the system to get the best accuracy given fixed computational resources.

4 Experiments

The dataset (Bauza et al., 2018), consists of 250 pushes for each of 250 objects, which were constructed by attaching 4 pieces to a central part with magnets and possibly adding masses, such as the one shown in figure. We also collected data for out-of-distribution objects and a different surface.

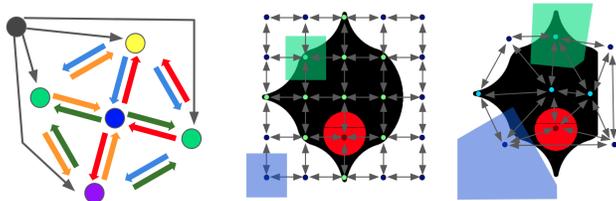


Figure 3: **Left:** AGN wheel graph plus 'pusher' node (details in appendix C). **Center:** GEN with a 5x5 grid, nodes conditioned on image; highlighted two Voronoi regions. **Right:** not implemented GEN with Delaunay edges and optimized positions

For pushing, we train two structures. 1) a few-shot learning abstract graph network structure which models the object as a wheel graph with bidirectional edges. This structure has many desirable properties: low graph diameter, sparsity and a notion of locality (the position of nodes matters). All details can be found in appendix C. 2) A multitask learning graph element network, where we also receive a top-view image for each object, from which we can extract a model without the need for any data. For the GEN, the graph is a 5 by 5 grid on $[-10cm, 10cm]^2$ with the module type of each node is entirely determined by the 'material' at its position (empty, small/big mass, no mass).

We normalize each dimension of input and output so that the MSE (mean square error) of each coordinate is 1 across all meta-training dataset points. This setting allows us to create a loss that combines angle changes with position changes, which have different orders of magnitude. As a result, the baseline of predicting the current position for the next position (no motion) has MSE 1, and allows to interpret the performance of other algorithms as a percentage of the total variance.

We use algorithm 1 to train our system on 200 objects/tasks and test on 50 tasks; within each task we train on 50 points and test on 200. We compare against an object-independent regressor that has a single training set with the full 250 points for 200 objects(50k points), and evaluates on the 50 remaining tasks. Additionally, we compare to modular meta-learning but on an attention-based non-AGN structure used in our previous work. We find that the method with pooled data performs the worst, outperformed by both modular meta-learning approaches, and that the use of AGNs significantly improves performance over the original modular structure. See table 1 for all results.

We also picked 5 objects for which we collected 2500 points, trained a single model for each of them on 2000 points and tested on 500 and obtained a MSE of 0.04. Previous experiments have shown that the test loss stops decreasing significantly after 2000 points (Bauza & Rodriguez, 2017), which leads us to believe this is a good approximation of the Bayes error rate: the irreducible error due to camera and actuation noise, position on the surface, and other factors. Therefore, even though our meta-learned models only use 50 points we are very close to the estimated Bayes error.

Evaluation	MSE	distance equivalent
Predicting no movement	1.00	21.6 mm
Estimated Bayes error rate	.04	4.3 mm
Original distribution (no meta-learning)	.14	8.1 mm
Original distribution (Alet et al. (2018))	.08	6.1 mm
Original distribution (meta-learning & AGN)	.06	5.3 mm
Original distribution (image-conditioned & GEN)	.05	4.7 mm
Out of distribution obj.&surface (no meta-learning)	.51	15.4 mm
Out of distribution obj.&surface (Alet et al. (2018))	.34	12.5 mm
Out of distribution obj.&surface (meta-learning & AGN)	.29	11.6 mm
Out of distribution objects (no meta-learning)	.34	12.7 mm
Out of distribution objects (Alet et al. (2018))	.30	11.8 mm
Out of distribution objects (meta-learning & AGN)	.25	10.8 mm
Out of distribution surface (no meta-learning)	.21	9.8 mm
Out of distribution surface (Alet et al. (2018))	.09	6.6 mm
Out of distribution surface (meta-learning & AGN)	.08	5.9 mm

Table 1: Summary of results. Different datasets are separated by horizontal lines.

We evaluate on 11 shapes that were not constructed using the four sides and magnets (see Bauza et al. (2018) for details). We also evaluated the effect of using a different surface (plywood) rather than the one used for the training data. In both cases we observe a predictable drop in performance, but we account for more than 70% of the variance, even when both factors change.

References

- Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pp. 5074–5082, 2016.
- Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P. Kaelbling, Joshua B. Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *International Conference on Intelligent Robots and Systems (IROS)*, 2018. URL <http://lis.csail.mit.edu/pubs/ajay-iros18.pdf>.
- Ferran Alet, Tomas Lozano-Perez, and Leslie P. Kaelbling. Modular meta-learning. In *Proceedings of The 2nd Conference on Robot Learning*, pp. 856–868, 2018.
- Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016.
- Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016.
- Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- Maria Bauza and Alberto Rodriguez. A probabilistic data-driven model for planar pushing. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3008–3015. IEEE, 2017.
- Maria Bauza, Ferran Alet, Leslie Lozano-Perez, Tomasand Kaelbling, and Alberto Rodriguez. Omnipush: accurate, diverse, real-world dataset of pushing dynamics with rgbd images. In *NeurIPS Physics Workshop*, 2018.
- Michael James Behrens. *Robotic Manipulation by Pushing at a Single Point with Constant Velocity: Modeling and Techniques*. PhD thesis, University of Technology, Sydney, 2013.
- Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 173–180. IEEE, 2017.
- Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.
- Michael B Chang, Abhishek Gupta, Sergey Levine, and Thomas L Griffiths. Automatically composing representation transformations as a means for generalization. *arXiv preprint arXiv:1807.04640*, 2018.
- Rohan Chitnis, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning quickly to plan quickly using modular meta-learning. *arXiv preprint arXiv:1809.07878*, 2018.
- Noam Chomsky. *Aspects of the Theory of Syntax*, volume 11. MIT press, 2014.
- Yan Duan, Marcin Andrychowicz, Bradley Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pp. 1087–1098, 2017.
- Kevin Ellis, Lucas Morales, Mathias Sablé Meyer, Armando Solar-Lezama, and Joshua B Tenenbaum. Search, compress, compile: Library learning in neurally-guided bayesian program learning. In *Advances in neural information processing systems*, 2018.
- Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

- Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pp. 64–72, 2016.
- Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.
- Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.
- Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pp. 729–734. IEEE, 2005.
- Francois Robert Hogan, Eudald Romo Grau, and Alberto Rodriguez. Reactive planar manipulation with convex hybrid mpc. *arXiv preprint arXiv:1710.05724*, 2017.
- Wilhelm Humboldt. On language: On the diversity of human language construction and its influence on the mental development of the human species. 1999.
- Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.
- Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.
- Marek Kopicki, Sebastian Zurek, Rustam Stolkin, Thomas Moerwald, and Jeremy L Wyatt. Learning modular and transferable forward models of the motions of push manipulated objects. *Autonomous Robots*, 41(5):1061–1082, 2017.
- Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.
- Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.
- Jason Liang, Elliot Meyerson, and Risto Miikkulainen. Evolutionary architecture search for deep multitask networks. *arXiv preprint arXiv:1803.03745*, 2018.
- Huan Liu. Pushing with a physics-based model. Master’s thesis, Massachusetts Institute of Technology, 2011.
- Kevin M Lynch, Hitoshi Maekawa, and Kazuo Tanie. Manipulation and active sensing by pushing using tactile feedback. In *IROS*, 1992.
- Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.
- Matthew T Mason. Mechanics and planning of manipulator pushing operations. *IJRR*, 5(3), 1986.
- Elliot Meyerson and Risto Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. *arXiv preprint arXiv:1711.00108*, 2017.
- Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. 2018.
- Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. *arXiv preprint arXiv:1806.08047*, 2018.
- Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. URL <http://arxiv.org/abs/1803.02999>.
- Judea Pearl. Theoretical impediments to machine learning with seven sparks from the causal revolution. *arXiv preprint arXiv:1801.04016*, 2018.

- Michael Peshkin and Arthur C Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal of Robotics and Automation*, 1988.
- Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *ECCV*, 2016.
- Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *CoRR*, abs/1711.01239, 2017.
- Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter W. Battaglia. Graph networks as learnable physics engines for inference and control. In *ICML*, 2018.
- Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.
- Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.
- Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022):1279–1285, 2011.
- Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.
- Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pp. 242–264. IGI Global, 2010.
- Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.
- Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
- Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pp. 3630–3638, 2016.
- Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. 2018.
- Nicholas Watters, Andrea Tacchetti, Theophane Weber, Razvan Pascanu, Peter Battaglia, and Daniel Zoran. Visual interaction networks. *arXiv preprint arXiv:1706.01433*, 2017.
- Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *arXiv preprint arXiv:1710.01813*, 2017.
- Jiaji Zhou and Matthew T Mason. Pushing revisited: Differential flatness, trajectory planning and stabilization. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2017.

A BOUNCEGRAD algorithm for abstract graph networks

Notation

- \mathcal{G} : graph, with node n_1, \dots, n_r and directed edges $e_1, \dots, e_{r'}$.
- f_{in} : encoding function from input to graph initial states.
- f_{out} : decoding function from graph final states to output.
- \mathbb{G} : set of node modules $g_1, \dots, g_{|\mathbb{G}|}$, where g_i is a network with weights θ_{g_i} .
- \mathbb{H} : set of edge modules $h_1, \dots, h_{|\mathbb{H}|}$, where h_i is a network with weights θ_{h_i} .
- S : a structure, one module per node m_{n_1}, \dots, m_{n_r} , one module per edge $m_{e_1}, \dots, m_{e_{r'}}$. Each m_{n_i} is a pointer to \mathbb{G} and each m_{e_j} is a pointer to \mathbb{H} .

- $\mathcal{T}^1, \dots, \mathcal{T}^k$: set of regression tasks, from which we can sample (x, y) pairs.
- $\text{MP}^{(T)}(\mathcal{G}, S)(x_t) \rightarrow x_{t+1}$: message-passing function applied T times.
- $\mathcal{L}(y_{\text{target}}, y_{\text{pred}})$: loss function; in our case $|y_{\text{target}} - y_{\text{pred}}|^2$.
- L : instantiation of a loss. This includes the actual loss value and infrastructure to backpropagate it.

Algorithm 2 Alternating optimization between structures and modules on abstract graph networks.

```

1: procedure INITIALIZESTRUCTURE( $\mathcal{G}, \mathbb{G}, \mathbb{H}$ )                                ▷ Initialize with random modules
2:   for  $n_i \in \mathcal{G}.\text{nodes}$  do  $S.m_{n_i} \leftarrow \text{random\_elt}(\mathbb{G})$ 
3:   for  $e_i \in \mathcal{G}.\text{edges}$  do  $S.m_{e_i} \leftarrow \text{random\_elt}(\mathbb{H})$ 
4: procedure PROPOSESTRUCTURE( $S, \mathcal{G}, \mathbb{G}, \mathbb{H}$ )
5:    $P \leftarrow S$ 
6:   if Bernouilli(1/2) then
7:      $idx \leftarrow \text{random\_elt}(\mathcal{G}.\text{nodes}); P.m_{n_{idx}} \leftarrow \text{random\_elt}(\mathbb{G})$ 
8:   else
9:      $idx \leftarrow \text{random\_elt}(\mathcal{G}.\text{edges}); P.m_{e_{idx}} \leftarrow \text{random\_elt}(\mathbb{H})$ 
10:  return  $P$ 
11: procedure EVALUATE( $\mathcal{G}, S, \mathcal{L}, \mathbf{x}, \mathbf{y}$ )
12:   $\mathbf{p} \leftarrow f_{\text{out}}(\text{MP}^{(T)}(\mathcal{G}, S)(f_{\text{in}}(\mathbf{x})))$                                 ▷ Running the AGN with modular structure S
13:  return  $\mathcal{L}(\mathbf{y}, \mathbf{p})$ 
14: procedure BOUNCEGRAD( $\mathcal{G}, f_{\text{in}}, f_{\text{out}}, \mathbb{G}, \mathbb{H}, \mathcal{T}^1, \dots, \mathcal{T}^k$ )          ▷ Modules in  $\mathbb{G}, \mathbb{H}$  start untrained
15:  for  $l \in [1, k]$  do
16:     $S^l \leftarrow \text{InitializeStructure}(\mathcal{G}, \mathbb{G}, \mathbb{H})$ 
17:    while not done do
18:       $l \leftarrow \text{random\_elt}([1, k])$ 
19:       $P \leftarrow \text{ProposeStructure}(S^l, \mathcal{G}, \mathbb{G}, \mathbb{H})$ 
20:       $(\mathbf{x}, \mathbf{y}) \leftarrow \text{sample}(\mathcal{T}^l)$                                           ▷ Train data
21:       $L_{S^l} \leftarrow \text{Evaluate}(\mathcal{G}, S^l, \mathcal{L}, \mathbf{x}, \mathbf{y})$ 
22:       $L_P \leftarrow \text{Evaluate}(\mathcal{G}, P, \mathcal{L}, \mathbf{x}, \mathbf{y})$ 
23:       $S^l \leftarrow \text{SimulatedAnnealing}((S^l, L_{S^l}), (P, L_P))$               ▷ Choose between  $S^l$  and  $P$ 
24:       $(\mathbf{x}', \mathbf{y}') \leftarrow \text{sample}(\mathcal{T}^l)$                                           ▷ Test data
25:       $L \leftarrow \text{Evaluate}(\mathcal{G}, S^l, \mathcal{L}, \mathbf{x}', \mathbf{y}')$ 
26:      for  $h \in \mathbb{H}$  do  $\theta_h \leftarrow \text{GradientDescent}(L, \theta_h)$ 
27:      for  $g \in \mathbb{G}$  do  $\theta_g \leftarrow \text{GradientDescent}(L, \theta_g)$ 
28:  return  $\mathbb{G}, \mathbb{H}$                                                                 ▷ Return specialized modules

```

B Extra related work

The literature in meta-learning (Schmidhuber, 1987; Thrun & Pratt, 2012; Lake et al., 2015) and multi-task learning (Torrey & Shavlik, 2010) is now too extensive to cover entirely, with (Koch et al., 2015; Vinyals et al., 2016; Duan et al., 2017; Mishra et al., 2018; Finn et al., 2017; Nichol et al., 2018) being some prominent examples. The work closest to ours in the meta-learning literature is by Kopicki et al. (2017) and Alet et al. (2018). However, the former uses a product of experts and needs information about the points of contact. The latter uses modular neural networks with an approach similar to a combination of experts, which we show generalizes less well than our graph neural network model.

We join modular meta-learning (Alet et al., 2018) and abstract graph networks, aiming for combinatorial generalization (Humboldt, 1999; Chomsky, 2014) by composing a set of modules in different structures. In this sense, we are part of a growing number of works aiming to merge the best of deep learning with structure aiming at broader generalizations (Tenenbaum et al., 2011; Andreas et al., 2016; Meyerson & Miikkulainen, 2017; Xu et al., 2017; Fernando et al., 2017; Rosenbaum et al., 2017; Marcus, 2018; Pearl, 2018; Liang et al., 2018; Battaglia et al., 2018; Chang et al., 2018; Chitnis et al., 2018; Ellis et al., 2018).

We are part of a growing body of work that merges the best aspects of deep learning with structured solution spaces in order to obtain broader generalizations (Tenenbaum et al., 2011; Andreas et al.,

2016; Meyerson & Miikkulainen, 2017; Xu et al., 2017; Fernando et al., 2017; Rosenbaum et al., 2017; Marcus, 2018; Pearl, 2018; Liang et al., 2018; Battaglia et al., 2018; Chang et al., 2018; Chitnis et al., 2018; Ellis et al., 2018).

Graph neural networks (Gori et al., 2005; Scarselli et al., 2009; Gilmer et al., 2017; Battaglia et al., 2018) perform computations over a graph, with the aim of incorporating *relational inductive biases*: assuming the existence of a set of entities and relations between them. In general, GNNs are applied in settings where nodes and edges are *explicit* and match concrete entities of the domain. For instance, nodes can be objects (Chang et al., 2016; van Steenkiste et al., 2018; Battaglia et al., 2016; Watters et al., 2017) or be parts of these objects (Wang et al., 2018; Sanchez-Gonzalez et al., 2018; Battaglia et al., 2016; Mrowca et al., 2018; Li et al., 2018). In general, nodes and edges represent explicit entities and relations, and supervised data is generally required for each component.

The interest on pushing is not new and it has been extensively studied theoretically (Mason, 1986; Peshkin & Sanderson, 1988; Lynch et al., 1992; Liu, 2011; Behrens, 2013; Zhou & Mason, 2017), with pure learning methods (Finn et al., 2016; Agrawal et al., 2016; Pinto et al., 2016; Bauza & Rodriguez, 2017; Alet et al., 2018) and by leveraging theory for machine learning methods (Hogan et al., 2017; Ajay et al., 2018; Byravan & Fox, 2017).

C All details about our *Abstract Graph Network* for pushing

We can describe the dynamics of the object as a function of the dynamics of its parts. To do so, we model the object as a graph, and more concretely, a wheel graph. A wheel graph has the property of having a very low diameter (2) yet, in contrast to a star graph, also has a notion of 'locality'. Moreover, all exterior nodes are computationally equivalent, which will make it easier to share modules between them.

The pusher is modeled via a single node. This node will interact with all the other nodes via the same 'edge module'. Since we would like the pusher to only interact with a very small part of the object we will use a (learned) attention mechanism, ensuring that the module node sends a stronger signal to a single part of the object. This is similar to Velickovic et al. (2017) using attention in the outgoing direction instead of incoming edges.

Our module composition is to build a graph neural network with the shape of a wheel graph. Therefore, for a wheel of N exterior nodes we will need $4N$ 'edge modules': N going clockwise, N counter-clockwise, N to the center and N out from the center. We then have $N + 1$ node modules (N exterior, 1 central). Filling these slots with different modules will give us different structures, which in turn will lead to different predictions. The pusher node and edge modules are constant across datasets.

As we mentioned earlier, all the exterior nodes are symmetric; which doesn't allow us to have a sense of direction. To break this symmetry set a custom 'code' to each node's hidden state; modules can read, but not write, to these parts of the hidden state. The code for the i -th exterior node to $[\cos \frac{2\pi i}{N}, \sin \frac{2\pi i}{N}, 0, 0, 0, 0, 0]$. The central node is initialized with $[0, 0, 1, 0, 0, 0, 0]$ and the pusher with $[0, 0, 0, 1, X_x, X_y, X_\theta]$, initially being the only node containing the input.

We run 5 steps of message passing; enough for the information to go from the pusher to node A(1 step), then node B (2 steps), and finally to C(2 steps) with the goal of capturing global dynamics. After these 5 steps we pick the last 3 positions of the each exterior node and multiply them by the cosine of the corresponding angle, we pick the -6:-3 positions and multiply them by the sine of the corresponding angle and average all these vectors into a 3-dimensional output. Note that, since the integral of sine and cosine is 0, the Graph Neural Network is forced to learn to place different states in different nodes.

Finally, note that we fix the angles, not the concrete positions. Since these angles can be defined for all shapes; this abstraction naturally extends to all 2d shapes. Moreover, we expect (and experimentally show) that modules can be reused to play similar local and global effects on very different shapes. We conjecture that the same architecture could model rich object-object interactions: this could be achieved by having multiple wheels and edges with attention created between nearby objects at each timestep; similar to Chang et al. (2016).