

# Compositionality and Modularity for Robot Learning

*Coline Devin*



Electrical Engineering and Computer Sciences  
University of California at Berkeley

Technical Report No. UCB/EECS-2020-207

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-207.html>

December 17, 2020

Copyright © 2020, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Compositionality and Modularity for Robot Learning

by

Coline Devin

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Co-chair

Assistant Professor Sergey Levine, Co-chair

Professor Pieter Abbeel, Professor Alison Gopnik

Fall 2020

# Compositionality and Modularity for Robot Learning

Copyright 2020  
by  
Celine Devin

## Abstract

## Compositionality and Modularity for Robot Learning

by

Coline Devin

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Trevor Darrell, Co-chair

Assistant Professor Sergey Levine, Co-chair

Humans are remarkably proficient at decomposing and recombining concepts they have learned [16, 102]. In contrast, while deep learning-based methods have been shown to fit large datasets and out-perform humans at some tasks [126], they often fail when presented with conditions even just slightly outside of the distribution they were trained on [59, 152, 7]. In particular, machine learning models fail at *compositional generalization*, where the model would need to predict how concepts fit together without having seen that exact combination during training [78]. This thesis proposes several learning-based methods that take advantage of the compositional structure of tasks and shows how they perform better than black-box models when presented with novel compositions of previously seen subparts. The first type of method is to directly decompose neural network into separate modules that are trained jointly in varied combinations. The second type of method is to learn representations of tasks and objects that obey arithmetic properties such that tasks representations can be summed or subtracted to indicate their composition or decomposition. We show results in diverse domains including games, simulated environments, and real robots.

To my parents, Catherine Granger and Matthieu Devin

# Contents

|  |           |
|--|-----------|
| <b>Contents</b>  | <b>ii</b> |
| <b>List of Figures</b>   | <b>iv</b> |
| <b>List of Tables</b>  | <b>ix</b> |
| <b>1 Introduction</b>  | <b>1</b>  |
| 1.1 Modularity and Compositionality . . . . .                  | 1         |
| 1.2 Organization, and Contributions . . . . .                  | 2         |
| <b>2 Related Work</b>  | <b>4</b>  |
| <b>3 Modular Networks</b>                                      | <b>6</b>  |
| 3.1 Introduction . . . . .                                     | 6         |
| 3.2 Relation to prior work . . . . .                           | 7         |
| 3.3 Modular Policy Networks . . . . .                          | 9         |
| 3.4 Experiments . . . . .                                      | 13        |
| 3.5 Discussion and Future Work . . . . .                       | 19        |
| <b>4 Object-Centric Representations</b>                        | <b>22</b> |
| 4.1 Introduction . . . . .                                     | 22        |
| 4.2 Relation to Prior Work . . . . .                           | 25        |
| 4.3 Deep Object-Centric Representations . . . . .              | 26        |
| 4.4 Implementation . . . . .                                   | 27        |
| 4.5 Experiments . . . . .                                      | 29        |
| 4.6 Discussion . . . . .                                       | 34        |
| <b>5 Grasp2Vec</b>   | <b>36</b> |
| 5.1 Introduction . . . . .                                     | 36        |
| 5.2 Relation to Prior Work . . . . .                           | 38        |
| 5.3 Grasp2Vec: Representation Learning from Grasping . . . . . | 38        |
| 5.4 Self-Supervised Goal-Conditioned Grasping . . . . .        | 40        |
| 5.5 Experiments . . . . .                                      | 43        |

|          |   |           |
|----------|---|-----------|
| 5.6      | Discussion . . . . .                        | 47        |
| <b>6</b> | <b>Plan Arithmetic</b>                      | <b>48</b> |
| 6.1      | Introduction . . . . .                      | 48        |
| 6.2      | Relation to Prior Work . . . . .            | 49        |
| 6.3      | Compositional Plan Vectors . . . . .        | 50        |
| 6.4      | Sequential Multitask Environments . . . . . | 54        |
| 6.5      | Experiments . . . . .                       | 55        |
| 6.6      | Discussion . . . . .                        | 58        |
| <b>7</b> | <b>Conclusion</b>                           | <b>60</b> |
|          | <b>Bibliography</b>                         | <b>62</b> |
| <b>A</b> | <b>Grasp2Vec</b>                            | <b>75</b> |
| A.1      | Qualitative Detection Results . . . . .     | 75        |
| A.2      | Simulation Experiment Details . . . . .     | 78        |
| <b>B</b> | <b>Compositional Plan Vectors</b>           | <b>84</b> |
| B.1      | Network Architectures . . . . .             | 84        |
| B.2      | Hyperparameters . . . . .                   | 85        |
| B.3      | Additional Experiments . . . . .            | 85        |



# List of Figures

|     |  |    |
|-----|--|----|
| 1.1 | While learning based methods can learn behaviors given enough data for that task, they often fail to perform well in novel situations, even if those situations are compositions of the previously seen tasks. . . . .   | 2  |
| 3.1 | The 3DoF and a 4DoF robot which specify one degree of variation (robots) in the universe described in Section 3.3 as well as the tasks of opening a drawer and pushing a block which specify the other degree of variation (tasks) in the universe.  | 8  |
| 3.2 | The possible worlds enumerated for all combinations of tasks and robots for the universe described in Section 3.3 . . . . .  | 8  |
| 3.3 | Modular policy composition for a universe with 2 tasks and 2 robots. There are 4 available modules - 2 task modules and 2 robot modules, and each module is a neural network. For the training worlds, these modules are composed together to form the individual policy networks. Modules of the same color share their weights. Policy networks for the same task share task modules and those for the same robot share robot modules. The training worlds are composed and then trained end-to-end. On encountering an unseen world, the appropriate previously trained modules are composed to give a policy capable of good zero-shot performance . | 12 |
| 3.4 | Basic visuomotor policy network for a single robot. The two convolutional layers and spatial softmax form the task module, while the last few fully connected layers form the robot module . . . . .   | 14 |
| 3.5 | Grid of tasks vs robots for the reaching colored blocks task in simulation described in 3.4.3. We train on all the worlds besides the 4link robot reaching to black, and test on this held out world. . . . .  | 15 |
| 3.6 | Grid of tasks vs robots for the object manipulation tasks described in 3.4.4. The horizontal drawer tasks involve sliding the grey drawer horizontally to the target in the direction indicated by the arrow on the image. The vertical drawer tasks involve sliding the grey drawer vertically up in the direction indicated by the arrow on the image. The block push tasks involve pushing the white block to the red goal. All positions shown in this image are the final successful positions for each world. . . . .  | 17 |

|     |   |    |
|-----|---|----|
| 3.7 | Results on the 3-link robot performing horizontal drawer pulling. The initialization from composing the 3-link robot module with the horizontal pull task module provides the fastest learning. Although the vertical drawer module was trained with the 3-link robot, the task is too different to directly transfer. Random initialization performs well with reward shaping, but without it is unable to learn the task. . . . .   | 19 |
| 3.8 | R-obots for visually distinct tasks mentioned in 3.4.5. The reach task involves reaching the white target. The push task involves pushing the white block to the red goal. The peg insertion task involves inserting the peg at the tip of the robot into the hole specified by the black square. These tasks involve manipulation and are also visually distinct in that they do not use the colors in the same way (the target and block colors are not consistent). We train on all combinations besides the 4link robot pushing the block. . . . .  | 20 |
| 3.9 | The final positions of the zero-shot performance of our method on the blockpushing task. Our method performs the task very well on zero shot and gets the white block to the red goal. . . . .  | 21 |
| 4.1 | Deep object-centric representations learn to attend to task-relevant objects from just a few trajectories. The representation is robust to clutter and generalizes to new object instances. . . . .   | 23 |
| 4.2 | Faster RCNN trained on MSCOCO does not differentiate between cups and mugs, and gives higher probability to the cup, making it difficult to train a policy that needs to locate the mug. With our method, the attention can learn to prioritize mugs over cups. The dustpan is labeled as a spoon and thus can be distracted by other spoon-like objects. As limes are not in the MSCOCO dataset, the object detector does not label them. . . . .  | 24 |
| 4.3 | Method Overview. The parameters outlined in bright green are optimized during pretraining, while those outlined in dark blue are optimized during policy learning. The attention is trained to predict the movement seen in the provided demonstrations or trajectories. The “attention map” illustrates a soft attention where vectors $f(o^i)$ close to $w$ are given high probability (black) and ones far away are have low probability (white). The distribution is regularized to have low entropy, and the weighted sum of bounding box coordinates is fed to the next layers of the bright green network. The policy (in blue) is trained with $w$ held fixed, and the arg-max bounding box is used instead of the weighted average. Note: this diagram illustrates only a single attention vector $w$ ; more attention vectors can be added as needed. . . . . | 26 |
| 4.4 | The region proposals (task-independent attention) are drawn in blue and the task-specific attended regions are drawn in red. For the pouring task, the attention locks on to the mug as its position defines the trajectory. For the sweeping task, we use two attention vectors, one attends to the orange and one attends to the dustpan, which each have variable starting positions. . . . .  | 28 |

|     |  |    |
|-----|--|----|
| 4.5 | Results for the pouring task testing on different mugs not seen during training. Each group of bars shows performance on different unseen mugs, comparing our method with a policy trained from raw pixels. Our policy successfully generalizes to a much wider range of mugs than the raw pixel policy, which does not benefit from the object-centric attention prior. The “no vision” baseline indicates the performance of always pouring at the average mug position. . . . .   | 31 |
| 4.6 | Left: Mugs used for evaluation. Note that only the brown mug was seen during training. Center: Successful pouring into the pink mug. Right: Pouring into the brown mug in a cluttered environment that was not seen during training. . . . .   | 32 |
| 4.7 | . . . . .  | 33 |
| 5.1 | Instance grasping and representation learning processes generate each other’s labels in a fully self-supervised manner. <b>Representation learning from grasping:</b> A robot arm removes an object from the scene, and observes the resulting scene and the object in the gripper. We enforce that the difference of scene embeddings matches the object embedding. <b>Supervising grasping with learned representations:</b> We use a similarity metric between object embeddings as a reward for instance grasping, removing the need to manually label grasp outcomes.   | 37 |
| 5.2 | Self-supervised robotic setup for learning grasp2vec and goal-conditioned grasping. Left: A KUKA iiwa uses a monocular RGB camera to pick up objects from a bin. Right: The same setup in simulation. Bottom right: Example images that are fed into the Q function shown in Figure 5.3. . . . .   | 38 |
| 5.3 | (a) The Grasp2Vec architecture. We use the first 3 blocks of Resnet-50 V2 to form $\phi_s$ and $\phi_o$ , which are randomly initialized. $\phi_s(s_{\text{pre}})$ and $\phi_s(s_{\text{post}})$ have tied weights. The output of the third resnet block is passed through a ReLU to yield a spatial feature map we call $\phi_{\text{spatial}}$ . This is then mean pooled globally to output the single vector embeddings $\phi_s$ and $\phi_o$ . The n-pairs loss is applied as described in Section 5.3. (b) The instance grasping architecture is conditioned on Grasp2Vec goal embeddings of goal images, and is trained via Q-learning. . . . . | 39 |
| 5.4 | An analysis of our learned embeddings. Examples shown were chosen at random from the dataset. . . . .  | 41 |
| 5.5 | Instance grasping with grasp2vec outcome similarity. The goal image is shown on the left, and the center shows the robot during the trajectory. The final outcome image is on the right along with its grasp2vec similarity to the goal. . . . .   | 46 |
| 6.1 | Compositional plan vectors embed tasks into a space where adding two vectors represents the composition of the tasks, and subtracting a sub-task leaves an embedding of the remaining sub-tasks needed for the task. . . . .   | 49 |
| 6.2 | By adding the CPVs for two different tasks, we obtain the CPV for the composition of the tasks. To determine what steps are left in the task, the policy subtracts the embedding of its current trajectory from the reference CPV. . . . .   | 50 |

|     |  |    |
|-----|--|----|
| 6.3 | Illustrations of the 5 skills in the crafting environment. To ChopTree, the agent must pick up the axe and bring it to the tree, which transforms the tree into logs. To BuildHouse, the agent picks up the hammer and brings it to logs to transform them into a house. To MakeBread, the agent brings the axe to the wheat which transforms it into bread. The agent eats bread if it lands on a state that contains bread. To BreakRock, the agent picks up a hammer and destroys the rock. . . .   | 53 |
| 6.4 | Two example skills from the pick and place environment. Time evolves from left to right. If the relevant objects are in the box, the agent must first remove the lid to interact with the object and also return the lid to the box in order to complete a task. . . . .   | 54 |
| 6.5 | The network architecture used for the crafting environment. Orange denotes convolutional layers and dark blue denotes fully connected layers. The trajectories $(\mathbf{o}_0^{\text{ref}}, \mathbf{o}_T^{\text{ref}})$ and $(\mathbf{o}_0, \mathbf{o}_t)$ are each passed through $g$ (the pale green box) independently, but with shared weights. The current observation $\mathbf{o}_t$ is processed through a separate convolutional network before being concatenated with the vector $g(\mathbf{o}_0^{\text{ref}}, \mathbf{o}_T^{\text{ref}}) - g(\mathbf{o}_0, \mathbf{o}_t)$ . . . . .                 | 55 |
| A.2 | Training objects. Recognize deformable object (bag) in a large amount of clutter.  | 75 |
| A.3 | Testing objects. Recognizes correct object, even when goal is presented from a different pose than the object's pose in the bin. . . . .   | 76 |
| A.4 | Testing objects. Recognizes objects by parts when the goal object is occluded by the gripper. . . . .  | 77 |
| A.5 | Training objects. Embedding localizes objects with the correct colors but wrong (though similar) shape. . . . .  | 78 |
| A.6 | Test objects. Failed grasp incorrectly labeled as successful, resulting in an empty (meaningless) goal for localization. . . . .   | 78 |
| A.1 | This table illustrates the object recall property. From left to right: The scene before grasping, the grasped object, the outcome image retrieved by subtracting the postgrasp scene embedding from the pregrasp scene embedding, and lastly the postgrasp scene. We show example successes and failures (the later are marked with a red X). Failures occur in the test object set because multiple white objects had similar embeddings. Failures occur in the real data because the diversity of objects is very high, which likely makes the embeddings less partitioned between object instances. . . . . | 80 |
| A.7 | (a) The detection analysis with an untrained model. This verifies that our loss, rather than the architecture on it's own, enables the detection property. (b) The failure of localization indicates that Imagenet features are not consistent between scene and outcome images, probably because of resolution and pose differences.  | 81 |
| A.8 | Objects used for evaluation on unseen (test) objects. . . . .  | 83 |

|     |   |    |
|-----|---|----|
| B.1 | The object-centric network architecture we use for the 3D grasping environment. Because the observations include the concatenated positions of the objects in the scene, the policy chooses a grasp position by predicting a discrete classification over the objects grasping at the weighted sum of the object positions. The classification logits are passed back to the network to output the position at which to place the object. . . . . | 86 |
| B.2 | First person view in VizDoom env. . . . .   | 86 |

# List of Tables

|     |  |    |
|-----|--|----|
| 3.1 | We evaluate the zero shot performance of the 4-link arm reach to the black block. The numbers shown in the table are average distances from the end-effector to the black block over the last 5 timesteps of a sample from the policy; a perfect policy would get 0. We see that composing the 4-link robot module with the reach to black-block task module generates very good performance (under the column Ours), while composing a different task module with the correct robot module, or running a random policy does quite poorly. . . . .                           | 16 |
| 3.2 | Zero-shot results on the 4-link performing the block-pushing task from section 3.4.5. The values are the distance between the drawer and its target position averaged over the last five time steps of each sample. Forming the policy by composing the 4-link module with the block pushing module performs best even those modules were not trained together. Choosing the reach module instead performs on par with a random network. We show that the task and robot modules are able to generalize to unseen robot-task combinations without additional training. . . . | 19 |
| 5.1 | Quantitative study of Grasp2Vec embeddings. As we cannot expect weights trained on ImageNet to exhibit the retrieval property, we instead evaluate whether two nearest neighbor outcome images contain the same object. For object localization, the ImageNet baseline is performed the same as the grasp2vec evaluation. See Figure 5.4b for examples heatmaps and Appendix A.1 for example retrievals. . .   | 44 |
| 5.2 | Evaluation and ablation studies on a simulated instance grasping task, averaged over 700 trials. In simulation, the scene graph is accessed to evaluate ground-truth performance, but it is withheld from our learning algorithms. Performance is reported as percentage of grasps that picked up the user-specified object. Table reports early stopping scores for instance grasping on training objects and evaluation for the same checkpoint on test objects. Best numbers (for unsupervised approaches) in bold font. . . . .  | 46 |

|     |   |    |
|-----|---|----|
| 6.1 | <b>Evaluation of generalization and compositionality in the crafting environment.</b> Policies were trained on tasks using between 1 and 4 skills. We evaluate the policies conditioned on reference trajectories that use 4, 8, and 16 skills. We also evaluate the policies on the composition of skills: “2, 2” means that the embeddings of two demonstrations that each use 2 skills were added together, and the policy was conditioned on this sum. For the naïve model, we instead average the observations of the references, which performed somewhat better. All models are variants on the architecture in Figure 6.5. The max horizon is three times the average number of steps used by the expert for that length of task: 160, 280, and 550, respectively. Numbers are all success percentages. . . . . | 57 |
| 6.2 | <b>3D Pick and Place Results.</b> Each model was trained on tasks with 1 to 2 skills. We evaluate the models on tasks with 1 and 2 skills, as well as the compositions of two 1 skill tasks. For each model we list the success rate of the best epoch of training. All numbers are averaged over 100 tasks. All models are variants of the object-centric architecture, shown in the supplement. We find that the CPV architecture plus regularizations enable composing two reference trajectories better than other methods. . . . .   | 58 |
| B.1 | <b>ViZDoom Navigation Results.</b> All numbers are success rates of arriving within 1 meter of each waypoint. . . . .   | 86 |

## Acknowledgments

I would like to thank my advisors Sergey Levine, Trevor Darrell, and Pieter Abbeel for their mentorship and support for the last five and a half year. Being able to work with three amazing researchers enabled me to pursue research directions across the disciplines of computer vision, robotics, and deep representation learning. I would like to thank Alison Gopnik for being on my qualification exam and thesis committees as well as my collaborators, Abhishek Gupta, Dequan Wang, Daniel Geng, Eric Jang, Charles Sun, Brian Yang, Glen Berseth Vincent Vanhoucke, Fisher Yu, and Philipp Krähenbühl, without whom this work would not have been possible. I also thank all the wonderful people in RAIL, RLL, and Darrell-Group for making me feel welcome at Berkeley and melding collegiality with friendship. In particular, I'd like to thank Abhishek Gupta, Gregory Khan, Anusha Nagabandi, Kate Rakelly, Marvin Zhang, Michael Janner, Vitchyr Pong, Alex Lee, Ashvin Nair, Nick Reinhardt, Glen Berseth, and many more for making social lunches my favorite part of the workday. Your insights into research problems and willingness to discuss any idea to its limit made the labs feel intellectually open and helped me grow as a researcher.

I would like to thank Grace Kuo and Regina Eckert for your friendship and for all the mac and cheeses, dumplings, and fries we shared. I also thank my college friends for keeping our book club going, and my high school friends for staying connected after so many years. I would like to give a special thank you to Ben Mildenhall for the love and support throughout this PhD. Finally, I would like to thank my family for always believing in me and encouraging me to be the best version of myself.



# Chapter 1

## Introduction

Advances in deep learning have resulted in impressive results across many domains, including computer vision and robotics. The key to these increased capabilities is the ability of deep neural networks to improve based on data (“learn”), rather than relying on humans to iterate on a system manually. However, if deep networks are only thought of as learning black boxes, where data goes in and a trained system comes out, we lose the opportunity to shape *how* the trained network is making decisions. For example, if we train a network to classify dogs and wolves, will it look at their ears, or will it just look for snow around the animal [117]? As long as the trained network is only used on images similar to the ones seen during training, it doesn’t matter, but if we expect the network to *generalize* to new images performance could drop if they depict wolves inside houses. Since we cannot yet train networks on all scenarios, how can we set up a learning system to have the most chance of generalizing to new situations while still benefiting from learning directly from data? We seek to answer this question in the domain of robot learning.

### 1.1 Modularity and Compositionality

A common paradigm in robot learning is to train on a particular distribution of environment or tasks, and then test the robot’s behaviors on the same distribution. While the particular instantiations of environment may differ between train and test, the learned policy is only asked to interpolate within what it has seen, rather than extrapolate to out-of-distribution situations.

In this thesis, we explore the problem of learning robotic skills that generalize to new situations. Taken at its extreme this is of course an impossible task: a robot cannot be expected to know what to do with a sauce pan if it has only been trained on mowing grass. Instead, we focus on the problem of generalizing learned behaviors to novel *compositions* of previously seen tasks. For example, a robot that has learned to pick up a cup with its right hand and slice vegetables with its left hand should be able combine these learned behaviors and infer how to pick up a cup with the left hand and vice versa, as illustrated in Figure 1.1.

“Compositionality” as a problem statement in machine learning can take various forms. At its center is the assumption that concepts we want agents to learn are all made up of subparts, and that these subparts are reused across many concepts. We hypothesize that learning algorithms that take advantage of the compositional structure of tasks should perform better than black-box models when presented with novel compositions of previously seen subparts. In this work we focus on a few types of compositionality: the compositions of different robot morphologies performing different behaviors, the compositions of behaviors and objects, the compositions of objects within visual scenes, and finally the compositions of tasks with each other over time.

## 1.2 Organization, and Contributions

We begin in Chapter 3 by demonstrating how a modular network architecture can enable generalization to novel compositions of robots and tasks. We propose to divide a neural network into modules that each assigned to a robot or a task. Without supervising how these modules should interface, we train them together from data of different robots performance different tasks. We show that by recombining the modules in novels ways at evaluation, the tasks can be transferred between the robots. This simple approach illustrates the power of neural network architecture for encoding structural biases. This structure can lead to improved compositional generalization while retaining the benefits of learning from data. This work was published at the International Conference on Robotics and Automation (ICRA) in 2017.

In Chapter 4 we study an *object-centric* approach to modular robotic learning, where we structure image-conditioned policies as the composition of object localization and motor control. Rather than train each component independently, we develop an attention based object detector that can be trained directly from a handful demonstrations of the robot task. This modular structure enables swapping different object attention parameters to change how the task is performed. Additionally, by basing the detection on deep features trained on an image classification dataset, we transfer visual robustness from the dataset to the robot’s behavior: if the robot is trained with a brown mug, the object attention detects all kinds of mugs without having been explicitly trained to do so. This work was published at ICRA in 2018.

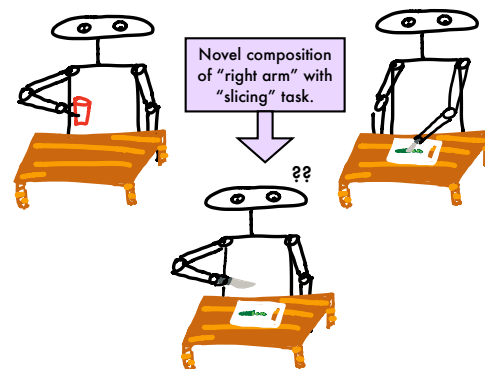


Figure 1.1: While learning based methods can learn behaviors given enough data for that task, they often fail to perform well in novel situations, even if those situations are compositions of the previously seen tasks.

In Chapter 5 we extend the ideas in the previous chapter by proposing a self-supervised approach to learning object-centric representations. Rather than relying on fixed dataset of human-labeled images, we posit that a robot should be able to learn to identify novel objects directly by interacting with them. When a robot picks up an object, the images before and after the object was grasped contain the information needed to learn about the object. By contrasting the difference between these images with a close up of the object in the robots gripper, the robot can learn to localize and grasp novel objects when presented with pictures of them. We achieve this by learning to embed images into a vector space where adding and subtracting vectors corresponds to adding and removing objects from the scene. This work was published at the Conference on Robot Learning (CoRL) in 2018.

Chapter 6 revisits the problem of task compositionality, this time from a temporal perspective. We previously showed that images containing multiple objects can be embedded into a vector equal to the sum of the embeddings of each of the objects. We hypothesize that embeddings of tasks can be learned with the same property: the embedding of a task should equal the sum of the embeddings of the sub-tasks within it. By applying this principle to a multi-task imitation learning setting, we are able to train policies that can perform novel, more complex, compositions of subtasks than what was seen during training. To conclude, Chapter 7 reflects on the limitations of this work and offers ideas for future research in compositionality and modularity for robot learning. This work was published at Neural Information Processing Systems (NeurIPS) in 2019.

# Chapter 2

## Related Work

Reinforcement learning (RL) can automate a wide variety of robotic skills [27, 71, 110, 112], but learning each new skill requires considerable real-world data collection and manual representation engineering to design policy classes or features. Using deep reinforcement learning to train general purpose neural network policies alleviates some of the burden of manual representation engineering by using expressive policy classes, but exacerbates the challenge of data collection, since such methods tend to be less efficient than RL with low-dimensional, hand-designed representations. The ability to transfer information from one skill or environment to another is crucial for robot learning in the real world, where collecting data is expensive and slow, as well as for training robots that can perform many tasks and which are robust to novel settings. Transfer learning has been recognized as an important direction in robotic learning [138, 11, 12, 31, 73], due to its potential for reducing the burden of expensive on-policy data collection for learning large repertoires of complex skills. [115] and [93] transfer between tasks by storing symbolic knowledge in knowledge bases.

We are particularly focused on transferring skills and generalization through compositional learning. Guestrin et al. learned to play many versions of a computer strategy game by decomposing the value function into different domains [46]. The PG-Ella algorithm uses policy gradients for sequential multitask learning [2]. Past work in transfer on robotics domains includes shaping the target reward function from the source policy [74, 96] and learning a mapping between tasks [139]. Another compositional approach used by [32] is to split each task into sub-tasks and transfer the sub-tasks between tasks. An early work by Caruana uses back-propagation to learn many tasks jointly [11]. Our work in Chapter 3, where we decompose neural networks into modules for tasks and robots, is based on the hypothesis that deep learning methods can find good representations for composing skills. The concurrent work by Andreas et al. is based on the same hypothesis, but focuses more on building primitive skills than on transferring tasks between robots [5].

The methods in Chapter 4 build upon the vast body of work in machine perception. Machine perception has often been concerned with object detection and localization at either the instance level or category level. Instance level approaches have used methods such as 3D representations [121] [57], keypoints [91], or deep neural networks [55]. At the category level,

deep learning of detectors with large datasets has been successful [116, 51]. Category-level reasoning is appealing because it can generalize across object instances, but is limited by the labels included in a dataset. Instance level reasoning can be limited in how well it generalizes to visual changes. Guadarrama et al. use textual descriptions of objects to combine task and category level reasoning [45].

Chapter 5 makes use of interaction and self-supervision to learn task-relevant object features. Past works on interactive learning have used egomotion of a mobile agent or poking motions [1, 34, 113, 63, 66, 41] to provide data-efficient learning of perception and control. Our approach learns representations that abstract away position and appearance, while preserving object identity and the combinatorial structure in the world (i.e., which objects are present) via a single feature vector. A recent body of work has focused on deep visuomotor policies for picking up arbitrary objects from RGB images [114, 84, 79, 151]. By automatically detecting whether some object was grasped, these methods can learn without human supervision.

Our work also draws from prior work in metric learning. For many types of high dimensional inputs, Euclidean distances are often meaningless in the raw input space. Words represented as one-hot vectors are equally distant from all other words, and images of the same scene may have entirely different pixel values if the viewpoint is shifted slightly. This has motivated learning representations of language and images that respect desirable properties. [17] showed that a simple contrastive loss can be used to learn face embeddings. A similar method was also used on image patches to learn general image features [127]. Word2vec found that word representations trained to be predictive of their neighbor words support some level of addition and subtraction [97, 85, 70]. More recently, Nagarajan used a contrastive approach in learning decomposable embeddings of images by representing objects as vectors and attributes as transformations of those vectors [105]. These methods motivate our goal of learning an embedding space over tasks that supports transformations such as addition and subtraction. Notably, these methods don't rely on explicit regularization for arithmetic operations, but rather use a simple objective combined with the right model structure to allow a compositional representation to emerge. Our approach also uses a simple end-to-end policy learning objective, combined with a structural constraint that leads to compositionality.

Chapter 6 combines the idea of learning arithmetic embeddings spaces with hierarchical multi-task imitation learning. Hierarchical RL algorithms learn representations of sub-tasks explicitly, by using primitives or goal-conditioning [20, 108, 89, 42, 94, 25, 8, 145], or by combining multiple Q-functions [48, 129]. Our approach does not learn explicit primitives or skills, but instead aims to summarize the task via a compositional task embedding. A number of prior works have also sought to learn policies that are conditioned on a goal or task [72, 26, 77, 133, 22, 47, 54, 95, 23, 120], but without explicitly considering compositionality.

## Chapter 3

# Learning Modular Networks for Multi-Task and Multi-Robot Transfer

### 3.1 Introduction

Deep reinforcement learning (RL) has been successful in multiple domains, including learning to play Atari games [101], simulated and real locomotion [123, 103] and robotic manipulation [83]. The onerous data requirements for deep RL make transfer learning appealing, but the policies learned by these algorithms lack clear structure, making it difficult to leverage what was learned previously for a new task or a new robot. The relationship between the optimal policies for different combinations of tasks and robots is not immediately clear, and doing transfer via finetuning does not work well for robotic RL domains due to the lack of direct supervision in the target domain.

However, much of what needs to be learned for robotic skills (dynamics, perception, task steps) is decoupled between the task and the robot. Part of the information gained during learning would help a new robot learn the task, and part of it would be useful in performing a new task with the same robot. Instead of throwing away experience on past tasks, we propose learning structured policies that decompose in a way that we can use transfer learning to help a robot benefit from its own past experience on other tasks, as well as from the experience of other, morphologically different robots, to learn new tasks more quickly.

In this chapter, we address the problem of transferring experience across different robots and tasks. Specifically, we consider the problem of transferring information across robots with varying morphology, including varying numbers of links and joints and across a diverse range of tasks. The discrepancy in the morphologies of the robots and the goals of the tasks prevents us from directly reusing policies learned on multiple tasks or robots for a new combination, and requires us to instead devise a novel modular approach to policy learning. An additional difficulty is determining which information can be transferred to a new robot and which can be transferred to a new task. As discussed in Chapter 1, consider a robot that has learned to slice vegetables. Given a new robot which only knows how to

pickup a cup, we would like to transfer something about slicing from the first to the second robot, which when combined with the second robot’s experience with the cup, would help it perform the slicing task. The first robot would transfer task information to the second, while the second robot would transfer its understanding of its own dynamics and kinematics from picking up to slicing.

In this chapter, we explore modular decomposable policies that are amenable to cross-robot and cross-task transfer. By separating the learned policies into a task-specific and robot-specific component, we can train the same task-specific component across all robots, and the same robot-specific component across all tasks. The robot and task-specific modules can then be mixed and matched to execute new task and robot combinations or, in the case of particularly difficult combinations, jump start the learning process from a good initialization. In order to produce this decomposition of policies into task-relevant and robot-relevant information, we show that policies represented by neural networks can be decomposed into task-specific or robot-specific modules. We demonstrate that these modules can be trained on a set of robot-task combinations and can be composed to enable zero-shot performance or significantly sped up learning for unseen robot-task combinations.

Our contributions are as follows:

1. We address robotic multi-task and transfer learning by training policy modules that are decomposed over robots and tasks, so as to handle novel robot-task combinations with minimal additional training.
2. We analyze regularization techniques that force the modules to acquire a generalizable bottleneck interface.
3. We present a detailed evaluation of our method on a range of simulated tasks for both visual and non-visual policies.

## 3.2 Relation to prior work

Our work differs from prior methods in that we explicitly consider transfer across tasks with two factors of variation, which in our experiments are robot identity and task identity. This allows us to decompose the policy into robot-specific and task-specific modules, which perform zero-shot transfer by recombining novel pairs of modules. Our method is complementary to prior transfer learning techniques in that we address primarily the question of policy representation, while prior methods focus on algorithmic questions.

Beyond robotic learning, recent work in computer vision and other passive perception domains has explored both transfer learning and recombination of neural network modules. Pretraining is a common transfer learning technique in deep learning [30]. However, pretraining cannot provide zero-shot generalization, and finetuning is ill-defined outside of supervised learning. Domain adaptation techniques have been used to adapt training data in the face of systematic domain shift [143], and more recently, work on modular networks for visual

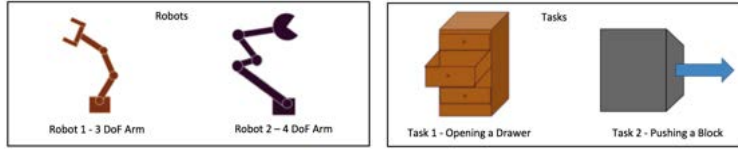


Figure 3.1: The 3DoF and a 4DoF robot which specify one degree of variation (robots) in the universe described in Section 3.3 as well as the tasks of opening a drawer and pushing a block which specify the other degree of variation (tasks) in the universe.

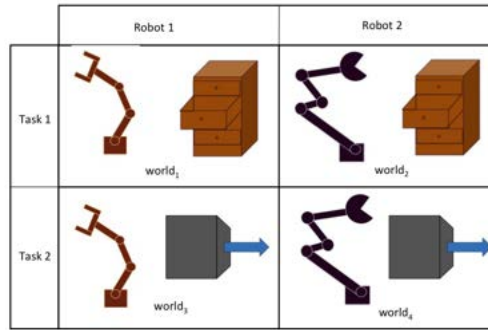


Figure 3.2: The possible worlds enumerated for all combinations of tasks and robots for the universe described in Section 3.3

question answering has been demonstrated with good results [5]. Our method differs from these prior approaches by directly considering robotic policy learning, where the policy must consider both the invariances and task-relevant differences across domains.

Although our method is largely agnostic to the choice of policy learning algorithm, we use the guided policy search method in our experiments [83]. This algorithm allows us to train high-dimensional neural network policy representations, which can be readily decomposed into multiple interconnected modules. Other recent work on high-dimensional neural network policy search has studied continuous control tasks for simulated robots [123, 86], playing Atari games [101], and other tasks [106]. Recent work on progressive neural network also proposes a representation suitable for transfer across Atari games [119], but does not provide for zero-shot generalization to new domains, and work by Braylen et al. used evolutionary algorithms to recombine networks trained for different Atari games, but again did not demonstrate direct zero-shot generalization [9]. We further emphasize that our approach is not in fact specific to neural networks, and our presentation of the method describes a generic framework of composable policy modules that can easily be extended to other representations.



### 3.3 Modular Policy Networks

The problem setting that this work addresses is enabling transfer across situations that can vary along some predefined discrete degrees of variation (DoVs). These DoVs can be different robot morphologies, different task goals, different object characteristics, and so forth. We define a “world”  $w$  to be an instantiation of these DoVs, and our “universe”  $\mathcal{U}$  to be the set of all possible worlds. To illustrate this formalism, consider a universe with the following 2 DoVs: robot structure (3 DoF and 4 DoF), and task (open drawer and pushing a block). This universe would have 4 possible worlds: 3 DoF arm opening a drawer, 3 DoF arm pushing a block, 4 DoF arm opening a drawer, and 4 DoF arm pushing a block.

Learning a single policy to act in all worlds is non-optimal because differences in the degrees of variation result in different optimal policies. Strategies required to push a block are quite different from those for a drawer, and robots with different numbers of joints would require different controls.

Standard reinforcement learning algorithms would treat each world  $w$  as a separate problem and learn an optimal policy for that world from scratch. However, the worlds have overlap: although a 3 DoF arm pushing a block and a 3 DoF arm opening a drawer are doing different tasks, they share the same arm, and thus will have commonalities in their optimal policy. This concept can be extended to other degrees of variation, such as when different arms perform the same task, or when the same arm interacts with different objects. Using this notion of commonality between worlds that share some DoVs, we tackle the problem of training policies for a subset of all worlds in a universe, and use these to enable fast transfer to new unseen worlds. Our experiments operate in the regime of 2 DoVs, which we take to be the identity of the robot and the identity of the task. However, we emphasize that this formalism can be extended to include variations like different objects, different environment dynamics, and so forth. In our subsequent method description, we adhere to the regime specified above, where our universe has 2 DoVs: the robot and the task being performed. We use  $R$  to denote the number of robots and  $K$  to denote the number of tasks. The robots can have different system parameters, such as link length, configuration, geometry, and even state and action spaces of completely different dimensionality, with different numbers of joints and actuators. We assume that the  $K$  tasks are achievable by all of the robots.

#### 3.3.1 Preliminaries

For each world  $w$ , let us define observations  $o_w$  and controls  $u_w$ . The observations  $o_w$  are the input that a robot would receive at test time, which could include images, encoder readings, motion capture markers, etc. In the case of complete information, the observations  $o_w$  would be the full state  $x_w$ . The controls  $u_w$  are the commands sent to the robot’s motors, which could be joint torques, velocities, positions, etc. We assume access to a policy optimization algorithm that can perform policy search to learn a separate optimal policy  $\pi_w$  for each world  $w$ . A policy  $\pi_w(u_w|o_w)$  specifies a distribution over controls  $u_w$  given an observation  $o_w$ . A policy search algorithm aims to search in policy space to find optimal parameters for  $\pi_w$ .

which minimize the expected sum of costs  $E_{\pi_w}(\sum_{t=0}^T c(o_w, t))$ . Given an optimal policy  $\pi_w^*$ , we can draw actions  $u_w$  given observations  $o_w$  from the distribution  $\pi_w^*(u_w|o_w)$ .

For most worlds we consider, an observation  $o_w$  can be split into a robot-specific “intrinsic” observation  $o_{w,\mathcal{R}}$  containing elements of the observation corresponding to the robot itself, and a task-specific “extrinsic” observation  $o_{w,\mathcal{T}}$  containing elements of the state corresponding to the task being performed.  $o_{w,\mathcal{R}}$  could include robot joint state and sensor readings, while  $o_{w,\mathcal{T}}$  could include images, object locations, and the position of the robot’s end-effector. We assume that the state can be decomposed in the same way into  $x_{w,\mathcal{R}}$  and  $x_{w,\mathcal{T}}$ . In order to decompose the policy by tasks and robots, we assume that the cost can be decomposed into a term that depends on the intrinsic state, and a term that depends on the extrinsic state:  $c(x_w, u_w) = c_{\mathcal{R}}(x_{w,\mathcal{R}}, u_w) + c_{\mathcal{T}}(x_{w,\mathcal{T}})$ , where the actions only affect the robot-dependent term, since they are inherently intrinsic. This assumption is reasonable, as the cost tends to be in terms of object locations and torque penalties. This decomposition of states and observations is crucial to being able to learn modular policies, as explained in Section 3.3.2.

### 3.3.2 Modularity

The problem we tackle is that of transferring information along values of each degree of variation while the remaining DoVs change. We intuit that for a particular degree of variation, all worlds with that DoV set to the same value can share some aspect of their policies. Going back to our 2-DoV universe as shown in Fig. 3.2, with 3 DoF and 4 DoF arms, performing the tasks of opening a drawer and pushing a block, consider 2 of the possible worlds:  $w_1$ , which is a 3 DoF arm opening a drawer, and  $w_3$ , a 3 DoF arm pushing a block. Although these worlds require different strategies due to the different tasks being performed, the robots are the same and hence robot kinematics, and control dimensionality matches across both worlds. We hypothesize that, for a particular degree of variation, all worlds with that DoV set to the same value can share some aspect of their policies. This is achieved by making the policies modular, so that the policies for worlds  $w_1$  and  $w_3$  share a “3 DoF arm” part of the policy.

We let  $\pi_{w_{rk}}(u|o)$  be the policy for the world  $w$  with robot  $r$  performing task  $k$ . To make the notation clearer, let us say that  $\pi_{w_{rk}}(u|o)$  is a distribution parametrized by a function  $\phi_{w_{rk}}(o)$ . For example,  $\pi_{w_{rk}}(u|o)$  can be a Gaussian  $\mathcal{N}(\phi_{w_{rk}}(o), \Sigma)$  with mean set to  $\phi_{w_{rk}}(o)$ , and  $\phi_{w_{rk}}$  can be an arbitrary function on observations.

For modular policies, we express  $\phi_{w_{rk}}(o)$  as a composition of functions  $f_r$  and  $g_k$  that represent robot-specific and task-specific parts of the policy for robot  $r$  and task  $k$ . Note that throughout our explanation,  $f$  shall represent robot-specific modules and  $g$  shall represent task-specific modules. These functions  $f_r$  and  $g_k$  act on the decomposed parts of the observation  $o_{w,\mathcal{R}}$  and  $o_{w,\mathcal{T}}$  respectively. The compositionality of modules can be represented as

$$\phi_{w_{rk}}(o_w) = \phi_{w_{rk}}(o_{w,\mathcal{T}}, o_{w,\mathcal{R}}) = f_r(g_k(o_{w,\mathcal{T}}), o_{w,\mathcal{R}}) \quad (3.1)$$

We refer to  $f_r$  as the robot-module for robot  $r$  and the function  $g_k$  as the task-module for task  $k$ . A separate set of parameters is used for each robot-module and task-module, such

that worlds with the same robot instantiation  $r$  would reuse the same robot module  $f_r$ , while worlds with the same task instantiation  $k$  would use the same task module  $g_k$ .

The reason the modules are composed in this particular order for the scenarios we consider is because we expect that the identity of the task would affect the task plan of the policy, while the robot configuration would affect the control output. An important point to note here is that the output of the task module  $g_k$  is not fixed or supervised to have a specific semantic meaning. Instead it is a latent representation that is learned while training the modules.

If we consider a larger number of DoVs, such as robots, tasks, and objects on which those tasks are performed, we could arrange the modules in an arbitrary ordering, so long as the ordering forms a directed acyclic graph (DAG). In the general case, each module receives inputs from its children and the observation corresponding to its DoV, and the root module outputs the action.

This definition of modularity now allows us to reuse modules across worlds. Given an unseen new world  $w_{\text{test}}$ , using robot  $r_{\text{test}}$  to perform task  $k_{\text{test}}$ , modules  $f_{r_{\text{test}}}$  and  $g_{k_{\text{test}}}$ , which have been learned from other worlds in the training set, can be composed to obtain a good policy. We do require that some world in the training set must have used robot  $r_{\text{test}}$  and some other world must have performed  $k_{\text{test}}$ , but they need not have ever been trained together. For the unseen world, we have

$$\begin{aligned}\phi_{w_{\text{test}}}(o_{\text{test}}) &= \phi_{w_{\text{test}}}(\phi_{w_{\text{test}}, \mathcal{T}}, o_{w_{\text{test}}, \mathcal{R}}) \\ &= f_{r_{\text{test}}}(g_{k_{\text{test}}}(\phi_{w_{\text{test}}, \mathcal{T}}), o_{w_{\text{test}}, \mathcal{R}})\end{aligned}$$

Note that we do not attempt to build a mapping relating different robots or tasks to each other, but instead use the experience of our desired robot on other task and the performance of our desired task with other robots to enable transfer of skills. As the number and variety of worlds which use a particular module increase, the module becomes increasingly invariant to changes in other DoV's, which is crucial for generalization. For example, as the number of robots being trained increases, each task module will need to work with various robot modules, which encourages them to become robot-agnostic. Similarly robot modules become task-agnostic as more tasks are performed with the same robot.

### 3.3.3 Architecture and Training

For this work, we choose to represent the modules  $f$  and  $g$  as neural networks due to their expressiveness and high capacity, as well as the ease with which neural networks can be composed. Specifically, for a world with robot  $r$  and task  $k$ , we choose  $\phi_{w_{rk}}(o)$  to be a neural network, and we choose the policy  $\pi_{w_{rk}}(u|o)$  to be a Gaussian with mean set to  $\phi_{w_{rk}}(o)$ , and a learned but observation-independent covariance. Each policy mean is thus a composition of two neural network modules  $f_r$ ,  $g_k$ , where the output of the task module is part of the input to the robot module.

Several training worlds are chosen with combinations of robots and tasks, such that every module has been trained for at least one world. This is illustrated in Fig. 3.3.

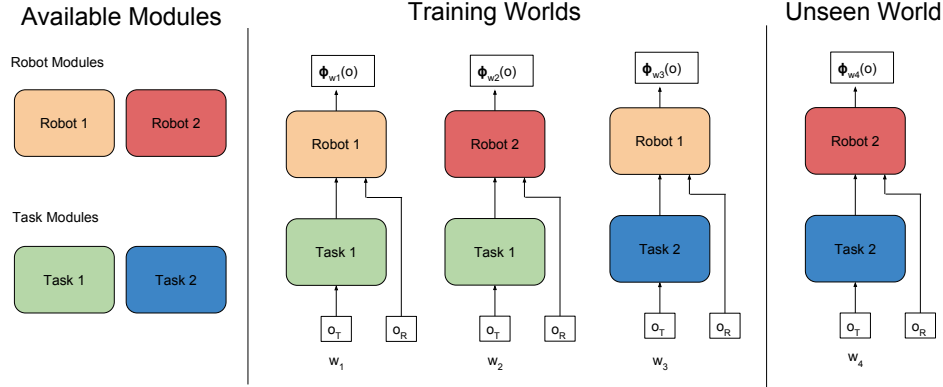


Figure 3.3: Modular policy composition for a universe with 2 tasks and 2 robots. There are 4 available modules - 2 task modules and 2 robot modules, and each module is a neural network. For the training worlds, these modules are composed together to form the individual policy networks. Modules of the same color share their weights. Policy networks for the same task share task modules and those for the same robot share robot modules. The training worlds are composed and then trained end-to-end. On encountering an unseen world, the appropriate previously trained modules are composed to give a policy capable of good zero-shot performance

The combined grid of policy networks, with weights tied between modules, are trained using inputs from all the worlds. This is done synchronously, by first collecting samples from each of the worlds and then feeding them forward through their corresponding modules to output predicted controls for each world. However, asynchronous training methods could also be explored in future work.

Formally, training proceeds by minimizing the reinforcement learning loss function  $\mathcal{L}$ , which is the sum of individual loss functions  $\mathcal{L}_w$  from each of the training worlds  $w$ . The details of the loss function and how it might be minimized, depends on the particular RL algorithm used to train the policies. In our experiments, we use guided policy search (GPS) [83], though other methods could be readily substituted. GPS proceeds by using local policy solvers to supervise the training of the final neural network policy, such that the loss for  $\mathcal{L}_w$  is a Euclidean norm loss for regression onto the generated training actions. A more standard policy gradient might instead use the approximate linearization of the expected return as the loss [123]. Most policy learning methods, including GPS and policy gradient methods, require computing the gradient of  $\log \pi(u|o)$  with respect to its parameters. In our method, as  $\pi_{w_{rk}}$  is parametrized by a neural network  $\phi_{w_{rk}}$  (with parameters  $\theta_k$  for the task module and parameters  $\theta_r$  for the robot module), we get the following gradients.

$$\frac{\partial \pi_{w_{rk}}}{\partial \theta_r} = \frac{\partial \pi_{w_{rk}}}{\partial \phi_{w_{rk}}} \frac{\partial \phi_{w_{rk}}}{\partial \theta_r}$$

$$\frac{\partial \pi_{w_{rk}}}{\partial \theta_k} = \frac{\partial \pi_{w_{rk}}}{\partial \phi_{w_{rk}}} \frac{\partial \phi_{w_{rk}}}{\theta_k}$$

As  $\phi_{w_{rk}} = f_r(g_k(o_w, \tau), o_w, \mathcal{R})$ , we can rewrite the gradients as follows,

$$\frac{\partial \pi_{w_{rk}}}{\partial \theta_r} = \frac{\partial \pi_{w_{rk}}}{\partial f_r} \frac{\partial f_r}{\partial \theta_r}$$

$$\frac{\partial \pi_{w_{rk}}}{\partial \theta_k} = \frac{\partial \pi_{w_{rk}}}{\partial f_r} \frac{\partial f_r}{\partial g_k} \frac{\partial g_k}{\partial \theta_k}$$

These gradients can be readily computed using the standard neural network backpropagation algorithm.

### 3.3.4 Regularization

In order to obtain zero-shot performance on unseen robot-task combinations, the modules must learn standardized interfaces. If, for example, a robot module overfits to the robot-task combinations seen during training, it might partition itself into different “receptors” for different tasks, instead of acquiring a task-invariant interface. With only a few robots and tasks (e.g. 3 robots and 3 tasks), we have found overfitting to be problematic. To mitigate this effect, we regularize our modules in two ways: by limiting the number of hidden units in the module interface, and by applying the dropout method, described below.

Limiting the number of hidden units in the outputs of the first module forces that module to pass on information compactly. A compact representation is less likely to be able to overfit by partitioning and specializing to each training world, since it would not be able to pass on enough information to the next module.

Dropout is a neural network regularization method that sets a random subset of the activations to 0 at each minibatch [132]. This prevents the network from depending too heavily on any particular hidden unit and instead builds redundancy into the weights. This limits the information flow between the task and robot modules, reducing their ability to overspecialize to the training conditions.

## 3.4 Experiments

To experimentally evaluate modular policy networks, we test our transfer approach on a number of simulated environments. For each experiment we use multiple robots and multiple tasks and demonstrate that using modular policy networks allows us to train on a subset of the possible worlds in a universe, and achieve faster or zero-shot learning for an unseen world. We evaluate our method against the baseline of training a separate policy network for each world instantiation. In order to evaluate our method on a challenging suite of simulated robotic tasks, we constructed several simulated environments using the MuJoCo physics simulator [142].

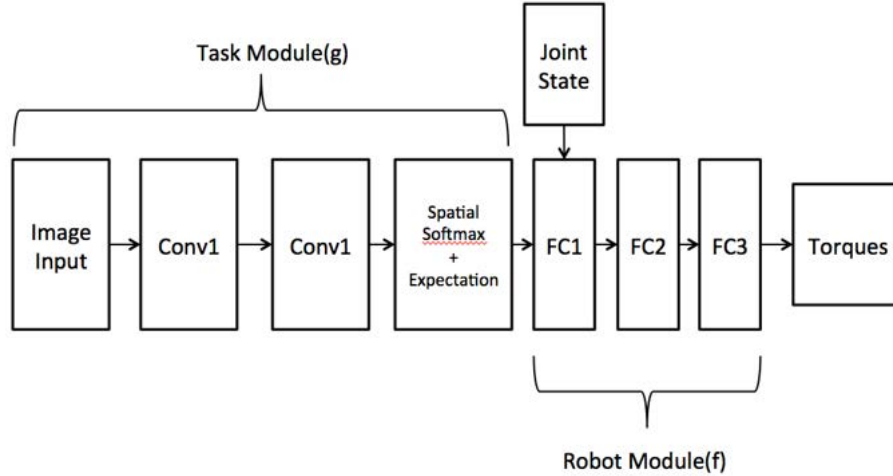


Figure 3.4: Basic visuomotor policy network for a single robot. The two convolutional layers and spatial softmax form the task module, while the last few fully connected layers form the robot module

We evaluate our method on tasks that involve discontinuous contacts, moving and grasping objects, and processing raw images from simulated cameras. For all experiments, further details and videos can be found at <https://sites.google.com/site/modularpolicynetworks/>

### 3.4.1 Reinforcement Learning Algorithm

The policy search algorithm we use to learn individual neural network policies is the guided policy search method described in [81]. This work splits policy search into trajectory optimization and supervised learning. To learn a number of local policies under unknown dynamics, the method uses a simple trajectory-centric reinforcement learning algorithm based on LQR. This algorithm generates simple local time-varying linear-Gaussian controllers from individual initial states of each system, with different controllers for different initial states. These controllers then provide supervision for training a global neural network policy using standard supervised learning, with a Euclidean loss regression objective. In our experiments, we use the BADMM-based variant of guided policy search which applies an additional penalty on the trajectory optimization for deviating from the neural network policy to stabilize the learning process [83]. This choice of learning algorithm enables us to train deep neural networks with a modest number of samples. However, more standard methods, such as policy gradient [148, 123] and actor-critic [86, 111] methods, could also be used with modular policy networks.







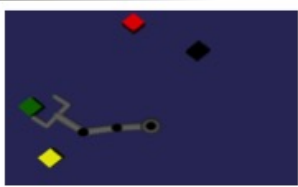
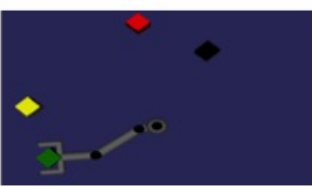



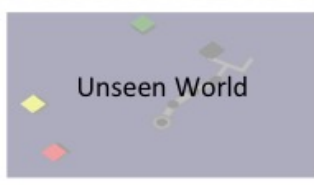
| Robots \ Tasks |  |   |  |
|----------------|--|---|--|
|                | 3link  | 3link different config  | 4link  |
| Reach Yellow   |   |   |   |
| Reach Red      |   |   |   |
| Reach Green    |   |   |   |
| Reach Black    |  |  |  |

Figure 3.5: Grid of tasks vs robots for the reaching colored blocks task in simulation described in 3.4.3. We train on all the worlds besides the 4link robot reaching to black, and test on this held out world.

### 3.4.2 Network Architecture

For tasks that require performing simulated vision, we used a neural network architecture as shown in Fig. 3.4. This architecture follows prior work [83]. In non-vision tasks, the convolutional layers are replaced with fully connected layers. In both cases, the task module also takes as input the position of the robot’s end-effector. Since the end-effector is present in all robots, we provide this to the task module so as to make it available to the policy in the earliest layers.

### 3.4.3 Reaching Colored Blocks in Simulation

In the first experiment, we evaluate a simple scenario that tests the ability of modular policy networks to properly disentangle task-specific and robot-specific factors in a visual perception

| Test Position | Random network | Wrong task module | <b>Ours</b> |
|---------------|----------------|-------------------|-------------|
| 1             | 1.16           | 2.34              | <b>0.12</b> |
| 2             | 1.29           | 1.75              | <b>0.28</b> |
| 3             | 1.35           | 1.65              | <b>0.21</b> |
| 4             | 1.29           | 2.41              | <b>0.08</b> |

Table 3.1: We evaluate the zero shot performance of the 4-link arm reach to the black block. The numbers shown in the table are average distances from the end-effector to the black block over the last 5 timesteps of a sample from the policy; a perfect policy would get 0. We see that composing the 4-link robot module with the reach to black-block task module generates very good performance (under the column Ours), while composing a different task module with the correct robot module, or running a random policy does quite poorly.

task. In this task, the environment consists of four colored blocks (red, green, yellow, and black) positioned at random locations in the workspace, and each task requires the robot to reach for a particular colored block. The universe for this scenario includes three robots: a 3-link arm, a 3-link arm with links of a different lengths, and a 4-link arm. Each robot has its own robot module, and is controlled at the level of joint torques. The size of the image passed is 80x64x3, and the state space for each robots is its joint angles and its joint angle velocities. This results in 15366 inputs for the 3-link robots and 15368 for the 4-link robots. An illustration of this task is shown in Figure 3.5.

Although this task is not kinematically difficult, it requires the task modules to pick up on the right cues, and the small number of tasks and robots makes overfitting a serious challenge. In order to evaluate zero-shot transfer in this setup, we train the modules on 11 out of the 12 possible world instantiations, with the 4 link robot reaching the black block being the unseen world. None of the other policies being learned can be trivially transferred to this world, as the 3 link robots have different dimensionality and the other task modules have never learned to reach towards other blocks. Successful transfer therefore requires perception and control to be decomposed cleanly across the task and robot modules. This is illustrated in Fig. 3.5.

We compare the performance of our method against two baselines: the first baseline involves executing a random policy, while the second involves running a policy learned for the 4 link robot but for a different colored block. These baselines are meant to test for trivial generalization from other tasks. The results, shown in Table 3.1, show that our method is able to perform the task well without any additional training, while the baselines have significant error. This illustrates that we are able to transfer visual recognition capabilities across robots, which is crucial for learning transferable visuomotor policies.







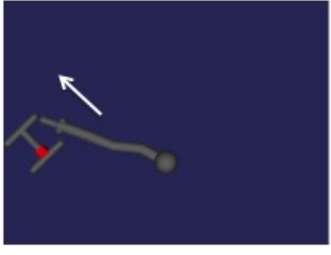
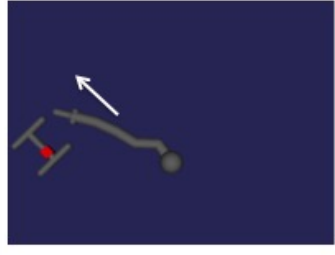
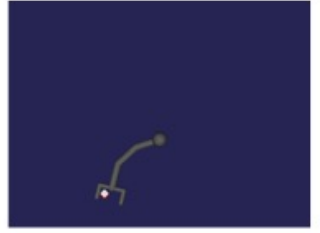


| Robots \ Tasks    | 3link  | 4link   | 5link  |
|-------------------|--|---|--|
| Horizontal Drawer |   |   |   |
| Vertical Drawer   |   |   |   |
| Block Push        |  |  |  |

Figure 3.6: Grid of tasks vs robots for the object manipulation tasks described in 3.4.4. The horizontal drawer tasks involve sliding the grey drawer horizontally to the target in the direction indicated by the arrow on the image. The vertical drawer tasks involve sliding the grey drawer vertically up in the direction indicated by the arrow on the image. The block push tasks involve pushing the white block to the red goal. All positions shown in this image are the final successful positions for each world.

### 3.4.4 Object Manipulation

In the next experiment, we evaluate our method on tasks that are more physically different to understand how well modular policy networks can transfer skills for manipulation tasks with complex contact dynamics. In this experiment, we use 3 robots and 3 tasks, as shown in Fig. 3.6. The robots have 3, 4, or 5 links, with state spaces of different dimensionality, and we input target coordinates instead of images. The tasks are: pulling a drawer horizontally, pushing a drawer vertically, and pushing a block to a target location. The arrows in the Fig. 3.6 indicate direction of motion for the tasks. Each of these tasks involve complex discontinuous contact dynamics at the point where the robot contacts the object. A grid of

tasks and robots is presented in Fig 3.6. In order to successfully transfer knowledge in this environment, kinematic properties of the tasks need to be transferred as well as dynamics of the robot.

We train 8 out of the 9 possible worlds, with the 3 link robot pulling the horizontal drawer being held out. Although our method does not successfully perform zero-shot generalization directly simply by composing the modules for the held-out world, the transferred policy provides an excellent initialization for further learning. Figure 3.7 shows the learning curves with policies initialized using four paradigms: composing modules appropriately using our method, composing modules using the incorrect task-module (vertical drawer), and learning from scratch with and without shaping. In this task, the shaping term in the cost encourages the robot’s gripper to reach for the drawer, while the standard cost without shaping simply considers the distance of the drawer from the target. Typically, tasks like this are extremely challenging to solve without shaping or a good initialization, since the robot must rely entirely on random exploration to learn to push the drawer before receiving any reward.

The results indicate that the transferred policy is able to learn the drawer task faster *without* shaping than the task can learned from scratch *with* shaping. When learning from scratch without shaping, the learning algorithm is unable to make progress at all. Therefore, if the shaping cost is not available, the policy obtained by transferring knowledge via modular policy networks is essential for successful learning. This indicates that, despite the wide variability between the tasks and robots and the small number of task-robot combinations, modular policy networks are able to transfer meaningful knowledge into the held-out world.

### 3.4.5 Visually Distinct Manipulation Tasks

In the third experiment, we evaluated our method on a set of worlds that require both vision and physically intricate manipulation skills to succeed. An illustration of the tasks and robots in the experiment is presented in Figure 3.8. The robots again include the 3-link arms with different link lengths and a 4-link robot. The tasks require reaching to a given position, pushing a block to a given target, and inserting a peg into a hole. The goals for each task are visually distinct, and the tasks require a different pattern of physical interactions to handle the contact dynamics.

We choose 8 out of the 9 possible worlds to train, with the held out world being the 4 link robot pushing the block. This task is particularly difficult, since it involves discontinuous dynamics. Modular policy networks were able to succeed at zero-shot transfer for this task, significantly outperforming both a random baseline and policies from different robot-task combinations. This indicates that the modules were able to decompose out both the perception and the kinematic goal of the task, with the robot modules handling robot-specific feedback control to determine the joint torques needed to realize a given task.

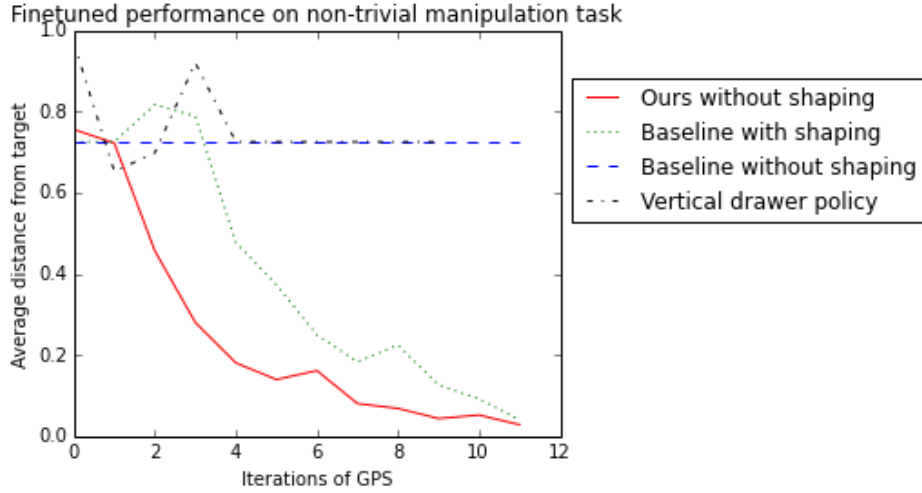


Figure 3.7: Results on the 3-link robot performing horizontal drawer pulling. The initialization from composing the 3-link robot module with the horizontal pull task module provides the fastest learning. Although the vertical drawer module was trained with the 3-link robot, the task is too different to directly transfer. Random initialization performs well with reward shaping, but without it is unable to learn the task.

| Test Position | Random network | Wrong task module | <b>Ours</b> |
|---------------|----------------|-------------------|-------------|
| 1             | 0.95           | 0.95              | <b>0.48</b> |
| 2             | 1.79           | 1.14              | <b>0.19</b> |
| 3             | 1.54           | 1.27              | <b>0.25</b> |
| 4             | 0.94           | 1.32              | <b>0.23</b> |

Table 3.2: Zero-shot results on the 4-link performing the block-pushing task from section 3.4.5. The values are the distance between the drawer and its target position averaged over the last five time steps of each sample. Forming the policy by composing the 4-link module with the block pushing module performs best even those modules were not trained together. Choosing the reach module instead performs on par with a random network. We show that the task and robot modules are able to generalize to unseen robot-task combinations without additional training.

### 3.5 Discussion and Future Work

In this chapter, we presented modular policy networks, a method for enabling multi-robot and multi-task transfer with reinforcement learning. Modular policy networks allow individual component modules for different degrees of variation, such as robots and tasks, to be trained together end-to-end using standard reinforcement learning algorithms. Once trained, the








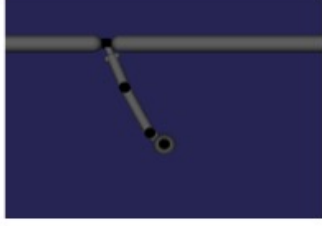

| Robots<br>Tasks | 3link  | 3link different config  | 4link  |
|-----------------|--|---|--|
| Reach           |   |   |   |
| Push            |   |   |   |
| Peg insert      |  |  |  |

Figure 3.8: R-obots for visually distinct tasks mentioned in 3.4.5. The reach task involves reaching the white target. The push task involves pushing the white block to the red goal. The peg insertion task involves inserting the peg at the tip of the robot into the hole specified by the black square. These tasks involve manipulation and are also visually distinct in that they do not use the colors in the same way (the target and block colors are not consistent). We train on all combinations besides the 4link robot pushing the block.

modules can be recombined to carry out new combinations of the degrees of variation, such as new robot-task combinations. The task-specific modules are robot-invariant, and the robot-specific modules are task-invariant. This invariance allows us to compose modules to perform tasks well for robot-task combinations that have never been seen before. In some cases, previously untrained combinations might generalize immediately to the new task, while in other cases, the composition of previously trained modules for a new previously unseen task can serve as a very good initialization for speeding up learning.

One of the limitations of the current work is that, by utilizing standard reinforcement learning algorithms, our method requires different task-robot combinations to be trained

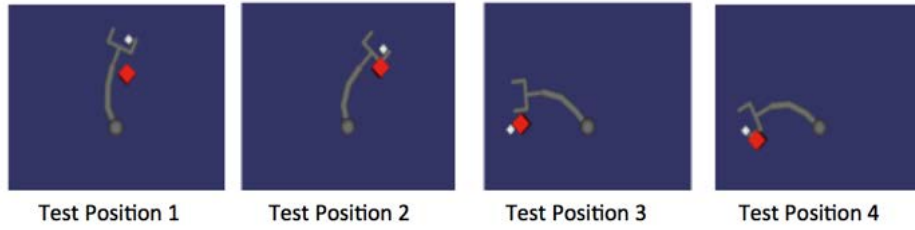


Figure 3.9: The final positions of the zero-shot performance of our method on the blockpushing task. Our method performs the task very well on zero shot and gets the white block to the red goal.

simultaneously. In a practical application, this might require multiple robots to be learning simultaneously. A promising direction for future work is to combine our approach with more traditional, sequential methods for transfer learning, such that the same robot can learn multiple tasks in sequence, and still benefit from modular networks. This would enable combined lifelong and multirobot learning, where multiple robots might learn distinct robot-specific modules, trained sequentially, while contributing to shared task-specific modules, trained in parallel. By training on a larger variety of robots and tasks, the generalization capability of modular policy networks is likely to increase also. This could make it practically to automatically train large repertoires of different skills across populations of heterogeneous robotic platforms.

Since publication, other papers have built upon this research direction. Modular networks have been used for hierarchical agents which must perform multiple tasks in a row [5, 107], or to navigate within multiple cities [98].

# Chapter 4

## Deep Object-Centric Representations for Generalizable Robot Learning

### 4.1 Introduction

The previous chapter’s use of a composable neural modules enabled transfer between tasks and robots. However, this method was only shown to work on manipulation tasks from state, or from very simple visual observations. In real world scenarios, the state of the world is not known, and visual observations are much more complex and varied. In this chapter we address learning visuomotor policies that generalize across objects and diverse visual conditions.

Recent years have seen impressive improvements in the performance of computer vision systems, brought about by larger datasets [118], improvements in computational capacity and GPU computing, and the widespread adoption of deep convolutional neural network models [53]. However, the gains in computer vision on standard benchmark problems such as ImageNet classification or object detection [87] do not necessarily translate directly into improved capability in *robotic* perception, and enabling a robot to perform complex tasks in unstructured real-world environments using visual perception remains a challenging open problem.

Part of this challenge for robot perception lies in the fact that the effectiveness of modern computer vision systems hinges in large part on the training data that is available. If the objects for a task happen to fall neatly into the labels of dataset, then using a trained object detector for perception makes sense. However, as shown in Figure 4.2, objects outside the label space may be labeled incorrectly or not at all, and objects that the robot should distinguish may be labeled as being the same category. These difficulties leave us with several unenviable alternatives: we can attempt to collect a large enough dataset for each task that we want the robot to do, painstakingly labeling our object of interest in a large number of images, or we can attempt to use the pretrained vision system directly, suffering from possible domain shift and a lack of flexibility. The Visual Genome dataset is notable for labelling

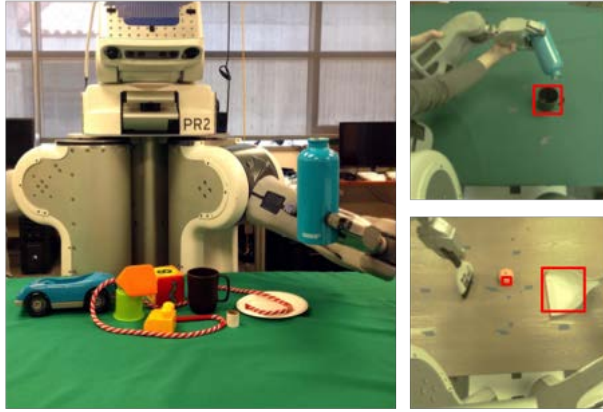


Figure 4.1: Deep object-centric representations learn to attend to task-relevant objects from just a few trajectories. The representation is robust to clutter and generalizes to new object instances.

regions with textual descriptions, bypassing the problem of rigid classes [75]. Another option is to train on both detection datasets and classification datasets with adaptation technique, as classification labels include more classes [58]. Similarly, using language to bridge instance matching and category matching can provide flexible object detection [45]. However, if the robot’s environment looks too different from the detector’s training data, the performance may suffer with limited recourse. A key property of our method is that small amounts of new data can be used to correct the model if it is not behaving as desired.

An alternative view of robotic vision has emerged in recent years with advances in deep reinforcement learning [60, 100], end-to-end learning from demonstrations [24, 150], and self-supervised learning [113, 114, 84]. These methods bypass the standard computer vision representation of class labels and bounding boxes and directly train task-specific models that predict actions or task success from raw visual observations. While these methods can overcome the challenges associated with large-scale semantic labeling and dataset bias by training directly on the task that the robot aims to solve, their ability to generalize is critically dependent on the distribution of training data. For example, if a robot must learn to pour liquid from a bottle into a cup, it can achieve instance-level proficiency with a moderate number of samples [38], but it must train on a huge number of bottles and cups in order to generalize at the category level. Switching from the standard vision framework to end-to-end training therefore allows us to bypass the need for costly human-provided semantic labels, but sacrifices the generalization that we can get from large computer vision datasets.

In this work, we seek to develop a robotic vision framework that operates on sets of objects rather than raw pixels, and leverages prior datasets to learn a generic object concept model. Our principal insight is that, if the robot will be using learning (e.g., reinforcement learning or learning from demonstration) to perform the final task that is set out before it, it

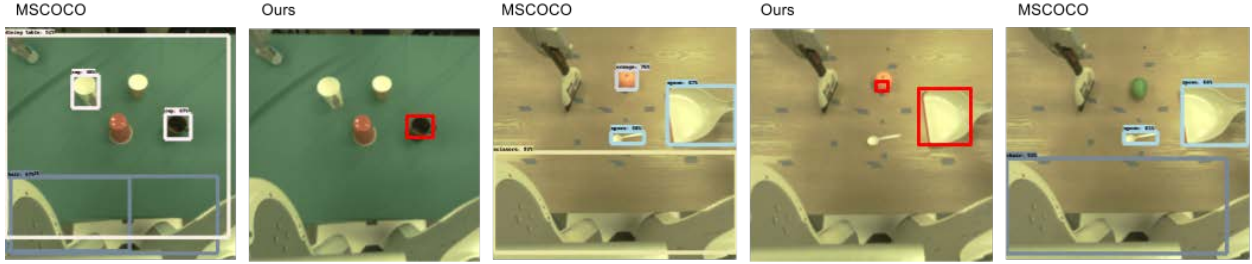


Figure 4.2: Faster RCNN trained on MSCOCO does not differentiate between cups and mugs, and gives higher probability to the cup, making it difficult to train a policy that needs to locate the mug. With our method, the attention can learn to prioritize mugs over cups. The dustpan is labeled as a spoon and thus can be distracted by other spoon-like objects. As limes are not in the MSCOCO dataset, the object detector does not label them.

does not require precise labels or segmentation. It simply needs to be able to consistently localize visual cues in the observed image that correlate with the objects that are necessary for it to perform the task. However, to learn generalizable policies, the visual cues should be semantic in nature such that a policy trained on one object instance can function on a similar object when desired.

We therefore take a two-stage approach to robotic vision: in the first stage, we construct an object-centric attentional prior based on a region proposal network. This stage requires a large amount of data, but does not require any task-specific data, and can therefore use a standard existing computer vision dataset. The second stage narrowly focuses this general-purpose attention by using a very small number of example trajectories, which can be provided by a human or collected automatically during the reinforcement learning process. This teaches the system to attend to task-relevant objects, while still benefiting from the generalizable representations present in the general-purpose attention. Furthermore, because the second stage is trained with only a handful of example trajectories, it makes it easy for the user to correct mistakes or control the class of objects that the system generalizes to, simply by providing additional demonstrations. For example, if the user needs a policy specific to a particular type of cup, they can simply provide demonstrations with other cups present in the scene, illustrating that they should be ignored. If the user prefers broader category-level generalization, for example to cause a robot generalize across all types of fruits, they might provide demonstrations that show interactions with fruits of different types. In all cases, the total number of provided trajectories remains very small (less than 15).

The main contribution of our work is a perception framework that facilitates generalization over objects and environments while requiring minimal data or supervision per task. Our method incorporates general-purpose object-centric priors in the form of an object attention trained on a large, generic computer vision dataset, and combines it with an extremely sample efficient task-specific attention mechanism that can either be learned from a very



small number of demonstrations, or even specified directly by a human operator. We show that this framework can be combined with reinforcement learning to enable a real-world robotic system to learn vision-based manipulation skills. Our experiments demonstrate that our approach achieves superior generalization to an end-to-end trained approach, through the incorporation of prior visual knowledge via the general-purpose attention. We illustrate how the user can control the degree of generalization by including or excluding other objects in the demonstrations. Our source code is available online in a stand-alone ROS package <sup>1</sup>. A video of results is also available <sup>2</sup>.

## 4.2 Relation to Prior Work

Our method combines prior knowledge about “objectness” from pretrained visual models with an attentional mechanism for learning to detect specific task relevant objects. A number of previous works have sought to combine general objectness priors with more specific object detection in the context of robotics and other visual tasks. Ekvall et al. used region proposals and SIFT features for quickly detecting objects in the context of SLAM [35]. Prior work used an active search approach where the camera could zoom in certain parts of the receptive field to search at higher resolutions [68]. In manipulation, SIFT features have also been used for 3D pose estimation and object localization, using object-specific training data gathered individually for the task [18, 136]. Similarly to these prior works, our method constrains the observations using an object-centric prior. However, we do not require object level supervision for each task, instead using visual features from a pretrained visual model to index into the proposals from the object-centric prior. This approach drastically reduces the engineering burden for each new task, picking out task-relevant objects from a few demonstrations, and provides good generalization over object appearance, lighting, and scale, as demonstrated in our experiments.

An alternative to building perception systems for task-specific objects is to learn the entire perception system end-to-end together with the control policy. A number of recent works have explored such end-to-end approaches in the context of skill learning, either for direct policy search [82, 113, 114], unsupervised learning of representations for control [44, 39], or learning predictive models [1, 37]. A major challenge with such methods is that their ability to generalize to varied scenes and objects depends entirely on the variety of objects and scenes seen during policy training. Some methods have sought to address this by collecting large amounts of data with many objects [114, 84]. In this work, we instead study how we can incorporate prior knowledge about objects from pretrained visual models, while still being able to train rich neural network control policies. In this way, we can obtain good generalization to appearance, lighting, and other nuisance variables, while only training the final policy on a single object instance and in a single scene.

---

<sup>1</sup><https://github.com/cdevin/objectattention>

<sup>2</sup><https://sites.google.com/berkeley.edu/object-representations>

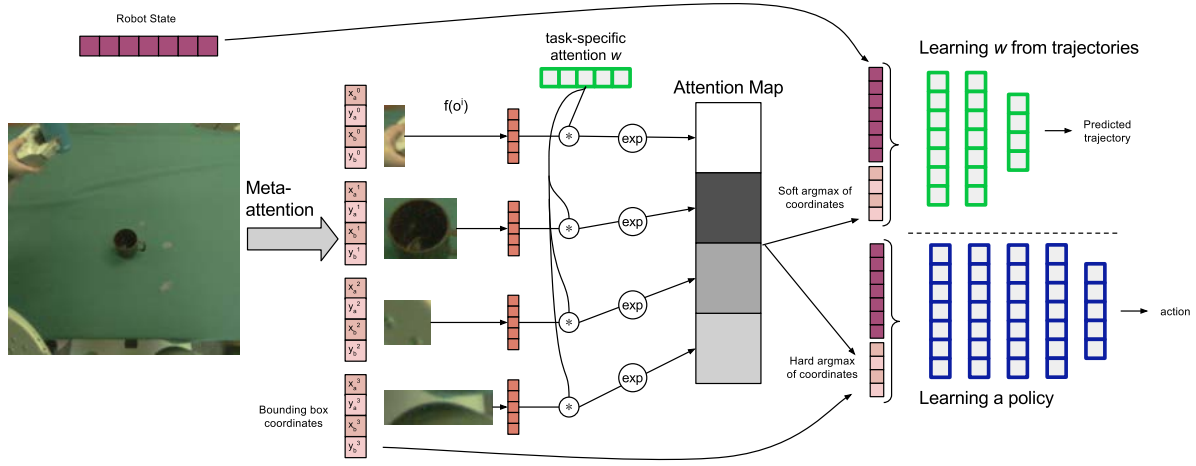


Figure 4.3: Method Overview. The parameters outlined in bright green are optimized during pretraining, while those outlined in dark blue are optimized during policy learning. The attention is trained to predict the movement seen in the provided demonstrations or trajectories. The “attention map” illustrates a soft attention where vectors  $f(o^i)$  close to  $w$  are given high probability (black) and ones far away have low probability (white). The distribution is regularized to have low entropy, and the weighted sum of bounding box coordinates is fed to the next layers of the bright green network. The policy (in blue) is trained with  $w$  held fixed, and the arg-max bounding box is used instead of the weighted average. Note: this diagram illustrates only a single attention vector  $w$ ; more attention vectors can be added as needed.

### 4.3 Deep Object-Centric Representations

The goal of our method is to provide a simple and efficient process for quickly acquiring useful visual representations in the context of policy learning. Specifically, we aim to compress an image into a vector  $\nu$  of object descriptors and select task-relevant objects from it. We impose an object-centric structure on our representation, which itself is learned from prior visual data in the form of standard computer vision image datasets. We define a 2-level hierarchy of attention over scenes for policy learning. The high level, which we call the **task-independent attention**, is shared for all tasks. The task-independent attention is intended to identify possible objects in the scene regardless of the task. The lower level, which we call **task-specific attention**, is learned per-task and identifies which of the possible objects is relevant to the task being performed.

---

**Algorithm 1** Robot Learning with Object Centric Representations

---

- 1: Train task-independent attention on an object detection dataset (shared for all tasks).
  - 2: Train a convolutional network  $f$  for image classification (shared for all tasks).
  - 3: **for** each task **do**
  - 4:     Collect demonstrations.
  - 5:     Learn task-specific attention  $W$  as described in Section 4.3.1 from the trajectories.
  - 6:     Train control policy using reinforcement learning, with the robot’s configuration and  $\nu$  as the observation inputs.  $W$  is fixed during this step.
- 

### 4.3.1 Task-Independent Attention

The task-independent attention is a function that takes an image and returns a set of object hypotheses  $\{o^i : i \in [0, N)\}$ . Each object hypothesis consists of a semantic component (given by  $f(o^i)$ ) and a position component (given by  $g(o^i)$ ). The semantic component describes the identity of the object with a feature vector, and the position component describes where the object is located within the image. Importantly, this task-independent attention is reused for all tasks without retraining, and can be trained entirely on standard vision datasets.

### 4.3.2 Task-Specific Attention

The goal of the task-specific attention is to choose which object(s) are relevant for a task given a small amount of task-specific data. In the tasks we examine, we assume that the relevant object(s) have consistent semantic features  $f(o^i)$  over the entire course of the task. For example, if the task is to pour into a mug, the object that looks like a mug should be selected, regardless of its current position. Thus, to select which objects to pay attention to, the model learns a *task-specific* attention over the semantic features  $f(o^i)$ . We assume that the user knows an upper bound  $K$  for the number of task-relevant objects. In order to be able to quickly learn the task-specific attention from only a small number of trajectories, it is parametrized as a matrix  $W \in \mathcal{R}^{K \times D}$ , where  $D$  is the dimension of  $f(o^i)$ . The probability of object proposal  $o^i$  being the  $k$ th relevant object is proportional to  $e^{W^{(k)\top} f(o^i)}$ .

## 4.4 Implementation

There are a number of possible choices in implementing our proposed object-centric representation. The choice of objectness prior may affect which objects are proposed, while the semantic features influence the generalizability of the task-specific attention.

### 4.4.1 Architecture

Although a number of task-independent attention mechanisms are possible, we use a region proposal method to provide a set of possible objects. The objects  $o^0, \dots, o^N$  are the proposed

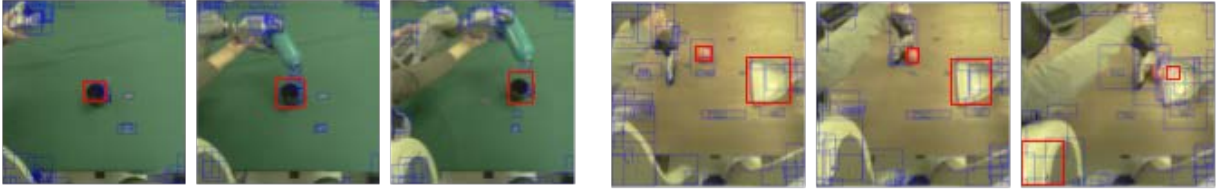


Figure 4.4: The region proposals (task-independent attention) are drawn in blue and the task-specific attended regions are drawn in red. For the pouring task, the attention locks on to the mug as its position defines the trajectory. For the sweeping task, we use two attention vectors, one attends to the orange and one attends to the dustpan, which each have variable starting positions.

crops and the position component  $g(o^i)$  are the bounding box coordinates of the proposal.

Because the task-specific attention is linear with respect to the semantic features, the choice of these semantic features is crucial for the flexibility and generalization capability of the method. While we could choose the features to simply correspond to semantic class (e.g., using classes from a standard image classification dataset), this would limit the flexibility of the method to identifying only those object classes. If we choose overly general features, such as a histogram of oriented gradients or even raw pixels, the task-specific attention would be too limited in its ability to generalize in the presence of changes in appearance, lighting, viewpoint, or pose. To strike the right balance between flexibility and generalization, we define the semantic component  $f$  to be a mean-pool over the region proposal crop of the convolutional features pretrained on ImageNet classification [118]. Such features have previously been shown to transfer effectively across visual perception tasks and provide a good general-purpose visual representation [128, 124]. An overview of our method is provided in Algorithm 1.

#### 4.4.2 Optimization

Given trajectories of a task, the objects that are relevant to the task will be predictive of future robot configurations. Trajectories could come from a variety of sources; in this chapter we use either kinesthetic demonstrations, or directly make use of the trajectories obtained during reinforcement learning (in our case, with guided policy search). We optimize for  $W$  as part of a larger neural network shown at the top of Figure 4.3 that aims to predict the next step in the trajectory: an action if available, or a change in position of the end-effector. The network for this is two hidden layers with 80 units each. In order to backpropagate through  $W$ , a soft attention mechanism is used. First, we use a Boltzmann distribution to obtain a

probability  $p(o^i|w_j)$  for each object proposal.

$$p(o^i|w_j) = \frac{e^{w_j^\top \frac{f(o^i)}{\|f(o^i)\|_2}}}{\sum_{i=0}^N e^{w_j^\top \frac{f(o^i)}{\|f(o^i)\|_2}}}$$

Then, the soft attention map is calculated by taking a weighted sum of the object locations.

$$\nu_{j,\text{soft}} = \sum_{i=0}^N g(o^i) p(o^i|w_j).$$

To obtain the prediction,  $\nu_{soft}$  is concatenated with robot joint state and end-effector state before being fed into the movement prediction network at the top of Figure 4.3. While  $f(o^i)$  is normalized for each  $o^i$ ,  $W$  is not normalized to allow the optimization to control the peakiness of the attention. To encourage more discrete attention distributions, the attention is regularized to have low entropy:

$$\mathcal{L}_{\text{ent}}(w) = \sum_{j=0}^M \sum_{i=0}^N -p(o^i|w_j) \log p(o^i|w_j)$$

The network is optimized with the Adam optimizer [69].

To better condition the optimization when the task-relevant objects are known, the task-specific attention can be initialized by providing one example crop of the desired object(s) before finetuning on the demonstration data. We found this to be unnecessary in the pouring task, which only has one object, but very useful in the seeping task.

## 4.5 Experiments

We evaluate our proposed object-centric model on several real-world robotic manipulation tasks. The experiments are chosen to evaluate two metrics: the reliability of this representation for robotic learning, and its ability to generalize across visual changes in the environment. The use of discrete region proposals should provide robustness against distractor objects. By attending over features trained on the diverse images found in ImageNet, we expect that policies learned with our visual representation will naturally generalize to visual changes. Through this evaluation, we demonstrate that the proposed model learns generalizable policies that behave correctly with new object instances and environments. Additionally, we show that the scope of generalization can be modified by showing different objects during demonstrations, which is particularly useful for correcting mistakes that the attention might make.

### 4.5.1 Training Details

The task-independent attention is provided by a region proposal network (RPN) [116] trained on the MSCOCO dataset [87]. For the semantic component of each object, we use conv5 of AlexNet [76], resulting in a 256-dimensional feature vector which is then normalized to have magnitude 1. Videos of the results can be found at <https://sites.google.com/berkeley.edu/object-representations>.

The attention vector  $w$  is learned by training a model on trajectory data as described in Section 4.3.2. The attended regions learned for both tasks are shown in Figure 4.4. To learn to perform the task, we use the guided policy search algorithm [81], which involves training local time-varying linear-Gaussian controllers for a number of different initial conditions. Then supervised learning is used to learn a single global neural network policy that can perform the task for all of the conditions. In our experiments, the conditions refer to different starting positions of objects in the world. The neural network policy takes as input the joint angles, joint velocities, end-effector positions and velocities, as well as the output of the perception system  $\nu$ , which corresponds to the attended region’s bounding box coordinates. The learned policies have 4 hidden layers with 80 hidden units each, and directly output the torques for the 7 joints of the PR2 robot used in our experiments. Note that our representation can be used with any reinforcement learning algorithm, including both deep reinforcement learning methods (such as the one used in our experiments) and trajectory-centric reinforcement learning algorithms such as PI2 [141] or REPS [109].

### 4.5.2 Generalizing Across Visual Changes

In this experiment, we evaluate the hypothesis that attending over high-level features trained on classification will generalize across objects of the same class. The goal of this task is to position a bottle to pour into a mug. Success requires the ability to locate a mug from an image and the global policy is not given the mug location, and we use different mugs for training and test. We compare against a task-specific approach from Levine et al., which learns the policy directly from raw pixels with a spatial softmax architecture [82]. While optimizing perception directly for the task objective performs well on particular mug seen during training, our method can generalize to new mug instances and to cluttered environments. Although the method in Levine et al. pretrains the first convolutional layer on ImageNet, conv1 features are too low-level to provide semantic generalization.

For evaluation, the policy is run with almonds in the bottle. A rollout is marked as successful if more almonds fall into the mug than are spilled, as seen in the included video. For evaluation, eight rollouts at different mug positions are performed for the uncluttered environments and three for the cluttered ones; results are in Figure 4.5 and environment photos are in Figure 4.6.

While the policy was only trained on a single brown mug in a plain environment, it successfully generalizes to other mugs of various colors. By using hard attention, the visual features are robust to clutter. Interestingly, when presented with all four mugs, the policy

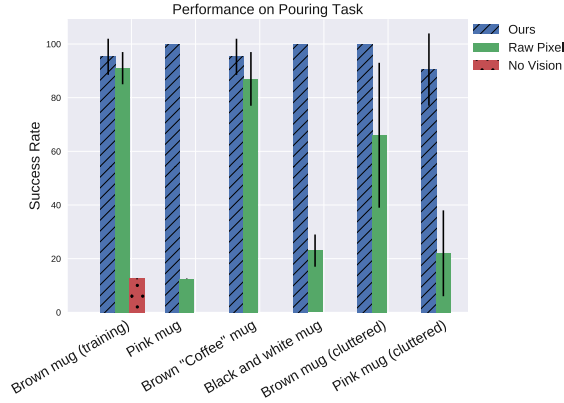


Figure 4.5: Results for the pouring task testing on different mugs not seen during training. Each group of bars shows performance on different unseen mugs, comparing our method with a policy trained from raw pixels. Our policy successfully generalizes to a much wider range of mugs than the raw pixel policy, which does not benefit from the object-centric attention prior. The “no vision” baseline indicates the performance of always pouring at the average mug position.

chose to pour into the pink mug rather than the brown mug the attention was trained with. The “no vision” baseline is a policy trained without visual features; its behavior is to pour to the average of the different targets seen during training. The low performance of this baseline indicates that the task requires a high level of precision. We compare to the method described in [82], where policies are learned directly from raw pixels and pretrained on a labeled data for detecting the target object.

Our model is able to generalize to new mugs of different appearances because it uses deep classification features that were trained on a wide variety of objects including mugs. An approach that learns robot skills directly from pixels such as [84] could not be expected to know that the brown mug and the pink mug are similar objects. We investigate this by training a policy from raw pixels with the architecture described in [84]. The convolutional layers are pretrained on detecting the training mug in 2000 labeled images. As shown in Table 4.5, this policy can perform the task on the training mug and on another brown mug, but completely ignores the other mugs. This indicates that a policy learned from raw pixel images can perform well on the training environments, the kinds of features it pays attention to have no incentive to be semantically meaningful and general. Our method of using features pretrained on image classification defines how a policy should generalize.

### 4.5.3 Learning to Ignore Distractors

In the first experiment, generalizing across mugs was a desired outcome. However, it is easy to imagine that a task might require pouring specifically into the brown mug and ignoring



Figure 4.6: Left: Mugs used for evaluation. Note that only the brown mug was seen during training. Center: Successful pouring into the pink mug. Right: Pouring into the brown mug in a cluttered environment that was not seen during training.

all other mugs. Our method provides a simple way for the user to adjust which features the task-specific attention is sensitive to. In order to learn a task-specific attention that has a narrower scope, the user can simply add another mug – in this case, a pink mug – as a distractor during the demonstrations. As described in Section 4.3.2, the vector  $W$  is trained such that the attended box allows predicting the arm’s movement in the demo trajectories. As the pink mug is not predictive of the trajectories, the gradient pushes the attention vector to lock on to brown mug specifically. We used 6 additional demonstrations to finetune the attention.

At test time, the pouring policy consistently chooses the brown training mug over both the pink mug and the black-and-white mug. This indicates that including distractors in the demonstrations helps restrict the scope of attention to ignore these distractors. Figure 4.7a shows how an attention initialized just on the brown mug is distracted by the distractor mug. After finetuning on the demonstrations, the attention is firmly on the brown mug. In experiments, the robot poured into the correct mug 100% of the time with either the pink mug or the black and white mug present as distractors. In comparison, the attention trained solely on demonstrations without distractors preferred the pink mug over the brown mug and obtained 50% success. This experiment shows that if a roboticist were to find that the attention vector is over-generalizing to distracting objects, it is easy for them to gather a couple more demonstrations to narrow down the attention.

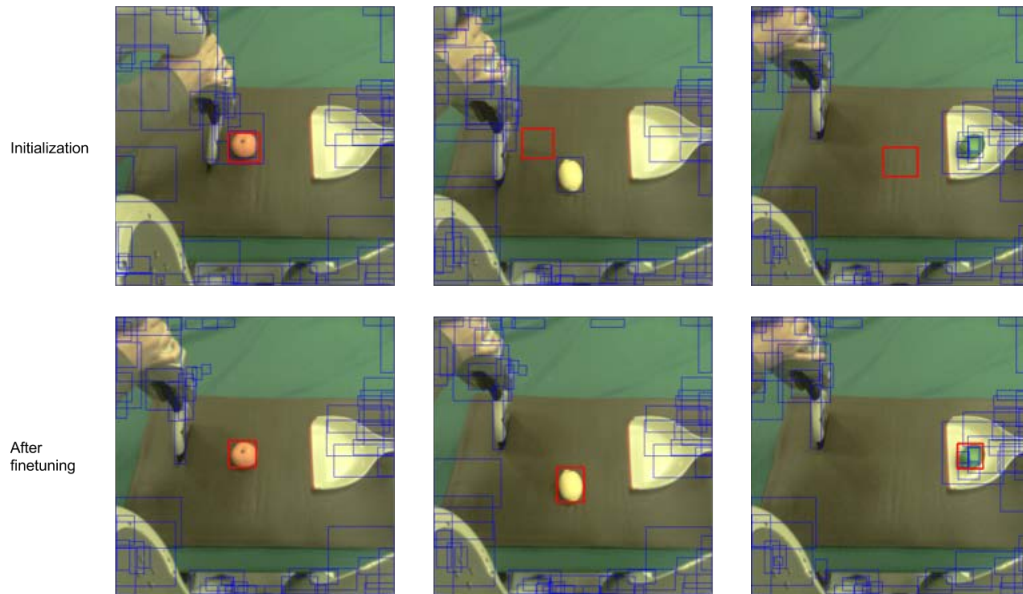
#### 4.5.4 Increasing the Scope of Generalization

Since our method is not limited by the labels present in available datasets, the attention vectors can also be pushed to attend to a greater variety objects. For example, a vector that attends to oranges may not always generalize to other citrus fruit. However, if generalizing across citrus is desired, the user can easily correct this mistake by adding a couple demonstrations with limes and lemons and finetuning  $W$ . In comparison, if a researcher relying on an off-the-shelf detector were to disagree with the detector’s performance, modifying the model could require relabeling data the model was trained on or collecting and labeling new detection





(a) Left: The soft attention from just training on the brown mug is shown in red. Right: The soft attention after finetuning on demonstrations where the pink mug is present. When initialized on only the brown mug, the attention is sensitive to "mug" features, and therefore can be distracted by the pink mug. After adding demonstrations of pouring into the brown mug with the pink mug in the background and finetuning, the attention has locked on to just the brown mug. The blue squares show the task-independent attention.



(b) Top: The soft attention in red from just training on the orange. Bottom: The soft attention after finetuning on demonstrations of sweeping oranges, lemons, and limes. Although the attention was initially sensitive to "orange-specific" features, finetuning on other fruit made the attention generalize to lemons and limes. The blue squares show the task-independent attention.

Figure 4.7

data.

As shown in Figure 4.7b, the attention only attends to oranges when first initialized, but finetuning expands the scope of the attention to include the lime and lemon present in the demonstration data. The resulting sweeping policy is robust to distractors including an apple, apricot, and a purple cup, but is confused by the orange and green cups. The round base of the citrus-colored cups perhaps appear to be idealized fruit.

### 4.5.5 Attending to Multiple Objects

In this experiment we demonstrate that we can use multiple attention vectors to learn a policy that depend on the location of two objects. The robot learns to perform a sweeping task where the object to be swept (a plastic orange) and the dustpan each can start in different positions. Ten kinesthetic demonstrations were collected to learn a pair of attention vectors, initialized with a single crop of the objects. This policy successfully sweeps the orange into the dustpan for 77% of the trials. The policy is robust to distractors and works even if the dustpan is held by a person. As shown in the video, this task is difficult because the robot’s angle of approach needs to be a function of the relative positions of the orange and dustpan, and the orange changes in appearance as it rolls. A baseline policy which did not use images at all succeeded at 11% of the test positions indicating that visual perception is necessary for this task.

## 4.6 Discussion

In this chapter, we proposed a visual representation for robotic skill learning that makes use of object-centric priors from pretrained visual models to achieve robust perception for policies trained in just a single scene with a single object. Our approach uses region proposal networks as a task-independent attention that picks out potential objects in the scene independently of the task, and then rapidly selects a task-specific representation via an attentional mechanism based on visual features, which can be trained from a few trajectories. Since the visual features used to index into the object proposals are themselves invariant to differences in lighting, appearance, viewpoint, and object instance, the resulting vision system can generalize effectively to new object instances with trivial additional training. The attention’s scope is easily controlled able by the objects seen during demonstrations. Our results indicate that this provides for a robust, generalizable, and customizable visual representation for sensorimotor skills. This representation generalize across different mugs when trained on only one mug, but could also be instance-specific if shown a handful of additional trajectories. In the opposite case, we show that an attention that was narrower than desired could be broadened as needed. Finally, for tasks that require interacting with multiple objects we can learn multiple attention vectors that and sensitive to different objects.

While our method attains good results on two real-world manipulation tasks, it has a number of limitations. First, the visual representation that is provided to the policy is

constrained to correspond to image-space object coordinates. Although this is sufficient for many manipulation tasks, some tasks, such as those that require fine-grained understanding of the configuration of articulated or deformable objects, might require a more detailed representation. Secondly, our current system is still trained in a stagewise manner, with the region proposals trained on prior vision data, the attention trained from demonstration, and the policy trained from experience. An exciting direction for future work would be to enable end-to-end finetuning of the entire system, which would lift many of these limitations. Since each stage in the current method is trained with simple and scalable gradient descent methods, end-to-end training should be entirely feasible, and should improve the performance of the resulting policies on tasks that require more subtle perception mechanisms.

Other work has continued this direction. In autonomous driving, drivign behavior has been shown to improve when supervising the perception layers with object detection [147] or segmentation [104]. Discrete object attention slots have also been used for scene representations [90] and affordance learning [49].

## Chapter 5

# Grasp2Vec: Learning Object Representations from Self-Supervised Grasping

### 5.1 Introduction

The method in the previous chapter relied on pre-trained ImageNet features for learning to attend to task relevant objects. However, these features offer no guarantees at representing novel objects, or at being consistent multiple image scales. In this chapter we develop a self-supervised approach for learning object features just by picking and placing objects randomly. This enables a robot to learn based on the objects it actually sees, rather than ones pre-collected in a dataset, without requiring humans to provide any labels.

We study a specific instance of the problem of self-supervised representation learning : acquiring object-centric representations through autonomous robotic interaction with the environment. By interacting with the real world, an agent can learn about the interplay of perception and action. For example, looking at and picking up objects enables a robot to discover relationships between physical entities and their surrounding contexts. If a robot grasps something in its environment and lifts it out of the way, then it could conclude that anything still visible was not part of what it grasped. It can also look at its gripper and see the object from a new angle. Through active interaction, a robot could learn which pixels in an image are graspable objects and recognize particular objects across different poses without any human supervision.

While object-centric representations can be learned from semantically annotated data (e.g., the MSCOCO dataset [88]), this precludes continuous self-improvement: additional experience that the robot collects, which lacks human annotations, is not directly used to improve the quality and robustness of the representation. In order to improve automatically, the representation must be self-supervised. In that regime, every interaction that the robot carries out with the world improves its representation.

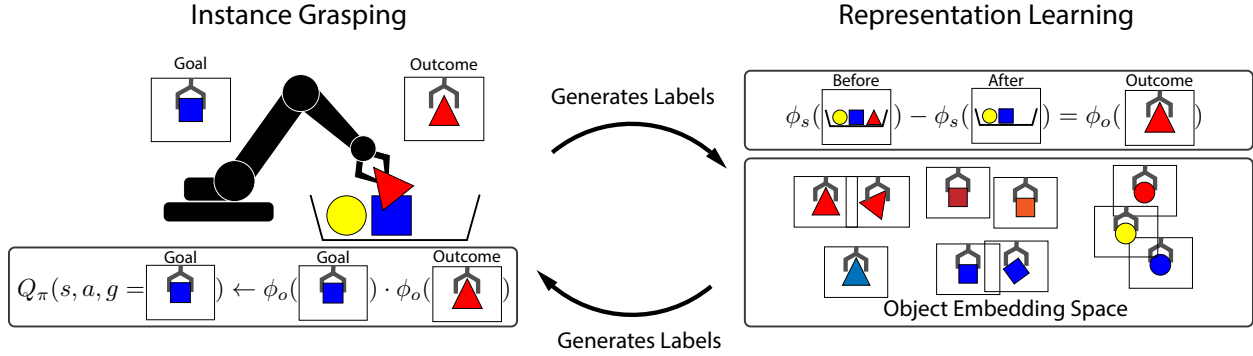


Figure 5.1: Instance grasping and representation learning processes generate each other’s labels in a fully self-supervised manner. **Representation learning from grasping:** A robot arm removes an object from the scene, and observes the resulting scene and the object in the gripper. We enforce that the difference of scene embeddings matches the object embedding. **Supervising grasping with learned representations:** We use a similarity metric between object embeddings as a reward for instance grasping, removing the need to manually label grasp outcomes.

Our representation learning method is based on object persistence: when a robot picks up an object and removes it from the scene, the representation of the scene should change in a predictable way. We can use this observation to formulate a simple condition that an object-centric representation should satisfy: the features corresponding to a scene should be approximately equal to the feature values for the same scene after an object has been removed, minus the feature value for that object (see Figure 5.1). We train a convolutional neural network feature extractor based on this condition, and show that it can effectively capture individual object identity and encode sets of objects in a scene without any human supervision.

Leveraging this representation, we propose learning a self-supervised grasping policy conditioned on an object feature vector or image. While labeling whether the correct object was grasped would typically require human supervision, we show that the similarity between object embeddings (learned with our method) provides an equally good reward signal.

Our main contribution is grasp2vec, an object-centric visual embedding learned with self-supervision. We demonstrate how this representation can be used for object localization, instance detection, and goal-conditioned grasping, where autonomously collected grasping experience can be relabeled with grasping goals based on our representation and used to train a policy to grasp user-specified objects. We find our method outperforms alternative unsupervised methods in a simulated goal-conditioned grasping results benchmark. Supplementary illustrations and videos are at <https://sites.google.com/site/grasp2vec/gma>

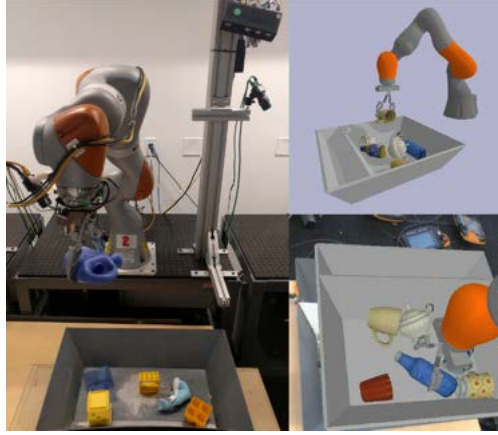


Figure 5.2: Self-supervised robotic setup for learning grasp2vec and goal-conditioned grasping. Left: A KUKA iiwa uses a monocular RGB camera to pick up objects from a bin. Right: The same setup in simulation. Bottom right: Example images that are fed into the  $Q$  function shown in Figure 5.3.

## 5.2 Relation to Prior Work

Addressing the task of instance grasping requires inferring whether the correct object was grasped. Jang et al. propose a system where the robot presents images of objects to the camera after it has grasped them and attempts to label their class given human labels. [62] Fang et al. obtain labels in simulation and use domain adaptation to generalize the policy to real-world scenes, which requires solving a simulation-to-reality transfer problem [36].

In the standard RL setting, several past works have studied labeling off-policy behavior with “unintentional rewards” [6, 10]. However, such algorithms do not address how to detect whether the desired goal was achieved, which is non-trivial outside of the simulator. Our methods circumvent the problem of labeling entirely via self-supervised representation learning. To our knowledge, this is the first work that learns the instance grasping task in a completely label-free manner.

## 5.3 Grasp2Vec: Representation Learning from Grasping

Our goal is to learn an object-centric embedding of images. The embeddings should represent objects via feature vectors, such that images with the same objects are close together, and those with different objects are far apart. Because labels indicating which objects are in an image are not available, we rely on a self-supervised objective. Specifically, we make use of the fact that, when a robot interacts with a scene to grasp an object, this interaction

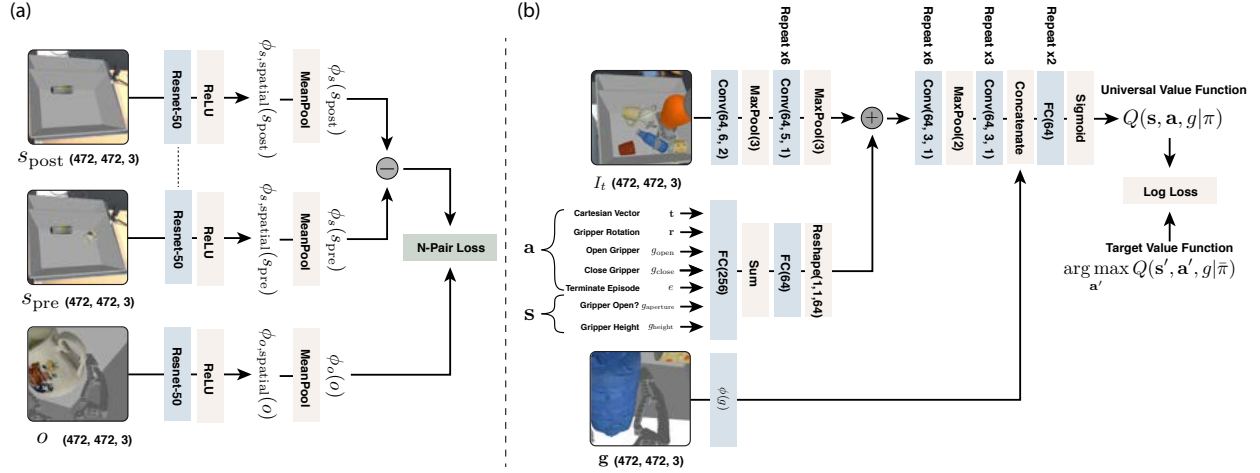


Figure 5.3: (a) The Grasp2Vec architecture. We use the first 3 blocks of Resnet-50 V2 to form  $\phi_s$  and  $\phi_o$ , which are randomly initialized.  $\phi_s(s_{\text{pre}})$  and  $\phi_s(s_{\text{post}})$  have tied weights. The output of the third resnet block is passed through a ReLU to yield a spatial feature map we call  $\phi_{\text{spatial}}$ . This is then mean pooled globally to output the single vector embeddings  $\phi_s$  and  $\phi_o$ . The n-pairs loss is applied as described in Section 5.3. (b) The instance grasping architecture is conditioned on Grasp2Vec goal embeddings of goal images, and is trained via Q-learning.

is quantized: it either picks up one or more whole objects, or nothing. When an object is picked up, we learn that the initial scene must have contained the object, and that the scene after must contain one fewer of that object. We use this concept to structure image embeddings by asking that feature difference of the scene before and after grasping is close to the representation of the grasped object.

We record grasping episodes as images triples:  $(s_{\text{pre}}, s_{\text{post}}, \mathbf{o})$ , where  $s_{\text{pre}}$  is an image of the scene before grasping,  $s_{\text{post}}$  is the same scene after grasping, and  $\mathbf{o}$  is an image of the grasped object held up to the camera. The specific grasping setup that we use, for both simulated and real image experiments, is described in Section 5.5. Let  $\phi_s(s_{\text{pre}})$  be a vector embedding of the input scene image (i.e., a picture of a bin that the robot might be grasping from). Let  $\phi_o(\mathbf{o})$  be a vector embedding of the outcome image, such that  $\phi_s(s_{\text{pre}})$  and  $\phi_o(\mathbf{o})$  are the same dimensionality. We can express the logic in the previous paragraph as an arithmetic constraint on these vectors: we would like  $(\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}}))$  to be equal to  $\phi_o(\mathbf{o})$ . We also would like the embedding to be non-trivial, such that  $(\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}}))$  is far from the embeddings of other objects that were not grasped.

**Architecture.** In order to embed images of scenes and outcomes, we employ convolutional neural networks to represent both  $\phi_s$  and  $\phi_o$ . The two networks are based on the ResNet-50 [52] architecture followed by a ReLU (see Figure 5.3), and both produce 3D ( $H \times W \times 1024$ ) convolutional feature maps  $\phi_{s,\text{spatial}}$  and  $\phi_{g,\text{spatial}}$ . Since our goal is to obtain a single representation of objects and sets of objects, we convert these maps into feature vectors by

globally average-pooling:

$$\phi_s = \frac{\sum_{i < H} \sum_{j < W} \phi_{s, \text{spatial}}(X)[i][j]}{H * W}$$

and equivalently for  $\phi_o$ . The motivation for this architecture is that it allows  $\phi_{s, \text{spatial}}$  to encode object position by which receptive fields produce which features. By applying a ReLU non-linearity to the spatial feature vectors, we constrain object representations to be non-negative. This ensures that a set of objects can only grow as more objects are added; one object cannot be the inverse of another.

**Objective.** The problem is formalized as metric learning, where the desired metric places  $(\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}}))$  close to  $\phi_o(\mathbf{o})$  and far from other embeddings. Many metric learning losses use the concept of an “anchor” embedding and a “positive” embedding, where the positive is brought closer to the anchor and farther from the other “negative” embeddings. One way to optimize this kind of objective is to use the n-pairs loss [130] to train the encoders  $\phi_s$  and  $\phi_o$ , such that paired examples (i.e.,  $(\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}}))$  and  $\phi_o(\mathbf{o})$ ) are pushed together, and unpaired examples are pushed apart. Rather than processing explicit (anchor, positive, negative) triplets, the n-pairs loss treats all *other* positives in a minibatch as negatives for an (anchor, positive) pair. Let  $i$  index into the anchors  $a$  of a minibatch and let  $j$  index into the positives  $p$ . The objective is to maximize  $a_i \top p_i$  while minimizing  $a_i \top p_{j \neq i}$ . The loss is the sum of softmax cross-entropy losses for each anchor  $i$  accross all positives  $p$ .

$$\text{NPairs}(a, p) = \sum_{i < B} -\log \left( \frac{e^{a_i \top p_i}}{\sum_{j < B} e^{a_i \top p_j}} \right) + \lambda(\|a_i\|_2^2 + \|p_i\|_2^2).$$

The hyperparameter  $\lambda$  regularizes the embedding magnitudes and  $B$  is the batch size. In our experiments,  $\lambda = 0.0005$  and  $B = 16$ . This loss is asymmetric for anchors and positives, so we evaluate with the embeddings in both orders, such that our final training objective is:

$$\mathcal{L}_{\text{Grasp2Vec}} = \text{NPairs}((\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}})), \phi_o(\mathbf{o})) + \text{NPairs}(\phi_o(\mathbf{o}), (\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}}))).$$

## 5.4 Self-Supervised Goal-Conditioned Grasping

The Grasp2Vec representation can enable effective goal-conditioned grasping, where a robot must grasp an object matching a user-provided query. In this setup, the same grasping system can both collect data for training the representation and utilize this representation for fulfilling specified goals. The grasping task is formulated as a Markov decision process (MDP), similar to the indiscriminate grasping system proposed by [67]. The actions  $\mathbf{a}$  correspond to Cartesian gripper motion and gripper opening and closing commands, and the state  $\lambda$  includes the current image and a representation of the goal  $\mathbf{g}$ . We aim to learn the function  $Q_\pi(\lambda, \mathbf{a}, \mathbf{g})$  under the following reward function: grasping the object specified by  $\mathbf{g}$  yields a



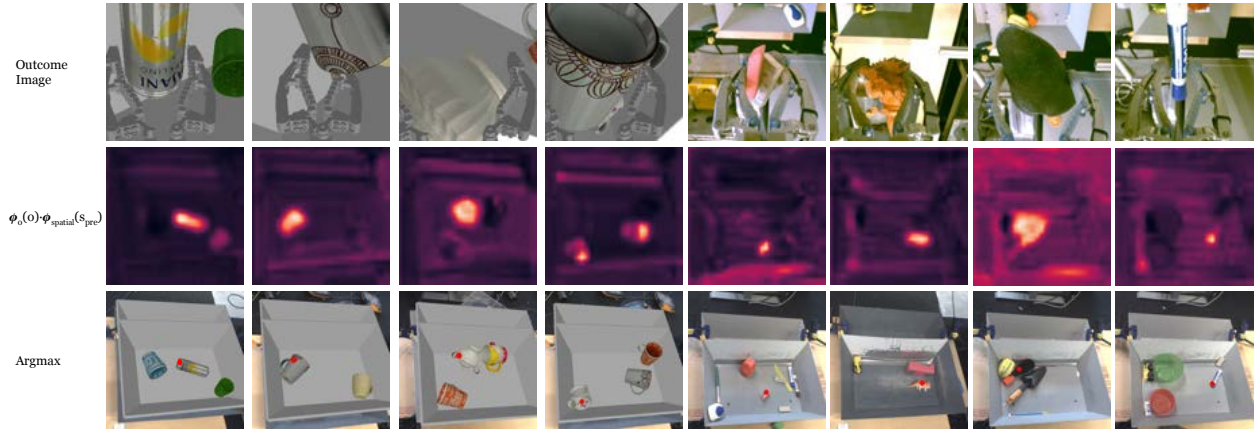
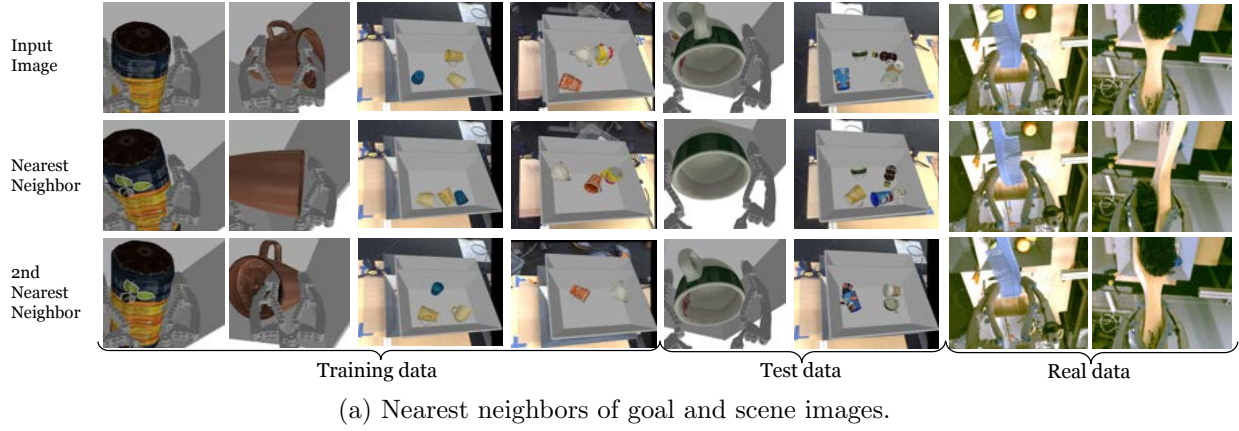


Figure 5.4: An analysis of our learned embeddings. Examples shown were chosen at random from the dataset.

terminal reward of  $\mathbf{r} = 1$ , and  $\mathbf{r} = 0$  for all other time steps. The architecture of  $Q_\pi$  is shown in Figure 5.3.

Learning this Q-function presents two challenges that are unique to self-supervised goal-conditioned grasping: we must find a way to train the policy when, in the early stages of learning, it is extremely unlikely to pick up the right object, and we must also extract the reward from the episodes without ground truth object labels.

---

**Algorithm 2** Goal-conditioned policy learning

---

```

1: Initialize goal set  $G$  with 10 present images
2: Initialize  $Q_\pi$  and replay buffer  $B$ 
3: while  $\pi$  not converged do
4:    $g, g' \leftarrow \text{sample}(G)$ 
5:    $(\backslash, \mathbf{a}, \mathbf{r}_{\text{any}}, \mathbf{o}) \leftarrow \text{ExecuteGrasp}(\pi, \mathbf{g})$ 
6:   // Posthoc Labeling (PL)
7:   if  $\mathbf{r}_{\text{any}} = 1$  then
8:      $B \leftarrow B \cup (\backslash, \mathbf{a}, [\mathbf{o}, 1])$  (outcome is successful goal)
9:   else
10:     $B \leftarrow B \cup (\backslash, \mathbf{a}, [\mathbf{g}, 0])$  (desired goal is a failure)
11:   end if
12:   // Embedding Similarity (ES)
13:   if  $\mathbf{r}_{\text{any}} = 1$  then
14:      $B \leftarrow B \cup (\backslash, \mathbf{a}, [\mathbf{g}, \hat{\phi}_o(\mathbf{o}) \cdot \hat{\phi}_o(\mathbf{g})])$ 
15:     // Auxiliary Goal (AUX)
16:      $B \leftarrow B \cup (\backslash, \mathbf{a}, [\mathbf{g}', \hat{\phi}_o(\mathbf{o}) \cdot \hat{\phi}_o(\mathbf{g}')])$ 
17:   end if
18:    $(\backslash, \mathbf{a}, [\mathbf{g}, \mathbf{r}], \mathbf{s}') \leftarrow \text{sample}(B)$ 
19:    $\pi \leftarrow \pi - \alpha \nabla_\pi (Q_\pi(\backslash, \mathbf{a}, \mathbf{g}) - (\mathbf{r} + \gamma V_\pi(\mathbf{s}', \mathbf{g})))^2$ 
20: end while
```

---

We assume that the grasping system can determine automatically whether it successfully grasped an object, but not which object was grasped. For example, the robot could check whether the gripper is fully closed to determine if it is holding something. We will use  $\mathbf{r}_{\text{any}}$  to denote the indiscriminate reward function, which is 1 at the last time step if an object was grasped, and 0 otherwise. Q-learning can learn from any valid tuple of the form  $(\backslash, \mathbf{a}, \mathbf{r}, \mathbf{g})$ , so we use the  $\mathbf{r}_{\text{any}}$  to generate these tuples without object labels. We utilize three distinct techniques to automatically augment the training data for Q-learning, making it practical to learn goal-conditioned grasping:

**Embedding Similarity (ES)** A general goal labeling system would label rewards based on a notion of similarity between what was commanded,  $\mathbf{g}$ , and what was achieved,  $\mathbf{o}$ , approximating the true on-policy reward function. If the Grasp2Vec representations capture this similarity between objects, setting  $\mathbf{r} = \hat{\phi}_o(\mathbf{g}) \cdot \hat{\phi}_o(\mathbf{o})$  would enable policy learning for instance grasping.

**Posthoc Labeling (PL)** Embedding similarity will give close to correct rewards to the Q function, but if the policy never grasps the right object there will be no signal to learn from. We use a data augmentation approach similar to the hindsight experience replay technique proposed by [6]. If an episode grasps any object, we can treat  $\mathbf{o}$  as a correct goal for that episode’s states and actions, and add the transition  $(\backslash, \mathbf{a}, \mathbf{o}, \mathbf{r} = 1)$  to the buffer. We refer to this as the posthoc label.

**Auxiliary Goal Augmentation (Aux)** We can augment the replay buffer even further by relabelling transitions with unacheived goals. Instead of sampling a single goal, we sample a pair of goals  $(\mathbf{g}, \mathbf{g}')$  from the goal set  $G$  without replacement. If  $\mathbf{r}_{\text{any}} == 1$  after executing the policy conditioned on  $\mathbf{g}$ , we add the transition  $(\backslash, \mathbf{a}, \mathbf{g}', \mathbf{r} = \hat{\phi}_o(\mathbf{g}') \cdot \hat{\phi}_o(\mathbf{o}))$  to the replay buffer. In baselines that do not use embeddings, the reward is replaced with 0 under the assumption that  $\mathbf{g}'$  is unlikely to be the grasped object.

Pseudocode for the goal-reward relabeling schemes, along with the self-supervised instance grasping routine, are summarized in Algorithm 2.

## 5.5 Experiments

Our experiments answer the following questions. For any grasp triplet  $(s_{\text{pre}}, \mathbf{o}, s_{\text{post}})$ , does the vector  $\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}})$  indicate which object was grasped? Can  $\phi_{s, \text{spatial}}(s_{\text{pre}})$  be used to localize objects in a scene? Finally, we show that instance grasping policies can be trained by using distance between grasp2vec embeddings as the reward function.

<https://sites.google.com/site/grasp2vec/>

**Experimental setup and data collection.** Real-world data collection for evaluating the representation on real images was conducted using KUKA LBR iiwa robots with 2-finger grippers, grasping objects with various visual attributes and varying sizes from a bin. Monocular RGB observations are provided by an over-the-shoulder camera. Actions  $\mathbf{a}$  are parameterized by a Cartesian displacement vector, vertical wrist rotation, and binary commands to open and close the gripper. The simulated environment is a model of the real environment simulated with Bullet [21]. Figure 5.2 depicts the simulated and real-world setup, along with the image observations from the robot’s camera. We train and evaluate grasp2vec embeddings on both the real and simulated environments across 3 tasks: object recall from scene differences, object localization within a scene, and use as a reward function for instance grasping.

### 5.5.1 Grasp2Vec embedding analysis.

We train the goal and scene embeddings on successful grasps. We train on 15k successful grasps for the simulated results and 437k for the real world results. The objective pushes embeddings of outcome images to be close to embedding difference of their respective scene

images, and far from each other. By representing scenes as the sum of their objects, we expect the scene embedding space to be structured by object presence and not by object location. This is validated by the nearest neighbors of scenes images shown in Figure 5.4a, where nearest neighbors contain the same same objects regardless of position or pose.

|                             | sim seen | sim novel | real seen | real novel |
|-----------------------------|----------|-----------|-----------|------------|
| Retrieval (ours)            | 88%      | 64%       | 89%       | 88%        |
| Outcome Neighbor (ImageNet) | —        | —         | 23%       | 22%        |
| Localization (ours)         | 96%      | 77%       | 83%       | 81%        |
| Localization (ImageNet)     | —        | —         | 18%       | 15%        |

Table 5.1: Quantitative study of Grasp2Vec embeddings. As we cannot expect weights trained on ImageNet to exhibit the retrieval property, we instead evaluate whether two nearest neighbor outcome images contain the same object. For object localization, the ImageNet baseline is performed the same as the grasp2vec evaluation. See Figure 5.4b for examples heatmaps and Appendix A.1 for example retrievals.

**For any grasp triplet  $(s_{\text{pre}}, \mathbf{o}, s_{\text{post}})$ , does the vector  $\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}})$  indicate which object was grasped?** We can evaluate the scene and and outcome feature spaces by verifying that the the difference in scene features  $\phi_s(s_{\text{pre}}) - \phi_s(s_{\text{post}})$  are near the grasped object features  $\phi_o(\mathbf{o})$ . As many outcome images of the same object will have similar features, we define the *retrieval* as correct if the nearest neighbor outcome image contains the same object as the one grasped from  $s_{\text{pre}}$ . Appendix A.1 shows example successes and failures. As shown in Table 5.1, retrieval accuracy is high for the training simulated objects and all real objects. Because the simulated data contained fewer unique objects that the real dataset, it is not surprising that the embeddings trained on real images generalized better.

**Can  $\phi_{s,\text{spatial}}(s_{\text{pre}})$  be used to localize objects in a scene?** The grasp2vec architecture and objective enables our method to localize objects without any spatial supervision.

By embedding scenes and single objects (outcomes) into the same space, we can use the outcome embeddings to localize that object within a scene. As shown in Figure 5.4b, we compute the dot product of  $\phi_o(o)$  with each pixel of  $\phi_{s,\text{spatial}}(s_{\text{pre}})$  to obtain a heatmap over the image corresponding to the affinity between the query object and each pixel’s receptive field. A localization is considered correct only if the point of maximum activation in the heatmap lies on the correct object. As shown in Table 5.1, grasp2vec embeddings perform localization at almost 80% accuracy on objects that were never seen during training, without ever receiving any position labels. The simulated objects seen during training are localized at even higher accuracy. We expect that such a method could be used to provide goals for pick and place or pushing task where a particular object position is desired. For this localization

evaluation, we compare grasp2vec embeddings against the same ResNet50-based architecture used in the embeddings, but trained on ImageNet [118]. This network is only able to localize the objects at 15% accuracy, because the features of an object in the gripper are not necessary similar to the features of that same object in the bin.

### 5.5.2 Simulated instance grasping

While past work has addressed self-supervised indiscriminate grasping, we show that instance grasping can be learned with no additional supervision. We perform ablation studies in simulation and analyze how choices in model architecture and goal-reward relabeling affect instance grasping performance and generalization to unseen test objects.

Overall instance grasping performance is reported in Table 5.2. All models using our method (Grasp2Vec embedding similarity) in the reward function achieve at least 78% instance grasping success on seen objects. Our experiments yield the following conclusions:

*How well can a policy learn without any measure of object similarity?* Looking at experiments 1 and 3, we see that posthoc labeling performs more than twice as well as indiscriminate grasping, while requiring no additional information. However, the PL only experiment is far behind the upper bound of using the true labels in experiment 3. Adding in auxiliary goal supervision in experiment 4, where data is augmented by randomly sampling a different goal image and marking it as a failed trajectory, only worsens the performance.

*Does any embedding of the goal image provide enough supervision?* The goal of the reward label is to indicate whether two goal images contain the same object, in order to reward the policy for grasping the correct object. We already found in Table 5.1 that ImageNet weights failed at this task. In experiment 5, we find that an autoencoder trained to encode goal images fails to provide a good reward label, performing no better than a indiscriminate policy.

*How close can grasp2vec embeddings get to the oracle performance?* The oracle labels requires knowing the true identity of the objects in the goal and outcome images. The cosine similarity of the goal and outcome image’s grasp2vec embeddings approximates this label much better than the autoencoder embeddings. Experiments 6 and 7 show that using grasp2vec similarity leads to performance on-par to the oracle on objects seen in training, and outperform the oracle on new objects that the policy was trained on. Unlike the oracle, grasp2vec similarity requires no object identity labels during either training or testing.

*Should the grasp2vec embedding also be used to condition the policy?* In experiment 8, we condition the policy on the embedding of the goal image instead of on the image itself. This reduces performance only on the unseen objects, indicating that the embeddings may hurt generalization by a small margin.

**Composite goals.** The additive compositionality of Grasp2Vec embeddings enables users to freely manipulate embeddings interactively to enable rich behavior at test time, without the policy ever having been explicitly trained to handle such goals. Our results show that policies conditioned on  $\phi_o$  can grasp one of two simultaneously commanded goal embeddings

| #        | Goal Conditioning       | Reward Labels             | Seen Objects | Unseen Objects |
|----------|-------------------------|---------------------------|--------------|----------------|
| 1        | Indiscriminate Grasping | N/A                       | 18.3         | 21.9           |
| 2        | Raw Image CNN           | Oracle Labels             | 83.9         | 43.7           |
| 3        | Raw Image CNN           | PL                        | 50.4         | 41.1           |
| 4        | Raw Image CNN           | PL + Aux(0)               | 22.1         | 19.0           |
| 5        | Raw Image CNN           | PL + ES (autoencoder)     | 18.7         | 20.9           |
| 6 (ours) | Raw Image CNN           | PL + ES (grasp2vec)       | <b>80.1</b>  | 53.9           |
| 7 (ours) | Raw Image CNN           | PL + Aux + ES (grasp2vec) | 78.0         | <b>58.9</b>    |
| 8 (ours) | $\phi_o(g)$             | PL + ES (grasp2vec)       | 78.4         | 45.4           |

Table 5.2: Evaluation and ablation studies on a simulated instance grasping task, averaged over 700 trials. In simulation, the scene graph is accessed to evaluate ground-truth performance, but it is withheld from our learning algorithms. Performance is reported as percentage of grasps that picked up the user-specified object. Table reports early stopping scores for instance grasping on training objects and evaluation for the same checkpoint on test objects. Best numbers (for unsupervised approaches) in bold font.

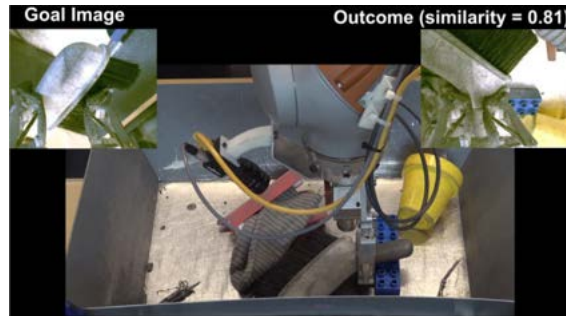


Figure 5.5: Instance grasping with grasp2vec outcome similarity. The goal image is shown on the left, and the center shows the robot during the trajectory. The final outcome image is on the right along with its grasp2vec similarity to the goal.

that are averaged together: in simulation the composite-goal conditioned policies obtains **51.9%** and **42.9%** success for seen and unseen objects, respectively. The policy, which was only trained on single goals, generalizes to composite goals due to the additive semantics of grasp2vec embeddings.

### 5.5.3 Real-world instance grasping.

To further evaluate the real-world grasp2vec embeddings, we design an instance grasping method that requires no additional on-policy training. We leverage the localization property of grasp2vec embeddings and use an indiscriminate grasping policy (for example, the one used to collect the original grasping data). To grasp a goal object pictured in  $\mathbf{g}$  from a scene  $s$ , we obtain the 2D localization coordinate  $(x, y) = \arg \max(\phi_o(\mathbf{g}) * \phi_{s, \text{spatial}}(s))$ . Using the known camera calibration, we convert this into a 3D coordinate in the base frame of the arm and move the end-effector to that position. From there, an indiscriminate grasping policy is executed, which grasps the object closest to the gripper. To reduce the likelihood of accidentally picking up the wrong object, we compute the grasp2vec cosine similarity of a wrist-mounted camera image with the goal image. If the robot has just grasped an object and similarity falls below a threshold of 0.7, we drop the object and re-localize the goal object. We run this policy for a maximum of 40 time steps. Using this method, we obtain **80.8%** and **62.9%** instance grasp success on training and test objects, respectively.

## 5.6 Discussion

We presented grasp2vec, a representation learning approach that learns to represent scenes as sets of objects, admitting basic manipulations such as removing and adding objects as arithmetic operations in the learned embedding space. Our method is supervised entirely with data that can be collected autonomously by a robot, and we demonstrate that the learned representation can be used to localize objects, recognize instances, and also supervise a goal-conditioned grasping method that can learn via goal relabeling to pick up user-commanded objects. Importantly, the same grasping system that collects data for our representation learning approach can also utilize it to become better at fulfilling grasping goals, resulting in an autonomous, self-improving visual representation learning method. Our work suggests a number of promising directions for future research: incorporating semantic information into the representation (e.g., object class), leveraging the learned representations for spatial, object-centric relational reasoning tasks (e.g., [64]), and further exploring the compositionality in the representation to enable planning compound skills in the embedding space.

Since the publication of this work, other papers have presented related self-supervised approaches for object picking [135, 29, 144], rope manipulation [134], surgical robotics [137], and general robot reinforcement learning [15].

## Chapter 6

# Plan Arithmetic: Compositional Plan Vectors for Multi-Task Control

### 6.1 Introduction

In this chapter we extend the idea of training representations to obey arithmetic properties to the problem of task composition. Instead of adding and subtracting objects, we train a policy conditioned on the addition and subtraction of embeddings of demonstrations. We show that CPVs can be learned within a one-shot imitation learning framework without any additional supervision or information about task hierarchy, and enable a demonstration-conditioned policy to generalize to tasks that sequence twice as many skills as the tasks seen during training.

Many tasks can be expressed as compositions of skills, where the same set of skills is shared across many tasks. For example, assembling a chair may require the subtask of picking up a hammer, which is also found in the table assembly task. We posit that a task representation that leverages this compositional structure can generalize more easily to more complex tasks. We propose learning an embedding space such that tasks could be composed simply by adding their respective embeddings. This idea is illustrated in Figure 6.2.

In order to learn these representations without additional supervision, we cannot depend on known segmentation of the trajectories into subtasks, or labels about which subtasks are shared between different tasks. Instead, we incorporate compositionality directly into the architecture of the policy. Rather than conditioning the policy on the static embedding of the reference demonstration, we condition the policy on the *difference* between the embedding of the whole reference trajectory and the partially completed trajectory that the policy is outputting an action for.

The main contributions of our work are the compositional plan vector (CPV) representation and a policy architecture that enables learning of CPVs without any sub-task level supervision. CPVs enable policies to generalize to significantly longer tasks, and they can be added together to represent a composition of tasks. We evaluate CPVs in the one-shot imitation learning



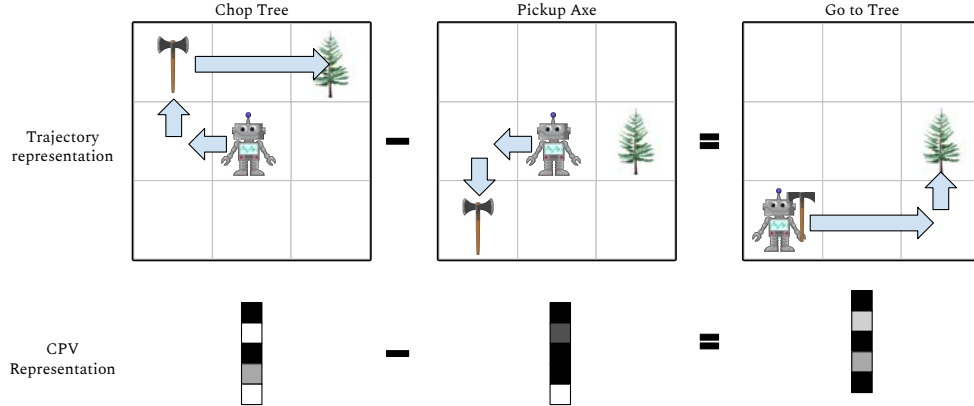


Figure 6.1: Compositional plan vectors embed tasks into a space where adding two vectors represents the composition of the tasks, and subtracting a sub-task leaves an embedding of the remaining sub-tasks needed for the task.

paradigm [33, 40, 61] on a discrete-action environment inspired by Minecraft, where tools must be picked up to remove or build objects, as well as on a 3D simulated pick-and-place environment.

## 6.2 Relation to Prior Work

In the one-shot imitation learning paradigm, the policy is conditioned on reference demonstrations at both test and train time. This problem has been explored with meta-learning [40] and metric learning [61] for short reaching and pushing tasks. Duan et al. used attention over the reference trajectory to perform block stacking tasks [33]. Our work differs in that we aim to generalize to new compositions of tasks that are out of the distribution of tasks seen during training. Hausman et al. obtain generalization to new compositions of skills by training a generative model over skills [50]. However, unlike our method, these approach does not easily allow for sequencing skills into longer horizon tasks or composing tasks via arithmetic operations on the latent representation.

Prior methods have learned composable task representations by using ground truth knowledge about the task hierarchy. Neural task programming and the neural subtask graph solver generalize to new tasks by decomposing a demonstration into a hierarchical program for the task, but require ground-truth hierarchical decomposition during training [149, 131]. Using supervision about the relations between tasks, prior approaches have uses analogy-based objectives to learn task representations that decompose across objects and actions [107] or have set up a modular architectures over subtasks [4] or environments [28]. Unlike our approach, these methods require labels about relationships. We implicitly learn to decompose tasks without supervising the task hierarchy.

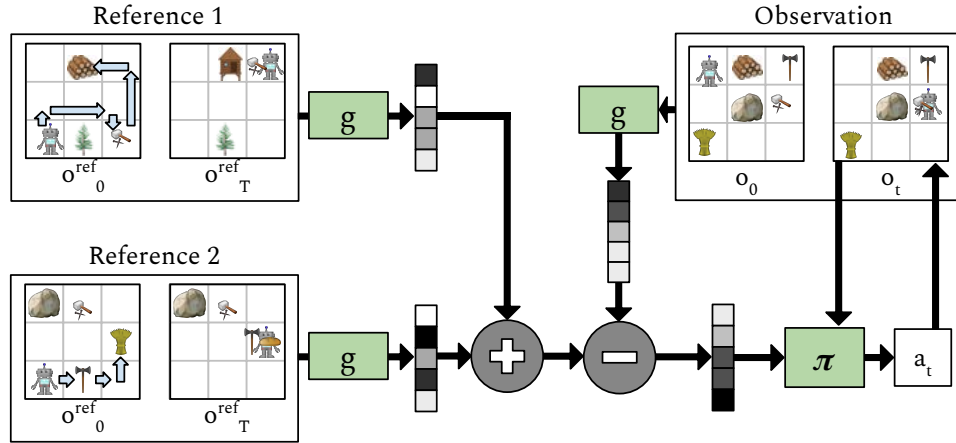


Figure 6.2: By adding the CPVs for two different tasks, we obtain the CPV for the composition of the tasks. To determine what steps are left in the task, the policy subtracts the embedding of its current trajectory from the reference CPV.

### 6.3 Compositional Plan Vectors

In this chapter, we introduce compositional plan vectors (CPVs). The goal of CPVs is to obtain policies that generalize to new compositions of skills without requiring skills to be labeled and without knowing the list of skills that may be required. Consider a task named “red-out-yellow-in” which involves taking a red cube out of a box and placing a yellow cube into the box. A plan vector encodes the task as the sum of its parts: a plan vector for taking the red cube out of the box plus the vector for putting the yellow cube into the box should equal the plan vector for the full task. Equivalently, the plan vector for the full task minus the vector for taking the red cube out of the box should equal the vector that encodes “put yellow cube in box.”

If the list of all possible skills was known ahead of time, separate policies could be learned for each skill, and then the policies could be used in sequence. However, this knowledge is often unavailable in general and limits compositionality to a fixed set of skills. Instead, our goal is to formulate an architecture and regularization that are compositional *by design* and do not need additional supervision. With our method, CPVs acquire compositional structure because of the structural constraint they place on the policy. To derive the simplest possible structural constraint, we observe that the minimum information that the policy needs about the task in order to complete it is *knowledge of the steps that have not yet been done*. That is, in the cube example above, after taking out the red cube, only the “yellow-in” portion of the task is needed by the policy. One property of this representation is that task ordering cannot be represented by the CPV because addition is commutative. If ordering is necessary

to choose the right action, the policy will have to learn to decode which component of the compositional plan vector must be done first.

As an example, let  $\vec{v}$  be a plan vector for the “red-out-yellow-in” task. To execute the task, a policy  $\pi(\mathbf{o}_0, \vec{v})$  outputs an action for the first observation  $\mathbf{o}_0$ . After some number  $t$  of timesteps, the policy has successfully removed the red cube from the box. This partial trajectory  $\mathbf{O}_{0:t}$  can be embedded into a plan vector  $\vec{u}$ , which should encode the “red-out” task. We would like  $(\vec{v} - \vec{u})$  to encode the remaining portion of the task, in this case placing the yellow block into the box. In other words,  $\pi(\mathbf{o}_t, \vec{v} - \vec{u})$  should take the action that leads to accomplishing the plan described by  $\vec{v}$  given that  $\vec{u}$  has already been accomplished. In order for both  $\vec{v}$  and  $\vec{v} - \vec{u}$  to encode the yellow-in task,  $\vec{u}$  must not encode as strongly as  $\vec{v}$ . If  $\vec{v}$  is equal to the sum of the vectors for “red-out” and “yellow-in,” then  $\vec{v}$  may not encode the ordering of the tasks. However, the policy  $\pi(\mathbf{o}_0, \vec{v})$  should have learned that the box must be empty in order to perform the yellow-in task, and that therefore it should perform the red-out task first.

We posit that this structure can be learned without supervision at the subtask-level. Instead, we impose a simple architectural and arithmetic constraints on the policy: the policy must choose its action based on the arithmetic difference between the plan vector embedding of the whole task and the plan vector embedding of the trajectory completed so far. Additionally, the plan vectors of two halves of the same trajectory should add up to the plan vector of the whole trajectory, which we can write down as a regularization objective for the embedding function. By training the policy and the embedding function together to optimize their objectives, we obtain an embedding of tasks that supports compositionality and generalizes to more complex tasks. In principle, CPVs can be used with any end-to-end policy learning objective, including behavioral cloning, reinforcement learning, or inverse reinforcement learning. In this work, we will validate CPVs in a one-shot imitation learning setting.

**One-shot imitation learning setup.** In one-shot imitation learning, the agent must perform a task conditioned on one reference example of the task. For example, given a demonstration of how to fold a paper crane, the agent would need to fold a paper crane. During training, the agent is provided with pairs of demonstrations, and learns a policy by predicting the actions in one trajectory by using the second as a reference. In the origami example, the agent may have trained on demonstrations of folding paper into a variety of different creatures.

We consider the one-shot imitation learning scenario where an agent is given a reference trajectory in the form of a list of  $T$  observations  $\mathbf{O}_{0:T}^{\text{ref}} = (\mathbf{o}_0^{\text{ref}}, \dots, \mathbf{o}_T^{\text{ref}})$ . The agent starts with  $\mathbf{o}_0 \sim p(\mathbf{o}_0)$ , where  $\mathbf{o}_0$  may be different from  $\mathbf{o}_0^{\text{ref}}$ . At each timestep  $t$ , the agent performs an action drawn from  $\pi(\mathbf{a}_t | \mathbf{O}_{0:t}, \mathbf{O}_{0:T}^{\text{ref}})$ .

**Plan vectors.** We define a function  $g_\phi(\mathbf{O}_{k:l})$ , parameterized by  $\phi$ , which takes in a trajectory and outputs a *plan vector*. The plan vector of a reference trajectory  $g_\phi(\mathbf{O}_{0:T}^{\text{ref}})$  should encode an abstract representation of the milestones required to accomplish the goal. Similarly, the plan

vector of a partially accomplished trajectory  $g_\phi(\mathbf{O}_{0:t})$  should encode the steps already taken. We can therefore consider the subtraction of these vectors to encode the steps necessary to complete the task defined by the reference trajectory. Thus, the policy can be structured as

$$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t, g_\phi(\mathbf{O}_{0:T}^{\text{ref}}) - g_\phi(\mathbf{O}_{0:t})), \quad (6.1)$$

a function parameterized by  $\theta$  that takes in the trajectory and is learned end-to-end.

In this work we use a fully observable state space and only consider tasks that cause a change in the state. For example, we do not consider tasks such as lifting a block and placing it exactly where it was, because this does not result in a useful change to the state. Thus, instead of embedding a whole trajectory  $\mathbf{O}_{0:t}$ , we limit  $g$  to only look at the first and last state of the trajectory we wish to embed. Then,  $\pi$  becomes

$$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t, g(\mathbf{o}_0^{\text{ref}}, \mathbf{o}_T^{\text{ref}}) - g(\mathbf{o}_0, \mathbf{o}_t)). \quad (6.2)$$

**Training.** With  $\pi$  defined as above, we learn the parameters of the policy with imitation learning. Dataset  $\mathcal{D}$  containing  $N$  demonstrations paired with reference trajectories is collected. Each trajectory may be a different arbitrary length, and the tasks performed by each pair of trajectories are unlabeled. The demonstrations include actions, but the reference trajectories do not. In our settings, the reference trajectories only need to include their first and last states. Formally,

$$\mathcal{D} = \{(\mathbf{O}_{[0,T^i]}^{\text{ref}}, \mathbf{O}_{[0,H^i]}^i, \mathbf{A}_{[0:H^i-1]}^i)\}_{i=1}^N,$$

where  $T^i$  is the length of the  $i^{\text{th}}$  reference trajectory and  $H^i$  is the length of the  $i^{\text{th}}$  demonstration. Given the policy architecture defined in Equation 6.1, the behavioral cloning loss for a discrete action policy is

$$\mathcal{L}_{\text{IL}}(\mathcal{D}, \theta, \phi) = \sum_{i=0}^N \sum_{t=0}^{H^i} -\log(\pi_\theta(\mathbf{a}_t^i | \mathbf{o}_t^i, g_\phi(\mathbf{o}_0^{\text{ref}^i}, \mathbf{o}_T^{\text{ref}^i}) - g_\phi(\mathbf{o}_0^i, \mathbf{o}_t^i))).$$

We also introduce a regularization loss function to improve compositionality by enforcing that the sum of the embeddings of two parts of a trajectory is close to the embedding of the full trajectory. We denote this a homomorphism loss  $\mathcal{L}_{\text{Hom}}$  because it constrains the embedding function  $g$  to preserve the structure between concatenation of trajectories and addition of real-valued vectors. We implement the loss using the triplet margin loss from [122] with a margin equal to 1:

$$l_{\text{tri}}(a, p, n) = \max\{\|a - p\|_2 - \|a - n\|_2 + 1.0, 0\}$$

$$\mathcal{L}_{\text{Hom}}(\mathcal{D}, \phi) = \sum_{i=0}^N \sum_{t=0}^{H^i} l_{\text{tri}}(g_\phi(\mathbf{o}_0^i, \mathbf{o}_t^i) + g_\phi(\mathbf{o}_t^i, \mathbf{o}_T^i), g_\phi(\mathbf{o}_0^i, \mathbf{o}_T^i), g_\phi(\mathbf{o}_0^i, \mathbf{o}_T^i))$$

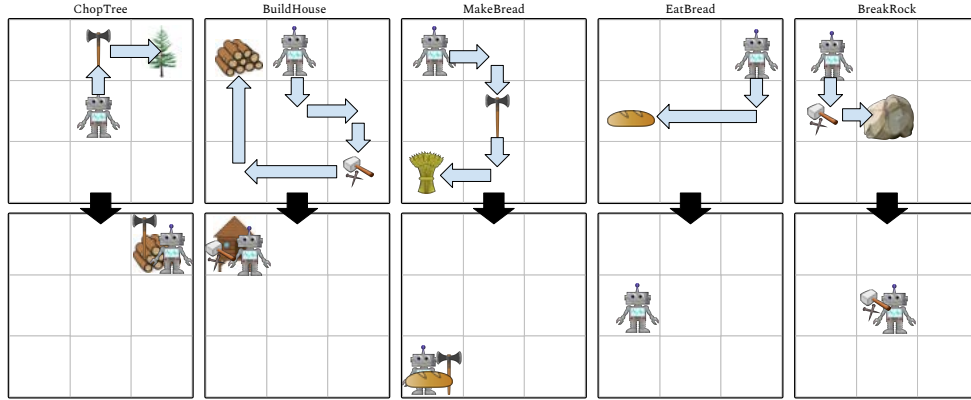


Figure 6.3: Illustrations of the 5 skills in the crafting environment. To ChopTree, the agent must pick up the axe and bring it to the tree, which transforms the tree into logs. To BuildHouse, the agent picks up the hammer and brings it to logs to transform them into a house. To MakeBread, the agent brings the axe to the wheat which transforms it into bread. The agent eats bread if it lands on a state that contains bread. To BreakRock, the agent picks up a hammer and destroys the rock.

Finally, we follow James et al. in regularizing embeddings of paired trajectories to be close in embedding space, which has been shown to improve performance on new examples [61]. This “pair” loss  $\mathcal{L}_{\text{Pair}}$  pushes the embedding of a demonstration to be similar to the embedding of its reference trajectory and different from other embeddings, which enforces that embeddings are a function of the behavior within a trajectory rather than the appearance of a state.

$$\mathcal{L}_{\text{Pair}}(\mathcal{D}, \phi) \sum_{i=0}^N \sum_{t=0}^{H^i} l_{\text{tri}}(g_{\phi}(\mathbf{o}_0^i, \mathbf{o}_T^i), g_{\phi}(\mathbf{o}_{0,T}^{\text{ref}^i}), g_{\phi}(\mathbf{o}_{0,T}^{\text{ref}^j}))$$

for any  $j \neq i$ . We empirically evaluate how these losses affect the composability of learned embeddings. While  $\mathcal{L}_{\text{Pair}}$  leverages the supervision from the reference trajectories,  $\mathcal{L}_{\text{Hom}}$  is entirely self-supervised.

**Measuring compositionality.** To evaluate whether the representation learned by  $g$  is compositional, we condition the policy on the sum of plan vectors from multiple tasks and measure the policy’s success rate. Given two reference trajectories  $\mathbf{O}_{0:T^i}^{\text{ref}^i}$  and  $\mathbf{O}_{0:T^j}^{\text{ref}^j}$ , we condition the policy on  $g_{\phi}(\mathbf{o}_{0,T^i}^{\text{ref}^i}) + g_{\phi}(\mathbf{o}_{0,T^j}^{\text{ref}^j})$ . The policy is successful if it accomplishes both tasks. We also evaluate whether the representation generalizes to more complex tasks.

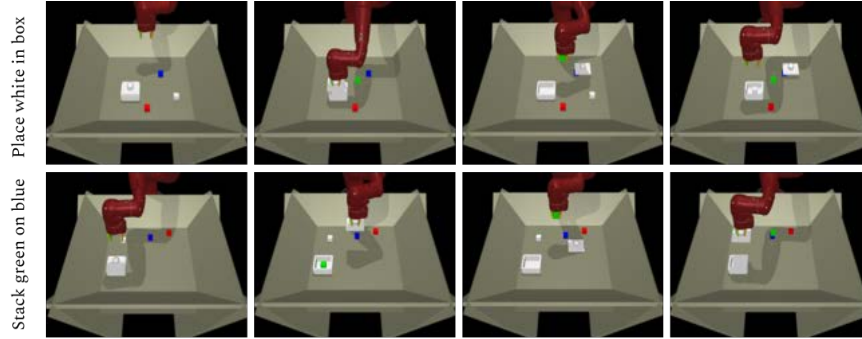


Figure 6.4: Two example skills from the pick and place environment. Time evolves from left to right. If the relevant objects are in the box, the agent must first remove the lid to interact with the object and also return the lid to the box in order to complete a task.

## 6.4 Sequential Multitask Environments

We introduce two new learning environments, shown in Figures 6.3 and 6.4, that test an agent’s ability to perform tasks that require different sequences and different numbers of sub-skills. We designed these environments such that the actions change the environment and make new sub-goals possible: in the 3D environment, opening a box and removing its contents makes it possible to put something else into the box. In the crafting environment, chopping down a tree makes it is possible to build a house. Along with the environments, we will release code to generate demonstrations of the compositional tasks.

### 6.4.1 Crafting Environment

The first evaluation domain is a discrete-action world where objects can be picked up and modified using tools. The environment contains 7 types of objects: tree, rock, logs, wheat, bread, hammer, axe. Logs, hammers, and axes can be picked up, and trees and rocks block the agent. The environment allows for 6 actions: up, down, left, right, pickup, and drop. The transitions are deterministic, and only one object can be held at a time. Pickup has no effect unless the agent is at the same position as a pickup-able object. Drop has no effect unless an object is currently held. When an object is held, it moves with the agent. Unlike the Malmö environment which runs a full game engine [65], this environment can be easily modified to add new object types and interaction rules. We define 5 skills within the environment, *ChopTree*, *BreakRock*, *BuildHouse*, *MakeBread*, and *EatBread*, as illustrated in Figure 6.3. A task is defined by a list of skills. For example, a task with 3 skills could be [*ChopTree*, *ChopTree*, *MakeBread*]. Thus, considering tasks that use between 1 and 4 skills with replacement, there are 125 distinct tasks and about 780 total orderings. Unlike in [4, 107], skill list labels are only used for data generation and evaluation; they are not used for training and are not provided to the model. The quantities and positions of each object

are randomly selected at each reset. The observation space is a top-down image view of the environment, as shown in Figure B.1a.

### 6.4.2 3D Pick and Place Environment

The second domain is a 3D simulated environment where a robot arm can pick up and drop objects. Four cubes of different colors, as well as a box with a lid, are randomly placed within the workspace. The robot’s action space is a continuous 4-dimensional vector: an  $(x, y)$  position at which to close the gripper and an  $(x, y)$  position at which to open the gripper. The  $z$  coordinate of the grasp is chosen automatically. The observation space is a concatenation of the  $(x, y, z)$  positions of each of the 4 objects, the box, and the box lid. We define 3 families of skills within the environment: *PlaceInCorner*, *Stack*, and *PlaceInBox*, each of which can be applied on different objects or pairs of objects. Considering tasks that use 1 to 2 skills, there are 420 different tasks. An example of each skill is shown in Figure 6.4.

## 6.5 Experiments

Our experiments aim to understand how well CPVs can learn tasks of varying complexity, how well they can generalize to tasks that are more complex than those seen during training (thus demonstrating compositionality), and how well they can handle additive composition of tasks, where the policy is expected to perform both of the tasks in sequence. We hypothesize that, by conditioning a policy on the subtraction of the current progress from the goal task embedding, we will learn a task representation that encodes tasks as the sum of their component subtasks. We additionally evaluate how regularizing objectives improve generalization and compositionality.

**Implementation.** We implement  $g_\phi$  and  $\pi_\theta$  as neural networks. For the crafting environment, where the observations are RGB images, we use the convolutional architecture in Figure 6.5. The encoder  $g$  outputs a 512 dimensional CPV. The policy, shaded in red, takes the subtraction

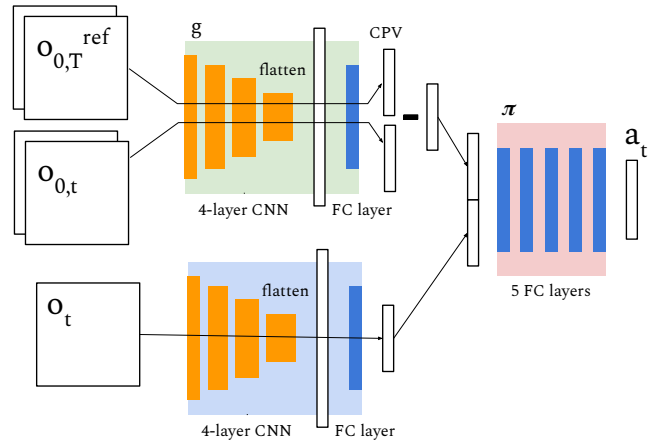


Figure 6.5: The network architecture used for the crafting environment. Orange denotes convolutional layers and dark blue denotes fully connected layers. The trajectories  $(\mathbf{o}_0^{\text{ref}}, \mathbf{o}_T^{\text{ref}})$  and  $(\mathbf{o}_0, \mathbf{o}_t)$  are each passed through  $g$  (the pale green box) independently, but with shared weights. The current observation  $\mathbf{o}_t$  is processed through a separate convolutional network before being concatenated with the vector  $g(\mathbf{o}_0^{\text{ref}}, \mathbf{o}_T^{\text{ref}}) - g(\mathbf{o}_0, \mathbf{o}_t)$ .

of CPVs concatenated with features from the current observation and outputs a discrete classification over actions.

For the 3D environment, the observation is a state vector containing the positions of each object in the scene, including the box and box lid. The function  $g$  again concatenates the inputs, but here the network is fully connected, and the current observation is directly concatenated to the subtraction of the CPVs. To improve the performance of all models and comparisons, we use an object-centric policy inspired by Chapter 4, where the policy outputs a softmaxed weighting over the objects in the state. The position of the most attended object is output as the first coordinates of the action (where to grasp). The object attention as well as the features from the observation and CPVs are passed to another fully connected layer to output the position for placing.

**Data generation.** For the crafting environment, we train all models on a dataset containing 40k pairs of demonstrations, each pair performs the same task. The demonstration pairs are not labeled with what task they are performing. The tasks are randomly generated by sampling 2-4 skills with replacement from the five skills listed previously. A planning algorithm is used to generate demonstration trajectories. For the 3D environment, we collect 180k trajectories of tasks with 1 and 2 skills. All models are trained on this dataset to predict actions from the environment observations shown in Figure B.1a. For both environments, we added 10% of noise to the planner’s actions but discarded any trajectories that were unsuccessful. The data is divided into training and validation sets 90/10. To evaluate the models, reference trajectories were either regenerated or pulled from the validation set. Compositions of trajectories were never used in training or validation.

**Comparisons.** We compare our CPV model to several one-shot imitation learning models. All models are based on Equation 6.2, where the policy is function of four images:  $\mathbf{o}_0, \mathbf{o}_t, \mathbf{o}_0^{\text{ref}}, \mathbf{o}_T^{\text{ref}}$ . The *naïve* baseline simply concatenates the four inputs as input to a neural network policy. The TECNets baseline is an implementation of task embedding control networks from [61], where the embeddings are normalized to a unit ball and a margin loss is applied over the cosine distance to push together embeddings of the same task. The policy in TECNets is conditioned on the static reference embedding rather than the subtraction of two embeddings. For both TECNets and our model,  $g$  is applied to the concatenation of the two input observations.

We perform several ablations of our model, which includes the CPV architecture (including the embedding subtraction as input the policy), the homomorphism regularization, and the pair regularization. We compare the plain version of our model, where the objective is purely imitation learning, to versions that use the regularizations. *CPV-Plain* uses no regularization, *CPV-Pair* uses only  $\mathcal{L}_{\text{Pair}}$ , *CPV-Hom* uses only  $\mathcal{L}_{\text{Hom}}$ , and *CPV-Full* uses both. To ablate the effect of the architecture vs the regularizations, we run the same set of comparisons for a model denoted TE (task embeddings) which has the same architecture as TECNets without normalizing embeddings. These experiments find whether the regularization losses



Table 6.1: **Evaluation of generalization and compositionality in the crafting environment.** Policies were trained on tasks using between 1 and 4 skills. We evaluate the policies conditioned on reference trajectories that use 4, 8, and 16 skills. We also evaluate the policies on the composition of skills: “2, 2” means that the embeddings of two demonstrations that each use 2 skills were added together, and the policy was conditioned on this sum. For the naïve model, we instead average the observations of the references, which performed somewhat better. All models are variants on the architecture in Figure 6.5. The max horizon is three times the average number of steps used by the expert for that length of task: 160, 280, and 550, respectively. Numbers are all success percentages.

| Model     | 4 Skills       | 8 Skills       | 16 Skills      | 1+1           | 2,2            | 4,4            |
|-----------|----------------|----------------|----------------|---------------|----------------|----------------|
| Naive     | 29 ± 2         | 9 ± 2          | 7 ± 2          | 29 ± 10       | 24 ± 5         | 5 ± 2          |
| TECNet    | 49 ± 11        | 17 ± 7         | 7 ± 6          | 59 ± 11       | 43 ± 11        | <b>29 ± 23</b> |
| TE        | 53 ± 4         | 28 ± 2         | <b>25 ± 20</b> | 32 ± 1        | <b>44 ± 25</b> | 18 ± 12        |
| TE-Pair   | 64 ± 1         | 31 ± 1         | 18 ± 2         | 55 ± 3        | 53 ± 8         | 21 ± 2         |
| TE-Hom    | 50 ± 4         | 27 ± 2         | 21 ± 1         | 51 ± 1        | 52 ± 1         | 20 ± 1         |
| TE-Full   | 61 ± 8         | 28 ± 8         | 13 ± 2         | 60 ± 1        | 47 ± 7         | 23 ± 7         |
| CPV-Naive | 51 ± 8         | 19 ± 5         | 9 ± 2          | 31 ± 16       | 30 ± 15        | 5 ± 2          |
| CPV-Pair  | <b>68 ± 11</b> | <b>44 ± 14</b> | <b>31 ± 13</b> | 2 ± 3         | 1 ± 2          | 0 ± 0          |
| CPV-Hom   | 63 ± 3         | 35 ± 5         | 27 ± 8         | <b>71 ± 8</b> | <b>60 ± 11</b> | <b>26 ± 14</b> |
| CPV-Full  | <b>73 ± 2</b>  | <b>40 ± 3</b>  | <b>28 ± 6</b>  | <b>76 ± 3</b> | <b>64 ± 6</b>  | <b>30 ± 10</b> |

produce compositionality on their own, or whether they work in conjunction with the CPV architecture.

**Results.** We evaluate the methods on both domains. To be considered successful in the crafting environment, the agent must perform the same sub-skills with the same types of objects as those seen in the reference trajectory. The results on the crafting environment are shown in Table 6.1, where we report the mean and standard deviation across 3 independent training seeds. We see that both the naïve model and the TECNet model struggle to represent these complex tasks, even the 4 skill tasks that are in the training distribution. We also find that both the CPV architecture and the regularization losses are necessary for both generalizing the longer tasks and composing multiple tasks. The pair loss seems to help mostly with generalization, while the homomorphism losses helps more with compositionality. CPVs are able to generalize to 8 and 16 skills, despite being trained on only 4 skill combinations, and achieve 76% success at composing two tasks just by adding their embedding vectors. Recall that CPVs are not explicitly *trained* to compose multiple reference trajectories in this way – the compositionality is an extrapolation from the training. The TE ablation, which does not use the subtraction of embeddings as input to the policy, shows worse compositionality than

Table 6.2: **3D Pick and Place Results.** Each model was trained on tasks with 1 to 2 skills. We evaluate the models on tasks with 1 and 2 skills, as well as the compositions of two 1 skill tasks. For each model we list the success rate of the best epoch of training. All numbers are averaged over 100 tasks. All models are variants of the object-centric architecture, shown in the supplement. We find that the CPV architecture plus regularizations enable composing two reference trajectories better than other methods.

| Model     | 1 Skill                      | 2 Skills                     | 1,1                          |
|-----------|------------------------------|------------------------------|------------------------------|
| Naive     | $65 \pm 7$                   | $34 \pm 8$                   | $6 \pm 2$                    |
| TECNet    | $82 \pm 6$                   | $50 \pm 2$                   | $33 \pm 4$                   |
| TE-Plain  | <b><math>91 \pm 2</math></b> | $55 \pm 5$                   | $22 \pm 2$                   |
| TE-Pair   | $81 \pm 11$                  | $51 \pm 8$                   | $15 \pm 3$                   |
| TE-Hom    | <b><math>92 \pm 1</math></b> | <b><math>59 \pm 1</math></b> | $24 \pm 12$                  |
| TE-Full   | $88 \pm 2$                   | $55 \pm 8$                   | $9 \pm 6$                    |
| CPV-Plain | $87 \pm 2$                   | $55 \pm 2$                   | $52 \pm 2$                   |
| CPV-Pair  | $82 \pm 4$                   | $42 \pm 3$                   | $7 \pm 1$                    |
| CPV-Hom   | $88 \pm 1$                   | $54 \pm 5$                   | <b><math>55 \pm 4</math></b> |
| CPV-Full  | $87 \pm 4$                   | $54 \pm 4$                   | <b><math>56 \pm 6</math></b> |

our method even with the homomorphism loss. This supports our hypothesis that structural constraints over the embedding representation contribute significantly to the learning.

These trends continue in the pick and place environment in Table 6.2, where we report the mean and standard deviation across 3 independent training seeds. In this environment, a trajectory is successful if the objects that were moved in the reference trajectory are in the correct positions: placed in each corner, placed inside the box, or stacked on top of a specific cube. As expected, TECNet performs well on 1 skill tasks which only require moving a single object. TECNet and the naïve model fail to compose tasks, but the CPV model performs as well at composing two 1-skill tasks as it does when imitating 2-skill tasks directly. As before, the TE ablation fails to compose as well as CPV, indicating that the architecture and losses together are needed to learn composable embeddings.

## 6.6 Discussion

Many tasks can be understood as a composition of multiple subtasks. To take advantage of this latent structure without subtask labels, we introduce the compositional plan vector (CPV) architecture along with a homomorphism-preserving loss function, and show that this learns a compositional representation of tasks. Our method learns a task representation and multi-task policy jointly. Our main idea is to condition the policy on the arithmetic difference

between the embedding of the goal task and the embedding of the trajectory seen so far. This constraint ensures that the representation space is structured such that subtracting the embedding of a partial trajectory from the embedding of the full trajectory encodes the portion of the task that remains to be completed. Put another way, CPVs encode tasks as a set of subtasks that the agent has left to perform to complete the full task. CPVs enable policies to generalize to tasks twice as long as those seen during training, and two plan vectors can be added together to form a new plan for performing both tasks.

We evaluated CPVs in a one-shot imitation learning setting. Extending our approach to a reinforcement learning setting is a natural next step, as well as further improvements to the architecture to improve efficiency. A particularly promising future direction would be to enable CPVs to learn from unstructured, self-supervised data, reducing the dependence on hand-specified objectives and reward functions.

# Chapter 7

## Conclusion

In this thesis, we approached the problem of compositional generalization in robotics by building modular neural network architectures and learning representations with arithmetic properties. In chapters 3 and 4, we showed that modular networks can be trained directly from data by applying different losses to different modules but still backpropagating gradients through the module interfaces. Particularly in chapter 3, we show that simply by training the modules in varied compositions with each other, they learn standard enough interfaces such that when modules that were never trained together are composed, they are able to produce a reasonable behavior.

In chapters 4 and 5 we took an object-centric view of robot learning. In chapter 4 we defined an object as a bounding box with consistent features as in and in chapter 5 we defined it more broadly as something that can be grasped and removed from a scene. We showed that by integrating these concepts of an object into the neural architecture and loss function of a robot learner, the learned behavior could learn more quickly and with less supervision than learning purely from image pixels.

Finally, in chapters 5 and 6 we developed an arithmetic approach to representing compositionality, where we trained embedding functions such that the embedding of a composition of concepts was equal to the sum of the embeddings of the concepts themselves. In chapter 5 the concepts were objects in the robot’s view, while in chapter 6 the concepts were primitive skills that composed into longer, more complex tasks for an agent. As a whole, this body of work demonstrated the effectiveness of structural priors, such as modularity and structured representations, in achieving compositional generalization for robotics without sacrificing the power of learning-based methods.

Of course, this work is only one step towards solving the problem of compositional generalization in robots. We offer some future directions of research that could advance this question:

**Lifelong learning.** While being able to generalize zero-shot to novel situations is important for artificial agents, in practice there will many situations that are simply too different for the agent to perform well on it’s first try. Instead, agents should be able to continue learning

from their experience in order adapt to new situations and acquire new concepts [80]. With a modular network architecture, the ability to spawn and train new modules as the learning progresses could enable continual learning without forgetting older skills. If an agent is learning an arithmetic feature space over tasks, then new tasks/objects could be integrated by continually optimizing for the arithmetic property as the agent acquires new data.

**Learning with language.** The compositional properties of natural language make it a good candidate for providing supervision for robots. A lot of progress has been made in language-conditioned instruction following [92, 146, 140, 13, 56], including policies operating over images [99, 3, 14] and in environments with object interactions [43, 125]. Modular networks have been shown to improve out of domain transfer in instruction following by composing neural task modules in accordance to the language instruction [19]. Further decomposition of the architecture across objects and space has the potential to improve generalization even more. A lifelong modular learning setting could benefit from in-the-loop language supervision to label novel concepts and decide when to create new modules. In a representation learning setting, language supervision can help structure the embedding space of tasks by enforcing that the task embeddings show the same compositional properties as the language.

# Bibliography

- [1] 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*. 2016, pp. 5074–5082.
- [2] Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew E. Taylor. “Online Multi-Task Learning for Policy Gradient Methods”. In: *Journal of Machine Learning Research* (2014).
- [3] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. “Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [4] Jacob Andreas, Dan Klein, and Sergey Levine. “Modular multitask reinforcement learning with policy sketches”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2017, pp. 166–175.
- [5] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. “Deep Compositional Question Answering with Neural Module Networks”. In: *CoRR* abs/1511.02799 (2015).
- [6] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, et al. “Hindsight Experience Replay”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5048–5058.
- [7] Aharon Azulay and Yair Weiss. “Why do deep convolutional networks generalize so poorly to small image transformations?” In: *arXiv preprint arXiv:1805.12177* (2018).
- [8] Pierre-Luc Bacon, Jean Harb, and Doina Precup. “The Option-Critic Architecture”. In: *AAAI*. 2016.
- [9] Alexander Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. “Reuse of Neural Modules for General Video Game Playing”. In: *CoRR* abs/1512.01537 (2015).

- [10] Serkan Cabi, Sergio Gómez Colmenarejo, Matthew W Hoffman, Misha Denil, Ziyu Wang, and Nando De Freitas. “The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously”. In: *arXiv preprint arXiv:1707.03300* (2017).
- [11] Rich Caruana. “Learning Many Related Tasks at the Same Time With Backpropagation”. In: *In Advances in Neural Information Processing Systems 7*. Morgan Kaufmann, 1995, pp. 657–664.
- [12] Richard Caruana. “Multitask Learning”. In: *Machine Learning* 28.1 (1997), pp. 41–75.
- [13] David L. Chen and Raymond J. Mooney. “Learning to Interpret Natural Language Navigation Instructions from Observations”. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*. 2011.
- [14] Howard Chen, Alane Shur, Dipendra Misra, Noah Snavely, and Yoav Artzi. “Touch-down: Natural Language Navigation and Spatial Reasoning in Visual Street Environments”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [15] Xi Chen, Ali Ghadirzadeh, Mårten Björkman, and Patric Jensfelt. “Adversarial feature training for generalizable robotic visuomotor control”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 1142–1148.
- [16] N. Chomsky and D.W. Lightfoot. *Syntactic Structures*. De Gruyter Reference Global. Mouton de Gruyter, 2002. ISBN: 9783110172799. URL: <https://books.google.co.uk/books?id=SNeHkMXHcd8C>.
- [17] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a similarity metric discriminatively, with application to face verification”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 539–546.
- [18] Alvaro Collet, Dmitry Berenson, Siddhartha S Srinivasa, and Dave Ferguson. “Object recognition and full pose registration from a single image for robotic manipulation”. In: *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*. IEEE. 2009, pp. 48–55.
- [19] Rodolfo Corona, Daniel Fried, Coline Devin, Dan Klein, and Trevor Darrell. “Modularity Improves Out-of-Domain Instruction Following”. In: *arXiv preprint arXiv:2010.12764* (2020).
- [20] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. “Robust Task-based Control Policies for Physics-based Characters”. In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 28.5 (2009), Article 170.
- [21] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. <http://pybullet.org>. 2016–2018.

- [22] Bruno Da Silva, George Konidaris, and Andrew Barto. “Learning parameterized skills”. In: *arXiv preprint arXiv:1206.6398* (2012).
- [23] Bruno Da Silva, George Konidaris, and Andrew Barto. “Learning parameterized skills”. In: *arXiv preprint arXiv:1206.6398* (2012).
- [24] Shreyansh Daftry, J Andrew Bagnell, and Martial Hebert. “Learning transferable policies for monocular reactive MAV control”. In: *International Symposium on Experimental Robotics*. Springer. 2016, pp. 3–11.
- [25] Peter Dayan and Geoffrey E. Hinton. “Feudal Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 5, [NIPS Conference]*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 271–278. ISBN: 1-55860-274-7. URL: <http://dl.acm.org/citation.cfm?id=645753.668239>.
- [26] Marc Peter Deisenroth, Peter Englert, Jan Peters, and Dieter Fox. “Multi-task policy search for robotics”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3876–3881.
- [27] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. “A Survey on Policy Search for Robotics”. In: *Found. Trends Robot* 2.1&#8211;2 (Aug. 2013), pp. 1–142. ISSN: 1935-8253.
- [28] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. “Learning modular neural network policies for multi-task and multi-robot transfer”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2169–2176.
- [29] Coline Devin, Payam Rowghanian, Chris Vigorito, Will Richards, and Khashayar Rohanimanesh. “Self-Supervised Goal-Conditioned Pick and Place”. In: *arXiv preprint arXiv:2008.11466* (2020).
- [30] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”. In: *CoRR* abs/1310.1531 (2013).
- [31] Chris Drummond. “Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks”. In: *J. Artif. Intell. Res. (JAIR)* (2002).
- [32] Chris Drummond. “Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks”. In: *JAIR* 16 (2002), pp. 59–104. URL: <http://jair.org/papers/paper904.html>.
- [33] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. “One-shot imitation learning”. In: *Advances in neural information processing systems*. 2017, pp. 1087–1098.
- [34] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. “Self-supervised visual planning with temporal skip connections”. In: *arXiv preprint arXiv:1710.05268* (2017).



- [35] Staffan Ekvall, Danica Kragic, and Patric Jensfelt. “Object detection and mapping for service robot tasks”. In: *Robotica* 25.2 (2007), pp. 175–187.
- [36] Kuan Fang, Yunfei Bai, Stefan Hinterstoisser, and Mrinal Kalakrishnan. “Multi-task Domain Adaptation for Deep Learning of Instance Grasping from Simulation”. In: *arXiv preprint arXiv:1710.06422* (2017).
- [37] Chelsea Finn and Sergey Levine. “Deep Visual Foresight for Planning Robot Motion”. In: *International Conference on Robotics and Automation (ICRA)*. 2017.
- [38] Chelsea Finn, Sergey Levine, and Pieter Abbeel. “Guided cost learning: Deep inverse optimal control via policy optimization”. In: *International Conference on Machine Learning*. 2016, pp. 49–58.
- [39] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. “Deep spatial autoencoders for visuomotor learning”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 512–519.
- [40] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. “One-shot visual imitation learning via meta-learning”. In: *arXiv preprint arXiv:1709.04905* (2017).
- [41] Peter R Florence, Lucas Manuelli, and Russ Tedrake. “Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation”. In: *arXiv preprint arXiv:1806.08756* (2018).
- [42] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. “Meta Learning Shared Hierarchies”. In: *CoRR* abs/1710.09767 (2017). arXiv: 1710.09767. URL: <http://arxiv.org/abs/1710.09767>.
- [43] Justin Fu, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama. “From language to goals: Inverse reinforcement learning for vision-based instruction following”. In: *arXiv preprint arXiv:1902.07742* (2019).
- [44] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. “Deep Predictive Policy Training using Reinforcement Learning”. In: *arXiv preprint arXiv:1703.00727* (2017).
- [45] Sergio Guadarrama, Erik Rodner, Kate Saenko, Ning Zhang, Ryan Farrell, Jeff Donahue, and Trevor Darrell. “Open-vocabulary Object Retrieval”. In: *Robotics Science and Systems (RSS)*. 2014.
- [46] Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. “Generalizing Plans to New Environments in Relational MDPs”. In: *In International Joint Conference on Artificial Intelligence*. 2003.

- [47] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. “Latent Space Policies for Hierarchical Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018*. 2018, pp. 1846–1855. URL: <http://proceedings.mlr.press/v80/haarnoja18a.html>.
- [48] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. “Composable deep reinforcement learning for robotic manipulation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6244–6251.
- [49] Aleksi Hämäläinen, Karol Arndt, Ali Ghadirzadeh, and Ville Kyrki. “Affordance learning for end-to-end visuomotor robot control”. In: *arXiv preprint arXiv:1903.04053* (2019).
- [50] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. “Learning an embedding space for transferable robot skills”. In: (2018).
- [51] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. “Mask r-cnn. arXiv preprint”. In: *arXiv preprint arXiv:1703.06870* (2017).
- [52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.
- [53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep residual learning for image recognition. CoRR abs/1512.03385* (2015). 2015.
- [54] Nicolas Heess, Gregory Wayne, Yuval Tassa, Timothy P. Lillicrap, Martin A. Riedmiller, and David Silver. “Learning and Transfer of Modulated Locomotor Controllers”. In: *CoRR abs/1610.05182* (2016). arXiv: 1610.05182. URL: <http://arxiv.org/abs/1610.05182>.
- [55] David Held, Sebastian Thrun, and Silvio Savarese. “Robust single-view instance recognition”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 2152–2159.
- [56] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, et al. “Grounded Language Learning in a Simulated 3D World”. In: *CoRR abs/1706.06551* (2017). arXiv: 1706.06551. URL: <http://arxiv.org/abs/1706.06551>.
- [57] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniard, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. “Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 858–865.
- [58] Judy Hoffman, Sergio Guadarrama, Eric Tzeng, Ronghang Hu, Jeff Donahue, Ross Girshick, Trevor Darrell, and Kate Saenko. “LSDA: Large Scale Detection through Adaptation”. In: *Neural Information Processing Systems (NIPS)*. 2014.

- [59] Ronghang Hu, Daniel Fried, Anna Rohrbach, Dan Klein, Trevor Darrell, and Kate Saenko. “Are you looking? grounding to multiple modalities in vision-and-language navigation”. In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*. 2019.
- [60] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. “Reinforcement learning with unsupervised auxiliary tasks”. In: *arXiv preprint arXiv:1611.05397* (2016).
- [61] Stephen James, Michael Bloesch, and Andrew J Davison. “Task-embedded control networks for few-shot imitation learning”. In: *arXiv preprint arXiv:1810.03237* (2018).
- [62] Eric Jang, Sudheendra Vijayanarasimhan, Peter Pastor, Julian Ibarz, and Sergey Levine. “End-to-End Learning of Semantic Grasping”. In: *arXiv preprint arXiv:1707.01932* (2017).
- [63] Dinesh Jayaraman and Kristen Grauman. “Learning image representations tied to ego-motion”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1413–1421.
- [64] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. “CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 1988–1997.
- [65] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. “The Malmo Platform for Artificial Intelligence Experimentation.” In: *IJCAI*. 2016, pp. 4246–4247.
- [66] Rico Jonschkowski and Oliver Brock. “Learning state representations with robotic priors”. In: *Autonomous Robots* 39.3 (2015), pp. 407–428.
- [67] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, et al. “QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation”. In: *arXiv preprint arXiv:1806.10293* (2018).
- [68] Takahito Kawanishi, Hiroshi Murase, and Shigeru Takagi. “Quick 3D object detection and localization by dynamic active search with multiple active cameras”. In: *Pattern Recognition, 2002. Proceedings. 16th International Conference on*. Vol. 2. IEEE. 2002, pp. 605–608.
- [69] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2014.
- [70] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. “Skip-Thought Vectors”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 3294–3302. URL: <http://papers.nips.cc/paper/5950-skip-thought-vectors.pdf>.

- [71] Jens Kober, J. Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *I. J. Robotics Res.* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.
- [72] J. Zico Kolter and Andrew Y. Ng. “Learning omnidirectional path following using dimensionality reduction”. In: *in Proceedings of Robotics: Science and Systems*. 2007.
- [73] George Konidaris and Andrew Barto. “Autonomous shaping: knowledge transfer in reinforcement learning”. In: *International Conference on Machine Learning (ICML)*. 2006, pp. 489–496.
- [74] George Konidaris and Andrew G Barto. “Building Portable Options: Skill Transfer in Reinforcement Learning.” In: *Proc. International Joint Conference on Artificial Intelligence*. 2007, pp. 895–900.
- [75] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, et al. “Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations”. In: 2016. URL: <https://arxiv.org/abs/1602.07332>.
- [76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [77] Andras Gabor Kupcsik, Marc Peter Deisenroth, Jan Peters, and Gerhard Neumann. “Data-efficient generalization of robot skills with contextual policy search”. In: *Twenty-Seventh AAAI Conference on Artificial Intelligence*. 2013.
- [78] Brenden Lake and Marco Baroni. “Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2873–2882.
- [79] Ian Lenz, Honglak Lee, and Ashutosh Saxena. “Deep learning for detecting robotic grasps”. In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 705–724.
- [80] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. “Continual learning for robotics”. In: *arXiv preprint arXiv:1907.00182* (2019).
- [81] Sergey Levine and Pieter Abbeel. “Learning neural network policies with guided policy search under unknown dynamics”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 1071–1079.
- [82] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-End Training of Deep Visuomotor Policies”. In: *Journal of Machine Learning Research (JMLR)* (2016).
- [83] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-End Training of Deep Visuomotor Policies.” In: *Journal of Machine Learning Research* 17 (2016), pp. 1–40.

- [84] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. “Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection”. In: *International Symposium on Experimental Robotics (ISER 2016)*. 2016.
- [85] Omer Levy and Yoav Goldberg. “Dependency-based word embeddings”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vol. 2. 2014, pp. 302–308.
- [86] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. “Continuous control with deep reinforcement learning”. In: *CoRR* abs/1509.02971 (2015).
- [87] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.
- [88] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.
- [89] Libin Liu and Jessica Hodgins. “Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning”. In: *ACM Transactions on Graphics* 36.3 (2017).
- [90] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. “Object-centric learning with slot attention”. In: *Advances in Neural Information Processing Systems* 33 (2020).
- [91] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.
- [92] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. “Walk the Talk: Connecting Language, Knowledge, and Action in Route Instructions”. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*. 2006.
- [93] Michael G. Madden and Tom Howley. “Transfer of Experience Between Reinforcement Learning Environments with Progressive Difficulty”. In: *Artificial Intelligence Review* 21.3 (2004).
- [94] Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. “Hierarchical Visuomotor Control of Humanoids”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=BJfYvo09Y7>.
- [95] Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. “Neural Probabilistic Motor Primitives for Humanoid Control”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=BJl6TjRcY7>.

- [96] Lilyana Mihalkova and Raymond J. Mooney. “Transfer Learning from Minimal Target Data by Mapping across Relational Domains”. In: *Transfer Learning from Minimal Target Data by Mapping across Relational Domains*. 2009.
- [97] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.
- [98] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. “Learning to navigate in cities without a map”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2018.
- [99] Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. “Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 2667–2678.
- [100] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. “Asynchronous methods for deep reinforcement learning”. In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.
- [101] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013).
- [102] Richard Montague. “Universal grammar”. In: *Theoria* 36.3 (1970), pp. 373–398.
- [103] Igor Mordatch, Nikhil Mishra, Clemens Eppner, and Pieter Abbeel. “Combining model-based policy search with online model learning for control of physical humanoids”. In: *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*. 2016, pp. 242–248.
- [104] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. “Driving policy transfer via modularity and abstraction”. In: *arXiv preprint arXiv:1804.09364* (2018).
- [105] Tushar Nagarajan and Kristen Grauman. “Attributes as operators: factorizing unseen attribute-object compositions”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 169–185.
- [106] Junhyuk Oh, Valliappa Chockalingam, Satinder P. Singh, and Honglak Lee. “Control of Memory, Active Perception, and Action in Minecraft”. In: *CoRR* abs/1605.09128 (2016).
- [107] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. “Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning”. In: *International Conference on Machine Learning*. 2017, pp. 2661–2670.

- [108] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. “Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning”. In: *ACM Trans. Graph.* 35.4 (July 2016), 81:1–81:12. ISSN: 0730-0301. DOI: 10.1145/2897824.2925881. URL: <http://doi.acm.org/10.1145/2897824.2925881>.
- [109] J. Peters, K. Mülling, and Y. Altun. “Relative Entropy Policy Search”. In: *AAAI*. 2010.
- [110] Jan Peters, Katharina Mülling, and Yasemin Altun. “Relative Entropy Policy Search”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. 2010.
- [111] Jan Peters and Stefan Schaal. “Natural actor-critic”. In: *Neurocomputing* 71 ().
- [112] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients”. In: *Neural Networks* (2008).
- [113] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. “The curious robot: Learning visual representations via physical interactions”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 3–18.
- [114] Lerrel Pinto and Abhinav Gupta. “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 3406–3413.
- [115] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. “Transfer Learning in Reinforcement Learning Problems Through Partial Policy Recycling”. In: *Machine Learning: ECML 2007: 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. Proceedings*. Ed. by Joost N. Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenić, and Andrzej Skowron. 2007.
- [116] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.
- [117] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “‘’ Why should I trust you?’’ Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.
- [118] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, et al. “Imagenet large scale visual recognition challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.
- [119] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. “Progressive Neural Networks”. In: *CoRR* abs/1606.04671 (2016).

- [120] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. “Universal Value Function Approximators”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1312–1320.
- [121] Tanner Schmidt, Katharina Hertkorn, Richard Newcombe, Zoltan Marton, Michael Suppa, and Dieter Fox. “Depth-based tracking with physical constraints for robot manipulation”. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 119–126.
- [122] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.
- [123] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. “Trust Region Policy Optimization”. In: *International Conference on Machine Learning (ICML)*. 2015.
- [124] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. “Overfeat: Integrated recognition, localization and detection using convolutional networks”. In: *arXiv preprint arXiv:1312.6229* (2013).
- [125] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. “ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2020.
- [126] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.
- [127] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. “Discriminative learning of deep convolutional feature point descriptors”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 118–126.
- [128] Karen Simonyan and Andrew Zisserman. “Two-stream convolutional networks for action recognition in videos”. In: *Advances in neural information processing systems*. 2014, pp. 568–576.
- [129] Satinder P Singh. “The efficient learning of multiple task sequences”. In: *Advances in neural information processing systems*. 1992, pp. 251–258.
- [130] Kihyuk Sohn. “Improved Deep Metric Learning with Multi-class N-pair Loss Objective”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 2016, pp. 1857–1865.



- [131] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. “Hierarchical Reinforcement Learning for Zero-shot Generalization with Subtask Dependencies”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 7156–7166.
- [132] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [133] Freek Stulp, Gennaro Raiola, Antoine Hoarau, Serena Ivaldi, and Olivier Sigaud. “Learning compact parameterized skills with a single regression”. In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE. 2013, pp. 417–422.
- [134] Priya Sundareshan, Jennifer Grannen, Brijen Thananjeyan, Ashwin Balakrishna, Michael Laskey, Kevin Stone, Joseph E Gonzalez, and Ken Goldberg. “Learning Rope Manipulation Policies Using Dense Object Descriptors Trained on Synthetic Depth Data”. In: *arXiv preprint arXiv:2003.01835* (2020).
- [135] Kanata Suzuki, Yasuto Yokota, Yuzi Kanazawa, and Tomoyoshi Takebayashi. “Online Self-Supervised Learning for Object Picking: Detecting Optimum Grasping Position using a Metric Learning Approach”. In: *2020 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2020, pp. 205–212.
- [136] Jie Tang, Stephen Miller, Arjun Singh, and Pieter Abbeel. “A textured object recognition pipeline for color and depth image data”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 3467–3474.
- [137] Ajay Kumar Tanwani, Pierre Sermanet, Andy Yan, Raghav Anand, Mariano Phielipp, and Ken Goldberg. “Motion2Vec: Semi-Supervised Representation Learning from Surgical Videos”. In: *arXiv preprint arXiv:2006.00545* (2020).
- [138] Matthew E. Taylor and Peter Stone. “Transfer Learning for Reinforcement Learning Domains: A Survey”. In: *Journal of Machine Learning Research* 10 (2009), pp. 1633–1685.
- [139] Matthew Taylor, Peter Stone, and Yaxin Liu. “Transfer Learning via Inter-Task Mappings for Temporal Difference Learning”. In: *Journal of Machine Learning Research* 8.1 (2007), pp. 2125–2167.
- [140] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. “Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation.” In: *AAAI*. Vol. 1. 2011, p. 2.
- [141] E. Theodorou, J. Buchli, and S. Schaal. “A Generalized Path Integral Control Approach to Reinforcement Learning.” In: *JMLR* 11 (2010).

- [142] E. Todorov, T. Erez, and Y. Tassa. “MuJoCo: A physics engine for model-based control”. In: *International Conference on Intelligent Robots and Systems (IROS)*. 2012.
- [143] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. “Simultaneous Deep Transfer Across Domains and Tasks”. In: *International Conference in Computer Vision (ICCV)*. 2015.
- [144] Matthew Veres, Ian Cabral, and Medhat Moussa. “Incorporating Object Intrinsic Features Within Deep Grasp Affordance Prediction”. In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6009–6016.
- [145] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. “FeUdal Networks for Hierarchical Reinforcement Learning”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3540–3549. URL: <http://dl.acm.org/citation.cfm?id=3305890.3306047>.
- [146] Adam Vogel and Dan Jurafsky. “Learning to follow navigational directions”. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2010, pp. 806–814.
- [147] Dequan Wang, Coline Devin, Qi-Zhi Cai, Fisher Yu, and Trevor Darrell. “Deep object-centric policies for autonomous driving”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8853–8859.
- [148] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Machine Learning* 8.3-4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: <http://dx.doi.org/10.1007/BF00992696>.
- [149] 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”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.
- [150] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. “End-to-end learning of driving models from large-scale video datasets”. In: *Computer Vision and Pattern Recognition*. 2016.
- [151] Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker, Alberto Rodriguez, and Jianxiong Xiao. “Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge”. In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 1386–1383.
- [152] Yubo Zhang, Hao Tan, and Mohit Bansal. “Diagnosing the Environment Bias in Vision-and-Language Navigation”. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (2020).

# Appendix A

## Grasp2Vec

### A.1 Qualitative Detection Results

**Success Cases** Figures A.2, A.3, A.4 depict examples from the real-world localization task in which grasp2vec demonstrates surprisingly effective localization capabilities.

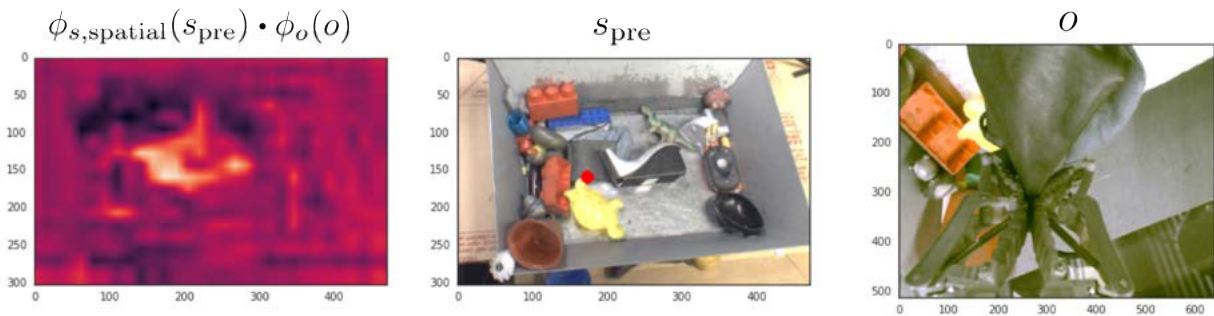


Figure A.2: Training objects. Recognize deformable object (bag) in a large amount of clutter.

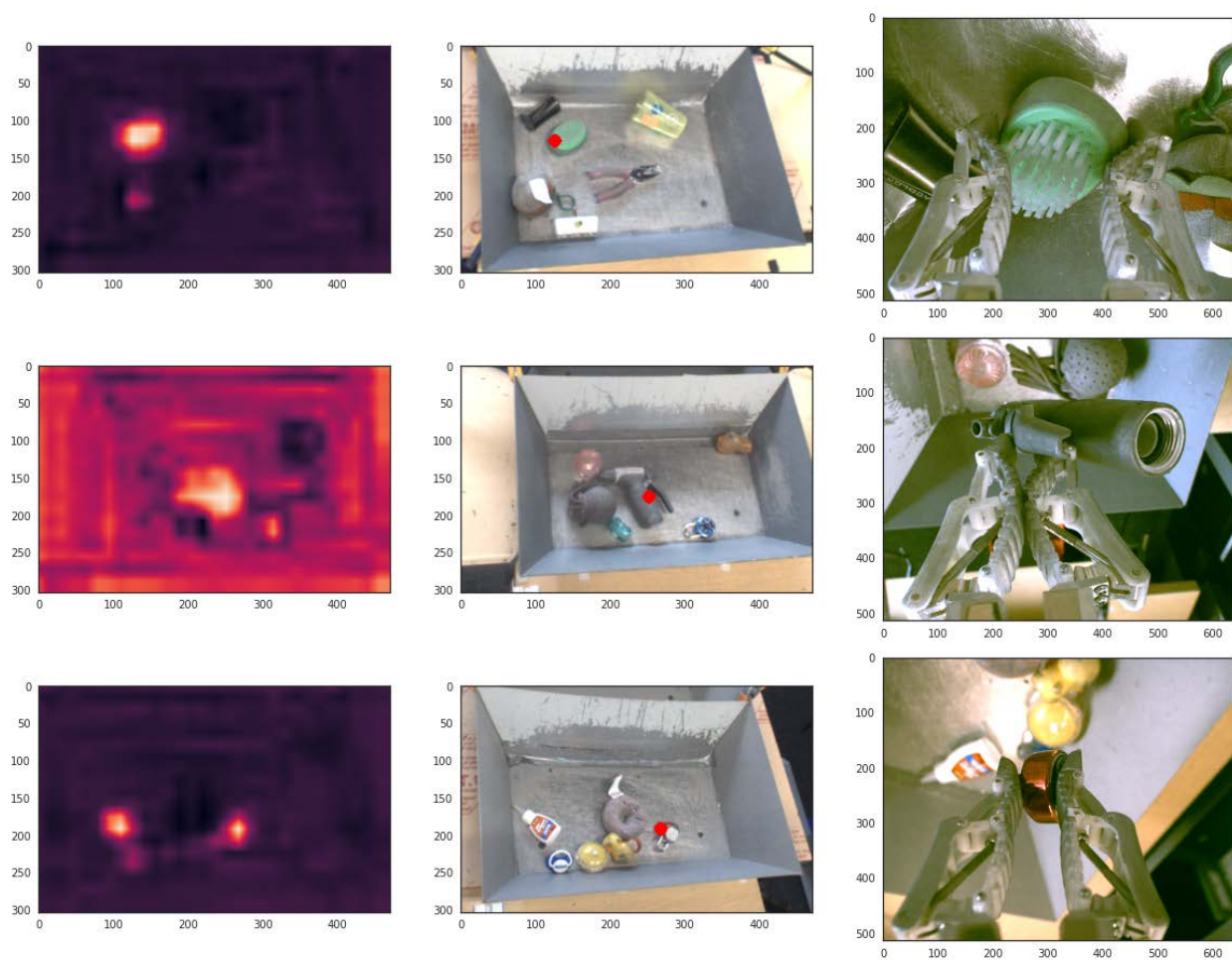


Figure A.3: Testing objects. Recognizes correct object, even when goal is presented from a different pose than the object's pose in the bin.

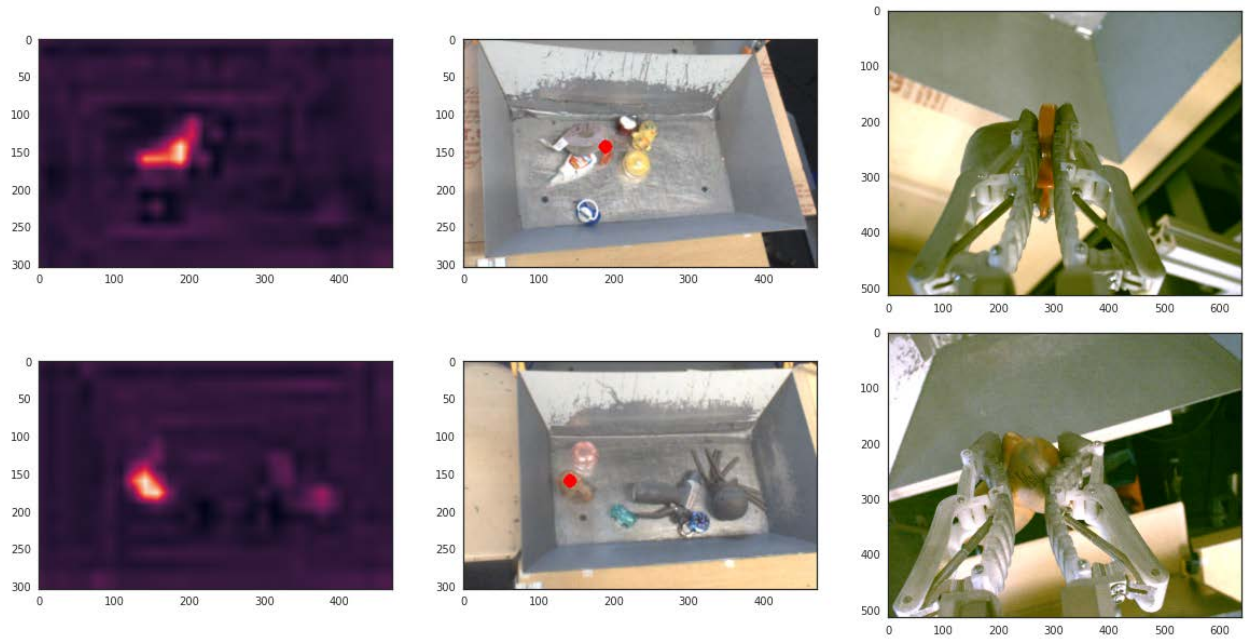


Figure A.4: Testing objects. Recognizes objects by parts when the goal object is occluded by the gripper.

**Failure Cases** Figures A.5, A.6 depict examples where Grasp2Vec embeddings make mistakes in localization.

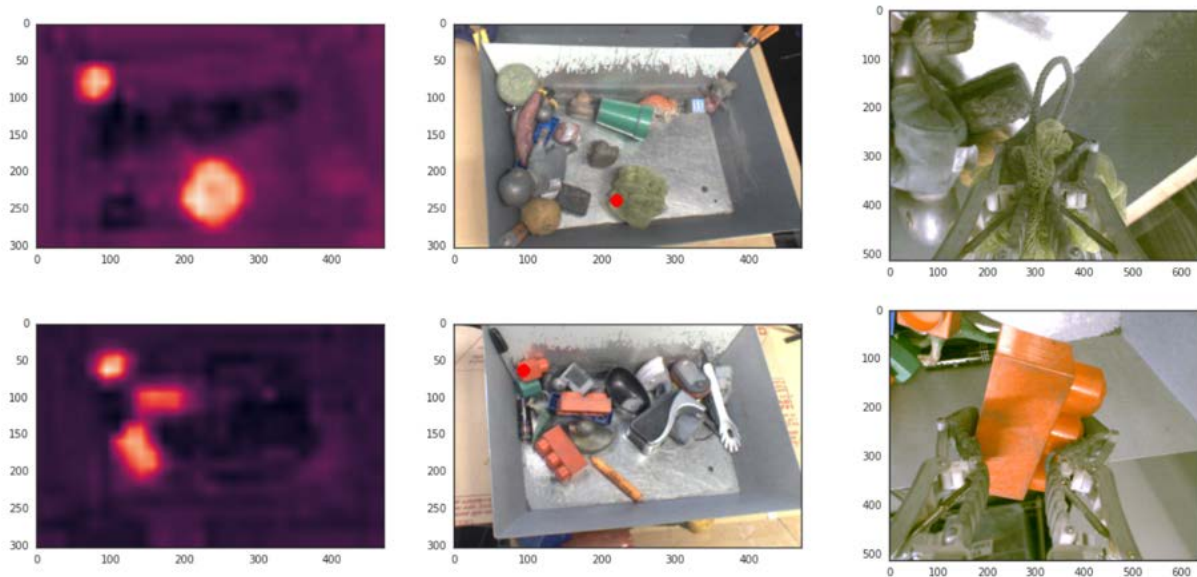


Figure A.5: Training objects. Embedding localizes objects with the correct colors but wrong (though similar) shape.

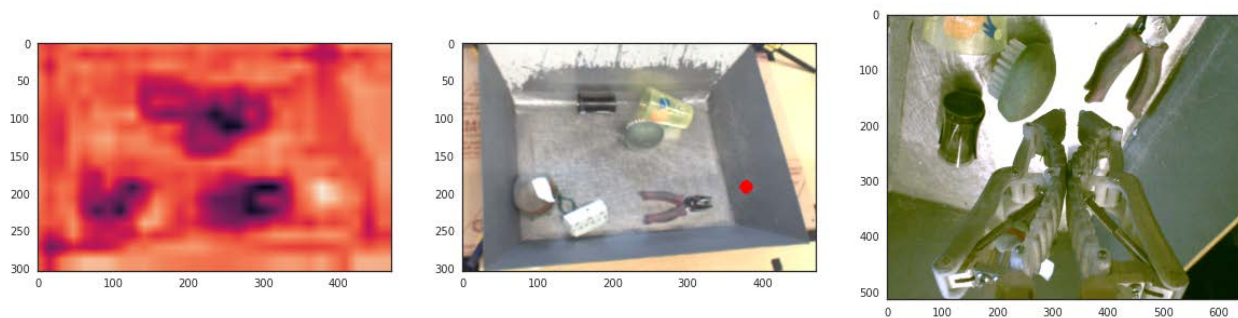


Figure A.6: Test objects. Failed grasp incorrectly labeled as successful, resulting in an empty (meaningless) goal for localization.

## A.2 Simulation Experiment Details

The simulated experiments use the Bullet [21] simulator with a model of a 7-DoF Kuka arm. We use 44 unique objects for training and 15 unique objects for evaluation. The training and evaluation objects are mutually exclusive, and each object has a unique mesh and texture. All objects are scans of mugs, bottles, cups, and bowls.

For data collection and evaluation, a particular scene has up to 6 objects sampled from the total objects without replacements; this means that no scene has multiple copies of a particular object. The objects are dropped into the scene at a random pose, and the objects may bounce onto each other.

To train the grasping policy, we use the following procedure. In each episode of the data collection and learning loop, an indiscriminate grasping policy collects 10 objects from the bin and saves their images to a goal set  $G$ . Afterwards, the data collection protocol is switched to on-policy instance grasping, using previously grasped images  $\mathbf{o}^{(1)} \dots \mathbf{o}^{(10)}$  as subsequent goals. Exploration policy hyperparameters are as described in [67], and we parallelize data collection across 1000 simulated robots for each experiment.

### A.2.1 Real-World Experiment Details

We use roughly 500 objects for training and 42 unseen objects for evaluation. Objects are not restricted to object categories. For data collection and evaluation, 6 objects are placed randomly into the bin. After grasping an object, the robot drops it back into the bin to continue. The objects in a bin are switched out about twice a day and 6-7 robots are used in parallel, each with its own bin.



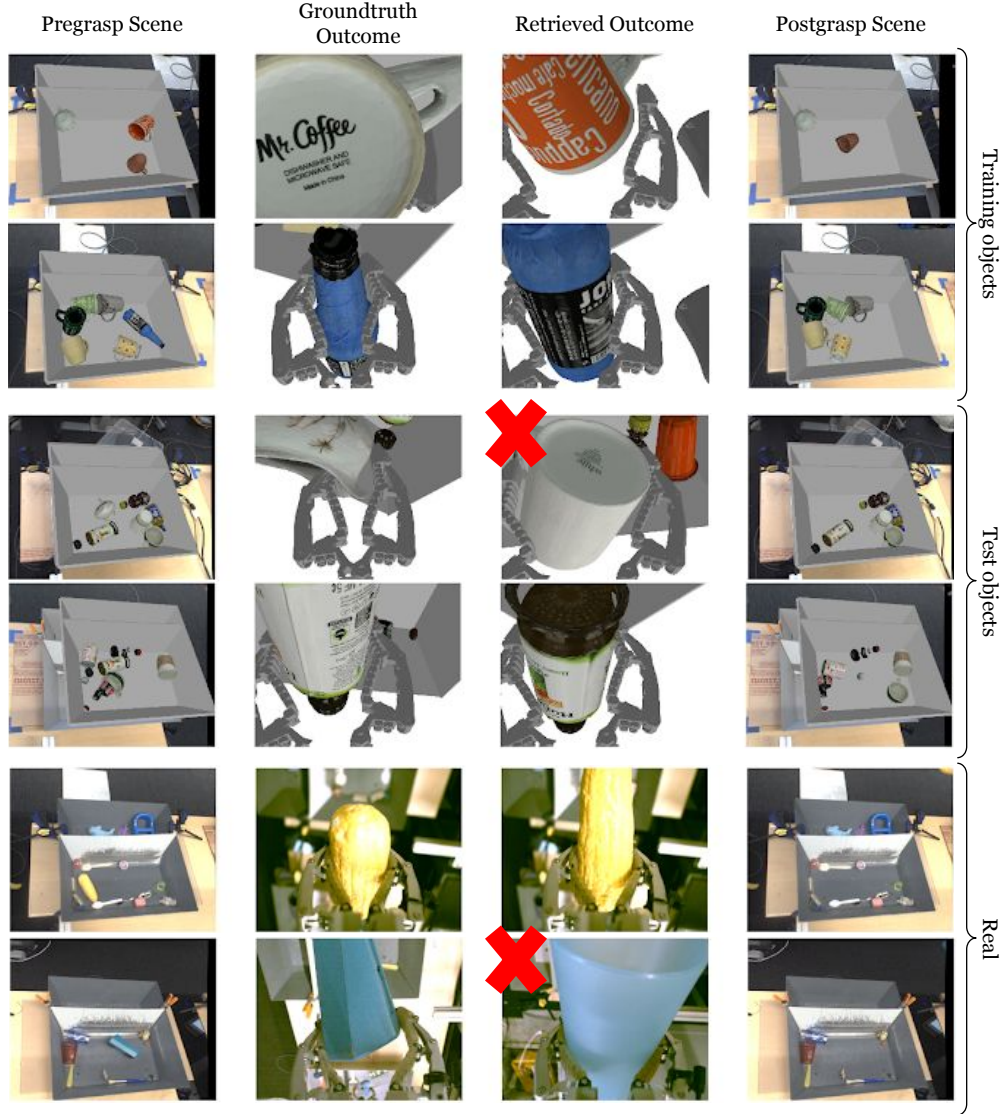
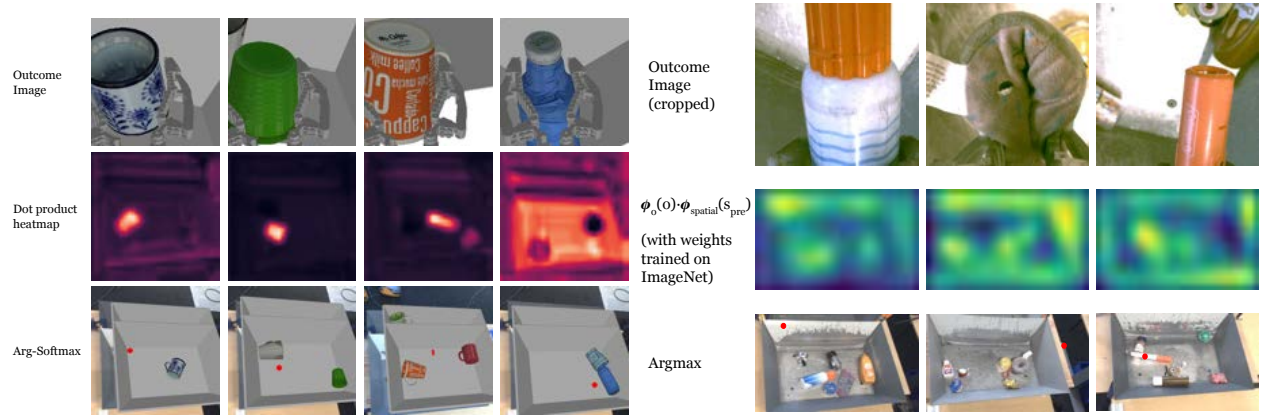


Figure A.1: This table illustrates the object recall property. From left to right: The scene before grasping, the grasped object, the outcome image retrieved by subtracting the postgrasp scene embedding from the pregrasp scene embedding, and lastly the postgrasp scene. We show example successes and failures (the later are marked with a red X). Failures occur in the test object set because multiple white objects had similar embeddings. Failures occur in the real data because the diversity of objects is very high, which likely makes the embeddings less partitioned between object instances.





(a) Localization using untrained weights in simulation. (b) Localization using weights trained on imagenet.

Figure A.7: (a) The detection analysis with an untrained model. This verifies that our loss, rather than the architecture on its own, enables the detection property. (b) The failure of localization indicates that ImageNet features are not consistent between scene and outcome images, probably because of resolution and pose differences.

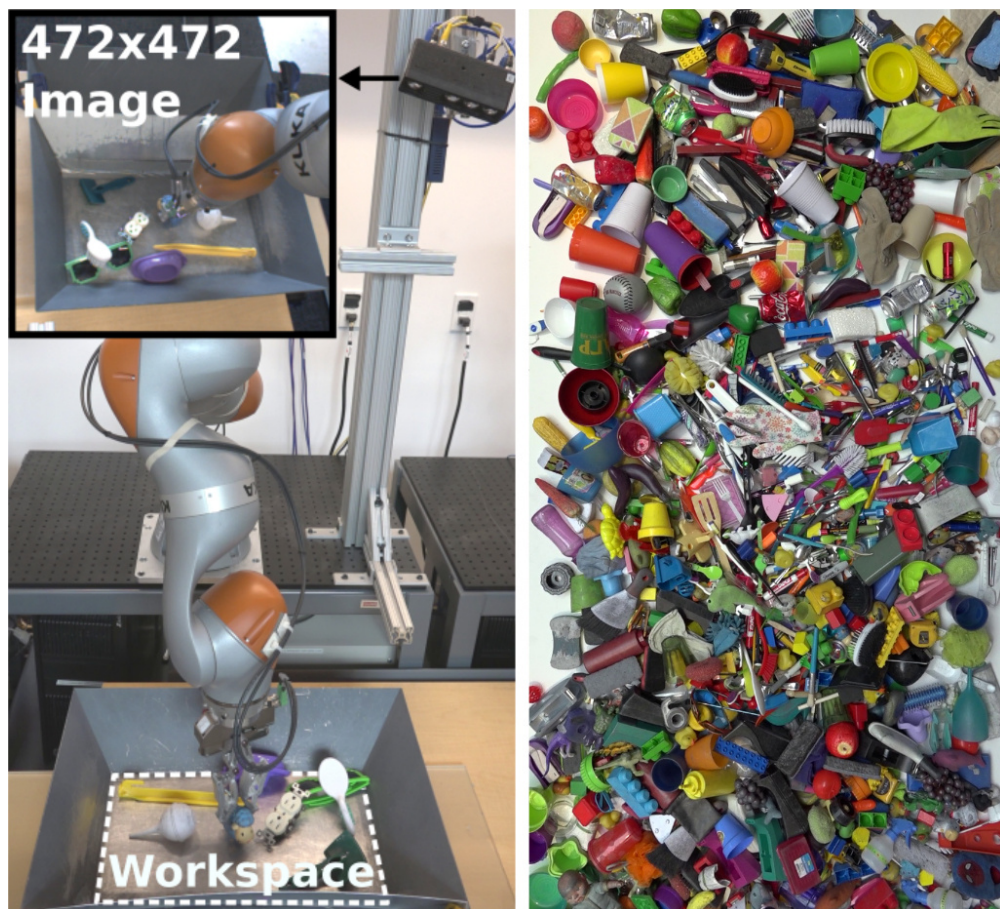




Figure A.8: Objects used for evaluation on unseen (test) objects.

# Appendix B

## Compositional Plan Vectors

### B.1 Network Architectures

**Crafting environment** The observation is an RGB image of 33x30 pixels. The architecture for  $g$  concatenates the first and last image of the reference trajectory along the channel dimension, to obtain an input size of 33x30x6. This is followed by 4 convolutions with 16, 32, 64, and 64 channels, respectively, with ReLU activations. The 3x3x64 output is flattened and a fully connected layer reduces this to the desired embedding dimension. The same architecture is used for the TECNet encoder. For the policy, the observation is passed through a convolutional network with the same architecture as above and the output is concatenated



(a) Shows a state observation as rendered for the agent. The white square in the bottom-left indicates that an object is held by the agent.

(b) Shows the same state, but rendered in a human-readable format. The axe shown in the bottom row indicates that the agent is currently holding an axe.

with the subtraction of embeddings as defined in the paper’s method. This concatenation is passed through a 4 layer fully connected network with 64 hidden units per layer and ReLU activations. The output is softmaxed to produce a distribution over the 6 actions. The TECNet uses the same architecture, but the reference trajectory embeddings are normalized there is no subtraction; instead, the initial image of the current trajectory is concatenated with the observation. The naive model uses the same architecture but all four input images are concatenated for the initial convolutional network and there is no concatenation at the embedding level.

**3D environment** The environment has 6 objects: 4 cubes (red, blue, green, white), a box body and a box lid. The state space is the concatenation of the  $(x, y, z)$  positions of these objects, resulting in an 18-dimensional state. As the object positions are known, we use an attention over the objects as part of the action, as shown in Figure B.1. The actions are 2 positions: the  $(x_0, y_0)$  position at which to grasp and the  $(x_1, y_1)$  position at which to place. When training the policy using the object centric model,  $(x_0, y_0)$  is a weighted sum of the object positions, with the  $z$  coordinate being ignored. Weights over the 6 object are output by a neural network given the difference of CPVs and the current observation. At evaluation time,  $(x_0, y_0)$  is the  $\arg \max$  object position. This means that all policies will always grasp at an object position. For  $(x_1, y_1)$ , we do not have the same constraint. Instead, the softmaxed weights over the objects are concatenated with the previous layer’s activations, and another fully connected layer maps this directly to continuous valued  $(x_1, y_1)$ . This means that the policy can place at any position in the workspace. The naïve model, TECNet model, and CPV models all use this object-centric policy, then only differ in how the input to the policy.

## B.2 Hyperparameters

We compared all models across embedding dimension sizes of [64,128,256, and 512]. In the crafting environment, the 512 size was best for all methods. In the grasping environment, the 64 size was best for all methods. For TECNets, we tested  $\lambda_{\text{ctr}} = 1$  and 0.1, and found that 0.1 was best. All models are trained on either k-80 GPUs or Titan X GPUs.

## B.3 Additional Experiments

We ran a pared down experiment on a ViZDoom environment to show the method working from first person images, as shown in B.3. In the experiment, the skills are reaching 4 different waypoints in the environment. The actions are “turn left,” “turn right,” and “go forward.” The observation space consists of a first person image observation as well as the  $(x, y)$  locations of the waypoints. We evaluate on trajectories that must visit 1 or 2 waypoints



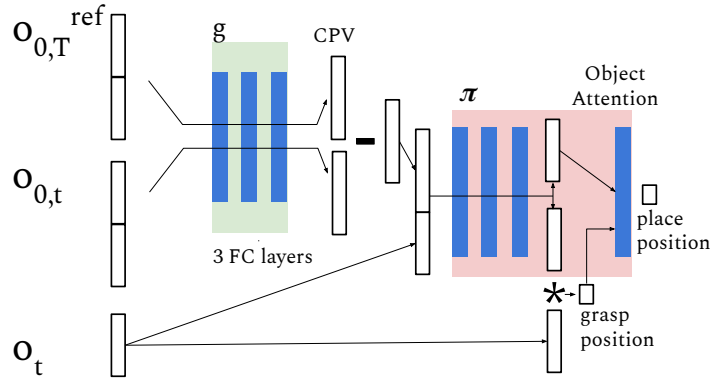


Figure B.1: The object-centric network architecture we use for the 3D grasping environment. Because the observations include the concatenated positions of the objects in the scene, the policy chooses a grasp position by predicting a discrete classification over the objects grasping at the weighted sum of the object positions. The classification logits are passed back to the network to output the position at which to place the object.

(skills), and also evaluate on the compositions of these trajectories. The policies were only trained on trajectories that visit up to 3 waypoints. These evaluations are shown in B.1.

Figure B.2: First person view in VizDoom env.

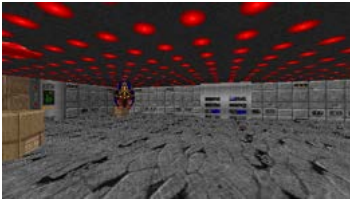


Table B.1: **ViZDoom Navigation Results.** All numbers are success rates of arriving within 1 meter of each waypoint.

| Model  | 1 Skill | 2 Skills | 1+1       | 2+2       |
|--------|---------|----------|-----------|-----------|
| Naive  | 97      | 94       | 36.7      | 2         |
| TECNet | 96      | 95.3     | 48.3      | 0         |
| CPV    | 93      | 90.7     | <b>91</b> | <b>64</b> |