Learning to Generalize via Self-Supervised Prediction

Deepak Pathak

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-132
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-132.html

August 26, 2019

Learning to Generalize via Self-Supervised Prediction

by

Deepak Pathak

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, Chair
Professor Alexei A. Efros, Chair
Professor Jitendra Malik
Professor Alison Gopnik

Summer 2019

Learning to Generalize via Self-Supervised Prediction

Abstract

Learning to Generalize via Self-Supervised Prediction

by

Deepak Pathak

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Trevor Darrell, Chair

Professor Alexei A. Efros, Chair

Generalization, i.e., the ability to adapt to novel scenarios, is the hallmark of human intelligence. While we have systems that excel at recognizing objects, cleaning floors, playing complex games and occasionally beating humans, they are incredibly specific in that they only perform the tasks they are trained for and are miserable at generalization. Could optimizing towards fixed external goals be hindering the generalization instead of aiding it? In this thesis, we present our initial efforts toward endowing artificial agents with a human-like ability to generalize in diverse scenarios. The main insight is to first allow the agent to learn general-purpose skills in a completely self-supervised manner, without optimizing for any external goal.

To be able to learn on its own, the claim is that an artificial agent must be embodied in the world, develop an understanding of its sensory input (e.g., image stream) and simultaneously learn to map this understanding to its motor outputs (e.g., torques) in an unsupervised manner. All these considerations lead to two fundamental questions: how to learn rich representations of the world similar to *what* humans learn?; and how to re-use such a representation of past knowledge to incrementally adapt and learn more about the world similar to *how* humans do? We believe *prediction* is the key to this answer. We propose generic mechanisms that employ prediction as a supervisory signal in allowing the agents to learn sensory representations as well as motor control. These two abilities equip an embodied agent with a basic set of general-purpose skills which are then later repurposed to perform complex tasks.

We discuss how this framework can be instantiated to develop curiosity-driven agents (virtual as well as real) that can learn to play games, learn to walk, and learn to perform real-world object manipulation without any rewards or supervision. These self-supervised robotic agents, after exploring the environment, can generalize to find their way in office environments, tie knots using rope, rearrange object configuration, and compose their skills in a modular fashion.

To my parents, Durga and Madhwa Nand Pathak.

Contents

List of Figures

List of Tables

Acknowledgments

It just so happens that I get to take credit as the sole author of this dissertation. In reality, this is a purely collaborative effort, from working alongside my advisors, mentors, colleagues, friends, and family. Several people have continually supported and guided me, throughout my Ph.D., which in turn has made this dissertation a reality.

I would like to start by expressing sincere gratitude to my dear Ph.D. advisors, Trevor Darrell, and Alyosha Efros, for taking me in as their advisee.

Ever since my admissions interview with Trevor, I was certain that I wanted to join his research group, and feel grateful for the opportunity to have been a part of it. Trevor taught me how to collect my ever wandering thoughts into a potentially cool idea. I learned from him how to think broad, yet not losing focus from the ultimate long-term agenda. His consistent reminder for being confident and broad has made me the researcher I am today. One of the biggest strengths Trevor has instilled in me is to not give up. Many of my paper submissions would not have been possible if he did not consistently have my back, always helping and encouraging. Despite being responsible for leading such a large research group, Trevor has this unique ability to take extremely good care of each of his students. Trevor shielded me from any external hiccups and allowed me to primarily focus on research. No matter what the issue was, I could always count on him to help me find a way out of it. I am thankful for his continued support and guidance as I embark on my academic journey ahead.

As the time to leave Berkeley gets closer, I wonder if Alyosha's imprint will ever leave me. In his words, I carry a good discriminator model of him in my mind. I used to be an engineer, but it is because of Alyosha that I can (even remotely) consider myself a scientist in disguise. I owe it to him, for showing me the Science in Computer Science. Looking at the big picture, constantly questioning fundamentals, writing concise and punchy sentences, being selfless in every circumstance, and being bold yet reasonable are some of the invaluable gems I received by working with him. I was and still, am fascinated by the stories of how huge scientific discoveries were made through argumentative discussions. I used to think that the era was over, but Alyosha showed me otherwise. From long late-night discussions, to even longer arguments during hikes, I think I spent way more hours with him than Ph.D. students get to these days. Of course, we were not always productive, but I am hopeful those discussions will guide me in the long term! I could go on and on had I begun writing this in time, but again, "if you do it at the last minute, it only takes a minute" – wish I could unlearn!

I am also extremely grateful to have Jitendra Malik and Alison Gopnik as my thesis

committee members. The ideology proposed in this dissertation is heavily motivated by their research principles. I learned from Jitendra about the importance of scientific rigor, pursuing big ideas and being well-read. He is a walking encyclopedia, and the gold standard for all of us to aim for. Much of my memories of the UC Berkeley culture revolve around the trio of Jitendra, Trevor, and Alyosha, participating actively in Monday morning reading groups.

I would like to also thank Pieter Abbeel and Sergey Levine for their precise critique of my ideas. My maneuvering into the Reinforcement Learning (RL) world would not have been possible without them. I would like to especially thank Pieter for helping with my faculty applications and taking the time out for frequent meetings to discuss research ideas.

Working with Abhinav Gupta at Pittsburgh was one of the best times in my Ph.D. I learned a lot from him, about research and mentoring students, in those few months. His research style and emphasis on figuring out *singlemost big contribution of a paper* have had a great impact on my thinking style. I am excited about having him as my colleague and mentor at CMU.

I have also been greatly influenced by the research journeys of Yann LeCun and Geoffrey Hinton as they helped make deep learning what it is today. I was amazed by Yann's humility and curiosity when I first met him in my first year. The several discussions I have had with him over the years have acted as great motivation for me to aspire to do great research. I am also thankful to Miro Dudík with whom I interned at Microsoft Research (MSR), New York, during my third year of undergraduate studies. If I had not met him, I would have never applied for a Ph.D. program. I also thank Vinay Namboodiri for being a great mentor.

I am extremely lucky to have also worked with Philipp Krähenbühl. Thank you for teaching me optimization, how to think concretely about research ideas, trusting intuitions, being confident, and the importance of being mathematically rigorous. My internship with Ross Girshick, Bharath Hariharan and Piotr Dollár further inculcated in me the extreme value of rigorous experimentation and not discarding any hypothesis without concrete evidence. I also had a great time collaborating with Phillip Isola on our crazy, over a year-long project – during which I learned a lot from him.

I would like to also thank Pulkit Agrawal who grew from being my academic senior to a great friend over the years. We did many projects together, and brainstormed some of the coolest ideas – some made it in writing and some eventually will. I am thankful for his help on numerous accounts, including my job talk preparation. However, the biggest thing that I owe him for is a piece of valuable advice he gave me once - to listen to and get feedback from everyone but decide on your own. He taught me how not to get discouraged by criticism.

My memories of SDH7 will not be complete without Judy Hoffman, Ning Zhang, Jeff Donahue, Jon Long, Evan Shelhamer, Lisa Hendricks, Eric Tzeng, and Samaneh Azadi. Evan has this unique ability to come up with the most amazing titles and brewing the strongest coffee – two things I have bugged him a lot for – thanks for bearing with me! I am also thankful to Jun-Yan and Richard for being great friends and peers. Futile attempts at continuing to go to the gym with Jun-Yan, Richard and Tae are one of the most comical memories of my graduate life. Jun-Yan and I used to often stay until quite late in the lab and I thank him for dropping me home several times. We have spent countless hours discussing

research and brainstorming several other not-so-important topics. I am excited to have him as my colleague at CMU. I thank Shubham for always helping me debug my ideas and showing me corner cases which I would have missed otherwise. Thanks to Saurabh for being a great academic senior and entertaining my endless research questions. Thanks to Somil for teaching me controls.

Special thank you to all the postdocs - Jiashi Feng, Marcus Rohrbach, Sergio Guadarrama, Anja Rohrbach, João Carreira, Andrew Owens, Dinesh Jayaraman, Angjoo Kanazawa, and David Fouhey. Thank you, David, for adding humor to our lab and Angjoo for preventing the labs from being awfully quiet.

My time at UC Berkeley would not have been the same without my awesome academic and research peers. I feel privileged that I overlapped with one of the most amazing student cohorts at UC Berkeley. I am extremely thankful to Carl Doersch, Shiry Ginosar, Tinghui Zhou, Yang Gao, Taesung Park, Dequan Wang, Alan Jabri, Ashish Kumar, Eliza Kosoy, Rachit Dubey, Coline Devin, Ronghang Hu, Huazhe Xu, Jasmine, Sasha, Erin Grant, Avi Singh, Chelsea Finn, Vitchyr Pong and Kelvin Xu.

Finally, I would like to thank the amazing masters and undergraduate students whom I was fortunate to work with. I especially thank Parsa for being patient with me all these years. I am very lucky to have worked with Chris, Fred, Dian, Michael, Dhiraj, Pratyusha and Wenxuan.

Hanging out with Abhishek, Somil, Shubham, Tejas, Varun, and Vivek has often helped me not to get homesick. 1735 Cedar St. has been my defacto mailing address at Berkeley. I would like to also thank Weicheng Kuo and Ke Li - my first friends at UC Berkeley. My first few years would not have been so much fun if it did not involve attending classes, working on assignments, and having lunch with them – a very special mention to them for agreeing to countless visits to Urban Turban!

I am extremely grateful to Nikita for being on my side, keeping me sane and for being my core strength all these years – thank you for entertaining my numerous last-minute requests for proofreading! I would also like to thank my sister Divya for her love, support and for keeping me attached to the home. Finally, I would like to thank my parents Durga and Madhwa Nand Pathak. I would not have been fortunate enough to be the first engineer and first doctorate in our entire family tree had it not been for their support. I wonder if I can ever match the hard work they put in to provide myself and Divya with the opportunities that made us reach where we are today. I am always humbled by remembering the roots of their journey. I believe that they made progress worthy of two generations in one, by allowing me to match shoulders with the most modern and developed society. I dedicate this thesis to them for their uncountable sacrifices, boundless support, and unconditional love.

Thanks to Facebook, Nvidia, and Snapchat for awarding me their prestigious fellowships.

Chapter 1

Introduction

Consider an American college student embarking on her first trip abroad. She lands in Paris and, incredibly, she can still navigate the streets of this new city, spot cars and traffic signs, enter stores, etc., even though all of those look quite different in France than in the States. We as humans are naturally able to re-use our past experiences to adapt to novel scenarios. This ability to generalize makes us who we are. Generalization is the hallmark of human intelligence. While we have algorithms that excel at learning to translate languages, play complex games and even beat humans at it, they are miserable at generalization. For instance, a classification model that achieves super-human performance on the famous ImageNet dataset fails dramatically when deployed on a wearable camera in the real world. A robot that is an expert in the intricate task of putting a car engine together will fail at an apparently 'simple' task of putting plates into a dishwasher. Indeed, as noted by Moravec: the *hard* things are easy, and the easy things are *hard*.

1.1 Background

Since the inception of early ideas proposed by Alan Turing 70 years ago [246], AI has come a long way in building expert specialist systems. We have systems that can master the complex game of Go [226], classify images [95, 128], find objects in them [94], recognize speech [89], generate audio [168], translate languages [265], play table tennis [157] and perform many mundane industrial tasks. However, in contrast to humans, our current machine learning algorithms are incredibly narrow in performing the tasks they are trained for. A system that is champion in Go cannot translate languages. Needless to say, our current systems are missing the common-sense possessed by humans. However, the problem is more fundamental and rooted in the way we approach artificial intelligence.

Consider the following case study. Classification challenge on ImageNet [205] has led to the development of the methods that can classify images with superhuman accuracy (over 85%). This has been the flagship benchmark result and a crown jewel in the success of deep neural networks. However, these models completely fail to generalize when tested on

(a) Samples from ImageNet (b) Predictions on frames of a YouTube video

Figure 1.1: Sample outputs of ResNet-50 model trained on ImageNet when tested on frames of a video from YouTube. Left are representative samples of Labrador class from ImageNet, and right are the examples of false predictions generated by the model on video frames.

real-world videos, and the accuracy dramatically falls to less than one third than that in ImageNet. As shown in Figure 1.1, one such model, ResNet-50 [95], misclassifies Labrador to detect classes like Ray (fish), Cobra (snake) and Flatworm (also see [204] for examples). Why is that the case? ImageNet contains images that have objects at the center of the image, taken from a good angle with photographer bias – these conditions do not prevail in real videos which contain arbitrary changes in pose, motion blur, etc.

The reason is rather unsurprising, but questions a fundamental assumption in machine learning. Most of the ML approaches assume that the distribution at test time will be the same as seen during training. This could be true for passively collected datasets or simulated environments, but it does not hold in the real world. We cannot hope to capture all the possibilities, which one could ever encounter in the future, in a training dataset collected once ahead of time.

Interestingly, this issue is not specific to recent approaches, it rather dates back to the inception of AI. As the story goes, ARPA (now called DARPA) organized a challenge for classifying the presence of a tank in images in the early 1960s. They released a dataset of images with and without tanks. AI researchers at the time trained the classifiers (perhaps Perceptron) and achieved high accuracy on the dataset. However, the models failed to generalize to the tanks in the real world. It turns out that the non-tank images in the dataset were of forests in cloudy days, and tank images were captured in sunny days. Thus, instead of understanding tanks, the model found a shortcut by simplifying thresholding brightness intensities. Whether apocryphal or not [1], this cautionary tale very accurately captures a problem that is still unsolved by all means.

[1]The oldest reference we could find is Kanal and Randall, 1964 [115]. Although, the true version of this story is still debatable.

1.2 Current Dominant Paradigms for Generalization

One of the reasons for assuming the train and test distribution be same is that it is easy to handle mathematically. In contrast, the scenarios with changing the task or data distribution are difficult to define precisely. Indeed, if the new data or tasks the model is supposed to generalize to are too far from the training ones, there might not be any overlap of reusable knowledge to generalize. It has to be rather a smooth continuum of changing distribution where this notion of *similar enough* is really hard to capture mathematically. To scale our AI systems beyond the lab environments to real-world setups, our models must learn to generalize across changing data as well as task distributions. There have been several ways to try to address this challenge, however, they all have come up short as we discuss below:

1.2.1 Generalization by Data

One way to force generalization is to use lots of data. The hope is that, given huge labeled data, the model would have seen almost all possibilities such that generalization at test time reduces to just an interpolation problem. To concretize it, consider that we would like to train a function $f(x)$ that takes an input x and outputs label y from dataset $\mathcal{D}_{train} = \{x, y\}$. The argument is that as $|\mathcal{D}_{train}| \rightarrow \inf$, generalization would become trivial. This is of course practically impossible to do for all possible tasks, however, it is worth analyzing scientifically. For instance, in our tank example, it would mean that we need to collect images of tanks in rainy days, cloudy days, and all possible scenarios. Would that be enough? Unlikely because our world is continuously changing, and by the time we are done recording the current snapshots, the model of a tank would have changed. Hence, just scaling data and compute is implausible as a solution to building agents that can function in our ever-changing, real, complex world.

1.2.2 Generalization by Regularization/Design

One of the major reasons for failure to generalize is that the model ends up overfitting to spurious correlations instead of desired properties. The high-level goal in this paradigm is to restrict the space of possible instantiations of our function f such that unwanted correlations are avoided by construction. Classic approaches like L2/L1/elastic-regularizations [91] apply broadly to almost all statistical methods. The less obvious ones are the constraints that one could use by exploiting the domain structure. For instance, the structure of convolution has been immensely successful in neural networks for dealing with raw images [134]. Other examples include residual connections for training bigger networks [95], modularity in learned components to incorporate structure in language and images, etc. [8] etc. However, these constraints or structure help by reducing the probability of spurious correlations, but not enough. In our tank classification example, these design choices will help with overfitting but still won't be able to avoid the shortcut of modeling brightness.

1.2.3 Generalization by Meta/Transfer Learning

Instead of training the models to optimize for low error rate on the training set, one could directly optimize for generalization to unseen examples. The idea is captured by the family of algorithms called meta learning [18, 160, 214, 244] or transfer learning [33]. The standard approach of optimizing for best performance on the training set is rephrased to optimizing for a solution on training set such that it achieves good performance on some held-out data. Hence, instead of training on a single fixed training set, the models are trained on a family of tasks to generalize from one to another (see [65] for a detailed discussion). This paradigm is certainly optimizing for the correct behavior and perhaps the model will learn less spurious features. However, it still relies on a large amount of supervised labeled examples in the meta-training phase. More importantly, to optimize transfer across tasks/dataset directly, the tasks/datasets in the family has to be closely tied in practice. In such a case, the main challenge again reduces to the original core problem, i.e., how to generalize beyond the (close) family of tasks/datasets seen during training.

1.2.4 Generalization by Domain Adaptation

Domain adaption follows an alternative approach to generalizing the learned model to test distribution. The idea is to rather transform the data at test time such that its distribution becomes similar to training [27, 43, 96, 207]. The most common approach is to transform the data such that discrepancy is minimized between training and test data without explicitly requiring labels for test examples (see [101] for discussion). For instance, in the tank classification case, one could transform the tank images from rainy days, cloudy days, etc. to look like a tank in sunny days by employing some perceptual losses on the images. However, this exposes a critical issue that this transformation has to be learned for every new scenario encountered by the agent in the future because the model still simply classifies brightness and does not understand what a tank is. This is so because this paradigm sidesteps the core problem of learning understanding the true structure to generalize.

1.2.5 Generalization by End-to-End Reinforcement

End-to-end learning from scratch is one of the most popular paradigms in machine learning today. Instead of solving intermediate tasks in stages, the agent could directly solve for the end goal and learn necessary abstractions without much specific domain knowledge. A generic family of algorithms called reinforcement learning (RL) [241] encapsulate this approach. RL has emerged as a popular method for training agents to perform complex sensorimotor tasks. In RL, the agent's policy is trained by maximizing a reward function that is designed to align with the task. However, designing a well-shaped reward function is a notoriously challenging engineering problem, and such rewards are often absent in the real world. Hence, in practice, most of the success in RL has been achieved where this dense and well-shaped reward is easily available, e.g., a running 'score' in video games [155], or zero-sum games like chess or

Go [226]. Moreover, rewards are very specific to the task and environment that they are defined for, thus, generalizing to new tasks/scenarios is a serious concern in RL [98].

1.3 Proposed Solution

The ability to generalize to the novel, changing scenarios is essential for artificial agents to function in real and diverse environments. As discussed above, the current dominant paradigm of supervised learning, which relies on training with human-labeled examples, is unlikely to achieve this goal. Having a human provide lots of labels for every possible task that an agent could ever encounter is like building a ladder to the moon — there might never be enough labels! Indeed, supervised learning is not how most learning transpires in ecological contexts. Instead, all that is available to a biological agent is just raw sensory data and the ability to act in the world to collect more data. More importantly, it could be that the mere presence of a fixed goal (conveyed by human-labeled examples or rewards) tempts our algorithms to cheat by modeling spurious correlations.

What if we remove this extrinsic goal? In the absence of a goal, our agent can no longer 'cheat' as there would be nothing to find a 'shortcut' for. This is not as strange as it sounds. Developmental psychologists talk about *intrinsic motivation* (i.e., curiosity) as the primary driver in the early stages of development [83, 206, 230]: babies appear to employ (extrinsic) goal-free exploration to learn skills that will be useful later on in life, for instance, throwing, pushing objects and interacting with the world in seemingly random ways. In fact, this form of learning is not specific to humans. The example, given by Alison Gopnik [82], of the contrast between domestic chicken and New Caledonian crow, precisely illustrates the crucial role of such intrinsic exploration phase in shaping the intelligence of species. Domestic chicken (also, ducks, geese, and turkeys) are often considered as the dumbest of all birds. They are very good at the task of pecking grains, but pretty much useless otherwise. In contrast, a New Caledonian crow is one of the intelligent bird species, and can even figure out the use of a tool by shaping hooks to retrieve food [256]. Despite having similar cortical structure, chickens are specialists while these crows are impressive generalists. Gopnik argues that it is connected to the length of their childhood period of exploration without any end-goals. While chickens become mature and functional in just a few months, the baby New Caledonian crows depend on their caretaker to feed and nurture them up to 2 years, which is a long time in the lifespan of a bird. This link between long childhood and intelligence of a species has been studied in depth by Piantadosi and Kidd, 2016 (see Figure 3 in [189]). Across several primate species, the length of the weaning period seemed to be strongly correlated with how intelligent the species is. One crucial aspect of this seemingly frivolous 'play' is that it allows these biological agents to learn how to continually adapt and increase knowledge about the world without any explicit supervision or extrinsic end-goal.

1.3.1 Learning to Generalize via Self-Supervised Prediction

In this dissertation, inspired by these prominent ideas, we propose initial directions towards the grand question of building artificial agents that learn, act and display seamless general-purpose behavior. Our key emphasis is on building agents that continually develop knowledge and acquire skills just from their own collected data by using *data as its own supervision*. We begin with minimal assumptions and build complete systems that learn from raw sensory data. To be able to learn from scratch, the claim is that an artificial agent must be embodied in the world, develop an understanding of its sensory input (e.g., image stream) and simultaneously learn to map this understanding to its motor outputs (e.g., torques) in an *unsupervised* manner.

All these considerations lead to two fundamental questions: how to learn rich representations of the world similar to *what* humans learn?; and how to re-use such a representation of past knowledge to incrementally adapt and learn more about the world similar to *how* humans do? We believe *prediction* is the key to this answer. We propose generic mechanisms that employ prediction as a self-supervisory signal in allowing the agents to learn sensory representations as well as act on these representations to govern motor control. These two abilities should equip an embodied agent with a basic set of general-purpose skills. Later, these skills can be stitched together to achieve end-goals or tasks given to the agent, by planning using the learned models with little to no supervision from outside. This thesis outlines this ideology in detail and concretizes these ideas into algorithms which are demonstrated via several case studies, ranging from passive datasets, games to real robotics setups, as summarized below.

I. Self-Supervised Representation Learning Our world is diverse, yet highly structured, and humans have an uncanny ability to make sense of it. This ability to build rich representations of raw sensory data is indispensable for agents learning to operate on their own. But how is this structure discovered in the first place? In Part I, we propose two approaches to learn sensory representations. Chapter 2 proposes Context Encoders which employ prediction of context as supervision to learn visual features. Then in Chapter 3, we extend this ideology to learn representation by leveraging the Gestalt principle of common-fate [257], i.e., pixels that *move* together tend to belong together, which explicitly guides the perception of object boundaries. These groupings are either obtained by watching passive videos, or by active interaction of a robot.

II. Learning to Act via Self-Supervised Exploration Learning to see the world the way we do is only the first step in building a self-supervised autonomous agent. An agent will be able to adapt and acquire increasingly complex behaviors only when it learns to use its sensory representation to act. However, acting in the world is unlike passive observations because the data is sequentially *dependent* on the past which leads to exponentially many possible trajectories and makes it intractable to figure out the ones that constitute useful skills. In Chapter 4, we leverage self-supervised prediction to formulate a notion of *curiosity*

in artificial agents that allows learning to act without any extrinsic supervision or rewards. Chapter 5 extends this formulation into an explicitly differentiable objective such that it can be efficiently scaled to real robots.

III. From Skills to Goal-Directed Expertise Parts I and II discuss how an agent could acquire sensorimotor skills of increasing complexity with no knowledge of labels or tasks. However, to be useful for real-world tasks, the agent will eventually have to develop the ability to perform a specific task given to it. One way is to finetune a pre-trained self-supervised agent with task-specific rewards. However, reward-based finetuning is inefficient and not scalable to real robots. In Chapter 6, we follow an alternative approach, where an agent first explores the environment without any expert supervision and distills this exploration data into goal-directed models. Then, an expert communicates only *what* needs to be done by providing a goal image while the agent infers *how* to perform the task by itself.

IV. Generalization via Modularity Above chapters focus on learning sensorimotor skills for artificial agents, but the agent itself always starts with an already complex physical body (e.g., a robotic arm). It is hard to generalize when the agent is already so complex. If the agent could itself (*hardware*) be composed of modular reusable components, it would become easier for the learned controller (*software*) to generalize. Indeed, one could argue that it is this modular design which allowed the multicellular organisms to successfully adapt, and generalize to the constantly changing environment of prehistoric Earth [6]. In Chapter 7, we take inspiration from multicellular evolution to study modularity as a model for emergent complexity in artificial agents. However, unlike most previous works, we see *modularity as a way of improving generalization* to unseen scenarios.

Finally, Chapter 8 concludes by summarizing the proposed ideology, algorithms, results, and describes some of the future directions that immediately follow from this agenda.

Part I

Self-Supervised Representation Learning

Chapter 2

Learning Representation via Context Prediction

Our visual world is very diverse, yet highly structured, and humans have an uncanny ability to make sense of this structure. In this chapter, we explore whether state-of-the-art computer vision algorithms can do the same. Consider the image shown in Figure 2.1a. Although the center part of the image is missing, most of us can easily imagine its content from the surrounding pixels, without having ever seen that exact scene. Some of us can even draw it, as shown on Figure 2.1b. This ability comes from the fact that natural images, despite their diversity, are highly structured (e.g. the regular pattern of windows on the facade). We humans are able to understand this structure and make visual predictions even when seeing only parts of the scene. We show that it is possible to learn and predict this structure using convolutional neural networks (CNNs), a class of models that have shown success across a variety of image understanding tasks.

Given an image with a missing region (e.g., Fig. 2.1a), we train a convolutional neural network to regress to the missing pixel values (Fig. 2.1d). We call our model *context encoder*, as it consists of an encoder capturing the context of an image into a compact latent feature representation and a decoder which uses that representation to produce the missing image content. The context encoder is closely related to autoencoders [19, 99], as it shares a similar encoder-decoder architecture. Autoencoders take an input image and try to reconstruct it after it passes through a low-dimensional "bottleneck" layer, with the aim of obtaining a compact feature representation of the scene. Unfortunately, this feature representation is likely to just compresses the image content without learning a semantically meaningful representation. Denoising autoencoders [248] address this issue by corrupting the input image and requiring the network to undo the damage. However, this corruption process is typically very localized and low-level, and does not require much semantic information to undo. In contrast, our context encoder needs to solve a much harder task: to fill in large missing areas

This chapter is based on the paper published previously at CVPR 2016 [184].

(a) Input context (b) Human artist (c) CE ($L2$) (d) CE ($L2$ + adv.)

Figure 2.1: Qualitative illustration of the task. Given an image with a missing region (a), a human artist has no trouble inpainting it (b). Automatic inpainting using our *context encoder* trained with $L2$ reconstruction loss is shown in (c), and using both $L2$ and adversarial losses in (d).

of the image, where it can't get "hints" from nearby pixels. This requires a much deeper semantic understanding of the scene, and the ability to synthesize high-level features over large spatial extents. For example, in Figure 2.1a, an entire window needs to be conjured up "out of thin air." This is similar in spirit to word2vec [151] which learns word representation from natural language sentences by predicting a word given its context.

Like autoencoders, context encoders are trained in a completely unsupervised manner. Our results demonstrate that in order to succeed at this task, a model needs to both understand the content of an image, as well as produce a plausible hypothesis for the missing parts. This task, however, is inherently multi-modal as there are multiple ways to fill the missing region while also maintaining coherence with the given context. We decouple this burden in our loss function by jointly training our context encoders to minimize both a reconstruction loss and an adversarial loss. The reconstruction (L2) loss captures the overall structure of the missing region in relation to the context, while the the adversarial loss [81] has the effect of picking a particular mode from the distribution. Figure 2.1 shows that using only the reconstruction loss produces blurry results, whereas adding the adversarial loss results in much sharper predictions.

We evaluate the encoder and the decoder independently. On the encoder side, we show that encoding just the context of an image patch and using the resulting feature to retrieve nearest neighbor contexts from a dataset produces patches which are semantically similar to the original (unseen) patch. We further validate the quality of the learned feature representation by fine-tuning the encoder for a variety of image understanding tasks, including classification, object detection, and semantic segmentation. We are competitive with the state-of-the-art unsupervised/self-supervised methods on those tasks. On the decoder side, we show that our method is often able to fill in realistic image content. Indeed, to the best of our knowledge, ours is the first parametric inpainting algorithm that is able to give reasonable results for semantic hole-filling (i.e. large missing regions). The context encoder can also be useful as a better visual feature for computing nearest neighbors in non-parametric inpainting methods.

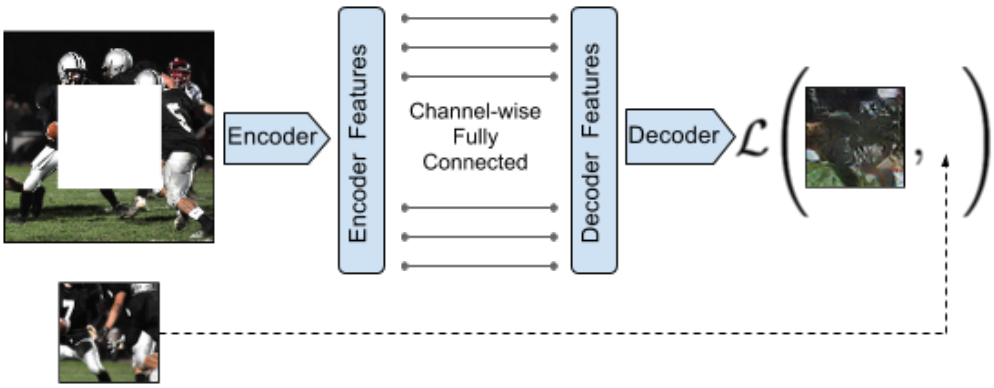


Figure 2.2: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer as described in Section 2.1.1. The decoder then produces the missing regions in the image.

2.1 Context encoders for image generation

We now introduce context encoders: CNNs that predict missing parts of a scene from their surroundings. We first give an overview of the general architecture, then provide details on the learning procedure and finally present various strategies for image region removal.

2.1.1 Encoder-decoder pipeline

The overall architecture is a simple encoder-decoder pipeline. The encoder takes an input image with missing regions and produces a latent feature representation of that image. The decoder takes this feature representation and produces the missing image content. We found it important to connect the encoder and the decoder through a channel-wise fully-connected layer, which allows each unit in the decoder to reason about the entire image content. Figure 2.2 shows an overview of our architecture.

Encoder Our encoder is derived from the *AlexNet* architecture [128]. Given an input image of size 227×227, we use the first five convolutional layers and the following pooling layer (called *pool5*) to compute an abstract $6 \times 6 \times 256$ dimensional feature representation. In contrast to *AlexNet*, our model is not trained for ImageNet classification; rather, the network is trained for context prediction "from scratch" with randomly initialized weights.

However, if the encoder architecture is limited only to convolutional layers, there is no way for information to directly propagate from one corner of the feature map to another. This is so because convolutional layers connect all the feature maps together, but never directly connect all locations within a specific feature map. In the present architectures, this information propagation is handled by *fully-connected* or *inner product* layers, where all the activations are directly connected to each other. In our architecture, the latent feature dimension is $6 \times 6 \times 256 = 9216$ for both encoder and decoder. This is so because, unlike

Figure 2.3: Semantic Inpainting results for context encoder trained jointly using reconstruction and adversarial loss. First four rows contain examples from Paris StreetView Dataset, and bottom row contains examples from ImageNet.

autoencoders, we do not reconstruct the original input and hence need not have a smaller *bottleneck*. However, fully connecting the encoder and decoder would result in an explosion in the number of parameters (over 100M!), to the extent that efficient training on current GPUs would be difficult. To alleviate this issue, we use a channel-wise fully-connected layer to connect the encoder features to the decoder, described in detail below.

Channel-wise fully-connected layer This layer is essentially a fully-connected layer with groups, intended to propagate information within activations of each feature map. If the input layer has m feature maps of size $n \times n$, this layer will output m feature maps of dimension $n \times n$. However, unlike a fully-connected layer, it has no parameters connecting different feature maps and only propagates information within feature maps. Thus, the number of parameters in this channel-wise fully-connected layer is mn^4, compared to m^2n^4 parameters in a fully-connected layer (ignoring the bias term). This is followed by a stride 1 convolution to propagate information across channels.

Decoder We now discuss the second half of our pipeline, the decoder, which generates pixels of the image using the encoder features. The "encoder features" are connected to the "decoder features" using a channel-wise fully-connected layer.

The channel-wise fully-connected layer is followed by a series of five *up-convolutional* layers [56, 145, 274] with learned filters, each with a rectified linear unit (ReLU) activation function. A up-convolutional is simply a convolution that results in a higher resolution image. It can be understood as upsampling followed by convolution (as described in [56]), or convolution with fractional stride (as described in [145]). The intuition behind this is straightforward – the series of up-convolutions and non-linearities comprises a non-linear weighted upsampling of the feature produced by the encoder until we roughly reach the original target size.

2.1.2 Loss function

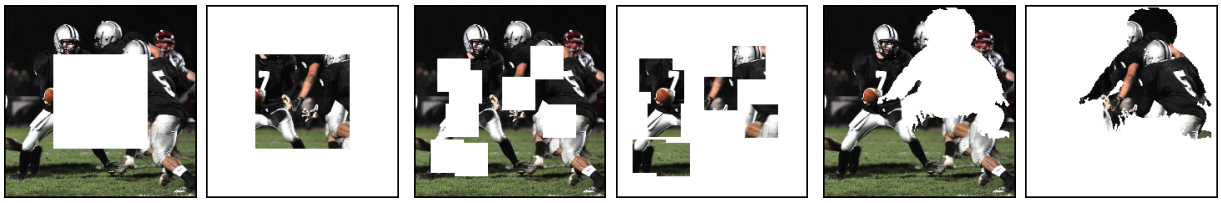
We train our context encoders by regressing to the ground truth content of the missing (dropped out) region. However, there are often multiple equally plausible ways to fill a missing image region which are consistent with the context. We model this behavior by having a decoupled joint loss function to handle both continuity within the context and multiple modes in the output. The reconstruction (L2) loss is responsible for capturing the overall structure of the missing region and coherence with regards to its context, but tends to average together the multiple modes in predictions. The adversarial loss [81], on the other hand, tries to make prediction look real, and has the effect of picking a particular mode from the distribution. For each ground truth image x, our context encoder F produces an output $F(x)$. Let $\hat{M}$ be a binary mask corresponding to the dropped image region with a value of 1 wherever a pixel was dropped and 0 for input pixels. During training, those masks are automatically generated for each image and training iterations, as described in Section 2.1.2. We now describe different components of our loss function.

Reconstruction Loss We use a masked $L2$ distance as our reconstruction loss, $\mathcal{L}_{rec}$,

$$\mathcal{L}_{rec}(x) = \|\hat{M} \odot (x - F((1 - \hat{M}) \odot x))\|_2, \tag{2.1}$$

where $\odot$ is the element-wise product operation. We experimented with both L1 and L2 losses and found no significant difference between them. While this simple loss encourages the decoder to produce a rough outline of the predicted object, it often fails to capture any high frequency detail (see Fig. 2.1c). This stems from the fact that the L2 (or L1) loss often prefer a blurry solution, over highly accurate textures. We believe this happens because it is much "safer" for the L2 loss to predict the mean of the distribution, because this minimizes the mean pixel-wise error, but results in a blurry averaged image. We alleviated this problem by adding an adversarial loss.

Adversarial Loss Our adversarial loss is based on Generative Adversarial Networks (GAN) [81]. To learn a generative model G of a data distribution, GAN proposes to



(a) Central region (b) Random block (c) Random region

Figure 2.4: An example of image x with our different region masks $\hat{M}$ applied to it.

jointly learn an adversarial discriminative model D to provide loss gradients to the generative model. G and D are parametric functions (e.g., deep networks) where $G : \mathcal{Z} \rightarrow \mathcal{X}$ maps samples from noise distribution $\mathcal{Z}$ to data distribution $\mathcal{X}$. The learning procedure is a two-player game where an adversarial discriminator D takes in both the prediction of G and ground truth samples, and tries to distinguish them, while G tries to confuse D by producing samples that appear as "real" as possible. The objective for discriminator is logistic likelihood indicating whether the input is real sample or predicted one:

$$\min_G \max_D \quad \mathbb{E}_{x\in\mathcal{X}}[\log(D(x))] + \mathbb{E}_{z\in\mathcal{Z}}[\log(1 - D(G(z)))]$$

This method has shown encouraging results in generative modeling of images [200]. We thus adapt this framework for context prediction by modeling generator by context encoder; i.e., $G \triangleq F$. To customize GANs for this task, one could condition on the given context information; i.e., the mask $\hat{M} \odot x$. However, conditional GANs don't train easily for context prediction task as the adversarial discriminator D easily exploits the perceptual discontinuity in generated regions and the original context to easily classify predicted versus real samples. We thus use an alternate formulation, by conditioning only the generator (not the discriminator) on context. We also found results improved when the generator was not conditioned on a noise vector. The GAN objective for our context encoders is as follows Hence the adversarial loss for context encoders, $\mathcal{L}_{adv}$, is

$$\begin{aligned}\mathcal{L}_{adv} = \max_D \quad & \mathbb{E}_{x\in\mathcal{X}}[\log(D(x)) \\ & + \log(1 - D(F((1 - \hat{M}) \odot x)))],\end{aligned} \tag{2.2}$$

where, in practice, both F and D are optimized jointly using alternating SGD. Note that this objective encourages the entire output of the context encoder to look realistic, not just the missing regions as in Equation (2.1).

Joint Loss We define the overall loss function as

$$\mathcal{L} = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}_{adv}. \tag{2.3}$$

Currently, we use adversarial loss only for inpainting experiments as AlexNet [128] architecture training diverged with joint adversarial loss. Details follow in Sections 2.3.1, 2.3.2.

Figure 2.5: Comparison with Content-Aware Fill (Photoshop feature based on [15]). Our method works better in semantic cases (top row) and works slightly worse in textured settings (bottom row).

Region masks

The input to a context encoder is an image with one or more of its regions "dropped out"; i.e., set to zero, assuming zero-centered inputs. The removed regions could be of any shape, we present three different strategies here:

Central region: The simplest such shape is the central square patch in the image, as shown in Figure 2.4a. While this works quite well for inpainting, the network learns low level image features than latch on to the boundary of the central mask. Those low level image features tend not to generalize well to images without masks, hence the features learned are not very general.

Random block: To prevent the network from latching on the the constant boundary of the masked region, we randomize the masking process. Instead of choosing a single large mask at a fixed location, we remove a number of smaller possibly overlapping masks, covering up to $\frac{1}{4}$ of the image. An example of this is shown in Figure 2.4b. However, the random block masking still has sharp boundaries convolutional features could latch onto.

Random region: To completely remove those boundaries, we experimented with removing arbitrary shapes from images, obtained from random masks in the PASCAL VOC 2012 dataset [62]. We deform those shapes and paste in arbitrary places in the other images (not from PASCAL), again covering up to $\frac{1}{4}$ of the image. Note that we completely randomize the region masking process, and do not expect or want any correlation between the source segmentation mask and the image. We merely use those regions to prevent the network from learning low-level features corresponding to the removed mask. See example in Figure 2.4c.

In practice, we found region and random block masks produce a similarly general feature, while significantly outperforming the central region features. We use the random region dropout for all our feature based experiments.

2.2 Implementation details

The pipeline was implemented in *Caffe* [112] and Torch. We used ADAM [119] for optimization. The missing region in the masked input image is filled with constant mean value. Hyper-parameter details are discussed in Sections 2.3.1, 2.3.2. We experimented with replacing all pooling layers with convolutions of the same kernel size and stride. The overall stride of the

Figure 2.6: Semantic Inpainting using different methods. Context Encoder with just L2 are well aligned, but not sharp. Using adversarial loss, results are sharp but not coherent. Joint loss alleviate the weaknesses of each of them. The last two columns are the results if we plug-in the best nearest neighbor (NN) patch in the masked region.

Method	Mean L1 Loss	Mean L2 Loss	PSNR (higher better)
NN-inpainting (HOG features)	19.92%	6.92%	12.79 dB
NN-inpainting (our features)	15.10%	4.30%	14.70 dB
Our Reconstruction (joint)	**10.33%**	**2.35%**	**17.59 dB**

Table 2.1: Semantic Inpainting accuracy for Paris StreetView dataset on held-out images. NN inpainting is basis for [93].

network remains the same, but it results in finer inpainting. Intuitively, there is no reason to use pooling for reconstruction based networks. In classification, pooling provides spatial invariance, which may be detrimental for reconstruction-based training. To be consistent with prior work, we still use the original AlexNet architecture (with pooling) for all feature learning results.

2.3 Evaluation

We now evaluate the encoder features for their semantic quality and transferability to other image understanding tasks. We experiment with images from two datasets: Paris StreetView [52] and ImageNet [205] without using any of the accompanying labels. In Section 2.3.1, we present visualizations demonstrating the ability of the context encoder to fill in semantic details of images with missing regions. In Section 2.3.2, we demonstrate the transferability of our learned features to other tasks, using context encoders as a pre-training step for image classification, object detection, and semantic segmentation. We compare our results on these tasks with those of other unsupervised or self-supervised methods, demonstrating that our approach outperforms previous methods.

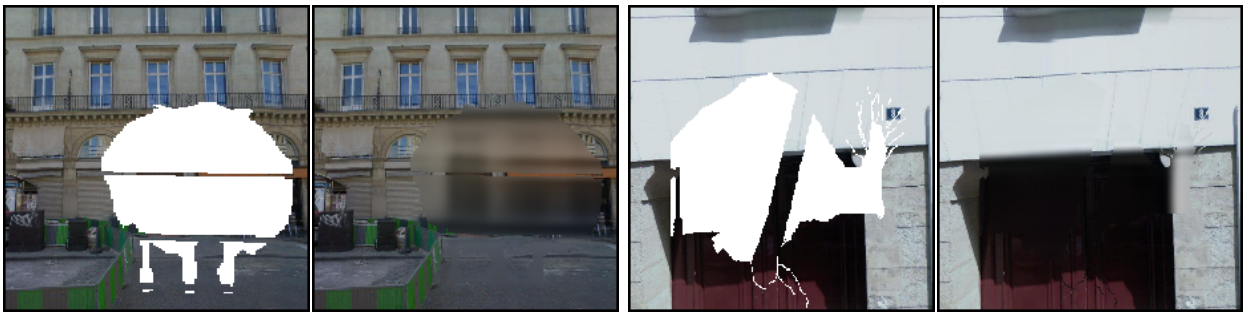


Figure 2.7: Arbitrary region inpainting for context encoder trained with reconstruction loss.

2.3.1 Semantic Inpainting

We train context encoders with the joint loss function defined in Equation (2.3) for the task of inpainting the missing region. The encoder and discriminator architecture is similar to that of discriminator in [200], and decoder is similar to generator in [200]. However, the bottleneck is of 4000 units (in contrast to 100 in [200]). We used the default solver hyper-parameters suggested in [200]. We use $\lambda_{rec} = 0.999$ and $\lambda_{adv} = 0.001$. However, a few things were crucial for training the model. We did not condition the adversarial loss (see Section 2.1.2) nor did we add noise to the encoder. We use a higher learning rate for context encoder (10 times) to that of adversarial discriminator. To further emphasize the consistency of prediction with the context, we predict a slightly larger patch that overlaps with the context (by 7px). During training, we use higher weight (10×) for the reconstruction loss in this overlapping region.

The qualitative results are shown in Figure 2.3. Our model performs generally well in inpainting semantic regions of an image. However, if a region can be filled with low-level textures, texture synthesis methods, such as [15,61], can often perform better (e.g. Figure 2.5). For semantic inpainting, we compare against nearest neighbor inpainting (which forms the basis of Hays *et al.* [93]) and show that our reconstructions are well-aligned semantically, as seen on Figure 2.6. It also shows that joint loss significantly improves the inpainting over both reconstruction and adversarial loss alone. Moreover, using our learned features in a nearest-neighbor style inpainting can sometimes improve results over a hand-designed distance metrics. Table 2.1 reports quantitative results on StreetView Dataset.

2.3.2 Feature Learning

For consistency with prior work, we use the AlexNet [128] architecture for our encoder. Unfortunately, we did not manage to make the adversarial loss converge with AlexNet, so we used just the reconstruction loss. The networks were trained with a constant learning rate of 10^{-3} for the center-region masks. However, for random region corruption, we found a learning rate of 10^{-4} to perform better. We apply dropout with a rate of 0.5 just for the channel-wise fully connected layer, since it has more parameters than other layers and might be prone to overfitting. The training process is fast and converges in about 100K iterations:

Figure 2.8: Context Nearest Neighbors. Center patches whose context (not shown here) are close in the embedding space of different methods (namely our context encoder, HOG and AlexNet). Note that the appearance of these center patches themselves was never seen by these methods. But our method brings them close just from their context.

14 hours on a Titan X GPU. Figure 2.7 shows inpainting results for context encoder trained with random region corruption using reconstruction loss. To evaluate the quality of features, we find nearest neighbors to the masked part of image just by using the features from the context, see Figure 2.8. Note that none of the methods ever see the center part of any image, whether a query or dataset image. Our features retrieve decent nearest neighbors just from context, even though actual prediction is blurry with L2 loss. AlexNet features also perform decently as they were trained with 1M labels for semantic tasks, HOG on the other hand fail to get the semantics.

Classification pre-training

For this experiment, we fine-tune a standard AlexNet classifier on the PASCAL VOC 2007 [62] from a number of supervised, self-supervised and unsupervised initializations. We train the classifier using random cropping, and then evaluate it using 10 random crops per test image. We average the classifier output over those random crops. Table 2.2 shows the standard mean average precision (mAP) score for all compared methods. A random initialization performs roughly 25% below an ImageNen-trained model; however, it does not use any labels. Context encoders are competitive with concurrent self-supervised feature learning methods [51, 254] and significantly outperform autoencoders and Agrawal *et al.* [4].

Detection pre-training

Our second set of quantitative results involves using our features for object detection. We use *Fast R-CNN* [80] framework (FRCN). We replace the ImageNet pre-trained network with our context encoders (or any other baseline model). In particular, we take the pre-trained encoder weights up to the *pool5* layer and re-initialize the fully-connected layers. We then follow the training and evaluation procedures from FRCN and report the accuracy (in mAP) of the resulting detector.

Pretraining Method	Supervision	Pretraining time	Classification	Detection	Segmentation
ImageNet [128]	1000 class labels	3 days	**78.2**%	**56.8**%	**48.0**%
Random Gaussian	initialization	< 1 minute	53.3%	43.4%	19.8%
Autoencoder	-	14 hours	53.8%	41.9%	25.2%
Agrawal *et al.* [4]	egomotion	10 hours	52.9%	41.8%	-
Doersch *et al.* [51]	context	4 weeks	55.3%	**46.6**%	-
Wang *et al.* [254]	motion	1 week	**58.4**%	44.0%	-
Ours	context	14 hours	56.5%	44.5%	**29.7**%

Table 2.2: Quantitative comparison for classification, detection and semantic segmentation. Classification and Fast-RCNN Detection results are on the PASCAL VOC 2007 test set. Semantic segmentation results are on the PASCAL VOC 2012 validation set from the FCN evaluation described in Section 2.3.2, using the additional training data from [90], and removing overlapping images from the validation set [145].

Our results on the test set of the PASCAL VOC 2007 [62] detection challenge are reported in Table 2.2. Context encoder pre-training is competitive with the existing methods achieving significant boost over the baseline. Krähenbühl *et al.* [126] proposed a data-dependent method for rescaling pre-trained model weights. This significantly improves the features in Doersch *et al.* [51] up to 65.3% for classification and 51.1% for detection. However, this rescaling doesn't improve results for other methods, including ours.

Semantic Segmentation pre-training

Our last quantitative evaluation explores the utility of context encoder training for pixel-wise semantic segmentation. *Fully convolutional networks* [145] (FCNs) were proposed as an end-to-end learnable method of predicting a semantic label at each pixel of an image, using a convolutional network pre-trained for ImageNet classification. We replace the classification pre-trained network used in the FCN method with our context encoders, afterwards following the FCN training and evaluation procedure for direct comparison with their original *CaffeNet*-based result.

Our results on the PASCAL VOC 2012 [62] validation set are reported in Table 2.2. In this setting, we outperform a randomly initialized network as well as a plain autoencoder which is trained simply to reconstruct its full input.

2.4 Related work

CNNs trained for ImageNet [205] classification with over a million labeled examples learn features which generalize very well across tasks [53]. However, whether such semantically informative and generalizable features can be learned from raw images alone, without any labels, remains an open question. Unsupervised learning is a broad area with a large volume

of work; Bengio *et al*. [20] provide an excellent survey. Here, we briefly revisit some of the recent work in this area.

Self-supervision via pretext tasks Instead of producing images, several recent studies have focused on providing alternate forms of supervision (often called 'pretext tasks') that do not require manual labeling and can be algorithmically produced. For instance, Doersch *et al*. [51] task a ConvNet with predicting the relative location of two cropped image patches. Noroozi and Favaro [164] extend this by asking a network to arrange shuffled patches cropped from a 3×3 grid. Other pretext tasks include predicting color channels from luminance [132, 276] or vice versa [277], and predicting sounds from video frames [46, 175]. The assumption in these works is that to perform these tasks, the network will need to recognize high-level concepts, such as objects, in order to succeed.

Most closely related to our idea are efforts at exploiting spatial context as a source of free and plentiful supervisory signal. Visual Memex [149] used context to non-parametrically model object relations and to predict masked objects in scenes, while [50] used context to establish correspondences for unsupervised object discovery. However, both approaches relied on hand-designed features and did not perform any representation learning. Doersch *et al*. [51] used the task of predicting the relative positions of neighboring patches within an image as a way to train an unsupervised deep feature representations. We share the same high-level goals with Doersch *et al*. but fundamentally differ in the approach: whereas [51] are solving a *discriminative* task (is patch A above patch B or below?), our context encoder solves a pure *prediction* problem (what pixel intensities should go in the hole?). Interestingly, similar distinction exist in using language context to learn word embeddings: Collobert and Weston [38] advocate a discriminative approach, whereas word2vec [151] formulate it as word prediction. One important benefit of our approach is that our supervisory signal is much richer: a context encoder needs to predict roughly 15,000 real values per training example, compared to just 1 option among 8 choices in [51]. Likely due in part to this difference, our context encoders take far less time to train than [51]. Moreover, context based prediction is also harder to "cheat" since low-level image features, such as chromatic aberration, do not provide any meaningful information, in contrast to [51] where chromatic aberration partially solves the task. On the other hand, it is not yet clear if requiring faithful pixel generation is necessary for learning good visual features.

Unsupervised learning by generating images Classical unsupervised representation learning approaches, such as autoencoders [19, 99] and denoising autoencoders [248], attempt to learn feature representations from which the original image can be decoded with a low error. An alternative to reconstruction-based objectives is to train generative models of images using generative adversarial networks [81]. These models can be extended to produce good feature representations by training jointly with image encoders [54, 59]. However, to generate realistic images, these models must pay significant attention to low-level details while potentially ignoring higher-level semantics. We train our context encoders using an

adversary jointly with reconstruction loss for generating inpainting results. We discuss this in detail in Section 2.1.2.

Dosovitskiy *et al*. [56] and Rifai *et al*. [202] demonstrate that CNNs can learn to generate novel images of particular object categories (chairs and faces, respectively), but rely on large labeled datasets with examples of these categories. In contrast, context encoders can be applied to any unlabeled image database and learn to generate images based on the surrounding context.

Inpainting and hole-filling It is important to point out that our hole-filling task cannot be handled by classical inpainting [22, 170] or texture synthesis [15, 61] approaches, since the missing region is too large for local non-semantic methods to work well. In computer graphics, filling in large holes is typically done via scene completion [93], involving a cut-paste formulation using nearest neighbors from a dataset of millions of images. However, scene completion is meant for filling in holes left by removing whole objects, and it struggles to fill arbitrary holes, e.g. amodal completion of partially occluded objects. Furthermore, previous completion relies on a hand-crafted distance metric, such as Gist [167] for nearest-neighbor computation which is inferior to a learned distance metric. We show that our method is often able to inpaint semantically meaningful content in a parametric fashion, as well as provide a better feature for nearest neighbor-based inpainting methods.

Chapter 3

Discovering Objects by Observation and Interaction

In previous chapter, we presented an unsupervised visual feature learning algorithm driven by context-based pixel prediction. As apparent in the related works discussed, a recurring theme in most self-supervised representation learning works is the idea of a 'pretext task': a task that is not of direct interest, but can be used to obtain a good visual representation as a byproduct of training. The challenge in this line of research lies in cleverly designing a pretext task that causes the ConvNet (or other representation learner) to learn high-level features.

In this chapter, we take a different approach that is motivated by human vision studies. Both infants [231] and newly sighted congenitally blind people [172] tend to *oversegment* static objects, but can group things properly when they *move* (Figure 3.1). To do so, they may rely on the Gestalt principle of common fate [177, 258]: pixels that move together tend to belong together. The ability to parse static scenes improves [172] over time, suggesting that while motion-based grouping appears early, static grouping is acquired later, possibly bootstrapped by motion cues. Moreover, experiments in [172] show that shortly after gaining sight, human subjects are better able to name objects that tend to be seen in motion compared to objects that tend to be seen at rest.

Inspired by these human vision studies, we propose to train ConvNets for the well-established task of object foreground *vs*. background segmentation, using unsupervised motion segmentation to provide 'pseudo ground truth'. Concretely, to prepare training data we use optical flow to group foreground pixels that move together into a single object. We then use the resulting segmentation masks as automatically generated targets, and task a ConvNet with predicting these masks from *single, static frames without any motion information* (Figure 3.2). Because pixels with different colors or low-level image statistics

This chapter is based on the papers published previously at CVPR 2017 [181] and CVPR Robotics Vision Workshop 2018 [187].

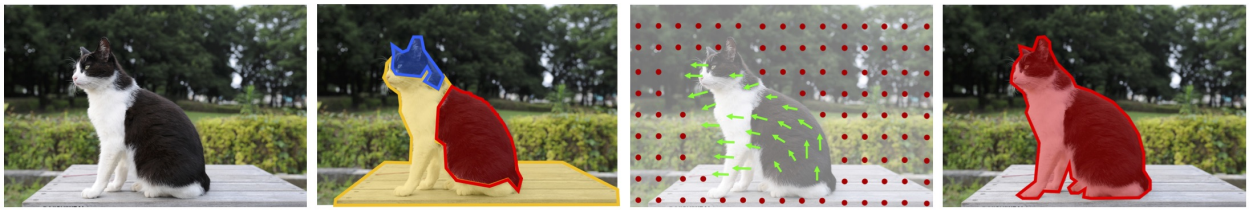


Figure 3.1: Low-level appearance cues lead to incorrect grouping (mid-left). Motion helps us to correctly group pixels that move together (mid-right) and identify this group as a single object (rightmost). We use unsupervised motion-based grouping to train a ConvNet to segment objects in *static images* and show that the network learns strong features that transfer well to other tasks.

can still move together and form a single object, the ConvNet cannot solve this task using a low-level representation. Instead, it may have to *recognize* objects that tend to move and identify their shape and pose. Thus, we conjecture that this task forces the ConvNet to learn a high-level representation.

We evaluate our proposal in two settings. First, we test if a ConvNet can learn a good feature representation when learning to segment from the high-quality, manually labeled segmentations in COCO [142], without using the class labels. Indeed, we show that the resulting feature representation is effective when transferred to PASCAL VOC object detection. It achieves state-of-the-art performance for representations trained without any semantic category labels, performing within 5 points AP of an ImageNet pretrained model and 10 points higher than the best unsupervised methods. This justifies our proposed task by showing that *given good ground truth segmentations, a ConvNet trained to segment objects will learn an effective feature representation.*

Our goal, however, is to learn features *without manual supervision.* Thus in our second setting we train with *automatically generated* 'pseudo ground truth' obtained through unsupervised motion segmentation on uncurated videos from the Yahoo Flickr Creative Commons 100 million (YFCC100m) [243] dataset. When transferred to object detection, our representation retains good performance even when most of the ConvNet parameters are frozen, significantly outperforming previous unsupervised learning approaches. It also allows much better transfer learning when training data for the target task is scarce. Our representation quality tends to increase logarithmically with the amount of data, suggesting the possibility of outperforming ImageNet pretraining given the countless videos on the web.

However, one does not have to wait passively for things to move as motion can also be caused by active interaction. While passive motion helps in learning about object boundaries, Smith and Gasser [230] argue that it is the "active interaction" that makes infants learn about the properties of individual object entities. We put this developmental hypothesis to the test in the final section of this chapter. We leverage the same idea of predicting motion segments, but instead of observing passive videos, we deploy an autonomous robotic agent that collects examples by actively interacting with objects in its arena.

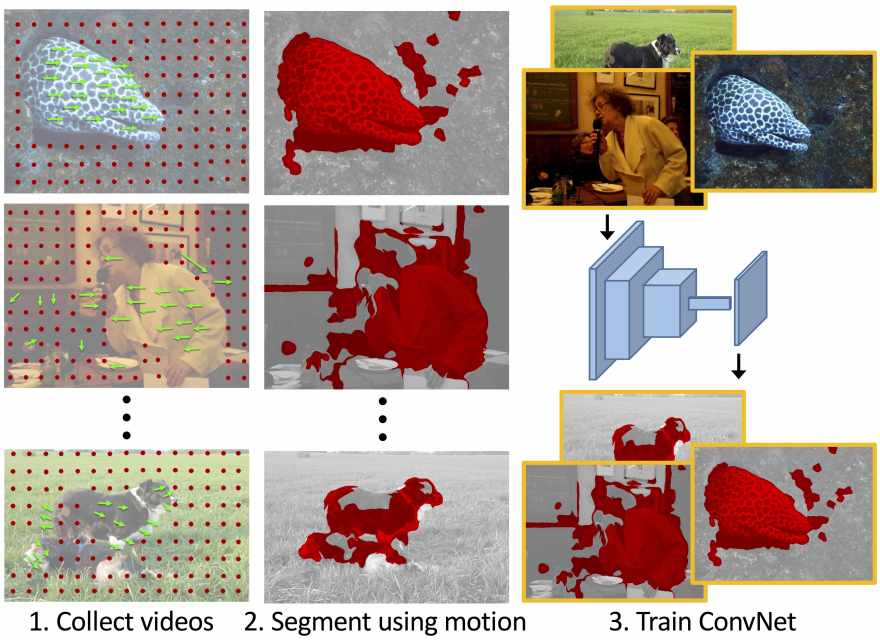


Figure 3.2: Overview of our approach. We use motion cues to segment objects in videos *without any supervision*. We then train a ConvNet to predict these segmentations from *static frames*, *i.e.* without any motion cues. We then transfer the learned representation to other recognition tasks.

3.1 Evaluating Feature Representations

To measure the quality of a learned feature representation, we need an evaluation that reflects real-world constraints to yield useful conclusions. Prior work on unsupervised learning has evaluated representations by using them as initializations for fine-tuning a ConvNet for a particular isolated task, such as object detection [51]. The intuition is that a good representations should serve as a good starting point for task-specific fine-tuning. While fine-tuning for each task can be a good solution, it can also be impractical.

For example, a mobile app might want to handle multiple tasks on device, such as image classification, object detection, and segmentation. But both the app download size and execution time will grow linearly with the number of tasks unless computation is shared. In such cases it may be desirable to have a general representation that is shared between tasks and task-specific, lightweight classifier 'heads'.

Another practical concern arises when the amount of labeled training data is too limited for fine-tuning. Again, in this scenario it may be desirable to use a fixed general representation with a trained task-specific 'head' to avoid overfitting. Rather than emphasizing any one of these cases, in this chapter we aim for a broader understanding by evaluating learned representations under a variety of conditions:

1. **On multiple tasks:** We consider object detection, image classification and semantic segmentation.

2. **With shared layers:** We fine-tune the pretrained ConvNet weights to different extents, ranging from only the fully connected layers to fine-tuning everything (see [164] for a similar evaluation on ImageNet).
3. **With limited target task training data:** We reduce the amount of training data available for the target task.

3.2 Learning Features by Learning to Group

The core intuition behind the idea in this chapter is that training a ConvNet to *group pixels in static images into objects without any class labels* will cause it to learn a strong, high-level feature representation. This is because such grouping is difficult from low-level cues alone: objects are typically made of multiple colors and textures and, if occluded, might even consist of spatially disjoint regions. Therefore, to effectively do this grouping is to implicitly *recognize* the object and understand its location and shape, even if it cannot be *named*. Thus, if we train a ConvNet for this task, we expect it to learn a representation that aids recognition.

To test this hypothesis, we ran a series of experiments using high-quality manual annotations on static images from COCO [142]. Although *supervised*, these experiments help to evaluate a) how well our method might work under ideal conditions, b) how performance is impacted if the segments are of lower quality, and c) how much data is needed. We now describe these experiments in detail.

3.2.1 Training a ConvNet to Segment Objects

We frame the task as follows: given an image patch containing a single object, we want the ConvNet to segment the object, *i.e.*, assign each pixel a label of 1 if it lies on the object and 0 otherwise. Since an image contains multiple objects, the task is ambiguous if we feed the ConvNet the entire image. Instead, we sample an object from an image and crop a box around the ground truth segment. However, given a precise bounding box, it is easy for the ConvNet to cheat: a blob in the center of the box would yield low loss. To prevent such degenerate solutions, we jitter the box in position and scale. Note that a similar training setup was used for recent segmentation proposal methods [190, 192].

We use a straightforward ConvNet architecture that takes as input a $w \times w$ image and outputs an $s \times s$ mask. Our network ends in a fully connected layer with s^2 outputs followed by an element-wise sigmoid. The resulting s^2 dimensional vector is reshaped into an $s \times s$ mask. We also downsample the ground truth mask to $s \times s$ and sum the cross entropy losses over the s^2 locations to train the network.

3.2.2 Experiments

To enable comparisons to prior work, we use AlexNet [128] as our ConvNet architecture. We use $s = 56$ and $w = 227$. We use images and annotations from the trainval set of the COCO

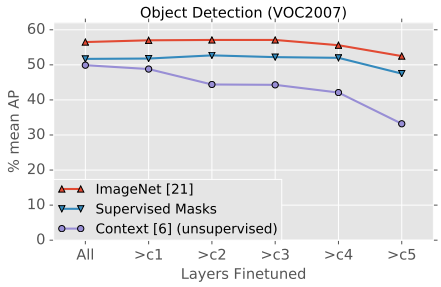


Figure 3.3: Our representation trained on manually-annotated segments from COCO (without class labels) compared to ImageNet pretraining and context prediction (unsupervised) [51], for object detection on PASCAL VOC 2007. '>cX': all layers above convX are fine-tuned; 'All': the entire net is fine-tuned.

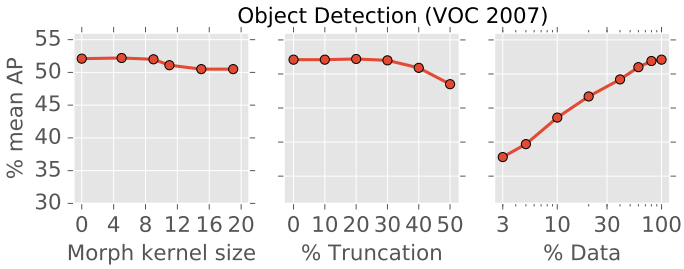


Figure 3.4: VOC object detection accuracy using our *supervised* ConvNet as noise is introduced in mask boundaries, the masks are truncated, or the amount of data is reduced. Surprisingly, the representation maintains quality even with large degradation.

dataset [142], *discarding the class labels* and only using the segmentations.

Does training for segmentation yield good features? Following recent work on unsupervised learning, we perform experiments on the task of object detection on PASCAL VOC 2007 using Fast R-CNN [80].[1] We use multi-scale training and testing [80]. In keeping with the motivation described in Section 3.1, we measure performance with ConvNet layers frozen to different extents. We compare our representation to a ConvNet trained on image classification on ImageNet, and the representation trained by Doersch *et al.* [51]. The latter is competitive with the state-of-the-art. (Comparisons to other recent work on unsupervised learning appear later.) The results are shown in Figure 3.3.

We find that our supervised representation outperforms the unsupervised context prediction model across all scenarios by a large margin, which is to be expected. Notably though, our model maintains a fairly small gap with ImageNet pretraining. This result is state-of-the-art for a model trained without semantic category labels. Thus, given high-quality segments, our proposed method can learn a strong representation, which validates our hypothesis.

Figure 3.3 also shows that the model trained on context prediction degrades rapidly as more layers are frozen. This drop indicates that the higher layers of the model have become overly specific to the *pretext* task [273], and may not capture the high-level concepts needed for object recognition. This is in contrast to the stable performance of the ImageNet trained model even when most of the network is frozen, suggesting the utility of its higher layers for recognition tasks. We find that this trend is also true for our representation: *it retains good performance even when most of the ConvNet is frozen*, indicating that it has indeed learned high-level semantics in the higher layers.

[1] https://github.com/rbgirshick/py-faster-rcnn

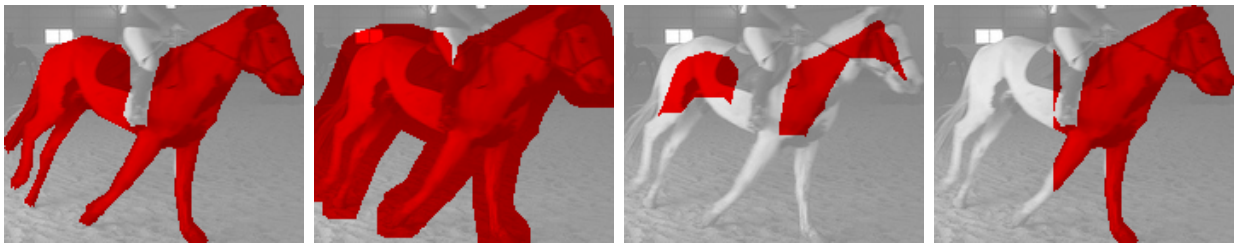


Figure 3.5: We degrade ground truth masks to measure the impact of segmentation quality on the learned representation. From left to right, the original mask, dilated and eroded masks (boundary errors), and a truncated mask (truncation can be on any side).

Can the ConvNet learn from noisy masks? We next ask if the quality of the learned representation is impacted by the quality of the ground truth, which is important since the segmentations obtained from unsupervised motion-based grouping will be imperfect. To simulate noisy segments, we train the representation with degraded masks from COCO. We consider two ways of creating noisy segments: introducing noise in the boundary and truncating the mask.

Noise in the segment boundary simulates the foreground leaking into the background or vice-versa. To introduce such noise during training, for each cropped ground truth mask, we randomly either erode or dilate the mask using a kernel of fixed size (Figure 3.5, second and third images). The boundaries become noisier as the kernel size increases.

Truncation simulates the case when we miss a part of the object, such as when only part of the object moves. Specifically, for each ground truth mask, we zero out a strip of pixels corresponding to a fixed percentage of the bounding box area from one of the four sides (Figure 3.5, last image).

We evaluate the representation trained with these noisy ground truth segments on object detection using Fast R-CNN with all layers up to and including conv5 frozen (Figure 3.4). We find that the learned representation is surprisingly resilient to both kinds of degradation. Even with large, systematic truncation (up to 50%) or large errors in boundaries, the representation maintains its quality.

How much data do we need? We vary the amount of data available for training, and evaluate the resulting representation on object detection using Fast-RCNN with all conv layers frozen. The results are shown in the third plot in Figure 3.4. We find that performance drops significantly as the amount of training data is reduced, suggesting that good representations will need large amounts of data.

In summary, these results suggest that training for segmentation leads to strong features even with imprecise object masks. However, building a good representation requires significant amounts of training data. These observations strengthen our case for learning features in an unsupervised manner on large unlabeled datasets.

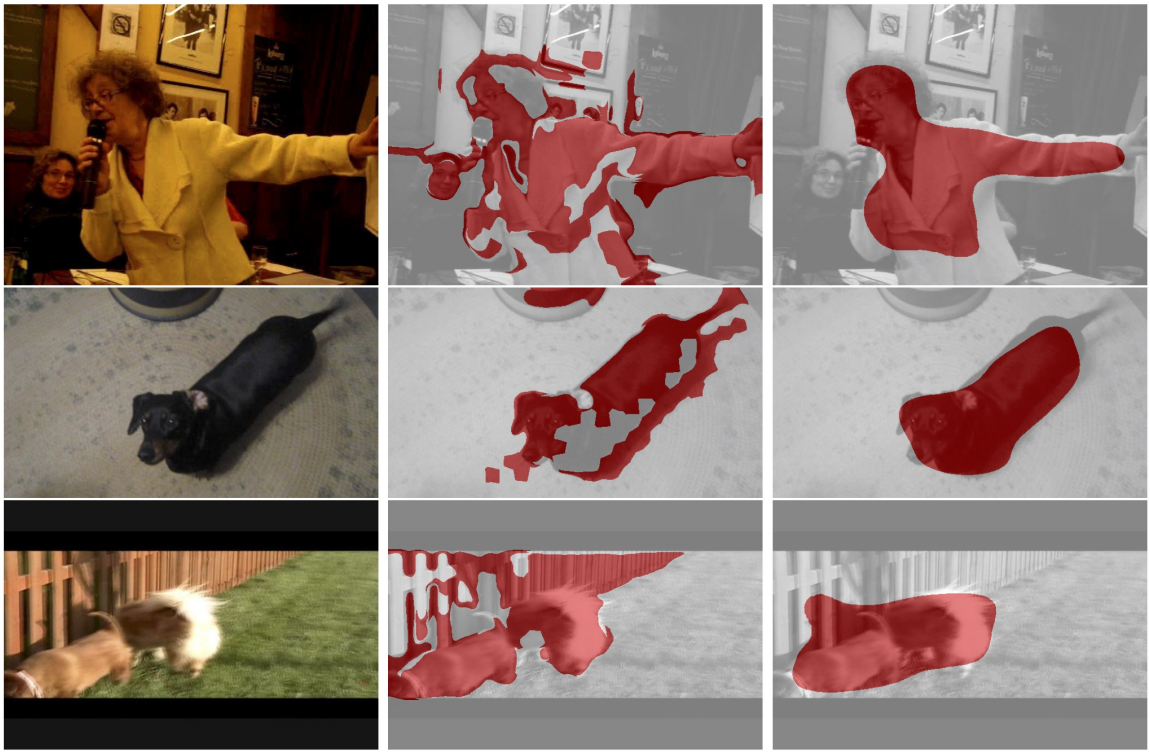


Figure 3.6: From left to right: a video frame, the output of uNLC that we use to train our ConvNet, and the output of our ConvNet. uNLC is able to highlight the moving object even in potentially cluttered scenes, but is often noisy, and sometimes fails (last two rows). Nevertheless, our ConvNet can still learn from this noisy data and produce significantly better and smoother segmentations.

3.3 Learning Features by Watching Objects Move

We first describe the motion segmentation algorithm we use to segment videos, and then discuss how we use the segmented frames to train a ConvNet.

3.3.1 Unsupervised Motion Segmentation

The key idea behind motion segmentation is that if there is a single object moving with respect to the background through the entire video, then pixels on the object will move differently from pixels on the background. Analyzing the optical flow should therefore provide hints about which pixels belong to the foreground. However, since only a part of the object might move in each frame, this information needs to be aggregated across multiple frames.

We adopt the NLC approach from Faktor and Irani [64]. While NLC is unsupervised with respect to video segmentation, it utilizes an edge detector that was trained on labeled edge images [195]. In order to have a purely unsupervised method, we replace the trained edge detector in NLC with unsupervised superpixels. To avoid confusion, we call our implementation of NLC as uNLC. First uNLC computes a per-frame saliency map based

on motion by looking for either pixels that move in a mostly static frame or, if the frame contains significant motion, pixels that move in a direction different from the dominant one. Per-pixel saliency is then averaged over superpixels [2]. Next, a nearest neighbor graph is computed over the superpixels in the video using location and appearance (color histograms and HOG [41]) as features. Finally, it uses a nearest neighbor voting scheme to propagate the saliency across frames.

We find that uNLC often fails on videos in the wild. Sometimes this is because the assumption of there being a single moving object in the video is not satisfied, especially in long videos made up of multiple shots showing different objects. We use a publicly available appearance-based shot detection method [198] (also unsupervised) to divide the video into shots and run uNLC separately on each shot.

Videos in the wild are also often low resolution and have compression artifacts, which can degrade the resulting segmentations. From our experiments using strong supervision, we know our approach can be robust to such noise. Nevertheless, since a large video dataset comprises a massive collection of frames, we simply discard badly segmented frames based on two heuristics. Specifically, we discard: (1) frames with too many (>80%) or too few (<10%) pixels marked as foreground; (2) frames with too many pixels (>10%) within 5% of the frame border that are marked as foreground. In preliminary tests, we found that results were not sensitive to the precise thresholds used.

We ran uNLC on videos from YFCC100m [243], which contains about 700,000 videos. After pruning, we ended up with 205,000 videos. We sampled 5-10 frames per shot from each video to create our dataset of 1.6M images, so we have slightly more frames than images in ImageNet. However, note that our frames come from fewer videos and are therefore more correlated than images from ImageNet. We stress that our approach in generating this dataset is completely unsupervised, and does not use any form of supervised learning in any part of the pipeline.

Our motion segmentation approach is far from state-of-the-art, as can be seen by the noisy segments shown in Figure 3.6. Nevertheless, we find that our representation is quite resilient to this noise (as shown below). As such, we did not aim to improve the particulars of our motion segmentation.

3.3.2 Learning to Segment from Noisy Labels

As before, we feed the ConvNet cropped images, jittered in scale and translation, and ask it to predict the motile foreground object. Since the motion segmentation output is noisy, we do not trust the absolute foreground probabilities it provides. Instead, we convert it into a *trimap* representation in which pixels with a probability <0.4 are marked as negative samples, those with a probability >0.7 are marked as positives, and the remaining pixels are marked as "don't cares" (in preliminary experiments, our results were found to be robust to these thresholds). The ConvNet is trained with a logistic loss only on the positive and negative pixels; don't care pixels are ignored. Similar techniques have been successfully explored earlier in segmentation [11, 123].

Figure 3.7: Examples of segmentations produced by our ConvNet on held out images. The ConvNet is able to identify the motile object (or objects) and segment it out from a single frame. Masks are not perfect but they do capture the general object shape.

Despite the steps we take to get good segments, the uNLC output is still noisy and often grossly incorrect, as can be seen from the second column of Figure 3.6. However, if there are no *systematic* errors, then these motion-based segments can be seen as perturbations about a true latent segmentation. Because a ConvNet has finite capacity, it will not be able to fit the noise perfectly and might instead learn something closer to the underlying correct segmentation.

Some positive evidence for this can be seen in the output of the trained ConvNet on its training images (Fig. 3.6, third column). The ConvNet correctly identifies the motile object and its rough shape, leading to a smoother, more correct segmentation than the original motion segmentation.

The ConvNet is also able to *generalize* to unseen images. Figure 3.7 shows the output of the ConvNet on frames from the DAVIS [188], FBMS [165] and VSB [74] datasets, which were not used in training. Again, it is able to identify the moving object and its rough shape from just a single frame. When evaluated against human annotated segments in these datasets, we find that the ConvNet's output is significantly *better* than the uNLC segmentation output as shown below:

Metric	uNLC	ConvNet (unsupervised)
Mean IoU (%)	13.1	**24.8**
Precision (%)	15.4	**29.9**
Recall (%)	45.8	**59.3**

These results confirm our earlier finding that the ConvNet is able to learn well even from noisy and often incorrect ground truth. However, the goal of this appraoch is not segmentation, but representation learning. We evaluate the learned representation in the next section.

	Full train set						150 image set						
Method	All	>c1	>c2	>c3	>c4	>c5	All	>c1	>c2	>c3	>c4	>c5	#wins
Supervised													
Imagenet	56.5	57.0	57.1	57.1	55.6	52.5	17.7	19.1	19.7	20.3	20.9	19.6	NA
Sup. Masks (Ours)	51.7	51.8	52.7	52.2	52.0	47.5	13.6	13.8	15.5	17.6	18.1	15.1	NA
Unsupervised													
Jigsaw‡ [164]	49.0	50.0	48.9	47.7	45.8	37.1	5.9	8.7	8.8	10.1	9.9	7.9	NA
Kmeans [126]	42.8	42.2	40.3	37.1	32.4	26.0	4.1	4.9	5.0	4.5	4.2	4.0	0
Egomotion [4]	37.4	36.9	34.4	28.9	24.1	17.1	–	–	–	–	–	–	0
Inpainting [183]	39.1	36.4	34.1	29.4	24.8	13.4	–	–	–	–	–	–	0
Tracking-gray [254]	43.5	44.6	44.6	44.2	41.5	35.7	3.7	5.7	7.4	9.0	9.4	9.0	0
Sounds [175]	42.9	42.3	40.6	37.1	32.0	26.5	5.4	5.1	5.0	4.8	4.0	3.5	0
BiGAN [54]	44.9	44.6	44.7	42.4	38.4	29.4	4.9	6.1	7.3	7.6	7.1	4.6	0
Colorization [276]	44.5	44.9	44.7	44.4	42.6	38.0	6.1	7.9	8.6	10.6	10.7	9.9	0
Split-Brain Auto [277]	43.8	45.6	45.6	46.1	44.1	37.6	3.5	7.9	9.6	10.2	11.0	10.0	0
Context [51]	**49.9**	**48.8**	44.4	44.3	42.1	33.2	6.7	**10.2**	9.2	9.5	9.4	8.7	3
Context-videos† [51]	47.8	47.9	46.6	**47.2**	44.3	33.4	6.6	9.2	10.7	12.2	11.2	9.0	1
Motion Masks (Ours)	48.6	48.2	**48.3**	47.0	**45.8**	**40.3**	**10.2**	**10.2**	**11.7**	**12.5**	**13.3**	**11.0**	**9**

Table 3.1: Object detection AP (%) on PASCAL VOC 2012 using Fast R-CNN with various pretrained ConvNets. All models are trained on `train` and tested on `val` using consistent Fast R-CNN settings. '–' means training didn't converge due to insufficient data. Our approach achieves the best performance in the majority of settings. †Doersch *et al.* [51] trained their original context model using ImageNet images. The Context-videos model is obtained by retraining their approach on our video frames from YFCC. This experiment controls for the effect of the distribution of training images and shows that the image domain used for training does not significantly impact performance. ‡Noroozi *et al.* [164] use a more computationally intensive ConvNet architecture (>2× longer to finetune) with a finer stride at conv1, preventing apples-to-apples comparisons. Nevertheless, their model works significantly worse than our representation when either layers are frozen or in case of limited data and is comparable to ours when network is finetuned with full training data.

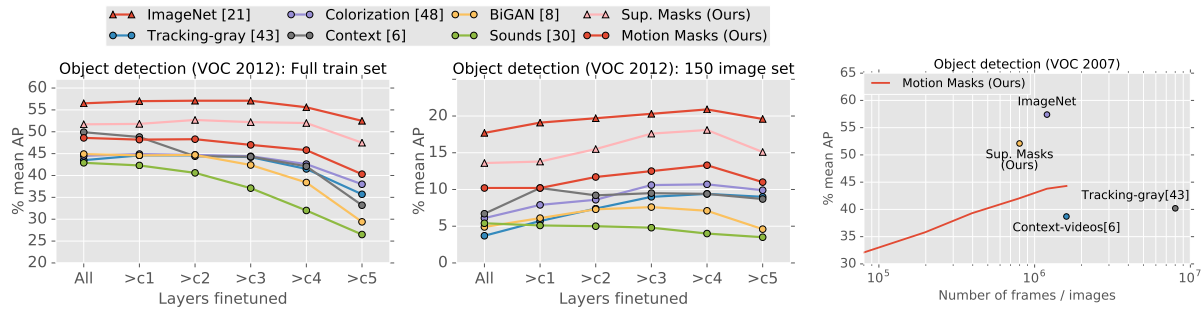
3.4 Evaluating the Learned Representation

3.4.1 Transfer to Object Detection

We first evaluate our representation on the task of object detection using Fast R-CNN. We use VOC 2007 for cross-validation: we pick an appropriate learning rate for each method out of a set of 3 values $\{0.001, 0.002$ and $0.003\}$. Finally, we train on VOC 2012 train and test on VOC 2012 val exactly once. We use multi-scale training and testing and discard difficult objects during training.

We present results with the ConvNet parameters frozen to different extents. As discussed in Section 3.1, a good representation should work well both as an initialization to fine-tuning and also when most of the ConvNet is frozen.

We compare our approach to ConvNet representations produced by prior work on unsupervised learning [4, 51, 54, 164, 175, 183, 254, 276]. We use publicly available models for all methods shown. Like our ConvNet representation, all models have the AlexNet architecture,



(a) Performance vs. Finetuning (b) Performance vs. Data

Figure 3.8: Results on object detection using Fast R-CNN. (a) VOC 2012 object detection results when the ConvNet representation is frozen to different extents. We compare to other unsupervised and supervised approaches. **Left**: using the full training set. **Right**: using only 150 training images (note the different y-axis scales). (b) Variation of representation quality (mean AP on VOC 2007 object detection with conv5 and below frozen) with number of training frames. A few other methods are also shown. Context-videos [51] is the representation of Doersch *et al*. [51] retrained on our video frames. Note that most other methods in Table 3.1 use ImageNet as their train set.

but differ in minor details such as the presence of batch normalization layers [51] or the presence of grouped convolutions [276].

We also compare to two models trained with strong supervision. The first is trained on ImageNet classification. The second is trained on manually-annotated segments (without class labels) from COCO (see Section 3.2).

Results are shown in Figure 3.8a (left) and Table 3.1 (left). We find that our representation learned from unsupervised motion segmentation performs on par or better than prior work on unsupervised learning across all scenarios.

As we saw in Section 3.2.2, in contrast to ImageNet supervised representations, the representations learned by previous unsupervised approaches show a large decay in performance as more layers are frozen, owing to the representation becoming highly specific to the pretext task. Similar to our *supervised* approach trained on segmentations from COCO, we find that our *unsupervised* approach trained on motion segmentation also shows *stable* performance as the layers are frozen. Thus, unlike prior work on unsupervised learning, the upper layers in our representation learn high-level abstract concepts that are useful for recognition.

It is possible that some of the differences between our method and prior work are because the training data is from different domains (YFCC100m videos *vs*. ImageNet images). To control for this, we retrained the model from [51] on frames from our video dataset (see Context-videos in Table 3.1). The two variants perform similarly: 33.4% mean AP when trained on YFCC with conv5 and below frozen compared to 33.2% for the ImageNet version. This confirms that the different image sources do not explain our gains.

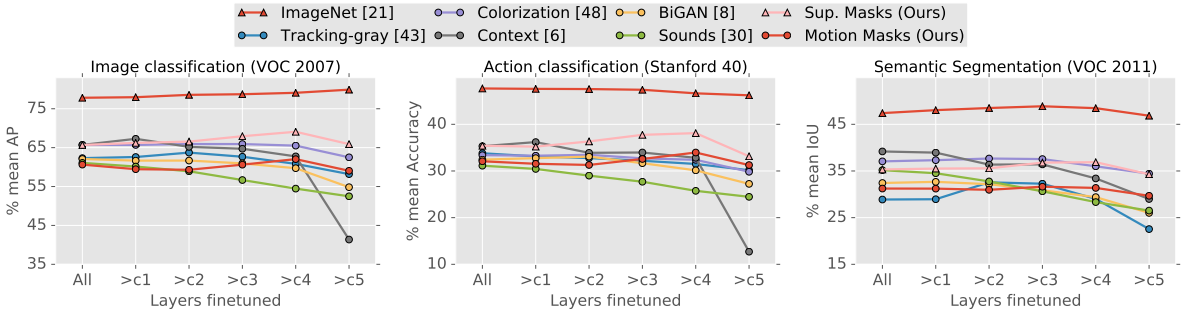


Figure 3.9: Results on image classification on VOC 2007, single-image action classification on Stanford 40 Actions, and semantic segmentation on VOC 2011. Results shown with ConvNet layers frozen to different extents (note that the metrics vary for each task).

3.4.2 Low-shot Transfer

A good representation should also aid learning when training data is scarce, as we motivated in Section 3.1. Figure 3.8a (right) and Table 3.1 (right) show how we compare to other unsupervised and supervised approaches on the task of object detection when we have few (150) training images. We observe that in this scenario it actually hurts to fine-tune the entire network, and the best setup is to leave some layers frozen. Our approach provides the best AP overall (achieved by freezing all layers up to and including conv4) among all other representations from recent unsupervised learning methods by a large margin.

Note that in spite of its strong performance relative to prior unsupervised approaches, our representation learned without supervision on video trails both the strongly supervised mask and ImageNet versions by a significant margin. We discuss this in the following subsection.

3.4.3 Impact of Amount of Training Data

The quality of our representation (measured by Fast R-CNN performance on VOC 2007 with all conv layers frozen) grows roughly logarithmically with the number of frames used. With 396K frames (50K videos), it is already better than prior state-of-the-art [51] trained on a million ImageNet images, see Figure 3.8b. With our full dataset (1.6M frames) accuracy increases substantially. If this logarithmic growth continues, our representation will be on par with one trained on ImageNet if we use about 27M frames (or 3 to 5 million videos, the same order of magnitude as the number of images in ImageNet). Note that frames from the same video are very correlated. We expect this number could be reduced with more algorithmic improvements.

3.4.4 Transfer to Other Tasks

As discussed in Section 3.1, a good representation should generalize across tasks. We now show experiments for two other tasks: image classification and semantic image segmentation.

For image classification, we test on both object and action classification.

Image Classification. We experimented with image classification on PASCAL VOC 2007 (object categories) and Stanford 40 Actions [269] (action labels). To allow comparisons to prior work [54, 276], we used random crops during training and averaged scores from 10 crops during testing (see [54] for details). We minimally tuned some hyper-parameters (we increased the step size to allow longer training) on VOC 2007 validation, and used the same settings for both VOC 2007 and Stanford 40 Actions. On both datasets, we trained with different amounts of fine-tuning as before. Results are in the first two plots in Figure 3.9.

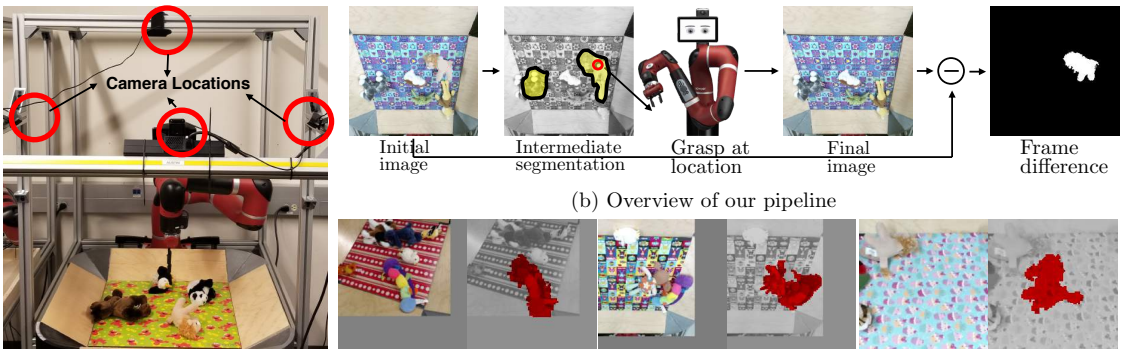
Semantic Segmentation. We use fully convolutional networks for semantic segmentation with the default hyper-parameters [145]. All the pretrained ConvNet models are finetuned on union of images from VOC 2011 train set and additional SBD train set released by Hariharan *et al*. [90], and we test on the VOC 2011 val set after removing overlapping images from SBD train. The last plot in Figure 3.9 shows the performance of different methods when the number of layers being finetuned is varied.

Analysis. Like object detection, all these tasks require semantic knowledge. However, while in object detection the ConvNet is given a tight crop around the target object, the input in these image classification tasks is the entire image, and semantic segmentation involves running the ConvNet in a sliding window over all locations. This difference appears to play a major role. Our representation was trained on object crops, which is similar to the setup for object detection, but quite different from the setups in Figure 3.9. This mismatch may negatively impact the performance of our representation, both for the version trained on motion segmentation and the strongly supervised version. Such a mismatch may also explain the low performance of the representation trained by Wang *et al*. [254] on semantic segmentation.

Nevertheless, when the ConvNet is progressively frozen, our approach is a strong performer. When all layers until conv5 are frozen, our representation is better than other approaches on action classification and second only to colorization [276] on image classification on VOC 2007 and semantic segmentation on VOC 2011. Our higher performance on action classification might be due to the fact that our video dataset has many people doing various actions.

3.5 Object-centric Representation via Interaction

It is believed that very early on in development, infants have a notion of objects and they expect objects to move as wholes on connected paths, which in turn guides their perception of object boundaries [231, 232]. Motion can either be obtained via passive observation or be caused by active interaction. While passive motion data helps in learning about objects as entities separated by boundaries, it is the "active interaction" that makes it possible for infants to learn about properties of individual entities and correlate these properties with



(a) Our robotic system

(b) Overview of our pipeline

(c) Examples of generated psuedo-masks

Figure 3.10: Overview of experimental setup and method: (a) Sawyer robot's interactions with objects placed on an arena are recorded by four cameras. The arena is designed to allow easy modification of background texture. (b) From its visual observation (initial image) the robot hypothesizes what group of pixels constitute an object (intermediate segmentation hypothesis). It randomly chooses to interact with one such group by attempting to grasp and place it to a different location on the arena. If the grasped group indeed corresponds to an object, the mask of the object can be obtained by computing the difference image between the image after and before the interaction. The mask obtained from the difference image is used as pseudo ground truth for training a neural network to predict object segmentation masks. (c) Sometimes masks produced by this process are good (first image), but they are often imperfect due to movement of multiple objects in the process of picking one object (second image) or creation of false masks due to lighting changes/shadows.

visual appearance [230]. So far, we showed that one could learn semantic representation by leveraging motion from passive observation. In this section, our focus is on developing a method that can make use of *motion signal obtained via active interaction* to learn an object segmentation model that does not require any human annotations.

To that end, we set up an agent, shown in Figure 3.10, to interact with its environment and record the resulting RGB images. The agent maintains a belief about how images can be decomposed into objects, and actively tests its belief by attempting to grasp potential objects in the world. Through such self-supervised interaction, we show that it is possible to learn to segment novel objects kept on textured backgrounds into individual instances.

A major challenge in learning a segmentation model via self-supervised interactions is that the training signal is very noisy as compared to object masks marked by human annotators (Figure 3.10). Typical error modes include: (a) false negatives due to complete failure in grasping an object; (b) failure in grasping that slightly perturbs the object resulting in incomplete masks; (c) in case two objects are located near each other, picking one object moves the other one, resulting in masks that span multiple objects; (d) erroneous masks due to variation in lighting, shadows and other nuisance factors.

Dealing with such noise requires the training procedure to be robust, analogous to how in regression, we need to be robust to outliers in the data. However, direct application of

Algorithm 1: Segmentation by Interaction

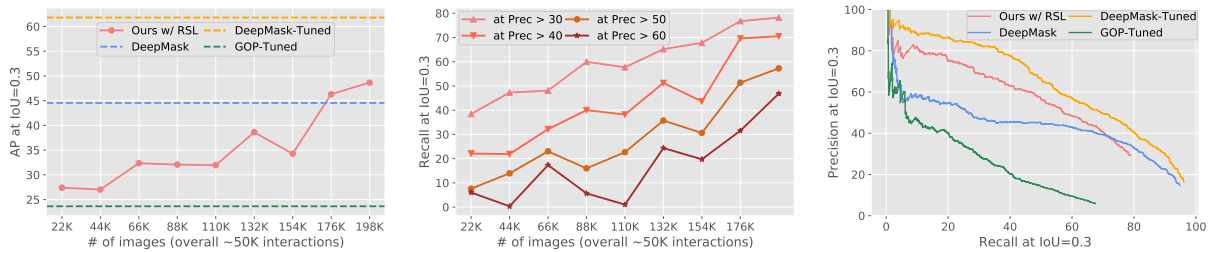
Pre-train network with passive unsupervised data
for *iteration* $t = 1$ *to* T **do**
  Record current observation I_t
  Generate object hypothesis: $\{s_1^t, \ldots s_K^t\} \Leftarrow \text{CNN}(I_t)$
  Randomly choose one hypothesis $s_j^t \in \{s_1^t, \ldots s_K^t\}$
  Interact with hypothesized object $(move(s_j^t))$
  Record observation I_{t+1}
  mask $\Leftarrow$ frame_difference(I_t, I_{t+1})
  if *mask is empty* **then**
    {(x,y), mask, I^t} is negative training example
  else
    {(x,y), mask, I^t} is positive training example
  end
  if t % *update_interval* $==$ *0* **then**
    Update CNN using positive/negative examples
  end
end

pixel-wise robust loss is sub-optimal because we are interested in a set-level statistic, such as the similarity between two sets of pixels (e.g. ground-truth and predicted masks) measured for instance using Jaccard index. Such a measurement depends on all the pixels and therefore requires one to define a robust loss over a set of pixels. We propose a technique, "robust set loss", to handle noisy segmentation training signal, with the general idea being that the segmenter is not required to predict exactly the pixels in the candidate object mask, rather that the predicted pixels as a set have a good Jaccard index overlap with the candidate mask. We show that robust set loss significantly improves segmentation performance and also reduces the variance in results.

3.5.1 Experimental Setup

The basic interaction primitive used by the robot allows it to attempt to pick and place objects in the scene. We set up the robot to interact autonomously with its environment without human supervision. Overall, the robot performed more than 50,000 interactions with its environment. Approximately the first 15,000 interactions were recorded using the main camera and the remainder of interactions were recorded using auxiliary four cameras. Data recording from four cameras was used to increase invariance to viewpoint.

Datasets: We use 24 backgrounds for training, 6 for validation and 10 for testing. We use 36 different objects for training, 8 for validation and 15 for testing. The validation set



(a) Performance vs. Interactions (b) Successes vs. Interactions (c) Precision vs. Recall

Figure 3.11: Quantitative evaluation of the segmentation model on the held-out test. (a) The performance of our system measured as mAP at IoU of 0.3 steadily increases with the amount of data. After 50K iterations our system significantly beats GOP-Tuned. (b) The efficacy of robot's interactions is computed as the recall of ground truth objects that have IoU of more than 0.3 with the group of pixels that the robot believes to be objects. The steady increase in recall at different precision threshold shows that the robot learns to perform more efficient interactions with time. (c) Precision-Recall curves re-confirm the results.

consisted of 30 images (5 images per background) and the test set consisted of 50 images (5 images per background). We manually annotated object masks in these images for the purpose of evaluation; no labels for training.

3.5.2 Instance Segmentation by Interaction

The training procedure is summarized in Algorithm 1. The major challenge in training a model with such self-generated masks is that they are far from perfect (Figure 3.10). Typical error modes include: (a) false negatives due to complete failure to grasp an object; (b) failure in grasping that slightly perturb the object resulting in incomplete masks; (c) in case two objects are located near each other, picking one object moves the other one, resulting in masks that span multiple objects; (d) erroneous masks due to variation in lighting, shadows and other nuisance factors. Any method attempting to learn object segmentation from interaction must deal with such imperfections in the self-generated *pseudo ground truth* masks.

Robust Set Loss Attempting to exactly fit the noisy masks is adversarial for the learning process, as (a) overfitting to noise would hamper the ability to generalize to unseen examples, and (b) inability to fit noise would increase variance in the gradients and thereby make training unstable.

The principled approach of learning with noisy training data is to use a robust loss for mitigating the effect of outliers. Robust loss functions have been extensively studied in statistics, in particular, Huber loss [105] applied to regression problems. However, such ideas have mostly been explored in the context of regression and classification for modeling independent outputs. Unfortunately, segmentation mask is a "set of pixels", where a statistic of interest such as the similarity between two sets of pixels (e.g., ground-truth and predicted

Method	Supervision	AP at IU 0.3	AP at IU 0.5
GOP	Bottom up	10.9	04.1
GOP (tuned)	Bottom up	23.6	16.3
DeepMask	Strong Sup.	44.5	34.3
DeepMask (tuned)	Strong Sup.	61.8	47.3
Ours + Human	Semi-sup.	43.1 ± 2.6	21.1 ± 2.6
Ours	Self-sup.	41.1 ± 2.4	16.0 ± 2.6
Ours + Robust Set Loss	Self-sup.	45.9 ± 2.1	22.5 ± 1.3

Table 3.2: Quantitative comparison of our method with bottom-up (GOP [127]), learned top-down (DeepMask [191]) segmentation methods and optimization without robust set loss on the full test set. We report the mean and standard deviation for our approach. Our approach significantly outperforms GOP, but is outperformed by DeepMask that uses strong manual supervision of 700K+ COCO segments and 1M ImageNet images. Adding 1470 images with clean manually annotated masks improves performance of our base system. The robust set loss improves both the mean performance and reduces variance over normal cross-entropy loss.

masks) measured for instance using Jaccard Index (i.e., intersection over union (IOU)) depends on all the pixels. The dependence of the statistic on a set of pixels makes it non-trivial to generalize ideas such as Huber loss in a straightforward manner. We formulate Robust Set Loss (RSL) to deal with "set-level" noise. The key insight is to impose a soft constraint for only matching a subset of target pixels while ensuring that (potentially non-differentiable) some metric of interest, such as IOU, between the prediction and the noisy target is greater than or equal to a certain threshold. We generalize the CCNN constrained formulation proposed in Pathak et. al. [182] to achieve this loss. Please refer to full paper for the details of RSL [187].

Bootstrapping via Passive Self-Supervision Without any prior knowledge, the agent's initial beliefs about objects will be arbitrary, causing it to spend most of its time interacting with the background. This process would be very inefficient. We address this issue by assuming that initially our agent can passively observe objects moving in its environment. For this purpose we use a prior robotic pushing dataset [5] that was constructed by a robot randomly pushing objects in a tabletop environment. We again apply uNLC, Section 3.3.1, to automatically extract masks from this data, which we use to pre-train our ResNet-18 network (initialized with random-weights). Note that this method of pre-training is completely self-supervised.

3.5.3 Results and Evaluations

We compare the performance of our method against a state-of-the-art bottom up segmentation method called Geodesic Object Proposals (GOP [127]), and a top-down instance segmentation method trained in a class agnostic manner using over 700K strongly supervised masks obtained

from the COCO dataset (DeepMask [191]) and pre-trained on 1M ImageNet, using the AP at IOU 0.3 metric on the held-out testing set as shown in Figure 3.11a. Our system significantly outperforms vanilla GOP and GOP with domain knowledge (tuned). These results are re-confirmed by the precision-recall curves shown in Figure 3.11c. The performance of our system steadily increases with the amount of data and from the performance curve (see Figure 3.11a). Our method performs similar to vanilla DeepMask, but worse than the one tuned to our domain for scaling and position of objects. This result is significant because DeepMask was trained with perfect ground truth segmentation masks for 700K COCO objects after being pre-trained to classify 1M ImageNet images, whereas our system was trained using imperfect masks (section 3.5.2) using only 50K self-supervised active interactions after pre-training with approximately 60K passive observations of moving objects. AP evaluation at IOU 0.5 (see Table 3.2) reveals that while our method significantly outperforms GOP, it is outperformed by DeepMask. We believe the main reason is that the masks obtained by robot interaction are imperfect.

3.6 Related Work

We dicussed relevant prior works in self-supervised learning in Chapter 2, Section 2.4. We now cover related literature specifically pertaining to learning from motion and interaction.

Learning from motion and action The human visual system does not receive static images; it receives a continuous video stream. The same idea of defining auxiliary pretext tasks can be used in unsupervised learning from videos too. Wang and Gupta [254] train a ConvNet to distinguish between pairs of tracked patches in a single video, and pairs of patches from different videos. Misra *et al*. [153] ask a network to arrange shuffled frames of a video into a temporally correct order. Another such pretext task is to make predictions about the next few frames: Goroshin *et al*. [85] predict pixels of future frames and Walker *et al*. [250] predict dense future trajectories. However, since nearby frames in a video tend to be visually similar (in color or texture), these approaches might learn low-level image statistics instead of more semantic features. Alternatively, Li *et al*. [140] use motion boundary detection to bootstrap a ConvNet-based contour detector, but find that this does not lead to good feature representations. Our intuitions are similar, but our approach produces semantically strong representations.

Animals and robots can also sense their own motion (proprioception), and a possible task is to predict this signal from the visual input alone [4, 76, 111]. While such cues undoubtedly can be useful, we show that strong representations can be learned even when such cues are unavailable.

Self-Supervised Robot Learning Our work draws upon the ideas from the active perception [7, 12, 13, 77] to build a self-supervised object segmentation system. Many recent papers have investigated use of self-supervised learning for performing sensorimotor tasks. This

includes self-supervised grasping [138, 148, 194], pushing [5, 31, 66, 193], navigation [75, 186] and rope-manipulation [161, 186]. However, the focus of these works was geared for an end task. Our goal is different – it is to a learn robust "segmentation" from noisy interaction signal. Such segmentation can be a building block for multiple robotic applications.

Interactive Segmentation Improving the result of segmentation by interaction has drawn a lot of interest [23, 68, 92, 118, 162, 176, 247]. However, most these works are concerned with using interaction to segment a specific scene. In contrast, our system uses interactions to actively gather supervision to train a segmentation system that can be used to segment objects in new images. The recent work on SE3 nets [31] learns to segment and model dynamics of rigid bodies in table-top environments containing boxes. We show object segmentation results from purely RGB images in visually more complex environment without using any depth.

3.7 Discussion

We have presented a simple and intuitive approach to unsupervised learning by using segments from low-level motion-based grouping to train ConvNets. Our experiments show that our approach enables effective transfer especially when computational or data constraints limit the amount of task-specific tuning we can do. Scaling to larger video datasets should allow for further improvements.

We noted in Figure 3.6 that our network learns to refine the noisy input segments. This is a good example of a scenario where ConvNets can learn to extract signal from large amounts of noisy data. Combining the refined, single-frame output from the ConvNet with noisy motion cues extracted from the video should lead to better pseudo ground truth, and can be used by the ConvNet to bootstrap itself. We leave this direction for future work.

In the final section, we presented a method for using *active* self-supervision to reorganize visual inputs into object instances. The performance of our system is likely to benefit from obtaining better pseudo ground truth masks by the use of better grasping techniques, use of other interaction primitives and joint learning of perceptual and control systems where the interaction mechanism also improves with time.

To build general purpose sensorimotor learning systems, it is critical to find ways to transfer knowledge across tasks. While one approach is to come up with better algorithms for transfer learning, the other is to make use of more structured representations of sensory data than obtained using vanilla feed-forward neural networks. This work, builds upon the second view, in proposing a method for segmenting an image into objects in the hope that object-centric representations might be an important aspect of future visuo-motor control systems. Our system is only the first step towards the grander goal of creating agents that can self-improve and continuously learn about their environment.

Part II

Learning to Act via Self-Supervised Exploration

Chapter 4

Curiosity-driven Exploration by Self-supervised Prediction

Learning to see the world the way we do is only the first step. An agent will be able to adapt and acquire increasingly complex behaviors only when it learns to use its sensory representation to act. However, acting in the world is unlike passive observations because the data is sequentially *dependent* on the past which leads to exponentially many possible trajectories and makes it intractable to figure out the ones that constitute useful skills. Reinforcement learning (RL) is currently one of the dominant paradigms in learning how to act.

RL algorithms aim at learning policies for achieving target tasks by maximizing rewards provided by the environment. In some scenarios, these rewards are supplied to the agent continuously, e.g. the running score in an Atari game [155], or the distance between a robot arm and an object in a reaching task [141]. However, in many real-world scenarios, rewards extrinsic to the agent are extremely sparse or missing altogether, and it is not possible to construct a shaped reward function. This is a problem as the agent receives reinforcement for updating its policy only if it succeeds in reaching a pre-specified goal state. Hoping to stumble into a goal state by chance (i.e. random exploration) is likely to be futile for all but the simplest of environments.

As human agents, we are accustomed to operating with rewards that are so sparse that we only experience them once or twice in a lifetime, if at all. To a three-year-old enjoying a sunny Sunday afternoon on a playground, most trappings of modern life – college, good job, a house, a family – are so far into the future, they provide no useful reinforcement signal. Yet, the three-year-old has no trouble entertaining herself in that playground using what psychologists call intrinsic motivation [206] or curiosity [227]. Motivation/curiosity have been used to explain the need to explore the environment and discover novel states. The French word *flâneur* perfectly captures the notion of a curiosity-driven observer, the "deliberately

This chapter is based on the papers published previously at ICML 2017 [179] and ICLR 2019 [29].

(a) learn to explore in Level-1

(b) explore faster in Level-2

Figure 4.1: Discovering how to play *Super Mario Bros* **without rewards**. (a) Using only curiosity-driven exploration, the agent makes significant progress in Level-1. (b) The gained knowledge helps the agent explore subsequent levels much faster than when starting from scratch. Watch the video at http://pathak22.github.io/noreward-rl/

aimless pedestrian, unencumbered by any obligation or sense of urgency" (Cornelia Otis Skinner). More generally, curiosity is a way of learning new skills which might come handy for pursuing rewards in the future.

Similarly, in reinforcement learning, intrinsic motivation/rewards become critical whenever extrinsic rewards are sparse. Most formulations of intrinsic reward can be grouped into two broad classes: 1) encourage the agent to explore "novel" states [17, 146, 199] or, 2) encourage the agent to perform actions that reduce the error/uncertainty in the agent's ability to predict the consequence of its own actions (i.e. its knowledge about the environment) [103, 156, 216, 217, 229, 234].

Measuring "novelty" requires a statistical model of the distribution of the environmental states, whereas measuring prediction error/uncertainty requires building a model of environmental dynamics that predicts the next state (s_{t+1}) given the current state (s_t) and the action (a_t) executed at time t. Both these models are hard to build in high-dimensional continuous state spaces such as images. An additional challenge lies in dealing with the stochasticity of the agent-environment system, both due to the noise in the agent's actuation, which causes its end-effectors to move in a stochastic manner, and, more fundamentally, due to the inherent stochasticity in the environment. To give the example from [217], if the agent receiving images as state inputs is observing a television screen displaying white noise, every state will be novel as it would be impossible to predict the value of any pixel in the future. This means that the agent will remain curious about the television screen because it is unaware that some parts of the state space simply cannot be modeled and thus the agent can fall into an artificial curiosity trap and stall its exploration. Other examples of such stochasticity include appearance changes due to shadows from other moving entities, presence of distractor objects, or other agents in the environment whose motion is not only hard to predict but is also irrelevant to the agent's goals. Somewhat different, but related, is the challenge of generalization across physically (and perhaps also visually) distinct but functionally similar parts of an environment, which is crucial for large-scale problems. One

proposed solution to all these problems is to only reward the agent when it encounters states that are hard to predict but are "learnable" [216]. However, estimating learnability is a non-trivial problem [146].

This work belongs to the broad category of methods that generate an intrinsic reward signal based on how hard it is for the agent to predict the consequences of its own actions However, we manage to escape most pitfalls of previous prediction approaches with the following key insight: we only predict those changes in the environment that could possibly be due to the actions of our agent or affect the agent, and ignore the rest. That is, instead of making predictions in the raw sensory space (e.g. pixels), we transform the sensory input into a feature space where only the information relevant to the action performed by the agent is represented. We learn this feature space using self-supervision – training a neural network on a proxy inverse dynamics task of predicting the agent's action given its current and next states. Since the neural network is only required to predict the action, it has no incentive to represent within its feature embedding space the factors of variation in the environment that do not affect the agent itself. We then use this feature space to train a forward dynamics model that predicts the feature representation of the next state, given the feature representation of the current state and the action. We provide the prediction error of the forward dynamics model to the agent as an intrinsic reward to encourage its curiosity.

The role of curiosity has been widely studied in the context of solving tasks with sparse rewards. In our opinion, curiosity has two other fundamental uses. Curiosity helps an agent explore its environment in the quest for new knowledge (a desirable characteristic of exploratory behavior is that it should improve as the agent gains more knowledge). Further, curiosity is a mechanism for an agent to learn skills that might be helpful in future scenarios. We evaluate the effectiveness of our curiosity formulation in all three of these roles.

We first compare the performance of an A3C agent [154] with and without the curiosity signal on 3-D navigation tasks with sparse extrinsic reward in the *VizDoom* environment. We show that a curiosity-driven intrinsic reward is crucial in accomplishing these tasks (see Section 4.2.2). Next, we show that even in the absence of any extrinsic rewards, a curious agent learns good exploration policies. For instance, an agent trained only with curiosity as its reward is able to cross a significant portion of Level-1 in *Super Mario Bros.* Similarly in *VizDoom*, the agent learns to walk intelligently along the corridors instead of bumping into walls or getting stuck in corners (see Section 4.2.3). A question that naturally follows is whether the learned exploratory behavior is specific to the physical space that the agent trained itself on, or if it enables the agent to perform better in unseen scenarios too? We show that the exploration policy learned in the first level of *Mario* helps the agent explore subsequent levels faster (shown in Figure 4.1), while the intelligent walking behavior learned by the curious *VizDoom* agent transfers to a completely new map with new textures (see Section 4.2.4). These results suggest that the proposed method enables an agent to learn generalizable skills even in the absence of an explicit goal.

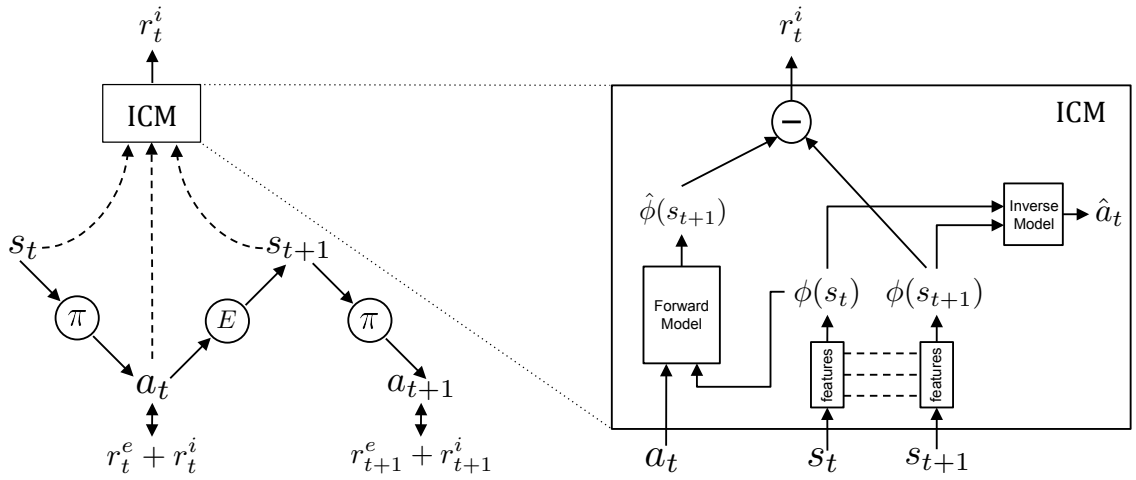


Figure 4.2: The agent in state s_t interacts with the environment by executing an action a_t sampled from its current policy π and ends up in the state s_{t+1}. The policy π is trained to optimize the sum of the extrinsic reward (r_t^e) provided by the environment E and the curiosity based intrinsic reward signal (r_t^i) generated by our proposed Intrinsic Curiosity Module (ICM). ICM encodes the states s_t, s_{t+1} into the features $\phi(s_t), \phi(s_{t+1})$ that are trained to predict a_t (i.e. inverse dynamics model). The forward model takes as inputs $\phi(s_t)$ and a_t and predicts the feature representation $\hat{\phi}(s_{t+1})$ of s_{t+1}. The prediction error in the feature space is used as the curiosity based intrinsic reward signal. As there is no incentive for $\phi(s_t)$ to encode any environmental features that can not influence or are not influenced by the agent's actions, the learned exploration strategy of our agent is robust to uncontrollable aspects of the environment.

4.1 Curiosity-Driven Exploration

Our agent is composed of two subsystems: a reward generator that outputs a curiosity-driven intrinsic reward signal and a policy that outputs a sequence of actions to maximize that reward signal. In addition to intrinsic rewards, the agent optionally may also receive some extrinsic reward from the environment. Let the intrinsic curiosity reward generated by the agent at time t be r_t^i and the extrinsic reward be r_t^e. The policy sub-system is trained to maximize the sum of these two rewards $r_t = r_t^i + r_t^e$, with r_t^e mostly (if not always) zero.

We represent the policy $\pi(s_t; \theta_P)$ by a deep neural network with parameters θ_P. Given the agent in state s_t, it executes the action $a_t \sim \pi(s_t; \theta_P)$ sampled from the policy. θ_P is optimized to maximize the expected sum of rewards,

$$\max_{\theta_P} \mathbb{E}_{\pi(s_t;\theta_P)}[\Sigma_t r_t] \tag{4.1}$$

Unless specified otherwise, we use the notation $\pi(s)$ to denote the parameterized policy $\pi(s; \theta_P)$. Our curiosity reward model can potentially be used with a range of policy learning methods; in the experiments discussed here, we use the asynchronous advantage actor critic policy gradient (A3C) [154] for policy learning. Our main contribution is in designing an intrinsic reward signal based on prediction error of the agent's knowledge about its

environment that scales to high-dimensional continuous state spaces like images, bypasses the hard problem of predicting pixels and is unaffected by the unpredictable aspects of the environment that do not affect the agent.

4.1.1 Prediction error as curiosity reward

Making predictions in the raw sensory space (e.g. when s_t corresponds to images) is undesirable not only because it is hard to predict pixels directly, but also because it is unclear if predicting pixels is even the right objective to optimize. To see why, consider using prediction error in the pixel space as the curiosity reward. Imagine a scenario where the agent is observing the movement of tree leaves in a breeze. Since it is inherently hard to model breeze, it is even harder to predict the pixel location of each leaf. This implies that the pixel prediction error will remain high and the agent will always remain curious about the leaves. But the motion of the leaves is inconsequential to the agent and therefore its continued curiosity about them is undesirable. The underlying problem is that the agent is unaware that some parts of the state space simply cannot be modeled and thus the agent can fall into an artificial curiosity trap and stall its exploration. Novelty-seeking exploration schemes that record the counts of visited states in a tabular form (or their extensions to continuous state spaces) also suffer from this issue. Measuring learning progress instead of prediction error has been proposed in the past as one solution [216]. Unfortunately, there are currently no known computationally feasible mechanisms for measuring learning progress.

If not the raw observation space, then what is the right feature space for making predictions so that the prediction error provides a good measure of curiosity? To answer this question, let us divide all sources that can modify the agent's observations into three cases: (1) things that can be controlled by the agent; (2) things that the agent cannot control but that can affect the agent (e.g. a vehicle driven by another agent), and (3) things out of the agent's control and not affecting the agent (e.g. moving leaves). A good feature space for curiosity should model (1) and (2) and be unaffected by (3). This latter is because, if there is a source of variation that is inconsequential for the agent, then the agent has no incentive to know about it.

4.1.2 Self-supervised prediction for exploration

Instead of hand-designing features for every environment, we propose a general mechanism for learning features for prediction error based curiosity. Given the raw state s_t, we encode it using a deep neural network into a feature vector $\phi(s_t; \theta_E)$, denoted as $\phi(s_t)$ for succinctness. We propose to learn the parameters of this feature encoder using two sub-modules described as follows. The first sub-module is the neural network g which takes the feature encoding $\phi(s_t), \phi(s_{t+1})$ of two consequent states as input and predicts the action a_t taken by the agent to move from state s_t to s_{t+1}, defined as:

$$\hat{a}_t = g\Big(\phi(s_t), \phi(s_{t+1}); \theta_I\Big) \tag{4.2}$$

where, $\hat{a}_t$ is the predicted estimate of the action a_t. The neural network parameters θ_I, θ_E are trained to optimize,

$$\min_{\theta_I, \theta_E} L_I(\hat{a}_t, a_t) \tag{4.3}$$

where, L_I measures the discrepancy between the predicted and actual actions. L_I is modeled as soft-max loss across all possible actions when a_t is discrete. The learned function g is also known as the inverse dynamics model and the tuple (s_t, a_t, s_{t+1}) required to learn g is obtained while the agent interacts with the environment using its current policy $\pi(s)$.

Simultaneously with the inverse model g, we train another sub-module that takes as inputs a_t and $\phi(s_t)$ to predict the feature encoding of the state at time step $t+1$,

$$\hat{\phi}(s_{t+1}) = f\Big(\phi(s_t), a_t; \theta_F\Big) \tag{4.4}$$

where $\hat{\phi}(s_{t+1})$ is the predicted estimate of $\phi(s_{t+1})$. The function f is also known as the forward dynamics model and is trained to optimize the regression loss,

$$\min_{\theta_F, \theta_E} L_F\Big(\hat{\phi}(s_{t+1}), \phi(s_{t+1})\Big) \tag{4.5}$$

Finally, the intrinsic reward signal r_t^i is computed as,

$$r_t^i = \frac{\eta}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2 \tag{4.6}$$

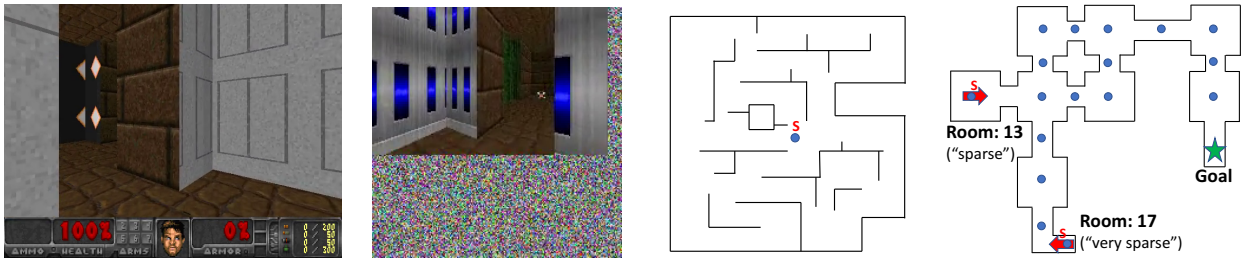
where $\eta > 0$ is a scaling factor. The inverse and forward dynamics losses, described in equations (4.3) and (4.5), are jointly optimized with the policy. The inverse model helps learn a feature space that encodes information relevant for predicting the agent's actions only and the forward model makes this learned feature representation more predictable. We refer to this proposed curiosity formulation as Intrinsic Curiosity Module (ICM). As there is no incentive for this feature space to encode any environmental features that are not influenced by the agent's actions, our agent will receive no rewards for reaching environmental states that are inherently unpredictable and its exploration strategy will be robust to nuisance sources of variation in the environment. See Figure 4.2 for illustration of the formulation.

The overall optimization problem can be written as,

$$\min_{\theta_P, \theta_I, \theta_F, \theta_E} \Bigg[-\lambda \mathbb{E}_{\pi(s_t;\theta_P)}[\Sigma_t r_t] + (1-\beta) L_I + \beta L_F \Bigg] \tag{4.7}$$

where $0 \leq \beta \leq 1$ is a scalar that weighs the inverse model loss against the forward model loss and $\lambda > 0$ weighs the importance of the policy gradient loss against the intrinsic reward signal. We do not backpropagate the policy gradient loss to the forward model to prevent degenerate solution of agent rewarding itself.

Previous work has investigated inverse models to learn features [4, 5, 111] and forward models to regularize those features [5] for recognition tasks. However, they do not learn any policy for the agent.



(a) Input (b) Input w/ noise (c) Train Map (d) Test Map

Figure 4.3: Frames from VizDoom 3-D environment which agent takes as input: (a) Usual 3-D navigation setup; (b) Setup when uncontrollable noise is added to the input. Maps for VizDoom 3-D environment (not shown to the agent): (c) For generalization experiments (c.f. Section 4.2.4), map of the environment where agent is pre-trained only using curiosity signal without any reward from environment. 'S' denotes the starting position. (d) Testing map for VizDoom experiments. Green star denotes goal location. Blue dots refer to 17 agent spawning locations in the map in the "dense" case. Rooms 13, 17 are the fixed start locations of agent in "sparse" and "very sparse" reward cases respectively. Note that textures are also different in train and test maps.

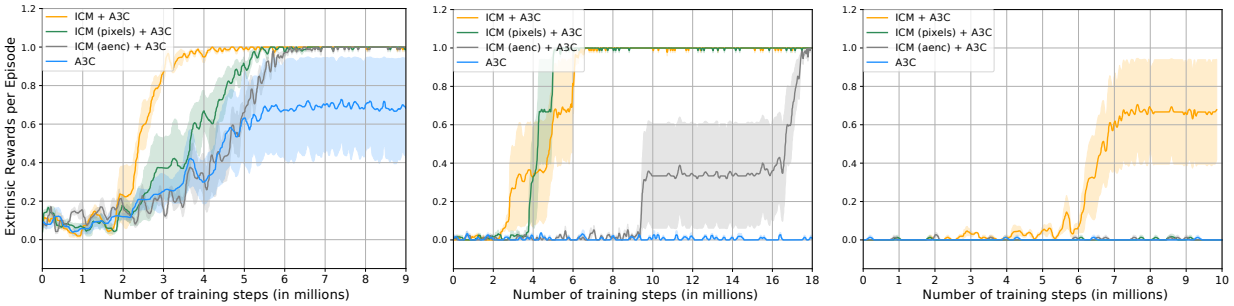
4.2 Experiments

Three broad settings are evaluated: a) sparse extrinsic reward on reaching a goal (Section 4.2.2); b) exploration with no extrinsic reward (Section 4.2.3); and c) generalization to novel scenarios (Section 4.2.4). Generalization is evaluated on a novel map with novel textures in *VizDoom* and on subsequent game levels in *Mario*.

4.2.1 Experimental Setup

Environments Our first environment is the VizDoom [117] game where we consider the 3D navigation task with four discrete actions – forward, left, right, and no-action. Our testing setup in all the experiments is the 'DoomMyWayHome-v0' environment which is available as part of OpenAI Gym [28]. The map consists of 9 rooms connected by corridors and the agent is tasked to reach some fixed goal location from its spawning location. Episodes are terminated either when the agent reaches the fixed goal or if the agent exceeds a maximum of 2100 time steps. The agent is only provided a sparse terminal reward of +1 if it finds the vest and zero otherwise. For generalization experiments, we pre-train on a different map with different random textures from [55] with 2100 step long episodes as there is no goal in pre-training. Sample frames and maps of VizDoom are shown in Figure 4.3. It takes approximately 350 steps for an optimal policy to reach the vest location from the farthest room in this map (sparse reward).

Our second environment is the classic Nintendo game Super Mario Bros with a reparamterized 14 dimensional action space following [178]. The actual game is played using a joystick allowing for multiple simultaneous button presses, where the duration of the press



(a) "dense reward" setting (b) "sparse reward" setting (c) "very sparse reward" setting

Figure 4.4: Comparing the performance of the A3C agent with no curiosity (blue) against the curiosity in pixel space agent (green) and the proposed curious ICM-A3C agent (orange) as the hardness of the exploration task is gradually increased from left to right. Exploration becomes harder with larger distance between the initial and goal locations: "dense", "sparse" and "very sparse". The results depict that succeeding on harder exploration task becomes progressively harder for the baseline A3C, whereas the curious A3C is able to achieve good score in all the scenarios. Pixel based curiosity works in dense and sparse but fails in very sparse reward setting. The protocol followed in the plots involves running three independent runs of each algorithm. Darker line represents mean and shaded area represents mean ± standard error of mean. We did not perform any tuning of random seeds.

affects what action is being taken. This property makes the game particularly hard, e.g. to make a long jump over tall pipes or wide gaps, the agent needs to predict the same action up to 12 times in a row, introducing long-range dependencies.

Baselines We compare our model ('ICM + A3C') against (a) vanilla 'A3C' with ϵ-greedy exploration; (b) 'ICM-pixels + A3C' where we predict the next observation in the pixel space instead of the feature space of the inverse model. The performance comparison between 'ICM-pixels + A3C' and 'ICM + A3C' is indicative of pros/cons of our method over established methods of computing curiosity reward by making predictions in observation space [217, 234]; (c) comparison with state-of-the-art exploration methods based on variational information maximization (VIME) [103].

4.2.2 Sparse Extrinsic Reward Setting

In the 'DoomMyWayHome-v0' 3D navigation setup (see section 4.2.1), the agent is provided with a sparse extrinsic reward only when it reaches the goal located at a fixed location. We systematically varied the difficulty of this task and constructed "dense", "sparse" and "very-sparse" reward (see Figure 4.3d) scenarios by varying the distance between the initial spawning location of the agent and the location of the goal. In the "dense" reward case, the agent is randomly spawned in any of the 17 spawning locations uniformly distributed across the map. This is not a hard exploration task because sometimes the agent is randomly

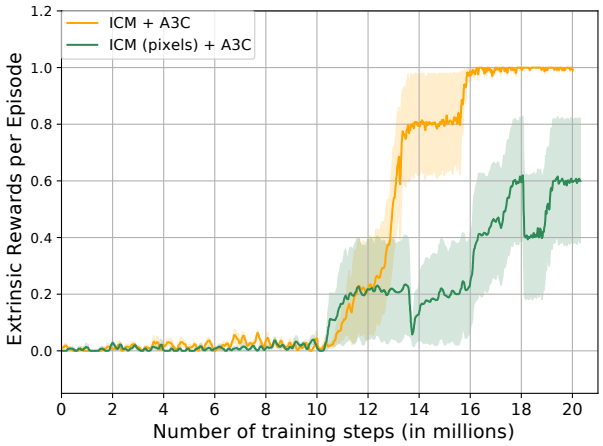


Figure 4.5: Evaluating the robustness of ICM to the presence of uncontrollable distractors in the environment. We created such a distractor by replacing 40% of the visual observation of the agent by white noise (see Figure 4.3b). The results show that while ICM succeeds most of the times, the pixel prediction model struggles.

initialized close to the goal and therefore by random ϵ-greedy exploration it can reach the goal with reasonably high probability. In the "sparse" and "very sparse" reward cases, the agent is always spawned in Room-13 and Room-17 respectively which are 270 and 350 steps away from the goal under an optimal policy. A long sequence of directed actions is required to reach the goals from these rooms, making these settings hard goal directed exploration problems.

Results in Figure 4.4 show that curious agents learn much faster indicating that their exploration is more effective in compared to ϵ-greedy exploration of the baseline agent. One possible explanation of the inferior performance of ICM-pixels in comparison to ICM is that in every episode the agent is spawned in one out of seventeen rooms with different textures. It is hard to learn a pixel-prediction model as the number of textures increases.

In the "sparse" reward case, as expected, the baseline A3C agent fails to solve the task, while the curious A3C agent is able to learn the task quickly. Note that ICM-pixels and ICM have similar convergence because, with a fixed spawning location of the agent, the ICM-pixels encounters the same textures at the starting of each episode which makes learning the pixel-prediction model easier as compared to the "dense" reward case. Finally, in the "very sparse" reward case, both the A3C agent and ICM-pixels never succeed, while the ICM agent achieves a perfect score in 66% of the random runs. This indicates that ICM is better suited than ICM-pixels and vanilla A3C for hard goal directed exploration tasks.

Robustness to uncontrollable dynamics For testing this, we augmented the agent's observation with a fixed region of white noise which made up 40% of the image (see Figure 4.3b) and evaluated on "sparse" reward setup of *VizDoom*. In navigation, ideally the agent should

be unaffected by this noise as the noise does not affect the agent in anyway and is merely a nuisance. Figure 4.5 shows that while the proposed ICM agent achieves a perfect score, ICM-pixels suffers significantly despite having succeeded at the "sparse reward" task when the inputs were not augmented with any noise (see Figure 4.4b). This indicates that in contrast to ICM-pixels, ICM is insensitive to nuisance changes in the environment.

Comparison to other baselines One possible reason for superior performance of the curious agent is that the intrinsic reward signal is simply acting as a regularizer by providing random rewards that push the agent out of the local minima. We systematically tested this hypothesis using many different random reward distributions on the "sparse VizDoom" task and found that with just random rewards the agents fail on sparse reward tasks. Comparison to the state of the art TRPO-VIME [103] agent in the table below shows that the ICM agent is superior in performance. The hyper-parameters and accuracy for TRPO and VIME agents follow from the concurrent work [71].

Method ("sparse" reward setup)	Mean (Median) Score (at convergence)
TRPO	26.0 % (0.0 %)
A3C	0.0 % (0.0 %)
VIME + TRPO	46.1 % (27.1 %)
ICM + A3C	**100.0** % (100.0 %)

4.2.3 No Reward Setting

For investigating how well does the ICM agent explore the environment, we trained it on *VizDoom* and *Mario* without any rewards from the environment. We then evaluated how much of the map was visited in *VizDoom* and how much progress the agent made on *Mario*. To our surprise, we have found that in both cases, the no-reward agent was able to perform quite well (see video at http://pathak22.github.io/noreward_rl/).

VizDoom: Coverage during Exploration. An agent trained with no extrinsic rewards was able to learn to navigate corridors, walk between rooms, and explore many rooms in the 3D Doom environment. On many occasions, the agent traversed the entire map and reached rooms that were farthest away from the room it was initialized in. Given that the episode terminates in 2100 steps and farthest rooms are over 250 steps away (for an optimally-moving agent), this result is quite remarkable, demonstrating that it is possible to learn useful skills without the requirement of any external supervision of rewards. Example explorations are shown in Figure 4.6. The first 3 maps show our agent explores a much larger state space without any extrinsic signal, compared to a random exploration agent (last 2 maps).

Mario: Learning to play with no rewards. Without any extrinsic reward from environment, our Mario agent can learn to cross over 30% of Level-1. The agent received no reward for killing or dodging enemies or avoiding fatal events, yet it automatically discovered

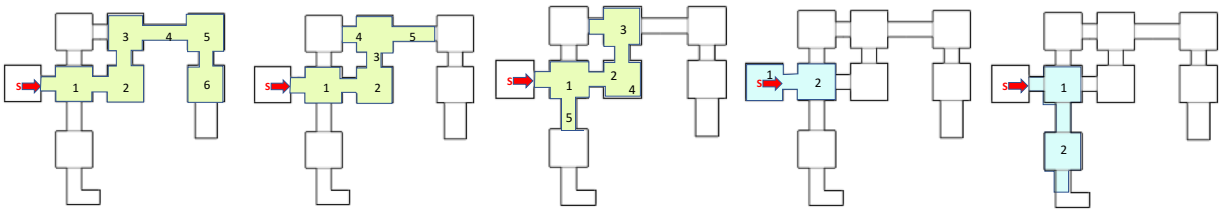


Figure 4.6: Each column in the figure shows the coverage of an agent by coloring the rooms it visits during 2100 steps of exploration. The red arrow shows the initial location and orientation of the agent at the start of the episode. The first three (in green) and the last two columns (in blue) show visitation of curious (ICM) and randomly exploring agents respectively. The results clearly show that the curious agent trained with intrinsic rewards explores a significantly larger number of rooms as compared to a randomly exploring agent.

these behaviors (see video). One possible reason is that getting killed by the enemy will result in only seeing a small part of the game space, making its curiosity saturate. In order to remain curious, it is in the agent's interest to learn how to kill and dodge enemies so that it can reach new parts of the game space. This suggests that curiosity provides indirect supervision for learning interesting behaviors.

To the best of our knowledge, this is the first work to show that the agent learns to navigate a 3D environment and discovers how to play a game directly from pixels without any extrinsic reward. Prior works [152, 154] have trained agents for navigation and ATARI games from pixels, but using rewards from environment.

4.2.4 Generalization to Novel Scenarios

In the previous section, we showed that our agent learns to explore large parts of the space where its curiosity-driven exploration policy was trained. However it remains unclear, when exploring a space, how much of the learned behavior is specific to that particular space and how much is general enough to be useful in novel scenarios? To investigate this question, we train a no reward exploratory behavior in one scenario (e.g. Level-1 of Mario) and then evaluate the resulting exploration policy in three different ways: a) apply the learned policy "as is" to a new scenario; b) adapt the policy by fine-tuning with curiosity reward only; c) adapt the policy to maximize some extrinsic reward. Happily, in all three cases, we observe some promising generalization results:

Evaluate "as is": The distance covered by the agent on Levels 1, 2, and 3 when the policy learned by maximizing curiosity on Level-1 of *Mario* is executed without any adaptation is reported in Table 4.1. The agent performs surprisingly well on Level 3, suggesting good generalization, despite the fact that Level-3 has different structures and enemies compared to Level-1. However, note that the running "as is" on Level-2 does not do well. At first, this seems to contradict the generalization results on Level-3. However, note that Level-3 has

Level Ids	Level-1	Level-2				Level-3			
Accuracy Iterations	Scratch 1.5M	Run as is 0	Fine-tuned 1.5M	Scratch 1.5M	Scratch 3.5M	Run as is 0	Fine-tuned 1.5M	Scratch 1.5M	Scratch 5.0M
Mean ± stderr	711 ± 59.3	31.9 ± 4.2	466 ± 37.9	399.7 ± 22.5	455.5 ± 33.4	319.3 ± 9.7	97.5 ± 17.4	11.8 ± 3.3	42.2 ± 6.4
% distance > 200	50.0 ± 0.0	0	64.2 ± 5.6	88.2 ± 3.3	69.6 ± 5.7	50.0 ± 0.0	1.5 ± 1.4	0	0
% distance > 400	35.0 ± 4.1	0	63.6 ± 6.6	33.2 ± 7.1	51.9 ± 5.7	8.4 ± 2.8	0	0	0
% distance > 600	35.8 ± 4.5	0	42.6 ± 6.1	14.9 ± 4.4	28.1 ± 5.4	0	0	0	0

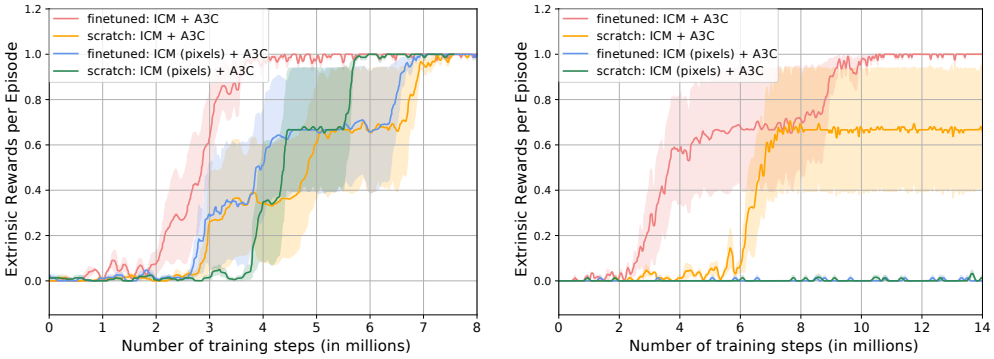
Table 4.1: Quantitative evaluation of the agent trained to play Super Mario Bros. using only curiosity signal without any rewards from the game. The policy trained on Level-1 is evaluated both when it is is run "as is", and further fine-tuned on subsequent levels. The results are compared to settings when Mario agent is trained from scratch in Level-2,3 using only curiosity without any extrinsic rewards. Evaluation metric is based on the distance covered by the Mario agent.

similar global visual appearance (day world with sunlight) to Level-1, whereas Level-2 is significantly different (night world). If this is indeed the issue, then it should be possible to quickly adapt the agent's exploration policy to Level-2 with a little bit of "fine-tuning".

Fine-tuning with curiosity only: From Table 4.1, we see that when the agent pre-trained (using only curiosity as reward) on Level-1 is fine-tuned (using only curiosity as reward) on Level-2 it quickly overcomes the mismatch in global visual appearance and achieves a higher score than training from scratch with the same number of iterations. Interestingly, training "from scratch" on Level-2 is worse than the fine-tuned policy, even when training for more iterations than pre-training + fine-tuning combined. One possible reason is that Level-2 is more difficult than Level-1, so learning the basic skills such as moving, jumping, and killing enemies from scratch is harder than in the relative "safety" of Level-1. This result, therefore, might suggest that first pre-training on an earlier level and then fine-tuning on a later one produces a form of curriculum which aids learning and generalization. In other words, the agent is able to use the knowledge it acquired by playing Level-1 to better explore the subsequent levels. Of course, the game designers do this on purpose to allow the human players to gradually learn to play the game.

However, interestingly, fine-tuning the exploration policy pre-trained on Level-1 to Level-3 deteriorates the performance, compared to running "as is". This is because Level-3 is very hard for the agent to cross beyond a certain point – the agent hits a curiosity blockade and is unable to make any progress. As the agent has already learned about parts of the environment before the hard point, it receives almost no curiosity reward and as a result it attempts to update its policy with almost zero intrinsic rewards and the policy slowly degenerates. This behavior is vaguely analogous to boredom, where if the agent is unable to make progress it gets bored and stops exploring.

Fine-tuning with extrinsic rewards: We first pre-trained an agent on *VizDoom* using only curiosity reward on the map shown in Figure 4.3c. We then test on the "sparse" and "very sparse" reward settings of 'DoomMyWayHome-v0' environment which uses a different



(a) "sparse reward" setting (b) "very sparse reward" setting

Figure 4.7: Curiosity pre-trained ICM + A3C when finetuned on the test map with environmental rewards outperforms ICM + A3C trained from scratch using both environmental and curiosity reward on the "sparse" (left) and "very sparse" (right) reward settings of *VizDoom*. The pixel prediction based ICM agent completely fails indicating that our curiosity formulation learns generalizable exploration policies.

map with novel textures (see Figure 4.3d). Results in Figure 4.7 show that the curiosity pre-trained ICM agent when fine-tuned with external rewards learns faster and achieves higher reward than an ICM agent trained from scratch to jointly maximize curiosity and the external rewards. This result confirms that the learned exploratory behavior is also useful when the agent is required to achieve goals in a new environment. It is also worth noting that ICM-pixels does not generalize to the test environment. This indicates that the proposed mechanism of measuring curiosity is significantly better for learning skills that generalize as compared to measuring curiosity in the raw sensory space.

4.3 Large-Scale Study of Curiosity-Driven Learning

We showed that our prediction-error based curiosity formulation allows the agent to learn navigation skills, and play games, e.g., Super Mario Bros., without using any game score. More importantly, a curious agent is able to *generalize* its skills from one game level to learn faster in the subsequent ones. This work was among the first ones to show exploration-driven deep reinforcement learning without using any rewards from the environment. To demonstrate its generality, we now perform a large-scale systematic study of learning with only intrinsic rewards.

We experiment with agents driven purely by intrinsic rewards across a range of 54 diverse environments: video games, physics engine simulations, and virtual 3D navigation tasks, shown in Figure 4.8. To develop a better understanding of curiosity-driven learning, we further study the crucial factors that determine its performance. In particular, predicting the future state in the high dimensional raw observation space (e.g., images) is a challenging

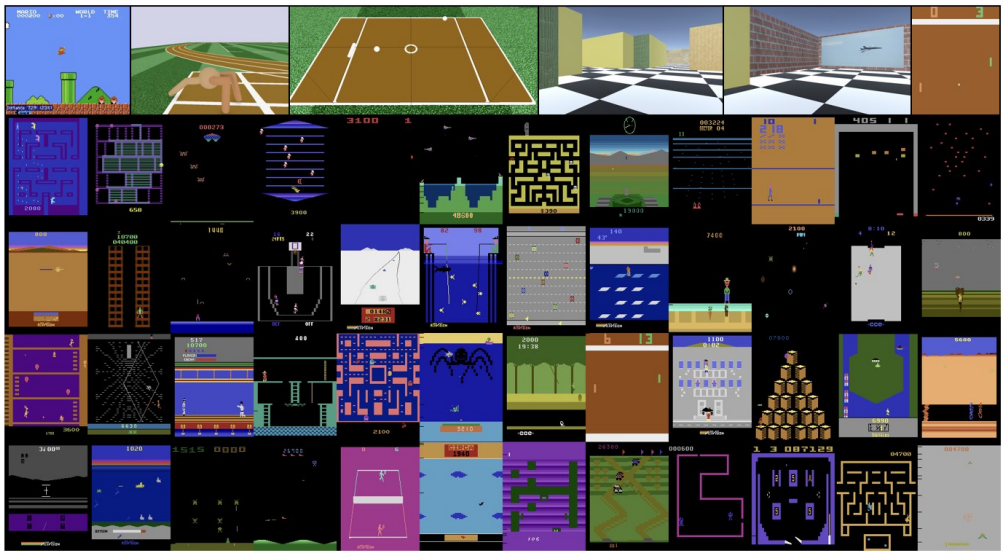


Figure 4.8: A snapshot of the 54 environments investigated in this work. We show that agents are able to make progress using no extrinsic reward, or end-of-episode signal, and only using curiosity. Video results, code and models at https://pathak22.github.io/large-scale-curiosity/.

problem and we saw that learning dynamics in an auxiliary feature space leads to improved results. However, how one chooses such an embedding space is a critical, yet open research problem. To ensure stable online training of dynamics, we argue that the desired embedding space should: 1) be *compact* in terms of dimensionality, 2) preserve *sufficient* information about the observation, and 3) be a *stationary* function of the observations.

Through systematic ablation, we examine the role of different ways to encode agent's observation such that an agent can perform well, driven purely by its own curiosity. Here "performing well" means acting purposefully and skillfully in the environment. This can be assessed quantitatively, in some cases, by measuring extrinsic rewards or environment-specific measures of exploration, or qualitatively, by observing videos of the agent interacting. We show that encoding observations via a random network turn out to be a simple, yet surprisingly effective technique for modeling curiosity across many popular RL benchmarks. This might suggest that many popular RL video game test-beds are not as visually sophisticated as commonly thought. Interestingly, we discover that although random features are sufficient for good performance in environments that were used for training, the learned features appear to generalize better (e.g., to novel game levels in Super Mario Bros.).

4.3.1 Feature spaces for forward dynamics

Consider the representation ϕ in the curiosity formulation above. If $\phi(x) = x$, the forward dynamics model makes predictions in the observation space. A good choice of feature space can make the prediction task more tractable and filter out irrelevant aspects of the observation

space. But what makes a good feature space for dynamics driven curiosity? We propose the qualities that a good feature space must have:

- *Compactness*: The features should be easy to model by being low(er)-dimensional and filtering out irrelevant parts of the observation space.
- *Sufficiency*: The features should contain all the important information. Otherwise, the agent may fail to be rewarded for exploring some relevant aspect of the environment.
- *Stability*: Non-stationary rewards make it difficult for reinforcement agents to learn. Exploration bonuses by necessity introduce non-stationarity since what is new and novel becomes old and boring with time. In our curiosity formulation, there are two sources of non-stationarity: the forward dynamics model is evolving over time as it is trained and the features are changing as they learn. The former is intrinsic to the method, and the latter should be minimized where possible

In this section, we systematically investigate the efficacy of a number of feature-learning methods, summarized briefly as follows:

Pixels The simplest case is where $\phi(x) = x$ and we fit our forward dynamics model in the observation space. Pixels are sufficient, since no information has been thrown away, and stable since there is no feature learning component. However, learning from pixels is tricky because the observation space may be high-dimensional and complex.

Random Features (RF) The next simplest case is where we take our embedding network, a convolutional network, and fix it after random initialization. Because the network is fixed, the features are stable. The features can be made compact in dimensionality, but they are not constrained to be. However, random features may fail to be sufficient.

Variational Autoencoders (VAE) VAEs were introduced in [120, 201] to fit latent variable generative models $p(x, z)$ for observed data x and latent variable z with prior $p(z)$ using variational inference. The method calls for an inference network $q(z|x)$ that approximates the posterior $p(z|x)$. This is a feedforward network that takes an observation as input and outputs a mean and variance vector describing a Gaussian distribution with diagonal covariance. We can then use the mapping to the mean as our embedding network ϕ. These features will be a low-dimensional approximately sufficient summary of the observation, but they may still contain some irrelevant details such as noise, and the features will change over time as the VAE trains.

	VAE	IDF	RF	Pixels
Stable	No	No	Yes	Yes
Compact	Yes	Yes	Maybe	No
Sufficient	Yes	Maybe	Maybe	Yes

Table 4.2: Table summarizing the categorization of different kinds of feature spaces considered.

Inverse Dynamics Features (IDF) Given a transition (s_t, s_{t+1}, a_t) the inverse dynamics task is to predict the action a_t given the previous and next states s_t and s_{t+1}. Features are learned using a common neural network ϕ to first embed s_t and s_{t+1}. The intuition is that the features learned should correspond to aspects of the environment that are under the agent's immediate control. This feature learning method is easy to implement and in principle should be invariant to certain kinds of noise as discussed earlier. A potential downside could be that the features learned may not be sufficient, that is they do not represent important aspects of the environment that the agent cannot immediately affect.

A summary of these characteristics is provided in Table 4.2. Note that the learned features are not stable because their distribution changes as learning progresses. One way to achieve stability could be to pre-train VAE or IDF networks. However, unless one has access to the internal state of the game, it is not possible to get a representative data of the game scenes to train the features. One way is to act randomly to collect data, but then it will be biased to where the agent started, and won't generalize further. Since all the features involve some trade-off of desirable properties, it becomes an empirical question as to how effective each of them is across environments.

4.3.2 Practical considerations in training with only curiosity

Deciding upon a feature space is only first part of the puzzle in implementing a practical system. Here, we detail the critical choices we made in the learning algorithm. Our goal is to reduce non-stationarity in order to make learning more stable and consistent across environments. Through the following considerations outlined below, we are able to get exploration to work reliably for different feature learning methods and environments with minimal changes to the hyper-parameters.

- *PPO.* In general, we have found the PPO algorithm [218] to be a robust learning algorithm that requires little hyper-parameter tuning and so we stick to it for our experiments.
- *Reward normalization.* Since the reward function is non-stationary, it is useful to normalize the scale of the rewards so that the value function can learn quickly. We did this by dividing the rewards by a running estimate of the standard deviation of the sum of discounted rewards.
- *Advantage normalization.* While training with PPO, we normalize the advantages [241] in a batch to have a mean of 0 and a standard deviation of 1.
- *Observation normalization.* We run a random agent on our target environment for 10000 steps, then calculate the mean and standard deviation of the observation and use these to normalize the observations when training. This is useful to ensure that the features do not have very small variance at initialization, and also ensure features have less variation across different environments.

- *More actors.* The stability of the method is greatly increased by increasing the number of parallel actors (which affects the batch-size) used. We typically use 128 parallel runs of the same environment for data collection while training an agent.
- *Normalizing the features.* In combining intrinsic and extrinsic rewards, we found it useful to ensure that the scale of the intrinsic reward was consistent across state space. We achieved this by using batch-normalization [108] in the feature embedding network.

4.3.3 'Death is not the end': infinite horizon

One important point is that the use of an end-of-episode signal, sometimes called 'done', can often leak information about the true reward function (assuming, as is common, that we have access to an extrinsic reward signal that we hide from the agent to measure pure exploration). If we don't remove the 'done' signal, many of the Atari games become too simple. For example, a simple strategy of giving +1 artificial reward at every time-step when the agent is alive and 0 upon death is sufficient to obtain a high score in some games, e.g. the Atari game 'Breakout' where it will seek to maximize the episode length and hence its score. In the case of negative rewards, the agent will try to end the episode as quickly as possible.

In light of this, if we want to study the behavior of a pure exploration agent, we should not bias it. In the infinite horizon setting (i.e., the discounted returns are not truncated at the end of the episode and always bootstrapped using the value function), death is just another transition to the agent, to be avoided only if it is "boring". Therefore, we removed 'done' to separate the gains of an agent's exploration from merely that of the death signal. In practice, we do find that the agent avoids dying in the games since that brings it back to the beginning of the game – an area it has already seen many times and where it can predict the dynamics well. This subtlety has been neglected by previous works showing experiments without extrinsic rewards.

4.4 Large-Scale Experiments

In all of our experiments, both the policy and the embedding network work directly from pixels. For our implementation details including hyper-parameters and architectures, please refer to the Appendix A.1. Unless stated otherwise, all curves are the average of three runs with different seeds, and the shaded areas are standard errors of the mean. We have released the code and videos of a purely curious agent playing across all environments on our website [1].

4.4.1 Curiosity-driven learning without extrinsic rewards

We begin by scaling up a pure curiosity-driven learning to a large number of environments without using any extrinsic rewards. We pick a total of 54 diverse simulated environments,

[1] https://pathak22.github.io/large-scale-curiosity/

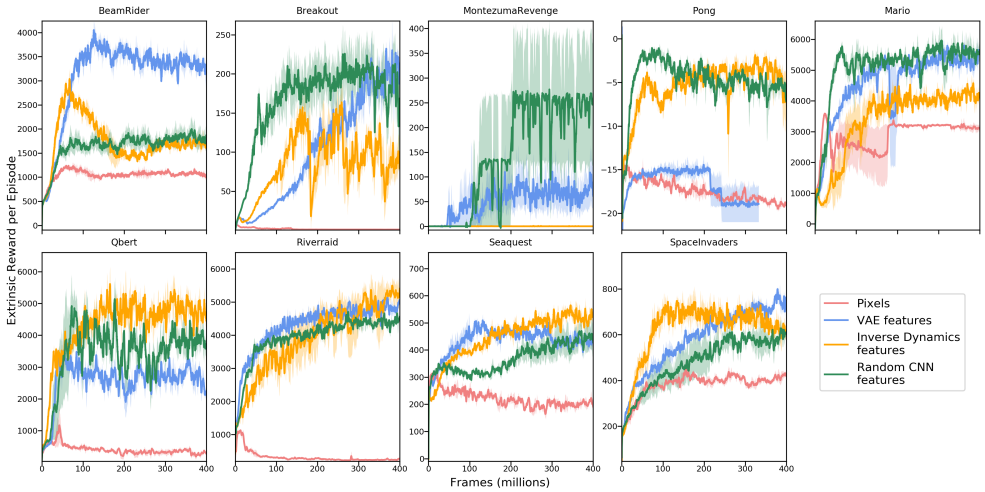


Figure 4.9: A comparison of feature learning methods on 8 selected Atari games and the Super Mario Bros. These evaluation curves show the mean reward (with standard error) of agents trained purely by curiosity, without reward or an end-of-episode signal. We see that our purely curiosity-driven agent is able to gather rewards in these environments without using any extrinsic reward at training. Results on all of the Atari games are in the appendix in Figure A.1. We find curiosity model trained on pixels does not work well across any environment and VAE features perform either same or worse than random and inverse dynamics features. Further, inverse dynamics-trained features perform better than random features in 55% of the Atari games. An interesting outcome of this analysis is that random features for modeling curiosity are a simple, yet surprisingly strong baseline and likely to work well in half of the Atari games.

as shown in Figure 4.8, including 48 Atari games, Super Mario Bros., 2 Roboschool scenarios (learning Ant controller and Juggling), Two-player Pong, 2 Unity mazes (with and without a TV controlled by the agent). The goal of this large-scale analysis is to investigate the following questions: (a) What happens when you run a pure curiosity-driven agent on a variety of games without any extrinsic rewards? (b) What kinds of behaviors can you expect from these agents? (c) What is the effect of the different feature-learning variants in our curiosity formulation on these behaviors?

Atari Games To answer these questions, we began with a collection of well-known Atari games and ran a suite of experiments with different feature-learning methods. One way to measure how well a purely curious agent performs is to measure the extrinsic reward it is able to achieve, i.e. how good is the agent at playing the game. We show the evaluation curves of mean extrinsic reward in on 8 common Atari games in Figure 4.9 and all 48 Atari suite in Figure A.1 in the appendix. It is important to note that the extrinsic reward is only used for evaluation, not for training. However, this is just a proxy for pure exploration because the game rewards could be arbitrary and might not align at all with how the agent explores out of curiosity.

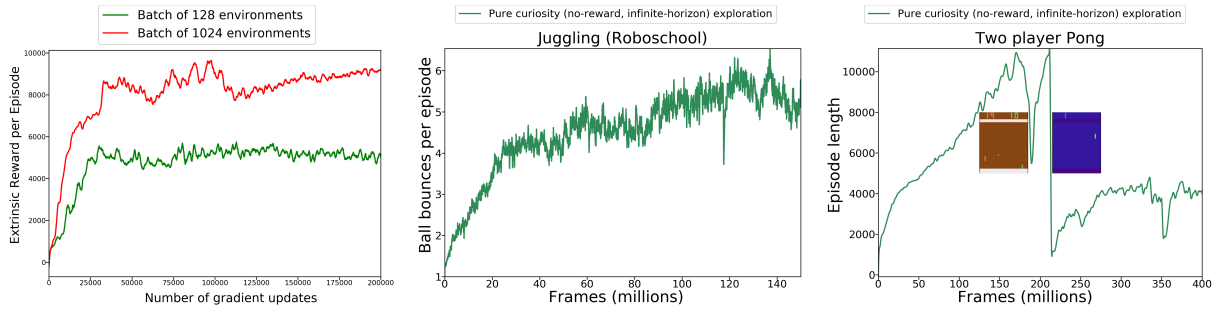
The first thing to notice from the curves is: *most of them are going up.* This shows that a pure curiosity-driven agent can often learn to obtain external rewards without seeing any extrinsic rewards during training! To understand why this is happening, consider the game 'Breakout'. The main control action of the game is to keep hitting the bouncing ball with the paddle, but this does not earn any points. The game score increases only when the ball hits a brick (which then disappears). But the more bricks are struck by the ball, the more complicated the pattern of remaining bricks becomes, making the agent more curious to explore further, hence, collecting points as a bi-product. Furthermore, when the agent runs out of lives, the bricks are reset to the initial configuration, which has been seen by the agent many times before and is hence very predictable, so the agent tries to increase curiosity by staying alive and avoiding the death reset.

The fact that the curiosity reward is often sufficient is an unexpected result and might suggest that many popular RL test-beds do not need an external reward at all. It is likely that game designers (similar to architects, urban planners, gardeners, etc.) are purposefully setting up curricula to guide agents through the task by curiosity alone. This could explain why curiosity-like objective aligns reasonably well with the extrinsic reward in many human-designed environments [39, 106, 133, 263]. However, this is not always the case, and sometimes a curious agent can even do worse than a random agent. This happens when the extrinsic reward has little correlation with the agent's exploration, or when the agent fails to explore efficiently (e.g. see games 'Atlantis' and 'IceHockey' in Figure A.1). We encourage the reader to refer to the game-play videos of the agent available on the website for a better understanding of the learned skills.

Comparison of feature learning methods: We compare four feature learning methods in Figure 4.9: raw pixels, random features, inverse dynamics features and VAE features. Training dynamics on raw pixels performs poorly across all the environments, while encoding pixels into features does better. This is likely because it is hard to learn a good dynamics model in pixel space, and prediction errors may be dominated by small irrelevant details.

Surprisingly, random features (RF) perform quite well across tasks and sometimes better than using learned features. One reason for good performance is that the random features are kept frozen (stable), the dynamics model learned on top of them has an easier time because of the stationarity of the target. In general, random features should work well in the domains where visual observations are simple enough, and random features can preserve enough information about the raw signal, for instance, Atari games. One scenario where IDF features consistently outperform random features is for generalization, e.g. training on one level of Mario Bros and testing on another (see Section 4.4.2 for details).

The VAE method also performed well but was somewhat unstable, so we decided to use RF and IDF for further experiments. The detailed result in appendix Figure A.1 compares IDF vs. RF across the full Atari suite. To quantify the learned behaviors, we compared our curious agents to a randomly acting agent. We found that an IDF-curious agent collects more game reward than a random agent in 75% of the Atari games, an RF-curious agent does better in 70%. Further, IDF does better than RF in 55% of the games. Overall, random features and inverse dynamics features worked well in general. Further details in the appendix.



(a) Mario w/ large batch (b) Juggling (Roboschool) (c) Two-player Pong

Figure 4.10: (a) Left: A comparison of the RF method on Mario with different batch sizes. Results are without using extrinsic reward. (b) Center: Number of ball bounces in the Juggling (Roboschool) environment. (c) Right: Mean episode length in the multiplayer Pong environment. The discontinuous jump on the graph corresponds to the agent reaching a limit of the environment - after a certain number of steps in the environment the Atari Pong emulator starts randomly cycling through background colors and becomes unresponsive to agent's actions

Super Mario Bros. We compare different feature-learning methods in Mario Bros in Figure 4.9. We already studied Super Mario Bros in the context of extrinsic reward-free learning earlier in small-scale experiments, and now we were keen to see how far curiosity alone can push the agent. We used a more efficient version of the Mario simulator, allowing for longer training, while keeping observation space, actions, and dynamics of the game the same. Due to 100x longer training and using PPO for optimization, our agent was able to pass several levels of the game, significantly improving over prior exploration results on Mario Bros.

Could we further push the performance of a purely curious agent by making the underlying optimization more stable? One way is to scale up the batch-size. We do so by increasing the number of parallel threads for running environments from 128 to 1024. We show the comparison between training using 128 and 1024 parallel environment threads in Figure 4.10a. As apparent from the graph, training with large batch-size using 1024 parallel environment threads performs much better. In fact, the agent is able to explore much more of the game: *discovering 11 different levels of the game*, finding secret rooms and defeating bosses. Note that the x-axis in the figure is the number of gradient steps, not the number of frames, since the point of this large-scale experiment is not a claim about sample-efficiency, but performance with respect to training the agent. This result suggests that the performance of a purely curiosity-driven agent would improve as the training of base RL algorithm (PPO in our case) gets better. The video is on the website.

Roboschool Juggling We modified the Pong environment from the Roboschool framework to only have one paddle and to have two balls. The action space is continuous with two-dimensions, and we discretized the action space into 5 bins per dimension giving a total of 25 actions. Both the policy and embedding network are trained on pixel observation space

(note: not state space). This environment is more difficult to control than the toy physics used in games, but the agent learns to intercept and strike the balls when it comes into its area. We monitored the number of bounces of the balls as a proxy for interaction with the environment, as shown in Figure 4.10b. See the video on the project website.

Roboschool Ant Robot We also explored using the Ant environment which consists of an Ant with 8 controllable joints on a track. We again discretized the action space and trained policy and embedding network on raw pixels (not state space). However, in this case, it was less easy to measure exploration because the extrinsic distance reward measures progress along the racetrack, but a purely curious agent is free to move in any direction. We find that a walking like behavior emerges purely out of a curiosity-driven training. We refer the reader to the result video showing that the agent is meaningfully interacting with the environment.

Multi-agent curiosity in Two-player Pong We have already seen that a purely curiosity-driven agent learns to play several Atari games without reward, but we wonder how much of that behavior is caused by the fact that the opposing player is a computer agent with a hard-coded strategy. What would happen if we were to make both the players curious-driven? To find out, we set up a two-player Pong game where both the sides (paddles) of the game are controlled by two curiosity-driven agents. We shared the initial layers of both the agents but have different action heads, i.e., total action space is now the cross product of the actions of player 1 by the actions of player 2.

Note that the extrinsic reward is meaningless in this context since the agent is playing both sides, so instead, we show the length of the episode. The results are shown in Figure 4.10c. We see from the episode length that the agent learns to have longer rallies over time, learning to play pong without any teacher – purely by curiosity on both sides. In fact, *the game rallies eventually get so long that they break our Atari emulator* causing the colors to change radically, which crashes the policy as shown in the plot.

4.4.2 Generalization across novel levels in Super Mario Bros.

In the previous section, we showed that our purely curious agent can learn to explore efficiently and learn useful skills, e.g., game playing behaviour in games, walking behavior in Ant etc. So far, these skills were shown in the environment where the agent was trained on. However, one advantage of developing reward-free learning is that one should then be able to utilize abundant "unlabeled" environments without reward functions by showing generalization to novel environments.

To test this, we first pre-train our agent using curiosity only in the Level 1-1 of Mario Bros. We investigate how well RF and IDF-based curiosity agents generalize to novel levels of Mario. In Figure 4.11, we show two examples of training on one level of Mario and fine-tuning on another testing level, and compare to learning on the testing level from scratch. The

training signal in all the cases is curiosity-only reward. In the first case, from Level 1-1 to Level 1-2, the global statistics of the environments match (both are 'day-time' environments, i.e., blue sky) but levels have different enemies, different geometry, and different difficulty. We see that there is strong transfer from for both methods in this scenario. However, the transfer performance is weaker in the second scenario from Level 1-1 to Level 1-3. This is so because the problem is considerably harder for the latter level pairing as there is a color pallette shift from day to night, as shown in Figure 4.11.

We further note that IDF-learned features transfer in both the cases and random features transfer in the first case, but do not transfer in the second scenario from day to night. These results might suggest that while random features perform well on training environments, learned features appear to generalize better to novel levels. However, this needs more analysis in the future across a large variety of environments. Overall, we find some promising evidence showing that skills learned by curiosity help our agent explore efficiently in novel environments.

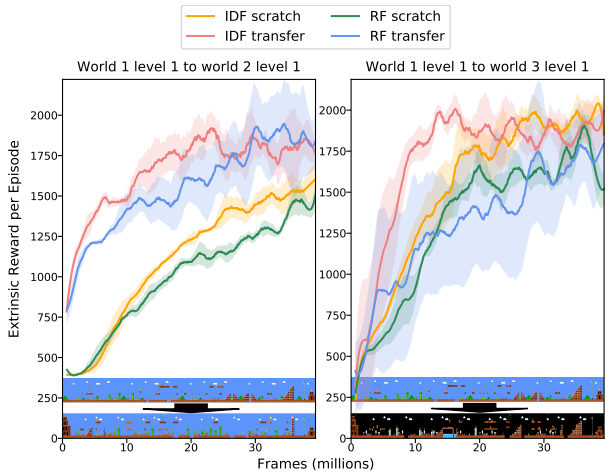


Figure 4.11: Mario generalization experiments. On the left we show transfer results from Level 1-1 to Level 1-2, and on the right we show transfer results from Level 1-1 to Level 1-3. Underneath each plot is a map of the source and target environments. All agents are trained without extrinsic reward.

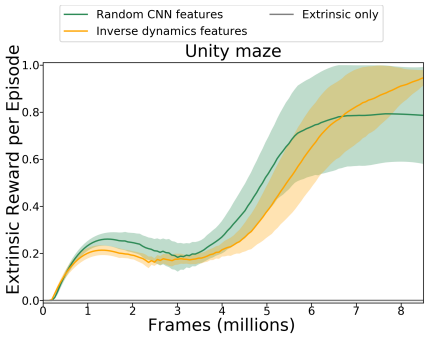


Figure 4.12: Mean extrinsic reward in the Unity environment while training with terminal extrinsic + curiosity reward. Note that the curve for extrinsic reward only training is constantly zero.

4.4.3 Curiosity with Sparse External Reward

In all our experiments so far, we have shown that our agents can learn useful skills without any extrinsic rewards, driven purely by curiosity. However, in many scenarios, we might want the agent to perform some particular task of interest. This is usually conveyed to the agent by defining extrinsic rewards. When rewards are dense (e.g. game score at every frame), classic RL works well and intrinsic rewards generally should not help performance. However,

designing dense rewards is a challenging engineering problem (see introduction for details). In this section, we evaluate how well curiosity can help an agent perform a task in presence of sparse, or just terminal, rewards.

Terminal reward setting: For many real problems, only terminal reward is available, e.g. in navigation, you only get rewards once you find what you were looking for. This is a setting where classic RL typically performs poorly. Hence, we consider the 3D navigation in a maze designed in the Unity ML-agent framework with 9 rooms and a sparse terminal reward. The action space is discrete, consisting of: move forward, look left 15 degrees, look right 15 degrees and no-op. The agent starts in room-1, which is furthest away from room-9 which contains the goal. We compare an agent trained with extrinsic reward (+1 when the goal is reached, 0 otherwise) to an agent trained with extrinsic + intrinsic reward. Extrinsic only (classic RL) never finds the goal in all our trials, which means it is impossible to get any meaningful gradients. Whereas extrinsic+intrinsic typically converges to getting the reward every time. Results in Figure 4.12 show results for vanilla PPO, PPO + IDF-curiosity and PPO + RF-curiosity.

Sparse reward setting: In preliminary experiments, we picked 5 Atari games which have sparse rewards (as categorized by [17]), and compared extrinsic (classic RL) vs. extrinsic+intrinsic (ours) reward performance. In 4 games out of 5, curiosity bonus improves performance (see Table A.1 in the appendix, the higher score is better). We would like to emphasize that this is not the focus here, and these experiments are provided just for completeness. We just combined extrinsic (coefficient 1.0) and intrinsic reward (coefficient 0.01) directly without any tuning. We leave the question on how to optimally combine extrinsic and intrinsic rewards as a future direction.

4.5 Related Work

Exploration is a well-studied problem in the field of reinforcement learning. Early approaches focused on studying exploration from theoretical perspective [239] and proposed Bayesian formulations [47, 125] but they are usually hard to scale to higher dimensions (e.g., images). In this chapter, we focus on the specific problem of exploration using intrinsic rewards.

Intrinsic Motivation: Classic work of Kearns *et al*. [116] and Brafman *et al*. [25] propose exploration algorithms polynomial in the number of state space parameters. Others have used empowerment, which is the information gain based on entropy of actions, as intrinsic rewards [122, 156]. A family of approaches to intrinsic motivation reward an agent based on prediction error [3, 216, 234], information gain [103, 143, 236, 237], or improvement [146, 215] of a forward dynamics model of the environment that gets trained along with the agent's policy. As a result the agent is driven to reach regions of the environment that are difficult to predict for the forward dynamics model, while the model improves its predictions in these regions. This adversarial and non-stationary dynamics can give rise to complex behaviors. Relatively little work has been done in this area on the pure exploration setting where there is no external reward. Of these mostly closely related are those that use a forward dynamics

model of a feature space such as Stadie *et al*. [234] where they use autoencoder features. Our approach of jointly training forward and inverse models for learning a feature space has similarities to [5, 113, 261], but these works use the learned models of dynamics for planning a sequence of actions instead of exploration. The idea of using a proxy task to learn a semantic feature embedding has been used in a number of works on self-supervised learning in computer vision [4, 51, 84, 111, 184, 254].

Smoothed versions of state visitation counts can be used for intrinsic rewards [17, 171, 242]. Count-based methods have shown strong results when combining with extrinsic rewards such as in the Atari game Montezuma's Revenge [17], and also showing significant exploration of the game without using the extrinsic reward. It is not yet clear in which situations count-based approaches should be preferred over dynamics-based approaches; we chose to focus on dynamics-based bonuses in this study since we found them straightforward to scale and parallelize. In our preliminary experiments, we did not have sufficient success with already existing count-based implementations in scaling up for a large-scale study.

Learning without extrinsic rewards or fitness functions has also been studied extensively in the evolutionary computing where it is referred to as 'novelty search' [136, 137, 235]. There the novelty of an event is often defined as the distance of the event to the nearest neighbor amongst previous events, using some statistics of the event to compute distances. One interesting finding from this literature is that often much more interesting solutions can be found by not solely optimizing for fitness.

Other methods of exploration are designed to work in combination with maximizing a reward function, such as those utilizing uncertainty about value function estimates [34, 169], or those using perturbations of the policy for exploration [69, 196]. Schmidhuber *et al*. [217] and Oudeyer *et al*. [173, 174] provide a great review of some of the earlier work on approaches to intrinsic motivation.

Random Features: One of the findings is the surprising effectiveness of random features, and there is a substantial literature on random projections and more generally randomly initialized neural networks. Much of the literature has focused on using random features for classification [110, 209, 268] where the typical finding is that whilst random features can work well for simpler problems, feature learning performs much better once the problem becomes sufficiently complex. Whilst we expect this pattern to also hold true for dynamics-based exploration, we have some preliminary evidence showing that learned features appear to generalize better to novel levels in Mario Bros.

Concurrent work: A number of interesting related papers have appeared in concurrent to this line of work. Sukhbaatar *et al*. [240] generates supervision for pre-training via asymmetric self-play between two agents to improve data efficiency during fine-tuning. Several methods propose improving data efficiency of RL algorithms using self-supervised prediction based auxiliary tasks [109, 224]. Fu *et al*. [71] learn discriminative models, and Gregor *et al*. [86] use empowerment,which is a measurement of the control an agent has over the state, to tackle exploration in sparse reward setups. However, none of these works show learning without extrinsic rewards or generalization of policy to novel scenarios. In [63], diversity is used as a measure to learn skills without reward functions.

4.6 Discussion

In this chapter, we proposed a mechanism for generating curiosity-driven intrinsic reward signal that scales to high dimensional visual inputs, bypasses the difficult problem of predicting pixels and ensures that the exploration strategy of the agent is unaffected by nuisance factors in the environment.

We have shown that our agents trained purely with a curiosity reward are able to learn useful behaviours: (a) Agent being able to play many Atari games without using any rewards. (b) Mario being able to cross over 11 levels without any extrinsic reward. (c) Walking-like behavior emerged in the Ant environment. (d) Juggling-like behavior in Robo-school environment (e) Rally-making behavior in Two-player Pong with curiosity-driven agent on both sides. But this is not always true as there are some Atari games where exploring the environment does not correspond to extrinsic reward. More generally, our results suggest that, in many game environments designed by humans, the extrinsic reward is often aligned with the objective of seeking novelty. The game designers seem to set up curricula to guide users while playing the game explaining the reason Curiosity-like objective decently aligns with the extrinsic reward in many human-designed games [39, 106, 133, 263].

Generalization: It is common practice to evaluate reinforcement learning approaches in the same environment that was used for training. However, we feel that it is also important to evaluate on a separate "testing set" as well. This allows us to gauge how much of what has been learned is specific to the training environment (i.e. memorized), and how much might constitute "generalizable skills" that could be applied to new settings. We evaluate generalization in two ways: 1) by applying the learned policy to a new scenario "as is" (no further learning), and 2) by fine-tuning the learned policy on a new scenario (we borrow the pre-training/fine-tuning nomenclature from the deep feature learning literature). We believe that evaluating generalization is a valuable tool and will allow the community to better understand the performance of various reinforcement learning algorithms. To further aid in this effort, we have made the code for our algorithm, as well as testing and environment setups freely available online.

Limitation of prediction error based curiosity: A more serious potential limitation is the handling of stochastic dynamics. If the transitions in the environment are random, then even with a perfect dynamics model, the expected reward will be the entropy of the transition, and the agent will seek out transitions with the highest entropy. Even if the environment is not truly random, unpredictability caused by a poor learning algorithm, an impoverished model class or partial observability can lead to exactly the same problem. We did not observe this effect in our experiments on games so we designed an environment to illustrate the point.

We return to the maze of Section 4.4.3 to empirically validate a common thought experiment called the noisy-TV problem. The idea is that local sources of entropy in an environment like a TV that randomly changes channels when an action is taken should prove to be an irresistible attraction to our agent. We take this thought experiment literally and add a TV to the maze along with an action to change the channel. In Figure 4.13, we show how

adding the noisy-TV affects the performance of IDF and RF. As expected the presence of the TV drastically slows down learning, but we note that if you run the experiment for long enough the agents do sometimes converge to getting the extrinsic reward consistently. We have shown empirically that stochasticity can be a problem, and so it is important for future work to address this issue in an efficient manner.

Another limitation of of our proposed as well as prior approaches is that they are too inefficient to be scalable to real robotics setups. In the next chapter, we propose a formulation for exploration inspired by the work in active learning literature which is able to handle both stochastic environments and is efficient enough to scale to real world robots.

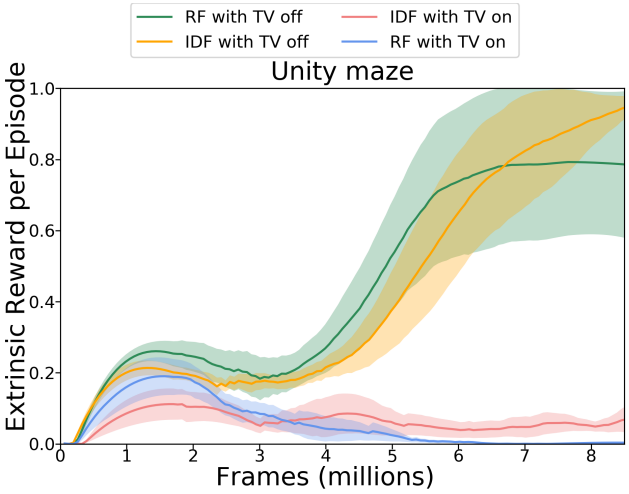


Figure 4.13: We add a noisy TV to the unity environment in Section 4.4.3. We compare IDF and RF with and without the TV.

Chapter 5

Self-Supervised Exploration via Disagreement

Generating intrinsic rewards, e.g. curiosity, requires building some form of a predictive model of the world. However, there is a key challenge in learning predictive models beyond noise-free simulated environments: how should the stochastic nature of agent-environment interaction be handled? Stochasticity could be caused by several sources: (1) noisy environment observations (e.g, TV playing noise), (2) noise in the execution of agent's action (e.g., slipping) (3) stochasticity as an output of the agent's action (e.g., agent flipping coin). One straightforward solution to learn a predictive forward model that is itself stochastic! Despite several methods to build stochastic models in low-dimensional state space [36, 103], scaling it to high dimensional inputs (e.g., images) still remains challenging. An alternative is to build deterministic models but encode the input in a feature space that is invariant to stochasticity. We discussed building such models in inverse model feature space in Section 4.3.1 which can handle stochastic observations but fail when the agent itself is the source of noise (e.g. TV with remote in Section 4.6).

Beyond handling stochasticity, a bigger issue in the current intrinsic reward formulations is that of sample efficiency. The agent performs an action and then computes the reward based on its own prediction and environment behavior. For instance, in curiosity (see Chapter 4), the policy is rewarded if the prediction model and the observed environment disagree. From an exploration viewpoint, this seems like a good formulation, i.e, rewarding actions whose effects are poorly modeled. But this reward is a function of environment dynamics with respect to the performed action. Since the environment dynamics is unknown, it is treated as black-box and the policy's gradients have to be estimated using high-variance estimators like REINFORCE [259] which are extremely sample-inefficient in practice.

We address both the challenges by proposing an alternative formulation for exploration taking inspiration from active learning. The goal of active learning is to selectively pick

This chapter is based on the paper published previously at ICML 2019 [180].

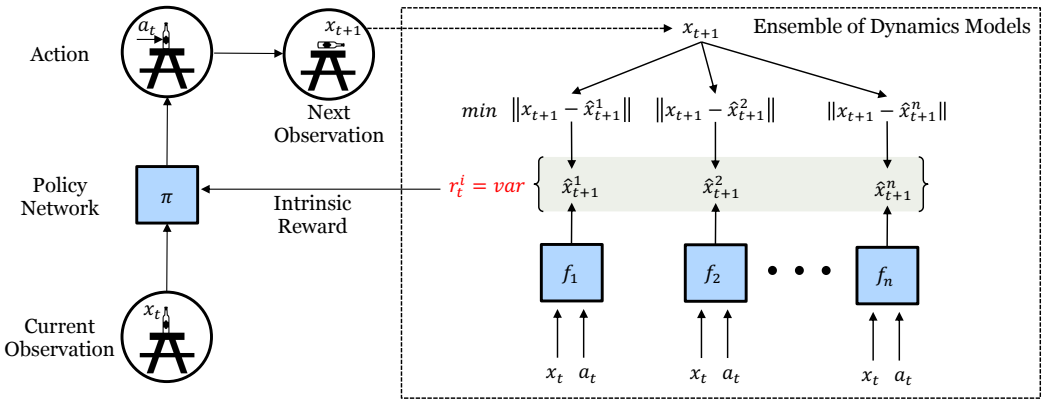


Figure 5.1: Self-Supervised Exploration via Disagreement. At time step t, the agent in the state x_t interacts with the environment by taking action a_t sampled from the current policy π and ends up in the state x_{t+1}. The ensemble of forward models $\{f_1, f_2, ..., f_n\}$ takes this current state x_t and the executed action a_t as input to predict the next state estimates $\{\hat{x}^1_{t+1}, \hat{x}^2_{t+2}, ..., \hat{x}^n_{t+1}\}$. The variance over the ensemble of network output is used as intrinsic reward r^i_t to train the policy π. In practice, we encode the state x into an embedding space $\phi(x)$ for all the prediction purposes.

samples to label such that the classifier is maximally improved. However, unlike current intrinsic motivation formulations where an agent is rewarded by comparing the prediction to the ground-truth, the importance of a sample is not computed by looking at the ground-truth label but rather by looking at the state of the classifier itself. For instance, a popular approach is to label the most uncertain samples by looking at the confidence of the classifier. However, since most of the high-capacity deep neural networks tend to overfit, confidence is not a good measure of uncertainty. Hence, taking an analogy from the Query-by-Committee algorithm [223], we propose a simple disagreement-based approach: we train an ensemble of forward dynamics models and incentivize the agent to explore the action space where there is maximum disagreement or variance among the predictions of models of this ensemble. Taking actions to maximize the model-disagreement allows the agent to explore in a completely self-supervised manner without relying on any external rewards. We show that this approach does not get stuck in stochastic-dynamics scenarios because all the models in the ensemble converge to mean, eventually reducing the variance of the ensemble.

Furthermore, we show that our new objective is a differentiable function allowing us to perform policy optimization via direct likelihood maximization – much like supervised learning instead of reinforcement learning. This leads to a sample efficient exploration policy allowing us to deploy it in a real robotic object manipulation setup with 7-DOF Sawyer arm. We demonstrate the efficacy of our approach on a variety of standard environments including stochastic Atari games [147], MNIST, Mujoco, Unity [114] and a real robot. Project videos and code are at https://pathak22.github.io/exploration-by-disagreement/.

5.1 Exploration via Disagreement

Consider an agent interacting with the environment $\mathcal{E}$. At time t, it receives the observation x_t and then takes an action predicted by its policy, i.e., $a_t \sim \pi(x_t; \theta_P)$. Upon executing the action, it receives, in return, the next observation x_{t+1} which is 'generated' by the environment. Our goal is to build an agent that chooses its action in order to maximally explore the state space of the environment in an efficient manner. There are two main components to our agent: an intrinsic forward prediction model that captures the agent's current knowledge of the states explored so far, and policy to output actions. As our agent explores the environment, we learn the agent's forward prediction model to predict the consequences of its own actions. The prediction uncertainty of this model is used to incentivize the policy to visit states with maximum uncertainty.

Both *measuring* and *maximizing* model uncertainty are challenging to execute with high dimensional raw sensory input (e.g. images). More importantly, the agent should learn to deal with 'stochasticity' in its interaction with the environment caused by either noisy actuation of the agent's motors, or the observations could be inherently stochastic. A deterministic prediction model will always end up with a non-zero prediction error allowing the agent to get stuck in the local minima of exploration.

Similar behavior would occur if the task at hand is too difficult to learn. Consider a robotic arm manipulating a keybunch. Predicting the change in pose and position of each key in the keybunch is extremely difficult. Although the behavior is not inherently stochastic, our agent could easily get stuck in playing with the same keybunch and not try other actions or even other objects. Existing formulations of curiosity reward or novelty-seeking count-based methods would also suffer in such scenarios. Learning probabilistic predictive models to measure uncertainty [103], or measuring learnability by capturing the change in prediction error [174, 215] have been proposed as solutions, but have been demonstrated in low-dimensional state space inputs and are difficult to scale to high dimensional image inputs.

5.1.1 Disagreement as Intrinsic Reward

Instead of learning a single dynamics model, we propose an alternate exploration formulation based on ensemble of models as inspired by the classical active learning literature [223], see Figure 5.1. The goal of active learning is to find the optimal training examples to label such that the accuracy is maximized at minimum labeling cost. While active learning minimizes optimal cost with an analytic policy, the goal of an exploration-driven agent is to learn a policy that allows it to best navigate the environment space. Although the two might look different at the surface, we argue that active learning objectives could inspire powerful intrinsic reward formulations. In this chapter, we leverage the idea of model-variance maximization to propose exploration formulation. Leveraging model variance to investigate a system is also a well-studied mechanism in optimal experimental design literature [24] in statistics.

As our agent interacts with the environment, it collects trajectory of the form $\{x_t, a_t, x_{t+1}\}$. After each rollout, the collected transitions are used to train an ensemble of forward prediction

models $\{f_{\theta_1}, f_{\theta_2} \ldots, f_{\theta_k}\}$ of the environment. Each of the model is trained to map a given tuple of current observation x_t and the action a_t to the resulting state x_{t+1}. These models are trained using straightforward maximum likelihood estimation that minimizes the prediction error, i.e, $\|f(x_t, a_t; \theta) - x_{t+1}\|_2$. To maintain the diversity across the individual models, we initialize each model's parameters differently and train each of them on a subset of data randomly sampled with replacement (bootstrap).

Each model in our ensemble is trained to predict the ground truth next state. Hence, the parts of the state space which have been well explored by the agent will have gathered enough data to train all models, resulting in an agreement between the models. Since the models are learned (and not tabular), this property should generalize to unseen but similar parts of the state-space. However, the areas which are novel and unexplored would still have high prediction error for all models as none of them are yet trained on such examples, resulting in disagreement on the next state prediction. Therefore, we use this disagreement as an intrinsic reward to guide the policy. Concretely, the intrinsic reward r_t^i is defined as the *variance* across the output of different models in the ensemble:

$$r_t^i \triangleq \mathbb{E}_\theta \Big[\| f(x_t, a_t; \theta) - \mathbb{E}_\theta[f(x_t, a_t; \theta)] \|_2^2 \Big] \tag{5.1}$$

Note that the expression on the right does not depend on the next state x_{t+1} — a property which will exploit in Section 5.1.3 to propose efficient policy optimization.

Given the agent's rollout sequence and the intrinsic reward r_t^i at each timestep t, the policy is trained to maximize the sum of expected reward, i.e., $\max_{\theta_P} \mathbb{E}_{\pi(x_t;\theta_P)} \big[\sum_t \gamma^t r_t^i \big]$ discounted by a factor γ. Note that the agent is self-supervised and does not need any extrinsic reward to explore. The agent policy and the forward model ensemble are jointly trained in an online manner on the data collected by the agent during exploration. This objective can be maximized by any policy optimization technique, e.g., we use proximal policy optimization (PPO) [218] unless specified otherwise.

5.1.2 Exploration in Stochastic Environments

Consider a scenario where the next state x_{t+1} is stochastic with respect to the current state x_t and action a_t. The source of stochasticity could be noisy actuation, difficulty or inherent randomness. Given enough samples, a dynamic prediction model should learn to predict the mean of the stochastic samples. Hence, the variance of the outputs in ensemble will drop preventing the agent from getting stuck in stochastic local-minima of exploration. Note this is unlike prediction error based objectives we discussed in previous chapter which will settle down to a mean value after large enough samples. Since, the mean is different from the individual ground-truth stochastic states, the prediction error remains high making the agent forever curious about the stochastic behavior. We empirically verify this intuition by comparing prediction-error to disagreement across several environments in Section 5.3.2.

5.1.3 Differentiable Exploration for Policy Optimization

One commonality between different exploration methods [17, 103, 179], is that the prediction model is usually learned in a supervised manner and the agent's policy is trained using reinforcement learning either in on-policy or off-policy manner. Despite several formulations over the years, the policy optimization procedure to maximize these intrinsic rewards has more or less remained the same – i.e. – treating the intrinsic reward as a "black-box" even though it is generated by the agent itself.

Let's consider an example to understand the reason behind the status quo. Consider a robotic-arm agent trying to push multiple objects kept on the table in front of it by looking at the image from an overhead camera. Suppose the arm pushes an object such that it collides with another one on the table. The resulting image observation will be the outcome of complex real-world interaction, the actual dynamics of which is not known to the agent. Note that this resulting image observation is a function of the agent's action (i.e., push in this case). Most commonly, the intrinsic reward $r^i(x_t, a_t, x_{t+1})$ is function of the next state (which is a function of the agent's action), e.g., information gain [103], prediction error (Chapter 4) etc. This dependency on the unknown environment dynamics absolves the policy optimization of analytical reward gradients with respect to the action. Hence, the standard way is to optimize the policy to maximize the sequence of intrinsic rewards using reinforcement learning, and not make any use of the structure present in the design of r^i_t.

We formulate our proposed intrinsic reward as a differentiable function so as to perform policy optimization using likelihood maximization – much like supervised learning instead of reinforcement. If possible, this would allow the agent to make use of the structure in r^i_t explicitly, i.e., the intrinsic reward from the model could very efficiently inform the agent to change its action space in the direction where forward prediction loss is high, instead of providing a *scalar* feedback as in case of reinforcement learning. Explicit reward (cost) functions are one of the key reasons for success stories in optimal-control based robotics [48, 73], but they don't scale to high-dimensional state space such as images and rely on having access to a good model of the environment.

We first discuss the one step case and then provide the general setup. Note that our intrinsic reward formulation, shown in Equation (5.1), does not depend on the environment interaction at all, i.e., no dependency on x_{t+1}. It is purely a mental simulation of the ensemble of models based on the current state and the agent's prediction action. Hence, instead of maximizing the intrinsic reward in expectation via PPO (RL), we can optimize for policy parameters θ_P using direct gradients by treating r^i_t as a differentiable loss function. The objective for a one-step reward horizon is:

$$\min_{\theta_1,\ldots,\theta_k} \; (1/k) \sum_{i=1}^{k} \| f_{\theta_i}(x_t, \pi(x_t; \theta_P)) - x_{t+1} \|_2 \tag{5.2}$$

$$\max_{\theta_P} \; (1/k) \sum_{i=1}^{k} \Big[\| f_{\theta_i}(x_t, \pi(x_t; \theta_P)) - (1/k) \sum_{j=1}^{k} f_{\theta_j}(x_t, \pi(x_t; \theta_P)) \|_2^2 \Big]$$

This is optimized in an alternating fashion where the forward predictor is optimized keeping the policy parameters frozen and vice-versa. Note that both policy and forward models are trained via maximum likelihood in a supervised manner, and hence, efficient in practice.

Generalization to multi-step reward horizon To optimize policy for maximizing a discounted sum of sequence of future intrinsic rewards r_t^i in a differentiable manner, the forward model would have to make predictions spanning over multiple time-steps. The policy objective in Equation (5.2) can be generalized to the multi-step horizon setup by recursively applying the forward predictor, i.e., $\max_{\theta_P} \sum_t r_t^i(\hat{x}_t, a_t)$ where $\hat{x}_t = f(\hat{x}_{t-1}, a_{t-1}; \theta)$, $a_t = \pi(x_t; \theta_P)$, $\hat{x}_0 = x_0$, and $r_t^i(.)$ is defined in Equation (5.1). Alternatively, one could use LSTM to make forward model itself multi-step. However, training a long term multi-step prediction model is challenging and an active area of research. In this thesis, we show differentiable exploration results for short horizon only and leave multi-step scenarios for future work.

5.2 Implementation Details and Baselines

Learning forward predictions in the feature space In Chapter 4, we showed that learning forward-dynamics predictor f_θ in a feature space leads to better generalization in contrast to raw pixel-space predictions. Our formulation is trivially extensible to any representation space ϕ because all the operations can be performed with $\phi(x_t)$ instead of x_t. Hence, in all of our experiments, we train our forward prediction models in feature space. In particular, we use random feature space in all video games and navigation, classification features in MNIST and ImageNet-pretrained ResNet-18 features in real world robot experiments. We use 5 models in the ensemble.

Back-propagation through forward model To directly optimize the policy with respect to the loss function of the forward predictor, as discussed in Section 5.1.3, we need to backpropagate all the way through action sampling process from the policy. In case of continuous action space, one could achieve this via making policy deterministic, i.e. $a_t = \pi_{\theta_P}$ with epsilon-greedy sampling [141]. For discrete action space, we found that straight-through estimator [21] works well in practice.

Baseline Comparisons 'Disagreement' refers to our exploration formulation optimized using PPO [218] as discussed in Section 5.1.1, unless mentioned otherwise. 'Disagreement [Differentiable]' refers to the direct policy optimization for our formulation as described in Section 5.1.3. 'Pathak et.al. [ICML 2017]' refers to the curiosity-driven exploration formulation based on the prediction error of the learned forward dynamics model in inverse model action space, discussed in Chapter 4. 'Burda et.al. [ICLR 2019]' refers to the random feature-based prediction-error [30]; also discussed in Chapter 4. 'Pred-Error Variance' is an

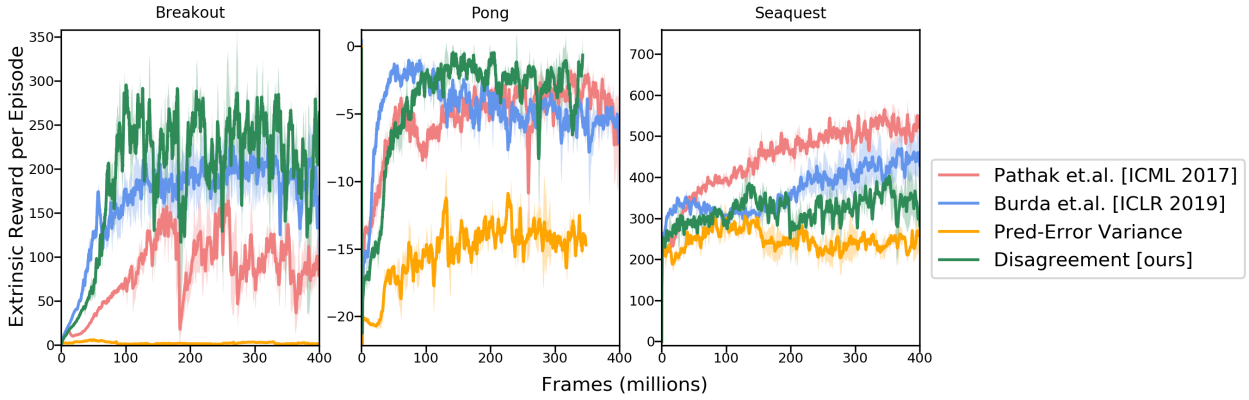


Figure 5.2: Sanity Check in Non-Stochastic Environments. We compare different intrinsic reward formulations across near-deterministic, non-stochastic standard benchmark of the Atari games. Our disagreement-based approach compares favorably to state-of-the-art approaches without losing accuracy in non-stochastic scenarios.

alternative ablation where we train the agent to maximize the variance of the prediction error as opposed to the variance of model output itself. Finally, we also compare our performance to Bayesian Neural Networks for measuring variance. In particular, we compared to Dropout NN [72] represented as 'Bayesian Disagreement'.

5.3 Experiments

We evaluate our approach on several environments including Atari games, 3D navigation in Unity, MNIST, object manipulation in Mujoco and real world robotic manipulation task using Sawyer arm. Our experiments comprise of three parts: a) verifying the performance on standard non-stochastic environments; b) comparison on environments with stochasticity in either transition dynamics or observation space; and c) validating the efficiency of differentiable policy optimization facilitated by our objective.

5.3.1 Sanity Check in Non-Stochastic Environments

We first verify whether our disagreement formulation is able to maintain the performance on the standard environment as compared to state of the art exploration techniques. Although the primary advantage of our approach is in handling stochasticity and improving efficiency via differentiable policy optimization, it should not come at the cost of performance in nearly-deterministic scenarios. We run this sanity check on standard Atari benchmark suite, as shown in Figure 5.2. These games are not completely deterministic and have some randomness as to where the agent is spawned upon game resets [155]. The agent is trained with only an intrinsic reward, without any external reward from the game environment. The external

reward is only used as a proxy to evaluate the quality of exploration and not shown to the agent.

We train our ensemble of models for computing disagreement in the embedding space of a random network as discussed in Section 5.2. The performance is compared to curiosity formulation with inverse dynamics and random features (discussed in Chapter 4), Bayesian network based uncertainty and variance of prediction error. As seen in the results, our method is as good as or slightly better than state-of-the-art exploration methods in most of the scenarios. Overall, these experiments suggest that our exploration formulation which is only driven by disagreement between models output compares favorably to state of the art methods. Note that the variance of prediction error performs significantly worse. This is so because the low variance in prediction error of different models doesn't necessarily mean they will agree on the next state prediction. Hence, 'Pred-Error Variance' may sometimes incorrectly stop exploring even if output prediction across models is drastically different.

5.3.2 Exploration in Stochastic Environments

A) Noisy MNIST. We first build a toy task on MNIST to intuitively demonstrate the contrast between disagreement-based intrinsic reward and prediction error-based reward (cf. Chapter 4) in stochastic setups. This is a one-step environment where the agent starts by randomly observing an MNIST image from either class 0 or class 1. The dynamics of the environment are defined as follows: 1) images with label 0 always transition to another image from class 0. 2) Images with label 1 transition to a randomly chosen image from class label 2 to 9. This ensures that a transition from images with label 0 has low stochasticity (i.e., transition to the same label). On the other hand, transitions from images with label 1 have high stochasticity. The ideal intrinsic reward function should give similar incentive (reward) to both the scenarios after the agent has observed a significant number of transitions.

Figure 5.3 shows the performance of these methods on the test set of MNIST as a function of the number of states visited by the agent. Even at convergence, the prediction error based model assigns more reward to the observations with higher stochasticity, i.e., images with label 1. This behavior is detrimental since the transition from states of images with label 1 cannot ever be perfectly modeled and hence the agent will get stuck forever. In contrast, our ensemble-based disagreement method converges to almost zero intrinsic reward in both the scenarios after the agent has seen enough samples, as desired.

B) 3D Navigation in Unity. The goal in this setup is to train the agent to reach a target location in the maze. The agent receives a sparse reward of +1 on reaching the goal. For all the methods, we train the policy of the agent to maximize the summation of intrinsic and sparse extrinsic reward. This particular environment is a replica of VizDoom-MyWayHome environment in unity ML-agent and was proposed in Burda *et al*. [30]. Interestingly, this environment has 2 variants, one of which has a TV on the wall. The agent can change the channel of the TV but the content is stochastic (random images appear after pressing button). The agent can start randomly anywhere in the maze in each episode, but the

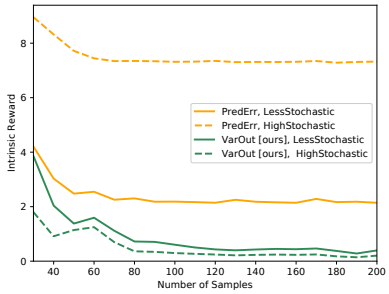


Figure 5.3: Performance of disagreement across ensemble vs prediction error based reward function on Noisy MNIST environment. This environment has 2 sets of state with different level of stochasticity associated with them. The disagreement-based intrinsic reward converges to the ideal case of assigning the same reward value for both states. However, the prediction-error based reward function assigns a high reward to states with high stochasticity.

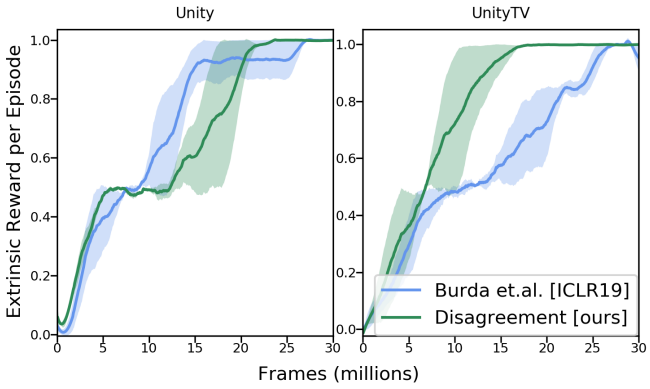


Figure 5.4: 3D Navigation in Unity. Comparison of prediction-error based curiosity reward with our proposed disagreement-based exploration on 3D navigation task in Unity with and without the presence of TV+remote. While both the approaches perform similar in normal case (left), disagreement-based approach performs better in the presence of stochasticity (right).

goal location is fixed. We compare our proposed method with state-of-the-art prediction error-based exploration [30]. The results are shown in Figure 5.4. Our approach performs similar to the baseline in the non-TV setup and outperforms the baseline in the presence of the TV. This result demonstrates that an ensemble-based disagreement could be a viable alternative in realistic stochastic setups.

C) Atari with Sticky Actions. As discussed in Section 5.3.1, the usual Atari setup is nearly deterministic. Therefore, a recent study [147] proposed to introduce stochasticity in Atari games by making actions 'sticky', i.e., at each step, either the agent's intended action is executed or the previously executed action is repeated with equal probability. As shown in Figure 5.5, our disagreement-based exploration approach outperforms previous state-of-the-art approaches. In Pong, our approach starts slightly slower than Burda et.al. [30], but eventually achieves a higher score. Further note that the Bayesian network-based disagreement does not perform as well as ensemble-based disagreement. This suggests that perhaps dropout [72] isn't able to capture good uncertainty estimate in practice. These experiments along with the navigation experiment, demonstrate the potential of ensembles in the face of stochasticity.

5.3.3 Differentiable Exploration in Structured Envs

We now evaluate the differentiable exploration objective proposed in Section 5.1.3. As discussed earlier, the policy is optimized via direct analytic gradients from the exploration

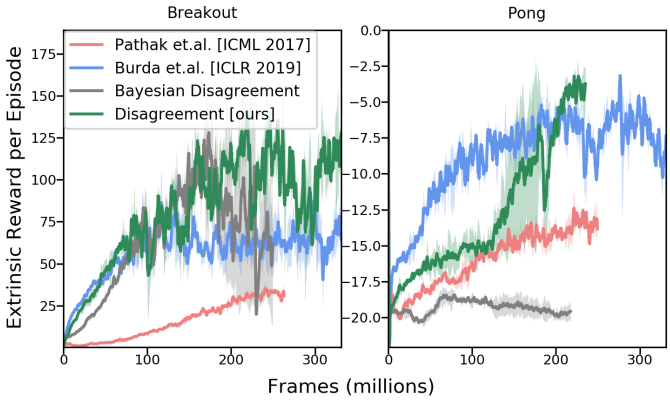


Figure 5.5: Stochastic Atari Games: Comparison of different exploration techniques in the the Atari ('sticky') environment. The disagreement-based exploration is robust across both the scenarios.

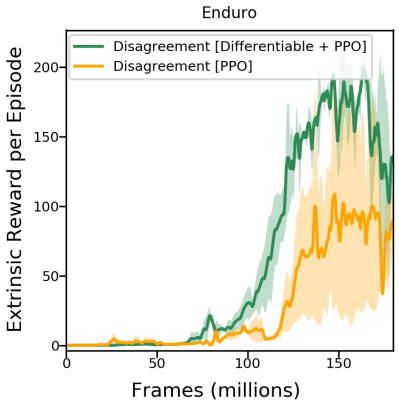


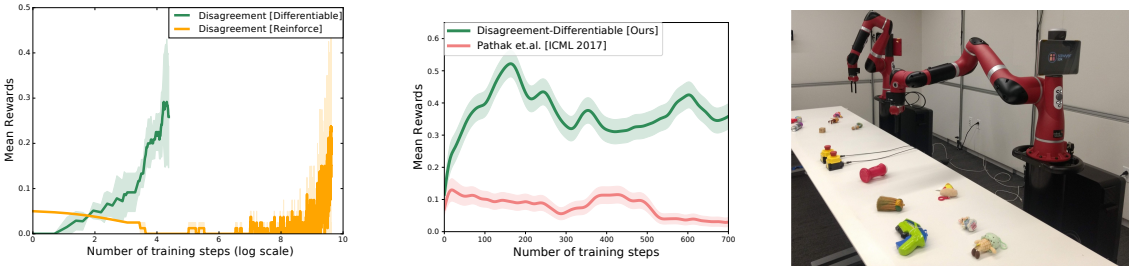
Figure 5.6: Performance comparison of disagreement-based exploration with or without the differentiable policy optimization in Enduro Atari Game. Differentiability helps the agent learn faster.

module. Therefore, the horizon of exploration depends directly on the horizon of the module. Since training long-horizon models from high dimensional inputs (images) is still an unsolved problem, we evaluate our proposed formulation on relatively short horizon scenarios. However, to compensate for the length of the horizon, we test on large action space setups for real-world robot manipulation task.

A) Enduro Video Game. In this game, the goal of the agent is to steer the car on racing track to avoid enemies. The agent is trained to explore via purely intrinsic rewards, and the extrinsic reward is only used for evaluation. In order to steer the car, the agent doesn't need to model long-range dependencies. Hence, in this environment, we combine our differentiable policy optimization with reinforcement learning (PPO) to maximize our disagreement based intrinsic reward. The RL captures discounted long term dependency while our differentiable formulation should efficiently take care of short-horizon dependencies. We compare this formulation to purely PPO based optimization of our intrinsic reward. As shown in Figure 5.6, our differentiable exploration expedites the learning of the agent suggesting the efficacy of direct gradient optimization. We now evaluate the performance of only differentiable exploration (without reinforcement) in short-horizon and large-structured action space setups.

B) Object Manipulation by Exploration.

We consider the task of object manipulation in complex scenarios. Our setup consists of a 7-DOF robotic arm that could be tasked to interact with the objects kept on the table in front of it. The objects are kept randomly in the workspace of the robot on the table.



(a) Mujoco (b) Real Robot (c) Real Robot Setup

Figure 5.7: Measuring object interaction rate with respect to the number of samples in (a) Mujoco, and (b) real-world robot. Note that the Mujoco plot is in log-scale. We measure the exploration quality by evaluating the object interaction frequency of the agent. In both the environments, our differentiable policy optimization explores more efficiently. (c) A snapshot of the real-robotic setup.

Robot's action space is end-effector position control: a) location (x, y) of point on the surface of table, b) angle of approach θ, and c) gripper status, a binary value indicating whether to grasp (open the gripper fingers) or push (keep fingers close). All of our experiments use raw visual RGBD images as input and predict actions as output. Note that, to accurately grasp or push objects, the agent needs to figure out an accurate combination of location, orientation and gripper status.

The action space is discretized into 224×224 locations, 16 orientations for grasping (fingers close) and 16 orientations for pushing leading to final dimension of $224 \times 224 \times 32$. The policy takes as input a 224×224 RGBD image and produces push and grasp action probabilities for each pixel. Following [275], instead of adding the 16 rotations in the output, we pass 16 equally spaced rotated images to the network and then sample actions based on the output of all the inputs. This exploits the convolutional structure of the network. The task has a short horizon but very large state and action spaces. We make no assumption about either the environment or the training signal. Our robotic agents explore the work-space purely out of their own intrinsic reward in a pursuit to develop useful skills. We have instantiated this setup in a Mujoco simulation as well as in the real world robotics scenarios.

B1) Object Manipulation in MuJoCo. We first carry out a study in simulation to compare the performance of differentiable variant of our disagreement objective against the reinforcement learning based optimization. We used MuJoCo to simulate the robot performing grasping and pushing on tabletop environment as described above.

To evaluate the quality of exploration, we measure the frequency at which our agent interacts (i.e., touches) with the object. This measure is just used to evaluate the exploration quantitatively and is not used as a training signal. It represents how quickly our agent's policy learns to explore an interesting part of space. Figures 5.7a shows the performance when the environment consists of just a single object which makes it really difficult to touch the object randomly. Our approach is able to exploit the structure in the loss, resulting in

order of magnitude faster learning than REINFORCE.

B2) Real-World Robotic Manipulation. We now deploy our sample-efficient exploration formulation on real-world robotics setup. The real-world poses additional challenges, unlike simulated environments in terms of behavior and the dynamics of varied object types. Our robotic setup consisted of a Sawyer-arm with a table placed in front of it. We mounted KinectV2 at a fixed location from the robot to receive RGBD observations of the environment.

In every run, the robot starts with 3 objects placed in front of it. Unlike other self-supervised robot learning setups, we keep fewer objects to make exploration problem harder so that it is not trivial to interact with the objects by acting randomly. If either the robot completes 100 interactions or there are no objects in front of it, objects are replaced manually. Out of a total of 30 objects, we created a set of 20 objects for training and 10 objects for testing. We use the same metric as used in the simulation above (i.e., number of object interactions) to measure the effectiveness of our exploration policy during training. We monitor the change in the RGBD image to see if the robot has interacted with objects. Figure 5.7b shows the effectiveness of differentiable policy optimization for disagreement over prediction-error based curiosity objective. Differentiable-disagreement allows the robotic agent to learn to interact with objects in less than 1000 examples.

We further test the skills learned by our robot during its exploration by measuring object-interaction frequency on a set of 10 held-out test objects. For both the methods, we use the checkpoint saved after 700 robot interaction with the environment. For each model, we evaluate a total of 80 robot interaction steps with three test objects kept in front. The environment is reset after every 10 robot steps during evaluation. Our final disagreement exploration policy interacts approximately **67%** of times with unseen objects, whereas a random policy performs at **17%**. On the other hand, it seems that REINFORCE-based curiosity policy just collapses and only **1%** of actions involve interaction with objects. Videos are available at https://pathak22.github.io/exploration-by-disagreement/.

5.4 Discussion

We discussed large body of literature in Chapter 4, Section 4.5 on exploration and curioity. However, most of those approaches are considered in the context of external rewards and are not efficient enough to be scalable to real robotics setup.

This chapter is inspired by large-body of work in active learning (AL). In the AL setting, given a collection of unlabeled examples, a learner selects which samples will be labeled by an oracle [222]. Common selection criteria include entropy [40], uncertainty sampling [139] and expected informativeness [102]. Our work is inspired by [223], and we apply the disagreement idea in a completely different setting of exploration and show its applicability to environments with stochastic dynamics and improving sample-efficiency. Concurrent to this work, [225] also show the effectiveness of model-based exploration in estimating novelty, and [97] use variance regularization for policy learning via imitation.

Part III

From Skills to Goal-Directed Expertise

Chapter 6

Zero-Shot Visual Imitation

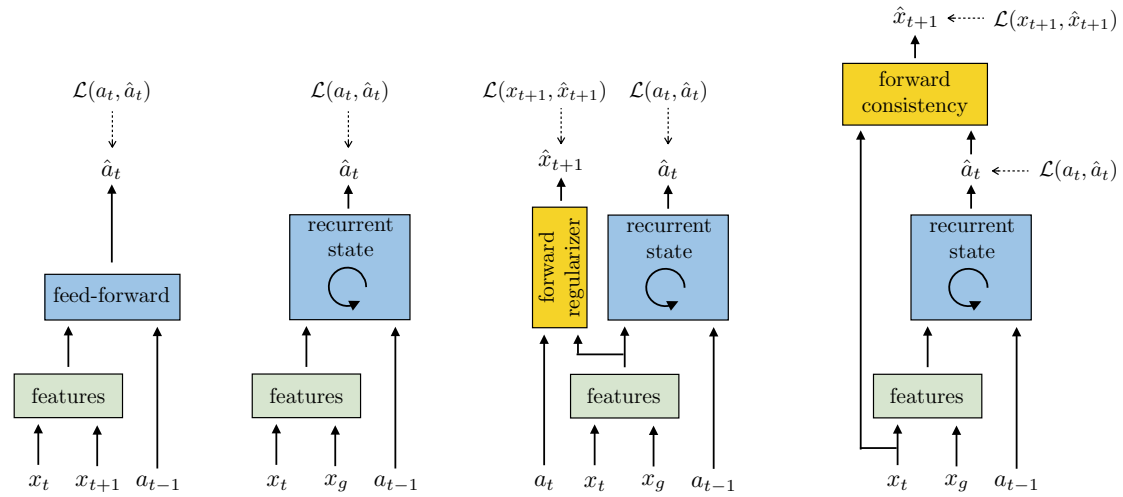
So far, we discussed how an agent could acquire sensorimotor skills of increasing complexity with no knowledge of labels or tasks. However, in order to be useful for real-world tasks, the agent will eventually have to develop the ability to perform a specific task given to it. One way to develop such expertise is to fine-tune our curious agents using rewards that are tuned for an external goal. We discuss results using this reward-based finetuning in Chapter 4. However, this reward-based finetuning relies on reinforcement learning and is therefore too sample inefficient to be scalable to complex real world robotic tasks. Furthermore, designing such reward function for real-world tasks is a complex engineering problem.

Therefore, learning from demonstration (LfD) [10, 163, 197, 211] has emerged as a powerful mechanism for learning to perform tasks from raw sensory observations with real robots. The current dominant paradigm in LfD requires the expert to either manually move the robot joints (i.e., kinesthetic teaching) or teleoperate the robot to execute the desired task. The expert typically provides multiple demonstrations of a task at training time, and this generates data in the form of observation-action pairs from the agent's point of view. The agent then distills this data into a policy for performing the task of interest. Such a heavily supervised approach, where it is necessary to provide demonstrations by controlling the robot, is incredibly tedious for the human expert. Moreover, for every new task that the robot needs to execute, the expert is required to provide a new set of demonstrations.

Instead of communicating *how* to perform a task via observation-action pairs, a more general formulation allows the expert to communicate only *what* needs to be done by providing the observations of the desired world states via a video or a sparse sequence of images. This way, the agent is required to infer how to perform the task (i.e., actions) by itself. In psychology, this is known as *observational learning* [14]. While this is a harder learning problem, it is a more interesting setting, because the expert can demonstrate multiple tasks quickly and easily.

An agent without any prior knowledge will find it extremely hard to imitate a task by

This chapter is based on the paper published previously at ICLR 2018 [186].



(a) Inverse Model (b) Multi-step GSP (c) Forward-regularized GSP (d) Forward-consistent GSP (ours)

Figure 6.1: The goal-conditioned skill policy (GSP) takes as input the current and goal observations and outputs an action sequence that would lead to that goal. We compare the performance of the following GSP models: (a) Simple inverse model; (b) Mutli-step GSP with previous action history; (c) Mutli-step GSP with previous action history and a forward model as regularizer, but no forward consistency; (d) Mutli-step GSP with forward consistency loss proposed in this chapter.

simply watching a visual demonstration in all but the simplest of cases. Thus, the natural question is: in order to imitate, what form of prior knowledge must the agent possess? A large body of work [26, 49, 107, 129, 130, 266] has sought to capture prior knowledge by manually pre-defining the state that must be inferred from the observations. The agent then infers how to perform the task (i.e., plan for imitation) using this state. Unfortunately, computer vision systems are often unable to estimate the state variables accurately and it has proven non-trivial for downstream planning systems to be robust to such errors.

In this chapter, we follow [5, 138, 194] in pursuing an alternative paradigm, where an agent explores the environment without any expert supervision and distills this exploration data into goal-directed skills. These skills can then be used to imitate the visual demonstration provided by the expert [161]. Here, by skill we mean a function that predicts the sequence of actions to take the agent from the current observation to the goal. We call this function a *goal-conditioned skill policy (GSP)*. The GSP is learned in a self-supervised way by re-labeling the states visited during the agent's exploration of the environment as goals and the actions executed by the agent as the prediction targets, similar to [5, 9]. During inference, given goal observations from a demonstration, the GSP can infer how to reach these goals in turn from the current observation, and thereby imitate the task step-by-step.

One critical challenge in learning the GSP is that, in general, there are multiple possible ways of going from one state to another: that is, the distribution of trajectories between states is multi-modal. We address this issue with our novel *forward consistency loss* based on

the intuition that, for most tasks, reaching the goal is more important than how it is reached. To operationalize this, we first learn a forward model that predicts the next observation given an action and a current observation. We use the difference in the output of the forward model for the GSP-selected action and the ground truth next state to train the GSP. This loss has the effect of making the GSP-predicted action *consistent* with the ground-truth action instead of exactly matching the actions themselves, thus ensuring that actions that are different from the ground-truth—but lead to the same next state—are not inadvertently penalized. To account for varying number of steps required to reach different goals, we propose to jointly optimize the GSP with a goal recognizer that determines if the current goal has been satisfied. See Figure 6.1 for a schematic illustration of the GSP architecture.

We call our method *zero-shot* because the agent never has access to expert actions, neither during training of the GSP nor for task demonstration at inference. In contrast, recent works on one-shot imitation learning requires full knowledge of actions and a wealth of expert demonstrations during training [57, 67]. In summary, we propose a method that (1) does not require any extrinsic reward or expert supervision during learning, (2) only needs demonstrations during inference, and (3) restricts demonstrations to visual observations alone rather than full state-actions. Instead of learning by imitation, our agent learns to imitate. Videos and code at https://pathak22.github.io/zeroshot-imitation/.

6.1 Learning to Imitate without Expert Supervision

Let $\mathcal{S} : \{x_1, a_1, x_2, a_2, ..., x_T\}$ be the sequence of observations and actions generated by the agent as it explores its environment using the policy $a = \pi_E(s)$. This exploration data is used to learn the goal-conditioned skill policy (GSP) π takes as input a pair of observations (x_i, x_g) and outputs sequence of actions $(\vec{a}_\tau : a_1, a_2...a_K)$ required to reach the goal observation (x_g) from the current observation (x_i).

$$\vec{a}_\tau = \pi(x_i, x_g; \theta_\pi) \tag{6.1}$$

where states x_i, x_g are sampled from the $\mathcal{S}$. The number of actions, K, is also inferred by the model. We represent π by a deep network with parameters θ_π in order to capture complex mappings from visual observations (x) to actions. π can be thought of as a variable-step generalization of the inverse dynamics model [113], or as the policy corresponding to a universal value function [70, 213], with the difference that x_g need not be the end goal of a task but can also be an intermediate sub-goal.

Let the task to be imitated be provided as a sequence of images $\mathcal{D} : \{x_1^d, x_2^d, ..., x_N^d\}$ captured when the expert demonstrates the task. This sequence of images $\mathcal{D}$ could either be temporally dense or sparse. Our agent uses the learned GSP π to imitate the sequence of visual observations $\mathcal{D}$ starting from its initial state x_0 by following actions predicted by $\pi(x_0, x_1^d; \theta_\pi)$. Let the observation after executing the predicted action be x_0'. Since multiple actions might be required to reach close to x_1^d, the agent queries a separate *goal recognizer* network to ascertain if the current observation is close to the goal or not. If the answer is

negative, the agent executes the action $a = \pi(x'_0, x^d_1; \theta_\pi)$. This process is repeated iteratively until the *goal recognizer* outputs that agent is near the goal, or a maximum number of steps are reached. Let the observation of the agent at this point be $\hat{x}_1$. After reaching close to the first observation (x^d_1) in the demonstration, the agent sets its goal as (x^d_2) and repeats the process. The agent stops when all observations in the demonstrations are processed.

Note that in the method of imitation described above, the expert is never required to convey to the agent what actions it performed. In the following subsections we describe how we learn the GSP, *forward consistency loss*, *goal recognizer* network and various baseline methods.

6.1.1 Learning the Goal-conditioned Skill Policy (GSP)

We first describe the one-step version of GSP and then extend it to variable length multi-step skills. One-step trajectories take the form of (x_t, a_t, x_{t+1}) and GSP, $\hat{a}_t = \pi(x_t, x_{t+1}; \theta_\pi)$, is trained by minimizing the standard cross-entropy loss $\mathcal{L}(a_t, \hat{a}_t)$,

$$\mathcal{L}(a_t, \hat{a}_t) = p(a_t | x_t, x_{t+1}) \log(\hat{a}_t) \tag{6.2}$$

with respect to parameters θ_π, where p and $\hat{a}_t$ are the ground-truth and predicted action distributions. While we do not have access to true p, we empirically approximate it using samples from the distribution, a_t, that are executed by the agent during exploration. For minimizing the cross-entropy loss, it is common to assume p as a delta function at a_t. However, this assumption is notably violated if p is inherently multi-modal and high-dimensional. If we optimize say a deep neural network assuming p to be a delta function, the same inputs will be presented with different targets (due to multi-modality) leading to high-variance in gradients which in turn would make learning challenging.

In our setup, such multi-modality can occur because multiple actions can lead the agent to the same future observation from the initial observation. For instance, in navigation, if the agent is stuck against a corner, turning or moving forward all collapse to the same effect. The issue of multi-modality becomes more critical as the length of trajectories grow, because more and more paths may take the agent from the initial observation to the goal observation given more time. Furthermore, it would require many samples to even obtain a good empirical estimate of a high-dimensional multi-modal action distribution p.

6.1.2 Forward Consistency Loss

One way to account for multi-modality is by employing the likes of variational auto-encoders [120, 201]. However, in many practical situations it is not feasible to obtain ample data for each mode. In this chapter, we propose an alternative based on the insight that in many scenarios, we only care about whether the agent reached the final state or not and the exact trajectory is of lesser interest. Instead of penalizing the actions predicted by the GSP to match the ground truth, we propose to learn the parameters of GSP by

minimizing the distance between observation $\hat{x}_{t+1}$ resulting by executing the predicted action $\hat{a}_t = \pi(x_t, x_{t+1}; \theta_\pi)$ and the observation x_{t+1}, which is the result of executing the ground truth action a_t being used to train the GSP. In this formulation, even if the predicted and ground-truth action are different, the predicted action will not be penalized if it leads to the same next state as the ground-truth action. While this formulation will not explicitly maintain all modes of the action distribution, it will reduce the variance in gradients and thus help learning. We call this penalty the *forward consistency loss*.

Note that it is not immediately obvious as to how to operationalize *forward consistency loss* for two reasons: (a) we need the access to a good *forward dynamics* model that can reliably predict the effect of an action (i.e., the next observation state) given the current observation state, and (b) such a dynamics model should be differentiable in order to train the GSP using the state prediction error. Both of these issues could be resolved if an analytic formulation of forward dynamics is known.

In many scenarios of interest, especially if states are represented as images, an analytic forward model is not available. In this chapter, we learn the forward dynamics f model from the data, and is defined as $\tilde{x}_{t+1} = f(x_t, a_t; \theta_f)$. Let $\hat{x}_{t+1} = f(x_t, \hat{a}_t; \theta_f)$ be the state prediction for the action predicted by π. Because the forward model is not analytic and learned from data, in general, there is no guarantee that $\tilde{x}_{t+1} = \hat{x}_{t+1}$, even though executing these two actions, $a_t, \hat{a}_t$, in the real-world will have the same effect. In order to make the outcome of action predicted by the GSP and the ground-truth action to be *consistent* with each other, we include an additional term, $\|x_{t+1} - \hat{x}_{t+1}\|_2^2$ in our loss function and infer the parameters θ_f by minimizing $\|x_{t+1} - \tilde{x}_{t+1}\|_2^2 \; + \; \lambda \|x_{t+1} - \hat{x}_{t+1}\|_2^2$, where λ is a scalar hyper-parameter. The first term ensures that the learned forward model explains ground truth transitions (x_t, a_t, x_{t+1}) collected by the agent and the second term ensures consistency. The joint objective for training GSP with forward model consistency is:

$$\min_{\theta_\pi, \theta_f} \; \|x_{t+1} - \tilde{x}_{t+1}\|_2^2 + \lambda \|x_{t+1} - \hat{x}_{t+1}\|_2^2 \; + \; \mathcal{L}(a_t, \hat{a}_t) \tag{6.3}$$

$$\begin{aligned} \text{s.t.} \quad & \tilde{x}_{t+1} = f(x_t, a_t; \theta_f) \\ & \hat{x}_{t+1} = f(x_t, \hat{a}_t; \theta_f) \\ & \hat{a}_t = \pi(x_t, x_{t+1}; \theta_\pi) \end{aligned}$$

Note that learning θ_π, θ_f jointly from scratch is precarious, because the forward model f might not be good in the beginning, and hence could make the gradient updates noisier for π. To address this issue, we first pre-train the forward model with only the first term and GSP separately by blocking the gradient flow and then fine-tune jointly.

Generalization to feature space dynamics Past work has shown that learning forward dynamics in the feature space as opposed to raw observation space is more robust and leads to better generalization [5, 179]. Following these works, we extend the GSP to make predictions in feature representation $\phi(x_t), \phi(x_{t+1})$ of the observations x_t, x_{t+1} respectively learned through the self-supervised task of action prediction. The forward consistency loss is then computed by making predictions in this feature space ϕ instead of raw observations.

The optimization objective for feature space generalization with mutli-step objective is shown in Equation (6.4).

Generalization to multi-step GSP We extend our one-step optimization to variable length sequence of actions in a straightforward manner by having a multi-step GSP π_m model with a step-wise forward consistency loss. The GSP π_m maintains an internal recurrent memory of the system and outputs actions conditioned on current observation x_t, starting from x_i to reach goal observation x_T. The forward consistency loss is computed at each time step, and jointly optimized with the action prediction loss over the whole trajectory. The final multi-step objective with feature space dynamics is as follows:

$$\min_{\theta_\pi,\theta_f,\theta_\phi} \sum_{t=i}^{t=T} \Big(\|\phi(x_{t+1}) - \tilde{\phi}(x_{t+1})\|_2^2 \; + \; \lambda \|\phi(x_{t+1}) - \hat{\phi}(x_{t+1})\|_2^2 \; + \; \mathcal{L}(a_t, \hat{a}_t) \Big) \tag{6.4}$$

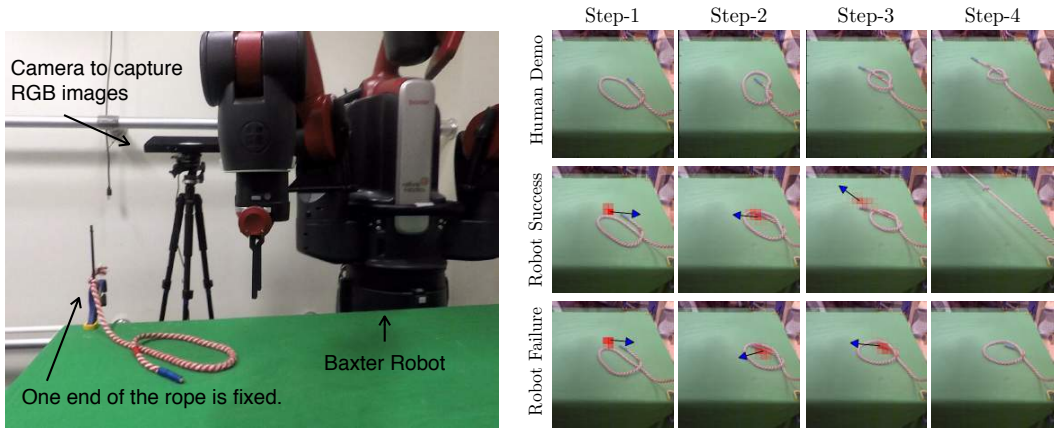
$$\begin{aligned} \text{s.t.} \quad & \tilde{\phi}(x_{t+1}) = f(\phi(x_t), a_t; \theta_f) \\ & \hat{\phi}(x_{t+1}) = f(\phi(x_t), \hat{a}_t; \theta_f) \\ & \hat{a}_t = \pi(\phi(x_t), \phi(x_T); \theta_\pi) \end{aligned}$$

where $\phi(.)$ is represented by a CNN with parameters θ_ϕ. The number of steps taken by the multi-step GSP π_m to reach the goal at inference is variable depending on the decision of goal recognizer; described in next subsection. Note that, in this objective, if ϕ is identity then the dynamics simply reduces to modeling in raw observation space. We analyze feature space prediction in *VizDoom* 3D navigation and stick to observation space in the rope manipulation and the office navigation tasks.

The multi-step *forward-consistent GSP* π_m is implemented using a recurrent network which at every step takes as input the feature representation of the current ($\phi(x_t)$) state, goal ($\phi(x_T)$) states, action at the previous time step (a_{t-1}) and the internal hidden representation h_{t-1} of the recurrent units and predicts $\hat{a}_t$. Note that inputting the previous action to GSP π_m at each time step could be redundant given that hidden representation is already maintaining a history of the trajectory. Nonetheless, it is helpful to explicitly model this history. This formulation amounts to building an auto-regressive model of the joint action that estimates probability $P(a_t|x_1, a_1, ...a_{t-1}, x_t, x_g)$ at every time step. It is possible to further extend our forward-consistent GSP π_m to build multi-step forward model, but we leave that direction of future work.

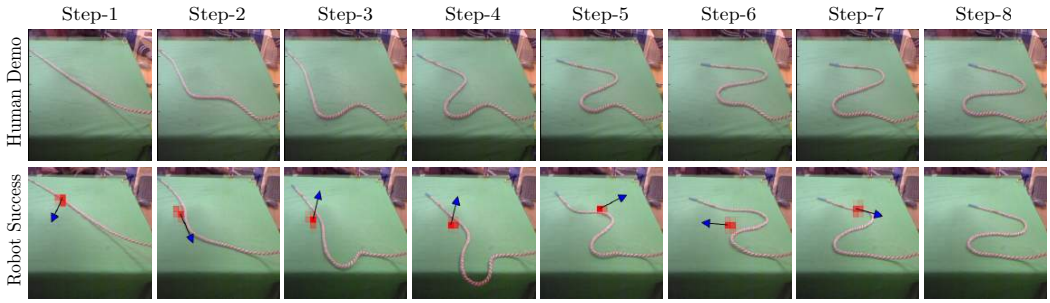
6.1.3 Goal Recognizer

We train a goal recognizer network to figure out if the current goal is reached and therefore allow the agent to take variable numbers of steps between goals. Goal recognition is especially critical when the agent has to transit through a sequence of intermediate goals, as is the case for visual imitation, as otherwise compounding error could quickly lead to divergence from the demonstration. This recognition is simple given knowledge of the true physical state, but difficult when working with visual observations. Aside from the usual challenges



(a) Robotics Setup

(b) Manipulating rope into tying a knot

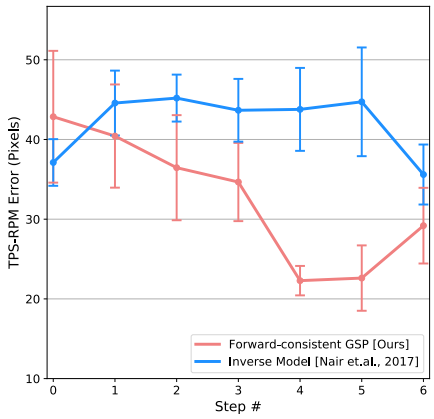


(c) Manipulating rope into 'S' shape

Figure 6.2: Qualitative visualization of results for rope manipulation task using Baxter robot. (a) Our robotics system setup. (b) The sequence of human demonstration images provided by the human during inference for the task of knot-tying (top row), and the sequences of observation states reached by the robot while imitating the given demonstration (bottom rows). (c) The sequence of human demonstration images and the ones reached by the robot for the task of manipulating rope into 'S' shape. Our agent is able to successfully imitate the demonstration.

of visual recognition, the dependence of observations on the agent's own dynamics further complicates goal recognition, as the same goal can appear different while moving forward or turning during navigation.

We pose goal recognition as a binary classification problem that given an observation x_i and the goal x_g infers if x_i is close to x_g or not. Lacking expert supervision of goals, we draw goal observations at random from the agent's experience during exploration, since they are known to be feasible. For each such pseudo-goal, we consider observations that were only a few actions away to be positives (i.e., close to the goal) and the remaining observations that were more than a fixed number of actions (i.e., a margin) away as negatives. We trained the goal classifier using the standard cross-entropy loss. Like the skill policy, our goal recognizer is conditioned on the goal for generalization across goals. We found that training an independent goal recognition network consistently outperformed the alternative approach that augments the action space with a "stop" action. Making use of temporal proximity as supervision has



(a) TPS-RPM error for 'S' shape manipulation

Method	Success %
Inverse Model [Nair et.al. 2017]	36% ± 9.6%
Forward-regularized GSP	44% ± 9.9%
Forward-consistent GSP [Ours]	**60**% ± **9.8**%

(b) Success rate for Knot-tying

Figure 6.3: GSP trained using forward consistency loss significantly outperforms the baselines at the task of (a) manipulating rope into 'S' shape as measured by TPS-RPM error and (b) knot-tying where we report success rate with bootstrap standard deviation.

also been explored for feature learning in the concurrent work of Sermanet *et al.* [220].

6.2 Experiments

We evaluate our model by testing its performance on: rope manipulation using Baxter robot, navigation of a wheeled robot in cluttered office environments, and simulated 3D navigation. The key requirements of a good skill policy are that it should generalize to unseen environments and new goals while staying robust to irrelevant distractors in the observations. For rope manipulation, we evaluate generalization by testing the ability of the robot to manipulate the rope into configurations such as knots that were not seen during random exploration. For navigation, both real-world and simulation, we check generalization by testing on a novel building/floor.

Ablations and Baselines Our proposed formulation of GSP composed of following components: (a) recurrent variable-length skill policy network, (b) explicitly encoding previous action in the recurrence, (c) goal recognizer, (d) forward consistency loss function, and (w) learning forward dynamics in the feature space instead of raw observation space. We systematically ablate these components of forward-consistent GSP, to quantitatively review the importance of each component and then perform comparisons to the prior approaches that could be deployed for the task of visual imitation.

The following methods will be evaluated and compared to in the subsequent experiments section: (1) Classical methods: In visual navigation, we attempted to compare against the state-of-the-art open source classical methods, namely, ORB-SLAM2 [44, 158] and OpenSFM [150]. (2) Inverse Model: Nair *et al.* [161] leverage vanilla inverse dynamics to follow

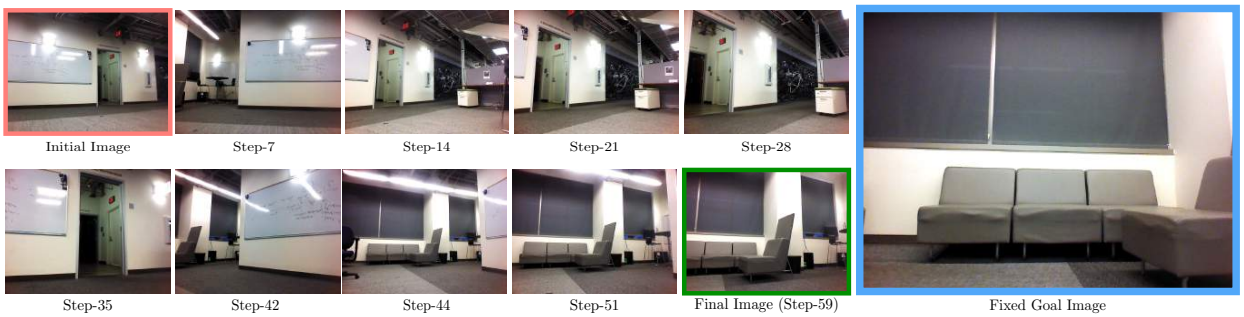


Figure 6.4: Visualization of the TurtleBot trajectory to reach a goal image (right) from the initial image (top-left). Since the initial and goal image have no overlap, the robot first explores the environment by turning in place. Once it detects overlap between its current image and goal image (i.e. step 42 onward), it moves towards the goal. Note that we did not explicitly train the robot to explore and such exploratory behavior naturally emerged from the self-supervised learning.

demonstration in rope manipulation setup. We compare to their method in both visual navigation and manipulation. (3) GSP-NoPrevAction-NoFwdConst is the ablation of our recurrent GSP without previous action history and without forward consistency loss. (4) GSP-NoFwdConst refers to our recurrent GSP with previous action history, but without forward consistency objective. (5) GSP-FwdRegularizer refers to the model where forward prediction is only used to regularize the features of GSP but has no role to play in the loss function of predicted actions. The purpose of this variant is to particularly ablate the benefit of consistency loss function with respect to just having forward model as feature regularizer. (6) GSP refers to our complete method with all the components. We now discuss the experiments and evaluate these baselines.

6.2.1 Rope Manipulation

Manipulation of non-rigid and deformable objects, e.g., rope, is a challenging problem in robotics. Even humans learn complex rope manipulation such as tying knots, either by observing an expert perform it or by receiving explicit instructions. We test whether our agent could manipulate ropes by simply observing a human perform it. We use the data collected by Nair *et al.* [161], where a Baxter robot manipulated a rope kept on the table in front of it. During exploration, the robot interacts with the rope by using a pick and place primitive that chooses a random point on the rope and displaces it by a randomly chosen length and direction. This process is repeated a number of times to collect about 60K interaction pairs of the form (x_t, a_t, x_{t+1}) that are used to train the GSP.

During inference, our proposed approach is tasked to follow a visual demonstration provided by a human expert for manipulating the rope into a complex 'S' shape and tying a knot. Our agent, Baxter robot, only gets to observe the image sequence of intermediate states, as human manipulates the rope, without any access to the corresponding actions. Note that the knot shape is never encountered during the self-supervised data collection

Model Name	Run Id-1	Run Id-2	Run Id-3	Run Id-4	Run Id-5	Run Id-6	Run Id-7	Run Id-8	Num Success
Random Search	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	0
Inverse Model [Nair et. al. 2017]	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	0
GSP-NoPrevAction-NoFwdConst	39 steps	34 steps	Fail	Fail	Fail	Fail	Fail	Fail	2
GSP-NoFwdConst	22 steps	22 steps	39 steps	48 steps	Fail	Fail	Fail	Fail	4
GSP (Ours)	119 steps	66 steps	144 steps	67 steps	51 steps	Fail	100 steps	Fail	6

Table 6.1: Quantitative evaluation of various methods on the task of navigating using a *single image* of goal in an unseen environment. Each column represents a different run of our system for a different initial/goal image pair. Our full GSP model takes longer to reach the goal on average given a successful run but reaches the goal successfully at a much higher rate.

phase and therefore the learned GSP model would have to generalize to be able to follow the human demonstration. More details follow in the supplementary material, Section B.1.

Metric The performance of the model is evaluated by measuring the non-rigid registration cost between the rope state achieved by the robot and the state demonstrated by the human at every step in the demonstration. The matching cost is measured using the thin plate spline robust point matching technique (TPS-RPM) described in [37]. While TPS-RPM provides a good metric for measuring performance for constructing the 'S' shape, it is not an appropriate metric for knots because the configuration of the rope in a knot is 3D due to intertwining of the rope, and it fails to find the correct point correspondences. We, therefore, use success rate as the metric in knot tying where the completion of a successful knot is judged by human verification.

Visual Imitation Qualitative examples of our agent trying to manipulate rope are shown in Figure 6.2. We compare our approach to the baseline that deploys an inverse model which takes as input a pair of current and goal images to output the desired action to reach the goal [161]. We re-implement the baseline and train in our setup for a fair comparison. To further ablate the importance of consistency loss, we compare to a baseline that just uses a forward model as a regularizer of features. The results in Figure 6.3 show that our method significantly outperforms the baseline at task of manipulating the rope in the 'S' shape and achieves a success rate of 60% in comparison to 36% achieved by the baseline.

6.2.2 Navigation in Indoor Office Environments

A natural way to instruct a robot to move in an indoor office environment is to ask it to go near a certain location, such as a refrigerator or a someone's office. Instead of using language to command the robot, in this chapter, we communicate with the robot by either showing it a single image of the goal, or a sequence of images leading to faraway goals. In both scenarios, the robot is required to autonomously determine the motor commands for moving to the goal. We used TurtleBot2 for navigation using an onboard camera for sensing RGB images. For learning the GSP, an automated self-supervised scheme for data collection was devised that doesn't require human supervision. The robot collected a number of navigation trajectories from two floors of a academic building which in total contain 230K interactions data, i.e.

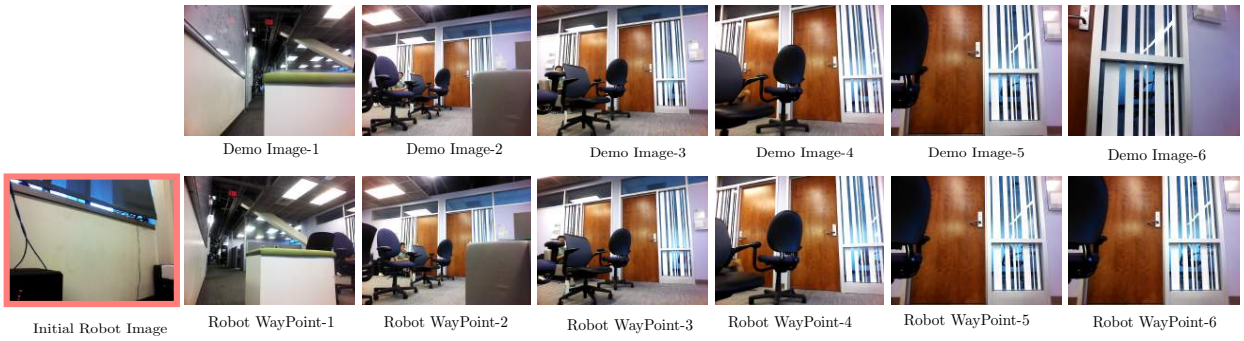


Figure 6.5: The performance of TurtleBot at following a visual demonstration given as a sequence of images (top row). The TurtleBot is positioned in a manner such that the first image in demonstration has no overlap with its current observation. Even under this condition the robot is able to move close to the first demo image (shown as Robot WayPoint-1) and then follow the provided demonstration until the end. This also exemplifies a failure case for classical methods; there are no possible keypoint matches between WayPoint-1 and WayPoint-2, and the initial observation is even farther from WayPoint-1.

(x_t, a_t, x_{t+1}). We then deployed the learned model on a separate floor of a building with substantially different textures and furniture layout for performing visual imitation at test time. The details of the robotic setup, data collection, and network architecture of GSP are described in supplementary material, Section B.2.

1) Goal Finding We first tested if the GSP learned by the TurtleBot can enable it to find its way to a goal that is within the same room from just a *single image* of the goal. To test the extrapolative generalization, we keep the Turtlebot approximately 20-30 steps away from the target location in a way that current and goal observations have no overlap as shown in Figure 6.4. We test the robot in an indoor office environment on a different floor that it has never encountered before. We judge the robot to be successful if it stops close to the goal and failure if it crashed into furniture or does not reach the goal within 200 steps. Since the initial and goal images have no overlap, classical techniques such as structure from motion that rely on feature matching cannot be used to infer the executed action. Therefore, in order to reach the goal, the robot must explore its surroundings. We find that our GSP model outperforms the baseline models in reaching the target location. Our model learns the exploratory behavior of rotating in place until it encounters an overlap between its current and goal image. Results are shown in Table 6.1 and videos are available at the website [1].

2) Visual Imitation In the previous paragraph, we saw that the robot can reach a goal that's within the same room. However, our agent is unable to reach far away goals such as in other rooms using just a single image. In such scenarios, an expert might communicate instructions like go to the door, turn right, go to the closest chair etc. Instead of language instruction, in our setup we provide a sequence of *landmark* images to convey the same high-level idea. These landmark images were captured from the robot's camera as the expert

[1] https://pathak22.github.io/zeroshot-imitation/

Model Name	Maze Demonstration			Loop Demonstration		
	Run-1	Run-2	Run-3	Run-1	Run-2	Run-3
SIFT	10%	5%	15%	—	—	—
GSP-NoPrevAction-NoFwdConst	60%	70%	100%	—	—	—
GSP-NoFwdConst	65%	90%	100%	0%	0%	0%
GSP (ours)	100%	60%	100%	0%	100%	100%

Table 6.2: Quantitative evaluation of TurtleBot's performance at following visual demonstrations in two scenarios: maze and the loop. We report the % of landmarks reached by the agent across three runs of two different demonstrations. Results show that our method outperforms the baselines. Note that 3 more trials of the loop demonstration were tested under significantly different lighting conditions and neither model succeeded. Detailed results are available in the supplementary materials.

moved the robot from the start to a goal location. However, note that it is not necessary for the expert to control the robot to capture the images because we don't make use of the expert's actions, but only the images. Instead of providing the image after every action in the demonstration, we only provided every fifth image. The rationale behind this choice is that we want to sample the demonstration sparsely to minimize the agent's reliance on the expert. Such sub-sampling (as shown in Figure 6.5) provides an easy way to vary the complexity of the task.

We evaluate via multiple runs of two demonstrations, namely, maze demonstration where the robot is supposed to navigate through a maze-like path and perturbed loop demonstration, where the robot is supposed to make a complete loop as instructed by demonstration images. The loop demonstration is longer and more difficult than the maze. We start the agent from different starting locations and orientations with respect to that of demonstration. Each orientation is initialized such that no part of the demonstration's initial frame is visible. Results are shown in Table 6.2. When we sample every frame, our method and classical structure from motion can both be used to follow the demonstration. However, at sub-sampling rate of five, SIFT-based feature matching approaches did not work and ORBSLAM2 [158] failed to generate a map, whereas our method was successful. Notice that providing sparse landmark images instead of dense video adds robustness to the visual imitation task. In particular, consider the scenario in which the environment has changed since the time the demonstration was recorded. By not requiring the agent to match every demonstration image frame-by-frame, it becomes less sensitive to changes in the environment.

6.2.3 3D Navigation in VizDoom

We have evaluated our approach on real-robot scenarios thus far. To further analyze the performance and robustness of our approach through large scale experiments, we setup the same navigation task as described in previous subsection in a simulated VizDoom environment.

Our goal is to measure: (1) the robustness of each method with proper error bars, (2) the role of initial self-supervised data collection for performance on visual imitation, (3) the quantitative difference in modeling forward consistency loss in feature space in comparison to raw visual space.

In VizDoom, we collect data by deploying two types of exploration methods: random exploration and curiosity-driven exploration (see Chapter 4). The hypothesis is that if the initial data collected by the robot is driven by a better strategy than just random, this should eventually help the agent follow long demonstrations better. Our environment consists of 2 maps in total. We train on one map with 5 different starting positions for collecting exploration data. For validation, we collect 5 human demonstrations in a map with the same layout as in training but with different textures. For zero-shot generalization, we collect 5 human demonstrations in a novel map layout with novel textures. Exact details for data collection and training setup are in the supplementary, Section B.3.

Metric We report the median of maximum distance reached by the robot in following the given sequence of demonstration images. The maximum distance reached is the distance of farthest landmark point that the agent reaches contiguously, i.e., without missing any intermediate landmarks. Measuring the farthest landmark reached does not capture how efficiently it is reached. Hence, we further measure efficiency of the agent as the ratio of number of steps taken by the agent to reach farthest contiguous landmark with respect to the number of steps shown in human demonstrations.

Visual Imitation The task here is same as the one in real robot navigation where the agent is shown a sparse sequence of images to imitate. The results are in Table 6.3. We found that the exploration data collected via curiosity significantly improves the final imitation performance across all methods including the baselines with respect to random exploration. Our baseline GSP model with a forward regularizer instead of consistency loss ends up overfitting to the training layout. In contrast, our forward-consistent GSP model outperforms other methods in generalizing to new map with novel textures. This indicates that the forward consistency is possibly doing more than just regularizing the policy features. Training forward consistency loss in feature space further enhances the generalization even when both pixel and feature space models perform similarly on training environment.

6.3 Related Work

Our work is closely related to imitation learning, but we address a different problem statement that gives less supervision and requires generalization across tasks during inference.

Imitation Learning The two main threads of imitation learning are behavioral cloning [10, 197], which directly supervises the mapping of states to actions, and inverse reinforcement learning [1, 100, 138, 163, 280], which recovers a reward function that makes the demonstration optimal (or nearly optimal). Inverse RL is most commonly achieved with state-actions, and is difficult to extend to fitting the reward to observations alone, though in principle state occupancy could be sufficient. Recent works in imitation learning [57, 67, 87] can generalize

Model Name	Same Map, Same Texture		Same Map, Diff Texture		Diff Map, Diff Texture	
	Median %	Efficiency %	Median %	Efficiency %	Median %	Efficiency %
Random Exploration for Data Collection:						
GSP-NoFwdConst	63.2 ± 5.7	36.4 ± 3.3	32.2 ± 0.7	28.9 ± 4.0	34.5 ± 0.6	23.1 ± 2.4
GSP (ours pixels)	62.2 ± 5.1	43.0 ± 2.6	32.4 ± 0.8	30.9 ± 2.9	35.4 ± 1.1	29.3 ± 3.9
GSP (ours features)	68.9 ± 6.9	53.9 ± 4.0	32.4 ± 0.7	47.4 ± 7.6	39.1 ± 2.0	30.4 ± 2.5
Curiosity-driven Exploration for Data Collection:						
GSP-NoFwdConst	78.2 ± 2.3	63.0 ± 4.3	43.2 ± 2.6	33.9 ± 3.0	40.2 ± 4.0	27.3 ± 1.9
GSP-FwdRegularizer	78.4 ± 3.4	59.8 ± 4.1	50.6 ± 4.7	30.9 ± 3.0	37.9 ± 1.1	28.9 ± 1.7
GSP (ours pixels)	78.2 ± 3.4	65.2 ± 4.2	47.1 ± 4.7	32.4 ± 3.0	44.8 ± 4.0	29.5 ± 1.9
GSP (ours features)	78.2 ± 4.6	67.0 ± 3.3	49.4 ± 4.8	26.9 ± 1.5	47.1 ± 3.0	24.1 ± 1.7

Table 6.3: Quantitative evaluation of our proposed GSP and the baseline models at following visual demonstrations in VizDoom 3D Navigation. Medians and 95% confidence intervals are reported for demonstration completion and efficiency over 50 seeds and 5 human paths per environment type.

to novel goals, but require a wealth of demonstrations comprised of expert state-actions for learning. Our approach does not require expert actions at all.

Visual Demonstration The common scenario in LfD is to assume full knowledge of expert states and actions during demonstrations, but several papers have focused on relaxing this supervision to visual observations alone. Nair *et al*. [161] observe a sequence of images from the expert demonstration for performing rope manipulations. Sermanet *et al*. [220, 221] imitate humans with robots by self-supervised learning but require expert supervision at training time. Third person imitation learning [233] and the concurrent work of imitation-from-observation [144] learn to translate expert observations into agent observations such that they can do policy optimization to minimize the distance between the agent trajectory and the translated demonstration, but they require demonstrations for learning. Visual servoing is a standard problem in robotics [124] that seeks to take actions that align the agent's observation with a target configuration of carefully-designed visual features [260, 272] or raw pixel intensities [32]. Classical methods rely on fixed features or policies, but more recently end-to-end learning has improved results [131, 135].

Forward/Inverse Dynamics and Consistency Numerous prior works, such as [60, 166, 255], have learned forward dynamics model for planning actions. The works of [5, 113, 179, 261] jointly learn forward and inverse dynamics model but do not optimize for consistency between the forward and inverse dynamics. We empirically show that learning models by our forward consistency loss significantly improves task performance. Enforcing consistency as a meta-supervision has also been successful in finding visual correspondences [278] or unpaired image translations [279].

Goal Conditioning By parameterizing the value or policy function with a goal, an agent can learn and do multiple tasks. The idea of learning goal-conditioned policies has been explored in [5, 9, 161, 213]. Similarly to hindsight experience replay [9] we draw goals from experience, but our policy optimization has better sample efficiency through supervised

learning and dynamics modeling instead of reinforcement learning. Moreover, we work from high-dimensional visual inputs instead of knowledge of the true states and do not make use of a task reward during training. In our setting, all of the expert goals are followed zero-shot since they are only revealed after learning.

6.4 Discussion

In this chapter, we presented a method for imitating expert demonstrations from visual observations alone. In contrast to most work in imitation learning, we never require access to expert actions. The key idea is to learn a GSP using data collected by self-supervised exploration. However, this limits the quality of the learned GSP as per the exploration data. For instance, we deploy random exploration on our real-world navigation robot, which means that it would almost never follow trajectories that go between rooms. Consequently, the learned GSP is unable to navigate towards a goal image taken in another room without requiring intermediate sub-goals. We showed in Chapter 4 that the agent learns to move along corridors and transition between rooms purely driven by curiosity in VizDoom. Training GSP on such a structured data could equip the agent with more interesting search behaviors, e.g., going across rooms to find a goal. In general, using better methods of exploration for training the GSP could be a fruitful direction toward generalizing zeroshot imitation.

One limitation of our approach is that we require first-person view demonstrations. Extension to third-person demonstrations [144, 233] would make the method applicable in more general scenarios. Another limitation is that, in the current framework, it is implicitly assumed that the statistics of visual observations when the expert demonstrates the task and the agent follows it are similar. For e.g., when the expert performs a demonstration in one setting, say in daylight and the agent needs to imitate say in the evening, the change in the lighting conditions might result in worse performance. Making the GSP robust to such nuisance changes or other changes in environment by domain adaptation would be necessary to scale the method to practical problems. Another thing to note is that, in the current framework, we do not learn from expert demonstrations, but simply imitate them. It would be interesting to investigate ways for an agent to learn from the expert to bias its exploration to more useful parts of the environment.

While we used a sequence of images to provide a demonstration, our work makes no image-specific assumptions and can be extended to using formal language for communicating goals. For instance, after training the GSP, instead of transforming an image into features ϕ as described in section 6.1.2, one could possibly learn a mapping to transform language instructions into this feature space.

Part IV

Generalization via Modularity

Chapter 7

Learning to Control Modular Self-Assembling Morphologies

People who are really serious about software should make their own hardware.
— Alan Kay

Possibly the single most pivotal event in the history of evolution was the point when single-celled organisms switched from always competing with each other for resources to sometimes cooperating, first by forming colonies, and later by merging into multicellular organisms [6]. These modular self-assemblies were successful because they combined the high adaptability of single-celled organisms while making it possible for vastly more complex behaviors to emerge. Indeed, one could argue that it is this modular design which allowed the multicellular organisms to successfully adapt, increase in complexity, and generalize to the constantly changing environment of prehistoric Earth. Like many researchers before us [159, 228, 245, 270, 271], we are inspired by the biology of multicellular evolution as a model for emergent complexity in artificial agents. Unlike most previous work however, we are primarily focused on modularity as a way of improving adaptability and generalization to *novel test-time scenarios*.

In this chapter, we present a study of modular self-assemblies of primitive agents — "limbs" — which can link up to solve a task. Limbs have the option to bind together by a magnet that connects their morphologies within magnetic range (Figure 7.1), and when they do so, they pass messages and share rewards. Each limb comes with a simple neural net that controls the torque applied to its joints. Linking and unlinking are treated as dynamic actions so that the limb assembly can change shape within an episode. Similar setup has previously been explored in robotics as "self-reconfiguring modular robots" [238]. However, unlike prior work on such robots, where the control policies are hand-defined, we show how to *learn* the policies and study the generalization properties that emerge.

This chapter is based on Pathak *et al*. 2019 [185].

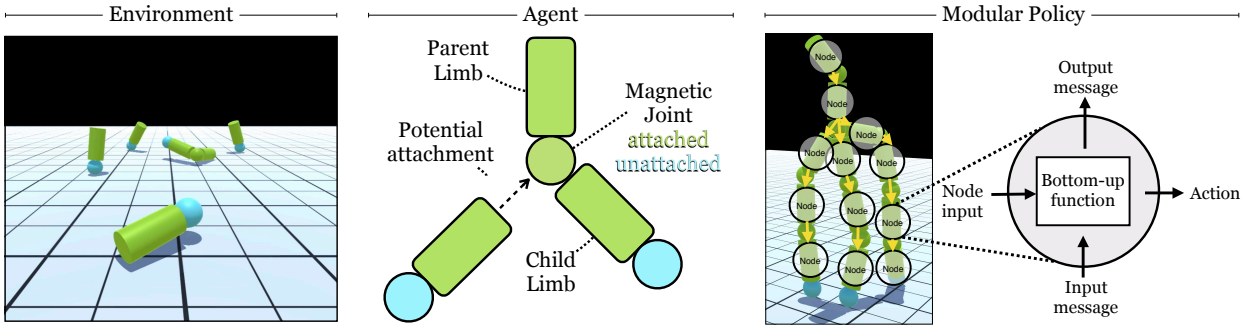


Figure 7.1: This work investigates the joint learning of control and morphology in self-assembling agents. Several primitive agents, containing a cylindrical body with a configurable motor, are dropped in a simulated environment (left). These primitive agents can self-assemble into collectives using magnetic joints (middle). Policy of the self-assembled agent is represented via proposed dynamic graph networks (DGN) with shared parameters (modular) across each limb (right).

Our self-assembled agent can be represented as a graph of primitive limbs. Limbs pass messages to their neighbors in this graph in order to coordinate behavior. All limbs have a common policy network with shared parameters, i.e., a modular policy which takes the messages from adjacent limbs as input and outputs a torque to rotate the limb in addition to the linking/un-linking action. We call the aggregate neural network a Dynamic Graph Network (DGN) since it is a graph neural network [210] that can dynamically change topology as a function of its own outputs.

We test our dynamic limb assemblies on two separate tasks: standing up and locomotion. We are particularly interested in assessing how well can the assemblies generalize to novel testing conditions, not seen at training, compared to static and monolithic baselines. We evaluate test-time changes to both the environment (changing terrain geometry, environmental conditions), as well as the agent structure itself (changing the number of available limbs). We show that the dynamic self-assembles are better able to generalize to these changes than the baselines. For example, we find that a single modular policy is able to control multiple possible morphologies, even those not seen during training, e.g., a 6-limb policy, trained to build a 6-limb tower, can be applied at test time on 3 or 12 limbs, and still able to perform the task.

7.1 Environment and Agents

Investigating the co-evolution of control (i.e., *software*) and morphology (i.e., *hardware*) is not supported within standard benchmark environments typically used for sensorimotor control, requiring us to create our own. We opted for a minimalist design for our agents, the environment, and the reward structure, which is crucial to ensuring that the emergence of limb assemblies with complex morphologies is not forced, but happens naturally.

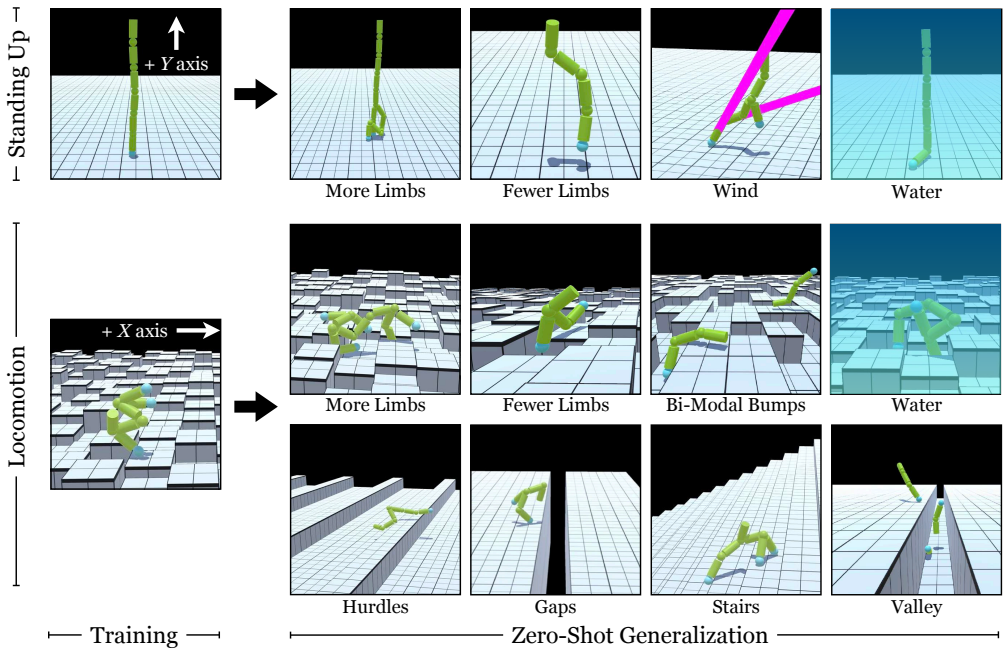


Figure 7.2: We illustrate our dynamic agents in two environments / tasks: standing up and locomotion. For each of these, we generate several new environment for evaluating generalization. Refer to project video at https://pathak22.github.io/modular-assemblies/ for better understanding of tasks.

Environment Structure Our environment contains an arena where a collection of primitive agent limbs can self-assemble to perform control tasks. This arena is a ground surface equipped with gravity and friction. The arena can be procedurally changed to generate a variety of novel terrains by changing the height of each tile on the ground (see Figure 7.2). To evaluate the generalization properties of our agents, we generate a series of novels terrains. This include generating bumpy terrain by randomizing the height of nearby tiles, stairs by incrementally increasing the height of each row of tiles, hurdles by changing the height of each row of tiles, gaps by removing alternating rows of tiles, etc. Some variations also include putting the arena 'under water' which basically amounts to increased drag (i.e. buoyancy). During training, we start our environment with a set of six primitive limbs on the ground which can assemble to form collectives to perform complex tasks.

Agent Structure All limbs share the same structure: a cylindrical body with a configurable motor on one end and the other end is free. The free-end of the limb can link up with the motor-end of the other limb, and then the motor acts as a joint between two limbs with three degrees of rotation. Hence, one can refer to the motor-end of the cylindrical limb as a *parent-end* and the free end as a *child-end*. Multiple limbs can attach their child-end to the parent-end of another limb, as shown in Figure 7.1, to allow for complex graph morphologies to emerge. The limb of the parent-end controls the torques of joint. The unlinking action can

be easily implemented by detaching two limbs, but the linking action has to deal with the ambiguity of which limb to connect to (if at all). To resolve this, we implement the linking action by attaching the closest limb within a small radius around the parent-node. The attachment mechanism is driven by a magnet inside the parent node which forces the closest child-limb within the magnetic range node to get docked onto itself if the parent signals to connect. If no other limb is present within the magnetic range, the linking action has no effect.

The primitive limbs are dropped in an environment to jointly solve a given control task. One key component of the self-assembling agent setup that makes it different from typical multi-agent scenarios [262] is that if some agents assemble to form a collective, the resulting morphology becomes a new *single agent* and all limbs within the morphology maximize a joint reward function. The output action space of each primitive agent contains the continuous torque values that are to be applied to the motor connected to the agent, and are denoted by $\{\tau_\alpha, \tau_\beta, \tau_\gamma\}$ for three degrees of rotation. In addition to the torque controls, each limb can decide to attach another link at its parent-end, or decide to unlink its child-end if already connected to other limb. The linking and unlinking decisions are binary. This complementary role assignment of child and parent ends, i.e., parent can only link and child can only unlink, makes it possible to decentralize the control across limbs.

Sensory Inputs In our self-assembling setup, each agent limb only has access to its local sensory information and does not know about other limbs. The sensory input of each agent includes its own dynamics, i.e., the location of the limb in 3-D euclidean coordinates, its velocity, angular rotation and angular velocity. Each end of the limb also has a trinary touch sensor to detect whether the end of the cylinder is touching 1) the floor, 2) another limb, or 3) nothing. Additionally, we also provide our limbs with a point depth sensor that captures the surface height on a 9×9 grid around the projection of center of limb on the surface.

One essential requirement to operationalize this setup is an efficient simulator to allow simultaneous simulation of several of these primitive limbs. We implement our environments in the Unity ML [114] framework, which is one of the dominant platforms for designing realistic games. For computational reasons, we do not allow the emergence of cycles in the self-assembling agents by not allowing the limbs to link up with already attached limbs within the same morphology. However, our setup is trivially extensible to general graphs.

7.2 Learning to Control Self-Assemblies

Consider a set of primitive limbs indexed by i in $\{1, 2, \ldots, n\}$ dropped in an environment arena $\mathcal{E}$ to perform a continuous control task. If needed, these limbs can assemble to form complex collectives in order to improve their performance on the task. The task is represented by a reward function r_t and the goal of the limbs is to maximize the discounted sum of rewards over time t. If some limbs assemble into a collective, the resulting morphology effectively becomes a single agent with a combined policy to maximize the combined reward of the

connected limbs. Further, the reward of an assembled morphology is a function of the whole morphology and not the individual agent limbs. For instance, in the task of learning to stand up, the reward is the height of the individual limbs if they are separate, but is the height of the whole morphology if those limbs have assembled into a collective.

7.2.1 Co-evolution: Linking/Unlinking as an Action

To learn a modular controller policy that could generalize to novel setups, our agents must learn the controller jointly as the morphology evolves over time. The limbs should simultaneously decide which torques to apply to their respective motors, while taking into account the connected morphology. Our hypothesis is that if a controller policy could learn in a modular fashion over iterations of increasingly sophisticated morphologies (see Figure 7.3), it could learn to be robust and generalizable to diverse situations. So, how can we optimize control and morphology under a common end-to-end framework?

We propose to treat the decision of linking and unlinking as additional actions of our primitive limbs. The total action space a_t at each iteration t can be denoted as $\{\tau_\alpha, \tau_\beta, \tau_\gamma, \sigma_{link}, \sigma_{unlink}\}$ where τ_* denote the raw *continuous* torque values to be applied at the motor and σ_* denote the *binary* actions whether to connect another limb at the parent-end or disconnect the child-end from the other already attached limb. This simple view of morphological evolution allows us to use ideas from RL [241].

7.2.2 Modularity: Self-Assembly as a Graph of Limbs

Integration of control and morphology in a common framework is only the first step. The key question is how to model this controller policy such that it is modular and reuses information across generations of morphologies. Let a_t^i be the action space and s_t^i be the local sensory input-space of the agent i. One naive approach to maximizing the reward is to simply combine the states of the limbs into the input-space, and output all the actions jointly using a single network. Formally, the policy is simply $\vec{a}_t = [a_t^0, a_t^1 \dots a_t^n] = \Pi(s_t^0, s_t^0 \dots, s_t^n)$. This interprets the self-assemblies as a single monolithic agent, ignoring the graphical structure. This is the current approach to solve many control problems, e.g., Mujoco humanoid [28] where the policy Π is trained to maximize the sum of rewards using RL.

In this work, we represent the agent's policy via a graph neural network [210] in such a way that it explicitly corresponds to the morphology of the agent. Consider a collection of primitive limbs as graph G where each node is a limb i. Two limbs being physically connected by a joint is analogous to having an edge in the graph. As discussed in Section-7.1, each limb has two endpoints, a *parent-end* and a *child-end*. At a joint, the limb which connects via its parent-end acts as a parent-node in the corresponding edge, and the other limbs, which connect via their child-ends, are child-nodes. The parent-node (i.e., the agent with the parent-end) controls the torque of the edge (i.e., the joint motor).

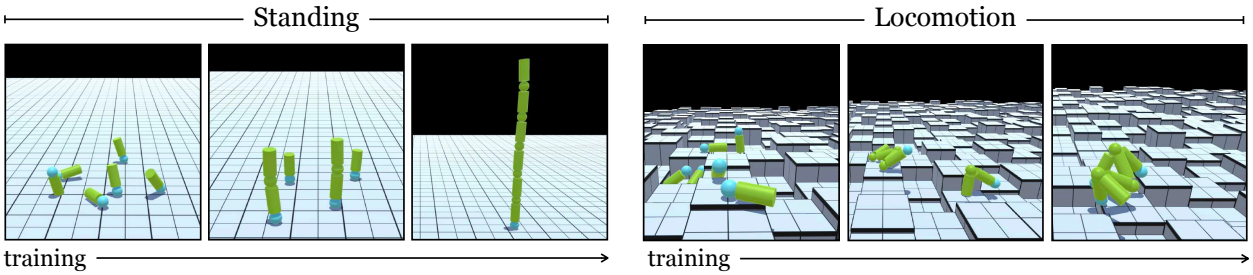


Figure 7.3: Co-evolution of Morphology w/ Control during Training: The gradual co-evolution of controller as well as the morphology of self-assembling agents over the course of training for the task of Standing Up (left) and Locomotion (right).

7.2.3 Dynamic Graph Networks (DGN)

Each primitive limb node i has a policy controller of its own, which is represented by a neural network π_θ^i and receives a corresponding reward r_t^i for each time step t. We represent the policy of the self-assembled agent by the aggregated neural network that is connected in the same graphical manner as the physical morphology. The edge connectivity of the graph is represented in the overall graph policy by passing messages that flow from each limb to the other limbs physically connected to it via a joint. The parameters θ are shared across each primitive limbs allowing the overall policy of the graph to be modular with respect to each node. However, recall that the agent morphologies are dynamic, i.e., the connectivity of the limbs changes based on policy outputs. This changes the edge connectivity of the corresponding graph network at every timestep, depending on the actions chosen by each limb's policy network in the previous timestep. Hence, we call this aggregate neural net a *Dynamic Graph Network (DGN)* since it is a graph neural network that can dynamically change topology as a function of its own outputs in the previous iteration.

DGN Optimization A typical rollout of our self-assembling agents during an episode of training contains a sequence of torques τ_t^i and the linking actions σ_t^i for each limb at each timestep t. The policy parameters θ are optimized to jointly maximize the reward for each limb:

$$\max_\theta \sum_{i=\{1,2...,n\}} \mathbb{E}_{\vec{a}^i \sim \pi_\theta^i}[\Sigma_t r_t^i] \tag{7.1}$$

We optimize this objective via policy gradients, in particular, PPO [218]. DGN pseudo-code (as well as source code) and all training implementation details are in Section C.1, C.3 of the appendix.

DGN Connectivity The topology is captured in the DGN by passing messages through the edges between individual network nodes. Since the parameters of these limb networks are shared across each node, these messages can be seen as context information that may inform the policy of its role in the corresponding connected component of graph.

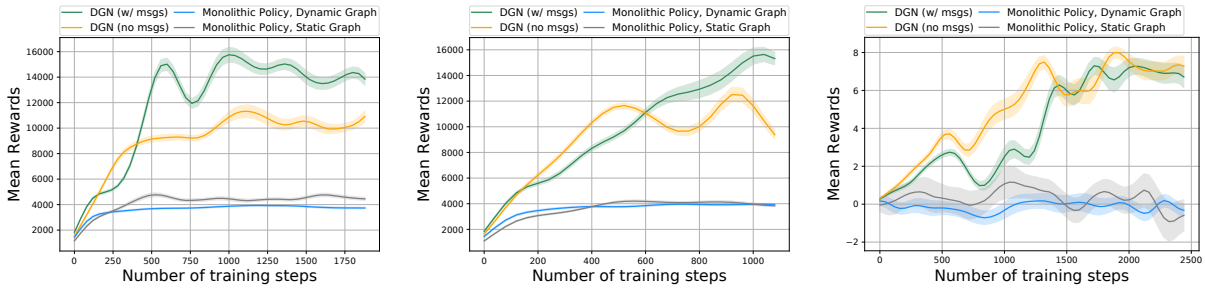
(a) Message passing: Messages are passed from leaf nodes to root, i.e., each agent gets information from its children. Instead of defining π^i_θ to be just as a function of state s^i_t, we pass each limb's policy network information about its children nodes. We redefine π^i_θ as $\pi^i_\theta : [s^i_t, m^{C_i}_t] \rightarrow [a^i_t, m^i_t]$ where m^i_t is the output message of policy that goes into the parent limb and $m^{C_i}_t$ is the aggregated input messages from all the children nodes, i.e, $m^{C_i}_t = \sum_{c \in C_i} m^c_t$. If i has no children (i.e, root), a vector of zeros is passed in $m^{C_i}_t$. Messages are passed recursively until the root node. Alternative way is to start from root node and recursively pass until the messages reach the leaf nodes.

(b) No message passing: Note that for some environments or tasks, the context from the other nodes might not be a necessary requirement for effective control. In such scenarios, message passing might creates extra overhead for training a DGN. Importantly, even with no messages, DGN still allows for coordination between limbs. This is similar to a typical cooperative multi-agent setup [262] where each limb makes its own decisions in response to the previous actions of the other agents. However, our setup differs in that our agents may physically join up, rather than just coordinating behavior.

7.3 Experiments

We test the co-evolution of morphology and control across two primary tasks where self-assembling agents learn to: (a) stand up, and (b) perform locomotion. Limbs start each episode disconnected and located just above the ground plane at random locations, as shown in Figure 7.3. In the absence of an edge, input messages are set to 0 and the output ones are ignored. Action space is continuous raw torque values. Across all the tasks, the number of limbs at training is kept fixed to 6. We take the model from each time step and evaluate it on 50 episode runs to plot mean and std-deviation confidence interval in training curves. At test, we report the mean reward across 50 episodes of 1200 environment steps. The main focus of our investigation is to evaluate if the emerged modular controller generalizes to novel morphologies and environments. Video is on the project website.

Baselines We further compare how well these dynamic morphologies perform in comparison to a learned monolithic policy for both dynamic and fixed morphologies. In particular, we compare to a (a) *Monolithic Policy, Dynamic Graph*: Baseline where agents are still dynamic and can self-assemble, but their controller is represented by a single monolithic policy that takes as input the combined state of all agents and outputs actions for each of them. (b) *Monolithic Policy, Fixed Graph*: Similar single monolithic policy as previous baseline, but the morphology is hand-designed constructed from the limbs and kept fixed and static during training and test. This is analogous to a standard robotics "vanilla RL" setup in which a morphology is predefined and then a policy is learned to control it. We chose the fixed morphology to be a straight chain of 6-limbs in all the experiments. This linear-chain may be optimal for standing as tall as possible, but it is not necessarily optimal for *learning* to stand; the same would hold for locomotion. However, we confirmed that the both standing and



(a) Standing Up (b) Standing Up w/ Wind (c) Locomotion

Figure 7.4: Training self-assembling agents: We show the performance of different methods for joint training of control and morphology for three tasks: learning to stand up (left), standing up in the presence of wind (center) and locomotion in bumpy terrain (right). These policies generalize to novel scenarios as shown in the tables.

locomotion task are solvable with linear-chain morphology (shown in Figure 7.3 and video on project website).

Although monolithic policy is more expressive (complete state information of all limbs), it is also harder to train as we increase the number of limbs, because the observation and action spaces increase in dimensionality. Indeed, this is what we find in the appendix Figure C.1: the monolithic policy can perform well on up to three limbs, but does not reach the optimum on four to six limbs. In contrast, the DGN limb policy (shared between all limbs) has a fixed size observation and action space, independent of the number of limbs under control.

7.3.1 Learning to Self-Assemble

We first validate if it is possible to train self-assembling agent policy end-to-end via Dynamic Graph Networks. Below, we discuss our environments and compare the training efficiency of each method.

Standing Up Task In this task, each agent's objective is to maximize the height of the highest point in its morphology. Limbs have an incentive to self-assemble because the potential reward would scale with the number of limbs if the self-assembled agent can control them. The training process begins with six-limbs falling on the ground randomly, as shown in Figure 7.3. These limbs act independently in the beginning but gradually learn to self-assemble as training proceeds. Figure 7.4a compares the training efficiency and performance of different methods during training. We found that our DGN policy variants perform significantly better than the monolithic policies for the standing up task.

Standing Up in the Wind Task Same as the previous task, except with the addition of 'wind', which we operationalize as random forces applied to random points of each limb at

Environment	DGN (ours)	Monolithic Policy (dynamic)	(fixed)
Standing Up Task			
Training Environment			
Standing Up	**17518**	4104	5351
Zero-Shot Generalization			
More (2x) Limbs	**19796** (113%)	n/a	n/a
Fewer (.5x) Limbs	**10839** (62%)	n/a	n/a
Standing Up in the Wind Task			
Training Environment			
Stand-Up in Wind	**18423**	4176	4500
Zero-Shot Generalization			
2x Limbs + (S)Wind	**15351** (83%)	n/a	n/a
Locomotion Task			
Training Environment			
Locomotion	**8.71**	0.96	2.96
Zero-Shot Generalization			
More (2x) Limbs	**5.47** (63%)	n/a	n/a
Fewer (.5x) Limbs	**6.64** (76%)	n/a	n/a

Table 7.1: Zero-Shot Generalization to Number of Limbs: Quantitative evaluation of the generalizability of the learned policies. For each method, we first pick the best performing model from the training and then evaluate it on each of the novel scenarios without further finetuning, i.e., in a zero-shot manner. We report the score attained by the self-assembling agent along with the percentage of training performance retained upon transfer in parenthesis. Higher is better.

Environment	DGN (ours)	Monolithic Policy (dynamic)	(fixed)
Standing Up Task			
Training Environment			
Standing Up	**17518**	4104	5351
Zero-Shot Generalization			
Water + 2x Limbs	**16871** (96%)	n/a	n/a
Winds	**16803** (96%)	3923 (96%)	4531 (85%)
Strong Winds	**15853** (90%)	3937 (96%)	4961 (93%)
Standing Up in the Wind Task			
Training Environment			
Stand-Up in Wind	**18423**	4176	4500
Zero-Shot Generalization			
(S)trong Wind	**17384** (94%)	4010 (96%)	4507 (100%)
Water+2x+SWd	**17068** (93%)	n/a	n/a
Locomotion Task			
Training Environment			
Locomotion	**8.71**	0.96	2.96
Zero-Shot Generalization			
Water + 2x Limbs	**6.57** (75%)	n/a	n/a
Hurdles	**6.39** (73%)	-0.77 (-79%)	-3.12 (-104%)
Gaps in Terrain	**3.25** (37%)	-0.32 (-33%)	2.09 (71%)
Bi-modal Bumps	**6.62** (76%)	-0.56 (-57%)	-0.44 (-14%)
Stairs	**6.6** (76%)	-8.8 (-912%)	-3.65 (-122%)
Inside Valley	**5.29** (61%)	0.47 (48%)	-1.35 (-45%)

Table 7.2: Zero-Shot Generalization to Novel Environments: The best performing model from the training is evaluated on each of the novel scenarios without any further finetuning. The score attained by the self-assembling agent is reported along with the percentage of training performance retained upon transfer in parenthesis. Higher value is better.

random times, see Figure 7.2(Wind). Figure 7.4b shows the superior performance of DGN compared to the baselines.

Locomotion Task The reward function for locomotion is defined as the distance covered by the agent along X-axis. The training is performed on a bumpy terrain shown in Figure 7.2. The training performance in Figure 7.4c shows that DGN variants outperform the monolithic baselines.

As shown in Figure 7.4, training our DGN algorithm with message passing either seems to perform better or similar to the one without message passing. In particular, message passing is significantly helpful where long-term reasoning is needed across limbs, for instance,

messages help in standing up task because there is only one morphological structure to do well (i.e., linear tower). In locomotion, it is possible to do well with a large variety of morphologies, and thus both DGN variants reach similar performance. Now onwards, we show results using DGN w/ msgs as our primary approach.

7.3.2 Zero-Shot Generalization to Number of Limbs

We investigate if our trained policy generalizes to changes in the number of limbs. We pick the best model from training and evaluate it without any finetuning at test-time, i.e., zero-shot generalization.

Standing Up Task We train the policy with 6 limbs and test with 12 and 4 limbs. As shown in Table 7.1, despite changes in number of limbs, DGN is able to retain similar performance w/o any finetuning. The co-evolution of morphology jointly with the controller allows the modular policy to experience increasingly complex morphological structures. We hypothesize that this morphological curriculum at training makes the agent more robust at test-time.

Note that we can not generalize *Monolithic policy* baselines to scenarios with more or fewer limbs because they can't accommodate different action and state space dimensions from training; it has to be retrained. Hence, we made a comparison to DGN by retraining baseline on Standing task: DGN is trained on 6 limbs and tested on 4 limbs w/o any finetuning, while baseline is trained both times. DGN achieves 17518 (6limbs - train), 10839 (4limbs - test) scores, while baseline achieves 5351 (6limbs - train), 7356 (4limbs - train). Even without any training on 4 limbs, DGN outperforms baseline because it is difficult to train monolithic policy with large action space (Figure C.1 in the appendix).

Standing Up in the Wind Task Similarly, we evaluate the agent policy trained for standing up task in winds with 6 limbs to 12 limbs. Table 7.1 shows that the DGN performs significantly better than monolithic policy at train, and able to retain most of its performance even with twice the limbs.

Locomotion Task We also evaluate the generalization of locomotion policies trained with 6 limbs to 12 and 4 limbs. As shown in Table 7.1, DGN not only achieves good performance at training but is also able to retain most of its performance.

7.3.3 Zero-Shot Generalization to Novel Environments

We now evaluate the performance of our modular agents in novel terrains by creating several different scenarios by varying environment conditions (described in Section 7.1) to test zero-shot generalization.

Standing Up Task We test our trained policy without any further finetuning in environments with increased drag (i.e., 'under water'), and adding varying strength of random push-n-pulls (i.e. , 'wind'). Table 7.2 shows that DGN seems to generalize better than monolithic policies. We believe that this generalization is result of both the learning being modular as well as the fact that limbs learned to assemble in physical conditions (e.g. forces like gravity) with gradually growing morphologies. Such forces with changing morphology are similar to setup with varying forces acting on fixed morphology resulting in robustness to external interventions like winds.

Standing Up in the Wind Task Similarly, the policies trained with winds are able to generalize to scenarios with either stronger winds or winds inside water.

Locomotion Task We generate several novel scenarios for evaluating locomotion: with water, a terrain with hurdles of a certain height, a terrain with gaps between platforms, a bumpy terrain with a bi-modal distribution of bump heights, stairs, and an environment with a valley surrounded by walls on both sides (see Figure 7.2). These variations are generated procedurally. The modular policies learned by DGN tend to generalize better than the monolithic agent policies as shown in Table 7.2.

This generalization could be explained by the incrementally increasing complexity of self-assembling agents at training. For instance, the training begins with all limbs separate which gradually form group of two, three and so on, until the training converges. Since the policy is *modular with shared parameters across limbs*, the training of smaller size assemblies with small bumps would in turn prepare the large assemblies for performing locomotion through higher hurdles, stairs etc at test. Furthermore, the training terrain has a finite length which makes the self-assemblies launch themselves forward as far as possible upon reaching the boundary to maximize the distance along X-axis. This behavior helps the limbs generalize to environments like gaps or valley where they end up on the next terrain upon jumping and continue to perform locomotion.

7.4 Related Work

Morphologenesis & self-reconfiguring modular robots The idea of modular and self-assembling agents goes back at least to Von Neumman's *Theory of Self-Reproducing Automata* [249]. In robotics, such systems have been termed "self-reconfiguring modular robots" [159, 238]. There has been a lot of work in modular robotics to design real hardware robotic modules that can be docked together to form complex robotic morphologies [42, 79, 203, 264, 270]. Alternatives to optimize agent morphologies include genetic algorithms that search over a generative grammar [228] and energy-based minimization to directly optimizing controllers [45, 251]. Conditioning on several hardware designs has also been shown to increase robustness [35]. We approach morphogenesis from a learning perspective, in particular deep RL, and study the resulting generalization properties. We achieve morphological co-evolution

via *dynamic actions* (linking), which agents take during their lifetimes, whereas the past approaches treat morphology as an optimization target to be updated between generations or episodes. Since the physical morphology also defines the connectivity of the policy net, our proposed algorithm can also be viewed as performing a kind of neural architecture search [281] in physical agents.

Graph neural networks Encoding graphical structures into neural networks [210] has been used for a large number of applications, including question answering [8], quantum chemistry [78], semi-supervised classification [121], and representation learning [267]. The works most similar to ours involve learning controllers [208, 253]. For example, Nervenet [253] represents individual limbs and joints as nodes in a graph and demonstrates multi-limb generalization. However, the morphologies on which Nervenet operates are not learned jointly with the policy and hand-defined to be compositional in nature. Others [16, 104] have shown that graph neural networks can also be applied to inference models as well as to planning. Prior graph neural network based approaches deal with static graph which is defined by auxiliary information, e.g. language parser [8]. In contrast, we propose dynamic graph networks where the graph policy changes itself dynamically over the training.

Concurrent Work [88, 212] use RL to improve limb design given fixed morphology. Wang *et al*. [252] gradually evolves the environment to improve robustness. However, both the work assume the topology of agent morphology to stay the same during train and test.

7.5 Discussion

Modeling intelligent agents as modular, self-assembling morphologies has long been a very appealing idea. The efforts to create practical systems to evolve artificial agents goes back at least two decades to the beautiful work of Karl Sims [228]. In this chapter, we are revisiting these ideas using the contemporary machinery of deep networks and reinforcement learning. Examining the problem in the context of machine learning, rather than optimization, we are particularly interested in modularity as a key to generalization, in terms of improving adaptability and robustness to novel environmental conditions. Poor generalization is the Achilles heel of modern robotics research, and the hope is that this could be a promising direction in addressing this key issue. We demonstrated a number of promising experimental results, suggesting that modularity does indeed improve generalization in simulated agents. While these are just the initial steps, we believe that the proposed research direction is promising and its exploration will be fruitful to the research community.

Chapter 8

Conclusion

We employ *self-supervised prediction* as a central theme in building agents that learn to represent the sensory input and simultaneously learn to map this understanding to their motor outputs to acquire skills using data as its own supervision. We begin with minimal assumptions and build complete systems that learn from just raw sensory data. This is inspired by the kind of learning that humans engage in from a very early stage of development and is the main driving force for our seamless general-purpose behavior [230].

We adopted this ideology toward learning visual representations via self-supervision; building curiosity-driven agents (virtual as well as real) that can learn to play video games, walk in simulation and perform real-world object manipulation without any rewards or supervision; and learning to control self-assembling modular agents. Later, this combination of skills is deployed to achieve the end-goals, i.e., tasks given to the agent, by explicitly planning using the learned models with little to no supervision from a teacher. These self-supervised robotic agents, after exploring the environment, can tie knots using rope, find their way in office environments and rearrange objects. In this dissertation, we have put forward some initial efforts toward a larger goal of developing human-like general-purpose intelligent systems. Now, we describe some of the future directions that immediately follow from this agenda.

Exploration: Intrinsic Goals, Hierarchy, and Representation In [58], we have quantified the exploration of humans players in video games. This study reveals is that humans often explore by generating intermediate goals, e.g., reaching object-like concepts, which is unlike the prediction error formulation as we discussed in Chapter 4. Formulating this intrinsic goal-driven exploration [219] in artificial agents motivates several deep questions: What constitutes a good intrinsic goal? How to represent goals? How should such representation be learned? Each of these questions could lead to a thesis in itself. In the long run, we believe that exploration formulations should not be limited to modeling intrinsic rewards, but should also take into account the hierarchy at which such a reward is generated.

Low-cost Robot Learning Today's robotic systems are too expensive to be deployable at scale, and hence, building low-cost robotic setups is an attractive proposal. We have taken some initial steps toward this direction by assembling a mobile robot (see Chapter 6), and a low-cost arm setup that costs less than 1400 USD (see Figure 8.1). However, low-cost actuators are also noisy and inaccurate in addition to being parallelizable. Therefore, such agents would also need multiple sensors and robust algorithms that could generalize to unexpected failures.

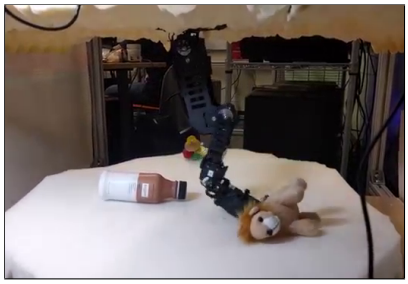


Figure 8.1: Attempt to deploy curiosity-driven exploration on a custom designed low-cost arm. Soft-padding is to prevent catastrophic damages.

Social Learning: Large-scale, Multi-agent, Imitation An agent learning on its own is limited by both its efficiency and the complexity of skills it learns. Learning in a multi-agent scenario is fundamental to acquiring complex skills which a single agent might never develop. While most of the works in robotic learning today use multi-agent settings to parallelize training, an adversarial competition between different agents could be the key to their success. A setup where different agents have competing objectives creates an arms race leading to progressively increasing complexity. Another important aspect in social learning is to be able to learn from a third person's point of view: an agent should be able to understand the actions and translate them into its own frame of reference.

Toward the long-term goal of understanding intelligence, we have drawn inspiration from psychology and biology and built practical systems at the interface of computer vision, machine learning, and robotics. In the words of Alan Turing,

We can only see a short distance ahead,
but we can see plenty there that needs to be done.

Indeed, only time will tell the true potential of these directions.

Appendix A

Experimental Details for Curiosity-driven Exploration

A.1 Implementation Details

We have released the training code and environments on our website [1].

Pre-processing: All experiments were done with pixels. We converted all images to grayscale and resized to size 84x84. We learn the agent's policy and forward dynamics function both on a stack of historical observations $[x_{t-3}, x_{t-2}, x_{t-1}, x_t]$ instead of only using the current observation. This is to capture partial observability in these games. In the case of Super Mario Bros and Atari experiments, we also used a standard frameskip wrapper that repeats each action 4 times.

Hyper-parameters: Our embedding network and policy networks had identical architectures and were based on the standard convolutional networks used in Atari experiments. The layer we take as features in the embedding network had dimension 512 in all experiments and no nonlinearity. To keep the scale of the prediction error consistent relative to extrinsic reward, in the Unity experiments we applied batchnorm to the embedding network. We also did this for the Mario generalization experiments to reduce covariate shift from level to level. For the VAE auxiliary task and pixel method, we used a similar deconvolutional architecture the exact details of which can be found in our code submission. The IDF and forward dynamics networks were heads on top of the embedding network with several extra fully-connected layers of dimensionality 512. We used a learning rate of 0.0001 for all networks. In most experiments, we used 128 parallel environments with the exceptions of the Unity and Roboschool experiments where we could only run 32 parallel environments, and the large scale Mario experiment where we used 1024. We used rollouts of length 128 in all experiments except for the Unity experiments where we used 512 length rollouts so that the

[1] Website at https://pathak22.github.io/large-scale-curiosity/

Algorithm 2: Curiosity-driven Learning

Initialize the networks $f(x_t, a_t; \theta_f)$, $\pi(x_t; \theta_\pi)$ and $\phi(x; \theta_\phi)$
$D = \{\}$
for *iteration* $i = 1$ *to* $\ldots$ **do**
  for *envs in parallel* $t = 1$ *to* 128 **do**
    for *iteration* $t = 1$ *to* 128 **do**
      Sample $a \sim \pi(x_t; \theta_\pi)$ and act using a in the environment
      $D \Leftarrow D + (x_t, a_t, x_{t+1}, r_t)$ where $r_t = \|f(x_t, a_t; \theta_f) - \phi(x_{t+1}; \theta_\phi)\|_2^2$
    end
  end
  for *steps* $k = 1$ *to* 64 **do**
    Sample batch size of 2048 from D and update using ADAM as follows:
      $\theta_f^{'} := \theta_f - \eta_1 \nabla_{\theta_f} \mathbb{E}\big[\|f(x_t, a_t; \theta_f) - \phi(x_{t+1}; \theta_\phi)\|_2^2\big]$
      $\theta_\phi^{'} := \theta_\phi - \eta_2 \nabla_{\theta_\phi} \mathbb{E}\big[\| \ldots \|_2^2\big]$: some auxiliary task
      $\theta_\pi^{'} := \theta_\pi + \eta_3 \nabla_{\theta_\pi} \mathbb{E}_{\pi(x_t; \theta_\pi)}\big[\sum_t r_t\big]$: use PPO with discounted returns
    $\theta_f \Leftarrow \theta_f^{'}$
    $\theta_\phi \Leftarrow \theta_\phi^{'}$
    $\theta_\pi \Leftarrow \theta_\pi^{'}$
  end
end

network could quickly latch onto the sparse reward. In the initial 9 experiments on Mario and Atari, we used 3 optimization epochs per rollout in the interest of speed. In the Mario scaling, generalization experiments, as well as the Roboschool experiments, we used 6 epochs. In the Unity experiments, we used 8 epochs, again to more quickly take advantage of sparse rewards.

A.2 Additional Results

Figure A.1 shows the performance of curiosity-driven agents based on Inverse Dynamics and Random features on 48 Atari games. For the majority of the training process RF perform better than a random agent in about 67% of the environments, while IDF perform better than a random agent in about 71% of the environments.

We include some results on combining intrinsic and extrinsic reward on several sparse reward Atari games. When combining with extrinsic rewards, we use the end of the episode signal. The reward used is the extrinsic reward plus 0.01 times the intrinsic reward. The results are shown in Table A.1. We don't observe a large difference between the settings, likely because the combination of intrinsic and extrinsic reward needs to be tuned. We did observe that one of the intrinsic+extrinsic runs on Montezuma's Revenge explored 10 rooms.

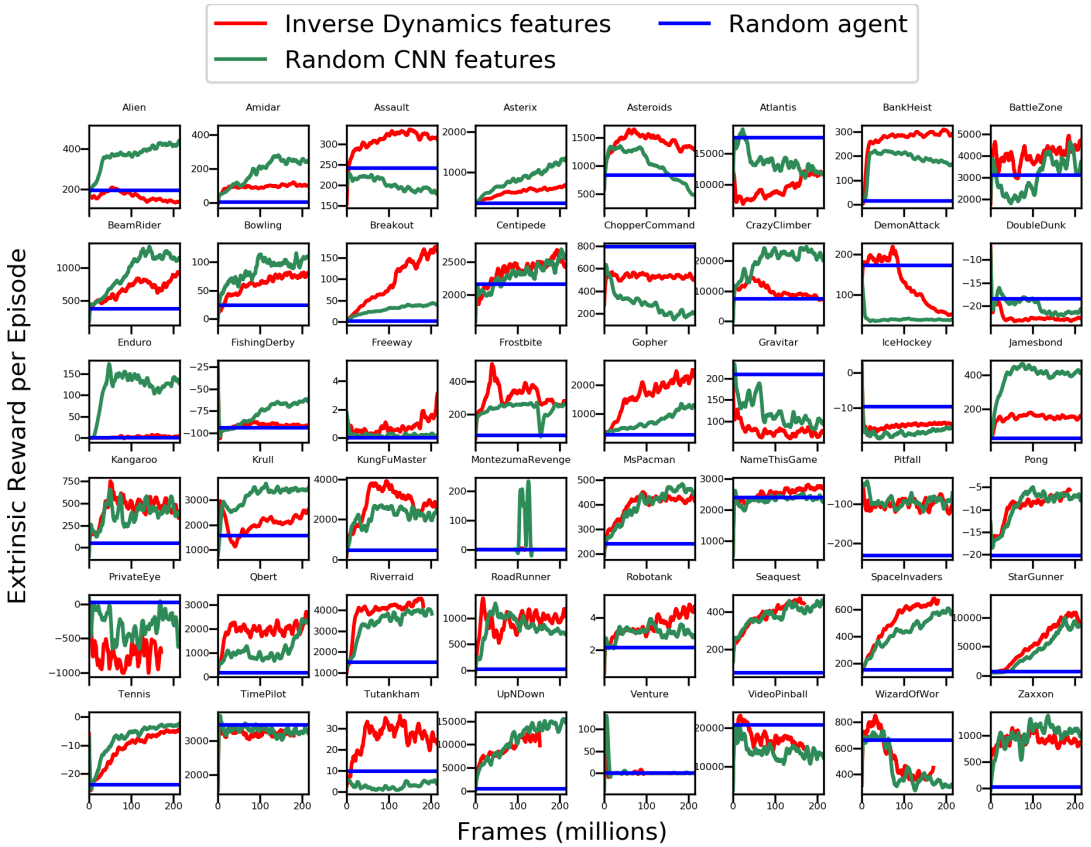


Figure A.1: Pure curiosity-driven exploration (no extrinsic reward, or end-of-episode signal) on 48 Atari games. We observe that the extrinsic returns of curiosity-driven agents often increases despite the agents having no access to the extrinsic return or end of episode signal. In multiple environments, the performance of the curiosity-driven agents is significantly better than that of a random agent, although there are environments where the behavior of the agent is close to random, or in fact seems to minimize the return, rather than maximize it.

Reward	Gravitar	Freeway	Venture	PrivateEye	MontezumaRevenge
Ext Only	999.3 ± 220.7	33.3 ± 0.6	0 ± 0	5020.3 ± 395	1783 ± 691.7
Ext + Int	1165.1 ± 53.6	32.8 ± 0.3	416 ± 416	3036.5 ± 952.1	2504.6 ± 4.6

Table A.1: These results compare the mean reward ($\pm$ std-error) after 100 million frames across 3 seeds for an agent trained with intrinsic plus extrinsic reward versus extrinsic reward only. The extrinsic (coefficient 1.0) and intrinsic reward (coefficient 0.01) were directly combined without any hyper-parameter tuning, and leave the question on how to optimally combine them for future work.

Appendix B

Experimental Details for Zero-Shot Imitation

B.1 Rope Manipuation

Robotic Setup Our setup of Baxter robot for rope manipulation task follows the one described in Nair *et al*. [161]. We re-use the data that is collected by a Baxter robot interacting with a rope kept on a table in front of it in a self-supervised manner, and consists of approximately 60K interaction pairs.

Implementation Details The base architecture for all the methods consists of a pre-trained AlexNet, whose features are fed into a skill policy that predicts the location of grasp, direction of displacement, and the magnitude of displacement. For the forward regularizer baseline, a forward model is trained to jointly regularize the AlexNet features along with the skill policy network with loss weight of forward model set to 0.1. For our proposed forward-consistent GSP, a forward consistency loss is then applied to the actions predicted by the skill policy network. The forward consistency loss weight is set to 0.1. Since this is a fully observed setup, we did not use recurrence in any of the skill policy networks. All the models are optimized using Adam [119] with a learning rate of $1e-4$. For the first 40K iterations, AlexNet weights were frozen, and then fine-tuned jointly with the later layers.

B.2 Navigation in Indoor Office Environments

Robotic Setup We used the TurtleBot2 robot comprising of a wheeled Kobuki base and an Orbbec Astra camera for capturing RGB images for all our experiments. The robot's action space had four discrete actions: move forward, turn left, turn right, and stand still (i.e., no-op). The forward action is approximately 10cm forward translation and the turning actions are approximately 14-18 degrees of rotation. These numbers vary due to the use of velocity control. A powerful on-board laptop was used to process the images and infer the motor commands. Several modifications were made to the default TurtleBot setup: the

Model Name	Maze Runs - Optimal Steps: 100 Run-1	Run-2	Run-3	Loop Runs - Optimal Steps: 85 Run-1	Run-2	Run-3
SIFT	2/20 (10)	1/20 (9)	3/20 (38)	—	—	—
GSP-NoPrevAction-NoFwdConst	12/20 (109)	14/20 (184)	20/20 (263)	—	—	—
GSP-NoFwdConst	13/20 (147)	18/20 (325)	20/20 (166)	0/17 (0)	0/17 (0)	0/17 (0)
GSP (ours)	20/20 (353)	12/20 (194)	20/20 (168)	0/17 (0)	17/17 (243)	17/17 (165)

Table B.1: Quantitative evaluation of TurtleBot's performance at following visual demonstrations in two conditions: maze and the loop. The fraction denotes how many landmarks it reaches out of the total number of landmarks in the full demonstration. The bracketed number represents the number of actions the agent took to reach its farthest landmark.

base's batteries were replaced with longer lasting ones, and the default NVIDIA Jetson TK1 embedded board was replaced with a more powerful GigaByte Aero laptop and an accompanying portable charging power bank.

Self-supervised Data Collection We devised an automated self-supervised scheme for data collection which does not require any human supervision. The robot first samples one out of four actions and then the number of times to repeat the selected action (i.e. action repeat). The no-op action is sampled with probability 0.05 and the other three actions are sampled with equal probability. In case the no-op action is chosen, an action repeat of {1, 2} steps is uniformly sampled. In case of other actions, an action repeat of 1-5 steps is randomly and uniformly chosen. The robot autonomously repeated this process and collected 230K interactions from two floors of a building. If the robot crashes into an object, it performs a reset maneuver by first moving backwards and then turning right/left by a uniformly sampled angle between 90-270 degrees. A separate floor of the building with substantially different furniture layout and visual textures is then used for testing the learned model.

Implementation Details The data collected by self-supervised exploration is then used to train our *recurrent forward-consistent GSP*. The base architecture of our model is an ImageNet pre-trained ResNet-50 [95] network. Input are the images and output are the actions of robot. The forward consistency model is first pre-trained and then fine-tuned together end-to-end with the GSP. The loss weight of the forward model is 0.1, and the objective is minimized using Adam [119] with learning rate of $5e-4$.

B.3 3D Navigation in VizDoom

Self-supervised Data Collection Our environment consists of two map. One map is used for training and validation, with different textures for validation. Second map has different textures than training and validation and is used for generalization experiments. For both curiosity and random exploration, we collect a total of 1.5 million frames each with action repeat of 4 collected in the standard `DoomMyWayHome` map used for training in Chapter 4. $\sim \frac{2}{3}$ of the data comes from random-room resets, and $\sim \frac{1}{3}$ of the data comes from a fixed-room

Model Name	Same Map, Same Texture Mean %	Same Map, Same Texture Efficiency %	Same Map, Diff Texture Mean %	Same Map, Diff Texture Efficiency %	Diff Map, Diff Texture Mean %	Diff Map, Diff Texture Efficiency %
Random Exploration for Data Collection:						
GSP-NoFwdConst	61.8 ± 0.9	60.4 ± 2.1	37.6 ± 0.7	68.6 ± 2.5	42.2 ± 0.8	50.6 ± 1.9
GSP (ours pixels)	61.0 ± 1.0	68.0 ± 2.2	38.1 ± 0.7	69.1 ± 2.5	40.3 ± 0.9	64.2 ± 2.3
GSP (ours features)	62.0 ± 1.0	75.8 ± 2.5	37.0 ± 0.7	87.1 ± 2.8	48.7 ± 0.9	52.5 ± 1.8
Curiosity-driven Exploration for Data Collection:						
GSP-NoFwdConst	70.7 ± 0.9	66.9 ± 1.4	49.8 ± 0.8	55.8 ± 2.2	51.2 ± 1.0	39.5 ± 1.3
GSP-FwdRegularizer	70.6 ± 0.9	67.9 ± 1.6	51.9 ± 0.8	49.3 ± 1.6	48.3 ± 1.0	49.3 ± 1.8
GSP (ours pixels)	71.0 ± 0.9	73.1 ± 2.7	53.3 ± 0.9	53.4 ± 2.0	52.2 ± 1.0	44.0 ± 1.5
GSP (ours features)	68.8 ± 1.0	72.0 ± 1.7	53.2 ± 0.8	53.0 ± 2.3	52.8 ± 0.9	37.7 ± 1.3

Table B.2: Quantitative evaluation of our proposed GSP and the baseline models at following visual demonstrations in VizDoom 3D Navigation. Means and standard errors are reported for demonstration completion and efficiency over 50 seeds and 5 human paths per environment type.

reset (i.e, room number 10). The curiosity policy was half sampled and half greedy with the exact split being 40% greedy policy random-room reset, 25% sample policy random-room reset, 25% sample policy fixed-room reset, and 10% greedy policy fixed-room reset.

For each scenario, we collect 5 human demonstrations each and give every 10th frame as input to the agent for the task of visual imitation. For each human path, we evaluate on 50 different seeds where the agent starts with a uniformly sampled orientation. We then get the median across 250 (50x5) total runs for each type of environment and report median of the percentage of the human path reached by the agent and how soon it got to that point relative to the human.

In the Chapter 6, we report median accuracy and the confidence interval for median [1]. Since the initial position of the agent is randomized in orientation compared to the one in visual demonstration, the mean results suffer from high variance due to outliers. Hence, median accuracy results in a more reliable metric. However, we report mean results in Table B.2 for the completion.

Implementation Details All models were trained with batch size 64, Adam Solver with 1e-4 learning rate, and landmark slices uniformly sampled between 5 to 15 action steps for each batch. The observations are 42x42 resolution, grayscale images with only one-time channel both for goal and current state. All models used the same goal recognizer that was trained on the curiosity data. For selecting the hyper-parameters in forward regularizer, pixel-based forward consistency, and feature-based forward consistency models, we selected the best loss coefficient among $\{0.01, 0.05, 0.1\}$ that achieved the highest median completion on our validation environment which consisted of the training maps with novel textures.

[1]Formula for computing median confidence intervals: http://www.ucl.ac.uk/ich/short-courses-events/about-stats-courses/stats-rm/Chapter_8_Content/confidence_interval_single_median

Appendix C

Experimental Details for Dynamic Graph Networks

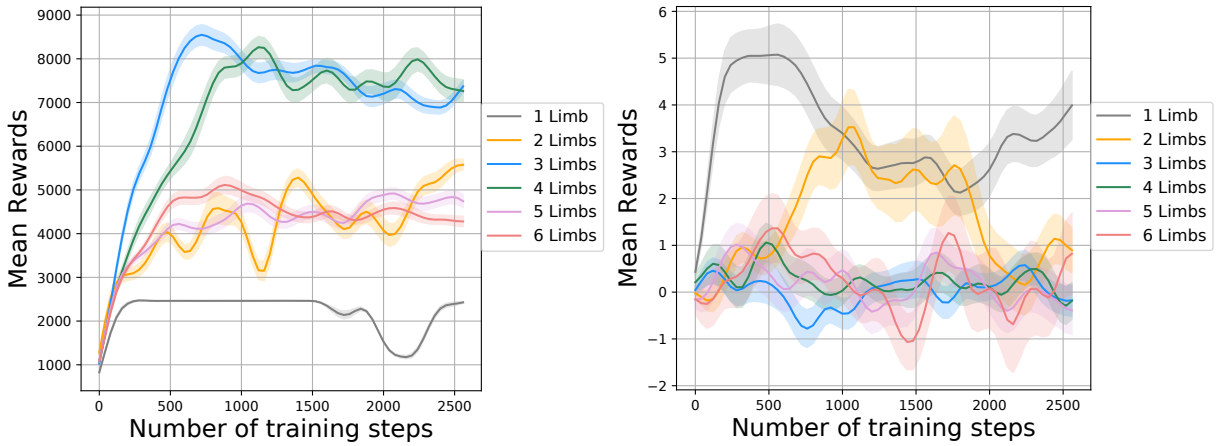
In this section, we provide additional details about the experimental setup, pseudo-code for DGN and extra experiments.

C.1 Implementation and Training details

We use PPO [218] as the underlying reinforcement learning method to optimize Equation 7.1 in Section 7.2.3. Each limb policy is represented by 4-layered fully-connected neural network with ReLU non-linearities and trained with a learning rate of $3e-4$, discount factor of 0.995, entropy coefficient of 0.01, advantage parameter of 0.95 and batch size of 2048. The messages are 32 length float vectors. The optimizer used to optimize PPO is RMS-Prop. Parameters are shared across network modules and they all predict action at the same time. Each episode is 5000 steps long at training. Across all the tasks, the number of limbs at training is kept fixed to 6. Limbs start each episode disconnected and located just above the ground plane at random locations, as shown in Figure 7.3. In the absence of an edge, input messages are set to 0 and the output ones are ignored. Action space is continuous raw torque values. We take the model from each time step and evaluate it on 50 episodes to plot mean and standard-deviation (confidence intervals) in training curves. At test, we report the mean reward across 50 episode runs of 1200 environment steps.

C.2 Fixed-Graph Baseline vs. Number of Limbs

To verify whether the training of *Monolithic Policy w/ Fixed Graph* is working, we ran it on standing up and locomotion tasks across varying number of limbs. We show in Figure C.1 that the baseline performs well with less number of limbs which suggests that the reason for failure in 6-limbs case is indeed the morphology graph being fixed, and not the implementation of this baseline.



(a) Standing Up (b) Locomotion

Figure C.1: The performance of *Monolithic Policy w/ Fixed Graph* baseline as the number of limbs varies in the two tasks: standing up (left) and locomotion (right). This shows that the monolithic baseline works well with less (1-3 limbs), but fails with 6 limbs during training.

C.3 Pseudo Code for DGN Algorithm

Notation is summarized in Algorithm 3, and the full pseudo-code is summarized in Algorithm 4. The pseudo-code for no-message DGN is same as Algorithm 4 but hard-code incoming child and parent messages to be always 0, i.e., $m_t^{C_i} = 0$ and $m_t^{p_i} = 0$ in each iteration. Full source code available at https://pathak22.github.io/modular-assemblies/.

Algorithm 3: Notation Summary (defined in Section 7.2.3)

foreach *node* i **do**
  $a_t^i, m_t^i = \pi_\theta^i(s_t^i, m_t^{C_i})$
end
where
s_t^i: observation state of agent limb i
a_t^i: action output of agent limb i: 3 torques, attach, detach
$m_t^{C_i}$: aggregated message from children nodes input to agent i (bottom-up-1)
m_t^i: output message that agent i passes to its parent (bottom-up-2)
θ: θ_1, θ_2
messages are 32 length floating point vectors.

Algorithm 4: Pseudo-code: DGN w/ Message Passing

Initialize parameters θ_1, θ_2 randomly.
Initialize all message vectors $m_t^{C_i}, m_t^i$ to be zero
Represent graph connectivity G as a list of edges
Note: In the beginning, all edges are zeros, i.e., non-existent
foreach *timestep* t **do**
  Each limb agent i observes its own state vector s_t^i
  foreach *agent* i **do**
    # Compute incoming child messages
    $m_t^{C_i} = 0$
    foreach *child node* c *of agent* i **do**
      $m_t^{C_i} += m_t^c$
    end
    # Compute action and message to parent p of agent i in G
    $a_t^i, m_t^i := \pi_\theta^i(s_t^i, m_t^{C_i})$
    # Execute morphology change as per a_t^i
    if $a_t^i[3] ==$ *attach* **then**
      find closest agent j within distance d from agent i, otherwise j=NULL
      add edge (i, j) in G
      also make physical joint between (i, j)
    end
    if $a_t^i[4] ==$ *detach* **then**
      delete edge (i, parent of i) in G
      also delete physical joint between (i, j)
    end
    # Execute torques from a_t^i
    Apply torques $a_t^i[0], a_t^i[1], a_t^i[2]$
  end
  # Update graph and agent morphology
  Find all connected components in G
  foreach *connected component* c **do**
    foreach *agent* $i \in c$ **do**
      reward r_t^i = reward of c (e.g. max height)
    end
  end
end
Update θ to maximize discounted reward using PPO as follows:
let $\vec{a}_t = [a_t^1, a_t^2..a_t^n]$
  $\vec{s}_t = [s_t^1, s_t^2..s_t^n]$
  $\hat{A}_t =$ advantage of discounted rewards, $r_t = \sum_{agenti} r_t^i$
PPO: $\max_\theta \mathbb{E}[\hat{A}_t \frac{\pi_\theta(\vec{a}_t|\vec{s}_t)}{\pi_{\theta_{\text{old}}}(\vec{a}_t|\vec{s}_t)} - \beta \text{KL}(\pi_{\theta_{\text{old}}}(.|\vec{s}_t)\pi_\theta(.|\vec{s}_t))]$
Repeat until training converges

Bibliography

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[2] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *TPAMI*, 2012.

[3] J. Achiam and S. Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

[4] P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. *ICCV*, 2015.

[5] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. *NIPS*, 2016.

[6] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*. Garland Publishing, New York, 1994.

[7] J. Aloimonos, I. Weiss, and A. Bandyopadhyay. Active vision. *International journal of computer vision*, 1(4):333–356, 1988.

[8] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *CVPR*, 2016.

[9] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. In *NIPS*, 2017.

[10] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 2009.

[11] C. Arteta, V. Lempitsky, and A. Zisserman. Counting in the wild. In *ECCV*, 2016.

[12] R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[13] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos. Revisiting active perception. *Autonomous Robots*, pages 1–20, 2016.

[14] A. Bandura and R. H. Walters. *Social learning theory*, volume 1. Prentice-hall Englewood Cliffs, NJ, 1977.

[15] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, 2009.

[16] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[17] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 2016.

[18] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei. On the optimization of a synaptic learning rule. In *Optimality in Artificial and Biological Neural Networks*, 1992.

[19] Y. Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2009.

[20] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *TPAMI*, 35(8), 2013.

[21] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013.

[22] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. *Computer graphics and interactive techniques*, 2000.

[23] M. Björkman and D. Kragic. Active 3d scene segmentation and detection of unknown objects. In *ICRA*, 2010.

[24] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[25] R. I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *JMLR*, 2002.

[26] C. Breazeal and B. Scassellati. Robots that imitate humans. *Trends in cognitive sciences*, 2002.

[27] J. S. Bridle and S. J. Cox. Recnorm: Simultaneous normalisation and classification applied to speech recognition. In *NIPS*, 1991.

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[29] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros. Large-scale study of curiosity-driven learning. In *ICLR*, 2019.

[30] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros. Large-scale study of curiosity-driven learning. *ICLR*, 2019.

[31] A. Byravan and D. Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *ICRA*, 2017.

[32] G. Caron, E. Marchand, and E. M. Mouaddib. Photometric visual servoing for omnidirectional cameras. *Autonomous Robots*, 2013.

[33] R. Caruana. Multitask learning: A knowledge-based source of inductive bias. In *ICML*, 1993.

[34] R. Y. Chen, J. Schulman, P. Abbeel, and S. Sidor. UCB and infogain exploration via q-ensembles. *arXiv preprint arXiv:1706.01502*, 2017.

[35] T. Chen, A. Murali, and A. Gupta. Hardware conditioned policies for multi-robot transfer learning. In *NIPS*, 2018.

[36] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

[37] H. Chui and A. Rangarajan. A new point matching algorithm for non-rigid registration. *CVIU*, 2003.

[38] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *ICML*, 2008.

[39] G. Costikyan. *Uncertainty in games*. Mit Press, 2013.

[40] I. Dagan and S. Engelson. Committee-based sampling for training probabilistic classifiers. *ICML*, 1995.

[41] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *CVPR*, 2005.

[42] J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell. An integrated system for perception-driven autonomy with modular robots. *Science Robotics*, 2018.

[43] H. Daumé III. Frustratingly easy domain adaptation. *ACL*, 2007.

[44] A. J. Davison and D. W. Murray. Mobile robot localisation using active vision. In *ECCV*, 1998.

[45] M. De Lasa, I. Mordatch, and A. Hertzmann. Feature-based locomotion controllers. In *ACM Transactions on Graphics (TOG)*, 2010.

[46] V. R. de Sa. Learning classification with unlabeled data. *NIPS*, 1994.

[47] M. Deisenroth and C. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. *ICML*, 2011.

[48] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[49] R. Dillmann. Teaching and learning of robot tasks via observation of human performance. *Robotics and Autonomous Systems*, 2004.

[50] C. Doersch, A. Gupta, and A. A. Efros. Context as supervisory signal: Discovering objects with predictable context. *ECCV*, 2014.

[51] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. *ICCV*, 2015.

[52] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. Efros. What makes paris look like paris? *ACM Transactions on Graphics*, 2012.

[53] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *ICML*, 2014.

[54] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial Feature Learning. *ICLR*, 2017.

[55] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. *ICLR*, 2016.

[56] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *CVPR*, 2015.

[57] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. One-shot imitation learning. In *NIPS*, 2017.

[58] R. Dubey, P. Agrawal, D. Pathak, T. L. Griffiths, and A. A. Efros. Investigating human priors for playing video games. In *ICML*, 2018.

[59] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville. Adversarially learned inference. *ICLR*, 2017.

[60] F. Ebert, C. Finn, A. X. Lee, and S. Levine. Self-supervised visual planning with temporal skip connections. *arXiv preprint arXiv:1710.05268*, 2017.

[61] A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. *ICCV*, 1999.

[62] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes challenge: A retrospective. *IJCV*, 2014.

[63] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. *ICLR*, 2019.

[64] A. Faktor and M. Irani. Video Segmentation by Non-Local Consensus voting. *BMVC*, 2014.

[65] C. Finn. *Learning to Learn with Gradients*. PhD thesis, UC Berkeley, 2018.

[66] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE, 2017.

[67] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. *CoRL*, 2017.

[68] P. Fitzpatrick. First contact: an active vision approach to segmentation. In *IROS*, 2003.

[69] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[70] D. Foster and P. Dayan. Structure in the space of value functions. *Machine Learning*, 2002.

[71] J. Fu, J. D. Co-Reyes, and S. Levine. Ex2: Exploration with exemplar models for deep reinforcement learning. *NIPS*, 2017.

[72] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

[73] Y. Gal, R. McAllister, and C. E. Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, 2016.

[74] F. Galasso, N. Nagaraja, T. Cardenas, T. Brox, and B. Schiele. A unified video segmentation benchmark: Annotation, metrics and analysis. *ICCV*, 2013.

[75] D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. *arXiv preprint arXiv:1704.05588*, 2017.

[76] R. Garg, V. K. B.G., G. Carneiro, and I. Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. *ECCV*, 2016.

[77] J. J. Gibson. *The ecological approach to visual perception*. Psychology Press, 1979.

[78] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[79] K. Gilpin, K. Kotay, D. Rus, and I. Vasilescu. Miche: Modular shape formation by self-disassembly. *IJRR*, 2008.

[80] R. Girshick. Fast R-CNN. *ICCV*, 2015.

[81] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *NIPS*, 2014.

[82] A. Gopnik. What do babies think? https://www.ted.com/talks/alison_gopnik_what_do_babies_think, 2011.

[83] A. Gopnik, A. N. Meltzoff, and P. K. Kuhl. *The scientist in the crib: Minds, brains, and how children learn.* William Morrow & Co, 1999.

[84] R. Goroshin, J. Bruna, J. Tompson, D. Eigen, and Y. LeCun. Unsupervised learning of spatiotemporally coherent metrics. *ICCV*, 2015.

[85] R. Goroshin, M. Mathieu, and Y. LeCun. Learning to linearize under uncertainty. *NIPS*, 2015.

[86] K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. *ICLR Workshop*, 2017.

[87] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. *CVPR*, 2017.

[88] D. Ha. Reinforcement learning for improving agent design. *arXiv preprint arXiv:1810.03779*, 2018.

[89] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[90] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. *ICCV*, 2011.

[91] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 2005.

[92] K. Hausman, D. Pangercic, Z.-C. Márton, F. Bálint-Benczédi, C. Bersch, M. Gupta, G. Sukhatme, and M. Beetz. Interactive segmentation of textured and textureless objects. In *Handling Uncertainty and Networked Structure in Robot Control*. Springer, 2015.

[93] J. Hays and A. A. Efros. Scene completion using millions of photographs. *SIGGRAPH*, 2007.

[94] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

[95] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[96] J. J. Heckman. Sample selection bias as a specification error (with an application to the estimation of labor supply functions), 1977.

[97] M. Henaff, A. Canziani, and Y. LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. *ICLR*, 2019.

[98] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[99] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

[100] J. Ho and S. Ermon. Generative adversarial imitation learning. In *NIPS*, 2016.

[101] J. Hoffman. *Adaptive Learning Algorithms for Transferable Visual Recognition*. PhD thesis, UC Berkeley, 2016.

[102] N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel. Bayesian active learning for classification and preference learning. *arXiv*, 2011.

[103] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. In *NIPS*, 2016.

[104] D.-A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. *arXiv preprint arXiv:1807.03480*, 2018.

[105] P. J. Huber. Robust estimation of a location parameter. *The annals of mathematical statistics*, 1964.

[106] R. Hunicke, M. LeBlanc, and R. Zubek. Mda: A formal approach to game design and game research. In *AAAI Workshop on Challenges in Game AI*, 2004.

[107] K. Ikeuchi and T. Suehiro. Toward an assembly plan from observation. i. task recognition with polyhedral objects. *IEEE Transactions on Robotics and Automation*, 1994.

[108] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[109] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *ICLR*, 2017.

[110] K. Jarrett, K. Kavukcuoglu, Y. LeCun, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[111] D. Jayaraman and K. Grauman. Learning image representations tied to ego-motion. *ICCV*, 2015.

[112] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, 2014.

[113] M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive science*, 1992.

[114] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[115] L. Kanal and N. Randall. Recognition system design by statistical analysis. In *Proceedings of the 1964 19th ACM national conference*, 1964.

[116] M. Kearns and D. Koller. Efficient reinforcement learning in factored mdps. In *IJCAI*, 1999.

[117] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, 2016.

[118] J. Kenney, T. Buckley, and O. Brock. Interactive segmentation for manipulation in unstructured environments. In *ICRA*, 2009.

[119] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[120] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[121] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[122] A. S. Klyubin, D. Polani, and C. L. Nehaniv. Empowerment: A universal agent-centric measure of control. In *Evolutionary Computation*, 2005.

[123] P. Kohli, P. H. Torr, et al. Robust higher order potentials for enforcing label consistency. *IJCV*, 2009.

[124] H. Koichi and H. Tom. *Visual servoing: real-time control of robot manipulators based on visual sensory feedback*. World scientific, 1993.

[125] Z. Kolter and A. Ng. Near-bayesian exploration in polynomial time. *ICML*, 2009.

[126] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent initializations of convolutional neural networks. *ICLR*, 2016.

[127] P. Krähenbühl and V. Koltun. Geodesic object proposals. In *ECCV*, 2014.

[128] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *NIPS*, 2012.

[129] Y. Kuniyoshi, M. Inaba, and H. Inoue. Teaching by showing: Generating robot programs by visual observation of human performance. In *International Symposium on Industrial Robots*, 1989.

[130] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 1994.

[131] T. Lampe and M. Riedmiller. Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, 2013.

[132] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. *ECCV*, 2016.

[133] N. Lazzaro. Why we play games: Four keys to more emotion in player experiences. In *Proceedings of GDC*, 2004.

[134] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[135] A. X. Lee, S. Levine, and P. Abbeel. Learning visual servoing with deep features and fitted q-iteration. *arXiv preprint arXiv:1703.11000*, 2017.

[136] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, 2008.

[137] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 2011.

[138] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with large-scale data collection. In *ISER*, 2016.

[139] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. *ACM SIGIR*, 1994.

[140] Y. Li, M. Paluri, J. M. Rehg, and P. Dollár. Unsupervised learning of edges. *CVPR*, 2016.

[141] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ICLR*, 2016.

[142] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. *ECCV*, 2014.

[143] D. Y. Little and F. T. Sommer. Learning and exploration in action-perception loops. *Closing the Loop Around Neural Systems*, 2014.

[144] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. *ICRA*, 2018.

[145] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015.

[146] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *NIPS*, 2012.

[147] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *CoRR*, abs/1709.06009, 2017.

[148] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[149] T. Malisiewicz and A. Efros. Beyond categories: The visual memex model for reasoning about object relationships. *NIPS*, 2009.

[150] Mapillary. Open source structure from motion pipeline. *https://github.com/mapillary/OpenSfM*, 2016.

[151] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *NIPS*, 2013.

[152] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *ICLR*, 2017.

[153] I. Misra, C. L. Zitnick, and M. Hebert. Shuffle and Learn: Unsupervised Learning using Temporal Order Verification. *ECCV*, 2016.

[154] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[155] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[156] S. Mohamed and D. J. Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *NIPS*, 2015.

[157] K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *IJRR*, 2013.

[158] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 2017.

[159] S. Murata and H. Kurokawa. Self-reconfigurable robots. *IEEE Robotics & Automation Magazine*, 2007.

[160] D. K. Naik and R. Mammone. Meta-neural networks that learn by learning. In *IJCNN*, 1992.

[161] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. *ICRA*, 2017.

[162] L. Nalpantidis, M. Björkman, and D. Kragic. Yes-yet another object segmentation: exploiting camera movement. In *IROS*, 2012.

[163] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670, 2000.

[164] M. Noroozi and P. Favaro. Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles. *ECCV*, 2016.

[165] P. Ochs, J. Malik, and T. Brox. Segmentation of moving objects by long term video analysis. *TPAMI*, 36(6), 2014.

[166] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *NIPS*, 2015.

[167] A. Oliva and A. Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in brain research*, 2006.

[168] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[169] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *NIPS*, 2016.

[170] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin. An iterative regularization method for total variation-based image restoration. *Multiscale Modeling & Simulation*, 2005.

[171] G. Ostrovski, M. G. Bellemare, A. v. d. Oord, and R. Munos. Count-based exploration with neural density models. *ICML*, 2018.

[172] Y. Ostrovsky, E. Meyers, S. Ganesh, U. Mathur, and P. Sinha. Visual parsing after recovery from blindness. *Psychological Science*, 2009.

[173] P.-Y. Oudeyer. Computational theories of curiosity-driven learning. *arXiv preprint arXiv:1802.10546*, 2018.

[174] P.-Y. Oudeyer and F. Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 2009.

[175] A. Owens, J. Wu, J. H. McDermott, W. T. Freeman, and A. Torralba. Ambient sound provides supervision for visual learning. *ECCV*, 2016.

[176] J. Pajarinen and V. Kyrki. Decision making under uncertain segmentations. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1303–1309. IEEE, 2015.

[177] S. E. Palmer. *Vision science: Photons to phenomenology*. MIT press, 1999.

[178] P. Paquette. Super mario bros. in openai gym. *github:ppaquette/gym-super-mario*, 2016.

[179] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[180] D. Pathak, D. Gandhi, and A. Gupta. Self-supervised exploration via disagreement. In *ICML*, 2019.

[181] D. Pathak, R. Girshick, P. Dollár, T. Darrell, and B. Hariharan. Learning features by watching objects move. *CVPR*, 2017.

[182] D. Pathak, P. Krähenbühl, and T. Darrell. Constrained convolutional neural networks for weakly supervised segmentation. In *ICCV*, 2015.

[183] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. Efros. Context Encoders: Feature Learning by Inpainting. *CVPR*, 2016.

[184] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016.

[185] D. Pathak, C. Lu, T. Darrell, P. Isola, and A. Efros. Learning to control self-assembling morphologies: A study of generalization via modularity. In *arXiv preprint arXiv:1902.05546*, 2019.

[186] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell. Zero-shot visual imitation. In *ICLR*, 2018.

[187] D. Pathak, Y. Shentu, D. Chen, P. Agrawal, T. Darrell, S. Levine, and J. Malik. Learning instance segmentation by interaction. In *CVPR Robotics Vision Workshop*, 2018.

[188] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. V. Gool, M. Gross, and A. Sorkine-Hornung. A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation. *CVPR*, 2016.

[189] S. T. Piantadosi and C. Kidd. Extraordinary intelligence and the care of infants. *PNAS*, 2016.

[190] P. O. Pinheiro, R. Collobert, and P. Dollár. Learning to Segment Object Candidates. *NIPS*, 2015.

[191] P. O. Pinheiro, R. Collobert, and P. Dollár. Learning to segment object candidates. In *Advances in Neural Information Processing Systems*, pages 1990–1998, 2015.

[192] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to Refine Object Segments. *ECCV*, 2016.

[193] L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. In *ECCV*, 2016.

[194] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *ICRA*, 2016.

[195] L. Z. Piotr Dollár. Structured forests for fast edge detection. *ICCV*, 2013.

[196] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter space noise for exploration. *ICLR*, 2018.

[197] D. A. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In *NIPS*, 1989.

[198] D. Potapov, M. Douze, Z. Harchaoui, and C. Schmid. Category-specific video summarization. *ECCV*, 2014.

[199] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete bayesian reinforcement learning. In *ICML*, 2006.

[200] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2016.

[201] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[202] S. Rifai, Y. Bengio, A. Courville, P. Vincent, and M. Mirza. Disentangling factors of variation for facial expression recognition. *ECCV*, 2012.

[203] J. W. Romanishin, K. Gilpin, and D. Rus. M-blocks: Momentum-driven, magnetic modular robots. In *IROS*, 2013.

[204] A. Rosenfeld, R. Zemel, and J. K. Tsotsos. The elephant in the room. *arXiv preprint arXiv:1808.03305*, 2018.

[205] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[206] E. L. Ryan, Richard; Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, 2000.

[207] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *ECCV*. Springer, 2010.

[208] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

[209] A. M. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng. On random weights and unsupervised feature learning. In *ICML*, pages 1089–1096, 2011.

[210] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Network*, 2009.

[211] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 1999.

[212] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. *arXiv preprint arXiv:1801.01432*, 2018.

[213] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *ICML*, 2015.

[214] J. Schmidhuber. *Evolutionary principles in self-referential learning*. PhD thesis, Technische Universität München, 1987.

[215] J. Schmidhuber. Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, 1991.

[216] J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *From animals to animats: Proceedings of the first international conference on simulation of adaptive behavior*, 1991.

[217] J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2010.

[218] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[219] L. Schulz. Finding new facts; thinking new thoughts. In *Advances in child development and behavior*. Elsevier, 2012.

[220] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, and S. Levine. Time-contrastive networks: Self-supervised learning from video. In *ICRA*, 2018.

[221] P. Sermanet, K. Xu, and S. Levine. Unsupervised perceptual rewards for imitation learning. In *RSS*, 2017.

[222] B. Settles. Active learning literature survey. *U Madison Tech Report*, 2010.

[223] H. Seung, M. Opper, and H. Sompolinsky. Query by committee. *COLT*, 1992.

[224] E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2017.

[225] P. Shyam, W. Jaśkowski, and F. Gomez. Model-Based Active Exploration. In *ICML*, 2019.

[226] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 2016.

[227] P. J. Silvia. Curiosity and motivation. In *The Oxford Handbook of Human Motivation*, 2012.

[228] K. Sims. Evolving virtual creatures. In *Computer graphics and interactive techniques*, 1994.

[229] S. P. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In *NIPS*, 2005.

[230] L. Smith and M. Gasser. The development of embodied cognition: Six lessons from babies. *Artificial life*, 2005.

[231] E. S. Spelke. Principles of object perception. *Cognitive science*, 1990.

[232] E. S. Spelke and K. D. Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

[233] B. C. Stadie, P. Abbeel, and I. Sutskever. Third-person imitation learning. In *ICLR*, 2017.

[234] B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *NIPS Workshop*, 2015.

[235] K. O. Stanley and J. Lehman. *Why greatness cannot be planned: The myth of the objective*. Springer, 2015.

[236] S. Still and D. Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 2012.

[237] J. Storck, S. Hochreiter, and J. Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. In *ICANN*, 1995.

[238] K. Stoy, D. Brandt, D. J. Christensen, and D. Brandt. *Self-reconfigurable robots: an introduction*. Mit Press Cambridge, 2010.

[239] A. Strehl and M. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 2008.

[240] S. Sukhbaatar, I. Kostrikov, A. Szlam, and R. Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *ICLR*, 2018.

[241] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[242] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS*, 2017.

[243] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. YFCC100M: The new data in multimedia research. *Communications of the ACM*, 59(2), 2016.

[244] S. Thrun and L. Pratt. Learning to learn: Introduction and overview. In *Learning to learn*. Springer, 1998.

[245] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994.

[246] A. M. Turing. Computing machinery and intelligence, 1950. One of the most influential papers in the history of the cognitive sciences: http://cogsci.umn.edu/millennium/final.html.

[247] H. Van Hoof, O. Kroemer, and J. Peters. Probabilistic segmentation and targeted exploration of objects in cluttered environments. *IEEE Transactions on Robotics*, 30(5):1198–1209, 2014.

[248] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. *ICML*, 2008.

[249] J. Von Neumann, A. W. Burks, et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 1966.

[250] J. Walker, C. Doersch, A. Gupta, and M. Hebert. An uncertain future: Forecasting from static images using variational autoencoders. *ECCV*, 2016.

[251] K. Wampler and Z. Popović. Optimal gait and form for animal locomotion. In *ACM Transactions on Graphics (TOG)*, 2009.

[252] R. Wang, J. Lehman, J. Clune, and K. O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

[253] T. Wang, R. Liao, J. Ba, and S. Fidler. Nervenet: Learning structured policy with graph neural networks. *ICLR*, 2018.

[254] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. *ICCV*, 2015.

[255] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, 2015.

[256] A. A. Weir, J. Chappell, and A. Kacelnik. Shaping of hooks in new caledonian crows. *Science*, 2002.

[257] M. Wertheimer. Laws of organization in perceptual forms. *A source book of Gestalt Psychology*, 1923.

[258] M. Wertheimer. Laws of organization in perceptual forms. *A source book of Gestalt psychology*, 1938.

[259] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

[260] W. J. Wilson, C. W. Hulls, and G. S. Bell. Relative end-effector control using cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 1996.

[261] D. M. Wolpert, Z. Ghahramani, and M. I. Jordan. An internal model for sensorimotor integration. *Science-AAAS-Weekly Paper Edition*, 1995.

[262] M. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[263] P. Wouters, H. Van Oostendorp, R. Boonekamp, and E. Van der Spek. The role of game discourse analysis and curiosity in creating engaging and effective serious games by implementing a back story and foreshadowing. *Interacting with Computers*, 2011.

[264] C. Wright, A. Johnson, A. Peck, Z. McCord, A. Naaktgeboren, P. Gianfortoni, M. Gonzalez-Rivero, R. Hatton, and H. Choset. Design of a modular snake robot. In *IROS*, 2007.

[265] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[266] Y. Yang, Y. Li, C. Fermüller, and Y. Aloimonos. Robot learning manipulation action plans by "watching" unconstrained videos from the world wide web. In *AAAI*, 2015.

[267] Z. Yang, B. Dhingra, K. He, W. W. Cohen, R. Salakhutdinov, Y. LeCun, et al. Glomo: Unsupervisedly learned relational graphs as transferable representations. *arXiv preprint arXiv:1806.05662*, 2018.

[268] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1476–1483, 2015.

[269] B. Yao, X. Jiang, A. Khosla, A. L. Lin, L. Guibas, and L. Fei-Fei. Human action recognition by learning bases of action attributes and parts. *ICCV*, 2011.

[270] M. Yim, D. G. Duff, and K. D. Roufas. Polybot: a modular reconfigurable robot. In *ICRA*, 2000.

[271] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics & Automation Magazine*, 2007.

[272] B. H. Yoshimi and P. K. Allen. Active, uncalibrated visual servoing. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, 1994.

[273] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *NIPS*, 2014.

[274] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *ECCV*, 2014.

[275] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. A. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *CoRR*, abs/1803.09956, 2018.

[276] R. Zhang, P. Isola, and A. A. Efros. Colorful Image Colorization. *ECCV*, 2016.

[277] R. Zhang, P. Isola, and A. A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. *CVPR*, 2017.

[278] T. Zhou, P. Krahenbuhl, M. Aubry, Q. Huang, and A. A. Efros. Learning dense correspondence via 3d-guided cycle consistency. In *CVPR*, 2016.

[279] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *ICCV*, 2017.

[280] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

[281] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.